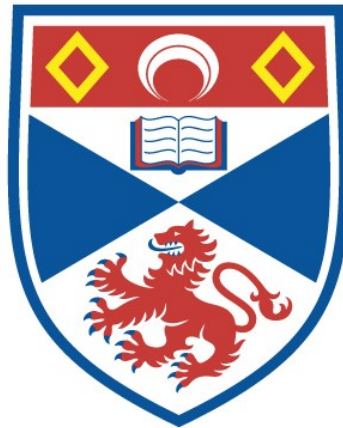


# Groups defined by language theoretic classes

Raad Sameer Al Sheikh Al Kohli

A thesis submitted for the degree of PhD  
at the  
University of St Andrews



2024

Full metadata for this thesis is available in  
St Andrews Research Repository  
at:

<https://research-repository.st-andrews.ac.uk/>

Identifier to use to cite or link to this thesis:

DOI: <https://doi.org/10.17630/sta/870>

This item is protected by original copyright

This item is licensed under a  
Creative Commons Licence

<https://creativecommons.org/licenses/by-nc-nd/4.0/>



### **Candidate's declaration**

I, Raad Sameer Al Sheikh Al Kohli, do hereby certify that this thesis, submitted for the degree of PhD, which is approximately 23,000 words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for any degree. I confirm that any appendices included in my thesis contain only material permitted by the 'Assessment of Postgraduate Research Students' policy.

I was admitted as a research student at the University of St Andrews in January 2018.

I received funding from an organisation or institution and have acknowledged the funder(s) in the full text of my thesis.

Date Signature of candidate 05/04/2024

### **Supervisor's declaration**

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of PhD in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree. I confirm that any appendices included in the thesis contain only material permitted by the 'Assessment of Postgraduate Research Students' policy.

05-04-2024

Date Signature of supervisor

5th April 2024

### **Permission for publication**

In submitting this thesis to the University of St Andrews we understand that we are giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. We also understand, unless exempt by an award of an embargo as requested below, that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that this thesis will be electronically accessible for personal or research use and that the library has the right to migrate this thesis into new electronic forms as required to ensure continued access to the thesis.

I, Raad Sameer Al Sheikh Al Kohli, confirm that my thesis does not contain any third-party material that requires copyright clearance.

The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

#### **Printed copy**

No embargo on print copy.

#### **Electronic copy**

No embargo on electronic copy.

Date Signature of candidate

05/04/2024

Date Signature of supervisor

05-04-2024

5th April 2024

**Underpinning Research Data or Digital Outputs**

**Candidate's declaration**

I, Raad Sameer Al Sheikh Al Kohli, hereby certify that no requirements to deposit original research data or digital outputs apply to this thesis and that, where appropriate, secondary data used have been referenced in the full text of my thesis.

Date Signature of candidate

05/04/2024



*To Mum and Dad,*

## Abstract

In this thesis we shall study classes of groups defined by formal languages. Our first main topic is the class of groups defined by having an ET0L co-word problem; i.e., the class of co-ET0L groups. We show this class is closed under taking direct products and standard restricted wreath products with virtually free top groups. We also show the class is closed under passing to finite index overgroups and finitely generated subgroups. Further, we show that this class contains the free product  $\mathbb{Z}^n * \mathbb{Z}^m$  as well as  $\mathbb{Z}^n * G$  for any virtually free group  $G$ .

The second topic that we consider is a new class of groups that we introduce called epiregular groups. We show that this class contains all automatic groups and the Baumslag-Solitar group  $BS(1, 2)$ . Further we show that the class of epiregular groups is closed under taking graph products, and passing to finite index overgroups.

## Acknowledgements

I would like to thank my supervisors Collin Bleak and Martyn Quick for their support and advice over the years.

A special thanks goes to Collin Bleak and James Mitchell, thank you very much for your friendship.

I would also like to thank my aunt, Aishah Murtada, for her invaluable support throughout the years.

I would also like to thank Pilar Duque for all her support and encouragement. A thank you must also go to Niki Stalker and all the occupants of Office 226 for all of the chats. I would also like to thank Matthew McDevitt for many long and enjoyable discussions about various topics of mathematics and life, as well as teaching me about drawing automata in TikZ.

This work was supported by the University of St Andrews (School of Mathematics and Statistics, and the St Leonard's Scholarship).



## Notation

This merely serves as quick guide on where to find some definitions for functions that are used throughout this thesis.

- The function  $\rho$  is defined in Definition [2.3.9](#).
- The functions  $\gamma$  and  $\phi$  are defined in Definition [2.3.13](#).
- The function  $\varphi$  is defined in Definition [4.2.3](#).

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Background</b>	<b>9</b>
2.1	Preliminaries . . . . .	9
2.2	Formal Language Theory . . . . .	14
2.2.1	Regular Languages and Finite State Automata . . . . .	16
2.2.2	Context-free Languages and Pushdown Automata . . . . .	22
2.2.3	Indexed Languages and Nested Stack Automata . . . . .	31
2.2.4	ETOL Languages and Check-stack Pushdown Automata . . . . .	47
2.3	Classes of groups defined by languages . . . . .	65
2.3.1	Regular Groups . . . . .	67
2.3.2	Context-free Groups . . . . .	69
2.3.3	$co\mathcal{CF}$ Groups . . . . .	85
2.3.4	co-indexed Groups . . . . .	96
<b>3</b>	<b>Stack Groups</b>	<b>98</b>
3.1	Introduction . . . . .	98
3.2	Technical Lemma due to Holt–Röver . . . . .	100
3.2.1	Definitions . . . . .	100
3.2.2	Lemma . . . . .	102
3.3	Free Products . . . . .	107
3.3.1	Outline . . . . .	107
<b>4</b>	<b>co-ETOL Groups</b>	<b>111</b>
4.1	Introduction . . . . .	111
4.2	Direct Products . . . . .	112
4.2.1	Definition of Automaton . . . . .	113
4.2.2	Proof of Theorem 4.2.1 . . . . .	116
4.3	Finite Index Overgroups . . . . .	122
4.3.1	Definition of Automaton . . . . .	122
4.3.2	Proof of Theorem 4.3.1 . . . . .	126

4.4	Finitely Generated Subgroups	135
4.4.1	Definition of Automaton	135
4.4.2	Proof of Theorem 4.4.1	138
4.5	Wreath Products With Virtually Free Top Groups	143
4.5.1	Definition of Automaton	144
4.5.2	Proof of Theorem 4.5.1	153
4.6	The free product $\mathbb{Z}^n * \mathbb{Z}^m$	173
4.6.1	Definition of Automaton	174
4.6.2	Proof of Theorem 4.6.1	183
4.6.3	Examples	187
4.7	The free product $\mathbb{Z}^n * G$ for a virtually free $G$	194
4.7.1	Definition of Automaton	195
4.7.2	Proof of Theorem 4.7.1	209
<b>5</b>	<b>Epiregular Groups</b>	<b>214</b>
5.1	Introduction	214
5.2	Well-definedness of epiregularity	215
5.3	Examples	216
5.3.1	Automatic groups	218
5.3.2	Baumslag-Solitar group $BS(1, 2)$	220
5.4	Closure Properties	222
5.4.1	Graph Products	223
<b>6</b>	<b>Conclusion</b>	<b>227</b>
	<b>Bibliography</b>	<b>233</b>

# Chapter 1

## Introduction

The mathematics in this thesis lies in the intersection of formal language theory and group theory. There is a long history of study of the interaction between these two topics. It traces its origin back to 1911 when Dehn introduced several questions in [19]. One of those questions is the *word problem* which is as follows.

Let  $G$  be a finitely generated group and let  $X$  be finite generating set for  $G$ . Does there exist an algorithm for determining, in finite time, whether or not a product  $w$  in the generators  $X$  of  $G$  is equal to the identity of  $G$ ?

It is now known that no such algorithm exists in general. In particular, in [36], Novikov showed that there exists a finitely presented group  $G$  for which the word problem is undecidable. The word problem has been related to group structure through a number of results, the most prominent of which is perhaps the Boone-Higman theorem [12] stating that a finitely generated group  $G$  has solvable word problem if and only if there exists a simple group  $H$  and finitely presented group  $K$  such that  $G$  embeds into  $H$  and  $H$  embeds into  $K$ . It is now believed that a finitely generated group  $G$  has solvable word problem if and only if  $G$  embeds into some finitely presented simple group. This is referred to as the Boone-Higman conjecture [11, 12]. For further information regarding the Boone-Higman conjecture the reader may wish to consult [6]. The word problem of a group naturally gives rise to a set (which we describe below), and it is this set that forms the basis of what we study in this thesis.

Let  $A$  be a finite set. A subset  $L$  of the set  $A^*$  of finite strings of letters in  $A$  is referred to as a *formal language over  $A$* . If  $G$  is a group with finite generating set  $X$ , then when studying the word problem in  $G$ , one is naturally led to consider an associated formal language over the alphabet  $X^\pm$ . (We write  $X^\pm$  for the union of  $X$  with the set of inverses of elements of  $X$ .) This language will be denoted by  $WP(G, X)$  and will consist of all finite strings over  $X^\pm$  that are equal to the

identity when evaluated in  $G$ . Thus the word problem asks whether there exists an algorithm that can determine in finite time whether or not a string over the alphabet  $X^\pm$  is in  $WP(G, X)$ . It has become customary to call  $WP(G, X)$  *the word problem of  $G$  with respect to  $X$*  as well. Further we call the language that is the complement in  $(X^\pm)^*$  of  $WP(G, X)$  *the co-word problem of  $G$  with respect to  $X$* , and we denote it by  $coWP(G, X)$ .

Various classes of formal languages can be defined by different types of theoretical machines called automata. The least complicated class of automata studied is the class of finite state automata. We may think of a finite state automaton as a directed graph with edge and vertex labels, where we specify *initial* and *final vertices*. The edge labels correspond to letters in the alphabet that are read as one traverses a path from an initial vertex to a final vertex. We say a string  $\omega$  is accepted if  $\omega$  labels such a path; i.e., there is a path in the graph from an initial vertex to a final vertex where the concatenation of consecutive edge labels is  $\omega$ . A language is said to be accepted by the automaton if every string in the language is accepted. The class of languages accepted by finite state automata is called *the class of regular languages*. More complicated language types are defined by more complicated machine types. These more complicated machine types are typically generalisations of finite state automata, and they are obtained by adjoining various memory structures to a finite state automaton. The classes of languages that we will be particularly concerned with are the context-free languages and the ETOL languages. We shall not define these at this point in the introduction, but will define them in Chapter 2 below (see Definition [2.2.15](#) and Definition [2.2.42](#)).

The main objective in relating groups and their word problems and co-word problems from a language theoretic perspective is the following task.

Let  $\mathcal{C}$  be a class of languages. Classify the groups with word problem belonging to  $\mathcal{C}$  and similarly classify groups with co-word problem belonging to  $\mathcal{C}$ .

This started with a theorem of Anisimov [\[3\]](#) in 1971 (see Proposition [2.3.6](#) below). He proved that a finitely generated group  $G$  has regular word problem if and only if  $G$  is finite. More progress was made in the 1980s by Muller and Schupp [\[34\]](#) [\[35\]](#) (see Theorem [2.3.8](#) below). Muller and Schupp proved that a finitely generated group  $H$  has context-free word problem if and only if  $H$  is virtually free. In fact, a finitely generated group is virtually free if and only if the word problem is deterministic context-free. In 2005 Holt, Rees, Röver, and Thomas introduced  $co\mathcal{CF}$  groups in [\[27\]](#); i.e., groups whose co-word problem is context-free. Since virtually free groups have deterministic word problem, it follows that they have context free co-word problem. To date, there is still no classification of  $co\mathcal{CF}$  groups. However, there is a conjecture known as *Lehnert's Conjecture*. Lehnert originally conjectured in [\[31\]](#) that every  $co\mathcal{CF}$  group embeds into the

group of quasi-automorphisms of the infinite binary 2-coloured tree  $QAut(\mathcal{T}_{2,c})$ . This conjecture has been reinterpreted to the question of whether every  $co\mathcal{CF}$  group embeds in Thompson's group  $V$ . This is due to the Bleak, Matucci, and Neunhöffer in [9] who embedded  $QAut(\mathcal{T}_{2,c})$  into Thompson's group  $V$  as well as embedded Thompson's group  $V$  into  $QAut(\mathcal{T}_{2,c})$ .

In their paper, Holt, Rees, Rover, and Thomas conjectured that the class of  $co\mathcal{CF}$  groups is not closed under free products. In particular they conjectured that  $\mathbb{Z}^2 * \mathbb{Z}$  is not  $co\mathcal{CF}$ . This is now related to Lehnert's Conjecture due to the work of Bleak and Salazar-Diaz in [10] as they proved that there is no embedding from  $\mathbb{Z}^2 * \mathbb{Z}$  into Thompson's group  $V$ . In [28], Holt and Röver proved that  $\mathbb{Z}^2 * \mathbb{Z}$  (among other free products) is co-indexed. In the same paper, Holt and Rover also proved that bounded automata groups (a class of groups that contains Grigorchuk's group) are co-indexed.

The class of ET0L languages is strictly contained between indexed and context-free languages. In recent years, a number of papers have been published that take specific groups that have been proven to be co-indexed in [28] and show that they are in fact co-ET0L. An example of this is [17] where Ciobanu et al. prove that Grigorchuk's group is co-ET0L. This was then extended to proving that bounded automata groups are co-ET0L by Elder and Bishop in [8].

The work in this thesis is concerned with related questions. We shall study the class of co-ET0L groups in more depth, and introduce a new class of groups named epiregular groups. Epiregular groups are groups where there is finite state automaton accepting a string representing every non-trivial element in the group (while not accepting any string representing the identity of the group).

In Chapter 2, we give a detailed account of every type of automaton that we discuss. For each automaton type, we start by giving an intuitive description of the automaton. We then follow this with the formal definition. Then we link different parts of the formal definition with the intuitive descriptions of the automata. We also provide examples for some of the types of automata. Then we give an overview of the known results linking automata and groups as well as some proofs. For example, we give a proof of Anisimov's theorem as well as provide a pushdown automaton accepting the word problem for virtually free groups. This automaton is later used in different theorems in Chapter 3 (such as Theorem 4.5.1 and Theorem 4.7.1).

In Chapter 3, we give an intuitive description of the automaton accepting the co-word problem of free products of stack groups, as in [28]. The automata we construct for proving Theorem 4.6.1 and Theorem 4.7.1 are inspired by the construction in [28]. The automata used in Theorem 4.6.1 and Theorem 4.7.1 behave in a similar way to that used in [28] to prove that stack groups are closed under free products, and we include it for completeness.

In Chapter 4 we explore closure properties of the class of co-ET0L groups, and we obtain the following results. Our first result proves that the class of co-ET0L groups is closed under direct products.

**Theorem 4.2.1** *If  $G$  and  $H$  are co-ET0L groups, then  $G \times H$  is co-ET0L.*

We also prove that the class of co-ET0L groups is closed under passing to under finite index overgroups.

**Theorem 4.3.1** *Let  $G$  be a group and let  $H$  be a finite indexed subgroup of  $G$ . If  $H$  is a co-ET0L group, then  $G$  is a co-ET0L group.*

The proof of the above theorem uses a technique where we simulate reading a string in  $H$  upon reading generators of  $G$ . This technique is also used in the following theorem showing that the class of co-ET0L groups are closed under passing to finitely generated subgroups.

**Theorem 4.4.1** *Let  $G$  be a finitely generated group and  $H$  be a finitely generated subgroup of  $G$ . If  $G$  is co-ET0L, then  $H$  is co-ET0L.*

We also prove that the class of co-ET0L groups is closed under taking a standard restricted wreath product with a virtually free top group.

**Theorem 4.5.1** *Let  $B$  be a co-ET0L group and let  $T$  be a finitely generated virtually free group. Then the standard restricted wreath product  $W = B \wr T$  is co-ET0L.*

We note that closure results of this type are in [27] and [28] with respect to the appropriate machine types explored in those papers. That is, [27] and [28] show that the classes of groups considered are closed under the operations which we consider in the above theorems.

We also sharpen results about groups that were proven to be co-indexed in [28]. We do this by proving that they are in fact co-ET0L. This follows the example of [17] and [8], where they prove that Grigorchuk's group and bounded automata groups are in fact co-ET0L respectively. We do this by proving that some free products are in the class of co-ET0L groups.

**Theorem 4.6.1** *The free product  $\mathbb{Z}^n * \mathbb{Z}^m$  is co-ET0L.*

Using similar methods we also prove the following result.

**Theorem 4.7.1** *Let  $G$  be a virtually free group. Then the free product  $\mathbb{Z}^n * G$  is co-ET0L.*

We note that by Theorem [4.6.1](#) we know that  $\mathbb{Z}^2 * \mathbb{Z}$  is co-ETOL. This is a refinement on what was previously known about  $\mathbb{Z}^2 * \mathbb{Z}$ , as it was known that  $\mathbb{Z}^2 * \mathbb{Z}$  is co-indexed. It is yet unknown whether the conjecture in [27](#) regarding  $\mathbb{Z}^2 * \mathbb{Z}$  being not  $co\mathcal{CF}$  is true or not. However we believe our result moves us a step closer to the answer as we have reduced the complexity of the machine involved in accepting the co-word problem.

It would be interesting to investigate whether the restrictions posed in [28](#) to obtain a class that is closed under free products can be used to obtain an analogous subclass of co-ETOL groups. We believe this proposed subclass to be closed under free products as well as the operations in Theorem [4.2.1](#), Theorem [4.3.1](#), Theorem [4.4.1](#), and Theorem [4.5.1](#). We further believe that the constructions used to prove the aforementioned theorems may be used to prove analogous results for this proposed subclass, but perhaps with some modifications. However, we have not checked this.

In the final chapter this thesis, we introduce a new class of groups defined by machines. This class is called *epiregular groups*. These groups are defined by weakening the definition of groups whose co-word problem is regular. That is, we require that a finite state automaton accepts a string representing a non-trivial element, for every non-trivial element of the group. We also require that no string representing the identity of the group is accepted.

**Definition** [5.1.1](#) Let  $G$  be a finitely generated group. Let  $X$  be a finite (symmetrically closed) generating set for  $G$ . We say that  $G$  is *epiregular with respect to  $X$*  if there exists a finite state automaton  $\mathcal{A}$  such that

- $\mathcal{L}(\mathcal{A}) \cap WP(G, X) = \emptyset$
- for every non-identity element,  $h$ , there exists  $w_h \in coWP(G, X)$  such that  $w_h \in \mathcal{L}(\mathcal{A})$  and  $w_h =_G h$ .

The class of epiregular groups arose first in an analysis by Jim Belk of a construction of Ville Salo. Salo's construction uses a regular language  $R$ , where every element of  $R$  represents a nontrivial element of a right angled Artin group  $H$  (and every non-trivial element of  $H$  is represented by a string in  $R$ ) to embed it into the Brin-Thompson group  $2V$  [40](#). This construction sharpened the main result of Belk, Bleak and Matucci in [5](#), where for each right angled Artin group  $H$  they find a natural  $k$  (which grows at most quadratically in the number of generators of  $H$ ) and an embedding of  $H$  into  $kV$ . Given the utility of Salo's construction it is of interest to explore the class of epiregular groups for their own sake, and also to try to discern the feasibility of generalising Salo's construction so as to embed epiregular groups in general into some plausible class of finitely presented simple



groups (e.g., the Brin—Thompson groups  $nV$ ); possibly achieving some progress on the Boone—Higman conjecture.

We prove that epiregularity is independent of generating set.

**Theorem 5.2.1** *Let  $G$  be a finitely generated group, and let  $X$  and  $Y$  be finite symmetrically closed generating sets for  $G$ . Then  $G$  is epiregular with respect to  $X$  if and only if  $G$  is epiregular with respect to  $Y$ .*

We also prove that epiregular groups are closed under passing to finite index overgroups.

**Theorem 5.4.1** *Let  $H$  be an epiregular group. Let  $G$  be a finite index overgroup of  $H$ . Then  $G$  is epiregular.*

Further we also show that epiregular groups are closed under taking graph product of groups (see Definition 5.4.2).

**Theorem 5.4.10** *Let  $\Gamma$  be a finite graph as above. For every  $v_I \in V(\Gamma)$ , let  $G_I$  be an epiregular group with finite generating set  $X_I$ . Let  $G_\Gamma$  be the graph product of groups  $\{G_I \mid v_I \in V(\Gamma)\}$  over the graph  $\Gamma$ . Then  $G$  is epiregular.*

We also prove that the class of epiregular groups contains some well-studied classes of groups.

**Theorem 5.3.6** *Every automatic group is epiregular.*

It is well known that Baumslag Solitar groups in general are not automatic. In particular  $BS(1, 2)$  is not automatic. However, we prove the following result.

**Theorem 5.3.8** *The Baumslag-Solitar group  $BS(1, 2) = \langle a, b \mid b^{-1}ab = a^2 \rangle$  is epiregular.*

We note the two above theorems prove that the class of epiregular groups intersects with the class of  $co\mathcal{CF}$  groups. We know this since context-free groups form a subclass of  $co\mathcal{CF}$  groups, and context-free groups are virtually free and thus automatic, but the classes are not equal since  $BS(1, 2)$  is not  $co\mathcal{CF}$  as is shown in [27]. Further since  $BS(1, 2)$  is not automatic, it follows from the previous two theorems that the class of epiregular groups contains the class of automatic groups, but the classes are not equal. We state these as corollaries (Corollary 5.3.10 and Corollary 5.3.11).

# Chapter 2

## Background

This chapter serves to set the foundation for the rest of the thesis. We start with a short preliminaries section where we develop the basic terminology that will be used throughout the thesis. The other two sections concern formal language theory and group theory. In [2.2](#) we describe what is meant by a language and the various classes of languages that concern us, all defined using particular types of theoretical machines. In [2.3](#) we are primarily concerned with the classes of groups that we define in terms of these formal languages.

### 2.1 Preliminaries

We start by introducing some basic objects which will be used throughout the thesis. One of the most fundamental concepts we will need is that of an *alphabet* and *strings* over that alphabet. These are tied to elements of the free monoid. We shall elaborate further on this below.

Let  $\Sigma$  be a finite set, which we shall refer to as an *alphabet*. A *string* over the alphabet  $\Sigma$  is a finite sequence of symbols where every symbol belongs to  $\Sigma$ . The set of all strings over an alphabet  $\Sigma$  is denoted by  $\Sigma^*$ . We shall write a string  $\omega$  over an alphabet  $\Sigma$  as  $\omega = \sigma_1\sigma_2 \cdots \sigma_n$  where each  $\sigma_i \in \Sigma$ . We say that the *length* of this  $\omega$  is  $n$  and we write  $|\omega|$  for the length of  $\omega$ . Further, we call the unique string of length 0 the *empty string* and denote it by  $\varepsilon$ . Observe that equipped with the concatenation operation,  $\Sigma^*$  is the free monoid over  $\Sigma$  with  $\varepsilon$  being the identity of  $\Sigma^*$  as a monoid. Sometimes we will stress that strings ought to be considered as belonging to a free monoid by decorating the equals symbol with the free monoid as a subscript, e.g.  $=_{\Sigma^*}$ .

We do this formally here. Let  $M$  be a monoid with  $X \subseteq M$ . Then there exists a monoid homomorphism from  $X^*$  to  $M$  induced by  $x \mapsto x$ . If  $w_1, w_2 \in X^*$ , we write  $w_1 =_M w_2$  when they have the same image. We think of this as an evaluation in

$M$ . A special case of this is the following. Let  $\Sigma$  be an alphabet and  $M = \Sigma^*$  with identity  $\varepsilon$  (the empty string). Consider strings in  $\Sigma \cup \{\varepsilon\}$ , these are in  $(\Sigma \cup \{\varepsilon\})^*$ . For such  $w_1, w_2$ ,  $w_1 =_{\Sigma^*} w_2$  means  $w_1$  and  $w_2$  evaluate to the same string in  $\Sigma^*$ .

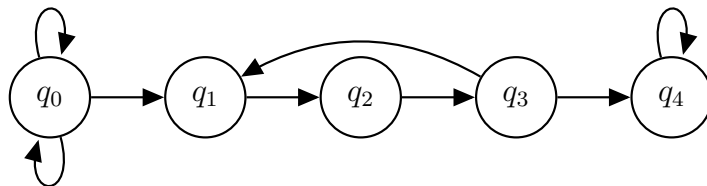
In what follows we introduce some graph theoretic notions which will help us in understanding finite state automata (Intuitive Description [2.2.1.1](#)). As we shall discuss in [2.2.1.1](#) we read a sequence of letters by concatenating along a path in an edge-labelled directed graph. The definitions below make these concepts precise. We start by defining *directed graphs*.

**Definition 2.1.1** (Directed Graph). A *directed graph*  $\Gamma$  is a pair  $(V, E)$  where  $V$  is called the *vertex set* of  $\Gamma$  and  $E \subseteq V \times V$  is called the *edge set* of  $\Gamma$ . We will sometimes refer to the vertex set of  $\Gamma$  by  $V(\Gamma)$ . Similarly, we will refer to the edge set of  $\Gamma$  by  $E(\Gamma)$ . We call elements of the vertex and edge sets *vertices* and *edges* respectively. If  $(v, w) \in E$  then we say that there's an *edge (or arrow)* from  $v$  to  $w$ . Finally, we say a directed graph is *finite* if the vertex set is finite.

We will draw finite directed graphs as figures where the vertices are represented by circles with the label of the vertex in the circle. An edge  $(v, w) \in E$  is represented by an arrow from the circle representing  $v$  to the one representing  $w$ .



**Example 2.1.2.**



We modify the definition above to obtain the following definition of *edge-labelled directed graph*.

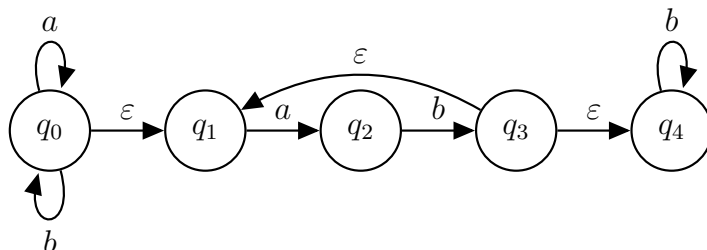
**Definition 2.1.3** (Edge-labelled Directed Graph). An *edge-labelled directed graph*  $\Gamma$  is a triple  $(V, \Sigma, E)$  where  $V$  is called the *vertex set*,  $\Sigma$  is called the *labelling set* and  $E \subseteq V \times \Sigma \times V$  is called the *edge set*. We will sometimes refer to the vertex set of  $\Gamma$  by  $V(\Gamma)$ . Similarly, we will refer to the edge set of  $\Gamma$  by  $E(\Gamma)$ . We call elements of the vertex and edge sets *vertices* and *edges* respectively. If  $(v, \sigma, w) \in E$  then there is an edge from  $v$  to  $w$  with label  $\sigma$  we say that there is a  $\sigma$  *edge (or arrow)* from  $v$  to  $w$ . Finally, we say an edge-labelled directed graph is *finite* if both the vertex set and the labelling set are finite.

We will draw finite edge-labelled directed graphs as figures where the vertices are represented by circles with the label of the vertex in the circle. An edge  $(v, \sigma, w) \in E$  is represented by an arrow from the circle representing  $v$  to the one representing  $w$  with  $\sigma$  on top of the arrow labelling it.



We give an example of a diagram representing a finite edge-labelled directed graph below.

**Example 2.1.4.**



We are now ready to define a path in an edge-labelled directed graph.

**Definition 2.1.5** (Path in an edge-labelled directed graph). Let  $\Gamma = (V, \Sigma, E)$  be an edge-labelled directed graph. Let  $v, w \in V$ . A *path* from  $v$  to  $w$  is a sequence of edges  $e_1, e_2, \dots, e_n \in E$  such that  $e_i = (v_{i-1}, \sigma_i, v_i)$ . We denote such a path by  $v_0 \xrightarrow{\sigma_1} v_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} v_n$ . ♣

As stated earlier we will concatenate along a path, we make this precise below.

**Definition 2.1.6** (Concatenation along a path). Let  $\Gamma = (V, \Sigma, E)$  be an edge-labelled directed graph. Let  $p = v_0 \xrightarrow{\sigma_1} v_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} v_n$  be a path. Then the *concatenation along the path  $p$*  is the string  $\sigma_1\sigma_2 \dots \sigma_n$ . ♣

**Definition 2.1.7** (Concatenation along a path with removal of  $\varepsilon$ ). Let  $\Gamma = (V, \Sigma \cup \{\varepsilon\}, E)$  be an edge-labelled directed graph. Let  $p = v_0 \xrightarrow{\sigma_1} v_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} v_n$  be a path. Then we say a string  $\omega$  is the *concatenation along the path  $p$  with removal of  $\varepsilon$ s* if  $\sigma_1\sigma_2 \dots \sigma_n =_{\Sigma^*} \omega$ , i.e., the string  $\omega$  is obtained from  $\sigma_1\sigma_2 \dots \sigma_n$  by deleting all occurrences of  $\varepsilon$  from  $\sigma_1\sigma_2 \dots \sigma_n$ . ♣

The distinction between the two definitions above is by removing occurrences of the symbol  $\varepsilon$  from a string, we are emphasising that the string should be viewed in the free monoid with  $\varepsilon$  being the identity of the monoid. As such, whenever we use the Definition [2.1.7](#) we are reminding the reader to view the string in the free monoid.

The following section explores formal language theory. Most definitions in the following section rely heavily on *relations* and properties such as *reflexivity* and *transitivity* of relations. We define these below together with the definition of *reflexive transitive closure* which we make use of when defining acceptance for various kinds of automata (as in Definition [2.2.6](#), Definition [2.2.19](#), Definition [2.2.32](#), and Definition [2.2.46](#)).

**Definition 2.1.8** (Relation, Reflexive and Transitive). Let  $X$  be a set. A *relation*  $\sim$  on  $X$  is a subset of  $X \times X$ . We write  $x \sim y$  to denote  $(x, y) \in \sim$ .

We say  $\sim$  is *reflexive* if  $x \sim x$  for all  $x \in X$ .

We say  $\sim$  is *transitive* if it satisfies the following condition: if  $x \sim y$  and  $y \sim z$  then  $x \sim z$ , for all  $x, y, z \in X$ . ♣

We now define the *reflexive transitive closure*.

**Definition 2.1.9** (Reflexive Transitive Closure). Let  $X$  be a set with  $\sim$  being a relation on  $X$ . The *reflexive transitive closure* of  $\sim$  is a relation  $\sim^*$  on  $X$  with the following two properties:

- $\sim^*$  is reflexive and transitive, and
- $\sim^*$  is the smallest reflexive transitive relation (with respect to containment) such that  $\sim \subseteq \sim^*$ .

♣

Throughout this thesis, we will make use of the notion of *prefixes*.

**Definition 2.1.10** (Prefixes). Let  $X$  be a finite set. Let  $w_1, w_2 \in X^*$  we say that  $w_1$  is a *prefix* of  $w_2$ , denoted  $w_1 \leq w_2$ , if and only if there exists a finite string (possibly the empty string)  $v \in X^*$  such that  $w_2 = w_1v$ . ♣

We will now recall some definitions from basic group theory as well as introduce some language that will be useful to us later.

All groups we discuss in this thesis shall be finitely generated and all generating sets shall be finite. Further, unless we specify otherwise all generating sets are *symmetrically closed*, i.e., have the following property: if the generating set contains an element  $g$  then it must also contain  $g^{-1}$ . Further we shall denote the identity element of a group  $G$  by  $1_G$  throughout.

Let  $G$  be a group with a subgroup  $H$  with  $H$  having finite index in  $G$ . We say  $G$  is a *finite index overgroup* of  $H$ .

In Chapter 4, we make use of free products and standard restricted wreath products to draw conclusions about closure results for a class of formal languages. Below we give the group theoretic background necessary for those arguments. We start with free products below.

**Definition 2.1.11** (Free products). The free product of  $G$  and  $H$  is a group  $F$  together with injective homomorphisms  $G \rightarrow F$  and  $H \rightarrow F$  such that any pair of homomorphisms from  $G$  and from  $H$  to a common group  $L$  can be extended to a homomorphism from  $F$  to  $L$ . ♣

We note that given groups  $G$  and  $H$ , the free product of  $G$  and  $H$  exists and is unique. We denote the free product by  $G * H$ . Further, the following theorem gives a presentation for the free product.

**Theorem 2.1.12.** *Let  $G := \langle X_G | R_G \rangle$  and  $H := \langle X_H | R_H \rangle$  be two presented groups. Set  $F$  to be the group presented by  $\langle X_G \cup X_H | R_G \cup R_H \rangle$ . Then  $F$  is the free product of  $G$  and  $H$ .*

Let  $G, H$  and  $F$  be as in the above theorem. Recall a word  $w$  in the generators of  $F$  is a product of the generators of  $F$  and can be expressed as

$$u_1 v_1 u_2 v_2 \cdots u_n v_n$$

where  $u_i \in X_G^*$  and  $v_i \in X_H^*$ , for  $i \in \{1, \dots, n\}$  and some  $n \in \mathbb{N}$ . (Recall that we have taken our generating sets to be symmetrically closed.)

We will introduce some terminology regarding the  $u_i$  and  $v_i$  above. However we need the following foundational definition.

**Definition 2.1.13** (Contiguous subword). Let  $G$  be a group with generating set  $X$ . Let  $w = x_1 x_2 \cdots x_n \in X^*$ . We say  $u \in X^*$  is a *contiguous subword* of  $w$  if  $w = u_1 u u_2$  for some  $u_1, u_2 \in X^*$ . ♣

We are now ready to introduce the terminology we need regarding words in a free product.

**Definition 2.1.14** (Syllables and Parts). Let  $F = G * H$  and a word  $w$  in the generators of  $F$  be

$$u_1 v_1 u_2 v_2 \cdots u_n v_n$$

where  $u_i \in X_G^*$  and  $v_i \in X_H^*$ , for  $i \in \{1, \dots, n\}$  and some  $n \in \mathbb{N}$ . We say that  $u_i$  and  $v_i$  are *syllables* for all  $i \in \{1, \dots, n\}$ .


Let  $x, y$  and  $z$  be syllables of a word  $w'$  in the generators of  $F$  such that  $xyz$  is a contiguous subword of  $w'$ . We say  $x$  and  $z$  are in the same *part* if  $y =_F 1$ . ♣

We now define standard restricted wreath products. We note that the definition relies on the definition of a semidirect product, but we shall not define that here as it can be found in most texts on group theory such as [37]. For the definition below, we use right actions.

**Definition 2.1.15.** [Standard restricted wreath product] Let  $C$  and  $T$  be groups where  $T$  acts on itself by right multiplication. Set  $B := \bigoplus_{t \in T} C_t$  where  $C_t \cong C$  for all  $t \in T$ . The group  $B$  is the direct product of the groups  $C_t$ , and all sequences in the direct product have the property that at most finitely many values are not

the identity. We define an action of  $T$  on  $B$  by right multiplication on the indices of  $(c_t)_{t \in T}$  as follows:

$$[(c_t)_{t \in T}] \cdot t_1 = (c_{tt_1^{-1}})_{t \in T}.$$

We set the *standard restricted wreath product* to be the group  $W$  defined by the semidirect product of  $B$  and  $T$  (with respect to the action defined above) and we write  $W = C \wr T$ . We call  $T$  the top group,  $C$  the bottom group and  $B$  the base group. 

We note that any element of the wreath product has a unique representation as a product  $xt$ , where  $x$  is in the base group and  $t$  is in the top group. We may view elements of the base group as sequences where every entry is in  $C$ , and only finitely many entries are allowed to be non-trivial elements of  $C$ .

We also make use of *Cayley graphs* in [2.3.1](#). We give the definition below.

**Definition 2.1.16** (Cayley Graph). Let  $G$  be a group with a generating set  $X$ . The *Cayley graph of  $G$  with respect to  $X$*  is the edge-labelled directed graph, and is denoted by  $\Gamma(G, X)$ , and is defined as follows. The vertex set of  $\Gamma(G, X)$  is  $G$ , the labelling set is  $X$  and the edges are defined as follows:

$$(g_1, x, g_2) \text{ is an edge if and only if } g_2 = g_1x.$$



## 2.2 Formal Language Theory

This section will be structured in the following way.

- We will first introduce what formal languages are.
- In each subsection, we introduce a formal language type, as well as a machine type (e.g. finite state automaton, pushdown automaton, nested stack automaton, etc.) which recognises languages of that type.
- For each machine type, we will do the following.
  1. First we will give an informal discussion of how that machine works, typically using an example.
  2. Then we will give a formal definition of the machine. This will be followed with
  3. a discussion as to how the intuition and formal definition are linked.
  4. Finally we will end each subsection with definitions that will be useful in helping with visualisation of the machines and their processes.

As mentioned above, we start by defining a language.

**Definition 2.2.1** (Language). A *language*  $L$  over an alphabet  $\Sigma$  is a subset of  $\Sigma^*$ .



In various places throughout this document we will use the reverse of a string and/or language.

**Definition 2.2.2** (Reverse of a string and language). Let  $L \subseteq \Sigma^*$  be a language and let  $\omega = \sigma_1\sigma_2 \cdots \sigma_n$  be a string in  $X^*$ . The *reverse of  $\omega$*  is the string  $\sigma_n\sigma_{n-1} \cdots \sigma_1$  and we denote the reverse of  $\omega$  by  $\omega^R$ . The *reverse of  $L$*  is the language  $\{\omega^R \mid \omega \in L\}$  and we denote the reverse of  $L$  by  $L^R$ .



Before exploring languages further, we shall set up some notation that will become useful later (e.g. Definition 2.2.15, Definition 2.2.26 and Definition 2.2.42). Let  $X$  be a finite set, then  $X^* = \{x_1x_2 \cdots x_n \mid x_i \in X \text{ and } n \geq 0\}$ . If  $A, B \subseteq X^*$  then we set

$$AB := \{\omega_1\omega_2 \mid \omega_1 \in A \text{ and } \omega_2 \in B\}.$$

If  $A \subseteq X^*$  and  $s$  is a symbol then we set

$$\begin{aligned} sA &:= \{s\omega \mid \omega \in A\}, \\ As &:= \{\omega s \mid \omega \in A\}, \end{aligned}$$

and other obvious analogous extensions.

We are now ready to give some examples of languages below.

**Example 2.2.3.** 1. The set of all strings over an alphabet  $\Sigma$  is a language. The empty set is a language over any alphabet.

2. Let  $\Sigma = \{0, 1\}$ . The set of all strings over  $\Sigma$  that contain 10 as a substring is a language.
3. Let  $\Sigma = \{0, 1\}$ . A *palindrome over  $\Sigma$*  is a string over  $\Sigma$  that is equal to its reverse, e.g. 11011 is a palindrome as reading it in reverse we get the same string. The set of all palindromes over  $\Sigma$  is a language.
4. Let  $\Sigma = \{0, 1, 2\}$ . The sets  $\{0^n1^n \mid n \in \mathbb{N}\}$  and  $\{0^n1^n2^n \mid n \in \mathbb{N}\}$  are languages.

One of the first classifications of formal languages is due Chomsky [16], and is usually referred to as the Chomsky hierarchy. This is a collection of four classes of languages contained in one another as follows:

Regular  $\subset$  Context-free  $\subset$  Context-Sensitive  $\subset$  Recursively Enumerable



We shall define the first two classes in this chapter, as well as two other subclasses of context-sensitive languages.

There are various ways to define different classes of languages. We will focus throughout on using “theoretical machines” called automata to define various classes of languages.

Informally, an automaton is a “theoretical machine” that reads a string, does some computation and either accepts or rejects the string. It is standard for various classes of languages to be defined by the kind of automata that accept those languages (we shall define what this means later). Different classes of automata provide different computational powers, enabling calculations that other classes do not. These computational powers are primarily determined by the presence (or lack thereof) of various types of memory structures, producing language classes with varying complexity.

## 2.2.1 Regular Languages and Finite State Automata

In this section, we will introduce regular languages. There are many (equivalent) ways of defining regular languages. We do this by defining finite state automata and defining the class of regular languages as the class of languages that are accepted by the class of finite state automata. Finite state automata are one of the best understood examples of automata. For a more comprehensive discussion of finite state automata, we redirect the reader to the monograph of Hopcroft and Ullman [29]. First we give an intuitive description of how finite state automata work.

### 2.2.1.1 Intuitive Description

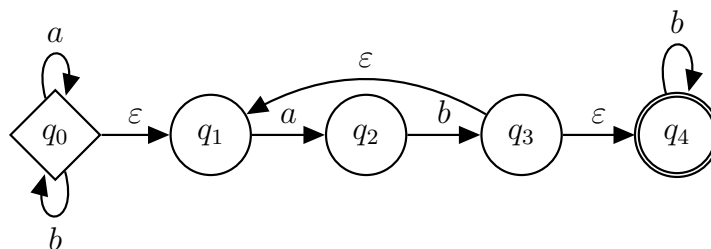
We may think of a finite state automaton as a finite edge-labelled directed graph with specified subsets of vertices called *initial* and *final* sets. These initial and final sets are denoted by  $I$  and  $F$  respectively. We note that when we think of graphs we refer to the vertices as vertices but when we wish to emphasise that the graph is carrying the structure of an automaton we may refer to the vertices as *states*. The edge labels come from a finite set  $\Sigma \cup \{\varepsilon\}$  (where  $\varepsilon \notin \Sigma$ ). We think of reading a string  $\omega \in \Sigma^*$  (from some vertex  $q \in I$ ) as traversing a path in the graph where  $\omega$  is the concatenation along that path with the removal of  $\varepsilon$ 's (as defined in Definition 2.1.7). Suppose upon reading  $\omega$  we traversed along the path  $v_0 \xrightarrow{x_1} v_1 \xrightarrow{x_2} \dots \xrightarrow{x_m} v_m$  such that  $x_1x_2 \dots x_m =_{\Sigma^*} \omega$  where  $v_0 \in I$ . We refer to  $v_i$  as being the *active state of the automaton* or the *state of the automaton* (or by similar language) having read  $x_1x_2 \dots x_i$  with  $x_{i+1}x_{i+2} \dots x_m$  yet to be read. For more complicated machine types, the analogous concept would be that of an *active configuration*. (We will define configurations to be pairs  $(q, \mathcal{S})$  where  $q$  is a

state and  $\mathcal{S}$  is the memory structure available for that machine type.) For more complicated machine types, when we use the phrase *state of the automaton* (or a similar one) we mean the state  $q$  in the active configuration  $(q, \mathcal{S})$ . We note that  $q$  will be the active state of the underlying finite state automaton. (We shall discuss this again in later sections.)

A string  $\omega = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^*$  is said to be *accepted* if there is a path in the graph starting at some  $q \in I$  and ending at some  $q' \in F$  where  $\omega$  is the concatenation along that path with the removal of  $\varepsilon$ 's. Finally, *the language accepted by the automaton* is the set of all accepted strings.

We provide an example below to explain the concepts we discussed above. For the purposes of demonstration, we will denote each vertex in  $I$  with a diamond instead of the usual circle. Similarly, we will denote each vertex in  $F$  with a double circle instead of a single circle.

**Example 2.2.4.**



Observe that  $q_0$  is the only initial vertex and  $q_4$  is the only final vertex. Thus to determine the language accepted by the above finite state automaton, we need to understand the paths from  $q_0$  to  $q_4$ . Let  $p$  be a path from  $q_0$  to  $q_4$ . Then we see that  $p$  can be broken down into three parts which are connected by an edge with  $\varepsilon$  as a label. The first is at  $q_0$ , where the edges  $q_0 \xrightarrow{a} q_0$  and  $q_0 \xrightarrow{b} q_0$  may have been used. The second is going through  $q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$  a non-zero number of times by using  $q_3 \xrightarrow{\varepsilon} q_1$ . Finally the last part is at  $q_4$ , where  $q_4 \xrightarrow{b} q_4$  could have been used.

Therefore all strings of the form  $xyz$  where  $x \in \{a, b\}^*$ ,  $y = (ab)^{n+1}$  for some  $n \in \mathbb{N}$  and  $z \in \{b\}^*$  are accepted.

It is important to note that finite state automata do not have a lot of memory structure built into them; we only have the structure of the graph itself available. Thus any information that we may need to store to help with computation has to be stored in the structure of the graph. This has an impact on the kinds of languages that can be accepted by finite state automata. We will expand more on this point when discussing various other kinds of automata. An example of a language that cannot be accepted by a finite state automaton is  $\{0^n 1^n \mid n \in \mathbb{N}\}$ . Note that for every string in this language, the number of occurrences of the letter

1 is the same as the number of occurrences as the letter 0. Using a finite state automaton, there is no way of storing the number of occurrences of the letter 0 in a string upon reading it. However we need to be able to do that in order to be able to accept a string with the correct number of occurrences of the letter 1.

We are now ready to give the formal definition of a finite state automaton.

### 2.2.1.2 Formal Definition

**Definition 2.2.5** (Non-Deterministic Finite State Automata). A *non-deterministic finite state automaton* (or NFSA)  $\mathcal{A}$  is a 5-tuple  $(Q, \Sigma, \delta, I, F)$  such that:

- $Q$  is a finite set, which is called the *set of states* of  $\mathcal{A}$ ;
- $\Sigma$  is a finite set, which is called the *alphabet* of  $\mathcal{A}$ ;
- $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\})) \times Q$  is called the *transition relation* of  $\mathcal{A}$  and we refer to an element of  $\delta$  as a *transition*;
- $I \subseteq Q$  is called the *set of initial states* of  $\mathcal{A}$ ; and
- $F \subseteq Q$  is called the *set of final (or accept) states*.



We note that transitions will become more complex in later definitions of other types of automata. This is to allow for more intricate processing needed with more complicated memory structures for various languages types.

We will use  $\delta$  to define a set of relations on  $Q$  defined as follows. For each  $x \in \Sigma \cup \{\varepsilon\}$ , we define a relation  $\sim_x$  on  $Q$  by

$$q \sim_x q' \text{ if and only if } ((q, x), q') \in \delta$$

for  $q, q' \in Q$ . We may say  $q \sim_x q'$  by  $((q, x), q')$ , or by similar language.

Let  $\sim_\varepsilon^*$  be the reflexive transitive closure of  $\sim_\varepsilon$ . We shall now use these relations to define acceptance in a finite state automaton.

**Definition 2.2.6** (Acceptance and Language accepted by a finite state automaton). Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  be a non-deterministic finite state automaton and let  $\omega = \sigma_1 \sigma_2 \cdots \sigma_n \in \Sigma^*$ . We say  $\omega$  is *accepted* by  $\mathcal{A}$  if there exist  $q_0, q_1, \dots, q_{2n+1} \in Q$  such that

$$q_0 \sim_\varepsilon^* q_1 \sim_{\sigma_1} q_2 \sim_\varepsilon^* q_3 \sim_{\sigma_2} q_4 \sim_\varepsilon^* \cdots \sim_{\sigma_n} q_{2n} \sim_\varepsilon^* q_{2n+1},$$

where  $q_0 \in I$  and  $q_{2n+1} \in F$ .

We write  $\mathcal{L}(\mathcal{A})$  for the set of all strings in  $\Sigma^*$  that are accepted by  $\mathcal{A}$ . We say that  $\mathcal{L}(\mathcal{A})$  is the *language accepted by  $\mathcal{A}$* . 

If there exists a sequence of states  $q_0, q_1, \dots, q_{2n+1} \in Q$  such that

$$q_0 \sim_\varepsilon^* q_1 \sim_{\sigma_1} q_2 \sim_\varepsilon^* q_3 \sim_{\sigma_2} q_4 \sim_\varepsilon^* \dots \sim_{\sigma_n} q_{2n} \sim_\varepsilon^* q_{2n+1}, \quad (\hat{1})$$

where  $\omega = \sigma_1, \sigma_2, \dots, \sigma_n \in \Sigma^*$  then we say  $\omega$  can be read from  $q_0$  by the automaton through the chain  $(\hat{1})$ , or by similar language. Note that for a string to be accepted, the automaton must be able to read the string from an initial state and the last state in a chain of relations (such as  $(\hat{1})$ ) is an accept state.

We conclude this section with a notational remark. We will make use of the term  $\varepsilon$ -transition (we will revisit this term again in the following section). A transition  $((s, x), t)$  is said to be an  $\varepsilon$ -transition if  $x = \varepsilon$ . Suppose in a chain such as the one above in  $(\hat{1})$ , there exists a state  $q$  appearing as  $q \sim_\varepsilon^* q$  in the chain. Further suppose there do not exist states  $q'_1, q'_2, \dots, q'_r$  and there does not exist a sequence of  $\varepsilon$ -transitions yielding

$$q \sim_\varepsilon q'_1 \sim_\varepsilon q'_2 \sim_\varepsilon \dots \sim_\varepsilon q'_r \sim_\varepsilon q.$$

Then  $q \sim_\varepsilon^* q$  appears in the chain due to the reflexive property of  $\sim_\varepsilon^*$ . In such a situation, we will remove the appearance of  $\sim_\varepsilon^* q$  from the chain to shorten it.

### 2.2.1.3 Link between intuition and formality for NFSAs

Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  be a non-deterministic finite state automaton. Recall that we intuitively think of  $\mathcal{A}$  as finite edge-labelled directed graph  $\Gamma$  with specified subsets of vertices called initial and final sets. We think of the set of states  $Q$  as the labelled vertices of  $\Gamma$ . The edge labels of  $\Gamma$  come from  $\Sigma \cup \{\varepsilon\}$ . A transition  $\rho = ((q, x), q') \in \delta$  corresponds to an edge from  $q$  to  $q'$  with label  $x$  in  $\Gamma$ . (This edge is denoted by  $q \xrightarrow{x} q'$ .) Due to this graph theoretic interpretation, we will refer to  $\rho \in \delta$  as being a transition from  $q$  to  $q'$  by  $x$ , or by similar language. When  $x = \varepsilon$ , we call  $\rho$  an  $\varepsilon$ -transition. Conversely when  $x \neq \varepsilon$ , we refer to  $\rho$  as a reading-transition. We note that we use similar language later when describing transitions of more complicated machine types as well. Further, we will sometimes refer to  $q_1 \xrightarrow{\varepsilon} q_2$  as an  $\varepsilon$ -arrow.

Let  $p, q \in Q$  such that  $p \sim_\varepsilon^* q$ . This corresponds to there existing a path in  $\Gamma$

$$p_1 \xrightarrow{\varepsilon} p_2 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} p_t$$

where  $p_1 = p$  and  $p_t = q$ , or  $p = q$  as we discussed at the end of the last subsection.

**Example 2.2.7.** Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  be a finite state automaton such that:

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$ ;
- $\Sigma = \{a, b\}$ ;

- The transition relation  $\delta$  consists of the following transitions:

$$\begin{aligned} &((q_0, a), q_0), ((q_0, b), q_0), \\ &((q_0, \varepsilon), q_1), ((q_1, a), q_2), \\ &((q_2, b), q_3), ((q_3, \varepsilon), q_1), \\ &((q_3, \varepsilon), q_4), ((q_4, b), q_4); \end{aligned}$$

- $I = \{q_0\}$ ; and
- $F = \{q_4\}$ .

This defines the finite state automaton corresponding to the graph in Example [2.2.4](#) (as we shall make precise below).

#### 2.2.1.4 Visualisation, Equivalence with Graphs, Determinism and Regular Languages

We will now make our intuition more precise with the following formal definition.

**Definition 2.2.8** (Transition Diagram/State Diagram). Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  be a non-deterministic finite state automaton. We will define the *transition diagram* (or *state diagram*) to be an edge-labelled directed graph  $\Gamma(\mathcal{A})$  as follows:

- The vertex set is  $Q$ .
- The edges of  $\Gamma(\mathcal{A})$  are all triples  $(q, x, q')$  such that  $((q, x), q') \in \delta$ . We shall denote the set of edges by  $E$ . We represent an edge  $(q, \sigma, q')$  by an arrow from  $q$  to  $q'$  with  $\sigma$  as its label.

We represent transition diagrams by figures of edge-labelled directed graphs. The vertices of the graphs will be denoted by circles with the corresponding states written in them. Each vertex representing a state in  $I$  will be denoted by a diamond instead of the usual circle. Similarly each vertex representing a state in  $F$  will be denoted by a double circle instead of a single one. ♣

An example of a transition diagram of a finite state automaton can be found in Example [2.2.4](#) where we give the transition diagram for the finite state automaton given in Example [2.2.7](#).

Note that given a finite edge-labelled directed graph with specified initial and final sets, we can determine the transition relation for an automaton by reversing the above process. Therefore, finite state automata do indeed correspond to graphs of the aforementioned type.

The reader will observe that in Definition [2.2.5](#) for NFSA, one has to account for  $\varepsilon$ -transitions, i.e., a transition of the form  $((q, \varepsilon), q')$ . Note that a transition  $((q, \varepsilon), q')$  represents changing state from  $q$  to  $q'$  without reading any input. Indeed, if one could remove the possibility of these transitions occurring, the automaton (in theory) would be more straightforward to understand. Such an automaton is sometimes referred to as a “non-deterministic finite state automaton without  $\varepsilon$ -transitions”.

The following is a classical result in the theory of finite state automata.

**Proposition 2.2.9.** [\[29\]](#) *Theorem 2.2]* Let  $\mathcal{A}_1$  be the class of non-deterministic finite state automata with  $\varepsilon$ -transitions and  $\mathcal{A}_2$  be the class non-deterministic finite state automata without  $\varepsilon$ -transitions. Then a language  $L$  is accepted by an automaton belonging to  $\mathcal{A}_1$  if and only if it is accepted by an automaton belonging to  $\mathcal{A}_2$ .

**Definition 2.2.10** (Deterministic FSA and Acceptance for Deterministic FSA). Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  be a non-deterministic finite state automaton without  $\varepsilon$ -transitions. If for every  $q \in Q$  and  $\sigma \in \Sigma$ , there exists a unique  $q' \in Q$  such  $((q, \sigma), q') \in \delta$  then we say that the finite state automaton is *deterministic*.

If  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  is a deterministic finite state automaton then a string  $\omega = \sigma_1 \sigma_2 \cdots \sigma_n$  is *accepted* if and only if there exist  $q_0, q_1, \dots, q_n \in Q$  such that

$$q_0 \sim_{\sigma_1} q_1 \sim_{\sigma_2} \cdots \sim_{\sigma_n} q_n$$

where  $q_0 \in I$  and  $q_n \in F$ . ♣

Suppose  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  is a deterministic finite state automaton. Since  $\mathcal{A}$  is deterministic, there are no  $\varepsilon$ -transitions and for every  $q \in Q$  and  $\sigma \in \Sigma$  there exists a unique  $q'$  such that  $((q, \sigma), q')$  is a transition. Thus every string can be read in a unique way from an initial state  $q_0$ . Suppose the string  $\omega = \sigma_1 \sigma_2 \cdots \sigma_n$  is read through the following chain

$$q_0 \sim_{\sigma_1} q_1 \sim_{\sigma_2} \cdots \sim_{\sigma_n} q_n. \tag{\hat{2}}$$

Observe there are no  $\sim_{\varepsilon}^*$  relations in the above chain. This is due there being no  $\varepsilon$ -transitions in  $\mathcal{A}$ . Therefore we would only use the reflexive property of  $\sim_{\varepsilon}^*$ , and the chain would have been

$$q_0 \sim_{\varepsilon}^* q_0 \sim_{\sigma_1} q_1 \sim_{\varepsilon}^* q_1 \sim_{\sigma_2} q_2 \sim_{\varepsilon}^* \cdots \sim_{\sigma_n} q_n \sim_{\varepsilon}^* q_n.$$

Thus we only write the chain  $(\hat{2})$  for short. Finally observe that  $\omega$  is accepted if the last state in the chain  $(\hat{2})$  is an accept state.

A non-deterministic finite state automaton is indeed equivalent to a deterministic one. One can convert a non-deterministic finite state automaton to a deterministic one via a standard construction known as the *powerset construction*.

**Theorem 2.2.11.** [29, Theorems 2.1 & 2.2] *The class of languages accepted by deterministic finite state automata is equal to the class that is accepted by non-deterministic finite state automata.*

Before continuing our discussion we remark that as a consequence of the powerset construction, the initial set of states  $I$  of a finite state automaton  $\mathcal{A}$  may be assumed to have size 1.

We are now ready to define regular languages. Due to Theorem 2.2.11 we will not specify whether the finite state automaton in the definition below needs to be deterministic or not.

**Definition 2.2.12** (Regular Languages). A language  $L$  is said to be *regular* if there is a finite state automaton  $\mathcal{A}$  such that  $L = \mathcal{L}(\mathcal{A})$ . ♣

Note that we shall use “(N)FSA” to mean a *(non-deterministic) finite state automaton* or the *class of (non-deterministic) finite state automata* and it will be clear from context which is meant when it arises.

Due to Theorem 2.2.11, we may think of regular languages as the class of languages accepted by deterministic finite state automata. Let  $R$  be a regular language, thus there exists a deterministic finite state automaton  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  such that  $\mathcal{L}(\mathcal{A}) = R$ . In particular, due to the powerset construction, we may assume  $|I| = 1$ . Further as  $\mathcal{A}$  is deterministic, reading a string  $\omega$  from any state  $q$  uniquely determines the state  $q'$  at end of the path  $p$  in the transition diagram  $\Gamma(\mathcal{A})$ , where  $\omega$  is the concatenation along  $p$ . We shall now construct a deterministic finite state automaton  $\mathcal{A}'$  such that  $\mathcal{L}(\mathcal{A}') = \Sigma^* \setminus R$ . Set  $F' := Q \setminus F$ . Let  $\mathcal{A}' := (Q, \Sigma, \delta, I, F')$ . Note that any string  $\omega \in \Sigma^*$  such that  $\omega \in R$ , then upon reading  $\omega$  one follows a path that ends in  $F$  and hence cannot be accepted by  $\mathcal{A}'$  as  $F \cap F' = \emptyset$ . Conversely, every string  $\omega' \in \Sigma^* \setminus R$  is accepted by  $\mathcal{A}'$  since the path followed ends in  $F'$  and  $F \cap F' = \emptyset$ . Therefore a corollary of the powerset construction is the following result.

**Proposition 2.2.13.** *The class of regular languages is closed under complementation.*

## 2.2.2 Context-free Languages and Pushdown Automata

In the Chomsky hierarchy, the class of regular languages is contained in the class of context-free languages. We shall focus on these languages in this subsection. Similar to regular languages, there are different (equivalent) ways of defining context-free languages. We define this class of languages as the class that is accepted by the class of pushdown automata. First we give an intuitive description of how pushdown automata work.

### 2.2.2.1 Intuitive Description

We can think of a pushdown automaton as a finite state automaton with an added memory structure, which we call a *stack*. The stack operates on a last-in-first-out basis. We may think of the stack as a list of letters from an alphabet set positioned one above the other. We will use a string to encode the information on a stack. The left end of the string will represent the top of the stack and is active. Conversely, the right end of the string will represent the bottom of the stack. The alphabet set used on the stack is finite and is referred to as *the stack alphabet*. Note that the stack alphabet need not be equal to the input alphabet. We will denote the stack alphabet by  $\chi$ . At any point in time, we can only view the top of the stack. Initially, the stack only contains a unique symbol, which we call *the bottom-of-stack symbol* and is denoted by  $\perp$ . When the stack contains  $\perp$  and no other symbol, we say the stack is empty.

A transition in a pushdown automaton not only changes the active state the machine is in but also rewrites the top of the stack. This is done by removing the top letter off the stack and replacing it with a string (which maybe empty). We note that  $\perp$  cannot be deleted nor can it be written in the middle of the stack. This will be enforced by transition rules. Recall in a finite state automaton, a transition  $((p, x), q)$  can be interpreted as reading a letter  $x \in \Sigma \cup \{\varepsilon\}$  from a state  $p$  and moving to a state  $q$ . In a pushdown automaton, transitions also depend on the letter at the top of the stack. Thus a transition in a pushdown automaton can be interpreted as reading a letter  $x \in \Sigma \cup \{\varepsilon\}$  from a state  $p$  with a letter  $y$  at the top of the stack, and moving to a state  $q$  while writing a string  $\omega$  on the stack after deleting  $y$ . Note that in a pushdown automaton, we can interpret a transition as a pair  $(X, Y)$ , where  $X$  is a triple and  $Y$  is a pair, which we shall describe in greater detail. The first component  $X$  has the form  $(p, x, y)$  while the second component  $Y$  has the form  $(q, \omega)$  (as in Definition 2.2.15). We note that we cannot have a transition if the first component does not match the first component of elements of  $\delta$  (in Definition 2.2.15.)

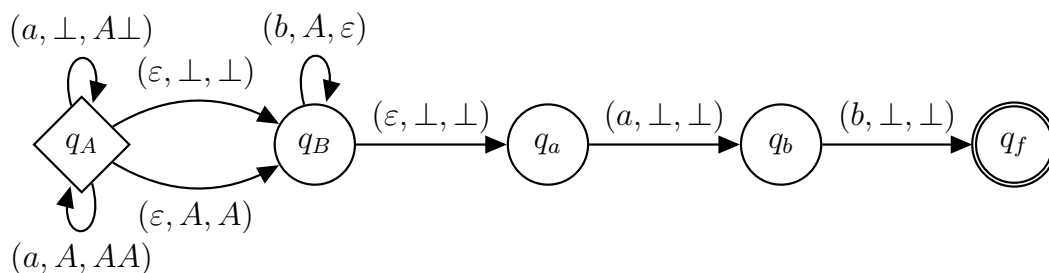
In a pushdown automaton, we interpret reading a string  $\omega = \sigma_1\sigma_2\cdots\sigma_n$  as traversing a path  $v_0 \xrightarrow{x_1} v_1 \xrightarrow{x_2} \cdots \xrightarrow{x_m} v_m$  in the graph of the underlying finite state automaton such that  $x_1x_2\cdots x_m =_{\Sigma^*} \omega$  where  $v_0$  is an initial state while editing the stack. We refer to  $v_i$  as being the *active state of the automaton* or the *state of the automaton* (or by similar language) having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read. The stack will be edited by adding or removing letters from the top of the stack as per the transitions. We call the stack associated to the automaton having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read *the active stack*. The *active configuration* is a pair  $(q, \mathcal{S})$  where  $q$  and  $\mathcal{S}$  are the active state and the active stack respectively having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read.



A string  $\omega' \in \Sigma^*$  is said to be *accepted* by a pushdown automaton if we can get from an initial state to a final state via a sequence of transitions where the concatenation of consecutive input letters is  $\omega'$ . Note that initially, the stack is empty. Finally, *the language accepted by the automaton* is the set of all accepted strings.

We provide an example below to explain the concepts we discussed above. Here we shall use an underlying edge-labelled directed graph structure similar to how we interpreted finite state automata. However, we add an extra decoration to the edge labels, where we will write  $(x, y, \omega)$  to denote reading  $x$  from the input alphabet, with  $y$  on the stack and writing a string  $\omega$  on the stack after deleting  $y$ . Further, similar to a finite state automaton, we shall draw each initial state with a diamond instead of the usual circle, and we will draw each final state with a double circle instead of a single one.

**Example 2.2.14.** In the example below the input alphabet is  $\{a, b\}$  while the stack alphabet is  $\{A\}$ .



First observe that  $q_A$  is the only initial state and  $q_f$  is the only final state. Thus to determine the language accepted by the above pushdown automaton, we need to understand the paths from  $q_A$  to  $q_f$ . Let  $p$  be a path from  $q_A$  to  $q_f$ . We note that  $p$  can be broken down into two main parts. The first is from  $q_A$  to  $q_a$  and the second is from  $q_a$  to  $q_f$ . We note that the edge  $q_B \xrightarrow{(b, A, \varepsilon)} q_B$  deletes an occurrence of  $A$  from the stack upon reading a  $b$  from the input. Thus  $q_B$  will be used to read as many occurrences of the letter  $b$  as there were of the letter  $a$ . Then the stack will be empty and the transition  $q_B \xrightarrow{(\varepsilon, \perp, \perp)} q_a$  can be used. Therefore, at  $q_a$  the string that was read by the automaton was  $a^k b^k$  for some  $k \in \mathbb{N}$ . The second part of the path  $p$  is from  $q_a$  to  $q_f$ , this simply allows us to read the suffix  $ab$ , i.e., the graph structure of the automaton is used to append  $ab$  at the end. Therefore the language accepted by the automaton is  $\{a^n b^n ab \mid n \in \mathbb{N}\}$ .

While pushdown automata have more memory than finite state automata, they are still limited by being able to view the top most letter of the stack and nothing else. In later subsections, we shall weaken this constraint.

In the following subsection, we shall formalise the intuition provided above.

### 2.2.2.2 Formal Definition

Recall the notation set up after Definition 2.2.2. In particular, with regards to the definition below

$$\chi^* \perp = \{\omega \perp \mid \omega \in \chi^*\} \subseteq (\chi \cup \{\perp\})^*.$$

**Definition 2.2.15** (Pushdown Automata). A *pushdown automaton* (or *PDA*)  $\mathcal{A}$  is a 7-tuple  $(Q, I, \Sigma, \chi, \perp, \delta, F)$  such that:

- $Q$  is a finite set, which is called the *set of states* of  $\mathcal{A}$ ;
- $I \subseteq Q$  is called the *set of initial states* of  $\mathcal{A}$ ;
- $\Sigma$  is a finite set, which is called the *tape (or input) alphabet* of  $\mathcal{A}$ ;
- $\chi$  is a finite set, which is called the *stack alphabet* of  $\mathcal{A}$ ;
- $\perp \notin \chi$  is the *bottom of stack symbol*;
- $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times (\chi \cup \{\perp\})) \times (Q \times (\chi^* \cup \chi^* \perp))$  is a finite set called the *transition relation* and we refer to an element of  $\delta$  as a *transition*. Further let  $p, q \in Q$  and  $x \in \Sigma \cup \{\varepsilon\}$ . If  $((p, x, y), (q, \omega)) \in \delta$  then one of the following two conditions holds:
  1.  $y \in \chi$  and  $\omega \in \chi^*$ ,
  2.  $y = \perp$  and  $\omega \in \chi^* \perp$ ; and
- $F \subseteq Q$  is called the *set of final (or accept) states*.

The *stack* is a string  $S = s_n s_{n-1} \cdots s_0$  such that  $s_0 = \perp$  and  $s_i \in \chi$  for  $i > 0$ . We define the *top* (of the stack) to be the left end of the string and the *bottom* (of the stack) to be the right end. ♣

We shall define *configurations* below. This is the first step in defining acceptance of strings by pushdown automata.

A configuration gives us the full information about the PDA at any given point.

**Definition 2.2.16** (Configuration). A configuration is a pair  $(q, S)$  where  $q \in Q$  and  $S \in \chi^* \perp$  (is the stack). ♣

We will now use  $\delta$  to define a set of relations on the set of configurations below.

For each  $x \in \Sigma \cup \{\varepsilon\}$ , we define a relation  $\sim_x$  on the set of configurations as follows. Let  $C_1 = (q_1, S_1)$  and  $C_2 = (q_2, S_2)$  be two configurations. Then  $C_1 \sim_x C_2$  if and only if the following condition holds:

- there exists an initial letter  $\mu_1$  of  $S_1$  and strings  $\mu_2$  and  $\kappa$  such that  $S_1 = \mu_1\kappa, S_2 = \mu_2\kappa$  and  $((q_1, x, \mu_1), (q_2, \mu_2)) \in \delta$ .

We may say  $C_1 \sim_x C_2$  by  $((q_1, x, \mu_1), (q_2, \mu_2))$ , and/or *the automaton uses the transition  $((q_1, x, \mu_1), (q_2, \mu_2))$  (from  $C_1$  to  $C_2$ )*, or by similar language. Let  $\sim_\varepsilon^*$  be the reflexive transitive closure of  $\sim_\varepsilon$ . To complete our discussions of configurations, we will define *initial* and *accepting configurations*.

**Definition 2.2.17** (Initial Configuration). Let  $C = (q, S)$  be a configuration. We say  $C$  is an *initial configuration* if  $q \in I$  and  $S = \perp$ . ♣

**Definition 2.2.18** (Accepting Configuration). Let  $C = (q, S)$  be a configuration. We say  $C$  is an *accepting configuration* if  $q \in F$ . ♣

We are now ready to define a version of acceptance.

**Definition 2.2.19** (Acceptance and Language accepted by a PDA). Let  $\mathcal{A} = (Q, I, \Sigma, \chi, \perp, \delta, F)$  be a pushdown automaton and let  $\omega = \sigma_1\sigma_2 \cdots \sigma_n \in \Sigma^*$ . We say  $\omega$  is *accepted by  $\mathcal{A}$*  if there exist configurations  $C_0, C_1, \dots, C_{2n+1}$  such that

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* C_3 \sim_{\sigma_2} C_4 \sim_\varepsilon^* \cdots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1},$$

where  $C_0$  is an initial configuration and  $C_{2n+1}$  is an accepting configuration.

Let  $\mathcal{L}(\mathcal{A})$  be the set of all strings in  $\Sigma^*$  that are accepted by  $\mathcal{A}$ . We say  $\mathcal{L}(\mathcal{A})$  is the *language accepted by  $\mathcal{A}$* . ♣

The above definition of acceptance is sometimes referred to as *accepting via final state*. It is also possible to define acceptance as follows.

**Definition 2.2.20** (Acceptance by emptying the stack). Let  $\mathcal{A}$  be a pushdown automaton such that  $A = (Q, I, \Sigma, \chi, \perp, \delta, F)$  where  $F = Q$ . Let  $\omega = \sigma_1\sigma_2 \cdots \sigma_n \in \Sigma^*$ . We say  $\omega$  is *accepted by  $\mathcal{A}$  via emptying the stack* if there exist configurations  $C_0, C_1, \dots, C_{2n+1}$  such that

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* \cdots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1},$$

where  $C_0$  is an initial configuration and  $C_{2n+1} = (q, \perp)$  is an accepting configuration.

Let  $\mathcal{L}_\perp(\mathcal{A})$  be the set of all strings in  $\Sigma^*$  that are accepted by  $\mathcal{A}$  via emptying the stack. We say a *language  $L$  is accepted by  $\mathcal{A}$  via emptying the stack* if  $\mathcal{L}_\perp(\mathcal{A}) = L$ . ♣

If there exists a sequence of configurations  $C_0, C_1, \dots, C_{2n+1} \in Q$  such that

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* \dots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1}, \quad (\hat{3})$$

where  $\sigma_1, \sigma_2, \dots, \sigma_n \in \Sigma$  then we say  $\omega = \sigma_1 \sigma_2 \dots \sigma_n$  can be read from  $C_0$  by the automaton through the chain  $(\hat{3})$ , or by similar language. The chain of relations (see e.g.  $(\hat{3})$ ) used to read a string is called a *run* of the automaton, we sometimes refer to it as a *run* and sometimes we simply just say *chain* or *chain of relations*. Note that for a string to be accepted, the automaton must be able to read the string from an initial initial configuration and the last configuration in a chain of relations (such as  $(\hat{3})$ ) is an accept configuration. If a string cannot be read from an initial configuration then it can never be accepted. If a string cannot be read in a particular run (or any runs), we may say the string is *rejected*. Usually if the term *rejected* is used, it simply means that there is some underlying reason as to whether the string cannot be accepted that is fundamentally opposed to how the machine operates. One of the reasons can be that the string simply cannot be read from an initial configuration (in a particular run). If there are other reasons, they will be made clear in the contexts in which they arise. It will sometimes be useful to designate a state  $q_r$  as *the reject state*. This state will not be a final state of the automaton, and the automaton will have transitions to move to  $q_r$  when the input string is to be rejected. A reject state  $q_r$  will be a *sink state*. A sink is a state from which no other state can be reached. We say a configuration  $C$  is a *sink*, if no other configuration can be reached from  $C$ .

We will now make a notational remark. We will make use of the term  $\varepsilon$ -transition (we will revisit this term again in the following section). A transition  $((q, x, y), (q', w))$  is said to be an  $\varepsilon$ -transition if  $x = \varepsilon$ . Suppose in a chain such as the one above in  $(\hat{3})$ , there exists a configuration  $C$  appearing as  $C \sim_\varepsilon^* C$  in the chain. Further suppose there do not exist configurations  $C'_1, C'_2, \dots, C'_r$  and there does not exist a sequence of  $\varepsilon$ -transitions yielding

$$C \sim_\varepsilon C'_1 \sim_\varepsilon C'_2 \sim_\varepsilon \dots \sim_\varepsilon C'_r \sim_\varepsilon C.$$

Then  $C \sim_\varepsilon^* C$  appears in the chain due to the reflexive property of  $\sim_\varepsilon^*$ . In such a situation, we will remove the appearance of  $\sim_\varepsilon^* C$  from the chain to shorten it.

Note that one may define acceptance in pushdown automata using a slightly different concept called *instantaneous descriptions*. However, here we have chosen to use configurations for the purpose of consistency with other definitions of acceptance in this section.

There is a well-known result in the theory of pushdown automata [29] [Theorems 5.1 & 5.2] stating that the two methods of acceptance defined above are equivalent. For our purposes, we will choose to define pushdown automata that accept by final states, unless otherwise stated.

**Definition 2.2.21** (Context free languages). A language  $L$  is said to be *context-free* if there exists a pushdown automaton  $\mathcal{A}$  such that  $L = \mathcal{L}(\mathcal{A})$ . ♣

Note that we shall use “PDA” to mean *a pushdown automaton* or *the class of pushdown automata* and it will be clear from context which is meant when it arises.

We shall link the formal definition to our intuitive description of pushdown automata in the following subsection.

### 2.2.2.3 Link between intuition and formality for PDA

Let  $\mathcal{A} = (Q, I, \Sigma, \chi, \perp, \delta, F)$  be a pushdown automaton. Like finite state automata,  $\mathcal{A}$  has a set of states  $Q$ , an input alphabet  $\Sigma$ , a set of initial states  $I$  and a set of final states  $F$ . However, as discussed in our intuitive description, the main difference is the existence of a stack. The stack alphabet is  $\chi$ .

A transition  $\alpha = (\beta, \gamma)$  is an element of  $\delta$ . Recall that  $\beta = (p, x, y) \in Q \times (\Sigma \cup \{\varepsilon\}) \times (\chi \cup \{\perp\})$ . We interpret this as reading  $x \in \Sigma \cup \{\varepsilon\}$  with  $y \in \chi \cup \{\perp\}$  at the top of the stack from state  $p$ . Recall that if  $\perp$  is at the top of the stack, then we say the stack is empty. Further recall that  $\gamma = (q, \omega) \in Q \times (\chi^* \cup \chi^* \perp)$ . We interpret this as moving to a state  $q$  and replacing  $y$  by  $\omega \in \chi^* \cup \chi^* \perp$ . We refer to  $\alpha$  as a transition from  $p$  with  $y$  at the top of the stack to  $q$  by  $x$  writing  $\omega$ , or by similar language. When  $x = \varepsilon$ , we call  $\alpha$  an  $\varepsilon$ -transition. Conversely when  $x \neq \varepsilon$ , we refer to  $\alpha$  as a *reading-transition*.

In our intuitive description, we stated that the bottom-of-stack symbol  $\perp$  cannot be deleted nor can it be written in the middle of the stack. We achieve these properties formally via points number 1 and 2 respectively in Definition [2.2.15](#)

Recall that in a finite state automaton an  $\varepsilon$ -transition changes the state of the automaton without reading any input. In a pushdown automaton an  $\varepsilon$ -transition is one of the form  $((p, \varepsilon, y), (q, \omega))$ . This is interpreted as changing both the state of the automaton as well as editing the stack, without reading any input. Formally we denote a sequence of  $\varepsilon$ -transitions through  $\sim_\varepsilon^*$ .

**Example 2.2.22.** Let  $A = (Q, I, \Sigma, \chi, \perp, \delta, F)$  be the pushdown automaton determined by setting:

- $Q = \{q_A, q_B, q_a, q_b, q_f\}$ ;
- $I = \{q_A\}$ ;
- $\Sigma = \{a, b\}$ ;
- $\chi = \{a, b, \perp\}$ ;

- The transition relation  $\delta$  consists of the following transitions:

$$\begin{aligned} &((q_A, a, \perp), (q_A, a\perp)), ((q_A, a, a), (q_A, aa)), \\ &((q_A, \varepsilon, \perp), (q_B, \perp)), ((q_A, \varepsilon, a), (q_B, a)), \\ &((q_B, b, a), (q_B, \varepsilon)), ((q_B, \varepsilon, \perp), (q_a, \perp)), \\ &((q_a, a, \perp), (q_b, \perp)), ((q_b, b, \perp), (q_f, \perp)); \text{ and} \end{aligned}$$

- $F = \{q_f\}$ .

The above automaton is represented by the graph in Example [2.2.14](#) as we will explain below.

#### 2.2.2.4 Visualisation and Determinisim

We can visualise pushdown automata in a similar way to how we visualised finite state automata, i.e., through an edge-labelled directed graph, called a transition diagram or state diagram.

**Definition 2.2.23** (Transition Diagram/State Diagram). Let  $\mathcal{A} = (Q, I, \Sigma, \chi, \perp, \delta, F)$  be a pushdown automaton. We will define the *transition diagram (or state diagram)* to be a directed graph  $\Gamma(\mathcal{A})$  as follows:

- The vertex set is  $Q$ .
- The edges of  $\Gamma(\mathcal{A})$  are all triples  $(q, (x, y, \omega), q')$  for some  $q, q' \in Q$  such that  $((q, x, y), (q', \omega)) \in \delta$ . We shall denote the set of edges by  $E$ . We represent an edge  $(q, (x, y, \omega), q')$  by an arrow from  $q$  to  $q'$  with  $(x, y, \omega)$  as its label.

We represent transition diagrams by figures of edge-labelled directed graphs. The vertices of the graphs will be denoted by circles with the corresponding states written in them. Each vertex representing a state in  $I$  will be denoted by a diamond instead of the usual circle. Similarly each vertex representing a state in  $F$  will be denoted by a double circle instead of a single one. ♣

An example of a transition diagram of a pushdown automaton can be found in Example [2.2.14](#) where we give the transition diagram for the pushdown automaton given in Example [2.2.22](#).

Note that in Example [2.2.22](#), there are multiple transitions coming out of a single state with the same input letter. This corresponds to there being multiple arrows coming out of a vertex with the same input letter in the transition diagram. Unlike the case for finite state automata, this does not necessarily mean that the automaton is non-deterministic.

**Definition 2.2.24** (Deterministic and Non-Deterministic).

Let  $\mathcal{A} = (Q, I, \Sigma, \chi, \perp, \delta, F)$  be a pushdown automaton. We say  $\mathcal{A}$  is *deterministic* if both of the following conditions hold:

- for every  $p \in Q$  and  $y \in \chi \cup \{\perp\}$ , if there exists a pair  $(q, \omega)$  such that  $((p, \varepsilon, y), (q, \omega)) \in \delta$  then for all  $x \in \Sigma \cup \{\varepsilon\}$  there is no pair  $(q', \omega')$  such that  $((p, x, y), (q', \omega')) \in \delta$ ,
- if  $p \in Q, \sigma \in \Sigma$  and  $y \in \chi \cup \{\perp\}$ , then there is at most one pair  $(q, \omega)$  such that  $((p, \sigma, y), (q, \omega)) \in \delta$ .

Otherwise, we say the automaton is *non-deterministic*. ♣

Observe that if  $\mathcal{A}$  is a deterministic PDA, then there exists a PDA  $\mathcal{A}'$  that accepts the complementary language. In particular it is also deterministic. This is similar to how regular languages are closed under complementation, Proposition [2.2.13](#). However, there are languages accepted by (non-deterministic) pushdown automata whose complement is not context-free.

It is sometimes useful to be able to view more than the top most letter in the stack. One can modify the definition of a pushdown automaton to allow for this. We do this below.

**Definition 2.2.25** (Pushdown Automata with viewing window of size  $n$ ). A *pushdown automaton with viewing window of size  $n$*  is a 8-tuple  $\mathcal{A} = (Q, I, \Sigma, \chi, \perp, \delta, F, n)$  such that:

- $Q$  is a finite set, which is called the *set of states* of  $\mathcal{A}$ ;
- $I \subseteq Q$  is called the *set of initial states* of  $\mathcal{A}$ ;
- $\Sigma$  is a finite set, which is called the *tape (or input) alphabet* of  $\mathcal{A}$ ;
- $\chi$  is a finite set, which is called the *stack alphabet* of  $\mathcal{A}$ ;
- $\perp \notin \chi$  is the *bottom of stack symbol*;
- define  $A$  and  $B$  as follows

$$A := Q \times \Sigma \times (\chi^n \cup \{\perp\}) \cup \left( \bigcup_{j=1}^{n-1} \chi^j \perp \right)$$

and

$$B := Q \times (\chi^* \cup \chi^* \perp).$$

Then  $\delta \subseteq A \times B$  is a finite set called the *transition relation* and we refer to an element of  $\delta$  as a *transition*. Further let  $q, q' \in Q$  and  $x \in \Sigma \cup \{\varepsilon\}$ . If  $((q, x, \omega), (q', \omega')) \in \delta$  then one of the following conditions hold:

1.  $\omega \in \chi^n$  and  $\omega' \in \chi^*$ ,
  2.  $\omega = \perp$  and  $\omega' \in \chi^* \perp$ ,
  3.  $\omega \in \bigcup_{j=1}^{n-1} \chi^j \perp$  and  $\omega' \in \chi^* \perp$ ;
- $F \subseteq Q$  is called the *set of final (or accept) states*; and
  - $n \in \mathbb{N}$  where  $n \geq 1$  and is called the *size of the viewing window*.

The *stack* is a string  $S = s_n s_{n-1} \cdots s_0$  such that  $s_0 = \perp$  and  $s_i \in \chi$  for  $i > 0$ . We define the *top* (of the stack) to be the left end of the string and the *bottom* (of the stack) to be the right end. ♣

Note that points 1, 2 and 3 in the definition above achieves the same as points 1 and 2 in Definition [2.2.15](#). These points restrict elements of  $\delta$  such that  $\perp$  does not get inserted in the middle of the stack nor is it deleted.

Further note, the class of machines defined by Definition [2.2.25](#) is equivalent to the one defined by Definition [2.2.15](#), in that both classes accept the same class of languages. This is a standard exercise in concatenation of paths.

### 2.2.3 Indexed Languages and Nested Stack Automata

In this subsection, we shall introduce a container class of context-free languages, namely, indexed languages. These were introduced by Aho in [\[1\]](#). This family of languages is a subclass of context sensitive languages. Similar to other kinds of languages, there are various ways of defining indexed languages. In this section we shall use a variation on Aho's machine-theoretic formulation in [\[2\]](#), with restrictions due to Holt and Röver as in [\[28\]](#). This results in a restricted subclass compared to that of Aho. However it is this restricted class of languages and corresponding machines that we are interested in (as the restrictions correspond to being able to read a string in the generators of a group letter by letter, which is our ultimate goal). Therefore, when we refer to *indexed languages* we are actually referring to the subclass formed by the restrictions in [\[28\]](#). Similarly, when discussing the machine used in defining indexed languages, we are actually referring to the machine used to define the restricted subclass of indexed languages. We will follow Holt–Röver's example and give the machines the same name as Aho does. These machines are known as *nested stack automata*. In the following subsection, we give an intuitive description of how they work.

#### 2.2.3.1 Intuitive Description

We can think of a nested stack automaton as a pushdown automaton with a modified version of the stack. Firstly the top of this modified version of the stack



is decorated with a unique symbol  $\mathbf{t}$ . Further we have a pointer attached to the stack that can go up and down the stack. Unlike in a pushdown automaton where the top of the stack was the viewable part, in a nested stack automaton we view the letter on the stack that is being pointed to. Further, sometimes we are also able to view a letter below the one being pointed to.

A transition in a pushdown automaton was interpreted as reading a letter  $x \in \Sigma \cup \{\varepsilon\}$  from a state  $p \in Q$  with  $y$  at the top of the stack and moving to a state  $q \in Q$  while writing a string  $\omega$  to the stack after deleting  $y$ . In a nested stack automaton, a transition is interpreted similarly. We read a letter  $x \in \Sigma \cup \{\varepsilon\}$  from a state  $p \in Q$  with  $y$  being pointed to by the pointer, and we move to a state  $q \in Q$  as well as modify the stack. Here, the modifications can occur in four different ways depending on what the pointer was pointing to. These are known as *pushdown mode*, *stack reading mode*, *branch creation mode*, and *branch destruction mode*. Below we give a description of how these modes work. This description is followed by some diagrams for illustrative purposes.

**1.1 Pushdown mode:** Suppose the pointer was pointing to  $\mathbf{t}$ . In this case, the automaton can also view the letter  $y$  under  $\mathbf{t}$  in the stack. Then the automaton can simply act as a pushdown automaton, this is done by replacing  $\mathbf{t}y$  by a string  $\mathbf{t}\omega$  (i.e., deleting  $\mathbf{t}y$  and writing  $\mathbf{t}\omega$ ). At the end of writing the pointer will be pointing to  $\mathbf{t}$  again. Thereby, replacing the letter  $y$  by the string  $\omega$  in a very similar way to how pushdown automata work. Note that if  $y \neq \perp$  then no letter in  $\omega$  can be  $\perp$ . Conversely if  $y = \perp$  then the last letter of  $\omega$  is  $\perp$ . We may think of this in a similar way to a PDA with a window of size 2, where  $\mathbf{t}$  is at the top.

**1.2 Stack Reading mode:** Suppose the pointer was pointing to a letter  $y$  on the stack. The automaton can move the pointer up or down by at most one position (i.e., go up or down by one position or remain in its position).

- (a) If  $y \notin \{\mathbf{t}, \perp\}$  then the pointer can go up and down by at most one position.
- (b) If  $y = \mathbf{t}$  then it can go down by at most one position.
- (c) If  $y = \perp$  then it can go up by at most one position.

We note that the stack has not been edited but it is now possible to view different parts of the stack.

**1.3 Branch Creation mode:** Suppose the pointer was pointing to a letter  $y$  that is not  $\mathbf{t}$ . This means that the pointer is not at the top of the stack and thus cannot be in pushdown mode. However a nested stack automaton can still write on the stack. This is done through a process we call *branching* or

*nesting*. We visualise this by an off-shoot (which we refer to as a *branch*) from the previous part of the stack. We also represent this in the stack by using a unique symbol  $\$$ . We call  $y$  the *branching point*. In this branch, the automaton writes a string  $\mathbf{t}\omega\$$  from  $y$  thereby signaling that this off-shoot is to be treated like the top of the stack and the pointer is pointing to the newly inserted  $\mathbf{t}$ . (We note that  $\$$  was used here and thereby signaling that a branch has been created). This is done without editing the previous part of the stack. Note that if one visualised the stack as a finite sequence where the top of the stack is at the left-most symbol and the  $\perp$  is the right-most symbol. Then  $\mathbf{t}\omega\$$  will be inserted to the left of  $y$ . If a branch is created, then the part of the stack above the branching point cannot be viewed via the stack reading mode until the branch is destroyed. See the diagrams below for an illustration of this.

- 1.4 **Branch Destruction mode:** Suppose the pointer is pointing to  $\mathbf{t}$  with  $\$$  under it (so the pointer is pointing to  $\mathbf{t}$  in a branch). Then we say that the branch is *empty*. The automaton can destroy the branch by deleting  $\mathbf{t}\$$ . Then the pointer will return to the branching point, i.e., at the position below where  $\$$  was.

We see that if the automaton would only use pushdown mode then the same information could be encoded with a pushdown automaton. That is languages accepted by these machines include context-free languages. (Note that we will define what it means for these machines to accept languages later.) Further note that the part of the stack from which the first branch is created is referred to as the *trunk*. Below we provide some diagrams to illustrate what modifications to the stack would look like depending on the kind of mode which was used and the structure of the stack.

We shall start with pushdown mode. The two diagrams below illustrate what happens to the stack when the transition is a pushdown one. The first diagram is one that is similar to the standard way we think of pushdown automata. The second diagram shows that the behaviour is the same in pushdown mode when on a branch. We shall use these diagrams to illustrate the rest of the modes.

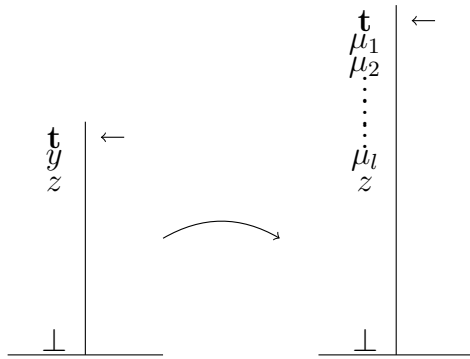


Figure 1: Pushdown mode on the trunk

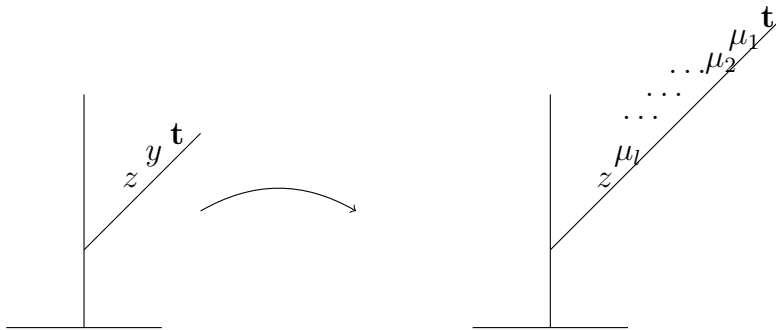


Figure 2: Pushdown mode on a branch

We note that in stack reading mode, there is a difference in what is viewable between a stack that is nested and one that is not. We shall start with a non-nested stack. Here the pointer can move at most one position up or down, with the exceptions of not being able to move past  $\mathbf{t}$  and  $\perp$ . Therefore the automaton can view the entirety of the stack, if there are transitions that would allow for that.

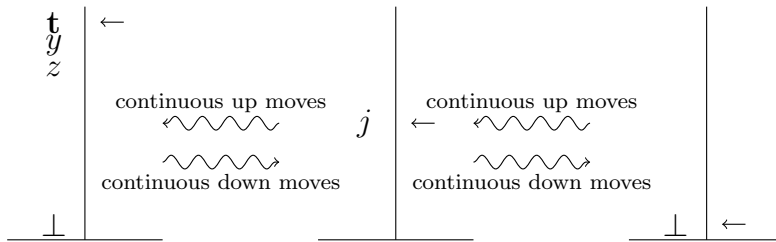


Figure 3: Stack reading mode on the trunk

For a nested stack, the pointer can move up or down with a slightly different behaviour at branching points. At a branching point, one position below moves the pointer back to the previous branch and one position above moves it into the new branch. That is to say portions above a branching point before nesting are no longer viewable after a branch has been created.

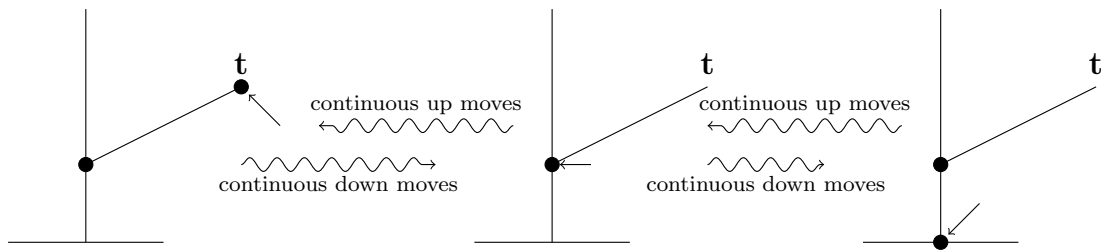


Figure 4: Stack reading mode with a branch

For branch creation mode, the position of the pointer is important as the stack can look different depending on where the branching occurs. This will in turn make parts of the stack possibly not viewable. We shall give some diagrams illustrating these below.

We start with a non-nested stack, with a pointer not at  $t$ . This introduces a branch in the stack.

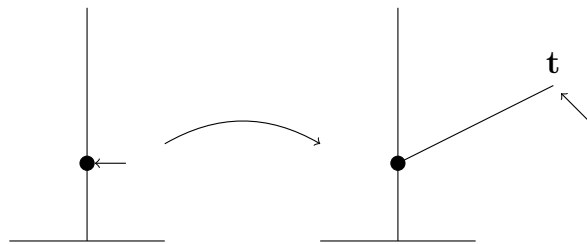


Figure 5: Branch creation mode from the trunk

For a nested stack, there are two options. Either the pointer is pointing to a letter within the last created branch or it is not. We illustrate how the branching occurs in certain cases with the diagrams below.

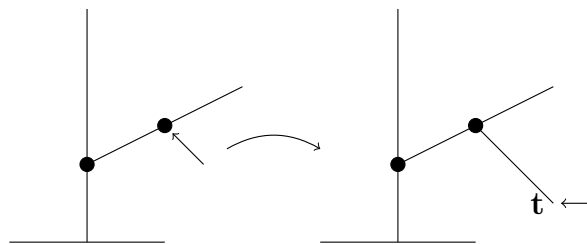


Figure 6: Branch creation mode from a branch

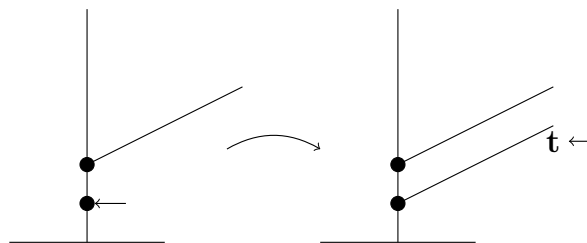


Figure 7: Branch creation mode from the trunk under a branch

For branch destruction mode we require the only letters on the branch to be  $\mathbf{t}$  and  $\$$ . During the destruction the string  $\mathbf{t}\$$  will be deleted (along with the branch) and the pointer will return to the branching point. We give some diagrams below to illustrate that.

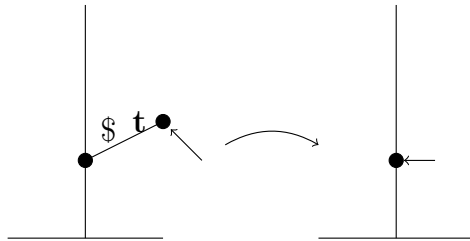


Figure 8: Branch destruction mode from a branch to the trunk

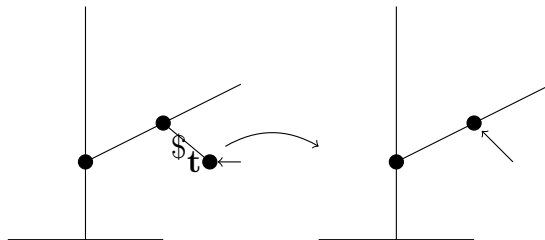


Figure 9: Branch destruction mode from a branch off a branch

We note that with branch creation mode above (see Figure 6 above) we created a stack with a branch nested into another branch. Below we illustrate how stack reading mode acts on that stack and thus what portions of it are viewable through the stack reading mode.

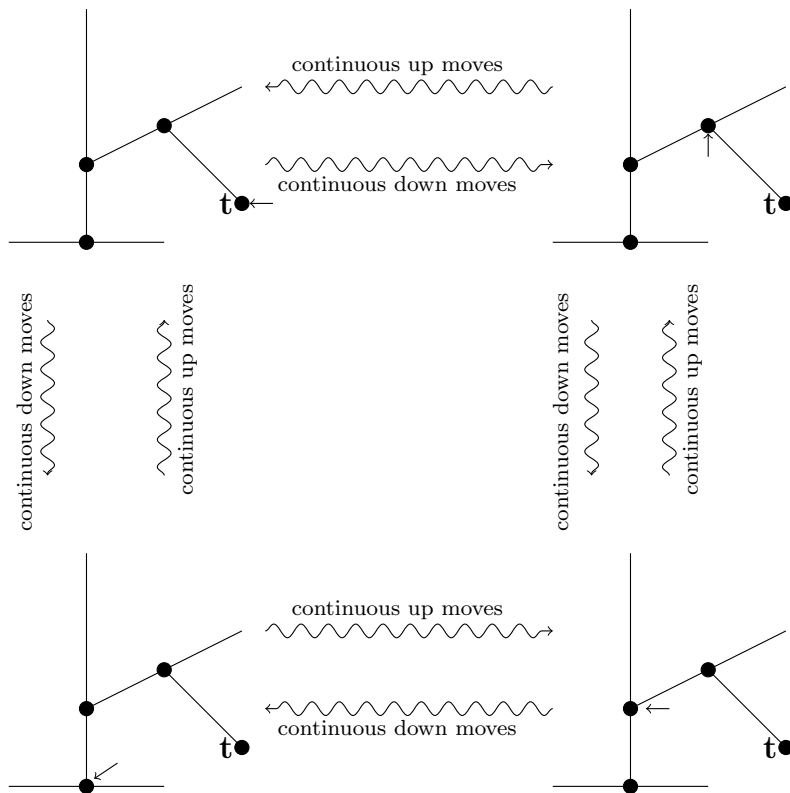


Figure 10: Stack reading mode with nested branching

We note that initially, the automaton has  $\mathbf{t} \perp$  on the stack, with the pointer at  $\mathbf{t}$ .

In a nested stack automaton, we interpret reading a string  $\omega = \sigma_1\sigma_2\cdots\sigma_n$  as traversing a path  $v_0 \xrightarrow{x_1} v_1 \xrightarrow{x_2} \cdots \xrightarrow{x_m} v_m$  in the graph of the underlying finite state automaton such that  $x_1x_2\cdots x_m =_{\Sigma^*} \omega$  where  $v_0$  is an initial state while editing the stack. We refer to  $v_i$  as being the *active state of the automaton* or the *state of the automaton* (or by similar language) having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read. The stack will be edited via the different modes we have discussed as per the transitions. We call the stack associated to the automaton having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read *the active stack*. The *active configuration* is a pair  $(q, \mathcal{S})$  where  $q$  and  $\mathcal{S}$  are the active state and the active stack respectively having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read.

A string  $\omega' \in \Sigma^*$  is said to be *accepted* by a nested stack automaton if we can get from an initial state to a final state via a sequence of transitions where the concatenation of consecutive input letters is  $\omega'$ . Finally, *the language accepted by the automaton* is the set of all accepted strings.

In the following subsection, we shall formalise the above discussion. Note that we have been paying close attention to the stack and how it gets modified, this is reflected in the transition relation  $\delta$ .

### 2.2.3.2 Formal Definitions

We define a nested stack automata below. The transition relation  $\delta$  is defined to be a union of four sets,  $\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3$ , and  $\mathcal{J}_4$ , representing the different modes discussed above. We allow for any choice of  $\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3$ , and  $\mathcal{J}_4$ . This yields more or less complicated machines.

**Definition 2.2.26** (Nested Stack Automaton). A *nested stack automaton* (or *NSA*)  $\mathcal{A}$  is a 9-tuple  $(Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  such that:

- $Q$  is a finite set, which is called the *set of states* of  $\mathcal{A}$ ;
- $\Sigma$  is a finite set, which is called the *tape (or input) alphabet* of  $\mathcal{A}$ ;
- $\chi$  is a finite set, which is called the *stack alphabet* of  $\mathcal{A}$ ;
- $I \subseteq Q$  is called the *set of initial states* of  $\mathcal{A}$ ;
- $F \subseteq Q$  is called the *set of final (or accept) states*;
- $\$ \notin \chi$  is the *bottom of embedded stack symbol*;
- $\mathbf{t} \notin \chi$  is the *top of (active) stack symbol*;
- $\perp \notin \chi$  is the *bottom of stack symbol*; and
- $\delta$  is the transition relation which we describe in greater detail below.

We shall define 4 sets of pairs,  $\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3$ , and  $\mathcal{J}_4$ , as follows.

2.1 Set  $A_1 := Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbf{t}(\chi \cup \{\perp\})$ ,  $B_1 := Q \times \mathbf{t}(\chi^* \cup \chi^* \perp)$  and  $\mathcal{J}_1 \subseteq A_1 \times B_1$ . Further let  $p, q \in Q$  and  $x \in \Sigma \cup \{\varepsilon\}$ . If  $((p, x, \mathbf{t}y), (q, \mathbf{t}\omega)) \in \mathcal{J}_1$  then one of the following conditions hold:

- (a)  $y \in \chi$  and  $\omega \in \chi^*$ , and
- (b)  $y = \perp$  and  $\omega \in \chi^* \perp$ .

2.2 (a) Set  $A_{2,1} := Q \times (\Sigma \cup \{\varepsilon\}) \times (\chi \cup \{\$\})$ ,  $B_{2,1} := Q \times \{-1, 0, 1\}$  and  $D_1 := A_{2,1} \times B_{2,1}$ .

(b) Set  $A_{2,2} := Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbf{t}\chi$ ,  $B_{2,2} := Q \times \{0, -1\}$  and  $D_2 := A_{2,2} \times B_{2,2}$ .

(c) Set  $A_{2,3} := Q \times (\Sigma \cup \{\varepsilon\}) \times \{\perp\}$ ,  $B_{2,3} = Q \times \{0, 1\}$  and  $D_3 := A_{2,3} \times B_{2,3}$ .

Set  $\mathcal{J}_2 \subseteq D_1 \cup D_2 \cup D_3$ .

2.3 Let  $\chi' := \chi \cup \{\$, \perp\}$ ,  $A_3 := Q \times (\Sigma \cup \{\varepsilon\}) \times \chi'$  and  $B_3 := Q \times \mathbf{t}\chi'\$$ .

Then  $\mathcal{J}_3 \subseteq A_3 \times B_3$ . Let  $p, q \in Q, x \in \Sigma \cup \{\varepsilon\}$  and  $a \in \chi'$ . If  $((p, x, a), (q, \xi)) \in \mathcal{J}_3$  then there exists  $\omega \in \chi^*$  such that  $\xi = \mathbf{t}\omega\$a$ .

2.4 Let  $A_4 := Q \times (\Sigma \cup \{\varepsilon\}) \times \{\mathbf{t}\$\}$  and  $B_4 := Q \times \{\varepsilon\}$ . Then  $\mathcal{J}_4 \subseteq A_4 \times B_4$ .

Then  $\delta = \mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}_3 \cup \mathcal{J}_4$  is a finite set called the *transition relation* and we refer to an element of  $\delta$  as a *transition*.

The stack  $S$  is formally a sequence  $(s_n, s_{n-1}, \dots, s_1, s_0)$  where  $s_i \in \chi \cup \{\$, \mathbf{t}, \perp\}$  such that  $s_0 = \perp, s_n = \mathbf{t}$  and there is no  $i \in \{1, 2, \dots, n-1\}$  such that  $s_i = \perp$ . ♣

We note that transitions coming from  $\mathcal{J}_3$  introduce  $\mathbf{t}\omega'\$$  substrings into the stack (for some strings  $\omega' \in \chi^*$ ), we call this a *nesting* or a *branching*. Conversely, transitions coming from  $\mathcal{J}_4$  delete  $\mathbf{t}\$$  substrings from the stack.

**Definition 2.2.27** (Non-nested and nested stacks). Let  $\mathcal{A} = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  be a nested stack automaton. Let  $S = (s_n, s_{n-1}, \dots, s_0)$  be a stack. We say  $S$  is *non-nested* if there does not exist an  $i \in \{0, 1, \dots, n\}$  such that  $s_i = \$$  and we say it is *nested* otherwise. ♣

If a stack  $S$  is non-nested we may also refer to it as a *trunk*. Note that for any stack of a nested stack automaton, there is a portion of the stack that is readable.

**Definition 2.2.28.** [Readable portion of the stack] Let  $\mathcal{A} = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  be a nested stack automaton. Let  $S = (s_n, s_{n-1}, \dots, s_0)$  be a stack. Let  $r(S)$  be the minimal natural number such that  $s_{r(S)} = \mathbf{t}$ . We refer to  $(s_{r(S)}, s_{r(S)-1}, \dots, s_0)$  as the *readable portion of the stack*. ♣

Note that in the above definition, if  $S$  was non-nested then  $r(S) = n$ . We shall define configurations below. This is the first step in defining acceptance of strings by nested stack automata.

**Definition 2.2.29** (Configuration). A *configuration* is a triple  $(q, S, \zeta)$  where  $q \in Q$ ,  $S = (s_n, s_{n-1}, \dots, s_0)$  is the stack and  $\zeta$  is the pointer. We realise the pointer via a pair  $\zeta \in \mathbb{N} \times (\chi \cup \{\mathbf{t}, \$, \perp\})$  such that  $\zeta = (k, s_k)$  where  $k \in \{0, 1, \dots, r(S)\}$ . ♣

We will now use  $\delta$  to define a set of relations on the set of configurations below.

For each  $x \in \Sigma \cup \{\varepsilon\}$ , we define a relation  $\sim_x$  on the set of configurations as follows. Let  $C_1 = (q, S, \zeta)$  and  $C_2 = (q', S', \zeta')$  be two configurations, where  $S = (s_n, s_{n-1}, \dots, s_0)$  and  $S' = (s'_m, s'_{m-1}, \dots, s_0)$ . Further the pointers are  $\zeta = (i, s_i)$  and  $\zeta' = (j, s'_j)$  such that  $0 \leq i \leq r(S)$  and  $0 \leq j \leq r(S')$  as in Definition 2.2.29. Then  $C_1 \sim_x C_2$  if and only if one of the following conditions hold.



3.1 (Pushdown Mode) Suppose the pointers  $\zeta$  and  $\zeta'$  have the forms  $(i, \mathbf{t})$  and  $(j, \mathbf{t})$  respectively. The stack  $S$  is

$$(s_n, \dots, s_{i+1}, s_i = t, s_{i-1}, s_{i-2}, \dots, s_0).$$

If we are able to obtain  $S'$  by inserting a string  $\omega = \mu_1\mu_2 \cdots \mu_l$  replacing the letter  $s_{i-1}$  as follows

$$S' = (s_n, \dots, s_{i+1}, s_i = t, \mu_1, \mu_2, \dots, \mu_l, s_{i-2}, \dots, s_0)$$

with  $j = i + |\omega| - 1$  (i.e.,  $s_i = \mathbf{t}$  in  $S'$  is in coordinate  $j$ ) and there exists a transition  $((q, x, \mathbf{t}s_{i-1}), (q', \mathbf{t}\omega)) \in \mathcal{J}_1 \subseteq \delta$ .

3.2 (Stack reading mode) If  $S_1 = S_2$  and

- (a)  $s_i \notin \{\mathbf{t}, \perp\}$  with  $j - i \in \{0, -1, 1\}$  and there exists a transition  $((q, x, s_i), (q', j - i)) \in D_1 \subseteq \mathcal{J}_2 \subseteq \delta$ ;
- (b)  $s_i = \mathbf{t}$  with  $j - i \in \{0, -1\}$  and there exists a transition  $((q, x, \mathbf{t}), (q', j - i)) \in D_2 \subseteq \mathcal{J}_2 \subseteq \delta$ ; and
- (c)  $\zeta_1 = (0, \perp)$  with  $j - i \in \{0, 1\}$  and there exists a transition  $((q, x, \perp), (q', j - i)) \in D_3 \subseteq \mathcal{J}_2 \subseteq \delta$ .

3.3 (Branch Creation mode) Suppose  $\zeta = (i, s_i)$  and  $\zeta' = (j, \mathbf{t})$  such that  $s_i \neq \mathbf{t}$ . The stack  $S$  is

$$(s_n, \dots, s_{i+1}, s_i, s_{i-1}, \dots, s_0).$$

If we are able to obtain the stack  $S'$  by inserting a substring  $\mathbf{t}\omega\mathbb{S}$  (where  $\omega = \mu_1\mu_2 \cdots \mu_l$ ) between  $s_{i+1}$  and  $s_i$  as follows

$$S' = (s_n, \dots, s_{i+1}, \mathbf{t}, \mu_1, \mu_2, \dots, \mu_l, \mathbb{S}, s_i, s_{i-1}, \dots, s_0)$$

with  $j = i + |\omega| + 2$  and there exists a transition  $((q, x, s_i), (q', \mathbf{t}\omega\mathbb{S})) \in \mathcal{J}_3 \subseteq \delta$ .

3.4 (Branch Destruction mode) Suppose  $\zeta = (i, \mathbf{t})$  and  $s_{i-1} = \mathbb{S}$ . Recall that the stack  $S'$  is

$$(s_n, \dots, s_i = \mathbf{t}, s_{i-1} = \mathbb{S}, s_{i-2}, \dots, s_0).$$

If we are able to obtain  $S'$  by removing  $s_i$  and  $s_{i-1}$  from  $S$  as follows

$$S' = (s_n, \dots, s_{i+2}, s_{i+1}, s_{i-2}, s_{i-3}, \dots, s_0)$$

with  $j = i - 2$  and there exists a transition  $((q, x, \mathbf{t}\mathbb{S}), (q', \varepsilon)) \in \mathcal{J}_4 \subseteq \delta$ .

Before continue, we establish some language regarding  $\sim_x$ . Note in the above definition of when configurations are related by  $\sim_x$  (for  $x \in \Sigma \cup \{\varepsilon\}$ ), we had the following general structure. Let  $C_1$  and  $C_2$  be configurations. Then  $C_1 \sim_x C_2$  if and only if two things held:

(A) some conditions on the configurations that corresponded to each mode in the automaton, and

(B) there existing a transition  $\alpha \in \delta$  compatible with (A).

We say  $C_1 \sim_x C_2$  by  $\alpha$ , and/or *the automaton uses the transition  $\alpha$  (from  $C_1$  to  $C_2$ )*, or by similar language. Let  $\sim_\varepsilon^*$  be the reflexive transitive closure of  $\sim_\varepsilon$ . To continue our discussion of configurations, we will define *initial* and *accepting configurations*.

**Definition 2.2.30** (Initial Configuration). Let  $C = (q, S, \zeta)$  be a configuration. We say  $C$  is an *initial configuration* if  $q \in I, S = (\mathbf{t}, \perp)$  and  $\zeta = (1, \mathbf{t})$ . ♣

**Definition 2.2.31** (Accepting Configuration). Let  $C = (q, S, \zeta)$  be a configuration. We say  $C$  is an *accepting configuration* if  $q \in F$ . ♣

We are now ready to define acceptance.

**Definition 2.2.32** (Acceptance and Language accepted by a NSA). Let  $\mathcal{A} = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  be a nested stack automaton. We say  $\omega$  is *accepted by  $\mathcal{A}$*  if there exist configurations  $C_0, C_1, \dots, C_{2n+1}$  such that

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* C_3 \sim_{\sigma_2} C_4 \sim_\varepsilon^* \dots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1},$$

where  $C_0$  is an initial configuration and  $C_{2n+1}$  is an accepting configuration.

Let  $\mathcal{L}(\mathcal{A})$  be the set of all strings in  $\Sigma^*$  that are accepted by  $\mathcal{A}$ . We say the *language  $L$  is accepted by  $\mathcal{A}$*  if  $\mathcal{L}(\mathcal{A}) = L$ . ♣

If there exists a sequence of configurations  $C_0, C_1, \dots, C_{2n+1} \in Q$  such that

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* \dots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1}, \quad (\hat{4})$$

where  $\sigma_1, \sigma_2, \dots, \sigma_n \in \Sigma$  then we say  $\omega = \sigma_1 \sigma_2 \dots \sigma_n$  can be read from  $C_0$  through the chain  $(\hat{4})$ , or by similar language. The chain of relations (see e.g.  $(\hat{4})$ ) used to read a string is called a *run* of the automaton, we sometimes refer to it as a *run* and sometimes we simply just say *chain* or *chain of relations*. Note that for a string to be accepted, the automaton must be able to read the string from an initial configuration and the last configuration in a chain of relations (such as  $(\hat{4})$ ) is an accept configuration. If a string cannot be read from an initial configuration then it can never be accepted. If a string cannot be read in a particular run (or any runs), we may say the string is *rejected*. Usually if the term *rejected* is used, it simply means that there is some underlying reason as to whether the string cannot be accepted that is fundamentally opposed to how the machine operates. One of the reasons can be that the string simply cannot be read from an initial

configuration (in a particular run). If there are other reasons, they will be made clear in the contexts in which they arise. It will sometimes be useful to designate a state  $q_r$  as *the reject state*. This state will not be a final state of the automaton, and the automaton will have transitions to move to  $q_r$  when the input string is to be rejected. A reject state  $q_r$  will be a *sink state*. A sink is a state from which no other state can be reached. We say a configuration  $C$  is a *sink*, if no other configuration can be reached from  $C$ .

We will now make a notational remark. We will make use of the term  $\varepsilon$ -*transition* (we will revisit this term again in the following section). A transition  $((q, x, w), (q', a))$  is said to be an  $\varepsilon$ -transition if  $x = \varepsilon$ . Suppose in a chain such as the one above in (4), there exists a configuration  $C$  appearing as  $C \sim_\varepsilon^* C$  in the chain. Further suppose there do not exist configurations  $C'_1, C'_2, \dots, C'_r$  and there does not exist a sequence of  $\varepsilon$ -transitions yielding

$$C \sim_\varepsilon C'_1 \sim_\varepsilon C'_2 \sim_\varepsilon \dots \sim_\varepsilon C'_r \sim_\varepsilon C.$$

Then  $C \sim_\varepsilon^* C$  appears in the chain due to the reflexive property of  $\sim_\varepsilon^*$ . In such a situation, we will remove the appearance of  $\sim_\varepsilon^* C$  from the chain to shorten it.

We are now ready to define an indexed language.

**Definition 2.2.33** (Indexed languages). A language  $L$  is said to be *indexed* if there exists a nested stack automaton  $\mathcal{A}$  such that  $L = \mathcal{L}(\mathcal{A})$ . ♣

Note we that shall use “NSA” to mean *a nested stack automaton* or *the class of nested stack automata* and it will be clear from context which is meant when it arises.

We shall link the formal definition to our intuitive description of nested stack automata in the following subsection.

### 2.2.3.3 Link between intuition and formality for NSA

Like PDA, the main complication in understanding how NSAs work comes from understanding how the automaton can modify the stack.

Let  $\mathcal{A} = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  be a nested stack automaton. Like previous types of automata,  $\mathcal{A}$  has a set of states  $Q$ , an input alphabet  $\Sigma$ , a set of initial states  $I$  and a set of final states  $F$ . The stack alphabet is  $\chi$ . Further we have two unique symbols  $\mathbf{t}$  and  $\$$  signaling the top of the active stack and branching respectively.

Let  $\alpha \in \delta$  be a transition. Recall from Definition [2.2.26](#) that  $\alpha = (X, Y)$  where  $X$  is a triple where the first two components represent the state  $q$  from which a letter  $x \in \Sigma \cup \{\varepsilon\}$  is read. This corresponds to the first two factors of each  $A_i$  in Definition [2.2.26](#). Unlike PDA where the automaton can only view the top of

the stack, in a nested stack automaton the type of transitions depend on what is pointed to. This is why we have various sets being the third factor of the  $A_i$ s in Definition [2.2.26](#). This influences the type of modification the automaton will do to the stack. We discuss these below. When  $x = \varepsilon$ , we call  $\alpha$  an  $\varepsilon$ -transition. Conversely when  $x \neq \varepsilon$ , we refer to  $\alpha$  as a *reading-transition*.

**4.1 Pushdown mode:** Suppose  $\alpha \in \mathcal{J}_1$ . Then the automaton can view the top of the readable portion of the stack  $\mathbf{t}y$  and will move to a state  $q'$  while writing a string  $\mathbf{t}\omega$  thus replacing  $y$  by  $\omega = \mu_1\mu_2 \cdots \mu_l$ . This is the pushdown mode we discussed in [1.1](#) in Intuitive Description [2.2.3.1](#). At the end of the processing the pointer is still at the top of the stack, this is reflected in the configurations. Suppose the configuration before reading  $x \in \Sigma \cup \{\varepsilon\}$  is

$$(q, (s_n, \dots, s_i, s_{i-1}, \dots, \perp), (i, \mathbf{t})).$$

Then after reading  $x$  the configuration will be

$$(q', (s_n, \dots, s_{i+1}\mathbf{t}, \mu_l, \mu_{l-1}, \dots, \mu_1.s_{i-2}, \dots, \perp), (i+l-1, \mathbf{t})).$$

**4.2 Stack Reading mode:** Suppose  $\alpha \in \mathcal{J}_2$ . This is the stack reading mode we discussed in Intuitive Description [2.2.3.1](#)

(a) Suppose  $\alpha \in D_1$ . Then  $\alpha \in A_{2,1} \times B_{2,1}$ . We interpret this as the pointer is pointing to the a stack letter that is neither  $\mathbf{t}$  nor  $\perp$  and is allowed to move up or down by one position (or remain where it is). This is reflected in the configurations as follows. Suppose the configuration before reading  $x$  is

$$(q, (\mathbf{t}, s_{n-1}, \dots, s_i, \dots, \perp), (i, s_i)).$$

Then after reading  $x$ , the configuration becomes

$$(q', (\mathbf{t}, s_{n-1}, \dots, s_i, \dots, \perp), (j, s_j))$$

where  $|j - i| \in \{0, 1\}$ . Note that if  $|j - i| = 0$  then  $i = j$  this is interpreted as the pointer remaining in its position. If  $j - i = 1$  then this is interpreted as the pointer moving up the stack by one position. Conversely if  $j - i = -1$  we interpret this as the pointer moving down the stack by one position.

(b) Suppose  $\alpha \in D_2$ . Then  $\alpha \in A_{2,2} \times B_{2,2}$ . We interpret this as the pointer pointing to  $\mathbf{t}$  and is allowed to move down by one position or remain

where it is. This is reflected in the configurations as follows. Suppose the configuration before reading  $x$  is

$$(q, (\mathbf{t}, s_{n-1}, \dots, s_i, \dots, \perp), (i, s_i))$$

where  $s_i = \mathbf{t}$ . Then after reading  $x$ , the configuration becomes

$$(q', (\mathbf{t}, s_{n-1}, \dots, s_i, \dots, \perp), (j, s_j))$$

where  $j - i \in \{0, -1\}$ . If  $j - i = 0$  then this is interpreted as the pointer remaining in its position. If  $j - i = -1$  then we interpret it as the pointer moving down one position.

- (c) Suppose  $\alpha \in D_3$ . Then  $\alpha \in A_{2,3} \times B_{2,3}$ . We interpret this as the pointer pointing to  $\perp$  and is allowed to move up by one position or remain in its position. This is reflected in the configurations as follows. Suppose the configuration before reading  $x$  is

$$(q, (\mathbf{t}, s_{n-1}, \dots, \perp), (0, \perp)).$$

Then after reading  $x$ , the configuration becomes

$$(q', (\mathbf{t}, s_{n-1}, \dots, \perp), (i, s_i))$$

where  $i \in \{0, 1\}$ . If  $i = 0$  then we interpret this as the pointer remained in its position and is still pointing to  $\perp$ . If  $i = 1$ , then we interpret this as the pointer has moved up by one position and is now pointing to  $s_1$ .

**4.3 Branch Creation mode:** Suppose  $\alpha \in \mathcal{J}_3$ . This is the branch creation mode we discussed in Intuitive Description [2.2.3.1](#). Then  $\alpha \in A_3 \times B_3$ . We interpret this as the pointer pointing to a letter  $y \neq \mathbf{t}$  and creating a branch from  $y$  with a string  $\mathbf{t}\omega\$$  on it, where  $\omega = \mu_1\mu_2 \cdots \mu_l$ . After the branch is created then the pointer is pointing to the top of the branch. This reflected in the configurations as follows. Suppose the configuration before reading  $x$  is

$$(q, (\mathbf{t}, s_{n-1}, \dots, s_{i+1}, s_i = y, \dots, \perp), (i, y)).$$

Then the configuration after reading  $x$  is

$$(q', (\mathbf{t}, s_{n-1}, \dots, s_{i+1}, \mathbf{t}, \mu_1, \mu_2, \dots, \mu_l, \$, s_i = y, \dots, \perp), (i + l + 2, \mathbf{t})).$$

**4.4 Branch Destruction mode:** Suppose  $\alpha \in \mathcal{J}_4$ . This is the branch destruction mode we discussed in Intuitive Description [2.2.3.1](#). Then  $\alpha \in A_4 \times B_4$ . We interpret this as the pointer pointing  $\mathbf{t}$  with  $\$$  under it (i.e., the branch

is empty), and deleting it. This is reflected in the configurations as follows. Suppose the configuration before reading  $x$  is

$$(q, (\mathbf{t}, \dots, s_{i+1}, s_i = \mathbf{t}, \$, s_{i-2}, \dots, \perp), (i, \mathbf{t})).$$

Then the configuration after reading  $x$  is

$$(q', (\mathbf{t}, \dots, s_{i+1}, s_{i-2}, \dots, \perp), (i-2, s_{i-2})).$$

In the following section, we shall discuss how to visualise nested stack automata and introduce some formal language that will be helpful later.

#### 2.2.3.4 Visualisation and Determinism

Similar to PDA, we can visualise nested stack automata via transition diagrams. We note that the difference between Definition 2.2.8 and Definition 2.2.23 is the inclusion of more information on the edge labels representing modifying the stack for pushdown automata. We do this in the following definition as well. However the difference is that we are allowed more flexibility since there are different ways that a nested stack automaton may modify the stack as per  $\delta$  in Definition 2.2.26.

Recall the sets  $A_1; A_{2,1}; A_{2,2}; A_{2,3}; A_3$  and  $A_4$ , and the sets  $B_1; B_{2,1}; B_{2,2}; B_{2,3}; B_3$  and  $B_4$  from Definition 2.2.26. Let  $A_x \in \{A_1, A_{2,1}, A_{2,2}, A_{2,3}, A_3, A_4\}$ . Then we define  $A_x^p$  to be the image of the projection of  $A_x$  onto its third factor. Similarly, let  $B_x \in \{B_1, B_{2,1}, B_{2,2}, B_{2,3}, B_3, B_4\}$ . Then we define  $B_x^p$  to be the image of the projection of  $B_x$  onto its second factor. Further set  $A^p := \{A_1^p, A_{2,1}^p, A_{2,2}^p, A_{2,3}^p, A_3^p, A_4^p\}$  and  $B^p := \{B_1^p, B_{2,1}^p, B_{2,2}^p, B_{2,3}^p, B_3^p, B_4^p\}$ .

**Definition 2.2.34** (Transition Diagram for Nested Stack Automata). Let  $\mathcal{A} = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  be a nested stack automaton. We will define the *transition diagram* (or *state diagram*) to be a directed graph  $\Gamma(\mathcal{A})$  as follows:

- the vertex set is  $Q$ .
- The edges of  $\Gamma(\mathcal{A})$  are all triples  $(q, (x, p, a), q')$  for some  $q, q' \in Q$  with  $p \in \bigcup_{A' \in A^p} A'$  and  $a \in \bigcup_{B' \in B^p} B'$  such that  $((q, x, p), (q', a)) \in \delta$ . We shall denote the set of edges by  $E$ . We represent as an edge  $(q, (x, p, a), q')$  by an arrow from  $q$  to  $q'$  with a label which we will describe below.

5.1 If  $((q, x, p), (q', a)) \in \mathcal{J}_1$  then we write the label as  $(x, p, a)$ .

5.2 If  $((q, x, p), (q', a)) \in \mathcal{J}_2$  then  $a \in \{-1, 0, 1\}$ .

(a) If  $a = -1$  then we write the label as  $(x, p, \text{Down})$ .

(b) If  $a = 0$  then we write the label as  $(x, p, -)$ .

(c) If  $a = 1$  then we write the label as  $(x, p, \text{Up})$ .

5.3 If  $((q, x, p), (q', a)) \in \mathcal{J}_3$  then  $a = \mathbf{t}\omega\$p$  for some  $\omega \in \chi^*$ . We write the label as  $(x, p, \text{Add } \omega)$ .

5.4 If  $((q, x, p), (q', a)) \in \mathcal{J}_4$  then we write the label as  $(x, p, \text{Del})$ .

We represent transition diagrams by figures of edge-labelled directed graphs. The vertices of the graphs will be denoted by circles with the corresponding states written in them. Each vertex representing a state in  $I$  will be denoted by a diamond instead of the usual circle. Similarly each vertex representing a state in  $F$  will be denoted by a double circle instead of a single one. ♣

As for previous machine types, we shall define below what it means for a nested stack automaton to be deterministic. In the following definition we make use of the  $A^p$  defined at the beginning of this subsection.

**Definition 2.2.35** (Deterministic and Non-Deterministic).

Let  $\mathcal{A} = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  be a nested stack automaton. We say  $\mathcal{A}$  is *deterministic* if both of the following conditions hold:

- for every  $q \in Q$  and  $p \in \bigcup_{A' \in A^p} A'$ , if there exists a pair  $(q', \omega')$  such that  $((q, \varepsilon, p), (q', \omega')) \in \delta$ , then for all  $\sigma \in \Sigma$  there is no pair  $(q'', \omega'')$  such that  $((q, \sigma, p), (q'', \omega'')) \in \delta$ ,
- if  $q \in Q, \sigma \in \Sigma$  and  $p \in \bigcup_{A' \in A^p} A'$ , then there exists at most one pair  $(q', \omega')$  such that  $((q, \sigma, p), (q', \omega')) \in \delta$ .

Otherwise, we say the automaton is *non-deterministic*. ♣

Further, we present below some definitions and ideas from [28] as we will use them in later chapters.

We will make use of a convention stated in [28] (on page 5 of [28]) that for every indexed language  $L$ , there exists a nested stack automaton  $\mathcal{A}$  with a specific property such that  $\mathcal{L}(\mathcal{A}) = L$ . This property is that the set of states  $Q$  of  $\mathcal{A}$  can be written as a disjoint union of two sets  $Q_\varepsilon$  and  $Q_\Sigma$  such that

- if a transition is from a state  $p \in Q_\varepsilon$  then it is an  $\varepsilon$ -transition, and
- if a transition is from a state  $q \in Q_\Sigma$  then there are no  $\varepsilon$ -transitions from  $q$ .

We refer to states in  $Q_\varepsilon$  and  $Q_\Sigma$  as  $\varepsilon$ - and *reading*-states respectively. Further let  $C = (q, S, \zeta)$  be a configuration of  $\mathcal{A}$ . We say  $C$  is a *reading configuration* if  $q \in Q_\Sigma$ . Conversely, we say  $C$  is an  $\varepsilon$  configuration if  $q \in Q_\varepsilon$ . Further, we will also make use of the following assumption (as in [28]):

- every sequence of consecutive  $\varepsilon$ -transitions must terminate after finitely many transitions.

For the remainder of this section we shall use the above convention (and assumption).

In later chapters, it will be useful to have language to describe a concept of determinism that occurs in a very specific way. Below we use definitions from [28] to be able to define that. First we need to think of determinism as being a property of states (and configurations) as in the following definition.

**Definition 2.2.36** (Determinism of states and configurations). A state  $q \in Q$  is (*state*) *deterministic* if for every configuration  $(q, S, \zeta)$  and every  $x \in \Sigma \cup \{\varepsilon\}$  there exists at most one configuration  $C'$  such that  $(q, S, \zeta) \sim_x C'$ .

A configuration  $(q', S', \zeta')$  is said to be *deterministic* if  $q$  is a deterministic state. ♣

We note the above Definition 2.2.36 is compatible with Definition 2.2.35 due to our convention above. We are now ready to make define what it means for an automaton to be *deterministic from a certain point onwards*.

**Definition 2.2.37** (Forwardly deterministic). Let  $q \in Q$ . We say  $q$  is *forwardly deterministic* if for every configuration  $C_0 = (q, S, \zeta)$ , every string  $w = \sigma_1 \cdots \sigma_n \in \Sigma^*$  and every collection of configurations  $C_1, C_2, \dots, C_{2n+1}$  such that

$$C_0 \sim_\varepsilon^* C_1 \sim_\sigma \sim_\varepsilon^* \cdots \sim_\varepsilon^* C_{2n+1},$$

all the states in  $C_i$  (for  $i = 0, 1, \dots, 2n + 1$ ) are deterministic. ♣

Due to our convention, it is possible that an initial state  $q_0$  is not a reading state. Thus we have the following definition.

**Definition 2.2.38** (Start Configuration). Let  $C$  be a reading configuration. Assume that an initial state  $q_0$  is not a reading state. We say  $C$  is a *start configuration* if  $(q_0, (\mathbf{t}, \perp), (1, \mathbf{t})) \sim_\varepsilon^* C$ . ♣

We are now ready to make the definition that we will use in later chapters.

**Definition 2.2.39.** A nested stack automaton  $\mathcal{A}$  is said to be *deterministic upon input* if every start configuration is forwardly deterministic. ♣

## 2.2.4 ETOL Languages and Check-stack Pushdown Automata

In this subsection we shall introduce a class of languages that is a subclass of indexed languages that contains context-free languages. These languages are known



as ETOL languages and have their origins stemming from L-systems [32, 33], [30]. Usually ETOL languages are defined by their grammar as in [4]. In [8], Elder and Bishop prove a theorem of van Leeuwen [42] stating that the class of languages accepted by the machine we define below is equivalent to the one defined by grammars in [4]. Therefore we shall focus on the machine theoretic formulation of ETOL languages. The machine we discuss below is known as *check-stack pushdown automaton*.

#### 2.2.4.1 Intuitive Description

We can think of a check-stack pushdown automaton as a pushdown automaton with a modified version of the stack. We think of a check-stack pushdown automaton as having two stacks, the first stack is called the *check-stack* and the second is called the *pushdown stack*. The check-stack has  $\Delta$  as an alphabet, while the pushdown stack has  $\chi$  as its alphabet. Both stacks have  $\perp$  as the symbol marking the bottom of stack. Further, there is a pointer with two heads that moves up and down the stacks. One head of the pointer points to the letter  $y \in \chi$  at the top of the pushdown stack while the other head points to the letter  $z \in \Delta$  in the check-stack in the same position as  $y$ . Further we require that the length of the check-stack is greater than or equal to the length of the pushdown stack at any point in the running of the automaton. (We shall refer to the later as the *stack length condition* in our discussion.) We also associate a regular language  $\mathcal{R} \subseteq \Delta^*$  to every check-stack pushdown automaton.

We think of processing in a check-stack pushdown machine to occur in two stages. The first stage happens initially before reading the input occurs. At this stage, a string  $\mu = \tau_1\tau_2 \cdots \tau_k \in \mathcal{R}$  is loaded onto the check-stack in reverse. That is to say the first letter loaded is  $\tau_k$ , then  $\tau_{k-1}$  and so on. The last letter to be loaded onto the top of the check-stack is  $\tau_1$ . Note that reading the contents of the check-stack top-to-bottom we will read  $\tau_1\tau_2 \cdots \tau_k\perp$ . The check-stack will not be edited from this point onward. Further note that at this stage the pushdown stack is empty and the pointer is pointing to  $\perp$  on both stacks.

The machine then enters its second stage of processing. It is in this stage that the transitions of the automaton are used. Here the automaton acts in a very similar way to a PDA, in that it reads a letter  $x \in \Sigma \cup \{\varepsilon\}$  from a state  $p \in Q$  with  $y$  at the top of the pushdown stack and  $z$  being the corresponding letter in the check-stack. Then the automaton deletes  $y$  and writes a string  $\omega$  to the top of the pushdown stack while moving to a state  $q \in Q$ . At the end of the transition, one head of the pointer will be pointing to the top of the pushdown stack, i.e., the initial letter  $a \in \chi$  of  $\omega$ . The other head of the pointer will be pointing to the letter  $b \in \Delta$  in the check-stack in the corresponding position to  $a$ . Further, similar to pushdown automata and nested stack automata,  $\perp$  cannot be deleted nor written

in the middle of the stack.

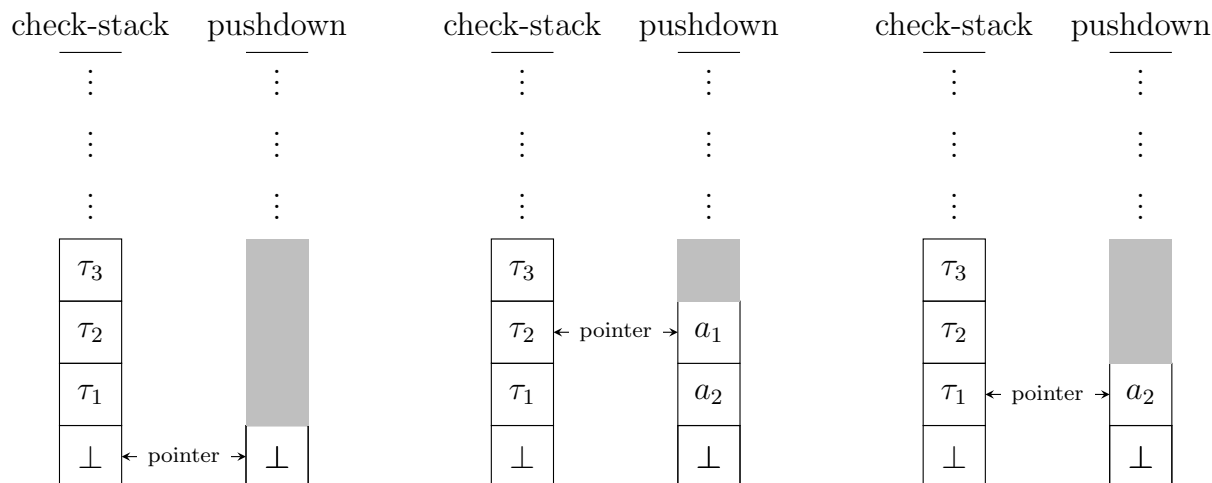
We note that while the transitions of a check-stack pushdown automaton are as described above, they also need to be compatible with the stack length condition. That is to say a transition  $\alpha$  is *valid* if the stacks before and after applying  $\alpha$  are such that the stack length condition holds. We say a transition is *invalid* otherwise.

In a check-stack pushdown automaton, we interpret reading a string  $\omega = \sigma_1\sigma_2\cdots\sigma_n$  as traversing a path  $v_0 \xrightarrow{x_1} v_1 \xrightarrow{x_2} \cdots \xrightarrow{x_m} v_m$  in the graph of the underlying finite state automaton such that  $x_1x_2\cdots x_m =_{\Sigma^*} \omega$  where  $v_0$  is an initial state while editing the stack. We refer to  $v_i$  as being the *active state of the automaton* or the *state of the automaton* (or by similar language) having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read. The stack will be edited as we have discussed as per the transitions. We call the stack associated to the automaton having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read *the active stack*. The *active configuration* is a pair  $(q, \mathcal{S})$  where  $q$  and  $\mathcal{S}$  are the active state and the active stack respectively having read  $x_1x_2\cdots x_i$  with  $x_{i+1}x_{i+2}\cdots x_m$  yet to be read.

A string  $\omega' \in \Sigma^*$  is said to be *accepted* by a check-stack pushdown automaton if we can get from an initial state to a final state via a sequence of valid transitions where the concatenation of the consecutive input letters is  $\omega'$ . We note that at the beginning of the second stage of processing, the pushdown stack is empty. Finally, *the language accepted by the automaton* is the set of all accepted strings.

We note that if a run of the automaton reading a string  $\omega_1$  does not yield a sequence of valid transitions, then that run is said to be an *invalid run* and the string  $\omega_1$  is immediately rejected *in that run*. This does not however mean that  $\omega_1$  is not accepted as it is possible that there exists a string  $\mu_1$  in the associated regular language such that with  $\mu_1$  on the check stack, reading  $\omega_1$  will yield a sequence of valid transitions.

Below we provide a diagram illustrating how the stack and the pointer work.



Note that in the first diagram, the pushdown stack is empty and thus one of the heads of the pointer is pointing to  $\perp$ . We see that the  $\perp$  is the corresponding letter on the check-stack and thus the other head of the pointer is also pointing at  $\perp$ .

In the second diagram, a string  $a_1a_2$  has been pushed onto the stack. Thus at the end of that transition, the pointer moved up and one head of the pointer is pointing to  $a_1$ . The other head is pointing to corresponding symbol  $\tau_2$  in the check-stack.

In the third diagram,  $a_1$  has been deleted from the pushdown stack and thus the pointer moved and one head is now pointing to  $a_2$ . The other head is pointing to the corresponding symbol on the check-stack  $\tau_1$ .

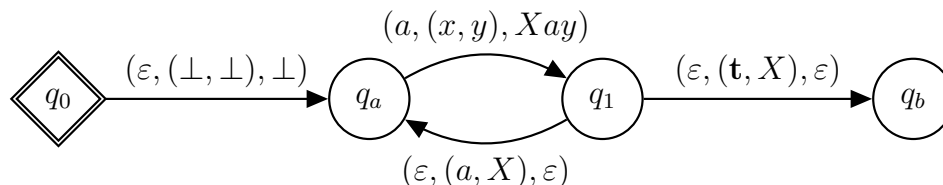
We provide an example below to explain the concepts we discussed above. Here we shall use an underlying edge-labelled directed graph structure similar to how we interpreted pushdown automata for the second stage of processing. However, we add an extra decoration to the edge labels, where we will write  $(x, (z, y), \omega)$  to denote reading  $x$  from the input alphabet, with one head of the pointer pointing to  $y$  on the pushdown stack with  $z$  being the corresponding letter on the check-stack and writing a string  $\omega$  on the pushdown stack after deleting  $y$ . Further, similar to a pushdown automaton, we shall draw each initial state with a diamond instead of the usual circle. Further we will draw each final state with a double circle instead of a single one. For the first stage, we shall simply state the regular language associated with the check-stack pushdown automaton.

**Example 2.2.40.** We shall describe an automaton that accepts  $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ .

For the first stage of processing, the regular language associated with the check-stack pushdown automaton is  $\mathbf{t}\{a\}^*$ .

For the second stage of processing, we give the edge-labelled directed graph below. However as the graph is large, we shall break it up into sub-stages. The first sub-stage is reading and writing the correct number of  $as$ . The second sub-stage deletes an  $a$  letter for every  $b$  letter read. Finally, the third sub-stage reads the same amount of  $c$  letters as  $a$  letters read in the first stage.

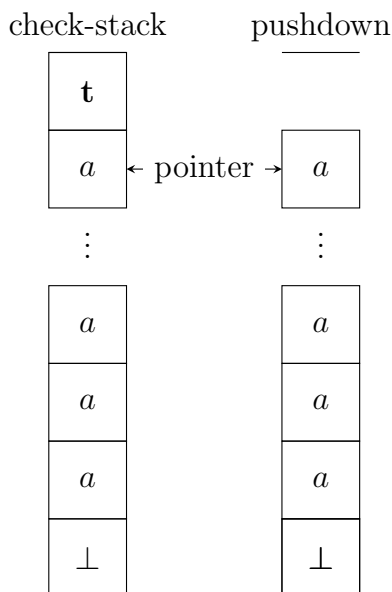
Below we give the graph for the first sub-stage.



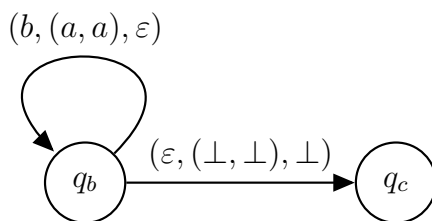
Recall that when trying to understand the language accepted by a check-stack pushdown automaton, we are only interested in sequences of valid transitions. We note that if check-stack has  $\mathbf{t}\perp$  on it, all transitions from  $q_a$  onward will not be valid as the pushdown stack will be longer than the check-stack. Further, the

regular language  $\mathbf{t}\{a\}^*$  has infinitely many strings and for any  $n \in \mathbb{N}$  there exists a string in the regular language of length  $n + 1$ . Therefore we will assume that the check-stack is long enough to allow for all transitions that we will require for the remainder of this argument.

There is a string of the form  $Xa^m\perp$  on the pushdown stack for some  $m \in \mathbb{N}$  at  $q_1$ . The string on the check-stack is  $\mathbf{t}a^m\perp$ . The states  $q_a$  and  $q_1$  serve to check for when the number of occurrences of  $a$  in the input is the same as the number of  $a$ s on the check-stack. To exit the circuit between  $q_a$  and  $q_1$ , the edge must  $q_1 \xrightarrow{(\varepsilon, (X, t), \varepsilon)} q_b$ . This would ensure that top of pushdown stack and the top of the check-stack are at the same height. Then using the edge would delete  $X$  at the top of the pushdown stack, thus ensuring the pointer is at the top most  $a$  of the pushdown stack. Therefore at the end of this stage the stack looks like the diagram below.

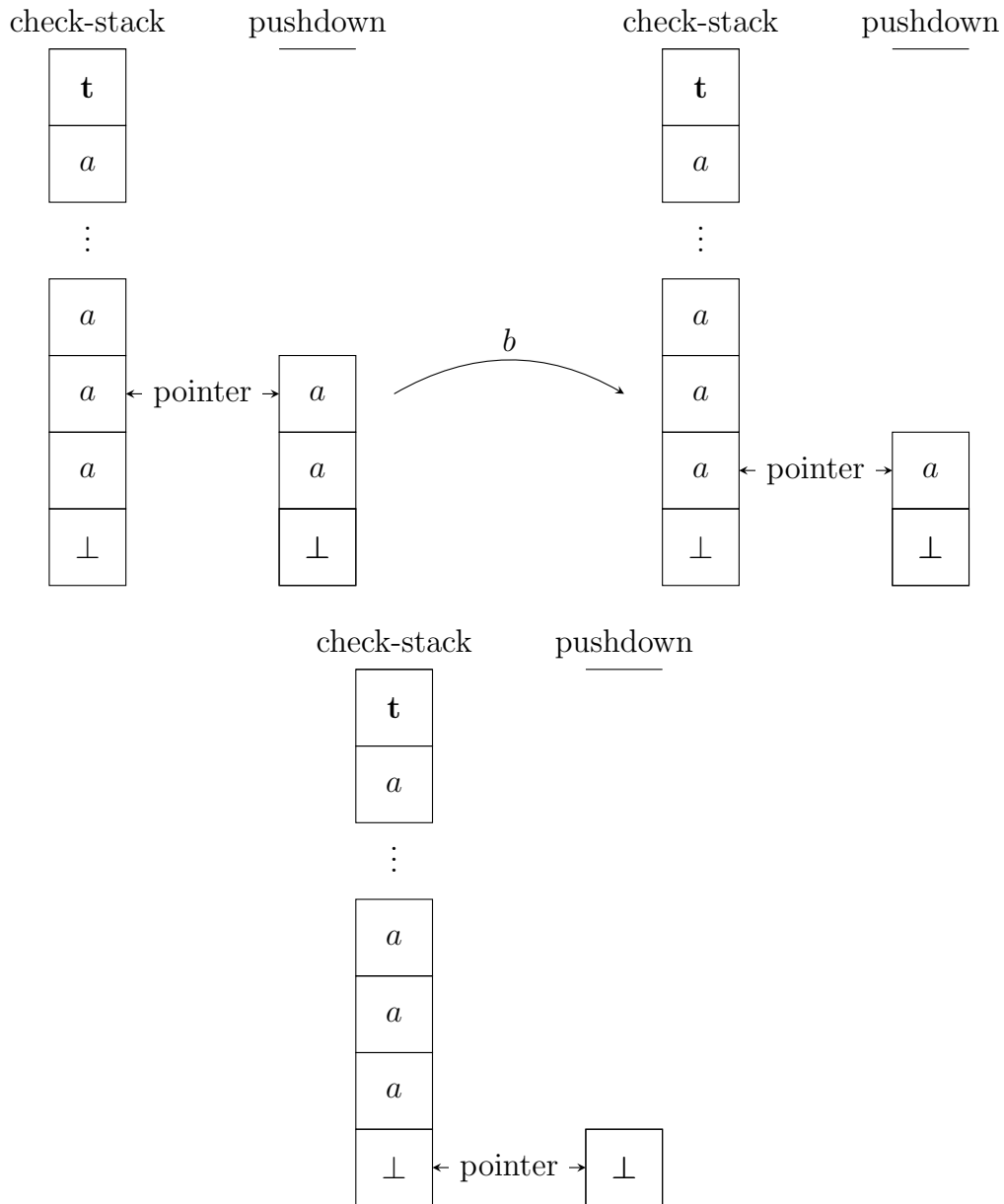


At  $q_b$  we enter the second sub-stage, where we read  $b$  letters. We give the graph for the second sub-stage below.

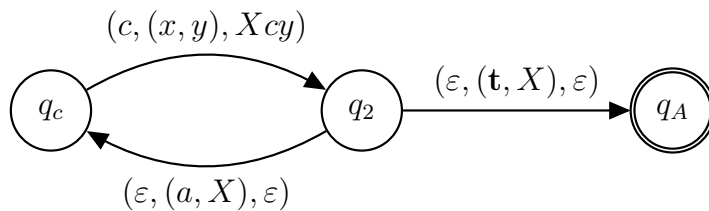


The state  $q_b$  checks that the number of occurrences of the letter  $b$  are the same as those of  $a$  by deleting an  $a$  letter from the pushdown stack for every  $b$  letter in

the input. We can only exit the loop at  $q_b$  by emptying the pushdown stack and then using the edge  $q_b \xrightarrow{(\varepsilon, (\perp, \perp), \perp)} q_c$ . Therefore at the end of the second stage, the number of  $b$  letters read are the same as  $a$  letters read and the pushdown stack is empty. We give two diagrams below, one showing what reading a single  $b$  letter does to the stack and pointer while the other shows what the stack and pointer look like at the end of this stage.



At  $q_c$  the automaton enters its third sub-stage of processing. In this stage we read  $c$  letters. We give the graph for the third sub-stage below.



The states  $q_c$  and  $q_2$  are analogues of  $q_a$  and  $q_1$ , in that they also serve to check that the number of  $c$  letters in the input is the same as  $a$  letters on the check-stack. The number of  $a$  letters on the check-stack is the number of  $a$  letters in the input, as we have seen in the first sub-stage, and also the same as the number of  $b$  letters in the input, as we have seen in the second sub-stage. Further, the automaton reads  $a$  letters then  $b$  letters and finally it reads  $c$  letters. Moreover, both  $q_0$  and  $q_A$  are accepting states. Therefore the language accepted by the machines is  $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ .

Recall in the beginning of the section, we introduced ETOL as a specific subclass of indexed languages [32, 38, 39]. However we cannot find a machine theoretic proof of this in the literature and thus we shall give a sketch of the argument below.

**Proposition 2.2.41.** *If  $L$  is an ETOL language then  $L$  is indexed.*

*Sketch of proof.* We start with an ETOL language  $L$  and construct a nested stack automaton  $\mathcal{A}$  that we argue accepts  $L$ . We note that if we ignore the symbol  $\mathbf{t}$  then each branch that we shall construct has length 1.

Since  $L$  is an ETOL language, it is accepted by a check-stack pushdown automaton  $\mathcal{A}'$  with states  $Q'$ , input alphabet  $\Sigma'$ , check-stack  $\Delta'$ , pushdown alphabet  $\Gamma'$ , an associated regular language  $\mathcal{R}' \subseteq (\Delta')^*$ , and transitions as described above. Finally  $I'$  and  $F'$  denote the sets of initial and final states of  $\mathcal{A}'$  respectively.

Recall that a check-stack pushdown automaton works in two stages, the first stage preloads a string  $\mu$  from the regular language  $\mathcal{R}$  onto the check-stack. Reading the check-stack top-to-bottom, we see the string  $\mu\perp$ . The second stage of a check-stack pushdown automaton deals with the pushdown stack. In our construction of the nested stack automaton  $\mathcal{A}$ , we shall use the trunk for the regular language (i.e., the first stage) and use branches to simulate the pushdown stack (i.e., the second stage).

Below we shall construct a nested stack automaton  $\mathcal{A}$  such that  $\mathcal{L}(\mathcal{A}) = L$ . We aim for  $\mathcal{A}$  to simulate the action of  $\mathcal{A}'$ .

We require nested stack automaton  $\mathcal{A}$  to have a stack alphabet  $\chi = \Delta' \cup \Gamma'$ . Further for the states  $Q$  of  $\mathcal{A}$ , we set  $Q := Q' \sqcup Q_1 \sqcup Q_{\mathcal{R}}$ , we describe the latter three sets as follows:

- $Q_1$  is a set of auxiliary states, that help with book-keeping which we shall describe later. We use these states to create deterministic sub-automata that enable us to simulate transitions of  $\mathcal{A}'$ .
- Recall that  $\mathcal{R}' \subset \Delta'$  is a regular language. Thus there exists a deterministic finite state automaton  $\mathcal{A}_{\mathcal{R}'}$  that accepts  $\mathcal{R}'$ . Consider the transition diagram of  $\mathcal{A}_{\mathcal{R}'}$ . Reversing the direction of the arrows as well as interchanging the initial and final sets defines a new finite state automaton  $\mathcal{A}_{\mathcal{R}} := (Q_{\mathcal{R}}, \Delta', \delta_{\mathcal{R}}, I_{\mathcal{R}}, F_{\mathcal{R}})$  that accepts the reverse of  $\mathcal{R}'$ . Denote  $\mathcal{L}(\mathcal{A}_{\mathcal{R}})$  by  $\mathcal{R}$ .

Note that we use the input alphabet  $\Sigma$  of  $\mathcal{A}'$  as the input alphabet of  $\mathcal{A}$ . Further the sets of initial and final states of  $\mathcal{A}$  are  $I_{\mathcal{R}}$  and  $F'$  respectively.

Finally, for the transition set  $\delta$  of  $\mathcal{A}$  we set  $\delta := \delta'_{\mathcal{R}} \sqcup \delta_i \sqcup \delta_1 \sqcup \delta_f \sqcup \delta_m$ , we describe these as follows:

- $\delta'_{\mathcal{R}}$  consists of transitions that are similar to the ones of  $\delta_{\mathcal{R}}$  but without reading input. The aim here is to write a string from  $\mathcal{R}$  onto the trunk of the nested stack automaton.
- $\delta_i$  consists of the initial part of transitions in  $\mathcal{A}'$ ,
- $\delta_1$  consists of auxiliary transitions that help with book-keeping. These are the transitions that create the deterministic sub-automata we mentioned above.
- $\delta_f$  consists of the final part of transitions in  $\mathcal{A}'$ .
- $\delta_m$  consists of transitions that move the automaton from simulating the first stage of the CSPD to start simulating the second stage of the CSPD.

We describe the above sets as well as  $Q_1$  in further detail below.

- For every  $q_{\mathcal{R}}, q'_{\mathcal{R}} \in Q_{\mathcal{R}}, x \in \Delta$  where  $((q_{\mathcal{R}}, x), q'_{\mathcal{R}}) \in \delta_{\mathcal{R}}$  we define a transition  $((q_{\mathcal{R}}, \varepsilon, \mathbf{t}x_1), (q'_{\mathcal{R}}, \mathbf{t}xx_1)) \in \mathcal{J}_1 \subseteq \delta'_{\mathcal{R}}$ . Note that reading the trunk top to bottom, we see  $\mathbf{t}\mu\perp$  where  $\mu \in \mathcal{R}$ .

Further note that at the end of these transitions, the automaton would have simulated the first stage of the CSPD machine, i.e., the preloading of the check-stack, but this was done upside down such that reading top to bottom we see  $\mathbf{t}\mu\perp$  where  $\mu \in \mathcal{R}'$ . In a CSPD automaton however, reading the check-stack bottom-to-top we see  $\perp\mu$  where  $\mu \in \mathcal{R}'$ . Finally at the end of the above transitions the automaton is at a state  $f_{\mathcal{R}} \in F_{\mathcal{R}}$  with the pointer pointing at  $\mathbf{t}$ .

- For each  $f_{\mathcal{R}} \in F_{\mathcal{R}}$ , we define an  $\varepsilon$ -transition to every initial state  $i \in I'$  of the automaton  $\mathcal{A}'$  such that  $((f_{\mathcal{R}}, \varepsilon, \mathbf{t}), (i, 0)) \in \mathcal{J}_2 \subseteq \delta_m$ . Note that the automaton is now ready to simulate the second stage of processing of the CSPD automaton.

Recall that a transition  $\xi$  in the CSPD machine  $\mathcal{A}'$  is interpreted as reading a letter  $x \in \Sigma \cup \{\varepsilon\}$  from a state  $p$  with  $y$  at the top of the pushdown stack and  $z$  being the corresponding letter on the check-stack. The automaton then moves to a state  $q$  as well as deletes  $y$  and writes a string  $\omega$  such that at the end of the transition the pointer points to the first letter of  $\omega$  on the pushdown stack and the corresponding letter on the check-stack. Here we shall break up such a transition in three parts. The first part involves reading  $x$  and is how we define the relations in  $\delta_i$ . The second part is a sequence of auxiliary relations that enable us to be able to simulate the writing of  $\omega$  letter by letter and is how we define the relations in  $\delta_1$ . The third and final part involves any readjusting of the pointer we may require to fully simulate the CSPD and is how we define the relations in  $\delta_f$ . We describe these in further detail below. There are three main cases:

- If the top of the pushdown (and thus the check-stack) is  $\perp$ .
- If the top of the pushdown stack is not  $\perp$  and  $\omega \in \Gamma^+$ .
- If the top of the pushdown stack is not  $\perp$  and  $\omega = \varepsilon$ .

For each one of the above cases, we shall define the required states in  $Q_1$  as well as the transitions we need in  $\delta_i, \delta_1$  and  $\delta_f$  below.

Suppose the top of the pushdown stack (and thus the check-stack) is  $\perp$ . Thus  $\omega = \omega_1 \perp$  for some  $\omega_1 \in \Gamma^*$ .

- If  $\omega_1 = \varepsilon$  then the transition  $\xi$  simply changes the automaton from state  $p \in Q'$  to state  $q \in Q'$ . We shall simulate this in  $\mathcal{A}$  as follows:
  - we define a new state  $q_\xi \in Q_1$  and transitions as follows.
  - We define a new transition  $((p, x, \mathbf{t}), (q_\xi, 0)) \in \mathcal{J}_2 \subseteq \delta_i$ .
  - We define a new transition  $((q_\xi, x, \mathbf{t}), (q, 0)) \in \mathcal{J}_2 \subseteq \delta_f$ .

Observe that the above defines a deterministic subautomaton of  $\mathcal{A}$  which simulates the action of  $\xi$  in  $\mathcal{A}'$ .

- If  $\omega_1 = \mu_1 \mu_2 \cdots \mu_k \in \Gamma^+$ , then we simulate the transition  $\xi$  as follows:



- For each  $j \in \{k, k-1, \dots, 1\}$  we define states  $q_{(B,j,\xi)}, q_{(B',j,\xi)} \in Q_1$  such that we define the following new transitions:

$$((p, x, \mathbf{t}), (q_{(B,k,\xi)}, -1)) \in \mathcal{J}_2 \subseteq \delta_i$$

and

$$((q_{B,j,\xi}, \varepsilon, y), (q_{(B',j,\xi)}, \mathbf{t}\mu_j\$y)) \in \mathcal{J}_3 \subseteq \delta_1$$

where  $y \in \Delta$ .

We note that the first transition moves the pointer from  $\mathbf{t}$  down by a single step on the trunk of the nested stack automaton. Where as the set of transitions in  $\mathcal{J}_3$  are branching transitions that create branches off the stack with letters of  $\omega_1$  in them.

- For each  $j \in \{k, k-1, \dots, 2\}$  we define states  $q_{(R_1,j,\xi)}, q_{(R_2,j,\xi)}, q_{(R_3,j,\xi)}$  such that we define the following new transitions:

$$\begin{aligned} ((q_{(B',j,\xi)}, \varepsilon, \mathbf{t}), (q_{(R_1,j,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(R_1,j,\xi)}, \varepsilon, \mu_i), (q_{(R_2,j,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(R_2,j,\xi)}, \varepsilon, \$), (q_{(R_3,j,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(R_3,j,\xi)}, \varepsilon, y), (q_{(B,j-1,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1 \end{aligned}$$

where  $y \in \Delta$ .

Note that the above transitions move the pointer down the stack by one step at a time from the end of a branch back into the trunk and then moving down the trunk by a single step.

- Finally we define a new transition  $((q_{(B',1,\xi)}, \varepsilon, \mathbf{t}), (q, -1)) \in \mathcal{J}_2 \subseteq \delta_f$ .

Observe that the above defines a deterministic subautomaton of  $\mathcal{A}$  which simulates the action of  $\xi$  in  $\mathcal{A}'$ . This is done by branching out by the letters of  $\omega_1$  and then using the pointer to move down the stack into the trunk and then down the trunk. This process is then repeated. Thus we ensure that the letters of  $\omega$  branch out from the trunk such that a letter of  $\omega, u$  branches out from a branching point  $v$  if and only if  $u$  and  $v$  can be viewed by the pointer in the CSPD automaton on the pushdown stack and check-stack respectively at the same time.

Now suppose the top of the pushdown stack (and thus the check-stack) is not  $\perp$ . Assume the top of the pushdown stack is  $y \in \Gamma$ , with the corresponding check-stack letter being  $z \in \chi$ . The transition  $\xi$  can either delete  $y$  or replace  $y$  by a string  $\omega$ . We define a reject state that is a sink,  $Z \in Q_1$ .

- Suppose  $\xi$  deletes  $y$ , we simulate  $\xi$  on the nested stack automaton as follows:
  - We define states  $q_{(C_1,\xi)}, q_{(C_2,\xi)}, q_{(C_3,\xi)}, q_{(C_4,\xi)} \in Q_1$  such that we define the following new transitions:

$$\begin{aligned}
 &((p, x, y), (q_{(C_1,\xi)}, -1)) \in \mathcal{J}_2 \subseteq \delta_i, \\
 &((q_{(C_1,\xi)}, \varepsilon, \$), (q_{(C_2,\xi)}, -1)) \in \mathcal{J}_2 \subseteq \delta_1, \\
 &((q_{(C_2,\xi)}, \varepsilon, z), (q_{(C_3,\xi)}, +1)) \in \mathcal{J}_2 \subseteq \delta_1, \\
 &((q_{(C_2,\xi)}, \varepsilon, \bar{z}), (Z, 0)) \in \mathcal{J}_2 \subseteq \delta_1 \text{ where } \bar{z} \neq z, \\
 &((q_{(C_3,\xi)}, \varepsilon, \$), (q_{(C_4,\xi)}, +1)) \in \mathcal{J}_2 \subseteq \delta_1.
 \end{aligned}$$

Observe that the above defines a deterministic subautomaton of  $\mathcal{A}$  which determines whether or not  $\xi$  can be simulated. If the branching point is not  $z$  then  $\xi$  cannot be simulated and the automaton will use move to the sink state  $Z$ . Otherwise, we can simulate  $\xi$  as the branching point is  $z$ .

- We define states  $P_\xi, D_\xi, q_{(R_1,\xi)} \in Q_1$  such that we define the following new transitions:

$$\begin{aligned}
 &((q_{(C_4,\xi)}, \varepsilon, y), (P_\xi, +1)) \in \mathcal{J}_2 \subseteq \delta_1, \\
 &((P_\xi, \varepsilon, \mathbf{t}y), (D_\xi, \mathbf{t})) \in \mathcal{J}_1 \subseteq \delta_1, \\
 &((D_\xi, \varepsilon, \mathbf{t}\$), (q_{(R_1,\xi)}, \varepsilon)) \in \mathcal{J}_4 \subseteq \delta
 \end{aligned}$$

Observe that the above transitions define a deterministic subautomaton of  $\mathcal{A}$  that deletes  $y$  and then destroys the branch. The pointer then is at the branching point  $z$ . This is the first part of simulating  $\xi$ . The second part returns the pointer of  $\mathcal{A}$  to the correct position on the stack that corresponds to the top of the pushdown stack in  $\mathcal{A}'$ . We do this below.

- We define states  $q_{(R_2,\xi)}, q_{(R_3,\xi)} \in Q_1$  such that we define the following new transitions:

$$\begin{aligned}
 &((q_{(R_1,\xi)}, \varepsilon, z), (q_{(R_2,\xi)}, +1)) \in \mathcal{J}_2 \subseteq \delta_1, \\
 &((q_{(R_2,\xi)}, \varepsilon, \mathbf{t}), (q, 0)) \in \mathcal{J}_2 \subseteq \delta_f, \\
 &((q_{(R_2,\xi)}, \varepsilon, z_1), (q_{(R_3,\xi)}, +1)) \in \mathcal{J}_2 \subseteq \delta_1, \\
 &((q_{(R_3,\xi)}, \varepsilon, \$), (q, +1)) \in \mathcal{J}_2 \subseteq \delta_f.
 \end{aligned}$$

Observe that the above defines a deterministic subautomaton of  $\mathcal{A}$  that moves the pointer to  $\mathbf{t}$  if the branch that was deleted had the branching point  $z$  one step below  $\mathbf{t}$  on the trunk. Otherwise, the pointer is

moved to a letter  $y' \in \Gamma'$  in the nearest branch to  $y$ . This simulates the behaviour of the pointer in a CSPD as the pointer would be at the top of the pushdown stack, this would either be  $\perp$  (simulated here by  $\mathbf{t}$ ) or would be the letter  $y' \in \Gamma'$ .

- Now suppose  $\xi$  replaces  $y$  by  $\omega = \mu_1\mu_2 \cdots \mu_l \in \Gamma^+$ . We simulate this below in a very similar way to how we simulated the case for when the top of the pushdown stack is  $\perp$  and a non-empty string  $\omega$  is written onto the pushdown stack.

- We define states  $q_{(C_1,\xi)}, q_{(C_2,\xi)}, q_{(C_3,\xi)}, q_{(C_4,\xi)} \in Q_1$  such that we define the following new transitions:

$$\begin{aligned} ((p, x, y), (q_{(C_1,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_i, \\ ((q_{(C_1,\xi)}, \varepsilon, \$), (q_{(C_2,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(C_2,\xi)}, \varepsilon, z), (q_{(C_3,\xi)}, +1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(C_2,\xi)}, \varepsilon, \bar{z}), (Z, 0)) &\in \mathcal{J}_2 \subseteq \delta_1 \text{ where } \bar{z} \neq z, \\ ((q_{(C_3,\xi)}, \varepsilon, \$), (q_{(C_4,\xi)}, +1)) &\in \mathcal{J}_2 \subseteq \delta_1. \end{aligned}$$

Observe that the above defines a deterministic subautomaton of  $\mathcal{A}$  which determines whether or not  $\xi$  can be simulated. If the branching point is not  $z$  then  $\xi$  cannot be simulated and the automaton will use move to the sink state  $Z$ . Otherwise, we can simulate  $\xi$  as the branching point is  $z$ .

- We define states  $q_{(P_\xi)}, q_{(B',l,\xi)} \in Q_1$  such that we define the following new transitions:

$$\begin{aligned} ((q_{(C_4,\xi)}, \varepsilon, y), (q_{(P_\xi)}, +1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(P_\xi)}, \varepsilon, \mathbf{t}y), (q_{(B',l,\xi)}, \mathbf{t}\mu_l)) &\in \mathcal{J}_1 \subseteq \delta_1. \end{aligned}$$

Observe that the above defines a deterministic subautomaton of  $\mathcal{A}$  which replaces  $y$  by  $\mu_l$ .

- For each  $j \in \{l-1, l-2, \dots, 1\}$  we define states  $q_{(B,j,\xi)}, q_{(B',j,\xi)} \in Q_1$  such that we define the following new transitions:

$$((q_{(B,j,\xi)}, \varepsilon, y'), (q_{(B',j,\xi)}, \mathbf{t}\mu_j \$y')) \in \mathcal{J}_3 \subseteq \delta_1$$

where  $y' \in \Delta$ .

The above branching transitions create branches off the trunk with the letters of  $\omega$  in them.

- For each  $j \in \{l, l-1, \dots, 2\}$  we define states  $q_{(R_1,j,\xi)}, q_{(R_2,j,\xi)}, q_{(R_3,j,\xi)}$  such that we define the following new transitions:

$$\begin{aligned} ((q_{(B',j,\xi)}, \varepsilon, \mathbf{t}), (q_{(R_1,j,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(R_1,j,\xi)}, \varepsilon, \mu_i), (q_{(R_2,j,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(R_2,j,\xi)}, \varepsilon, \$), (q_{(R_3,j,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1, \\ ((q_{(R_3,j,\xi)}, \varepsilon, y'), (q_{(B,j-1,\xi)}, -1)) &\in \mathcal{J}_2 \subseteq \delta_1 \end{aligned}$$

where  $y' \in \Delta$ .

The above transitions move the pointer down the stack one step at a time from the end of a branch back into the trunk and then moving down the trunk by a single step.

- Finally we define a new transition  $((q_{(B',1,\xi)}, \varepsilon, \mathbf{t}), (q, -1)) \in \mathcal{J}_2 \subseteq \delta_f$ .

Observe that the above defines a deterministic subautomaton of  $\mathcal{A}$  which simulates the action of  $\xi$  in  $\mathcal{A}'$ . This is done by branching out by the letters of  $\omega_1$  and then using the pointer to move down the stack into the trunk and then down the trunk. This process is then repeated. Thus we ensure that the letters of  $\omega$  branch out from the trunk such that a letter of  $\omega$ ,  $u$  branches out from a branching point  $v$  if and only if  $u$  and  $v$  can be viewed by the pointer in the CSPD automaton on the pushdown stack and check-stack respectively at the same time.

Observe that we have written strings backwards in our construction above, this is done to allow the use of the pointer in order to simulate transitions of a CSPD automaton. As we have discussed above the simulations are done via deterministic sub-automata and thus once a transition has started its simulation the only way to read another letter is by exiting the subautomaton and thus finishing the simulation. Therefore the language accepted by the nested stack automaton is equal to the one accepted by a CSPD automaton and thus we see that ETOL languages are indeed indexed.  $\blacksquare$

In the following subsection we formalise some of the ideas discussed above.

#### 2.2.4.2 Formal Definition

We define a check-stack pushdown automaton as in [8] below.

**Definition 2.2.42.** A *check-stack pushdown (or CSPD) automaton*  $\mathcal{A}$  is a 9-tuple  $(Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$  such that:

- $Q$  is a finite set, which is called the *set of states* of  $\mathcal{A}$ ;

- $\Sigma$  is a finite set, which is called the *the tape (or input) alphabet* of  $\mathcal{A}$ ;
- $\Delta$  is a finite set, which is called the *check-stack alphabet* of  $\mathcal{A}$ ;
- $\Gamma$  is a finite set, which is called the *pushdown stack alphabet* of  $\mathcal{A}$ ;
- $I \subseteq Q$  is called the *set of initial states* of  $\mathcal{A}$ ;
- $F \subseteq Q$  is called the *set of final (or accept) states*;
- $\mathcal{R} \subseteq \Delta^*$  is a regular language;
- $\perp \notin \Delta \cup \Gamma$  is the *bottom of stack symbol*; and
- $\delta$  is the transition relation which we describe in greater detail below.

Set

$$A := Q \times (\Sigma \cup \{\varepsilon\}) \times \{(\Delta \times \Gamma) \cup \{(\perp, \perp)\}\}$$

and

$$B := Q \times (\Gamma^* \cup \Gamma^* \perp).$$

Then  $\delta \subseteq A \times B$  is a finite set called the *transition relation* and we refer to an element of  $\delta$  as a transition. Let  $p, q \in Q$  and  $x \in \Sigma \cup \{\varepsilon\}$ . If  $((p, x, (y, z)), (q, \omega)) \in \delta$  then one of the following two conditions hold:

1.  $(y, z) \in \Delta \times \Gamma$  and  $\omega \in \Gamma^*$
2.  $y = z = \perp$  and  $\omega \in \Gamma^* \perp$ .

The stack  $\mathcal{S}$  is formally a pair  $(S_1, S_2)$  where  $S_1 = (s_n, s_{n-1}, \dots, s_1, s_0)$  and  $S_2 = (s'_m, s'_{m-1}, \dots, s'_1, s'_0)$  such that  $s_0 = s'_0 = \perp$ ,  $s_i \in \Delta$ ,  $s'_j \in \Gamma$ ,  $s_n s_{n-1} \cdots s_1 \in \mathcal{R}$  and  $n \geq m$ . ♣

We shall define *configurations* below. This is the first step in defining acceptance of strings by CSPD automata.

**Definition 2.2.43** (Configuration). A *configuration* is a pair  $(q, \mathcal{S})$  where  $q \in Q$  and  $\mathcal{S}$  is the stack. We may refer to the pair  $\mathcal{S}$  as the *stack of the configuration*  $(q, \mathcal{S})$  (or by similar language). ♣

We will now use  $\delta$  to define a set of relations on the set of configurations below.

For each  $x \in \Sigma \cup \{\varepsilon\}$ , we define a relation  $\sim_x$  on the set of configurations as follows. Let  $C_1 = (q_1, (S, S_1))$  and  $C_2 = (q_2, (S, S_2))$  be two configurations, where  $S = (s_n, s_{n-1}, \dots, s_0)$  and  $s_n s_{n-1} \cdots s_1 \in \mathcal{R}$ . Further,  $S_1 = (s'_{m_1}, s'_{m_1-1}, \dots, s'_0)$  and  $S_2 = (s''_{m_2}, s''_{m_2-1}, \dots, s''_0)$ . Then  $C_1 \sim_x C_2$  if and only if both of the following conditions hold:

- $S_2 = (s''_{m_2}, s''_{m_2-1}, \dots, s''_{m_1}, s'_{m_1-1}, s'_{m_1-2}, \dots, s'_0)$  and
- $((q_1, x, (s_{m_1}, s'_{m_1})), (q_2, s''_{m_2} s''_{m_2-1} \dots s''_{m_1})) \in \delta,$

for  $q_1, q_2 \in Q$ . We say  $C_1 \sim_x C_2$  by  $((q_1, x, (s_{m_1}, s'_{m_1})), (q_2, s''_{m_2} s''_{m_2-1} \dots s''_{m_1}))$ , and/or *the automaton uses the transition*  $((q_1, x, (s_{m_1}, s'_{m_1})), (q_2, s''_{m_2} s''_{m_2-1} \dots s''_{m_1}))$  (from  $C_1$  to  $C_2$ ), or by similar language. Let  $\sim_\varepsilon^*$  be the reflexive transitive closure of  $\sim_\varepsilon$ . Before we are ready to define acceptance, we will define *initial* and *accepting configurations*.

**Definition 2.2.44** (Initial Configuration). Let  $C = (q, \mathcal{S})$  be a configuration where  $\mathcal{S} = (S_1, S_2)$ . We say  $C$  is an *initial configuration* if  $q \in I$  and  $S_2 = (\perp)$ . ♣

**Definition 2.2.45** (Accepting Configuration). Let  $C = (q, \mathcal{S})$  be a configuration. We say  $C$  is an *accepting configuration* if  $q \in F$ . ♣

We are now ready to define acceptance.

**Definition 2.2.46** (Acceptance and Language accepted by a CSPD automaton). Let  $\mathcal{A} = (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$  be a CSPD automaton and let  $\omega = \sigma_1 \sigma_2 \dots \sigma_n \in \Sigma^*$ . We say  $\omega$  is *accepted by*  $\mathcal{A}$  if there exist a string  $r_1 r_2 \dots r_k \in \mathcal{R}$  and configurations  $C_0, C_1, \dots, C_{2n+1}$  such that  $C_i = ((r_1, r_2, \dots, r_k, \perp), S_i)$  (for  $0 \leq i \leq 2n+1$ ) and

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* C_3 \sim_{\sigma_2} C_4 \sim_\varepsilon^* \dots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1},$$

where  $C_0$  is an initial configuration and  $C_{2n+1}$  is an accepting configuration.

Let  $\mathcal{L}(\mathcal{A})$  be the set of all strings in  $\Sigma^*$  that are accepted by  $\mathcal{A}$ . We say the *language*  $L$  is *accepted by*  $\mathcal{A}$  if  $\mathcal{L}(\mathcal{A}) = L$ . ♣

If there exists a sequence of configurations  $C_0, C_1, \dots, C_{2n+1} \in Q$  such that

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* \dots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1}, \quad (\hat{5})$$

where  $\sigma_1, \sigma_2, \dots, \sigma_n \in \Sigma$  then we say  $\omega = \sigma_1 \sigma_2 \dots \sigma_n$  can be read from  $C_0$  through the chain  $(\hat{5})$ , or by similar language. The chain of relations (see e.g.  $(\hat{5})$ ) used to read a string is called a *run* of the automaton, we sometimes refer to it as a *run* and sometimes we simply just say *chain* or *chain of relations*. Note that for a string to be accepted, the automaton must be able to read the string from an initial configuration and the last configuration in a chain of relations (such as  $(\hat{5})$ ) is an accept configuration. If a string cannot be read from an initial configuration then it can never be accepted. If a string cannot be read in a particular run (or any runs), we may say the string is *rejected*. Usually if the term *rejected* is used, it simply means that there is some underlying reason as to whether the string cannot be accepted that is fundamentally opposed to how the machine operates.

One of the reasons can be that the string simply cannot be read from an initial configuration (in a particular run). If there are other reasons, they will be made clear in the contexts in which they arise. It will sometimes be useful to designate a state  $q_r$  as *the reject state*. This state will not be a final state of the automaton, and the automaton will have transitions to move to  $q_r$  when the input string is to be rejected. A reject state  $q_r$  will be a *sink state*. A sink is a state from which no other state can be reached. We say a configuration  $C$  is a *sink*, if no other configuration can be reached from  $C$ .

We will now make a notational remark. We will make use of the term  $\varepsilon$ -*transition* (we will revisit this term again in the following section). A transition  $((q, x, (y, z)), (q', w))$  is said to be an  $\varepsilon$ -transition if  $x = \varepsilon$ . Suppose in a chain such as the one above in (5), there exists a configuration  $C$  appearing as  $C \sim_\varepsilon^* C$  in the chain. Further suppose there do not exist configurations  $C'_1, C'_2, \dots, C'_r$  and there does not exist a sequence of  $\varepsilon$ -transitions yielding

$$C \sim_\varepsilon C'_1 \sim_\varepsilon C'_2 \sim_\varepsilon \dots \sim_\varepsilon C'_r \sim_\varepsilon C.$$

Then  $C \sim_\varepsilon^* C$  appears in the chain due to the reflexive property of  $\sim_\varepsilon^*$ . In such a situation, we will remove the appearance of  $\sim_\varepsilon^* C$  from the chain to shorten it.

We are now ready to define an ETOL language.

**Definition 2.2.47** (ETOL languages). A language  $L$  is said to be *ETOL* if there exists a CSPD automaton  $\mathcal{A}$  such that  $L = \mathcal{L}(\mathcal{A})$ . ♣

We shall link the formal definition to our intuitive description of check-stack pushdown automata in the following subsection.

### 2.2.4.3 Link between intuition and formality for CSPD automata

Like various other kinds of automata, the main complication in understanding how CSPD automata work is understanding the transitions.

Let  $\mathcal{A} = (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$  be a check-stack pushdown automaton. Similar to other kinds of automata,  $\mathcal{A}$  has a set of states  $Q$ , an input alphabet  $\Sigma$ , a set of initial states  $I$  and a set of final states  $F$ . In this type of automaton there are two stacks, one called the *check-stack* and the other called the *pushdown stack*. These have alphabet  $\Delta$  and  $\Gamma$  respectively.

Recall that in our Intuitive Description [2.2.4.1](#) we discussed that processing happens in two stages, the first is done by preloading a string from the regular language  $\mathcal{R}$  onto the check-stack. Formally, we do this via initial configurations as the pushdown stack is empty and the check-stack has a string from  $\mathcal{R}$  on it. The second stage in processing involves transitions and we describe these below.

Let  $\alpha \in \delta$  be a transition. Recall from Definition [2.2.42](#) that  $\alpha = (X, Y)$  where  $X$  is a triple and  $Y$  is a pair. The components of  $X$  represent the state  $q$

from which a letter  $x \in \Sigma \cup \{\varepsilon\}$  is read while the letter  $y \in \Gamma \cup \{\perp\}$  is at the top of the pushdown stack with  $z \in \Delta \cup \{\perp\}$  being the corresponding letter in the check-stack. The components of  $Y$  represent the state  $q'$  that the automaton moves to while replacing  $y$  with a string  $\omega \in \Gamma^* \cup \Gamma^* \perp$ . When  $x = \varepsilon$ , we call  $\alpha$  an  $\varepsilon$ -transition. Conversely when  $x \neq \varepsilon$ , we refer to  $\alpha$  as a *reading-transition*.

In our intuitive description, we stated that the bottom-of-stack symbol  $\perp$  cannot be written in the middle of the stack nor can it be deleted. We achieve this formally by the via points number 1 and 2 (respectively) in Definition [2.2.42](#).

We note that in our intuitive description, we say that the pointer is used to view the top of the pushdown stack and the corresponding letter (i.e., the one at the same height) on the check-stack. Recognising that letters on the check-stack and pushdown stack are at the same height is done formally via having the stacks be sequences. Letters at the same height intuitively are at the same position in the sequences. For instance, the top of the pushdown stack is the letter  $s'_m$  on the left side of  $S_2$  in Definition [2.2.42](#),  $s'_m$  is in position  $m$  as we see in Definition [2.2.42](#) when we count from the right. The corresponding letter on the check-stack is the  $m^{\text{th}}$  coordinate in  $S_1$ .

**Example 2.2.48.** Let  $\mathcal{A} = (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$  be a CSPD automaton such that:

- $Q = \{q_0, q_1, q_2, q_a, q_b, q_c, q_A\}$ ;
- $\Sigma = \{a, b, c\}$ ;
- $\Delta = \{a, \mathbf{t}\}$ ;
- $\Gamma = \{a, c, X\}$ ;
- $I = \{q_0\}$ ;
- $F = \{q_0, q_A\}$ ;
- $\mathcal{R} = \mathbf{t}\{a\}^*$ ; and
- the transition relation  $\delta$  consists of the following transitions:

$$\begin{aligned}
& ((q_0, \varepsilon, (\perp, \perp)), (q_a, \perp)), ((q_a, a, (\perp, \perp)), (q_1, Xa\perp)), \\
& ((q_a, a, (a, a)), (q_1, Xaa)), ((q_1, \varepsilon, (a, X)), (q_a, \varepsilon)), \\
& ((q_1, \varepsilon, (\mathbf{t}, X)), (q_b, \varepsilon)), ((q_b, b, (a, a)), (q_b, \varepsilon)), \\
& ((q_b, \varepsilon, (\perp, \perp)), (q_c, \perp)), ((q_c, c, (\perp, \perp)), (q_2, Xc\perp)) \\
& ((q_c, c, (a, c)), (q_2, Xcc)), ((q_2, \varepsilon, (a, X)), (q_c, \varepsilon)) \text{ and} \\
& ((q_2, \varepsilon, (\mathbf{t}, X)), (q_A, \varepsilon)).
\end{aligned}$$



This defines the check-stack pushdown automaton corresponding to the graph in Example [2.2.40](#).

#### 2.2.4.4 Visualisation and Determinism

We can visualise check-stack pushdown automata in a similar way to how we visualised other kinds of automata, i.e., through an edge-labelled directed graph, called a transition diagram or state diagram.

**Definition 2.2.49** (Transition Diagram/State Diagram).

Let  $\mathcal{A} = (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$  be a check-stack pushdown automaton. We will define the *transition diagram (or state diagram)* to be a directed graph  $\Gamma(\mathcal{A})$  as follows:

- The vertex set is  $Q$ .
- The edges of  $\Gamma(\mathcal{A})$  are all triples  $(q, (x, (z, y), \omega), q')$  for some  $q, q' \in Q$  such that  $((q, x, (z, y), (q', \omega)) \in \delta$ . We shall denote the set of edges by  $E$ . We represent an edge  $(q, (x, (z, y), \omega), q')$  by an arrow from  $q$  to  $q'$  with  $(x, (z, y), \omega)$  as its label.

We represent transition diagrams by figures of edge-labelled directed graphs. The vertices of the graphs will be denoted by circles with the corresponding states written in them. Each vertex representing a state in  $I$  will be denoted by a diamonds instead of the usual circle. Similarly each vertex representing a state in  $F$  will be denoted by a double circle instead of a single one. ♣

An example of a transition diagram of a check-stack pushdown automaton can be found in Example [2.2.40](#) where we give the transition diagram for the pushdown automaton given in Example [2.2.48](#).

As with previous types of automata, we define determinism for check-stack pushdown automata below.

**Definition 2.2.50** (Deterministic and Non-deterministic).

Let  $\mathcal{A} = (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$  be a check-stack pushdown automaton. We say  $\mathcal{A}$  is deterministic if both of the following conditions hold:

- for every  $p \in Q$  and pair  $(y, z) \in \Delta \times \Gamma \cup \{(\perp, \perp)\}$  if there exists a pair  $(q, \omega)$  such that  $((p, \varepsilon, (y, z), (q, \omega)) \in \delta$ , then for all  $\sigma \in \Sigma$  there is no pair  $(q', \omega')$  such that  $((p, \sigma, (y, z), (q', \omega')) \in \delta$ , and
- if  $p \in Q, \sigma \in \Sigma$  and  $(y, z) \in \Delta \times \Gamma \cup \{(\perp, \perp)\}$ , then there is at most one pair  $(q'', \omega)$  such that  $((p, \sigma, (y, z), (q'', \omega)) \in \delta$ .

Otherwise, we say the automaton is *non-deterministic*. ♣

## 2.3 Classes of groups defined by languages

This thesis is about the interplay between group theory and formal languages. There has been a history in trying to classify groups by certain language theoretic constraints. We now introduce two sets that link groups and languages which we use to create constraints later.

**Definition 2.3.1** (Word problem and co-word problem). Let  $G$  be a finitely generated group and let  $X$  be a symmetrically closed finite generating set for  $G$ . Then

$$WP(G, X) := \{w \in X^* \mid w =_G 1_G\}$$

is the *word problem* of  $G$  with respect to  $X$ . Similarly,

$$coWP(G, X) := \{w \in X^* \mid w \neq_G 1_G\}$$

is the *co-word problem* of  $G$  with respect to  $X$ . ♣

Observe that  $WP(G, X), coWP(G, X) \subset X^*$  and thus are formal languages with  $X$  as an alphabet. The task we are interested in is the following.

Let  $\mathcal{C}$  be a class of languages. Classify the groups with word problem belonging to  $\mathcal{C}$  and similarly classify groups with co-word problem belonging to  $\mathcal{C}$ . (★)

Before we continue we note that the word problem and the co-word problem by definition are dependent on the choice of generating set of the group in question. However, in certain cases, the language class of the word problem and co-word problem is independent of the choice of generating set. Therefore our task is well defined. We make this more precise below, but first we make the following definition.

**Definition 2.3.2** (Closure under inverse homomorphism). [27] [29] Let  $\mathcal{C}$  be a class of languages. We say  $\mathcal{C}$  is *closed under inverse homomorphisms* if whenever  $\phi : X^* \rightarrow Y^*$  is a monoid homomorphism and  $L \subseteq Y^*$  is in  $\mathcal{C}$  then  $\phi^{-1}(L) := \{w \in X^* \mid \phi(w) \in L\}$  is in  $\mathcal{C}$ . ♣

**Lemma 2.3.3.** ([27] Lemma 1) *Let  $\mathcal{C}$  be a class of languages that are closed under inverse homomorphisms. Let  $G$  be a finitely generated group with finite generating sets  $X$  and  $Y$ . Then:*

1.  $WP(G, X) \in \mathcal{C}$  if and only if  $WP(G, Y) \in \mathcal{C}$ , and
2.  $coWP(G, X) \in \mathcal{C}$  if and only if  $coWP(G, Y) \in \mathcal{C}$ .

*Proof.* Let  $X$  and  $Y$  be finite symmetric generating sets for  $G$ . Now suppose  $WP(G, X) \in \mathcal{C}$ . The sets  $X$  and  $Y$  then determine natural surjective homomorphisms

$$\pi : X^* \rightarrow G \text{ and } \rho : Y^* \rightarrow G.$$

For each  $y \in Y$ , fix  $w_y \in X^*$  such that

$$\pi(w_y) = \rho(y);$$

that is,  $w_y =_G y$ . As  $Y^*$  is a free monoid, there is a monoid homomorphism

$$\phi : Y^* \rightarrow X^*$$

given by

$$y \mapsto w_y \text{ for each } y \in Y.$$

Then  $\pi\phi(y) = \pi(w_y) = \rho(y)$  for each  $y \in Y$ , so  $\pi\phi = \rho$ . Now let  $\omega \in Y^*$ . Then

$$\begin{aligned} \omega \in \phi^{-1}(WP(G, x)) &\text{ if and only if } \phi(\omega) = 1_G \\ &\text{ if and only if } \pi\phi(\omega) = 1_G \\ &\text{ if and only if } \rho(\omega) = 1_G \\ &\text{ if and only if } \omega \in WP(G, Y). \end{aligned}$$

Therefore

$$WP(G, Y) = \phi^{-1}(WP(G, X))$$

and

$$coWP(G, Y) = \phi^{-1}(coWP(G, X)).$$

As  $\mathcal{C}$  is closed under inverse homomorphisms and  $WP(G, X) \in \mathcal{C}$ , then  $WP(G, Y)$  belongs to  $\mathcal{C}$  as well. Similarly if  $coWP(G, X) \in \mathcal{C}$  then  $coWP(G, Y) \in \mathcal{C}$ . ■

With Lemma [2.3.3](#) in mind, it is permissible to say that a group  $G$  has word problem, or co-word problem, in the class of languages  $\mathcal{C}$  without referring to the generating set, provided  $\mathcal{C}$  is closed under inverse homomorphisms. Let  $G$  be a group. If the word problem of  $G$  is in a class of languages  $\mathcal{C}$  then we say  $G$  is a  $\mathcal{C}$ -group, or simply  $G$  is  $\mathcal{C}$ . Similarly if the co-word problem of  $G$  in  $\mathcal{C}$  then we say  $G$  is a  $co\mathcal{C}$  group, or simply  $G$  is  $co\mathcal{C}$ . We shall return to this nomenclature various times for various classes of languages. Finally we note that all classes of languages we discuss in this thesis are known to be closed under inverse homomorphisms [\[29\]](#) [\[1\]](#) [\[18\]](#). In particular, it is shown in Theorems 3.5 and 6.3 of [\[29\]](#) that the classes of regular languages and context-free languages are closed under inverse homomorphisms. It is also shown in Lemma 3.2, Corollary 1 and Corollary 2(iv) of [\[1\]](#) that the class of indexed languages are closed under inverse homomorphisms.

Finally in [18], it is shown that the class of ETOL languages are closed under inverse homomorphisms.

Before we discuss classes of groups defined by their (co-)word problem belonging to a specific language class, we will briefly state a property that will be useful in various places throughout this thesis. This property is *ignoring the word problem*. Suppose  $G$  is a group whose co-word problem is accepted by a certain kind of automaton  $\mathcal{M}$ . We say  $\mathcal{M}$  *ignores the word problem* of  $G$  if upon reading a string  $v =_G 1_G$ ,  $\mathcal{M}$  returns to the configuration it was in before reading  $v$ . (This occurs unless during the processing of  $v$ ,  $\mathcal{M}$  moves to a sink state.) This is formally defined in Definition 2.3.27. However, we simply wish to highlight that this concept not only applies to co-indexed groups, but can be generalised to other settings as well.

### 2.3.1 Regular Groups

In what follows we shall explore what happens when we set  $\mathcal{C}$  to be the class of regular languages in  $\star$  on page 56.

Recall the class of regular languages is closed under complementation (see Proposition 2.2.13). Therefore we have the following result.

**Lemma 2.3.4.** *Let  $G$  be a finitely generated group, with a symmetrically closed finite generating set  $X$ . Then  $WP(G, X)$  is regular if and only if  $coWP(G, X)$  is regular.*

Before we continue our discussion regarding classifying groups with regular word problem, we shall state a lemma that we make use of later.

**Lemma 2.3.5.** *Let  $G$  be finitely generated infinite group and let  $X$  be a finite generating set for  $G$ . Then for all  $n \in \mathbb{N}$  there exists strings of length  $n$  in  $X^*$  which do not contain substrings of positive length that are equal in  $G$  to  $1_G$ .*

*Proof.* Assume there exists  $n \in \mathbb{N}$  such that for every string length  $n$  in  $X^*$  there exists a substring of positive length that is equal to  $1_G$  when evaluated in  $G$ .

We shall now show that every element  $g \in G$  can be written as a product of length less than  $n$  in  $X$ . Thus we conclude that  $G$  is finite which is a contradiction.

Assume there exists an element  $g \in G$  that cannot be written as a product of length less than  $n$  in  $X$ . Let  $\omega \in X^*$  be a minimal string representing  $g$ ;  $w =_G g$ . Note that by our assumption  $|\omega| \geq n$ . Write  $\omega = \omega_1\omega_2$  where  $|\omega_1| = n$  where  $\omega_1, \omega_2 \in X^*$ . By our initial assumption,  $\omega_1 = s\omega_0t$  where  $|\omega_0| \geq 1$  and  $w_0 =_G 1_G$ . Then

$$g =_G \omega_1\omega_2 = s\omega_0t\omega_2 =_G st\omega_2$$

and  $|st\omega_2| = |\omega| - |\omega_0| < |\omega|$ . This contradicts that  $\omega$  is an expression for  $g$  of minimal length.

Therefore every element  $g \in G$  can be written as a product of length less than  $n$  in  $X$  and  $G$  is finite. This is a contradiction to our original assumption and thus the claim holds.  $\blacksquare$

Due to Lemma 2.3.4 we shall only consider the word problem. The following result due to Ansimov ([3]) classifies groups whose word problem (and thus whose co-word problem) is regular.

**Proposition 2.3.6.** (Anisimov, [3]) *Let  $G$  be a group and let  $X$  be a symmetrically closed finite generating set for  $G$ . Then  $G$  is finite if and only if  $WP(G, X)$  is regular.*

*Proof.* Let  $G$  be a finite group and let  $X$  be a symmetrically closed generating set. Let  $\Gamma(G, X)$  be the Cayley graph of  $G$  with respect to  $X$ . In our discussion in 2.2.1.4 we discussed how to convert a finite edge-labelled directed graph into a finite state automaton. We shall now convert  $\Gamma(G, X)$  into a finite state automaton  $\mathcal{A} = (G, X, \delta, I, F)$  following the process described in 2.2.1.4. The states of  $\mathcal{A}$  are the vertices of  $\Gamma(G, X)$ , i.e., the state set is the underlying set of the group  $G$ ; the alphabet set is labelling set for  $\Gamma(G, X)$  and thus is  $X$ ; the transition relation  $\delta$  is defined by the edge set of  $\Gamma(G, X)$ , i.e., if  $(g_1, x, g_2)$  is an edge in  $\Gamma(G, X)$  then  $((g_1, x), g_2)$  is a transition, and finally  $I = F = \{1_G\}$ . By definition of the Cayley graph, there is a unique edge labelled by  $x$  (for every  $x \in X$ ) from every vertex  $g$  to some vertex  $g'$  therefore  $\mathcal{A}$  is deterministic and we write  $g \sim_x g'$  to represent this transition as defined in 2.2.1.2 (and used in Definition 2.2.10). Therefore the automaton  $\mathcal{A}$  can read any input string  $\omega$  (from  $1_G$ ) and end at a uniquely determined state  $q_\omega$ . Further recall that since  $\mathcal{A}$  is deterministic, there are no  $\varepsilon$ -transitions therefore when reading a string we do not have to consider sequences of  $\varepsilon$ -transitions.

Now, let  $\omega = x_1x_2 \cdots x_n \in X^*$ . Since  $\mathcal{A}$  is deterministic there exist uniquely determined states  $q_1, q_2, \dots, q_n$  such that

$$1_G \sim_{x_1} q_1 \sim_{x_2} q_2 \sim_{x_3} \cdots \sim_{x_n} q_n.$$

Observe that by construction of  $\mathcal{A}$  (since  $\mathcal{A}$  is induced by  $\Gamma(G, X)$ ),  $x_1x_2 \cdots x_i =_G q_i$ . Therefore  $q_n = 1_G$  if and only if  $x_1x_2 \cdots x_n =_G 1_G$ . So  $\omega \in \mathcal{L}(\mathcal{A})$  if and only if  $\omega =_G 1_G$  since  $F = \{1_G\}$ . Therefore  $\mathcal{L}(\mathcal{A}) = WP(G, X)$ .

Let  $G$  be a finitely generated infinite group with a symmetrically closed finite generating set  $X$  such that  $WP(G, X)$  is regular. Let  $\mathcal{A}$  be a deterministic finite state automaton with initial state  $q_0$  such that  $\mathcal{L}(\mathcal{A}) = WP(G, X)$ . Set  $k := |Q_{\mathcal{A}}|$ . Let  $t > k$ . By Lemma 2.3.5, there is a string  $\omega$  of length  $t$  such that  $\omega$  does not

contain any substring of positive length that evaluates to  $1_G$  in  $G$ . Since  $t > k$ , when  $\mathcal{A}$  reads  $\omega$ , it must visit a state  $q_l$  multiple times in the reading process, i.e., there exists prefixes  $\omega_1 = x_1x_2 \cdots x_l$  and  $\omega_1\omega_2 = x_1x_2 \cdots x_lx_{l+1} \cdots x_m$  of  $\omega$  such that

$$q_0 \sim_{x_1} q_1 \sim_{x_2} q_2 \sim_{x_3} \cdots \sim_{x_l} q_l$$

and

$$q_0 \sim_{x_1} q_1 \sim_{x_2} q_2 \sim_{x_3} \cdots \sim_{x_l} q_l \sim_{x_{l+1}} q_{l+1} \sim_{x_{l+2}} q_{l+2} \sim_{x_{l+3}} \cdots \sim_{x_m} q_m$$

where  $q_l = q_m$ . Since  $\mathcal{A}$  is deterministic and  $\omega_1\omega_1^{-1} \in WP(G, X)$  there are states  $q_{l+1}, q_{l+2}, \dots, q_{2l}$  such that

$$q_0 \sim_{x_1} q_1 \sim_{x_2} q_2 \sim_{x_3} \cdots \sim_{x_l} q_l \sim_{x_{l-1}^{-1}} q_{l+1} \sim_{x_{l-2}^{-1}} q_{l+2} \sim_{x_{l-3}^{-1}} \cdots \sim_{x_1^{-1}} q_{2l}$$

where  $q_{2l} \in F$ . Note that when reading  $\omega_1^{-1}$  one does not know a priori that the states visited are the ones passed through when  $\omega_1$  was read but in reverse (as in the case for the Cayley graph). We shall now show that  $\omega_1\omega_2\omega_1^{-1}$  is also in  $\mathcal{L}(\mathcal{A})$ . Note that the state of the automaton upon reading  $\omega_1\omega_2$  is the same as the state of the automaton upon reading  $\omega_1$  (as above). Therefore reading  $\omega_1^{-1}$  after reading  $\omega_1$  puts the automaton in the same state as reading  $\omega_1^{-1}$  after reading  $\omega_1\omega_2$ . Thus if  $\omega_1\omega_1^{-1}$  is accepted then so is  $\omega_1\omega_2\omega_1^{-1}$ . Formally we see this since

$$q_0 \sim_{x_1} q_1 \sim_{x_2} q_2 \sim_{x_3} \cdots \sim_{x_l} q_l \sim_{x_{l+1}} q_{l+1} \sim_{x_{l+2}} q_{l+2} \sim_{x_{l+3}} \cdots \sim_{x_m} q_m = q_l$$

and

$$q_l \sim_{x_{l-1}^{-1}} q_{l+1} \sim_{x_{l-2}^{-1}} q_{l+2} \sim_{x_{l-3}^{-1}} \cdots \sim_{x_1^{-1}} q_{2l}.$$

However since  $\omega_2 \neq 1_G$  (by choice of  $\omega$ ),  $\omega_1\omega_2\omega_1^{-1} \neq 1_G$  and therefore  $\omega_1\omega_2\omega_1^{-1} \notin WP(G, X)$  contradicting  $\mathcal{L}(\mathcal{A}) = WP(G, X)$  as required.  $\blacksquare$

With the above theorem, groups with regular word problem and co-word problem are classified. In the following section we turn our attention to a generalisation of the above result.

### 2.3.2 Context-free Groups

Unlike the case of regular languages, context-free languages are not closed under complementation. Therefore we do not know whether or not an analogous result to Lemma [2.3.4](#) is true a priori. As such we shall in this section focus on the word problem.

**Definition 2.3.7** (Context-free group). Let  $G$  be a finitely generated group,  $G$  is said to be *context-free* if there exists a pushdown automaton that accepts its word problem with respect to some (and hence any) symmetrically closed finite generating set.  $\clubsuit$

A generalisation of the result due to Anisimov (Proposition [2.3.6](#)) is the following result by Muller and Schupp. The result first appears in [\[34\]](#) (where it is Theorem III). There, *accessibility* is one of the conditions. However the version we give below relies on the work of Dunwoody [\[20\]](#) and thus removes the *accessibility* condition.

**Theorem 2.3.8** (Muller–Schupp). *Let  $G$  be a finitely generated group and let  $X$  be a symmetrically closed set finite generating set. Then  $G$  is virtually free if and only if  $WP(G, X)$  is context-free.*

We will not prove the above theorem. However we will construct a pushdown automaton below that will accept the word problem of a virtually free group. Further we note that the automata we present below are non-deterministic. However the complement of the languages they accept, i.e., the co-word problem of the groups are also accepted by pushdown automata. (We prove this below as well.) We give non-deterministic automata because we believe the automata to be more natural and more easily understandable. However, it is shown in [\[34, 35\]](#) that virtually free groups are precisely the class of groups whose word problem is accepted by deterministic pushdown automata.

We shall start by defining a function  $\rho$  which we shall use in various places throughout this thesis.

**Definition 2.3.9.** Let  $F_n$  be the free group of rank  $n \geq 2$  and let  $Z \subseteq F_n$  be any finite symmetrically closed set. We now define a function  $\rho_Z : Z^* \rightarrow Z^*$  recursively as follows. First  $\rho_Z(\varepsilon) = \varepsilon$  and  $\rho_Z(z) = z$  for all  $z \in Z$ . Suppose that  $\omega = z_1 z_2 \cdots z_n$  is a string of length  $n$  in  $Z^*$  and  $\rho_Z(z_1 z_2 \cdots z_{n-1})$  has already been defined. Further suppose  $\rho_Z(z_1 z_2 \cdots z_{n-1}) = y_1 y_2 \cdots y_k$  where  $y_1, y_2, \dots, y_k \in Z$ . Then define

$$\rho_Z(z_1 z_2 \cdots z_{n-1} z_n) = \begin{cases} y_2 y_3 \cdots y_k & \text{if } z_n = y_1^{-1} \\ z_n y_1 y_2 \cdots y_k & \text{if } z_n \neq y_1^{-1}. \end{cases}$$

♣

Note that  $\rho_Z$  has the effect of performing a recursive free reduction of a string  $\omega$  followed by reversing the result. Equivalently, we may think of  $\rho_Z$  as reversing  $\omega$  followed by a free reduction of  $\omega^R$ . Observe that  $\rho(\omega) = \varepsilon$  if and only if  $\omega \in WP(F_Z, Z)$ . Further, we will usually drop the subscript  $Z$  from  $\rho_Z$  as it will be clear from context. We shall illustrate, by way of example, how  $\rho$  works. We do this by calculating the image of the string  $abb^{-1}a^{-1}ba$  under  $\rho$ . (Here the set  $Z$  is assumed to be  $\{a, b, a^{-1}, b^{-1}\}$ .)

$$\begin{aligned}
\rho(a) &= a \\
\rho(ab) &= ba \\
\rho(abb^{-1}) &= a \\
\rho(abb^{-1}a^{-1}) &= \varepsilon \\
\rho(abb^{-1}a^{-1}b) &= b\varepsilon = b \\
\rho(abb^{-1}a^{-1}ba) &= ab.
\end{aligned}$$

We will now construct a pushdown automaton that accepts the word problem of a virtually free group  $G$ . We shall start by constructing a pushdown automaton to accept the word problem of the free group  $F_n$  on  $n$  generators. Let  $X' := \{x_1, x_2, \dots, x_n\}$  be a free generating set for  $F_n$  and let  $X$  be the symmetric closure of  $X'$ .

**Definition of  $P_{F_n}^{(W,1)}$ :** We define a pushdown automaton

$$P_{F_n}^{(W,1)} := (Q_{F_n}^{(W,1)}, I_{F_n}^{(W,1)}, \Sigma_{F_n}^{(W,1)}, \chi_{F_n}^{(W,1)}, \delta_{F_n}^{(W,1)}, \perp, F_{F_n}^{(W,1)})$$

accepting  $WP(F_n, X)$  as follows.

- The state set is  $Q_{F_n}^{(W,1)} = \{i_0, i_1, q_A, \Psi\}$ ;
- the set of initial states is  $I_{F_n}^{(W,1)} = \{i_0\}$ ;
- the input alphabet is  $\Sigma_{F_n}^{(W,1)} = X$ ;
- the stack alphabet is  $\chi_{F_n}^{(W,1)} = X$ ;
- the transition set is  $\delta_{F_n}^{(W,1)}$  consisting of the following transitions:
  - (T1) for every  $x \in X$  there is a transition  $((i_0, x, \perp), (i_0, x\perp))$ ; that is, there is an edge in the transition diagram from  $i_0$  to  $i_0$  labelled  $(x, \perp, x\perp)$ ,
  - (T2) for every  $x \in X$  there is a transition  $((i_0, x, x^{-1}), (i_0, \varepsilon))$ ; that is, there is an edge in the transition diagram from  $i_0$  to  $i_0$  labelled  $(x, x^{-1}, \varepsilon)$ ,
  - (T3) for every  $x \in X$  and every  $y \in X \setminus \{x^{-1}\}$  there is a transition  $((i_0, x, y), (i_0, xy))$ ; that is, there is an edge in the transition diagram from  $i_0$  to  $i_0$  labelled  $(x, y, xy)$ ,
  - (T4) for every  $z \in X \cup \{\perp\}$  there is a transition  $((i_0, \varepsilon, z), (i_1, z))$ ; that is, there is an edge in the transition diagram from  $i_0$  to  $i_1$  labelled  $(\varepsilon, z, z)$ ,



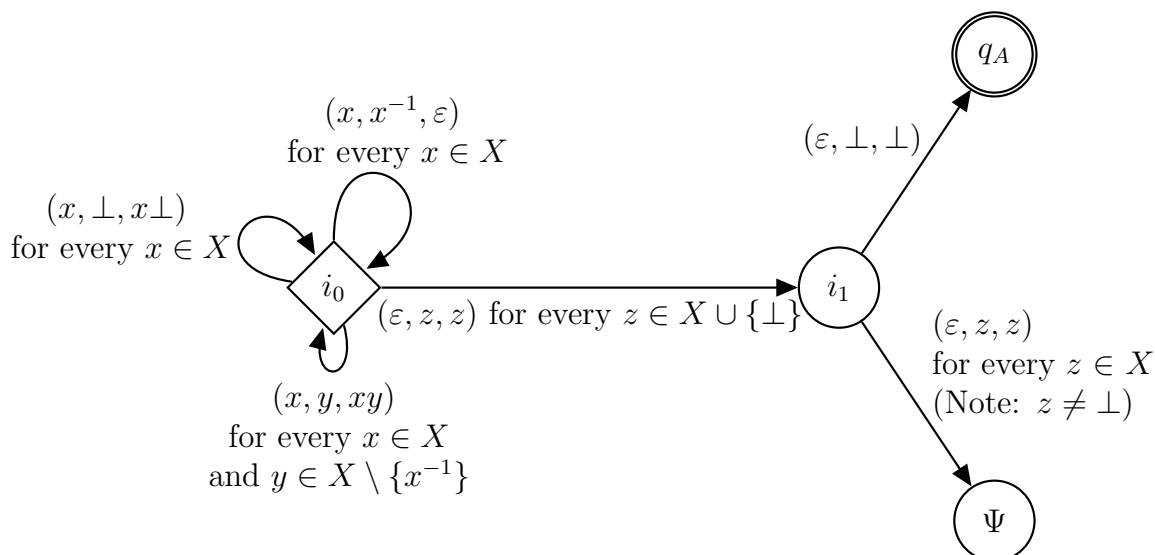


Figure 2.1: Automaton for  $F_n$

- (T5) there is a transition  $((i_1, \varepsilon, \perp), (q_A, \perp))$ ; that is, there is an edge in the transition diagram from  $i_1$  to  $q_A$  labelled  $(\varepsilon, \perp, \perp)$ ,
- (T6) for every  $z \in X$  (and  $z \neq \perp$ ), there is a transition  $((i_1, \varepsilon, z), (\Psi, z))$ ; that is, there is an edge in the transition diagram from  $i_1$  to  $\Psi$  labelled  $(\varepsilon, z, z)$ ; and

- the set of final states is  $F_{F_n}^{(W,1)} = \{q_A\}$ .

We represent the above automaton in [2.1](#). The reader should note however that for compactness we have used a single edge in the diagram to represent multiple transitions.

**Detailed discussion about the transitions of  $P_{F_n}^{(W,1)}$ :** We will elaborate on the transitions listed above and make some observations that will be useful to us when proving  $\mathcal{L}(P_{F_n}^{(W,1)}) = WP(F_n, X)$ .

Let  $\omega = x_1x_2 \cdots x_m \in X^*$ .

- ( $\tau 1$ ) First we recall Definition [2.2.19](#);  $\omega$  is accepted if there is a run whose last configuration is an accept configuration. In particular, all input letters must be read. Note that all transitions from  $i_0$  to  $i_1$ , from  $i_1$  to  $q_A$ , and from  $i_1$  to  $\Psi$  (i.e., the transitions in (T4), (T5) and (T6)) are  $\varepsilon$ -transitions. Further, there are no transitions from  $i_1, q_A$  or  $\Psi$  to  $i_0$ . Therefore the automaton

cannot use the transitions in (T4), (T5) and (T6) until it has read the whole string  $\omega$ . Thus we have our first observation below.

**Observation 1:** Any chain of relations reading  $\omega$  must be of the following form

$$C_0 \sim_{x_1} C_1 \sim_{x_2} C_2 \sim_{x_3} \cdots \sim_{x_m} C_m \sim_{\varepsilon}^* C_{m+1}.$$

We also note that the only state with reading-transitions is  $i_0$ .

- ( $\tau 2$ ) If  $\omega \neq \varepsilon$  then the only way the automaton can read the first letter  $x_1$  of  $\omega$  is by using a transition in (T1) (as initially the stack is always empty by definition, i.e  $C_0$  is always  $(i_0, \perp)$ .) Thus after that transition the stack is  $x_1\perp$ , i.e  $C_1 = (i_0, x_1\perp)$ .
- ( $\tau 3$ ) We shall show that at  $C_i$ , the stack is  $\rho(x_1x_2 \cdots x_i)\perp$ . We do this by induction. Note that the base was established in ( $\tau 2$ ) as  $\rho(x_1) = x_1$ .

Suppose the first  $i - 1$  letters  $x_1, x_2, \dots, x_{i-1}$  of  $\omega$  have been read by the automaton and the stack is  $\rho(x_1x_2 \cdots x_{i-1})\perp$ . Upon reading  $x_i$ , there are three choices:

- (a) if  $\rho(x_1x_2 \cdots x_{i-1}) = \varepsilon$  then the automaton uses a transition in (T1) and the stack becomes  $x_i\perp = \rho(x_1x_2 \cdots x_i)\perp$ ;
- (b) if the first letter of  $\rho(x_1x_2 \cdots x_{i-1})$  is  $x_i^{-1}$  then the automaton uses a transition in (T2) and the letter at the top of the stack gets deleted, i.e., the stack becomes  $\rho(x_1x_2 \cdots x_i)\perp$ ; and
- (c) otherwise, the first letter of  $\rho(x_1x_2 \cdots x_{i-1})$  is not  $x_i^{-1}$  then the automaton uses a transition in (T3) and  $x_i$  gets written onto the stack, i.e., the stack becomes  $\rho(x_1x_2 \cdots x_i)\perp$ .

Thus we have our second observation.

**Observation 2:**  $C_i = (i_0, \rho(x_1x_2 \cdots x_i)\perp)$  for  $1 \leq i \leq r$ .

- ( $\tau 4$ ) Thus after reading the string  $\omega$ , the stack is  $\rho(\omega)\perp$ .
- ( $\tau 5$ ) Now, as there are no more letters left to be read in  $\omega$ , the automaton can use a transition in (T4) to move from  $i_0$  to  $i_1$  without editing the stack.
- ( $\tau 6$ ) At state  $i_1$ , if the stack is  $\rho(\omega)\perp$  with  $\rho(\omega) = \varepsilon$ , i.e., the stack is empty then the automaton can use the transition in (T5) to move to  $q_A$ . Otherwise, the automaton can use the transitions in (T6).

**Observation 3:**  $q_A$  is the only accept state. Thus any configuration with the state being  $q_A$  is an accept configuration. However, the only transition that moves the state of the automaton to  $q_A$  is  $((i_1, \varepsilon, \perp), (q_A, \perp))$ .

**Proof of  $\mathcal{L}(\mathcal{A}) = WP(F_n, X)$ :** Let  $\omega = x_1x_2\cdots x_m \in X^*$ . By Observations 1 and 2 above, the automaton must read  $\omega$  through the following chain

$$\begin{aligned} (i_0, \perp) &\sim_{x_1} (i_0, \rho(x_1)\perp) \\ &\sim_{x_2} (i_0, \rho(x_1x_2)\perp) \\ &\vdots \\ &\sim_{x_n} (i_0, \rho(x_1x_2\cdots x_m)\perp). \end{aligned} \tag{1}$$

Denote the configuration  $(i_0, \rho(y_1y_2\cdots y_m)\perp)$  by  $C_\omega$ .

Observe that  $\rho(y_1y_2\cdots y_m)\perp = \perp$  if and only if  $\rho(y_1y_2\cdots y_m) = \varepsilon$  which occurs precisely when  $\omega \in WP(F_n, X)$ . Therefore we see that if  $\omega \in WP(F_n, X)$  then the chain of relations in (1) continues as

$$C_\omega = (i_0, \perp) \sim_\varepsilon^* (q_A, \perp).$$

By Observation 3,  $\omega$  is accepted.

Conversely if  $\omega \in coWP(F_n, X)$  then  $\rho(\omega) \neq \varepsilon$ . The chain in (1) becomes

$$C_\omega = (i_0, \rho(\omega)\perp) \sim_\varepsilon^* (i_0, \rho(\omega)\perp) \tag{2.1}$$

$$\text{or } C_\omega = (i_0, \rho(\omega)\perp) \sim_\varepsilon^* (i_1, \rho(\omega)\perp) \tag{2.2}$$

$$\text{or } C_\omega = (i_0, \rho(\omega)\perp) \sim_\varepsilon^* (\Psi, \rho(\omega)\perp) \tag{2.3}$$

due to transitions in (T4) and (T6). Recall the only reading-transitions are from  $i_0$  to  $i_0$  as stated in Observation 1. Thus the automaton always reads  $\omega$  through the chain in (1). Therefore the chain of relations always continues as in (2.1) or (2.2) or (2.3) and  $\omega$  is not accepted.

Therefore the  $WP(F_n, X)$  is the language accepted by the above automaton. ■

We make a notational remark before we continue. Recall the automaton defined above was presented as

$$P_{F_n}^{(W,1)} := (Q_{F_n}^{(W,1)}, I_{F_n}^{(W,1)}, \Sigma_{F_n}^{(W,1)}, \chi_{F_n}^{(W,1)}, \delta_{F_n}^{(W,1)}, \perp, F_{F_n}^{(W,1)}).$$

We note that the superscript  $(W, 1)$  refers to this machine accepting the word problem and that it has a viewing window of size 1. We include the aforementioned details in the naming of the machine as they will be useful later. We will modify various automata and it will be useful to refer to them while understanding what they do without ambiguity.

Let  $G$  be a finite index overgroup of  $F_n$ . As above, we take  $X$  to be the symmetric closure of the free generating set for  $F_n$ . Let  $T'$  be a right transversal for  $F_n$  in  $G$  and we assume that  $1_G \in T'$ . Let  $T$  be the symmetric closure of  $T'$ . Observe that  $Y := X \cup T$  is a symmetrically closed finite generating set for  $G$ . For each

$y \in Y$  and  $t \in T'$ ,  $ty \in G$  thus  $ty = f_{ty}t'$  for some  $f_{ty} \in F_n$  and  $t' \in T'$ . Fix a string  $w_{ty} := x_1x_2 \cdots x_r \in X^*$  such that  $x_1x_2 \cdots x_r =_{F_n} f_{ty}$ .

We shall construct an automaton  $P_G^{(W,1)}$  that accepts the word problem of  $G$ . First we will give an intuitive description of how  $P_G^{(W,1)}$  will work. We then define  $P_G^{(W,1)}$  and follow that with a detailed discussion about the transitions. Finally we prove that  $\mathcal{L}(P_G^{(W,1)}) = WP(G, Y)$ .

**Intuitive Description of  $P_G^{(W,1)}$ :** The state set of the automaton will be a union of  $T'$ , a final state  $q_A$  and some auxiliary states which we shall describe later. The initial state is  $1_G$ . (Recall that  $1_G \in T'$ .) Let  $\omega = \sigma_1\sigma_2 \cdots \sigma_k \in Y^*$ . Upon reading an input letter  $\sigma_i$  from a state  $t \in T'$  the automaton will move through a sequence of auxiliary states ending with the state  $t' \in T'$  such that  $t\sigma_i = f_{t\sigma_i}t'$ . Through the auxiliary states, the automaton will simulate the action of  $P_{F_n}^{(W,1)}$  reading  $w_{t\sigma_i}$ ; that is the auxiliary states will use the transitions in (T1), (T2) and (T3) in our definition of  $P_{F_n}^{(W,1)}$  to read  $w_{t\sigma_i}$  letter by letter. Thus at the end of the sequence of auxiliary states, the stack would consist of the recursive free reduction of  $w_{t\sigma_i}^R S \perp$ ; i.e  $\rho(S^R w_{t\sigma_i}) \perp$ , where  $S \perp$  is the stack before entering the sequence of auxiliary states. After reading the last letter of  $\omega$ , i.e.,  $\sigma_k$  (and going through the sequence of auxiliary states), if the automaton is at the state  $1_G$  then the input string is in the coset that is equal to  $F_n 1_G$  and thus we will check whether the stack is empty or not. If the stack is empty then the input string  $\omega =_G 1_G$  and therefore  $\omega$  will be accepted by moving to the accept state  $q_A$ .

**Definition of  $P_G^{(W,1)}$ :** We are now ready to formally define the automaton

$$P_G^{(W,1)} := (Q_G^{(W,1)}, I_G^{(W,1)}, \Sigma_G^{(W,1)}, \chi_G^{(W,1)}, \delta_G^{(W,1)}, \perp, F_G^{(W,1)}),$$

we do this below:

- The state set  $Q_G^{(W,1)}$  consists of the following states
  - for every element  $t$  in the transversal  $T'$  we shall have a state  $t$  representing that element. We shall refer to this subset of the state set as  $T'$  as well.
  - The states  $q_A$  and  $\Psi$ .
  - For ever  $t \in T'$  and  $y \in Y$ , there is a state  $(t, y)$ . We shall call this collection of states  $K$ .
  - Recall that for every  $(t, y) \in K$  we have fixed a string  $w_{ty} = x_1x_2 \cdots x_r$  such that  $ty =_G w_{ty}t'$  for some  $t' \in T'$ . There is a state representing every letter in the string  $w_{ty}$  and is further indexed by  $(t, y)$ . That is, for

every  $(t, y) \in K$  there exist states named  $q_{(t,y,1)}, q_{(t,y,2)}, \dots, q_{(t,y,r)}$  where  $r$  depends only on  $(t, y)$ ;

- the set of initial states is  $I_G^{(W,1)} = \{1_G\}$ ;
- the input alphabet is  $\Sigma_G^{(W,1)} = Y = X \cup T$ ;
- the stack alphabet is  $\chi_G^{(W,1)} = \chi_{F_n}^{(W,1)} = X$ ;
- the set of final states is  $F_G^{(W,1)} = \{q_A\}$ ;

and finally the transition relation  $\delta_G^{(W,1)}$  consists of the following transitions:

(T1) for every  $t \in T', y \in Y$  and  $z \in X \cup \{\perp\}$  there is a transition  $((t, y, z), ((t, y), z))$ ; that is at state  $t \in T'$  upon reading  $y \in Y$  with  $z$  at the top of the stack one passes to the corresponding state  $(t, y) \in K$  (thus recording both  $t$  and  $y$ ) while not changing the stack.

(T2) (a) For every  $(t, y) \in K$  and  $z_1 \in X \cup \{\perp\}$  there is a transition

$$(((t, y), \varepsilon, z_1), (q_{(t,y,1)}, P_{F_n}^{(W,1)}(i_0, x_1, z_1)))$$

where  $P_{F_n}^{(W,1)}(i_0, x_1, z_1)$  denotes the unique string such that

$$((i_0, x_1, z_1), (i_0, P_{F_n}^{(W,1)}(i_0, x_1, z_1))) \in \delta_{F_n}^{(W,1)};$$

that is, at state  $(t, y)$ , when the top letter of the stack is  $z_1$ , there is a transition to the state  $q_{(t,y,1)}$  that places the same string on the stack as the automaton for  $F_n$  did when reading the letter  $x_1$  with  $z_1$  at the top of its stack.

(b) For every  $(t, y) \in K$  and every  $z' \in X \cup \{\perp\}$  there is a transition

$$((q_{(t,y,i)}, \varepsilon, z'), (q_{(t,y,i+1)}, P_{F_n}^{(W,1)}(i_0, x_{i+1}, z')))$$

for all  $1 \leq i \leq r - 1$  where  $P_{F_n}^{(W,1)}(i_0, x_{i+1}, z')$  denotes the unique string such that

$$((i_0, x_{i+1}, z'), (i_0, P_{F_n}^{(W,1)}(i_0, x_{i+1}, z'))) \in \delta_{F_n}^{(W,1)};$$

that is, at state  $q_{(t,y,i)}$  when the top letter of the stack is  $z'$ , there is a transition to the state  $q_{(t,y,i+1)}$  that places the same string on the stack as the automaton for  $F_n$  did when reading letter  $x_{i+1}$  with  $z'$  at the top of its stack.

- (c) For every  $(t, y) \in K$  and every  $z'' \in X \cup \{\perp\}$  there exists a transition  $((q_{(t,y,r)}, \varepsilon, z''), (t', z''))$ ; that is, at state  $q_{(t,y,r)}$  there is a transition to state  $t'$  that does not edit the stack, such that  $ty =_G w_{ty}t'$ , and  $w_{ty}$  is of length  $r$ .

(T3) There is a transition  $((1_G, \varepsilon, \perp), (q_A, \perp))$ .

(T4) For every  $z \in X$  (and  $z \neq \perp$ ) there is a transition  $((1_G, \varepsilon, z), (\Psi, z))$ .

**Detailed discussion about the transitions of  $P_G^{(W,1)}$ :** Let  $\omega = y_1y_2 \cdots y_k \in Y^*$ . In what follows we assume  $\omega$  is read by  $\mathcal{A}$ , and we shall describe how  $\mathcal{A}$  will use the transitions to read  $\omega$ . We do this regardless of whether or not  $\omega$  is accepted. We will make some structural observations about how  $\mathcal{A}$  reads  $\omega$  in the points labelled by  $(\tau i)$  below and we highlight key points using the **Observations**. This will be useful to us when proving  $\mathcal{L}(P_G^{(W,1)}) = WP(G, Y)$ .

In  $(\tau 1)$  we shall give an overview of what the transitions of the automaton do. We shall also relate this to the definition of acceptance and thus conclude that some transitions must only be used after the last letter of  $\omega$  is read and processed. In  $(\tau 2)$  we discuss the order in which the automaton must use the transitions to process  $\omega$ . We then discuss effects of the transitions on the states and the stack. We separate these into two different points,  $(\tau 2.A)$  and  $(\tau 2.B)$ . These will be recorded in observations to be used later. Finally, we shall make an observation about the accept configurations in  $(\tau 3)$ .

- ( $\tau 1$ ) We first note that transitions in  $(T1)$  are used to read input alphabet. Transitions in  $(T2)$  are  $\varepsilon$ -transitions used to simulate reading strings of the form  $w_{ty}$  in the automaton for  $F_n$ . Transitions in  $(T3)$  and  $(T4)$  are  $\varepsilon$ -transitions that are used to determine whether the input string is equal to the  $1_G$  or not.

Recall Definition [2.2.19](#);  $\omega$  is accepted if there is a run whose last configuration is an accept configuration. In particular, all input letters must be read. As mentioned above, the only reading-transitions are those in  $(T1)$  and occur from states in  $T'$ . Further transitions in  $(T3)$  and  $(T4)$  are  $\varepsilon$ -transitions and there are no transitions from  $q_A$  or  $\Psi$  to any other state. Therefore the automaton cannot use transitions in  $(T3)$  or  $(T4)$  until it has read the whole string  $\omega$ .

- ( $\tau 2$ ) Upon reading an input letter  $y_i \in Y$  from a state  $t \in T'$ , with the being stack  $S\perp$  where  $S \in X^*$ , the automaton must use a sequence of transitions in  $(T1)$  and  $(T2)$  ending at a state  $t'$  such that  $ty_i =_G w_{ty_i}t'$  where  $w_{ty_i} = x_1x_2 \cdots x_r$ . We shall explain why the automaton cannot use any other transitions below

while describing what the transitions are. We do this in  $(\tau 2.A)$ . Moreover, in  $(\tau 2.A)$  we also describe how these transitions change the state of the automaton. In  $(\tau.B)$ , we describe the effects of these transitions on the stack.

$(\tau 2.A)$  Note that given the stack is  $S\perp$  for some  $S \in X^*$ , the transitions below are uniquely determined. We explain why each transition is uniquely determined after each transition below.

- (a) First the automaton uses a transition in  $(T1)$  to move to the state  $(t, y_i)$ .  $(T3)$  and  $(T4)$  may only be used after all input letters have been processed. Accordingly, if we are in state  $t$  and reading a letter  $y_i$ , then we must use a transition from  $(T1)$ . Suppose the letter at the top of the stack is  $z$  then there is only one choice for what the transition is, namely  $((t, y_i, z), ((t, y_i), z))$ . As the other transitions used in (b)-(d) are from  $(T2)$ , this transition is the only reading-transition. Note that using this transition the stack has not been edited.
- (b) The only transitions from  $(t, y_i)$  are those in  $(T2)(a)$ . Thus the automaton must use a transition in  $(T2)(a)$ . Recall that for every  $(t, y_i) \in K$  and  $z \in X \cup \{\perp\}$  there is a unique transition

$$(((t, y), \varepsilon, z), (q_{(t,y,1)}, P_{F_n}^{(W,1)}(i_0, x_1, z))),$$

and thus this is the transition that must be used by the automaton. Thus the transition moves the state to  $q_{(t,y_i,1)}$ , while writing the string  $P_{F_n}^{(W,1)}(i_0, x_1, z)$  onto the stack. Recall the string  $P_{F_n}^{(W,1)}(i_0, x_1, z)$  is the string that is written on the stack by  $\mathcal{A}_{F_n}^{(W,1)}$  upon reading  $x_1$  from  $i_0$  with  $z$  on the stack.

- (c) For every  $1 \leq j \leq r-1$ , the only transitions from  $q_{(t,y_i,j)}$  are those in  $(T2)(b)$ . Thus the automaton must use a transition in  $(T2)(b)$  to move from state  $q_{(t,y_i,j)}$  to state  $q_{(t,y_i,j+1)}$ . Recall that for every  $1 \leq j \leq r-1$ , with  $z'_j$  at the top of the stack, there is a unique transition from  $q_{(t,y_i,j)}$ , namely

$$(((q_{(t,y_i,j)}, \varepsilon, z'_j), (q_{(t,y_i,j+1)}, P_{F_n}^{(W,1)}(i_0, x_{j+1}, z'_j))),$$

and thus this is the transition that must be used by the automaton. Thus the transition moves the state from  $q_{(t,y_i,j)}$  to  $q_{(t,y_i,j+1)}$  via the transition writing  $P_{F_n}^{(W,1)}(i_0, x_{j+1}, z'_j)$  at the top of the stack.

- (d) The only transitions from  $q_{(t,y_i,r)}$  are those in  $(T2)(c)$ . Thus the automaton must use a transition in  $(T2)(c)$ . Recall for every  $z \in X \cup \{\perp\}$

at the top of the stack, there is a unique transition from  $q_{(t,y_i,r)}$  to  $t'$  where  $ty_i =_G w_{ty_i}t'$ , namely

$$((q_{(t,y_i,r)}, \varepsilon, z), (t', z)).$$

As the transitions in (T2)(b) are uniquely determined, the letter at the top of the stack from state  $q_{(t,y_i,r)}$ , say  $z'$ , is uniquely determined. Thus the transition in (T2)(c) that must be used by the automaton is

$$((q_{(t,y_i,r)}, \varepsilon, z'), (t', z')).$$

This moves the automaton from state  $q_{(t,y_i,r)}$  to state  $t'$  while not editing the stack.

Observe that in the above points, the transitions used were the only ones that could have been used. Thus there is no other way to be able to read an input letter apart from going through the sequence above. Therefore the automaton must go through the sequence of states as stated above every time an input letter  $y_i$  is read from a state  $t \in T'$  and so we have the following observation. Recall that a configuration of a PDA is a pair whose first coordinate is a state and the second is the stack.

**Observation 4:** Any chain of relations reading  $\omega = y_1y_2 \cdots y_k$  must be of the following form

$$\begin{aligned} C_0 &\sim_{y_1} C_{y_1}^1 \sim_{\varepsilon} C_1^1 \sim_{\varepsilon} C_2^1 \sim_{\varepsilon} \cdots \sim_{\varepsilon} C_{|w_{1_G y_1}|}^1 \sim_{\varepsilon} C_1 \\ &\sim_{y_2} C_{y_2}^2 \sim_{\varepsilon} C_1^2 \sim_{\varepsilon} C_2^2 \sim_{\varepsilon} \cdots \sim_{\varepsilon} C_{|w_{t_1 y_2}|}^2 \sim_{\varepsilon} C_2 \\ &\vdots \\ &\sim_{y_i} C_{y_i}^i \sim_{\varepsilon} C_1^i \sim_{\varepsilon} C_2^i \sim_{\varepsilon} \cdots \sim_{\varepsilon} C_{|w_{t_{i-1} y_i}|}^i \sim_{\varepsilon} C_i \\ &\vdots \\ &\sim_{y_k} C_{y_k}^k \sim_{\varepsilon} C_1^k \sim_{\varepsilon} C_2^k \sim_{\varepsilon} \cdots \sim_{\varepsilon} C_{|w_{t_{k-1} y_k}|}^k \sim_{\varepsilon} C_k \\ &\sim_{\varepsilon}^* C_{\omega} \end{aligned}$$

where the following conditions hold.

- (a) The configurations  $C_0, C_1, C_2, \dots, C_k$  have their first coordinate being  $t_0, t_1, \dots, t_k \in T'$  respectively such that  $t_0 = 1_G$  and

$$t_{i-1}y_i =_G w_{t_{i-1}t_i}$$

for  $1 \leq i \leq k$ .



- (b) The configurations  $C_{y_i}^i$  have their first coordinate being  $(t_{i-1}, y_i)$  for  $1 \leq i \leq k$ .
- (c) The configurations  $C_j^i$  have their first coordinate being  $q_{(i-1,i,j)}$ .
- (d) Finally the chain of relations can be partitioned in the following way

$$\overbrace{C_{i-1} \sim_{y_i} C_{y_i}^i}^{(T1)} \underbrace{\sim_{\varepsilon} C_1^i}_{(T2)(a)} \overbrace{\sim_{\varepsilon} C_2^i \sim_{\varepsilon} \dots \sim_{\varepsilon} C_{|w_{t_{i-1}y_i}|}^i}^{(T2)(b)} \underbrace{\sim_{\varepsilon} C_i}_{(T2)(c)}$$

where

- $C_{i-1} \sim_{y_i} C_i$  by a transition in  $(T1)$ ,
- $C_{y_i}^i \sim_{\varepsilon} C_1^i$  by a transition in  $(T2)(a)$ ,
- $C_j^i \sim_{\varepsilon} C_{j+1}^i$  by a transition in  $(T2)(b)$  for  $1 \leq j \leq |w_{t_{i-1}y_i}| - 1$ ,
- $C_{|w_{t_{i-1}y_i}|}^i \sim_{\varepsilon} C_i$  by a transition in  $(T2)(c)$

for  $1 \leq i \leq k$  and the transitions above are uniquely determined as  $C_0$  is always the initial configuration, i.e  $C_0 = (1_G, \perp)$  and that is independent of  $\omega$ . Finally, as at  $C_k$  the automaton has finished processing the last input letter

- $C_k \sim_{\varepsilon}^* C_{\omega}$  by a transition in  $(T3), (T4)$  or  $C_{\omega} = C_k$ .

If  $\omega \neq \varepsilon$  then there exists first letter  $y_1$  of  $\omega$ . The automaton reads  $y_1$  from the initial state  $1_G$  with the stack being empty, going through the states

$$(1_G, y_1), q_{(1_G, y_1, 1)}, q_{(1_G, y_1, 2)}, \dots, q_{(1_G, y_1, |w_{1_G y_1}|)}$$

and ending at state  $t_1$  where  $1_G y_1 =_G w_{1_G y_1} t_1$ .

( $\tau 2.B$ ) We shall now elaborate on the effect of the above transitions on the stack. In what follows, whenever we refer to a transition, we mean the unique transition that the automaton must use from a certain state with a particular letter at the top of the stack, as outlined above.

We note that since transitions in  $(T1), (T2)(c), (T3)$ , and  $(T4)$  do not alter the stack we have the following observation.

**Observation 5:** For  $1 \leq i \leq k$ , the following configurations have the same second coordinate

- (a)  $C_{i-1}$  and  $C_{y_i}^1$ ,
- (b)  $C_{|w_{t_{i-1}y_i}|}^i$  and  $C_i$ , and

(c)  $C_k$  and  $C_\omega$ .

Below, we investigate what the stack is for the configurations  $C_{|w_{t_{i-1}y_i}|}^i$ . We do so by first investigating what the stack is for  $C_{|w_{1_G y_1}|}^1$ , this is done in Point I below by going through the transitions as outlined in ( $\tau 2.A$ ). In Point II, we generalise what we do in Point I, to understand the stack for  $C_{|w_{t_{i-1}y_i}|}^i$ . Similar to I, in II we go through the transitions as outlined in ( $\tau 2.A$ ) to understand their effects on the stack.

I We start with the reading of the first letter  $y_1$  of  $\omega$  (if  $\omega \neq \varepsilon$ ). Recall that initially the stack is empty. First recall that  $1_G y_1 =_G w_{1_G y_1} t_1$  and suppose  $w_{1_G y_1} = x_1 x_2 \cdots x_{|w_{1_G y_1}|}$ . As in **Observation 4** (and ( $\tau 2.A$ )), the automaton goes through the following sequence of transitions in order.

- i. First the automaton uses a transition in ( $T1$ ). The only transition that can be used is  $((1_G, y_1, \perp), ((1_G, y_1), \perp))$ . Here the stack is unchanged, i.e., the second coordinate of  $C_{y_1}^1$  is  $\perp$ .
- ii. Now the automaton uses a transition in ( $T2$ )(a). Here the only transition that can be used is  $((1_G, y_1), \varepsilon, \perp), (q_{1_G, y_1, 1}, x_1 \perp)$  and thus the second coordinate of  $C_1^1$  is  $x_1 \perp = \rho(x_1) \perp$ .
- iii. Now the automaton uses transitions in ( $T2$ )(b). We shall show that the second coordinate of  $C_i^1$  is  $\rho(x_1 x_2 \cdots x_i) \perp$ . We do this by induction. Note that the base case is in the previous point.

Suppose the second coordinate of  $C_{i-1}^1$  is  $\rho(x_1 x_2 \cdots x_{i-1}) \perp$ . There are three choices for what the transition is by which  $C_{i-1}^1 \sim_\varepsilon C_i^1$ .

- If  $\rho(x_1 x_2 \cdots x_{i-1}) = \varepsilon$  then the only choice for the transition is  $((q_{1_G, y_1, i-1}, \varepsilon, \perp), (q_{1_G, y_1, i}, x_i \perp))$  and thus stack becomes  $x_i \perp = \rho(x_1 x_2 \cdots x_i) \perp$ . That is, the second coordinate of  $C_i^1$  is  $\rho(x_1 x_2 \cdots x_i) \perp$ .
- If the first letter of  $\rho(x_1 x_2 \cdots x_{i-1})$  is  $x_i^{-1}$  then the only choice for the transition is  $((q_{1_G, y_1, i-1}, \varepsilon, x_i^{-1}), (q_{1_G, y_1, i}, \varepsilon))$  and the letter at the top of the stack gets deleted. That is, the second coordinate of  $C_i^1$  is  $\rho(x_1 x_2 \cdots x_i) \perp$ .
- Otherwise, the first letter of  $\rho(x_1 x_2 \cdots x_{i-1})$  is  $z \neq x_i^{-1}$  then the only choice for the transition is  $((q_{1_G, y_1, i-1}, \varepsilon, z), (q_{1_G, y_1, i}, x_i z))$  and  $x_i$  gets written onto the stack. That is, the second coordinate of  $C_i^1$  is  $\rho(x_1 x_2 \cdots x_i) \perp$ .

Therefore for every configuration  $C_i^1$ , the second coordinate is  $\rho(x_1 x_2 \cdots x_i) \perp$ . In particular  $C_{|w_{1_G y_1}|}^1$  has  $\rho(w_{1_G y_1}) \perp$  as its second coordinate.

- iv. Finally the automaton uses a transition in  $(T2)(c)$ . There is only one that can be used; if the top of the stack at  $C_{|w_{1Gy_1}|}^1$  is  $z$  then the transition is  $((q_{(1G, y_1, |w_{1Gy_1}|)}, \varepsilon, z), (t_1, z))$  and the stack is unchanged. Thus the second coordinate of  $C_1$  is  $\rho(w_{1Gy_1})\perp$ .

II We shall now show that the second coordinate of  $C_{|wt_{i-1}y_i|}^i$  is

$$\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-1}y_i})\perp.$$

We do this by induction. Note that the base case is in I above.

Suppose the second coordinate of  $C_{|wt_{i-2}y_{i-1}|}^{i-1}$  is

$$\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}})\perp.$$

First we note that by **Observation 5** the second coordinate of  $C_{i-1}$  is the same as that of  $C_{|wt_{i-2}y_{i-1}|}^{i-1}$  and thus  $C_{y_i}^i$  also has the same second coordinate. Recall that  $t_{i-1}y_i =_G w_{t_{i-1}y_i}t_i$  and suppose  $w_{t_{i-1}y_i} = b_1b_2 \cdots b_m \in X^*$ . As in **Observation 4** (and the  $(\tau 2.A)$ ), the automaton goes through the following sequence of transitions in order.

i. First the automaton uses a transition in  $(T2)(a)$ . There are three choices for what the transition is by which  $C_{y_i}^i \sim_\varepsilon C_1^i$ .

- If  $\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}}) = \varepsilon$  then the only choice for the transition is  $((t_{i-1}, y_i), \varepsilon, \perp), (q_{(t_{i-1}, y_i, 1)}, b_1\perp)$ ; thus the second coordinate of  $C_1^i$  is  $b_1\perp = \rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}}b_1)\perp$ .
- If the first letter of  $\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}})$  is  $b_1^{-1}$  then the only choice for the transition is  $((t_{i-1}, y_i), \varepsilon, b_1^{-1}), (q_{(t_{i-1}, y_i, 1)}, \varepsilon)$  and the letter at the top of the stack gets deleted. Thus the second coordinate of  $C_1^i$  is  $\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}}b_1)\perp$ .
- Otherwise, the first letter of  $\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}})$  is  $z \neq b_1^{-1}$ . Then the only choice for the transition is

$$(((t_{i-1}, y_i), \varepsilon, z), (q_{(t_{i-1}, y_i, 1)}, b_1z))$$

and  $b_1$  gets written onto the stack. Thus the second coordinate of  $C_1^i$  is  $\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}}b_1)\perp$ .

ii. Now the automaton uses transitions in  $(T2)(b)$ . We shall show that the second coordinate is  $C_j^i$  is  $\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}}b_1b_2 \cdots b_j)\perp$ . We do this by induction. We note the base case is in the previous point.

Suppose the second coordinate of  $C_{j-1}^i$  is

$$\rho(w_{1Gy_1}w_{t_1y_2} \cdots w_{t_{i-2}y_{i-1}}b_1b_2 \cdots b_{j-1})\perp.$$

There are three choices for the transition by which  $C_{j-1}^i \sim_\varepsilon C_j^i$ .

- If  $\rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}} b_1 b_2 \cdots b_{j-1}) = \varepsilon$  then the only choice for the what the transition is

$$((q_{(t_{i-1}, y_i, j-1)}, \varepsilon, \perp), (q_{(t_{i-1}, y_i, j)}, b_j));$$

the second coordinate of  $C_j^i$  is

$$b_j \perp = \rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}} b_1 b_2 \cdots b_{j-1} b_j).$$

- If the first letter of  $\rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}} b_1 b_2 \cdots b_{j-1})$  is  $b_j^{-1}$  then the only choice for the transition is

$$((q_{(t_{i-1}, y_i, j-1)}, \varepsilon, b_j^{-1}), (q_{(t_{i-1}, y_i, j)}, \varepsilon))$$

and the letter at the top of the stack gets deleted. Thus the second coordinate of  $C_j^i$  is

$$\rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}} b_1 b_2 \cdots b_{j-1} b_j).$$

- Otherwise the first  $\rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}} b_1 b_2 \cdots b_{j-1})$  is  $z \neq b_j^{-1}$ . Then the only choice for the transition is

$$((q_{(t_{i-1}, y_i, j-1)}, \varepsilon, z), (q_{(t_{i-1}, y_i, j)}, b_j z))$$

and  $b_j$  gets written onto the stack. Thus the second coordinate of  $C_j^i$  is

$$\rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}} b_1 b_2 \cdots b_{j-1} b_j).$$

This finishes our induction and in particular we see the second coordinate of  $C_{|w_{t_{i-1} y_i}|}^i$  is

$$\rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp$$

as required. Thus we have the following observation.

**Observation 6:** For  $1 \leq i \leq k$ , the second coordinate of  $C_{|w_{t_{i-1} y_i}|}^i$  is

$$\rho(w_{1_G y_1} w_{t_1 y_2} \cdots w_{t_{i-1} y_i}) \perp.$$

( $\tau 3$ ) Finally by definition of the automaton we have the following observation.

**Observation 7:**  $q_A$  is the only accept state. Thus any configuration with the state being  $q_A$  is an accept configuration. However, the only transition that moves the state of the automaton to  $q_A$  is  $((1_G, \varepsilon, \perp), (q_A, \perp))$ .

**Proof of  $\mathcal{L}(P_G^{(W,1)}) = WP(G, Y)$ :** Let  $\omega = \sigma_1\sigma_2\cdots\sigma_k \in Y^*$ . By **Observations 4, 5 and 6** the automaton must read  $\omega$  through the following chain

$$\begin{aligned}
(1_G, \perp) &\sim_{\sigma_1} ((1_G, \sigma_1), \perp) \sim_{\varepsilon}^* (t_1, \rho(w_{1_G\sigma_1})\perp) \\
&\sim_{\sigma_2} ((t_1, \sigma_2), \rho(w_{1_G\sigma_1})\perp) \sim_{\varepsilon}^* (t_2, \rho(w_{1_G\sigma_1}w_{t_1\sigma_2})\perp) \\
&\sim_{\sigma_3} ((t_2, \sigma_3), \rho(w_{1_G\sigma_1}w_{t_1\sigma_2})\perp) \sim_{\varepsilon}^* (t_3, \rho(w_{1_G\sigma_1}w_{t_1\sigma_2}w_{t_2\sigma_3})\perp) \\
&\vdots \\
&\sim_{\sigma_k} ((t_{k-1}, \sigma_k), \rho(w_{1_G\sigma_1}w_{t_1\sigma_2}w_{t_2\sigma_3}\cdots w_{t_{k-2}\sigma_{k-1}})\perp) \\
&\sim_{\varepsilon}^* (t_k, \rho(w_{1_G\sigma_1}w_{t_1\sigma_2}w_{t_2\sigma_3}\cdots w_{t_{k-2}\sigma_{k-1}}w_{t_{k-1}\sigma_k})\perp), \tag{3}
\end{aligned}$$

where  $t_{i-1}\sigma_i =_G w_{t_{i-1}\sigma_i}t_i$ . Recall that the last configuration in (3) above is the configuration  $C_k$  in **Observation 4**. We shall call it  $C_k$  throughout this proof as well.

Now observe

$$\sigma_1\sigma_2\cdots\sigma_i =_G w_{1_G\sigma_1}w_{t_1\sigma_2}\cdots w_{t_{i-1}\sigma_i}t_i$$

and

$$w_{1_G\sigma_1}w_{t_1\sigma_2}\cdots w_{t_{i-1}\sigma_i} =_{F_n} \rho(w_{1_G\sigma_1}w_{t_1\sigma_2}\cdots w_{t_{i-1}\sigma_i})^R.$$

Therefore  $\omega =_G 1_G$  if and only if

$$\rho(w_{1_G\sigma_1}w_{t_1\sigma_2}\cdots w_{t_{k-1}\sigma_k}) = \varepsilon$$

and  $t_k = 1_G$ . Thus  $\omega =_G 1_G$  if and only if  $C_k = (1_G, \perp)$ . Recall there is a transition  $(T3) ((1, \varepsilon, \perp), (q_A, \perp)) \in \delta_G^{(W,1)}$  and so the chain in **(3)** continues

$$C_k \sim_{\varepsilon}^* (q_A, \perp).$$

Therefore if  $\omega \in WP(G, Y)$  then  $\omega$  is accepted by **Observation 7**.

Conversely suppose  $\omega \in coWP(G, Y)$  then either

$$\rho(w_{1_G\sigma_1}w_{t_1\sigma_2}\cdots w_{t_{k-1}\sigma_k}) \neq \varepsilon$$

or  $t_k \neq 1_G$ . Since the only transition to  $q_A$  is from  $1_G$  (and there is no chain of  $\varepsilon$ -transitions from  $t_k$  to  $1_G$  if  $t_k \neq 1_G$ ) then if  $t_k \neq 1_G$  then  $\omega$  cannot be accepted. If  $t_k = 1$  then

$$\rho(w_{1_G\sigma_1}w_{t_1\sigma_2}\cdots w_{t_{k-1}\sigma_k}) \neq \varepsilon.$$

Set  $\overline{\rho(\omega)} := \rho(w_{1_G\sigma_1}w_{t_1\sigma_2}\cdots w_{t_{k-1}\sigma_k})$ . The chain in **(3)** continues

$$C_k = (1_G, \overline{\rho(\omega)}\perp) \sim_{\varepsilon}^* (1_G, \overline{\rho(\omega)}\perp) \tag{4.1}$$

$$C_k = (1_G, \overline{\rho(\omega)}\perp) \sim_{\varepsilon}^* (\Psi, \overline{\rho(\omega)}\perp) \tag{4.2}$$

due to transitions in (T4). Finally we recall that the only reading transitions are those listed in (T1) in  $\delta_G^{(W,1)}$  and thus the automaton can only read  $\omega$  through the chain in (3) as in **Observations 4, 5 and 6**. Therefore the chain of relations always continues as in (4.1) or (4.2) (or  $t_k \neq 1$ ) and  $\omega$  is not accepted.

Therefore  $WP(G, Y)$  is the language accepted by  $P_G^{(W,1)}$ . ■

**Remark 2.3.10.** The above automaton accepts after reading the string if the state is  $q_A$ . The only transition to  $q_A$  is from state  $1_G$  when the stack is empty. ♠

As mentioned before  $P_G^{(W,1)}$  is nondeterministic, however a very simple modification defines a new automaton  $P_G^{(cW,1)}$  accepting  $coWP(G, Y)$ . The modification is as follows: the new automaton is defined in exactly the same way, with the exception of the final states  $F_G^{(cW,1)}$ . We define  $F_G^{(cW,1)} := \{\Psi\} \cup (T' \setminus \{1\})$ . Note the transition relation for  $P_G^{(cW,1)}$  is the same as that for  $P_G^{(W,1)}$  thus they behave the same way. Therefore the above arguments hold with the caveat that the final states are now different. In particular, if a string was accepted in  $P_G^{(W,1)}$  it cannot be accepted in  $P_G^{(cW,1)}$  and vice versa. By construction,  $P_G^{(cW,1)}$  accepts  $coWP(G, Y)$ . (We shall revisit this PDA in Chapter 4.) Further observe that if  $\omega' =_G 1$  and  $t \in T'$  then  $t\omega = f_{t\omega}t$  with  $f_{t\omega} =_{F_n} 1$ . Therefore from a configuration  $(t, S)$  reading a string  $\omega' =_G 1$ , we return to  $(t, S)$ . This observation will be useful to us later, and in particular we call this property *ignoring the word problem*.

In the following section we shall discuss groups with context-free co-word problem.

### 2.3.3 $co\mathcal{CF}$ Groups

In this section we shall briefly discuss the class of groups whose co-word problem is context-free. We start with the definition below.

**Definition 2.3.11** ( $co\mathcal{CF}$  Group). [27] Let  $G$  be a finitely generated group,  $G$  is said to be  $co\mathcal{CF}$  if there exists a pushdown automaton that accepts its co-word problem with respect to some (and hence any) symmetrically closed finite generating set. ♣

As we have discussed in the previous section, the word problem for a virtually free group  $G$  is accepted by a (deterministic) pushdown automaton. Thus the co-word problem of  $G$  is also accepted by a (deterministic) pushdown automaton, and so we see that  $G$  is also  $co\mathcal{CF}$ . Therefore context-free groups are contained in the class of  $co\mathcal{CF}$  groups. However there are groups that are  $co\mathcal{CF}$  but not context-free. In particular,  $\mathbb{Z}^2$  is  $co\mathcal{CF}$  (we see this below) but not virtually-free and thus is not context-free. Therefore the class of  $co\mathcal{CF}$  is a proper container class for context-free groups.

The class of  $co\mathcal{CF}$  groups has a number of interesting closure properties that have become standard to aim for when studying groups defined by a language theoretic class.

**Proposition 2.3.12.** [27] *The class of  $co\mathcal{CF}$  groups is closed under:*

- *passing to finitely generated subgroups,*
- *passing to finitely indexed overgroups,*
- *taking (finite) direct products, and*
- *standard restricted wreath product with context-free top group.*

Since  $\mathbb{Z}$  is context-free it is  $co\mathcal{CF}$ . Due to Proposition 2.3.12  $\mathbb{Z}^2$  is  $co\mathcal{CF}$ . However for the sake of completeness and familiarising the reader with the technicalities involved, we show this below directly. However, we will start by giving a definition of various functions  $\gamma_X$  which we shall use in various places throughout this thesis.

**Definition 2.3.13.** Let  $Z' := \{a_1, a_2, \dots, a_n\}$  be any set and let  $Z$  be the symmetric closure of  $Z'$ . Further let  $x \in Z'$  then set  $X := \{x, x^{-1}\}$ . We define a monoid homomorphism  $\phi_X : Z^* \rightarrow Z^*$  as follows. First let  $z \in Z \cup \{\varepsilon\}$  then

$$\phi_X(z) = \begin{cases} z & \text{if } z \in X \\ \varepsilon & \text{otherwise.} \end{cases}$$

Now let  $\omega' = z_1 z_2 \dots z_n \in Z^*$  then we define  $\phi_X(\omega') = \phi_X(z_1) \phi_X(z_2) \dots \phi_X(z_n)$ .

Now we define a function  $\gamma_X : Z^* \rightarrow Z^*$  by  $\gamma_X(\omega) = \rho_X \circ \phi_X(\omega)$ . ♣

We note the effect of  $\gamma_X$  on a string  $\omega$  is deleting all occurrences of letters not in  $X$  to get a string  $\omega'$  and then applying  $\rho_X$  to  $\omega'$ . We illustrate this by way of example. Let  $\{a, b, a^{-1}, b^{-1}\}$  be a set. We calculate the image of  $abb^{-1}a^{-1}ba$  under  $\gamma_{\{a, a^{-1}\}}$ . First we apply  $\phi_{\{a, a^{-1}\}}$ :  $\phi_{\{a, a^{-1}\}}(abb^{-1}a^{-1}ba) = aa^{-1}a$ . Now we apply  $\rho_{\{a, a^{-1}\}}$  (as defined in Definition 2.3.9):  $\rho_{\{a, a^{-1}\}}(aa^{-1}a) = a$ .

**Proposition 2.3.14.**  $\mathbb{Z}^2$  is  $co\mathcal{CF}$ .

We shall first define a pushdown automaton  $\mathcal{A}$  accepting the co-word problem of  $\mathbb{Z}^2$  with respect to the standard generating set  $X := \{a, a^{-1}, b, b^{-1}\}$ . We follow this with a brief intuitive description of how  $\mathcal{A}$  works. We then give a sequence of lemmata ending with a proof of  $\mathcal{L}(\mathcal{A}) = coWP(\mathbb{Z}^2, X)$ . Following the proof, we provide two example with explicit calculations, the reader may wish to take detour and visit the examples before reading the proof.

**Definition of  $\mathcal{A}$ :** We define  $\mathcal{A} := (Q, I, \Sigma, \chi, \delta, \perp, F)$  as follows.

- The state set is  $Q = \{i_0, A, B, C, q_A\}$ ;
- the set of initial states is  $I = \{i_0\}$ ;
- the input alphabet is  $\Sigma = \{a, a^{-1}, b, b^{-1}\}$ ;
- the stack alphabet is  $\chi = \Sigma$ ;
- the set of final states is  $F = \{q_A\}$ ; and

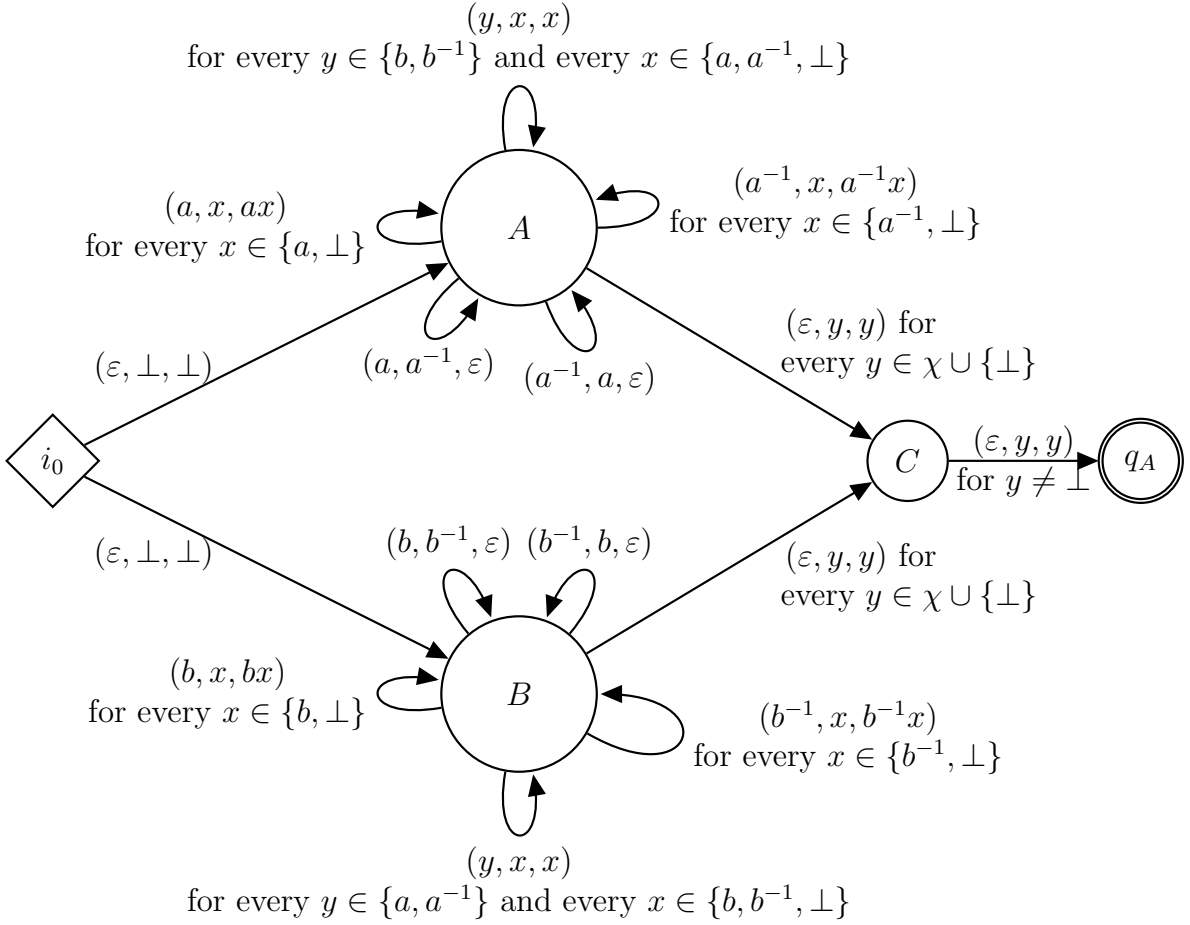
we describe the transition relation  $\delta$  below. The transition relation  $\delta$  consists of the following transitions.

- (T1) There exist transitions  $((i_0, \varepsilon, \perp), (A, \perp))$  and  $((i_0, \varepsilon, \perp), (B, \perp))$ ; that is, there is two edges in the transition diagram from  $i_0$  labelled  $(\varepsilon, \perp, \perp)$ , one to  $A$  and the other to  $B$ .
- (T2) (a) For every  $x \in \{a, \perp\}$  there is a transition  $((A, a, x), (A, ax))$ ; that is, there is an edge in the transition diagram from  $A$  to  $A$  labelled  $(a, x, ax)$ ,  
(b) for every  $x \in \{a^{-1}, \perp\}$  there is a transition  $((A, a^{-1}, x), (A, a^{-1}x))$ ; that is, there is an edge in the transition diagram from  $A$  to  $A$  labelled  $(a^{-1}, x, a^{-1}x)$ ,  
(c) there are two transitions  $((A, a, a^{-1}), (A, \varepsilon))$  and  $((A, a^{-1}, a), (A, \varepsilon))$ ; that is, there are two edges in the transition diagram from  $A$  to  $A$  labelled  $(a, a^{-1}, \varepsilon)$  and  $(a^{-1}, a, \varepsilon)$  respectively, and  
(d) for every  $x \in \{b, b^{-1}\}$  and every  $y \in \{a, a^{-1}, \perp\}$  there is a transition  $((A, x, y), (A, y))$ ; that is, there is an edge in the transition diagram from  $A$  to  $A$  labelled  $(x, y, y)$ .
- (T3) (a) For every  $x \in \{b, \perp\}$  there is a transition  $((B, b, x), (B, bx))$ ; that is, there is an edge in the transition diagram from  $B$  to  $B$  labelled  $(b, x, bx)$ ,  
(b) for every  $x \in \{b^{-1}, \perp\}$  there is a transition  $((B, b^{-1}, x), (B, b^{-1}x))$ ; that is, there is an edge in the transition diagram from  $B$  to  $B$  labelled  $(b^{-1}, x, b^{-1}x)$ ,  
(c) there are two transitions  $((B, b, b^{-1}), (B, \varepsilon))$  and  $((B, b^{-1}, b), (B, \varepsilon))$ ; that is, there are two edges in the transition diagram from  $B$  to  $B$  labelled  $(b, b^{-1}, \varepsilon)$  and  $(b^{-1}, b, \varepsilon)$  respectively, and  
(d) for every  $x \in \{a, a^{-1}\}$  and every  $y \in \{b, b^{-1}, \perp\}$  there is a transition  $((B, x, y), (B, y))$ ; that is, there is an edge in the transition diagram from  $B$  to  $B$  labelled  $(x, y, y)$ .



- (T4) (a) For every  $y \in \chi \cup \{\perp\}$  there is a transition  $((A, \varepsilon, y), (C, y))$ ; that is, there is an edge in the transition diagram from  $A$  to  $C$  labelled  $(\varepsilon, y, y)$ , and
- (b) for every  $y \in \chi \cup \{\perp\}$  there is a transition  $((B, \varepsilon, y), (C, y))$ ; that is, there is an edge in the transition diagram from  $B$  to  $C$  labelled  $(\varepsilon, y, y)$ .
- (T5) For every  $y \in \chi$  (i.e., when  $y \neq \perp$ ) there is a transition  $((C, \varepsilon, y), (q_A, y))$ ; that is, there is an edge from  $C$  to  $q_A$  labelled  $(\varepsilon, y, y)$ .

We represent the above automaton by the following diagram. The reader should note however that for compactness we have used a single edge in the diagram to represent multiple transitions when needed.



**Intuitive Description:** Intuitively, we think of a run in this automaton as follows. First the automaton non-deterministically moves from the initial configuration to  $(A, \perp)$  or  $(B, \perp)$  through transitions in 1. At state  $A$  (or  $B$ ), the automaton

processes the input letters through the transitions in 2 (or 3). The states  $A$  and  $B$  serve to freely reduce the projection of the input string onto one of the factors of  $\mathbb{Z}^2$ . After the final letter has been read, the automaton uses the transitions in 4 to move from configuration  $(A, S)$  (or  $(B, S')$ ) for some stack  $S$  (or  $S'$ ) to  $(C, S)$  (or  $(C, S')$ ). If the stack is non-empty then the state of automaton is moved to  $q_A$  and the input string is accepted.

**Lemma 2.3.15.** *Let  $\omega = x_1x_2 \cdots x_n \in X^*$ . Then  $\omega$  can only be read by  $\mathcal{A}$  through chains of the following form only*

$$C_0 \sim_\varepsilon C_1^A \sim_{x_1} C_2^A \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^A \sim_\varepsilon^* C_1$$

or

$$C_0 \sim_\varepsilon C_1^B \sim_{x_1} C_2^B \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^B \sim_\varepsilon^* C_2$$

where  $C_0 = (i_0, \perp)$  and the states of  $C_i^A$  and  $C_i^B$  are  $A$  and  $B$  respectively for  $1 \leq i \leq n + 1$ .

*Proof.* First, we note that the only reading transitions are those in  $(T2)$  and  $(T3)$ . Further we note there is no sequence of  $\varepsilon$ -transitions from  $C$  to  $A$  or  $B$ . Therefore transitions in  $(T4)$  and  $(T5)$  can only be used after every input letter has been read. Moreover, we observe that there are no transitions (or sequence of transitions) from  $A$  to  $B$ , or vice versa. Finally the only transitions from  $i_0$  to  $A$  (or  $B$ ) are those in  $(T1)$ .

Thus an input string  $\omega = x_1x_2 \cdots x_n$  must be read by  $\mathcal{A}$  as follows. First a transition from  $(T1)$  is used to move the state to either  $A$  or  $B$ , thus

$$C_0 \sim_\varepsilon C_1^A \tag{5.1a}$$

or

$$C_0 \sim_\varepsilon C_1^B \tag{5.2a}$$

where  $C_0$  is the initial configuration  $(i_0, \perp)$  and the states of  $C_1^A$  and  $C_1^B$  are  $A$  and  $B$  respectively.

Now the automaton uses transitions in  $(T2)$  to continue the chain (5.1a) as in (5.1b) below since there are no other reading transitions from  $A$  apart from those in  $(T2)$ . Similarly, the automaton uses transitions in  $(T3)$  to continue the chain (5.2a) as in (5.2b) below since there are no other reading transitions from  $B$  apart from those in  $(T3)$ . Therefore the chains continue as

$$C_0 \sim_\varepsilon C_1^A \sim_{x_1} C_2^A \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^A \tag{5.1b}$$

and

$$C_0 \sim_\varepsilon C_1^B \sim_{x_1} C_2^B \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^B \tag{5.2b}$$

respectively. Due to the transitions used being those in (T2) and (T3) respectively, the states of  $C_i^A$  and  $C_i^B$  are  $A$  and  $B$  respectively for  $1 \leq i \leq n+1$ . Finally the automaton may use transitions in (T4) and (T5) thereby the chains continue

$$C_0 \sim_\varepsilon C_1^A \sim_{x_1} C_2^A \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^A \sim_\varepsilon^* C_1$$

and

$$C_0 \sim_\varepsilon C_1^B \sim_{x_1} C_2^B \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^B \sim_\varepsilon^* C_2.$$

■

Now we prove another lemma that will provide insight about what the stack is at a configuration.

**Lemma 2.3.16.** *Let  $\omega = x_1x_2 \cdots x_n \in X^*$ . If  $\mathcal{A}$  reads  $\omega$  through the chain*

$$C_0 \sim_\varepsilon C_1^S \sim_{x_1} C_2^S \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^S \sim_\varepsilon^* C$$

where  $S \in \{A, B\}$  as in Lemma 2.3.15. Then the second coordinate of  $C_i^S$  is  $\gamma_{\{s, s^{-1}\}}(x_1x_2 \cdots x_{i-1})\perp$  where  $s = a$  if  $S = A$ , and  $s = b$  if  $S = B$ .

*Proof.* Let  $\omega = x_1x_2 \cdots x_n \in X^*$  and by Lemma 2.3.15 the automaton reads  $\omega$  through a chain of the form

$$C_0 \sim_\varepsilon C_1^S \sim_{x_1} C_2^S \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^S \sim_\varepsilon^* C$$

where  $S \in \{A, B\}$ . We note that if  $S = A$  then the automaton must use transitions in (T2) for the chain of relations

$$C_0 \sim_\varepsilon C_1^S \sim_{x_1} C_2^S \sim_{x_2} \cdots \sim_{x_n} C_{n+1}^S$$

and if  $S = B$  then the automaton must use transitions in (T3). Note that since (T2) and (T3) are analogous we shall assume  $S = A$ .

Note that  $C_0 \sim_\varepsilon C_1^A$  by the transition in (T1) that moves from  $i_0$  to  $A$ , and thus the second coordinate of  $C_1^A$  is  $\perp$ . If  $\omega \neq \varepsilon$  the automaton uses (T2) to read the first letter  $x_1$  and there are 2 choices for what may happen:

1. either  $x_1 \notin \{a, a^{-1}\}$  then the automaton must use (T2)(d) as (T2)(a)-(c) require the input letter to be in  $\{a, a^{-1}\}$ . Thus the transition leaves the stack empty. Thus the second coordinate of  $C_2^A$  is  $\perp = \gamma_{\{a, a^{-1}\}}(x_1)\perp$ .
2. Otherwise  $x_1 \in \{a, a^{-1}\}$ , then the automaton writes  $x_1$  on the stack (via (T2) (a) or (b)). (The choice is dependent on whether the input letter is  $a$  or  $a^{-1}$  as per the definition of (T2).) Thus the second coordinate of  $C_2^A$  is  $x_1\perp = \gamma_{\{a, a^{-1}\}}(x_1)\perp$ .

We shall now proceed by induction. Suppose the second coordinate of  $C_i^A$  is  $\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1})\perp$ . Now we consider the transition by which  $C_i^A \sim_{x_i} C_{i+1}^A$ . There are four choices for what may happen.

1. If  $x_i \notin \{a, a^{-1}\}$  then the automaton uses a transition in (T2)(d) and the stack is unchanged. Therefore the second coordinate of  $C_{i+1}^A$  is

$$\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1})\perp = \gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1}x_i)\perp.$$

For the remainder of the cases we assume  $x_i \in \{a, a^{-1}\}$ .

2. If  $\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1}) = \varepsilon$  then the automaton must the transition

$$((A, x_i, \perp), (A, x_i\perp))$$

(in (T2)(a) or (b)) as there is no other choice of transition, and writes  $x_i$  onto the stack. Thus the second coordinate of  $C_{i+1}^A$  is  $x_i\perp = \gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1}x_i)\perp$ .

3. If the first letter of  $\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1})$  is  $x_i^{-1}$  then the automaton must the transition  $((A, x_i, x_i^{-1}), (A, \varepsilon))$  (in (T2)(c)) as there is no other choice of transition available, and deletes the letter at the top of the stack. Thus the second coordinate of  $C_{i+1}^A$  is  $\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1}x_i)\perp$ .

4. Otherwise, the first letter of  $\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1})$  is  $x_i$  then the automaton uses the transition  $((A, x_i, x_i), (A, x_ix_i))$  as there is no other choice of transition, and writes the letter  $x_i$  onto the top of the stack. Thus the second coordinate of  $C_{i+1}^A$  is  $\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1}x_i)\perp$ .

Therefore the second coordinate of  $C_{i+1}^A$  is  $\gamma_{\{a,a^{-1}\}}(x_1x_2\cdots x_{i-1}x_i)\perp$  as required. ■

We also record a final observation below. This follows immediately from the the definition of  $\mathcal{A}$ .

**Corollary 2.3.17.** *The only accept state is  $q_A$ . Further, a string  $\omega$  can only be accepted if the last configuration in a chain such as in Lemma 2.3.15 is of the form  $(q_A, S\perp)$  where  $S \in \chi^+$ .*

*Proof.* From the definition of the automaton it follows that  $q_A$  is the only accept state. Further, the only transitions to  $q_A$  are those in (T5). By Lemma 2.3.15 we know that a string  $\omega$  can only be read by the chains in the statement of Lemma 2.3.15. Therefore if  $\omega$  is to be accepted then it must be that for at least one chain in Lemma 2.3.15, the last configuration is an accept configuration. By definition, an accept configuration is one whose state is an accept state. Thus it must be that the state is  $q_A$ . However, as the only transitions to  $q_A$  are those in (T5), it must be that the preceding configuration in the chain did not have an empty stack. Otherwise, (T5) would not have been used. ■

Now we are ready to present the proof of Proposition [2.3.14](#)

*Proof of Proposition [2.3.14](#).* Let  $\omega = \sigma_1\sigma_2\cdots\sigma_n \in \{a, a^{-1}, b, b^{-1}\}^*$ . Set  $A' := \{a, a^{-1}\}$  and  $B' := \{b, b^{-1}\}$ . By Lemma [2.3.15](#) and Lemma [2.3.16](#) the automaton reads  $\omega$  through the following chains only

$$\begin{aligned}
(i_0, \perp) &\sim_\varepsilon (A, \perp) \sim_{\sigma_1} (A, \gamma_{A'}(\sigma_1)\perp) \\
&\sim_{\sigma_2} (A, \gamma_{A'}(\sigma_1\sigma_2)\perp) \\
&\vdots \\
&\sim_{\sigma_n} (A, \gamma_{A'}(\sigma_1\sigma_2\cdots\sigma_n)\perp) \\
&\sim_\varepsilon (C, \gamma_{A'}(\sigma_1\sigma_2\cdots\sigma_n)\perp)
\end{aligned} \tag{6}$$

and

$$\begin{aligned}
(i_0, \perp), &\sim_\varepsilon (B, \perp) \sim_{\sigma_1} (B, \gamma_{B'}(\sigma_1)\perp) \\
&\sim_{\sigma_2} (B, \gamma_{B'}(\sigma_1\sigma_2)\perp) \\
&\vdots \\
&\sim_{\sigma_n} (B, \gamma_{B'}(\sigma_1\sigma_2\cdots\sigma_n)\perp) \\
&\sim_\varepsilon (C, \gamma_{B'}(\sigma_1\sigma_2\cdots\sigma_n)\perp).
\end{aligned} \tag{7}$$

Denote the last configurations in (6) and (7) by  $C_{A',\omega}$  and  $C_{B',\omega}$  respectively. Observe that  $\omega \neq_{\mathbb{Z}^2} 1$  if and only if

$$\gamma_{A'}(\sigma_1\sigma_2\cdots\sigma_n) \neq \varepsilon$$

or

$$\gamma_{B'}(\sigma_1\sigma_2\cdots\sigma_n) \neq \varepsilon.$$

Therefore  $\omega \neq_{\mathbb{Z}^2} 1$  if and only if

$$C_{A',\omega} \neq (A, \perp)$$

or

$$C_{B',\omega} \neq (B, \perp).$$

Suppose, without loss of generality,  $C_{A',\omega} \neq (A, \perp)$ . Then the chain in (5) continues as

$$C' \sim_\varepsilon^* (q_A, \gamma_{A'}(\sigma_1\sigma_2\cdots\sigma_n)\perp)$$

and so  $\omega$  is accepted by Corollary [2.3.17](#)

If  $\omega =_{\mathbb{Z}^n} 1_{\mathbb{Z}^n}$  then

$$\gamma_{A'}(\omega') = \varepsilon$$

and

$$\gamma_{B'}(\omega') = \varepsilon.$$

Thus  $C_{A',\omega} = (C, \perp)$  and  $C_{B',\omega} = (C, \perp)$  and the automaton cannot use (T5). So by Corollary 2.3.17,  $\omega$  cannot be accepted. Therefore  $\mathcal{L}(\mathcal{A}) = \text{coWP}(\mathbb{Z}^2, X)$ . ■

Now we provide some examples.

**Example 2.3.18.** We shall work through how  $\mathcal{A}$  reads the following strings.

1.  $\omega_1 := aba^{-1}$ : First note that  $\omega_1 \in \mathbb{Z}^2, X$ . As we have discussed there are two ways to read a string in  $\mathcal{A}$ , one using the state  $A$  and the other using the state  $B$ . Using  $A$ , we read through the following chain

$$\begin{aligned} (i_0, \perp) &\sim_{\varepsilon}^* (A, \perp) \sim_a (A, a\perp) \\ &\sim_b (A, a\perp) \\ &\sim_{a^{-1}} (A, \perp) \sim_{\varepsilon}^* (C, \perp) \end{aligned}$$

and  $\omega_1$  is not accepted.

However using  $B$ , we read through the following chain

$$\begin{aligned} (i_0, \perp) &\sim_{\varepsilon}^* (B, \perp) \sim_a (B, \perp) \\ &\sim_b (b, b\perp) \\ &\sim_{a^{-1}} (B, b\perp) \sim_{\varepsilon}^* (q_A, b\perp) \end{aligned}$$

and  $\omega_1$  is accepted.

2.  $\omega_2 := ab^{-1}ba^{-1}$  : Using  $A$ , we read through the following chain

$$\begin{aligned} (i_0, \perp) &\sim_{\varepsilon}^* (A, \perp) \sim_a (A, a\perp) \\ &\sim_{b^{-1}} (A, a\perp) \\ &\sim_b (A, a\perp) \\ &\sim_{a^{-1}} (A, \perp) \sim_{\varepsilon}^* (C, \perp) \end{aligned}$$

and  $\omega_2$  is not accepted. Similarly, using  $B$  we read through the following chain

$$\begin{aligned} (i_0, \perp) &\sim_{\varepsilon}^* (B, \perp) \sim_a (B, \perp) \\ &\sim_{b^{-1}} (B, b^{-1}\perp) \\ &\sim_b (B, \perp) \\ &\sim_{a^{-1}} (B, \perp) \sim_{\varepsilon}^* (C, \perp) \end{aligned}$$

and  $\omega_2$  is not accepted.

Due to [27], we also have the following results concerning some decision problems of  $co\mathcal{CF}$  groups.

**Proposition 2.3.19.** [27] *Let  $G$  be a  $co\mathcal{CF}$  group.*

- *The word problem of  $G$  is solvable in cubic time in terms of the length of the input.*
- *The order problem of  $G$  is solvable. Moreover, if a string in the generators of  $G$  represents an element of finite order, then its order can be determined.*
- *There exist  $co\mathcal{CF}$  groups with unsolvable conjugacy problem and the generalised word problem is, in general, unsolvable for  $co\mathcal{CF}$  groups.*

The study of R. Thompson's Group  $V$  (originally introduced by R. Thompson in the 1960's but first appears in Thompson's handwritten notes [41] circulated circa 1970) has become central to the study of  $co\mathcal{CF}$  groups. We shall give a brief introduction to Thompson's Group  $V$  below.

### 2.3.3.1 Thompson's Group $V$

In this section we shall define Thompson's group  $V$ . For further background on Thompson's groups in general, we recommend the work of Cannon, Floyd and Parry [15].

We start by defining Cantor space.

**Definition 2.3.20** (Cantor Space). The  $n$ -ary Cantor space, denoted  $\mathfrak{C}_n$ , is defined to be  $\{0, 1, \dots, n-1\}^\omega$  with the product topology induced by equipping  $\{0, 1, \dots, n-1\}$  with the discrete topology. An element  $z \in \mathfrak{C}_n$  is viewed as a infinite sequence  $z = x_1x_2x_3 \cdots$  such that for all  $i \in \mathbb{N}, x_i \in \{0, 1, \dots, n-1\}$ . We will refer to elements of  $\mathfrak{C}_n$  as *points in  $\mathfrak{C}_n$* . ♣

We note that Tychonoff's theorem proves that  $\mathfrak{C}_n$  is a compact space and as a product of Hausdorff spaces, it is also Hausdorff.

In order to make our definition of Thompson's group  $V$ , we first need to recall the prefix relation.

**Definition 2.3.21** (Prefixes). Let  $X$  be a finite set. Let  $w_1, w_2 \in X^*$  we say that  $w_1$  is a prefix of  $w_2$ , denoted  $w_1 \leq w_2$ , if and only if there exists a finite string (possibly the empty string)  $v \in X^*$  such that  $w_2 = w_1v$ .

Similarly a finite string  $w \in X^*$  is a prefix of an infinite string  $z \in X^\omega$ , denoted  $w \leq z$  if and only if there exists an infinite string  $y \in X^\omega$  such that  $z = wy$ .

Let  $a, b \in X^*$ , if neither  $a$  is a prefix of  $b$  nor  $b$  a prefix of  $a$ , we say that  $a$  and  $b$  are *incomparable* and we write  $a \perp b$ . ♣

**Definition 2.3.22.** (Antichain) Let  $X$  be a finite set and let  $A \subseteq X^*$ . We say  $A$  is an *antichain* if for every  $a \in A$ ,  $a$  and  $b$  are incomparable for every  $b \in A \setminus \{a\}$ . That is to say,  $A$  is an antichain if any two distinct elements in  $A$  are incomparable. ♣

We require one more definition before defining Thompson's Group  $V$ .

**Definition 2.3.23.** (Finite Complete Antichain for  $\mathfrak{C}_2$ ) Let  $A \subseteq \{0, 1\}^*$  be a finite set. We say  $A$  is a *finite complete antichain for  $\mathfrak{C}_2$*  if it satisfies the following conditions:

- $A$  is an antichain, and
- for any point  $x \in \mathfrak{C}_2$ , there exists  $a \in A$  such that  $a$  is a prefix of  $x$ .

♣

Equivalently, a finite complete antichain for  $\mathfrak{C}_2$  is a finite set  $A \subseteq \{0, 1\}^*$  such that for every point in  $x \in \mathfrak{C}_2$  there exists a unique prefix  $y$  of  $x$  in  $A$ .

We are now ready to define Thompson's Group  $V$ .

**Definition 2.3.24** (Thompson's Group  $V$ ). Let  $A$  and  $B$  be two finite complete antichains for  $\mathfrak{C}_2$  of equal size. Thus there is a bijection  $\phi : A \rightarrow B$ . This bijection induces a homeomorphism  $\phi^*$  on  $\mathfrak{C}_2$  which is defined below.

For every  $x \in \mathfrak{C}_2$ , there exists a unique prefix  $a \in A$  of  $x$  and  $x_1 \in \mathfrak{C}_2$  of  $x$  such that  $x = ax_1$ . We set  $\phi^*(x) := \phi(a)x_1$ .

Thompson's group  $V$  is the subgroup of  $\text{Homeo}(\mathfrak{C}_2)$  of homeomorphisms induced by bijections between finite complete antichains as above. ♣

Thompson's group  $V$  has become important in understanding the class of  $co\mathcal{CF}$  groups because of the following conjecture (widely known as *Lehnert's Conjecture*) due to the work of Lehnert [31] and, Bleak, Matucci, and Neunhöffer [9].

**Conjecture:** Let  $G$  be a  $co\mathcal{CF}$  group. Then  $G$  embeds into Thompson's group  $V$ .

Further, it is believed that  $co\mathcal{CF}$  groups are not closed under free products. In particular it is conjectured in [27] that  $\mathbb{Z}^2 * \mathbb{Z}$  is not  $co\mathcal{CF}$ . There has been very little progress in proving either statements. Attempting to prove the it is indeed a  $co\mathcal{CF}$  group is difficult as there is not a lot of room for storing extra information that one needs while doing computations in the PDA without needing to look arbitrarily deep into the stack. This of course, is not something that can be done with PDA. The non-determinism in the automaton of  $\mathbb{Z}^2$  plays a major role in complicating the problem. Proving that groups are not  $co\mathcal{CF}$  is in general a difficult problem



as there are not a lot of good tools available in proving negative statements about groups defined by language classes. However, these two questions (whether or not  $\mathbb{Z}^2 * \mathbb{Z}$  is  $co\mathcal{CF}$  and Lehnert's Conjecture) have been linked in a paper by Bleak and Salazar-Diaz [10] as they proved that  $\mathbb{Z}^2 * \mathbb{Z}$  does not embed into  $V$ .

To conclude this section, we would like to note that the full strength of PDA has not yet been utilised in most machines proving groups are  $co\mathcal{CF}$ . Most of the PDA designed so far have non-determinism at the beginning or use results from the theory of PDA enabling constructing PDA with the aforementioned property. This is understandable in that proofs are easier than they theoretically could be if non-determinism appeared in a lot of places in the machine. However this also means that there's some reluctance to subscribe to Lehnert's conjecture, as the full reach of PDA is not well understood. Having non-determinism only in the beginning turns out to be very useful for other kinds of machines as well. However it still means that the limits of these machines in general regarding recognising co-word problems is difficult to understand.

### 2.3.4 co-indexed Groups

In this section, we shall define co-indexed groups and present some of their properties as in [28].

**Definition 2.3.25** (co-indexed Group). [28] Let  $G$  be a finitely generated group,  $G$  is said to be *co-indexed* if there exists a nested-stack automaton that accepts its co-word problem with respect to some (and hence any) symmetrically closed finite generating set. ♣

Similar to Proposition 2.3.12 the class of co-indexed groups is closed under a number of operations.

**Proposition 2.3.26.** [28] *The class of co-indexed groups is closed under:*

- *passing to finitely generated subgroups,*
- *passing to finitely indexed overgroups,*
- *taking (finite) direct products, and*
- *standard restricted wreath product with context-free top group.*

We shall now turn our attention to defining what it means for an automaton to *ignore the word problem*. This will be useful to us in chapters 3 and 4. Using the convention stated in 2.2.3.4 (also see [28]), we may separate the states of a nested stack automaton  $\mathcal{A}$  into reading-and  $\varepsilon$ -states. Thus we have reading-

and  $\varepsilon$ -configurations. Let  $G$  be a co-indexed group with  $\mathcal{A}$  accepting its co-word problem. Let  $C$  be a start configuration and let  $\omega = \sigma_1\sigma_2\cdots\sigma_n$  be a string in the generators of  $G$ . There is a unique reading configuration  $C'$  such that

$$C \sim_{\sigma_1} C_1 \sim_{\varepsilon}^* C_2 \sim_{\sigma_2} C_3 \sim_{\varepsilon}^* C_4 \sim_{\sigma_3} \cdots \sim_{\sigma_n} C_{2n-1} \sim_{\varepsilon}^* C',$$

and we set  $C^\omega := C'$ . We are now ready to state the following definition.

**Definition 2.3.27** (Ignoring the word problem). Let  $G$  be a co-indexed group with  $\mathcal{A}$  accepting its co-word problem. We say  $\mathcal{A}$  *ignores the word problem* of  $G$  if for any reading configuration  $C$  and any string  $w = uv$  in the generators of  $G$  ( $u$  and  $v$  are also strings) such that  $v =_G 1_G$ , then  $C^w = C^u$  unless  $C^w$  is a sink. We note that  $u$  may be the empty string. ♣

We note that *sink* is a standard term in the theory of automata, and simply means a state that no other state can be reached from. For more details on this we recommend [\[28\]](#).

# Chapter 3

## Stack Groups

### 3.1 Introduction

In this chapter we mainly discuss Theorem 11 of [28]. This chapter is an expository chapter for the purpose of completeness as the constructions used in Theorem 11 inspire our own constructions in the proofs of Theorem 4.6.1 and Theorem 4.7.1. However, we shall not prove the theorem in this chapter as the proof can be found in [28]. We will give a detailed discussion of how the automaton involved works and give a brief argument that it does indeed accept the language claimed.

The main takeaway of this chapter is understanding the construction used in the proof of Theorem 11 of [28]. We shall highlight parts of the construction and their implications that are of particular interest to us throughout by using remarks. First we remind the reader of the conventions of [28] below, which we shall also use here.

**Convention:** For every indexed language  $L$ , there exists a nested stack automaton  $\mathcal{A}$  such that  $\mathcal{L}(\mathcal{A}) = L$ , and  $\mathcal{A}$  has the following property. The set of states  $Q$  of  $\mathcal{A}$  can be written as a disjoint union of two sets  $Q_\varepsilon$  and  $Q_\Sigma$  such that

- if a transition is from a state  $p \in Q_\varepsilon$  then it is an  $\varepsilon$ -transition, and
- if a transition is from a state  $q \in Q_\Sigma$  then it is not an  $\varepsilon$ -transition.

We refer to states in  $Q_\varepsilon$  and  $Q_\Sigma$  as  $\varepsilon$ - and *reading*-states respectively. Further let  $C = (q, S, \zeta)$  be a configuration of  $\mathcal{A}$ . We say  $C$  is a *reading-configuration* if  $q \in Q_\Sigma$ . Conversely, we say  $C$  is an  $\varepsilon$ -configuration if  $q \in Q_\varepsilon$ . Further, we will also assume (as in [28]):

- every sequence of consecutive  $\varepsilon$ -transitions must terminate after finitely many transitions.

We are now ready to define *stack groups*.

**Definition 3.1.1** (Stack Groups). Let  $G$  be a co-indexed group and suppose  $X$  is a finite generating set for  $G$ . Let  $\mathcal{A}$  be a nested stack automaton. We say  $(G, \mathcal{A})$  is a *stack pair with respect to  $X$*  if  $\mathcal{A}$  has the following properties:

1.  $\mathcal{L}(\mathcal{A}) = \text{coWP}(G, X)$ ,
2.  $\mathcal{A}$  is deterministic upon input (see Definition [2.2.39](#)),
3. for every start configuration  $C = (q, S, \xi)$ , the stack  $S$  is non-nested, i.e., there is no coordinate in  $S$  with \$ in it, and
4.  $\mathcal{A}$  ignores the word problem (see Definition [2.3.27](#)).

We say  $G$  is a *stack group with respect to  $X$*  if there exists a nested stack automaton  $\mathcal{A}$  such that  $(G, \mathcal{A})$  is a stack pair with respect to  $X$ . ♣

We shall intuitively discuss the above definition. We say a group  $G$  with finite generating set  $X$  is a stack group if  $G$  is a co-indexed group such that there exists a nested stack automaton  $\mathcal{A}$  accepting  $\text{coWP}(G, X)$  satisfying conditions 2, 3, and 4 above. Condition 2 is defined in Definition [2.2.39](#). This means that any non-deterministic move can only occur before input letters are read. Further, once the first input letter is read, there is no choice allowed and the automaton is deterministic from that point onwards. Condition 3 means that upon reading the first input letter the stack must be non-nested (see Definition [2.2.27](#)). Finally we shall now explain condition 4 (see Definition [2.3.27](#)). Suppose the automaton was in a reading configuration  $C$  and read a substring  $v$  of the input string such that  $v =_G 1_G$  from  $C$ . Then the automaton returns to  $C$  after reading  $v$ . This happens unless during the reading of  $v$  the automaton moved to a sink state. In that case, it shall remain in that sink state. Similarly, if the state of  $C$  was a sink state  $q$  then the automaton shall also remain in  $q$  after reading  $v$ .

We note that these conditions are very restrictive and it is unknown whether every co-indexed group admits an automaton accepting its co-word problem that satisfies the conditions of Definition [3.1.1](#). That is, it is unknown whether the class of stack groups is a proper subclass of the class of co-indexed groups.

Unlike the classes of groups defined by languages in [2.3](#), it is not clear a priori whether stack groups are independent of choice of generators. The following result from [28](#) proves that stack groups are independent of choice of generators.

**Proposition 3.1.2.** [28](#) [Proposition 9] *Let  $G$  be group and suppose  $X$  and  $Y$  are finite generating sets for  $G$ . Then  $G$  is a stack group with respect to  $X$  if and only if  $G$  is a stack group with respect to  $Y$ .*

Therefore it is permissible for us to use the phrase *class of stack groups* and we may drop the reference to a specific generating set when discussing stack groups as we do when we use the phrase “the group  $H$  is  $co\mathcal{CF}$ ”. The class of stack groups also shares some closure properties with various other classes of groups defined by languages, such as  $co\mathcal{CF}$  and co-indexed groups.

**Proposition 3.1.3.** [28, Proposition 9] *The class of stack groups is closed under:*

- *passing to finitely generated subgroups,*
- *passing to finite index overgroups,*
- *taking (finite) direct products, and*
- *standard restricted wreath product with context-free top group.*

## 3.2 Technical Lemma due to Holt–Röver

In this section we discuss Lemma 13 in [28]. We do not prove this lemma completely but we do give a sketch of the proof for a part of the lemma as we rely on understanding that part in our context in Theorem 4.7.1. For the other part of the lemma, we explain what the statement of the lemma means and the right frame to view it in. Before doing the above, we shall define some terminology that is used in the lemma. We do this in the following subsection.

### 3.2.1 Definitions

We shall present the following set up which our next two definitions (Definition 3.2.1 and Definition 3.2.2) rely on.

Let  $G$  be a stack group with respect to  $X$ . Thus there exists a nested stack automaton  $\mathcal{A}$  such that  $(G, \mathcal{A})$  is a stack pair with respect to  $X$ . By Definition 3.1.1,  $\mathcal{A}$  is deterministic upon input. Thus for a run

$$C_0 \sim_{\varepsilon}^* \overbrace{C_1 \sim_{x_1} C_2 \sim_{\varepsilon}^* C_3 \sim_{x_2} C_4 \sim_{\varepsilon}^* \cdots \sim_{x_n} C_{2n} \sim_{\varepsilon}^* C_{2n+1}}^{\eta}$$

reading  $x_1 x_2 \cdots x_n$ , the transitions used in  $\eta$  (in the chain above) are uniquely determined given a configuration  $C_1$ . Note that  $C_1$  is the start configuration of the run above. Suppose  $C_1 = (q, S, \xi)$ . By Definition 3.1.1, we know that the choice stack  $S$  is non-nested, i.e., there is no coordinate in  $S$  with  $\$$  in it. Suppose  $S = (\mathbf{t}, s_n, s_{n-1}, \dots, s_1, \perp)$ .

**Definition 3.2.1.** We assume the set up defined above.

1. We call the stack  $S$  *the choice stack* of the run.
2. Given a run such as the one above, if we can obtain the stack  $S_i$  of every configuration  $C_i = (q_i, S_i, \xi_i)$  (for  $i \geq 2$ ) by inserting subsequences of the form  $\mathbf{t}, y_k, y_{k-1}, \dots, y_1, \$$  between components of  $S$  then we say *the choice stack is not altered during the deterministic phase*.



Before we continue, we shall elaborate on the above and give an example. The choice stack is not altered during the deterministic phase simply means that modes available to the machine during the deterministic phase do not alter the trunk of the stack. Therefore, the automaton can use reading, branch creation and branch destruction modes. However, the automaton must only use the pushdown mode when the pointer is in a branch.

Suppose the choice stack of some run of some machine is  $S' = (\mathbf{t}, A, B, A, \perp)$ . Observe that we cannot obtain  $(\mathbf{t}, C, A, B, A, \perp)$  by inserting a sequence of the form  $\mathbf{t}, y_k, y_{k-1}, \dots, y_1, \$$  into  $S'$ . However, by inserting  $\mathbf{t}, a, a, \$$  between  $B$  and the  $A$  to the right of it we can obtain  $(\mathbf{t}, A, B, \mathbf{t}, a, a, \$, A, \perp)$ .

For our next definition, we shall assume the same set up as Definition [3.2.1](#) with the added assumption that the choice stack is not altered during the deterministic phase. We note that if that is the case, then the stack  $S_i$  of any configuration  $C_i = (q_i, S_i, \xi_i)$  is either branched or  $S_i = S$  (for all  $i \geq 2$ ).

**Definition 3.2.2.** We assume the same set up as Definition [3.2.1](#). Further suppose the choice stack is not altered during the deterministic phase. (Then the stack  $S_i$  of any configuration  $C_i = (q_i, S_i, \xi_i)$  is either branched or  $S_i = S$ , for all  $i \geq 2$ .)

1. We may refer to the readable portion of the stack (as in Definition [2.2.28](#)) of a configuration  $C_i$  as the *active stack* of  $C_i$ .
2. Suppose  $S_i$  of a configuration  $C_i$  is branched, for some  $i$ . Let  $(\mathbf{t}, s'_k, \dots, s'_1, \$, s_l, \dots, s_1, \perp)$  be the readable portion of  $S_i$ . If  $s'_1 \neq s_l$  then we say *the active stack of  $C_i$  overlaps as much as possible with the choice stack*.
3. If the active stack of  $C_i$  overlaps as much as possible with the choice stack for every  $i \geq 2$  then we say *the active stack always overlaps as much as possible with the choice stack in that run*. We may also refer to this by similar language.



We note that in [28], the stack does not have the marker  $\mathbf{t}$ , thus in the statement of Lemma 3.2.4 when the phrase *top of the stack* is used, we mean the symbol that is to the right of  $\mathbf{t}$  in the active stack. Further, the pointer is referred to as *read-write-head* or **rw** for short.

We also note that a run is referred to as a *computation*. Further note that strings are accepted in [28] by *giving a description of a set of accept configurations*.

We shall now present the final definition section.

**Definition 3.2.3.** Let  $C = (q, \mathcal{S})$  be a reading configuration. We say *the top of the stack contains a symbol representing the current state* if the top of the active stack contains a symbol representing  $q$ ; i.e, the symbol in the  $r(S) - 1^{\text{th}}$  coordinate represents  $q$ . ♣

For the purposes of this lemma, as stated earlier, we shall give a sketch of the proof of part of the lemma. In that sketch, we use the ideas in [2.2.3] in particular Definition [2.2.32]

### 3.2.2 Lemma

We are now ready to present the lemma.

**Lemma 3.2.4.** [28, Lemma 13] *Let  $G$  be a stack group with respect to a finite generating set  $X$ . Then there exists a nested stack automaton  $\mathcal{A}$  such that  $(G, \mathcal{A})$  is a stack pair with respect to  $X$  and  $\mathcal{A}$  satisfies the following properties.*

- (S4) *In each computation of  $\mathcal{A}$  the choice stack is not altered during the deterministic phase and the active stack always overlaps as much as possible with the choice stack*
- (S5) *In every reading configuration of  $\mathcal{A}$  the **rw** is scanning the top of the stack, which contains a symbol representing the current state.*

As mentioned in [3.2], we shall not provide a proof of the above lemma as it can be found in [28]. By Definition [3.1.1] there exists a nested stack automaton  $\mathcal{A}' = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$  (as defined in Definition [2.2.26]) such that  $(G, \mathcal{A}')$  is a stack pair with respect to  $X$ . The lemma does not claim that  $\mathcal{A}'$  has the properties stated. Instead that there exists, possibly a different nested stack automaton,  $\mathcal{A}$  such that  $(G, \mathcal{A})$  is a stack pair with respect to  $X$  satisfying the properties stated. As stated in [3.2], we provide a sketch of proof for one part of the lemma, namely (S5) as we rely on a similar construction in the automaton used to prove Theorem [4.7.1]. We do so by constructing an automaton by modifying  $\mathcal{A}'$  to satisfy (S5), following the proof in [28].

**Remark 3.2.5.** In the construction of the automaton that we will use to prove Theorem 4.7.1, we will use some transitions after reaching some states in order to write the names of those states on the pushdown stack. This technique is important as it means that we will not need to rely on the states of the automaton to know where a computation has ended and we can simply use the stack. ♠

*Sketch of Proof of 3.2.2 (S5).* By Definition 3.1.1 there exists a nested stack automaton

$$\mathcal{A}' = (Q, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta)$$

such that  $(G, \mathcal{A}')$  is a stack pair with respect to  $X$ .

By the convention in 3.1, the state set  $Q$  of  $\mathcal{A}'$  is a disjoint union of  $\varepsilon$ -states  $Q_\varepsilon$  and reading-states  $Q_\Sigma$ . We shall define a new automaton

$$\mathcal{A}'_1 = (Q_1, \Sigma, \chi, I, F, \$, \mathbf{t}, \perp, \delta_1)$$

such that  $\mathcal{L}(\mathcal{A}'_1) = \mathcal{L}(\mathcal{A}')$  by modifying  $Q_1$  and  $\delta_1$  so that every reading-transition in  $\delta_1$  moves the automaton from a reading-state to an  $\varepsilon$ -state as follows. First we start with  $Q_1 = Q$  and  $\delta_1 = \delta$ , and we refer to the reading-states and  $\varepsilon$ -states of  $Q_1$  as  $Q_{1,\Sigma}$  and  $Q_{1,\varepsilon}$  respectively. For every reading-state  $p \in Q_\Sigma$  such that there exists a transition  $\alpha = ((p, \sigma, y), (q, w)) \in \delta_1$  with  $q \in Q_\Sigma$  we do the following. Since  $\delta_1$  is finite there are finitely many transitions from  $p$ . In particular, there are only finitely many transitions from  $p$  to reading-states. Let  $T_p$  be the set consisting of those transitions in  $\delta_1$ . For every  $\lambda = ((p, \sigma', y'), (q', w')) \in T_p$  we modify  $Q_1$  and  $\delta_1$  as follows:

- adjoin a state  $q_\lambda$  to  $Q_{1,\varepsilon}$ ,
- we delete  $\lambda$  from  $\delta_1$ , and
- we adjoin the following transitions to  $\delta_1$

$$\lambda_1 = ((p, \sigma', y'), (q_\lambda, w')) \text{ and } \lambda_2 = ((q_\lambda, \varepsilon, y''), (q', 0))$$

where  $y''$  is the letter pointed to if  $\lambda$  is used in  $\delta$ .

Observe that  $C_1 \sim_{\sigma'} C_2$  by  $\lambda$  if and only if  $C_1 \sim_{\sigma'} C'$  by  $\lambda_1$  and  $C' \sim_\varepsilon C_2$  by  $\lambda_2$ . Thus whenever we have a chain of relations reading a string  $\omega$  by transitions of  $\mathcal{A}$  with  $C_1 \sim_{\sigma'} C_2$  by  $\lambda$  appearing in the chain, then we can replace that occurrence of

$$C_1 \sim_{\sigma'} C_2$$

by

$$C_1 \sim_{\sigma'} C' \sim_\varepsilon C_2$$



where  $C_1 \sim_{\sigma'} C'$  by  $\lambda_1$  and  $C' \sim_{\varepsilon} C_2$  by  $\lambda_2$  to yield a chain of relations reading a string  $\omega$  by transitions of  $\mathcal{A}'$ , and vice versa. Therefore,  $\mathcal{L}(\mathcal{A}'_1) = \mathcal{L}(\mathcal{A}')$ . Further,  $(G, \mathcal{A}'_1)$  is a stack pair as we have not modified the automaton in anyway that would violate the definition of a stack pair. We shall now modify  $\mathcal{A}'_1$  to satisfy (S5).

We shall define a new automaton  $\mathcal{A}'_2 = (Q_2, \Sigma, \chi_2, I, F, \$, \mathbf{t}, \perp, \delta_2)$  and we start by setting  $Q_2 = Q_1$ ,  $\chi_2 = \chi$ , and  $\delta_2 = \delta_1$ . We start by assuming that  $Q_2 \cap \chi_2 = \emptyset$ . For every  $q \in Q_2$ , we shall adjoin a stack alphabet letter to  $\chi_2$  representing  $q$  and we denote it by  $q$  as well. We shall refer to the reading-states and  $\varepsilon$ -states of  $\mathcal{A}'_2$  as  $Q_{2,\Sigma}$  and  $Q_{2,\varepsilon}$  respectively.

To ensure that (S5) is satisfied we shall modify  $\delta_2$  (and  $Q_2$ ) further. There are two cases:

1. transitions that move the automaton to a reading-state, and
2. transitions that move the automaton from a reading-state.

We address these below.

1. For each transition  $\mu = ((q_1, \varepsilon, y_1), (q_2, w_1)) \in \delta_1$  where  $q_2 \in Q_{1,\Sigma}$  we do the following modifications:
  - we delete  $\mu$  from  $\delta_2$ ,
  - we adjoin a new state  $q_\mu$  to  $Q_{2,\varepsilon}$ , and
  - we add the following transitions to  $\delta_2$

$$\begin{aligned} \mu_1 &= ((q_1, \varepsilon, y_1), (q_\mu, w_1)), \\ \mu_{\mathbf{t}} &= ((q_\mu, \varepsilon, \mathbf{t}y_2), (q_2, \mathbf{t}q_2y_2)), \text{ and} \\ \mu_{y_1} &= (q_\mu, \varepsilon, y_2), (q_2, \mathbf{t}q_2y_2\$) \end{aligned}$$

where  $y_1, y_2 \in \chi \cup \{\perp\}$ .

Observe that having done these modifications, for every reading configuration whose state is  $q \in Q_{2,\Sigma}$ , the **rw**h is at the top of the stack and is pointing to  $q$ .

2. For each transition  $\nu = ((q'_1, \sigma, y'_1), (q'_2, w'_1)) \in \delta_1$  where  $q'_1 \in Q_{1,\Sigma}$  we do the following modifications:
  - we delete  $\nu$  from  $\delta_2$ ,
  - we adjoin new states  $q_{\nu,1}$  and  $q_{\nu,2}$  to  $Q_{2,\varepsilon}$ , and

- we add the following transitions to  $\delta_2$

$$\begin{aligned} &((q_1, \sigma, \mathbf{t}q_1), (q_{\nu,1}, \mathbf{t})), \\ &((q_{\nu,1}, \varepsilon, \mathbf{t}\$), (q_{\nu,2}, \varepsilon)), \\ &((q_{\nu,1}, \varepsilon, \mathbf{t}), (q_{\nu,2}, -1)), \text{ and} \\ &((q_{\nu,2}, \varepsilon, y'_1), (q'_2, w'_1)). \end{aligned}$$

Observe that having done these modifications, from reading-states we first read an input letter and delete the top of the stack. If the stack is branched, we delete the symbol underneath  $\mathbf{t}$  if the result is an empty branch we delete the branch. If the stack is not branched then we simply delete the symbol under  $\mathbf{t}$  and move the pointer down by one move. Then the pointer will point to  $y'_1$ . Then we move the state of the automaton to  $q'_2$  and modify the stack the same way  $\nu$  does.

Observe that these modifications insert auxiliary subchains of configurations in a chain of configurations without changing the last configuration in a meaningful way and thus they do not change the language accepted, so  $\mathcal{L}(\mathcal{A}'_2) = \mathcal{L}(\mathcal{A}'_1)$ . ■

We shall now discuss (S4). Recall we will not prove that (S4) can be satisfied, but we will briefly discuss the main points of how to satisfy (S4). First we define a new automaton  $\mathcal{A}'_3 = (Q_3, \Sigma, \chi_3, I, F, \$, \mathbf{t}, \perp, \delta_3)$  and we start by setting  $Q_3 = Q_2$  and  $\delta_3 = \delta_2$ . Further we define a set  $\overline{\chi_2} := \{\bar{z} \mid z \in \chi_2\}$  consisting of barred versions of symbols for every symbol in  $\chi_2$ . More formally, let  $\chi'$  be a set disjoint from  $\chi_2$  and that there exists a bijection  $\theta : \chi_2 \rightarrow \chi'$ . We denote the image  $\theta(x)$  by  $\bar{x}$  for every  $x \in \chi_2$  and we say  $\bar{x}$  is a *barred version* of  $x$ . We further denote  $\chi'$  by  $\overline{\chi_2}$ . Now we set  $\chi_3 = \chi_2 \cup \overline{\chi_2}$ .

We call a state  $q$  a *start state* if it is the state of a start configuration. Let  $Q_N$  be the subset of  $Q_{2,\varepsilon}$  with the following property. For every  $q \in Q_N$  there exists a configuration  $C_q$  where  $q$  is the state of  $C_q$  such that there exists a start configuration  $C$  where

$$(q_0, (\mathbf{t}, \perp), (1, \mathbf{t})) \sim_{\varepsilon}^* C_q \sim_{\varepsilon}^* C$$

for every  $q_0 \in I$ . The states in  $Q_N$  represent the states that are involved in the transitions before the first reading-transition occurs from a start configuration, i.e., the states in  $Q_N$  are those used in the non-deterministic phase of  $\mathcal{A}'_3$ . Observe that  $I \subseteq Q_N$ .

We modify the automaton so that the stack alphabet used in the non-deterministic phase of  $\mathcal{A}'_3$  are barred. That is, the transitions used in a run from the initial configuration to the start configuration, have the same impact on the stack as the non-deterministic phase of  $\mathcal{A}'_2$ , but the stack alphabet used are barred versions of

those used in  $\mathcal{A}_2$ . Thus let  $q$  be a start state of  $\mathcal{A}'_2$ . Then  $(q, S, (k+2, \mathbf{t}))$  where  $S = (\mathbf{t}, z_1, z_2, \dots, z_k, \perp)$  is a start configuration of  $\mathcal{A}'_2$  if and only if  $(q, \overline{S}, (k+2, \mathbf{t}))$  where  $\overline{S} = (\mathbf{t}, \overline{z_1}, \overline{z_2}, \dots, \overline{z_k}, \perp)$  is a start configuration of  $\mathcal{A}'_3$ .

Set  $Q_D := Q_2 \setminus Q_N$ . The states in  $Q_D$  are those used in transitions from reading the first input letter onwards, i.e.,  $Q_D$  consists of the states involved in the deterministic phase of  $\mathcal{A}'_3$ . In particular, start states are in  $Q_D$ . One can modify the transitions involving  $Q_D$  such that the following conditions are satisfied.

1. The alphabet written or deleted with these transitions are unbarred.
2. If the pointer is pointing to a barred letter  $\overline{y}$  on the stack and the transition does not write nor delete then the transition, i.e., the transition belongs to  $\mathcal{J}_2$  in Definition [2.2.26](#) then the transition has the same effect on the stack as it would if the pointer was pointing to  $y$ .
3. Suppose the pointer is at  $z$  where  $z$  is either barred or the bottom of stack symbol. Further, suppose the transition is creating a branch, consisting of  $tx_1x_2 \cdots x_r$  say. Then the pointer moves up by 1 step for every letter of  $x_1x_2 \cdots x_r$  starting from the right. This happens until the letter  $x_i$  in  $x_1x_2 \cdots x_r$  is not the unbarred version of what the pointer is pointing to. Then the pointer is moved down by 1 step again. Finally the automaton branches from the trunk with the branch consisting of  $\mathbf{t}x_1x_2 \cdots x_i$ . Thus we ensure that the active stack overlaps as much as possible with the choice stack.
4. Conversely, when deleting top of the active stack, we replace transitions that delete the top of the branch by transitions that move the pointer down by one step if the letter being deleted is barred. Thus the pointer will reach the same position it would if the deletion occurred but while satisfying 1 as well.

**Remark 3.2.6.** The alphabet used in non-deterministic phase are disjoint from the alphabet used in the deterministic phase. The trunk will not be edited after the non-deterministic phase, but a part of it is viewable by the pointer during the deterministic phase. We use this in the construction of the automata in Theorem [4.6.1](#) and Theorem [4.7.1](#). The first point is easy to do as shall define a disjoint of alphabet for the check-stack alphabet that we use in a similar way; i.e., to inform decisions made during the computation. The second point is more complicated in a CSPD machine. This is because, in a NSA the pointer can move up and down without editing the stack unlike in a CSPD automaton. We shall use the pushdown stack alphabet, the states, and the pointer in a CSPD automaton to achieve a similar effect. We do this by adding onto the pushdown stack some letters ( $W$  and  $M$  representing “wait” and “move”) in order to be able to move

up on the pushdown stack and thus be able to see more of the check-stack. Conversely, deleting these letters will result in viewing letters on the check-stack that we have already been able to see at an earlier point. ♠

When discussing Theorem 11 of [28] in the following section, we shall assume that the automata satisfy (S4) and (S5).

### 3.3 Free Products

In this section we shall explore Theorem 11 of [28]. Our goal is to elaborate on the proof and familiarise the reader with ideas and the key points. First, we state the theorem below. Then we shall give intuitive outline of the algorithm. We then follow with a detailed discussion of how the automaton works. We end with a brief argument that the automaton does indeed accept the language claimed.

**Theorem 3.3.1.** [28, Theorem 11] *Let  $G_1$  and  $G_2$  be stack groups. Then the free product  $G_1 * G_2$  is also a stack group.*

#### 3.3.1 Outline

To prove  $G_1 * G_2$  is a stack group we must show that there exists a nested stack automaton  $\mathcal{A}$  such that  $(G_1 * G_2, \mathcal{A})$  is a stack pair with respect to some generating set for  $G_1 * G_2$ .

Suppose  $X_1$  and  $X_2$  are finite generating sets for  $G_1$  and  $G_2$  respectively. Recall since  $G_1$  and  $G_2$  are stack groups then there exist nested stack automata  $\mathcal{A}_1$  and  $\mathcal{A}_2$  such that  $(G_1, \mathcal{A}_1)$  and  $(G_2, \mathcal{A}_2)$  are stack pairs with respect to  $X_1$  and  $X_2$  respectively. Further, we may assume that  $\mathcal{A}_1$  and  $\mathcal{A}_2$  satisfy the conditions in Lemma 3.2.4. Recall in the previous section, the automata have two phases *a non-deterministic phase* and *a deterministic phase*.

We shall first describe the non-deterministic phase of  $\mathcal{A}$ . The automaton first runs through the non-deterministic phase of  $\mathcal{A}_1$  followed by the non-deterministic phase of  $\mathcal{A}_2$ , and the automaton repeats this process some number of times. Then the automaton uses an  $\varepsilon$ -transition to move to the start state  $q_r$  while pushing the symbol  $\bar{q}_r$  onto the stack. From this point on wards,  $\mathcal{A}$  will be deterministic. Recall by our discussion in 3.2.2, the letters used in non-deterministic phase of  $\mathcal{A}$  will be barred. After the automaton runs through the non-deterministic phase of  $\mathcal{A}_1$  the letter under  $\mathbf{t}$  is from the stack alphabet of  $\mathcal{A}_1$ . This letter will be treated the same way  $\perp$  is treated by  $\mathcal{A}_2$ . Similarly, at the end of the non-deterministic phase of  $\mathcal{A}_2$  the letter under  $\mathbf{t}$  is from the stack alphabet of  $\mathcal{A}_2$ . This letter will be treated the same way  $\perp$  is treated by  $\mathcal{A}_1$ .

After each time the automaton runs a non-deterministic phase of  $\mathcal{A}_i$  the symbol under  $\mathbf{t}$  is a symbol representing a start state of  $\mathcal{A}_i$ . Thus the stack after running the non-deterministic phases of  $\mathcal{A}_1$  and  $\mathcal{A}_2$  repeatedly in an alternating manner, the stack is

$$(\mathbf{t}, NP(\mathcal{A}_2)_m, NP(\mathcal{A}_1)_m \dots, NP(\mathcal{A}_2)_2, NP(\mathcal{A}_1)_2, NP(\mathcal{A}_2)_1, NP(\mathcal{A}_1)_1, \perp)$$

where  $NP(\mathcal{A}_i)_j$  denotes a sequence of barred stack alphabet (i.e., excluding  $\mathbf{t}$  or  $\perp$ ) written on the stack during the non-deterministic phase of  $\mathcal{A}_i$  when run for the  $j^{\text{th}}$  time by  $\mathcal{A}$ . Note that the first letter of every  $NP(\mathcal{A}_i)_j$  must represent a start state of  $\mathcal{A}_i$  by Lemma 3.2.4. Moreover, there are no branches in any  $NP(\mathcal{A}_i)_j$  and thus there are no branches in the stack above. Finally, the last step in the non-deterministic phase is moving to the start state  $q_r$ , pushing a barred symbol  $\bar{q}_r$  representing that on top of the stack, and moving the pointer down by one position. So the stack at the end of the non-deterministic phase is

$$\mathcal{S} = (\mathbf{t}, \bar{q}_r, NP(\mathcal{A}_2)_m, NP(\mathcal{A}_1)_m \dots, NP(\mathcal{A}_2)_2, NP(\mathcal{A}_1)_2, NP(\mathcal{A}_2)_1, NP(\mathcal{A}_1)_1, \perp).$$

Finally, we note that the pointer at the end of the non-deterministic phase is at  $\bar{q}_r$ .

Now we describe the deterministic phase of  $\mathcal{A}$ . During this phase the automaton follows one “meta-rule”, which we state below and elaborate on afterwards.

**Meta-Rule:** In a given run, suppose the input letter  $x$  being read belongs to  $X_i$ . Further suppose that the last input letter from  $X_i$  left the automaton at configuration  $C$ . If the automaton can use a reading-transition with input letter  $x$  from  $C$ , then it does so. Thus  $x$  is processed from  $C$ . Otherwise, the automaton uses the pointer to move through the stack to find the first state  $q$  on the stack such that  $q \in \chi_i$  and thereby the configuration is now one an input letter from  $X_i$  can be processed from.

Now we shall elaborate on our “meta-rule”. Suppose the first letter of the input string is  $x \in X_i$ . The automaton moves the pointer from  $\bar{q}_r$  down until the first letter representing a state in  $\mathcal{A}_i$  is seen on the stack. In the above stack, this is either the first letter of  $NP(\mathcal{A}_2)_m$  or that of  $NP(\mathcal{A}_1)_m$ . Suppose without loss of generality, that the first letter on the stack representing a state of  $\mathcal{A}_i$  is the first letter of  $NP(\mathcal{A}_1)_m$ , say  $\bar{q}$ . Note this implies that  $x \in X_1$ . The automaton lets  $\mathcal{A}_1$  run from the reading state  $q$  reading  $x$  with  $NP(\mathcal{A}_1)_m$  on the stack and the pointer at  $\bar{q}$ . Recall that by Definition 3.1.1  $\mathcal{A}_1$  is deterministic upon input. We follow all the transitions available until we reach a reading configuration, this yields the following chain of relations in  $\mathcal{A}_1$

$$C \sim_x C_1 \sim_\varepsilon^* C_2$$

such that  $C = (q, (\mathbf{t}, NP(\mathcal{A}_1), \perp), \xi)$  with  $\xi = (n-1, \bar{q})$  where  $n$  is the length of the stack, and  $C_1$  and  $C_2$  are the unique configurations determined by the transitions of  $\mathcal{A}_1$  such that  $C_2$  is a reading-configuration. By Lemma 3.2.4, we know that the choice stack is not altered in  $\mathcal{A}_1$  and the active stack overlaps as much as possible with the choice stack. Suppose  $C_2 = (q_1, S_1, \xi_1)$  where the pointer points to a symbol representing  $q_1$  on the stack  $S_1$ . The automaton  $\mathcal{A}$  will simulate the transitions used in above chain of relations with the addition of returning to state  $q_r$  at the end of simulating transitions in  $\mathcal{A}_1$ . The automaton also determines whether or not a branch exists, if a branch was created in the chain of relations, then the symbol  $q_r$  gets pushed onto the branch. If there was no branch that was created, then the automaton simply moves the pointer up the stack until it sees the symbol  $\bar{q}_r$ . All of these transitions (simulating  $\mathcal{A}_1$  and the pointer pointing to a state representing  $q_r$ ) yield the following chain

$$(q_r, \mathcal{S}, \xi_0) \sim_x C'_1 \sim_\varepsilon^* C'_2$$

where  $C'_2 = (q_r, \mathcal{S}', \xi')$  where  $\mathcal{S}'$  is the stack obtained by replacing  $NP(\mathcal{A}_1)_m$  in  $\mathcal{S}$  with  $S_1$  (excluding the left-most  $\mathbf{t}$  and  $\perp$ ) and then carrying out the transitions necessary for the pointer to point at a symbol representing  $q_r$ . Now  $\mathcal{A}$  is ready to read another input letter.

Suppose the input letter being read is  $y$ , with the pointer at a symbol representing  $q_r$ . If that symbol is  $q_r$  then there is a branch and we delete that symbol. Now the automaton can view the symbol under  $q_r$ , this symbol represents a state  $p$  of  $\mathcal{A}_1$  or  $\mathcal{A}_2$ . If  $y \in X_i$  and  $p$  is a state of  $\mathcal{A}_i$  then we can process  $y$  from the state  $p$ . Otherwise, the pointer will travel down the stack until it finds the first letter representing a state belong to the same automaton as  $y$ . That is, if  $y \in X_i$  then the automaton finds the first letter representing a state of  $p'$  in  $\mathcal{A}_i$ . Then the automaton uses it to run the  $\mathcal{A}_i$  from  $p'$  with the stack of  $\mathcal{A}_i$  being what is viewable to the automaton when it uses its pointer to move down, until the symbols on the stack are no longer those of  $\mathcal{A}_i$ . Then it uses the first symbol not from  $\mathcal{A}_i$  as a bottom of stack symbol. The stack gets edited the way  $\mathcal{A}_i$  would edit the stack. Finally if there is a branching the automaton moves the pointer to the top of the branch and writes  $q_r$ . Otherwise, the automaton moves to the top of the trunk and points at  $\bar{q}_r$ .

This process ensures that when the automaton reads a syllable  $w_i$  from  $\mathcal{A}_i$ , it will process that syllable in the way  $\mathcal{A}_i$  would process it given the state of  $\mathcal{A}_i$  that was visible (say  $q_i$ ) and what the stack was (i.e. the contents of the stack from  $q_i$  to the first symbol that is not from  $\mathcal{A}_i$ ). Then if the following syllable  $w_{i+1}$  is equal to the identity in its respective group, there will be no new branching. We see this since there are two possibilities. The automaton  $\mathcal{A}_j$  ( $i \neq j$ ) gets used from a state that is above  $q_i$  (this will happen if states above  $q_i$  now become visible

due to the part that  $w_i$  belonged to now becoming trivial after the processing of  $w_i$ ). Alternatively the state processing  $w_{i+1}$  is in the trunk and thus since  $w_{i+1}$  is equal to the identity, and the configuration from which  $\mathcal{A}_j$  runs was a start configuration. The configuration after  $w_{i+1}$  is processed must be the same and thus is a start configuration. Therefore the automaton will not create a new branch. Then the automaton moves its pointer upwards until it sees a symbol representing  $q_r$ . Therefore the next syllable  $w_{i+2}$  will be viewed as being continuation of the syllable  $w_i$ . More importantly, the above is also true if no branch is created even if  $w_{i+1}$  is not equal to the identity. This is very crucial. The reason is if there is a run which in one of the automata does not accept a string (regardless of whether or not it is equal to the identity), this is simulated in the automaton  $\mathcal{A}$  if the start state of that run is visible for that string where  $\mathcal{A}$  views it as a syllable and provided the stack is the same as it was in the start configuration of the run that did not accept the string.

However, whenever syllable  $w_j$  is equal to the identity in its respective groups, the automaton treats  $w_{j-1}w_{j+1}$  as a single syllable.

**Remark 3.3.2.** This technique of processing by syllables is one that we also make use of. This is important because if a string in a free product is non-trivial then its normal form has non-trivial syllables. So if one can prove that every element of the normal form is accepted by such an automaton then the proof would follow with the added assumption that the automaton used returns to its original configuration upon reading a string whose free reduction is empty. This is because by inserting finitely many strings with empty free reduction we can create any string in the co-word problem from strings in normal form. ♠

In fact the above remark is also what the proof in [28] relies on.

**Remark 3.3.3.** We achieve this in Theorem 4.6.1 by projecting syllables from both groups onto their free subgroups and then freely reducing in the structure of the automaton, where the check-stack is used to determine the generators of the free subgroups that we project onto.

We achieve this in Theorem 4.7.1 by projecting the syllables from the group  $\mathbb{Z}^n$  onto its free subgroups and then freely reducing in the structure of the automaton, where the check-stack is used to determine the generators of the free subgroups. For syllables from the virtually free group, we simply use the automaton accepting the word problem for a virtually free group in 2.3.2 to determine whether or not the syllable is trivial. ♠

# Chapter 4

## co-ETOL Groups

### 4.1 Introduction

In recent years, a number of papers have been published that take specific co-indexed groups from [28] and show that their co-word problem is an ETOL language. An example of this is [17] where Ciobanu et al. prove that the co-word problem of Grigorchuk's group is an ETOL language. This was then extended to proving that the co-word problem of bounded automata groups are ETOL languages by Elder and Bishop in [8]. In this chapter we shall discuss the class of co-ETOL groups which we define below. We recall from [2.3], it is shown in [18] that the class of ETOL languages are closed under inverse homomorphisms. Therefore we have the following definition.

**Definition 4.1.1** (co-ETOL group). Let  $G$  be a finitely generated group,  $G$  is said to be *co-ETOL* if there exists a CSPD automaton that accepts its co-word problem with respect to some (and hence any) symmetrically closed finite generating set.



**Remark 4.1.2.** The above statement is in the style of [28] and [27] where they defined co-indexed groups and  $co\mathcal{CF}$ , respectively. In the above statement we are implicitly using Lemma [2.3.3] to show that being a co-ETOL group is a property of the group and not of its chosen finite generating set, as ETOL languages are closed under inverse homomorphisms as shown in [18].



In Theorems [4.2.1], [4.3.1], [4.4.1] and [4.5.1] we establish that co-ETOL groups are closed under the following operations:

- taking finitely many direct products,
- passing to finitely generated subgroups,



- passing to finite index overgroups, and
- taking standard restricted wreath products with virtually free top groups.

We also prove that the class of co-ETOL groups contains  $\mathbb{Z}^n * \mathbb{Z}^m$  and  $\mathbb{Z}^n * G$  for a virtually free  $G$  in Theorems [4.6.1](#) and [4.7.1](#), respectively. We shall follow a very strict format when presenting and proving these results. Broadly speaking each section is dedicated to a theorem, and there are at least two subsections for each theorem. (Formally, there is a family of automata being used per theorem, and we describe the general structure of the members sufficiently for the arguments to be understood.) The first subsection will always be dedicated to the definition of the automaton which we shall use to prove the theorem. The second subsection will be the subsection where the formal proof is given, usually by means of providing a sequence of lemmata, and a final proof combining the lemmata.

Excepting the first definition subsection, the definition subsections will be organised in the following way. First, we give a high level summary of the intuitive description of each automaton, including a brief summary of why the automaton works. This will be labelled as **Intuitive Summary**, and will be presented after some initial setup. Then, we shall provide a detailed intuitive account of how the automaton works. This will be labelled as **Intuitive Idea**. We then present the formal definition of the automaton, and it will be labelled as **Formal Definition of  $\mathcal{A}$** . For the first definition subsection we shall only present an intuitive summary as Theorem [4.2.1](#) has a relatively short intuitive explanation.

We recall from [2.2.4](#) that for every CSPD automaton, there is an associated regular language. We may intuitively think of a run in the automaton as being divided into two stages. In the first stage, the automaton non-deterministically chooses a string from the regular language and places it onto the check-stack. In the second stage, the automaton processes the input string. Further recall that an input string is accepted if there exists a run of the automaton reading the input string such that the automaton is left in an accept state at the end of the run. That is, there exists a string from the associated regular language such that with that string on the check-stack, there is a processing of the input string (which depends on the choice of string on the check-stack) that leaves the automaton in an accept state at the end of processing.

## 4.2 Direct Products

In this section we shall present the following theorem.

**Theorem 4.2.1.** *If  $G$  and  $H$  are co-ETOL groups, then  $G \times H$  is co-ETOL.*

Throughout this section we shall assume that  $X$  and  $Y$  are finite generating sets for  $G$  and  $H$ , respectively.

We prove this theorem (including providing an intuitive explanation of our automaton) following our two-subsection structure as described above.

### 4.2.1 Definition of Automaton

Let  $\mathcal{A}_1 = (Q_1, \Sigma_1, \Delta_1, \Gamma_1, I_1, F_1, \mathcal{R}_1, \perp, \delta_1)$  be a CSPD automaton that accepts the  $coWP(G, X)$ . Further let  $\mathcal{A}_2 = (Q_2, \Sigma_2, \Delta_2, \Gamma_2, I_2, F_2, \mathcal{R}_2, \perp, \delta_2)$  be a CSPD automaton accepting  $coWP(H, Y)$ . We shall construct a CSPD automaton  $\mathcal{A}$  accepting  $coWP(G \times H, X \cup Y)$ .

**Intuitive Summary 4.2.2.** The check-stack will contain a string  $r \in \mathcal{R}_1 \cup \mathcal{R}_2$ . Let  $\omega$  be an input string and let  $k \in G \times H$  such that  $\omega =_{G \times H} k$ . Since the group is a direct product, for the element  $k$  there is  $g \in G$  and  $h \in H$  so that  $k = (g, h)$ . The automaton processes  $\omega$  as follows. The automaton non-deterministically chooses which coordinate of  $k$  to check is non-trivial. If it decides to check the  $i^{th}$  coordinate with  $r \in \mathcal{R}_i$  then the automaton simulates  $\mathcal{A}_i$  while ignoring letters from the other generating set as they project trivially onto the  $i^{th}$  coordinate. The automaton accepts if  $\mathcal{A}_i$  reaches an accept state. We note that  $\omega \neq_{G \times H} 1_{G \times H}$  if and only if the projection onto one of the coordinates of  $k$  is non-trivial. In that case, there exists a simulation of  $\mathcal{A}_1$  or  $\mathcal{A}_2$  with some  $r' \in \mathcal{R}_1 \cup \mathcal{R}_2$  on the check-stack yielding an accept state. This is because one of  $\mathcal{A}_1$  or  $\mathcal{A}_2$  will simulate the projection of  $\omega$  to  $X^*$  or  $Y^*$  respectively, and at least one of these projections is non-trivial in its respective group.  $\blacklozenge$

**Formal Definition of  $\mathcal{A}$ :** We define a CSPD automaton

$$\mathcal{A} := (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$$

accepting  $coWP(G \times H, X \cup Y)$  as follows.

- The state set consists of the following states
  - for every state in  $q \in Q_1$ , there is a state in  $Q$  representing that state which we shall denote by the same symbol. We shall refer to this subset of  $Q$  as  $Q_1$  as well, and it will be clear from context which automaton the set belongs to.
  - For every state in  $q \in Q_2$ , there is a state in  $Q$  representing that state which we shall denote by the same symbol. We shall refer to this subset of  $Q$  as  $Q_2$  as well, and it will be clear from context which automaton the set belongs to.

- The states  $q_0$  and  $q_f$ .
- The input alphabet  $\Sigma = \Sigma_1 \cup \Sigma_2 = X \cup Y$ .
- The check-stack alphabet  $\Delta = \Delta_1 \cup \Delta_2$ .
- The pushdown stack alphabet  $\Gamma = \Gamma_1 \cup \Gamma_2$ .
- The set of initial states  $I = \{q_0\}$ .
- The set of final states  $F = \{q_f\}$ .
- The regular language  $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ .
- The transition relation  $\delta$  consists of the following transitions:
  - (T0a) for every  $q \in I_1 \subseteq Q_1 \subseteq Q$  there is a transition  $((q_0, \varepsilon, (\perp, \perp)), (q, \perp)) \in \delta$ ; i.e., for every  $q \in I_1 \subseteq Q_1 \subseteq Q$  there is an  $\varepsilon$ -transition from  $q_0$  with the pushdown stack being empty to state  $q$  without editing the pushdown stack.
  - (T0b) For every  $q \in I_2 \subseteq Q_2 \subseteq Q$  there is a transition  $((q_0, \varepsilon, (\perp, \perp)), (q, \perp)) \in \delta$ ; i.e., for every  $q \in I_2 \subseteq Q_2 \subseteq Q$  there is an  $\varepsilon$ -transition from  $q_0$  with the pushdown stack being empty to state  $q$  without editing the pushdown stack.
  - (T1a) For every transition  $((q, x, (y, z)), (q_1, \omega)) \in \delta_1$  there is a transition  $((q, x, (y, z)), (q_1, \omega)) \in \delta$ .
  - (T1b) For every  $y \in Y$ , every pair  $(z, z_1) \in \{(\Delta_1 \times \Gamma_1) \cup \{(\perp, \perp)\}\}$  and every state  $q \in Q_1 \subseteq Q$  there is a transition

$$((q, y, (z, z_1)), (q, z_1)) \in \delta;$$

i.e., there is a transition from state  $q \in Q_1 \subseteq Q$  with  $z_1 \in \Gamma_1 \cup \{\perp\}$  at the top of the pushdown stack with  $z \in \Delta_1 \cup \{\perp\}$  being the corresponding letter on the check-stack (such that  $(z, z_1) \in \{(\Delta_1 \times \Gamma_1) \cup \{(\perp, \perp)\}\}$ ) upon reading an input letter  $y \in Y$  to state  $q$  without editing the stack.

- (T2a) For every transition  $((q, x, (y, z)), (q_1, \omega)) \in \delta_2$  there is a transition  $((q, x, (y, z)), (q_1, \omega)) \in \delta$ ,
- (T2b) for every  $x \in X$ , every pair  $(z, z_1) \in \{(\Delta_2 \times \Gamma_2) \cup \{(\perp, \perp)\}\}$  and every state  $q \in Q_2 \subseteq Q$  there is a transition

$$((q, x, (z, z_1)), (q, z_1)) \in \delta;$$

i.e., there is a transition from state  $q \in Q_2 \subseteq Q$  with  $z_1 \in \Gamma_2 \cup \{\perp\}$  at the top of the pushdown stack with  $z \in \Delta_2 \cup \{\perp\}$  being the corresponding letter on the check-stack (such that  $(z, z_1) \in \{(\Delta_2 \times \Gamma_2) \cup \{(\perp, \perp)\}\}$ ) upon reading an input letter  $x \in X$  to state  $q$  without editing the stack.

(T3a) For every state  $q \in F_1 \subseteq Q_1 \subseteq Q$  and every pair  $(y, z) \in \{(\Delta_1 \times \Gamma_1) \cup \{(\perp, \perp)\}\}$  there is a transition

$$((q, \varepsilon, (y, z)), (q_f, z)) \in \delta;$$

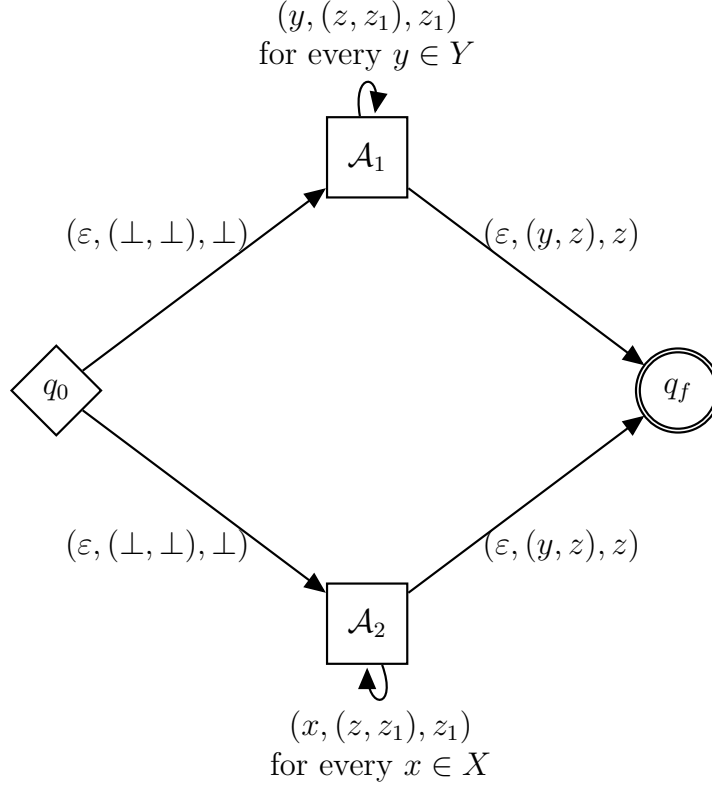
i.e., there is an  $\varepsilon$ -transition from state  $q \in F_1 \subseteq Q_1 \subseteq Q$  with  $z \in \Gamma_1 \cup \{\perp\}$  at the top of the pushdown stack with  $y \in \Delta_1 \cup \{\perp\}$  being the corresponding letter on the check-stack (such that  $(y, z) \in \{(\Delta_1 \times \Gamma_1) \cup \{(\perp, \perp)\}\}$ ) to state  $q_f$  without editing the stack.

(T3b) For every state  $q \in F_2 \subseteq Q_2 \subseteq Q$  every pair  $(y, z) \in \{(\Delta_2 \times \Gamma_2) \cup \{(\perp, \perp)\}\}$  there is a transition

$$((q, \varepsilon, (y, z)), (q_f, z)) \in \delta;$$

i.e., there is an  $\varepsilon$ -transition from state  $q \in F_2 \subseteq Q_2 \subseteq Q$  with  $z \in \Gamma_2 \cup \{\perp\}$  at the top of the pushdown stack with  $y \in \Delta_2 \cup \{\perp\}$  being the corresponding letter on the check-stack (such that  $(y, z) \in \{(\Delta_2 \times \Gamma_2) \cup \{(\perp, \perp)\}\}$ ) to state  $q_f$  without editing the stack.

Below we give a diagram to aid in understanding of how the automaton works. We note while the diagram bears resemblance to a transition diagram, it is not a transition diagram as all the states and transitions are not presented. We merely present arrows representing (T0a), (T0b), (T1b), (T2b), (T3a), and (T3b). Further, we do not present all the states. The states  $q_0$  and  $q_f$  are explicitly presented and the rest are implicit within the squares with labels  $\mathcal{A}_1$  and  $\mathcal{A}_2$ . We shall elaborate further on the diagram below.



In the above diagram, the arrow from  $q_0$  to  $\mathcal{A}_1$  with label  $(\varepsilon, (\perp, \perp), \perp)$  represents an arrow from  $q_0$  to every initial state of  $\mathcal{A}_1$  with label  $(\varepsilon, (\perp, \perp), \perp)$ . Similarly, the arrow from  $q_0$  to  $\mathcal{A}_2$  with label  $(\varepsilon, (\perp, \perp), \perp)$  represents an arrow from  $q_0$  to every initial state of  $\mathcal{A}_2$  with label  $(\varepsilon, (\perp, \perp), \perp)$ . These arrows reflect (T0a) and (T0b) respectively. Further, the arrow  $\mathcal{A}_1$  to  $q_f$  with label  $(\varepsilon, (y, z), z)$  represents an arrow from each final state of  $\mathcal{A}_1$  to  $q_f$  with label “ $(\varepsilon, (y, z), z)$  for every pair  $(y, z) \in \{(\Delta_1 \times \Gamma_1) \cup \{(\perp, \perp)\}\}$ ”. The arrow from  $\mathcal{A}_2$  to  $q_f$  should be interpreted analogously. These arrows reflect (T3a) and (T3b) respectively. The squares  $\mathcal{A}_1$  and  $\mathcal{A}_2$  represent the transition diagrams for  $\mathcal{A}_1$  and  $\mathcal{A}_2$  respectively. The arrow from  $\mathcal{A}_1$  to  $\mathcal{A}_1$  with label “ $(y, (z, z_1), z_1)$  for every  $y \in Y$ ” represents an arrow from every state  $q$  of  $\mathcal{A}_1$  to itself with label “ $(y, (z, z_1), z_1)$  for every  $y \in Y$  and every pair  $(z, z_1) \in \{(\Delta_1 \times \Gamma_1) \cup \{(\perp, \perp)\}\}$ ”. This reflects (T1b). The arrow from  $\mathcal{A}_2$  to itself should be interpreted analogously.

## 4.2.2 Proof of Theorem [4.2.1](#)

In this section we first provide a sequence of technical lemmata. We then present the proof of Theorem [4.2.1](#). Before we continue we recall Definition [2.3.13](#). Throughout this subsection, we shall let  $G, H, X, Y, \mathcal{A}_1, \mathcal{A}_2$  and  $\mathcal{A}$  as in the subsection above.

We start with giving a modified version of one of the functions defined in Definition [2.3.13](#)

**Definition 4.2.3.** Let  $Z' := \{a_1, a_2, \dots, a_n\}$  be any set and let  $Z$  be the symmetric closure of  $Z'$ . Let  $X$  be a symmetrically closed proper subset of  $Z$ . We define a monoid homomorphism  $\varphi_X : Z^* \rightarrow Z^*$  as follows. First let  $z \in Z \cup \{\varepsilon\}$  then

$$\varphi_X(z) = \begin{cases} z & \text{if } z \in X \\ \varepsilon & \text{otherwise.} \end{cases}$$

Now let  $\omega' = z_1 z_2 \cdots z_n \in Z^*$  then define  $\varphi_X(\omega') = \varphi_X(z_1) \varphi_X(z_2) \cdots \varphi_X(z_n)$ . ♣

Observe  $\varphi_X$  simply deletes letters of a string that are not in  $X$ . We are now ready to present a structural lemma about the automaton  $\mathcal{A}$ .

**Lemma 4.2.4.** Let  $\omega = \sigma_1 \sigma_2 \cdots \sigma_n \in (X \cup Y)^*$ . Set  $\varphi_1 := \varphi_X$  and  $\varphi_2 := \varphi_Y$ . Set  $\varphi_1(\omega) = z_1 z_2 \cdots z_l \in X^*$  and  $\varphi_2(\omega) = z'_1 z'_2 \cdots z'_t \in Y^*$ . There exist strings

$$u_1, u_2, \dots, u_{l+1} \in Y^*,$$

and

$$v_1, v_2, \dots, v_{t+1} \in X^*$$

such that

$$\omega = u_1 z_1 u_2 z_2 \cdots u_l z_l u_{l+1},$$

and

$$\omega = v_1 z'_1 v_2 z'_2 \cdots v_t z'_t v_{t+1}.$$

Further set  $u_i = y_{i,1} y_{i,2} \cdots y_{i,s_i}$  and  $v_i = x_{i,1} x_{i,2} \cdots x_{i,r_i}$ .

Then the following holds.

1. If  $\mathcal{A}_1$  can read  $\varphi_1(\omega)$  through a chain of the following form

$$C_0^1 \underset{\varepsilon}{\sim}^* C_1^1 \underset{z_1}{\sim} C_2^1 \underset{\varepsilon}{\sim}^* C_3^1 \underset{z_2}{\sim} \cdots \underset{z_l}{\sim} C_{2l}^1 \underset{\varepsilon}{\sim}^* C_{2l+1}^1$$

then  $\mathcal{A}$  can read  $\omega$  through the chain

$$\begin{aligned} C_0 &\underset{\varepsilon}{\sim} C_0^1 \underset{\varepsilon}{\sim}^* C_1^1 \underset{y_{1,1}}{\sim} C_1^1 \underset{y_{1,2}}{\sim} C_1^1 \underset{y_{1,3}}{\sim} \cdots \underset{y_{1,s_1}}{\sim} C_1^1 \\ &\underset{z_1}{\sim} C_2^1 \underset{\varepsilon}{\sim}^* C_3^1 \underset{y_{2,1}}{\sim} C_3^1 \underset{y_{2,2}}{\sim} C_3^1 \underset{y_{2,3}}{\sim} \cdots \underset{y_{2,s_2}}{\sim} C_3^1 \\ &\underset{z_2}{\sim} C_4^1 \underset{\varepsilon}{\sim}^* C_4^1 \underset{y_{3,1}}{\sim} C_4^1 \underset{y_{3,2}}{\sim} C_4^1 \underset{y_{3,3}}{\sim} \cdots \underset{y_{3,s_3}}{\sim} C_4^1 \\ &\vdots \\ &\underset{z_l}{\sim} C_{2l}^1 \underset{\varepsilon}{\sim}^* C_{2l+1}^1 \underset{y_{l+1,1}}{\sim} C_{2l+1}^1 \underset{y_{l+1,2}}{\sim} C_{2l+1}^1 \underset{y_{l+1,3}}{\sim} \cdots \underset{y_{l+1,s_{l+1}}}{\sim} C_{2l+1}^1 \\ &\underset{\varepsilon}{\sim}^* C_1. \end{aligned}$$

2. If  $\mathcal{A}_2$  can read  $\varphi_2(\omega)$  through a chain of the following form

$$C_0^2 \sim_{\varepsilon}^* C_1^2 \sim_{z'_1} C_2^2 \sim_{\varepsilon}^* C_3^2 \sim_{z'_2} \cdots \sim_{z'_t} C_{2t}^2 \sim_{\varepsilon}^* C_{2t+1}^2$$

then  $\mathcal{A}$  can read  $\omega$  through the chain

$$\begin{aligned} C_0 &\sim_{\varepsilon} C_0^1 \sim_{\varepsilon}^* C_1^1 \sim_{x_{1,1}} C_1^1 \sim_{x_{1,2}} C_1^1 \sim_{x_{1,3}} \cdots \sim_{x_{1,r_1}} C_1^1 \\ &\sim_{z'_1} C_2^1 \sim_{\varepsilon}^* C_3^1 \sim_{x_{2,1}} C_3^1 \sim_{x_{2,2}} C_3^1 \sim_{x_{2,3}} \cdots \sim_{x_{2,r_2}} C_3^1 \\ &\sim_{z'_2} C_4^1 \sim_{\varepsilon}^* C_4^1 \sim_{x_{3,1}} C_4^1 \sim_{x_{3,2}} C_4^1 \sim_{x_{3,3}} \cdots \sim_{x_{3,r_3}} C_4^1 \\ &\vdots \\ &\sim_{z'_t} C_{2t}^1 \sim_{\varepsilon}^* C_{2t+1}^1 \sim_{x_{t+1,1}} C_{2t+1}^1 \sim_{x_{t+1,2}} C_{2t+1}^1 \sim_{x_{t+1,3}} \cdots \sim_{x_{t+1,r_{t+1}}} C_{2t+1}^1 \\ &\sim_{\varepsilon}^* C_2. \end{aligned}$$

*Proof.* Since 1 and 2 are analogous, we shall only prove 1. Let  $\omega = \sigma_1 \sigma_2 \cdots \sigma_n \in (X \cup Y)^*$ . We shall assume as in statement of the lemma, that  $\mathcal{A}_1$  can read  $\varphi_1(\omega)$  through a chain of the form

$$C_0^1 \sim_{\varepsilon}^* C_1^1 \sim_{z_1} C_2^1 \sim_{\varepsilon}^* C_3^1 \sim_{z_2} \cdots \sim_{z_l} C_{2l}^1 \sim_{\varepsilon}^* C_{2l+1}^1. \quad (8)$$

By definition of  $\mathcal{A}_1$  as a CSPD automaton (see Definition [2.2.42](#)), the stack of configurations is a pair  $(S_1, S_2)$ , and the first entry  $S_1$  of the pair is a tuple  $(a_k, a_{k-1}, \dots, a_1, \perp)$  such that  $a_k a_{k-1} \cdots a_1 \in \mathcal{R}_1$ . Further  $S_1$  cannot be edited using transitions. That is, if  $\mathcal{A}_1$  reads  $\varphi_1(\omega)$  through the chain above, then it must be that there exists a string  $a_k a_{k-1} \cdots a_1 \in \mathcal{R}_1$  on the check-stack in the run above. We shall use this string on the check-stack in order to read  $\omega$  below. Further, we shall analyse the chain above to induce a chain in  $\mathcal{A}$  reading  $\omega$ . First we note that the only initial state of  $\mathcal{A}$  is  $q_0$ . Further observe that the state of  $C_0^1$ , say  $q'_0$  is an initial state of  $\mathcal{A}_1$ . The automaton  $\mathcal{A}$  uses  $((q_0, \varepsilon, (\perp, \perp)), (q'_0, \perp))$  (in (T0a)) to move the state from  $q_0$  to  $q'_0$  without editing the stack. Thus the chain starts

$$C_0 = (q_0, ((a_k, a_{k-1}, \dots, a_1, \perp), (\perp))) \sim_{\varepsilon} C_0^1.$$

Since  $C_0^1 \sim_{\varepsilon}^* C_1^1$ , it must be that  $\mathcal{A}_1$  used a sequence of  $\varepsilon$ -transitions from  $C_0^1$  to  $C_1^1$ . Since (T1a) consists of all the transitions of  $\mathcal{A}_1$ , we shall use precisely the same transitions in the same order and thus continue the chain as

$$C_0 = (q_0, ((a_k, a_{k-1}, \dots, a_1, \perp), (\perp))) \sim_{\varepsilon} C_0^1 \sim_{\varepsilon}^* C_1^1.$$

We shall now use transitions in (T1b); these transitions allow us to read input letters from  $Y$  without changing the state of the automaton or editing the stack.

Therefore,  $\mathcal{A}$  can use these transitions without changing the configuration. So the chain continues as

$$C_0 \sim_\varepsilon C_0^1 \sim_\varepsilon^* C_1^1 \sim_{y_{1,1}} C_1^1 \sim_{y_{1,2}} C_1^1 \sim_{y_{1,3}} \cdots \sim_{y_{1,s_1}} C_1^1.$$

The next input letter to be read is  $z_1 \in X$ . Recall that  $\mathcal{A}_1$  can read  $\varphi_1(\omega)$  through the chain in (8). Therefore it must be that there exists a transition  $\alpha \in \delta_1$  by which  $C_1^1 \sim_{z_1} C_2^1$ . Since  $\alpha$  is also in  $\delta$  (in (T1a)), we must have the chain continuing as

$$\begin{aligned} C_0 \sim_\varepsilon C_0^1 \sim_\varepsilon^* C_1^1 \sim_{x_{1,1}} C_1^1 \sim_{x_{1,2}} C_1^1 \sim_{x_{1,3}} \cdots \sim_{x_{1,r_1}} C_1^1 \\ \sim_{z_1'} C_2^1. \end{aligned}$$

Continuing in this manner, we see that  $\mathcal{A}$  reads  $\omega$  through the chain in the statement of the lemma.  $\blacksquare$

**Lemma 4.2.5.** *Let  $\omega = \sigma_1\sigma_2 \cdots \sigma_n \in (X \cup Y)^*$ . Then the only chains by which  $\mathcal{A}$  can read  $\omega$  are those in Lemma [4.2.4](#).*

*Proof.* Let  $\omega = \sigma_1\sigma_2 \cdots \sigma_n \in (X \cup Y)^*$ . Now suppose that  $\mathcal{A}$  reads  $\omega$  through a chain

$$C_0 \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* C_3 \sim_{\sigma_2} \cdots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1}. \quad (9)$$

We shall analyse the chain above and make some conclusions about transitions that must have been used by  $\mathcal{A}$  to read  $\omega$ . Using these transitions we shall construct a chain by which  $\mathcal{A}_1$  reads  $\varphi_1(\omega)$  or  $\mathcal{A}_2$  reads  $\varphi_2(\omega)$ .

First we note that there are no transitions from  $q_f$ , thus if transitions from (T3a) or (T3b) were used, it must be that they were used after every input letter has been read. We shall assume that these transitions were not used, as if they were the chain can simply be extended.

We note that if  $\mathcal{A}$  reads  $\omega$  through the chain in (9), then it must do so with a string  $b_1b_2 \cdots b_m$  on the check-stack such that  $b_1b_2 \cdots b_m \in \mathcal{R}_1 \cup \mathcal{R}_2$ . Now observe that the transitions from  $q_0$  are  $\varepsilon$  transitions to the initial states of  $\mathcal{A}_1$  and  $\mathcal{A}_2$  (as per (T0a) and (T0b)). Therefore it must be that the first transition in the sequence of  $\varepsilon$ -transitions used to obtain  $C_0 \sim_\varepsilon^* C_1$  in (9) is a transition in (T0a) or (T0b). Let  $\alpha$  be that transition and suppose  $\alpha$  is a transition from  $q_0$  to an initial state of  $\mathcal{A}_1$ , say  $q'_0$ . (The case for  $\alpha$  being a transition to an initial state of  $\mathcal{A}_2$  is similar.) Therefore we shall start by making this transition explicit in the chain.

$$C_0 = (q_0, (S_1, \perp)) \sim_\varepsilon (q'_0, (S_1, \perp)) \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* C_3 \sim_{\sigma_2} \cdots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1}. \quad (9)$$

Now observe that there are no transitions from any state in  $Q_1$  to any state in  $Q_2$ . Thus it must be that the state of  $C_1$  is in  $Q_1$ . Thus  $(q'_0, (S_1, \perp)) \sim_\varepsilon^* C_1$  is due



to a sequence of  $\varepsilon$ -transitions from (T1a). Therefore this sequence is in  $\delta_1$ , and thus

$$(q'_0, (S_1, \perp)) \sim_\varepsilon^* C_1 \quad (9.a)$$

is a chain of relations in  $\mathcal{A}_1$ .

Similarly, we note that the state of  $C_2$  must be in  $Q_1$  as there are no reading-transitions from a state in  $Q_1 \subseteq Q$  to a state not in  $Q_1$ . Either  $\sigma_1 \in X$  or  $\sigma_1 \in Y$ . We shall now consider both these cases. If  $\sigma_1 \in X$  then it must be that there is a transition in (T1a) by which  $C_1 \sim_{\sigma_1} C_2$ . This transition by definition must be in  $\delta_1$ , and therefore the chain in (9.a) continues as

$$(q'_0, (S_1, \perp)) \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2. \quad (9.a)$$

If  $\sigma_1 \in Y$ , then it must be that  $C_2 = C_1$  as the transitions from a state in  $Q_1$  upon reading a letter from  $Y$  remain at the same state and do not edit the stack.

Continuing in this manner, we see that the chain in (9) is induced by a chain in  $\mathcal{A}_1$  reading  $\varphi_1(\omega)$  in the sense of Lemma 4.2.4.  $\blacksquare$

Before we are ready to provide the proof of Theorem 4.2.1 we shall record an immediate consequence of the definition of  $\mathcal{A}$  as well as the above lemmata.

**Corollary 4.2.6.** *The only accept state is  $q_f$ . Further, let  $\omega = \sigma_1\sigma_2\cdots\sigma_n \in (X \cup Y)^*$  be a string read by  $\mathcal{A}$  through a chain such as*

$$C_0 = (q_0, (S_1, \perp)) \sim_\varepsilon (q'_0, (S_1, \perp)) \sim_\varepsilon^* C_1 \sim_{\sigma_1} C_2 \sim_\varepsilon^* C_3 \sim_{\sigma_2} \cdots \sim_{\sigma_n} C_{2n} \sim_\varepsilon^* C_{2n+1},$$

where  $q'_0 \in I_1$  (or  $I_2$ ) and  $S_1 = (a_m, a_{m-1}, \dots, a_1, \perp)$  for some string  $a_m a_{m-1} \cdots a_1 \in \mathcal{R}_1$  (or  $\mathcal{R}_2$  respectively).

If  $\omega$  is accepted through the above chain then

- the configuration  $C_{2n+1} = (q_f, (S_1, S_2))$  where there exists a configuration  $C = (q_A, (S_1, S_2))$  such that

$$C_{2n} \sim_\varepsilon^* C \sim_\varepsilon C_{2n+1},$$

- and further  $C$  is an accept configuration of  $\mathcal{A}_1$  (or  $\mathcal{A}_2$  respectively).

*Proof.* By the definition of  $\mathcal{A}$  we know that  $q_f$  is the only accept state. Further, we know that if  $\omega$  is read through a chain, such as the in the statement then by Lemma 4.2.5 it must be that this chain was induced by  $\mathcal{A}_1$  (or  $\mathcal{A}_2$ ) reading  $\varphi_1(\omega)$  (or  $\varphi_2(\omega)$  respectively) as in Lemma 4.2.4. Further, by definition, if  $\omega$  is accepted it must be that transitions in (T3a) (or (T3b) respectively) were used

since there are no other transitions to  $q_f$ . Therefore there must exist a configuration  $C = (q_A, (S_1, S_2))$  where  $q_A \in F_1$  (or  $q_A \in F_2$  respectively) such that

$$C_{2n} \sim_{\varepsilon}^* C \sim_{\varepsilon} C_{2n+1}.$$

Further as  $q \in F_1$  (or  $F_2$ ), then  $C$  is an accept configuration of  $\mathcal{A}_1$  (or  $\mathcal{A}_2$  respectively).  $\blacksquare$

We are now ready to prove Theorem [4.2.1](#)

*Proof of Theorem [4.2.1](#).* We shall show that  $\mathcal{A}$ , defined in [4.2.1](#), accepts  $coWP(G \times H, X \cup Y)$ . Let  $\omega = \sigma_1 \sigma_2 \cdots \sigma_n \in (X \cup Y)^*$ . By Lemma [4.2.4](#) and Lemma [4.2.5](#) we know that if  $\mathcal{A}$  can read  $\omega$ , then it must do so as follows. If  $\mathcal{A}$  can read  $\omega$ , then by Lemma [4.2.5](#) it must be that at least one of  $\mathcal{A}_1$  or  $\mathcal{A}_2$  can read  $\varphi_1(\omega)$  or  $\varphi_2(\omega)$  respectively such that any chain whereby  $\mathcal{A}_i$  reads  $\varphi_i(\omega)$  induces a chain by which  $\mathcal{A}$  reads  $\omega$  as in Lemma [4.2.4](#)

Observe that  $\omega \in coWP(G \times H, X \cup Y)$  if and only if  $\varphi_1(\omega) \neq_G 1_G$  or  $\varphi_2(\omega) \neq_H 1_H$ . Thus if  $\omega \in coWP(G \times H, X \cup Y)$  then either  $\varphi_1(\omega) \in \mathcal{L}(\mathcal{A}_1)$  or  $\varphi_2(\omega) \in \mathcal{L}(\mathcal{A}_2)$ .

Suppose  $\omega \in coWP(G \times H, X \cup Y)$  and further suppose without loss of generality that  $\varphi_1(\omega) \neq_G 1_G$ . Let  $\varphi_1(\omega) = z_1 z_2 \cdots z_l$ . Then there must exist a chain in  $\mathcal{A}_1$  accepting  $\varphi_1(\omega)$  such as

$$C_0^1 \sim_{\varepsilon}^* C_1^1 \sim_{z_1} C_2^1 \sim_{\varepsilon}^* C_3^1 \sim_{z_2} \cdots \sim_{z_l} C_{2l}^1 \sim_{\varepsilon}^* C_{2l+1}^1. \quad (10a)$$

Then by Lemma [4.2.4](#) there is a chain in  $\mathcal{A}$  (of the form given in the statement of Lemma [4.2.4](#)). Assuming the conditions of Lemma [4.2.4](#), the chain induced by (10a) above is

$$\begin{aligned} C_0 &\sim_{\varepsilon} C_0^1 \sim_{\varepsilon}^* C_1^1 \sim_{y_{1,1}} C_1^1 \sim_{y_{1,2}} C_1^1 \sim_{y_{1,3}} \cdots \sim_{y_{1,s_1}} C_1^1 \\ &\sim_{z_1} C_2^1 \sim_{\varepsilon}^* C_3^1 \sim_{y_{2,1}} C_3^1 \sim_{y_{2,2}} C_3^1 \sim_{y_{2,3}} \cdots \sim_{y_{2,s_2}} C_3^1 \\ &\sim_{z_2} C_4^1 \sim_{\varepsilon}^* C_4^1 \sim_{y_{3,1}} C_4^1 \sim_{y_{3,2}} C_4^1 \sim_{y_{3,3}} \cdots \sim_{y_{3,s_3}} C_4^1 \\ &\vdots \\ &\sim_{z_l} C_{2l}^1 \sim_{\varepsilon}^* C_{2l+1}^1 \sim_{y_{2l+1,1}} C_{2l+1}^1 \sim_{y_{2l+1,2}} C_{2l+1}^1 \sim_{y_{2l+1,3}} \cdots \sim_{y_{2l+1,s_{l+1}}} C_{2l+1}^1 \\ &\sim_{\varepsilon}^* C_1. \end{aligned}$$

Since  $\varphi_1(\omega)$  is accepted by  $\mathcal{A}_1$ , then  $C_{2l+1}^1$  is an accept configuration of  $\mathcal{A}_1$ . In particular the state of  $C_{2l+1}^1$  is an accept state, say  $q_A$ . Therefore  $\mathcal{A}$  can use a transition in (T3a) such that  $C_{2l+1}^1 \sim_{\varepsilon} (q_f, S)$  such that  $C_{2l+1}^1 = (q_A, S)$ . By Corollary [4.2.6](#), we see that  $\omega$  is accepted.

Now suppose  $\omega \notin coWP(G \times H, X \cup Y)$ . Observe that  $\omega \notin coWP(G \times H, X \cup Y)$  if and only if  $\varphi_1(\omega) =_G 1_G$  and  $\varphi_2(\omega) =_H 1_H$ . Therefore  $\mathcal{A}_1$  does not accept

$\varphi_1(\omega) =_G 1_G$ , and  $\mathcal{A}_2$  does not accept  $\varphi_2(\omega) =_H 1_H$ . Thus there does not exist a chain in  $\mathcal{A}_1$  ending with an accept configurations reading  $\varphi_1(\omega)$ . Similarly, there does not exist a chain in  $\mathcal{A}_2$  ending with an accept configurations reading  $\varphi_2(\omega)$ . Thus if  $\mathcal{A}_i$  reads  $\varphi_i(\omega) = z_1^i z_2^i \cdots z_{j^i}^i$  (where if  $i = 1$  then  $z^i = z$  and  $j^i = l$ , and if  $i = 2$  then  $z^i = z''$  and  $j^i = t$ ) through a chain such as

$$C_0^i \sim_\varepsilon^* C_1^i \sim_{z_1^i} C_2^1 \sim_\varepsilon^* C_3^i \sim_{z_2^i} \cdots \sim_{z_{j^i}^i} C_{2j^i}^i \sim_\varepsilon^* C_{2j^i+1}^i \quad (10b)$$

then  $C_{2j^i+1}^i$  is not an accept configuration. Therefore the last configuration in the induced chain (by Lemma 4.2.4) cannot be an accept configuration of  $\mathcal{A}$  as (T3a) and (T3b) could not have been used. Further, by Lemma 4.2.5 there is no other way to read  $\omega$ . Therefore by Corollary 4.2.6  $\omega$  is not accepted by  $\mathcal{A}$  as the state of the last configuration induced by the chain in (10b) cannot be an accept configuration of  $\mathcal{A}$ .

Therefore  $\mathcal{L}(\mathcal{A}) = \text{coWP}(G \times H, X \cup Y)$ . ■

## 4.3 Finite Index Overgroups

In this section we shall present the following theorem.

**Theorem 4.3.1.** *Let  $G$  be a group and let  $H$  be a finite indexed subgroup of  $G$ . If  $H$  is a co-ETOL group, then  $G$  is a co-ETOL group.*

We prove this theorem (including providing an intuitive explanation of our automaton) following our two-subsection structure as described in the introduction of the chapter.

### 4.3.1 Definition of Automaton

Let  $X$  be a finite generating set for  $H$ . Let  $\mathcal{A}' = (Q', \Sigma', \Delta', \Gamma', I', F', \mathcal{R}', \perp, \delta')$  be a CSPD automaton accepting  $\text{coWP}(H, X)$ .

Let  $G$  be a finite index overgroup of  $H$ . Suppose  $H$  has index  $m$  in  $G$ . Let  $T'$  be a right transversal for  $H$  in  $G$  such that  $1_G \in T'$ .

**Intuitive Summary 4.3.2.** The check-stack in  $\mathcal{A}$  will contain strings  $r \in \mathcal{R}'$  just as the check-stack in  $\mathcal{A}'$  did. Let  $\omega$  be an input string and let  $g \in G$  be such that  $g =_G \omega$ . Since  $G$  is a finite index overgroup  $H$  then the element  $g$  representing  $\omega$  maybe expressed as  $ht$  where  $h \in H$  and  $t \in T'$ . The automaton processes  $\omega$  as follows. The automaton non-deterministically chooses to determine whether  $h$  is non-trivial or  $t$  is non-trivial.

To determine if  $t$  is non-trivial the automaton does the following. Upon reading an input letter, the element representing the string read so far changes. Thus

the coset that element is in also changes. The automaton moves between states representing cosets in a way that reflects these changes. We note that if  $t \neq_G 1_G$  then the state reached at the end of this process will not be a state representing the coset  $H1$ , and the automaton will accept  $\omega$ . The automaton does not rely on the contents of the check-stack during the process we just described.

To determine if  $h$  is non-trivial the automaton does the following. The automaton simulates a run of  $\mathcal{A}_H$  reading a specific string  $\mu_\omega =_H h$  with  $r$  on the check-stack. (We describe the process of obtaining  $\mu_\omega$  later.) If the simulated run of  $\mathcal{A}_H$  ends in an accept state the automaton will accept  $\omega$ . We note that  $\omega \neq_G 1_G$  if and only if either  $h \neq_H 1_H$  or  $t \neq 1_G$ . These cases will be witnessed by the automaton in some run, and thus  $\omega$  will be accepted.  $\blacklozenge$

Let  $T$  be the symmetric closure of  $T'$ . Observe that  $Y := X \cup T$  is a symmetrically closed generating set for  $G$ . For each  $y \in Y$  and  $t \in T'$ ,  $ty \in G$  thus  $ty = h_{ty}t'$  for some  $h_{ty} \in H$  and  $t' \in T'$ . Fix a string  $w_{ty} := x_1x_2 \cdots x_r \in X^*$  such that  $x_1x_2 \cdots x_r =_H h_{ty}$ . (The string  $\mu_\omega$  is the string  $w_{t_0y_1}w_{t_1y_2} \cdots w_{t_{n-1}y_n}$ .) Let  $Z$  denote the set  $\{w_{ty} \mid t \in T', y \in Y\}$ .

We will make use of the following definition.

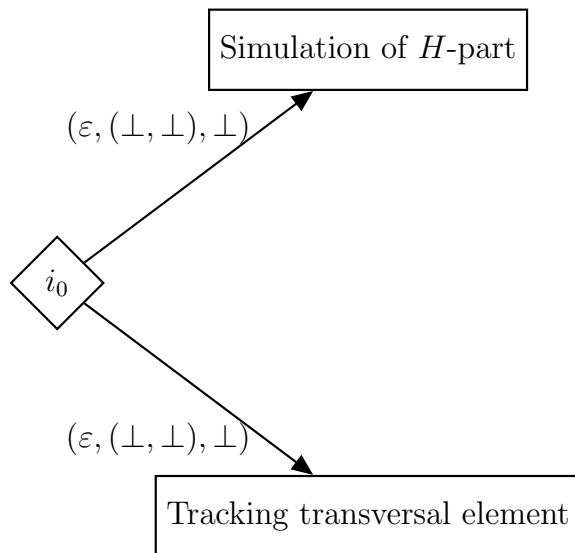
**Definition 4.3.3.** Let  $Z$  be a finite set. Let  $w \in Z^*$ . We say  $v \in Z^*$  is a *suffix* of  $w$  if there exists a  $u \in Z^*$  such that  $w = uv$ .

We denote the set of all suffixes of a string  $w$  by  $Suffix(w)$ . Similarly, we set  $Suffix(A) := \bigcup_{w \in A} Suffix(w)$  for  $A \subseteq Z^*$ .  $\clubsuit$

**Intuitive Idea:** The check-stack will contain strings from  $\mathcal{R}'$ . The state set  $Q$  can be viewed as a disjoint union of three sets which we describe below.

- The state  $i_0$ , this state is also the initial state.
- For every  $q \in Q'$ ,  $t \in T'$ , and  $u \in Suffix(Z)$ , there is a state  $(q, t, u)$ . We denote the subset of states of the form  $(q_0, 1_G, \varepsilon)$  where  $q_0 \in I'$  by  $Q_0$ .
- For every  $t \in T'$ , there is a state  $t$  representing that element. We denote this set by  $T'_1$ .

Below we give a diagram to aid in understanding how the automaton works. We note that while the diagram resembles a transition diagram, it is not a transition diagram. In particular we only represent the state  $i_0$  explicitly. The two other squares represent two ways in which the automaton processes an input string. We shall elaborate further on the diagram below.



In the above diagram, the arrow from  $i_0$  to the rectangle labelled “Simulation of  $H$ -part” represents the transitions from  $i_0$  to the subset of the states  $Q_0$ , while the arrow from  $i_0$  to the rectangle labelled “Tracking transversal element” represents the transition from  $i_0$  to  $1_G \in T'_1$ .

The rectangle labelled “Simulation of  $H$ -part” represents the part of the automaton consisting of states of the form  $(q, t, u)$  where  $q \in Q'$ ,  $t \in T'$ , and  $u \in \text{Suffix}(Z)$  together with all the transitions between these states. However, the rectangle labelled “Tracking transversal element” represents the part of the automaton consisting of states in  $T'_1$  and all the transitions between them.

The diagram highlights the non-deterministic nature of the automaton as well as the goal of its respective parts. Let  $y_1 y_2 \cdots y_r \in Y^*$  be an input string that is being read by  $\mathcal{A}$ . We shall informally discuss what the transitions of  $\mathcal{A}$  do in a run reading  $y_1 y_2 \cdots y_r$ . We shall do so in two parts as there are two ways  $y_1 y_2 \cdots y_r$  could be processed by  $\mathcal{A}$ .

Firstly, we shall discuss how “Tracking transversal element” processes the input string. Upon reading a prefix  $v = y_1 y_2 \cdots y_l$  of  $y_1 y_2 \cdots y_r$ , the state of the automaton will be  $t \in T'_1$  such that  $v =_G ht$  where  $t \in T'$  and  $h \in H$ . From state  $t$ , upon reading  $y_{l+1}$ , the automaton will transition to state  $t' \in T'_1$  such that  $ty_{l+1} = h't'$  where  $t' \in T'_1$  and  $h' \in H$ . This is what the transitions in (T4) do. These transitions process the input string in the way described independently of the choice of string on the check-stack.

We shall now discuss how “Simulation of  $H$ -part” processes the input string. Suppose the check-stack contains a string  $r \in \mathcal{R}'$ . Let  $t_0 = 1_G$  and recall that the strings  $w_{t_{i-1}y_i}$  are those that we have fixed such that  $w_{t_{i-1}y_i} =_H h_{t_{i-1}y_i}$  where  $h_{t_{i-1}y_i}$  are defined by

$$t_{i-1}y_i =_G h_{t_{i-1}y_i} t_i.$$

If  $\mathcal{A}'$  can read  $w_{t_0y_1}w_{t_1y_2} \cdots w_{t_{n-1}y_n}$  through a chain of relations. Then the part of  $\mathcal{A}$  represented in the rectangle with the label “Simulation of  $H$ -part” simulates a read of  $w_{t_0y_1}w_{t_1y_2} \cdots w_{t_{n-1}y_n}$  by  $\mathcal{A}'$  through the transitions in (T1), (T2), and (T3) below (while also keeping track of the transversal element in a similar way as the above).

**Formal Definition of  $\mathcal{A}$ :** We define a CSPD automaton

$$A := (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$$

accepting  $coWP(G, Y)$  as follows.

- The state set  $Q$  consists of the following states:
  - the state  $i_0$ ;
  - for every state  $q \in Q'$ ,  $t \in T'$ , and  $u \in Suffix(Z)$ , there is a state  $(q, t, u)$ ;
  - and
  - for every  $t \in T'$ , there is a state  $t$  representing that element. We shall refer to this subset of the state set as  $T'_1$ .
- The input alphabet  $\Sigma = Y$ .
- The check-stack alphabet  $\Delta = \Delta'$ .
- The pushdown stack alphabet  $\Gamma = \Gamma'$ .
- The set of initial states  $I = \{i_0\}$ .
- The set of final states  $F$  consists of the following states:
  - for every  $q_f \in F'$ , and every  $t \in T'$  there is a state  $(q_f, t, \varepsilon)$ , and
  - for every  $t \in T'_1 \setminus \{1_G\} \subseteq Q$ , there is a state representing  $t$  in  $F$ , and we refer to it as  $t$ .
- The regular language  $\mathcal{R} = \mathcal{R}'$ .
- The transition relation  $\delta$  consists of the following transitions:

(T0a) for every  $q_0 \in I'$  there are transitions

$$((i_0, \varepsilon, (\perp, \perp)), ((q_0, 1_G, \varepsilon), \perp));$$

that is, there is an  $\varepsilon$ -transition from  $i_0$  with  $\perp$  at the top of the pushdown stack to  $(q_0, 1_G, \varepsilon)$  without pushing a non-trivial string onto the pushdown stack, for every  $q_0 \in I'$ .

(T0b) There is a transition

$$((i_0, \varepsilon, (\perp, \perp)), (1_G, \perp));$$

that is, there is an  $\varepsilon$ -transition from  $i_0$  with  $\perp$  at the top of the pushdown stack to  $1_G \in T'_1$  without pushing a non-trivial string onto the pushdown stack.

(T1) For every  $q \in Q, t \in T', y \in Y$ , and every pair  $(A, B)$ , there is a transition

$$(((q, t, \varepsilon), y, (A, B)), ((q, t', w_{ty}), B));$$

that is, there is a transition from state  $(q, t, \varepsilon)$  upon reading an input letter  $y \in Y$  to state  $(q, t', w_{ty})$  without editing the stack, such that  $ty =_G w_{ty}t'$ .

(T2) For every  $q \in Q, t \in T', u \in \text{Suffix}(Z)$ , and every pair  $(A, B)$  there is a transition

$$(((q, t, u), \varepsilon, (A, B)), ((q_1, t, u), w))$$

such that  $((q, \varepsilon, (A, B)), (q_1, w)) \in \delta'$ ; that is, there is an  $\varepsilon$ -transition from  $(q, t, u)$  to  $(q_1, t, u)$  writing  $w$  onto the pushdown stack, whenever  $((q, \varepsilon, (A, B)), (q', w)) \in \delta'$ .

(T3) For every  $q \in Q, t \in T', xu \in \text{Suffix}(Z)$  where  $x \in X$ , and every pair  $(A, B)$  there is a transition

$$(((q, t, xu), \varepsilon, (A, B)), ((q_1, t, u), w))$$

such that  $((q, x, (A, B)), (q_1, w)) \in \delta'$ ; that is, there is an  $\varepsilon$ -transition from  $(q, t, xu)$  to  $(q_1, t, u)$  writing  $w$  onto the pushdown stack, whenever  $((q, x, (A, B)), (q_1, w)) \in \delta'$ .

(T4) for every  $t \in T'_1, y \in Y$ , and every pair  $(A, B)$ , there exists a transition

$$((t, y, (A, B)), (t', B)),$$

such that  $ty =_G ht_yt'$ .

### 4.3.2 Proof of Theorem 4.3.1

In this section, we first provide a sequence of technical lemmata. We then present the proof of Theorem 4.3.1 Throughout this section, we shall let  $G, H, X, Y, Z, \mathcal{A}'$  and  $\mathcal{A}$  as in the subsection above.

**Lemma 4.3.4.** Let  $\omega = y_1 y_2 \cdots y_n \in Y^*$ . Further set  $t_0 := 1_G$  and recall that we have fixed strings  $w_{t_{i-1}y_i}$  where  $w_{t_{i-1}y_i} =_H h_{t_{i-1}y_i}$  where  $h_{t_{i-1}y_i}$  are defined by

$$t_{i-1}y_i =_G h_{t_{i-1}y_i} t_i.$$

Further set  $l_i$  to be the length of the string  $w_{t_{i-1}y_i}$ . Then the following hold:

1.  $\omega$  is read by  $\mathcal{A}$  through the chain

$$\begin{aligned} (i_0, (S, \perp)) &\sim_\varepsilon (1_G, (S, \perp)) \sim_{y_1} (t_1, (S, \perp)) \sim_{y_2} (t_2, (S, \perp)) \\ &\quad \cdot \\ &\quad \cdot \\ &\quad \cdot \\ &\quad \sim_{y_3} \cdots \sim_{y_n} (t_n, (S, \perp)), \end{aligned}$$

for any  $S = (a_k, a_{k-1}, \dots, a_1, \perp)$  such that  $a_k a_{k-1} \cdots a_1 \in \mathcal{R}'$ .

2. Suppose  $\mathcal{A}'$  can read  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n} = x_1 x_2 \cdots x_t \in X^*$  through a chain, such as

$$\begin{aligned} (q_0, (S', \perp)) &\sim_\varepsilon^* (q_1, (S', S_1)) \sim_{x_1} (q_2, (S', S_2)) \sim_\varepsilon^* (q_3, (S', S_3)) \\ &\quad \sim_{x_2} (q_4, (S', S_4)) \sim_\varepsilon^* (q_5, (S', S_5)) \\ &\quad \vdots \\ &\quad \sim_{x_t} (q_{2t}, (S', S_{2t})) \sim_\varepsilon^* (q_{2t+1}, (S', S_{2t+1})), \end{aligned}$$

where  $q_0 \in I'$  and  $(S', S_i)$  are stacks for  $1 \leq i \leq 2t+1$  such that  $S' = (b_{k'}, b_{k'-1}, \dots, b_1, \perp)$  for some  $b_{k'} b_{k'-1} \cdots b_1 \in \mathcal{R}'$ . Then  $\mathcal{A}$  can read  $\omega$  through the following chain

$$\begin{aligned} (i_0, (S', \perp)) &\sim_\varepsilon ((q_0, 1_G, \varepsilon), (S', \perp)) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S', S_1)) \\ &\quad \sim_{y_1} ((q_1, t_1, w_{1_G y_1}), (S', S_1)) \\ &\quad \sim_\varepsilon^* ((q_{2l_1+1}, t_1, \varepsilon), (S', S_{2l_1+1})) \\ &\quad \sim_{y_2} ((q_{2l_1+1}, t_2, w_{t_1 y_2}), (S', S_{2l_1+1})) \\ &\quad \sim_\varepsilon^* ((q_{2l_2+1}, t_2, \varepsilon), (S', S_{2l_2+1})) \\ &\quad \vdots \\ &\quad \sim_{y_n} ((q_{2l_{n-1}+1}, t_n, w_{t_{n-1} y_n}), (S', S_{2l_{n-1}+1})) \\ &\quad \sim_\varepsilon^* ((q_{2l_n+1}, t_n, \varepsilon), (S', S_{2l_n+1})). \end{aligned}$$

*Proof.* First we note that initial configurations of  $\mathcal{A}$  are of the form  $(i_0, (S, (\perp)))$  where  $S = (a_k, a_{k-1}, \dots, a_1, \perp)$  for some  $a_k a_{k-1} \cdots a_1 \in \mathcal{R}'$  since  $i_0$  is the only initial state.



1. Observe that the reading-transitions are those in (T1) and (T4). The states in (T4) are those in  $T'_1$ . Further observe that the automaton may use the transition in (T0b) to move from an initial configuration to  $(1_G, (S, (\perp)))$  for any choice of check-stack  $S$  as

$$(i_0, (S, (\perp))) \sim_\varepsilon (1_G, (S, (\perp))). \quad (11.a)$$

Now we note that  $1_G \in T'_1$ , and the only transitions from states in  $T'_1$  are those in (T4), these transitions do not edit the stack. Thus if the automaton used the transition in (T0b), then it must use the transitions

$$((t_{i-1}, y_i, (\perp, \perp)), (t_i, \perp))$$

in (T4) to read  $\omega$  and thus the chain (11.a) continues as

$$\begin{aligned} (i_0, (S, (\perp))) \sim_\varepsilon (1_G, (S, (\perp))) \sim_{y_1} (t_1, (S, (\perp))) \sim_{y_2} (t_2, (S, (\perp))) \\ \vdots \\ \sim_{y_n} (t_n, (S, (\perp))). \end{aligned} \quad (11.b)$$

2. Suppose  $\mathcal{A}'$  can read  $w_{t_0y_1}w_{t_1y_2} \cdots w_{t_{n-1}y_n} = x_1x_2 \cdots x_t \in X^*$  through a chain, such as

$$\begin{aligned} (q_0, (S', (\perp))) \sim_\varepsilon^* (q_1, (S', S_1)) \sim_{x_1} (q_2, (S', S_2)) \sim_\varepsilon^* (q_3, (S', S_3)) \\ \sim_{x_2} (q_4, (S', S_4)) \sim_\varepsilon^* (q_5, (S', S_5)) \\ \vdots \\ \sim_{x_t} (q_{2t}, (S', S_{2t})) \sim_\varepsilon^* (q_{2t+1}, (S', S_{2t+1})), \end{aligned}$$

where  $q_0 \in I'$  and  $(S', S_i)$  are stacks for  $1 \leq i \leq 2t + 1$  for some choice of  $S'$ . (Note a choice of  $S'$  is equivalent to a choice of a string in the regular language  $\mathcal{R}'$  be definition.)

First we observe that the reading transitions are those in (T1) and (T4). The states in (T1) are those in the form  $(q, t, u)$  where  $q \in Q'$ ,  $t \in T'$ , and  $u \in \text{Suffix}(Z)$ . Note  $\mathcal{A}$  may use the transition

$$(i_0, \varepsilon, (\perp, \perp)), ((q_0, 1_G, \varepsilon), \perp)$$

in (T0a) to move from the initial configuration  $(i_0, (S', (\perp)))$  to  $((q_0, 1_G, \varepsilon), (S', (\perp)))$  as

$$(i_0, (S', (\perp))) \sim_\varepsilon ((q_0, 1_G, \varepsilon), (S', (\perp))). \quad (12.a)$$

The automaton  $\mathcal{A}'$  uses a sequence (possibly of length 0) of  $\varepsilon$ -transitions to move the from configuration  $(q_0, (S', (\perp)))$  to  $(q_1, (S', S_1))$ . Let this sequence be denoted by  $E_1$ . The automaton  $\mathcal{A}$  can use a sequence transitions in (T2) that is analogous to  $E_1$ ; i.e, if  $m^{\text{th}}$  transition in the sequence used by  $\mathcal{A}$  is

$$(((q', 1_G, \varepsilon), \varepsilon, (A', B')), ((q'', 1_G, \varepsilon), w))$$

then the  $m^{\text{th}}$  transition in  $E_1$  must be  $((q', \varepsilon, (A', B')), (q'', w))$ . These transitions then must continue the chain (12.a) as follows

$$(i_0, (S', (\perp))) \sim_\varepsilon ((q_0, 1_G, \varepsilon), (S', (\perp))) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S', S_1)). \quad (12.b)$$

Note that the states from which transitions in (T1) are used to read input letters all have  $\varepsilon$  in the third component. Thus the automaton may use a transition from (T1) that moves from state  $(q_1, 1_G, \varepsilon)$  to  $(q_1, t_1, w_{1_G y_1})$ , these transitions do not edit the stack and therefore the stack will be unchanged. Therefore the chain (12.b) will continue as

$$(i_0, (S', (\perp))) \sim_\varepsilon ((q_0, 1_G, \varepsilon), (S', (\perp))) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S', S_1)) \\ \sim_{y_1} ((q_1, t_1, w_{1_G y_1}), (S', S_1)). \quad (12.c)$$

Now since the third component of the last configuration is not  $\varepsilon$ , the automaton cannot use a transition in (T1). Therefore the automaton must use transitions in (T2) and (T3) to continue the chain. We shall describe which ones the automaton uses below.

Suppose  $w_{1_G y_1} = x_1 x_2 \cdots x_{l_1}$ . Now consider the relevant part of the chain by which  $\mathcal{A}'$  reads  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n}$  below.

$$(q_0, (S', (\perp))) \sim_\varepsilon^* (q_1, (S', S_1)) \sim_{x_1} (q_2, (S', S_2)) \sim_\varepsilon^* (q_3, (S', S_3)) \\ \sim_{x_2} (q_4, (S', S_4)) \sim_\varepsilon^* (q_5, (S', S_5)) \\ \vdots \\ \sim_{x_{l_1}} (q_{2l_1}, (S', S_{2l_1})) \sim_\varepsilon^* (q_{2l_1+1}, (S', S_{2l_1+1})).$$

Let the sequence of transitions of  $\mathcal{A}'$  used to move from  $(q_1, (S', S_1))$  to  $(q_{2l_1+1}, (S', S_{2l_1+1}))$  be denoted by  $E_2$ . The automaton  $\mathcal{A}$  can use a sequence of  $\varepsilon$ -transitions from (T2) and (T3) that is analogous to  $E_2$ ; that is, if the  $m^{\text{th}}$  transition in the sequence used by  $\mathcal{A}$  is of the form

$$(((q', t_1, u), \varepsilon, (A', B')), ((q'', t_1, u), w'))$$

then the  $m^{\text{th}}$  transition in  $E_2$  is  $(q', \varepsilon, (A', B')), (q'', w')$ , and if the  $m^{\text{th}}$  transition in the sequence used by  $\mathcal{A}$  is of the form

$$(((q', t_1, xu), \varepsilon, (A', B')), ((q'', t_1, u), w'))$$

then the  $m^{\text{th}}$  transition used in  $E_2$  is  $((q', x, (A', B')), (q'', w'))$ . Note that the sequence of transitions used by  $\mathcal{A}$  will have the same effect on the stack as  $E_2$ . We note that this sequence must progressively reduce the length of the string in the third coordinate of the state since  $E_2$  contains reading-transitions, and therefore this sequence must contain transitions from (T3) that reduce the length of the third coordinate. Therefore the third coordinate will eventually become  $\varepsilon$  using the transitions in the sequence. Therefore using this sequence of  $\varepsilon$ -transitions the chain in (12.c) continues as

$$\begin{aligned} (i_0, (S', (\perp))) &\sim_\varepsilon ((q_0, 1_G, \varepsilon), (S', (\perp))) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S', S_1)) \\ &\sim_{y_1} ((q_1, t_1, w_{1_G y_1}), (S', S_1)) \\ &\sim_\varepsilon^* ((q_{2l_1+1}, t_1, \varepsilon), (S', S_{2l_1+1})). \end{aligned}$$

Since the third coordinate of the state in the last configuration is now  $\varepsilon$ , the automaton can now read another input letter. Continuing in this manner, we see that  $\mathcal{A}$  can read  $\omega$  through the chain in the statement of the lemma. ■

**Lemma 4.3.5.** *Let  $\omega = y_1 y_2 \cdots y_n \in Y^*$ . The only chains by which  $\mathcal{A}$  can read  $\omega$  are those listed in Lemma [4.3.4](#)*

*Proof.* Observe the only reading transitions are those in (T1) and (T4). Further, note that there are no transitions from a state of the form  $(q, t, u)$  to one of the form  $t'$ . Thus if  $\mathcal{A}$  reads  $\omega$  through a chain, then either (apart from the initial state) all states in the configurations of the chain are of the form  $(q, t, u)$  or are in  $T'_1$ .

Suppose the states are in  $T'_1$ . Let  $S$  be any choice of check-stack. (Recall that a choice of check-stack is equivalent to a choice of string from the regular language  $\mathcal{R}$ .) The only transition that can be used from the initial state to one of the states in  $T'_1$  is the one in (T0b). Thus the chain starts as

$$(i_0, (S, (\perp))) \sim_\varepsilon (1_G, (S, (\perp)))$$

Now the only transitions that can be used are those in (T4). Transitions in (T4) do not change the stack. Thus, throughout the chain the pushdown stack must be empty. Further, there are no  $\varepsilon$ -transitions from states in  $T'_1$  thus the automaton must use the transitions

$$((t_{i-1}, y_i, (\perp, \perp)), (t_i, \perp))$$

where  $t_0 = 1_G$  and  $t_{i-1} y_i =_G h_{t_{i-1} y_i} t_i$ . Therefore the chain continues as in 1 in Lemma [4.3.4](#)

Now suppose that the states in the chain are of the form  $(q, t, u)$ . Further suppose we have the check stack being  $S$ . The only transitions from the initial state  $i_0$  to states of the form  $(q, t, u)$  are those in (T0a). These transitions are from  $i_0$  to  $(q_0, 1_G, \varepsilon)$  where  $q_0 \in I'$ . We also note that (T0a) does not change the stack, and so the chain starts as

$$(i_0, (S, (\perp))) \sim_\varepsilon ((q_0, 1_G, \varepsilon), (S, (\perp))). \quad (13.a)$$

Now the automaton may use a sequence of  $\varepsilon$ -transitions  $K_1$  in (T2) to move to a configuration  $C = ((q_1, 1_G, \varepsilon), (S, S_1))$  from which  $\mathcal{A}$  uses a reading-transition. There is an analogous sequence  $K'_1$  of  $\varepsilon$ -transitions in  $\mathcal{A}'$ ; that is, suppose the  $k^{th}$  transition in  $K'_1$  is  $((q', \varepsilon, (A', B')), (q'', w))$  then the  $k^{th}$  transition in  $K_1$  is

$$(((q', 1_G, \varepsilon), \varepsilon, (A', B')), ((q'', 1_G, \varepsilon), w)).$$

Note these transitions have the same effect on the stack of configurations of  $\mathcal{A}'$  as they do on the stack of configurations of  $\mathcal{A}$ . Thus if the sequence of transitions  $K_1$  continues the chain (13.a) as

$$(i_0, (S, (\perp))) \sim_\varepsilon ((q_0, 1_G, \varepsilon), (S, (\perp))) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S, S_1)) \quad (13.b)$$

then the sequence  $K'_1$  results in the following chain of configurations in  $\mathcal{A}'$

$$(q_0, (S, (\perp))) \sim_\varepsilon^* (q_1, (S, S_1)). \quad (14.b)$$

(We note that if  $\mathcal{A}$  does not use a sequence of transitions as mentioned above, then we may simply take  $q_1 = q_0$ .)

Now the automaton must use a transition in (T1), to read the first input letter  $y_1$  and move to the state  $(q_1, t_1, w_{1_G y_1})$  (such that  $1_G y_1 =_G h_{1_G y_1} t_1$ ) without changing the stack and thus the chain (13.b) continues as

$$(i_0, (S, (\perp))) \sim_\varepsilon ((q_0, 1_G, \varepsilon), (S, (\perp))) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S, S_1)) \\ \sim_{y_1} ((q_1, t_1, w_{1_G y_1}), (S, S_1)). \quad (13.c)$$

Now the automaton must use some sequence of  $\varepsilon$ -transitions in (T2) and (T3). We note that transitions in (T3) must be used as they reduce the length of the third component of the state and input letters can only be read when the third component is of length 0. Thus the sequence of transitions used by  $\mathcal{A}$  progressively reduces the length of the third component of the state (except when the transitions are from (T2)). Let  $K_2$  be a sequence of transitions that is used by  $\mathcal{A}$  to move from configuration  $((q_1, t_1, w_{1_G y_1}), (S, S_1))$  to a configuration  $((q_2, t_1, \varepsilon), (S, S_2))$  where we shall assume that  $\mathcal{A}$  will read an input letter from  $((q_2, t_1, \varepsilon), (S, S_2))$ . There is

an analogous sequence of transitions  $K'_2$  of  $\mathcal{A}'$ ; i.e., suppose the  $k^{\text{th}}$  transition in  $K'_2$  is  $((q', \varepsilon, (A', B')), (q'', w))$  then the  $k^{\text{th}}$  transition in  $K_2$  is of the form

$$(((q', t_1, u), \varepsilon, (A', B')), ((q'', t_1, u), w)),$$

and if the  $k^{\text{th}}$  transition in  $K'_2$  is

$$((q', x, (A', B')), (q'', w))$$

then the  $k^{\text{th}}$  transition in  $K_2$  is of the form

$$(((q', t_1, xu), \varepsilon, (A', B')), ((q'', t_1, u), w)).$$

Note these transitions have the same effect on the stack of configurations of  $\mathcal{A}'$  as they do on the stack of configurations of  $\mathcal{A}$ . Thus if the sequence of transitions  $K_2$  continues the chain (13.c) as

$$\begin{aligned} (i_0, (S, (\perp))) &\sim_{\varepsilon} ((q_0, 1_G, \varepsilon), (S, (\perp))) \sim_{\varepsilon}^* ((q_1, 1_G, \varepsilon), (S, S_1)) \\ &\sim_{y_1} ((q_1, t_1, w_{1_G y_1}), (S, S_1)) \\ &\sim_{\varepsilon}^* ((q_2, t_1, \varepsilon), (S, S_2)) \end{aligned} \quad (13.d)$$

then the sequence  $K'_2$  continues the chain (14.b) as

$$\begin{aligned} (q_0, (S, (\perp))) &\sim_{\varepsilon}^* (q_1, (S, S_1)) \sim_{x_1} (p_1, (S, S_{1,2})) \sim_{\varepsilon}^* (p_2, (S, S_{2,2})) \\ &\sim_{x_2} (p_3, (S, S_{3,2})) \sim_{\varepsilon}^* (p_4, (S, S_{4,2})) \\ &\vdots \\ &\sim_{x_{l_1}} (q_{l_1+1}, (S, S_{l_1+1})) \sim_{\varepsilon}^* (q_2, (S, S_2)), \end{aligned} \quad (14.d)$$

where  $(p_i, (S, S_{i,2}))$  are the configurations obtained through the transitions in  $K'_2$ .

Observe that continuing in this way we see if  $\omega$  is read by  $\mathcal{A}$  through a chain whose states are of the form  $(q, t, u)$  (with the exception of the initial configuration) then  $\mathcal{A}'$  can read  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n}$  where the states in the configurations are related in the way outlined above.  $\blacksquare$

We are now ready to present the proof of Theorem [4.3.1](#).

*Proof of Theorem [4.3.1](#).* Let  $\omega = y_1 y_2 \cdots y_n \in Y^*$ . Further set  $t_0 = 1_G$  and recall that we have fixed strings  $w_{t_{i-1} y_i}$  where  $w_{t_{i-1} y_i} =_G h_{t_{i-1} y_i}$  such that  $h_{t_{i-1} y_i}$  are defined by

$$t_{i-1} y_i =_G h_{t_{i-1} y_i} t_i.$$

We note  $\omega =_G h_{t_0y_1} h_{t_1y_2} \cdots h_{t_{n-1}y_n} t_n$ . Thus  $\omega \in coWP(G, Y)$  if and only if  $h_{t_0y_1} h_{t_1y_2} \cdots h_{t_{n-1}y_n} \neq_H 1_H$  or  $t_n \neq 1_G$ .

Suppose  $t_n \neq 1_G$ . By Lemma 4.3.4, we know that the automaton can read  $\omega$  through the following chain

$$\begin{aligned} (i_0, (S, (\perp))) &\sim_\varepsilon (1_G, (S, (\perp))) \sim_{y_1} (t_1, (S, (\perp))) \sim_{y_2} (t_2, (S, (\perp))) \\ &\quad \cdot \\ &\quad \cdot \\ &\quad \cdot \\ &\quad \sim_{y_n} (t_n, (S, (\perp))), \end{aligned}$$

for any choice of check-stack  $S$ .

Since  $t_n \neq 1_G$  and thus is a final state, therefore  $(t_n, (S, (\perp)))$  is an accept configuration and  $\omega$  is accepted.

Now suppose  $h_{t_0y_1} h_{t_1y_2} \cdots h_{t_{n-1}y_n} \neq_H 1_H$ , thus

$$w_{t_0y_1} w_{t_1y_2} \cdots w_{t_{n-1}y_n} \in coWP(G, X).$$

Therefore  $w_{t_0y_1} w_{t_1y_2} \cdots w_{t_{n-1}y_n}$  is accepted by the automaton  $\mathcal{A}'$ . In particular, there is a chain of relations by which  $\mathcal{A}'$  reads  $w_{t_0y_1} w_{t_1y_2} \cdots w_{t_{n-1}y_n}$

$$\begin{aligned} (q_0, (S', (\perp))) &\sim_\varepsilon^* (q_1, (S', S_1)) \sim_{x_1} (q_2, (S', S_2)) \sim_\varepsilon^* (q_3, (S', S_3)) \\ &\sim_{x_2} (q_4, (S', S_4)) \sim_\varepsilon^* (q_5, (S', S_5)) \\ &\quad \vdots \\ &\sim_{x_t} (q_{2t}, (S', S_{2t})) \sim_\varepsilon^* (q_{2t+1}, (S', S_{2t+1})) \end{aligned}$$

where  $q_0 \in I'$  and some choice of choice-stack  $S'$ . Since,  $w_{t_0y_1} w_{t_1y_2} \cdots w_{t_{n-1}y_n} \in \mathcal{L}(\mathcal{A}')$  we may assume  $w_{t_0y_1} w_{t_1y_2} \cdots w_{t_{n-1}y_n}$  is accepted through the chain above and so we can assume  $q_{2t+1} \in F'$ . By Lemma 4.3.4,  $\omega$  is read by  $\mathcal{A}$  by the following chain

$$\begin{aligned} (i_0, (S', (\perp))) &\sim_\varepsilon ((q_0, 1_G, \varepsilon), (S', (\perp))) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S', S_1)) \\ &\sim_{y_1} ((q_1, t_1, w_{1Gy_1}), (S', S_1)) \\ &\sim_\varepsilon^* ((q_{2l_1+1}, t_1, \varepsilon), (S', S_{2l_1+1})) \\ &\sim_{y_2} ((q_{2l_1+1}, t_2, w_{t_1y_2}), (S', S_{2l_1+1})) \\ &\sim_\varepsilon^* ((q_{2l_2+1}, t_2, \varepsilon), (S', S_{2l_2+1})) \\ &\quad \vdots \\ &\sim_{y_n} ((q_{2l_{n-1}+1}, t_n, w_{t_{n-1}y_n}), (S', S_{2l_{n-1}+1})) \\ &\sim_\varepsilon^* ((q_{2l_n+1}, t_n, \varepsilon), (S', S_{2l_n+1})). \end{aligned}$$

Since  $q_{2t+1} \in F'$  the state  $(q_{2l_{n+1}}, t_r, \varepsilon) \in F$  since  $q_{2l_{n+1}} = q_{2t+1}$  and thus  $\omega$  is accepted.

Therefore if  $\omega \in \text{coWP}(G, Y)$  then  $\omega \in \mathcal{L}(\mathcal{A})$ .

We shall now assume  $\omega \in \text{WP}(G, Y)$ . Thus  $\omega \in \text{WP}(G, Y)$  if and only if  $h_{t_0 y_1} h_{t_1 y_2} \cdots h_{t_{n-1} y_n} =_H 1_H$  and  $t_n =_G 1_G$ .

By Lemma 4.3.5, we know that the only way to read  $\omega$  is through chains in listed in Lemma 4.3.4. Thus by chain 1 in Lemma 4.3.4,  $\omega$  is read by  $\mathcal{A}$  through the following chain

$$\begin{aligned} (i_0, (S'', (\perp))) &\sim_\varepsilon (1_G, (S'', (\perp))) \sim_{y_1} (t_1, (S'', (\perp))) \sim_{y_2} (t_2, (S'', (\perp))) \\ &\quad \cdot \\ &\quad \cdot \\ &\quad \cdot \\ &\sim_{y_n} (t_n, (S'', (\perp))) \end{aligned}$$

for any choice of check-stack  $S''$ . As  $t_n =_G 1_G$ ,  $t_n \notin F$  and thus  $\omega$  is not accepted in this run.

By Lemma 4.3.5 a run reading  $\omega$  that is not the above run, must be a run of the form in chain in 2 in Lemma 4.3.4. Suppose  $\mathcal{A}'$  reads  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n}$  through a chain such as

$$\begin{aligned} (q_0, (S''', (\perp))) &\sim_\varepsilon^* (q_1, (S''', S_1)) \sim_{x_1} (q_2, (S''', S_2)) \sim_\varepsilon^* (q_3, (S''', S_3)) \\ &\sim_{x_2} (q_4, (S''', S_4)) \sim_\varepsilon^* (q_5, (S''', S_5)) \\ &\quad \vdots \\ &\sim_{x_t} (q_{2t}, (S''', S_{2t})) \sim_\varepsilon^* (q_{2t+1}, (S''', S_{2t+1})) \end{aligned}$$

for some choice of choice-stack  $S'''$ . Since

$$w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n} =_H h_{t_0 y_1} h_{t_1 y_2} \cdots h_{t_{n-1} y_n}$$

then  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n} \notin \mathcal{L}(\mathcal{A}')$ . Therefore we may assume that whenever  $\mathcal{A}'$  reads  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n}$  then the state of the last configuration is not an accept

state. So  $q_{2t+1} \notin F'$ . By 2 in Lemma 4.3.4,  $\mathcal{A}$  can read  $\omega$  through

$$\begin{aligned}
(i_0, (S''', (\perp))) &\sim_\varepsilon ((q_0, 1_G, \varepsilon), (S''', (\perp))) \sim_\varepsilon^* ((q_1, 1_G, \varepsilon), (S''', S_1)) \\
&\sim_{y_1} ((q_1, t_1, w_{1_G y_1}), (S''', S_1)) \\
&\sim_\varepsilon^* ((q_{2l_1+1}, t_1, \varepsilon), (S''', S_{2l_1+1})) \\
&\sim_{y_2} ((q_{2l_1+1}, t_2, w_{t_1 y_2}), (S''', S_{2l_1+1})) \\
&\sim_\varepsilon^* ((q_{2l_2+1}, t_2, \varepsilon), (S''', S_{2l_2+1})) \\
&\vdots \\
&\sim_{y_n} ((q_{2l_{n-1}+1}, t_n, w_{t_{n-1} y_n}), (S''', S_{2l_{n-1}+1})) \\
&\sim_\varepsilon^* ((q_{2l_n+1}, t_n, \varepsilon), (S''', S_{2l_n+1})).
\end{aligned}$$

Since  $q_{2t+1} \notin F'$  then  $(q_{2l_n+1}, t_n, \varepsilon) \notin F$  (as  $q_{2t+1} = q_{2l_n+1}$ ). Therefore  $\omega$  is not accepted. We note that if  $\mathcal{A}$  reads  $\omega$  through another chain whose states (apart from  $i_0$ ) are of the form  $(q, t, u)$  then the chain induces a read of  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n}$  in  $\mathcal{A}'$  as seen in the proof of Lemma 4.3.5. Therefore it must be that  $\mathcal{A}'$  reads  $w_{t_0 y_1} w_{t_1 y_2} \cdots w_{t_{n-1} y_n}$  through a corresponding chain whenever  $\omega$  is read by  $\mathcal{A}$  in a chain such as the one above whose states (apart from  $i_0$ ) are of the form  $(q, t, u)$ .

Therefore  $\mathcal{L}(\mathcal{A}) = \text{coWP}(G, Y)$ .  $\blacksquare$

## 4.4 Finitely Generated Subgroups

In this section we shall present the following theorem.

**Theorem 4.4.1.** *Let  $G$  be a finitely generated group and  $H$  be a finitely generated subgroup of  $G$ . If  $G$  is co-ETOL, then  $H$  is co-ETOL.*

We prove this theorem (including providing an intuitive explanation of our automaton) following our two-subsection structure as described in the introduction of the chapter.

### 4.4.1 Definition of Automaton

Let  $G$  be a finitely generated co-ETOL group with finite generating set  $X$ . Let  $H$  be a finitely generated subgroup of  $G$ , with finite generating set  $Y$ . Since  $G$  is co-ETOL then there exists a CSPD automaton  $\mathcal{A}_G = (Q_G, \Sigma_G, \Delta_G, \Gamma_G, I_G, F_G, \mathcal{R}_G, \perp, \delta_G)$  accepting  $\text{coWP}(G, X)$ .

**Intuitive Summary 4.4.2.** The check-stack in  $\mathcal{A}_H$  contains a string  $r \in \mathcal{R}_G$  just as the check-stack in  $\mathcal{A}_G$  did. For a string  $\omega \in Y^*$ , the automaton  $\mathcal{A}_H$  simulates a run of  $\mathcal{A}_G$  reading a specific string  $\omega' \in X^*$  such that  $\omega =_G \omega'$  with  $r$  on the



check-stack. (We shall describe the process of obtaining  $\omega'$  later.) If an accept state is reached in a simulated run, then  $\omega$  is accepted. We note that if  $\omega \neq_H 1_H$  then  $\omega' \neq_G 1_G$ . Thus there is a run of  $\mathcal{A}_G$  reading  $\omega'$  that ends in an accept state. When  $\mathcal{A}_H$  simulates this run with an appropriate string from  $\mathcal{R}_G$  on the check-stack, the run will end in an accept state thereby accepting  $\omega$ .  $\blacklozenge$

First, for every  $y \in Y$  we shall fix a string  $w_y = x_1x_2 \cdots x_r \in X^*$  such that  $w_y =_G y$ . Let  $Z$  denote the set  $\{w_y \mid y \in Y\}$ . In what follows, we shall use Definition [4.3.3](#)

**Intuitive Idea:** The check-stack contains a string  $r \in \mathcal{R}_G$  just as the check-stack in  $\mathcal{A}_G$  did. The state set  $Q_H$  consists of the following states:

- for every  $q \in Q_G$  and every string  $u \in \text{Suffix}(Z)$  there is a state  $(q, u)$ .

The initial states are the states  $(q_0, \varepsilon)$  where  $q_0 \in \mathcal{A}_G$ .

Let  $\omega = y_1y_2 \cdots y_n \in Y^*$ . Note by definition of the strings  $w_y$  we see that  $\omega =_G w_{y_1}w_{y_2}w_{y_3} \cdots w_{y_n}$ . (The string  $w_{y_1}w_{y_2}w_{y_3} \cdots w_{y_n}$  is the string we called  $\omega'$  in the intuitive summary above.) The automaton  $\mathcal{A}_H$  reads a string  $\omega$  in a similar way to how the “Simulation of  $H$ -part” worked in [4.3.1](#). Namely, we can read input letters from states of the form  $(q, \varepsilon)$ . Suppose the stack is  $(S_1, S_2)$  upon reading a letter  $y$  from state  $(q, \varepsilon)$  with  $B$  at the top of the pushdown stack with  $A$  being the corresponding letter on the check stack, the automaton  $\mathcal{A}_H$  moves to the state  $(q, w_y)$  without editing the pushdown stack. This is reflected in transitions in (T1) below. The automaton  $\mathcal{A}_H$  then uses  $\varepsilon$ -transitions ending at a state  $(q', \varepsilon)$ , these transitions simulate the automaton  $\mathcal{A}_G$  reading the string  $w_y$  from state  $q$  with the stack being  $(S_1, S_2)$  to state  $q'$ . This is reflected in transitions in (T2) and (T3) below. In particular transitions in (T2) simulate  $\varepsilon$ -transitions in  $\mathcal{A}_G$  as the second component of the state remains the same after the transition is made. Conversely, transitions in (T3) simulate reading-transition in  $\mathcal{A}_G$  as the second component of the state reduces in length by 1. Finally, as we can only read input letters in  $\mathcal{A}_H$  from states whose second component is  $\varepsilon$ , it must be that the transitions that were used resulted in reducing the length of the second component until it reached 0. This ensures that we simulate reading the entirety of  $w_y$  in  $\mathcal{A}_G$  before reading another input letter in  $\mathcal{A}_H$ . When the second component of the state is  $\varepsilon$ , we can then read another input letter using transitions in (T1), or we can use  $\varepsilon$ -transitions in (T2) to simulate  $\varepsilon$ -transitions in  $\mathcal{A}_G$  before reading another input letter. By the definition of transitions in (T2) and (T3) we see that a transition  $\alpha_H = (((q, u), \varepsilon, (A, B)), ((q', v), w)) \in \delta_H$  precisely when there is a corresponding transition  $\alpha_G = ((q, x, (A, B)), (q', w)) \in \delta_G$  where  $x \in X$  if  $u = xw$ , and  $x = \varepsilon$  if  $u = v$ . This link between the transitions of  $\mathcal{A}_G$  and  $\mathcal{A}_H$  is why it permissible for us to use the word “simulate” in our above description.

We note that we write things as a list below even though there sometimes is one item in the list. We do this to be consistent in our presentation.

**Formal Definition of  $\mathcal{A}_H$ :** We define a *CSPD* automaton

$$\mathcal{A}_H := (Q_H, \Sigma_H, \Delta_H, \Gamma_H, I_H, F_H, \mathcal{R}_H, \perp, \delta_H)$$

accepting  $coWP(H, Y)$  as follows.

- The state set  $Q_H$  consists of the followings states:
  - for every  $q \in Q_G$  and  $u \in Suffix(Z)$ , there is a state  $(q, u)$ .
- The input alphabet  $\Sigma_H = Y$ .
- The check-stack alphabet  $\Delta_H = \Delta_G$ .
- The pushdown stack alphabet  $\Gamma_H = \Gamma_G$ .
- The set of initial states  $I_H$  consists of the following states:
  - for every  $q_0 \in Q_G$ , there is a state  $(q_0, \varepsilon)$ .
- The set of final states  $F_H$  consists of the following states:
  - for every  $q_f \in F_G$ , there is a state  $(q_f, \varepsilon)$ .
- The regular language  $\mathcal{R}_H = \mathcal{R}_G$ .
- The transition relation  $\delta_H$  consists of the following transitions:

(T1) For every  $q \in Q_G, y \in Y$ , and every pair  $(A, B)$ , there is a transition

$$(((q, \varepsilon), y, (A, B)), ((q, w_y), B));$$

that is, there is a transition from state  $(q, \varepsilon)$  upon reading an input letter  $y \in Y$  to state  $(q, w_y)$  without editing the stack.

(T2) For every  $q \in Q_G, u \in Suffix(Z)$ , and every pair  $(A, B)$  there is a transition

$$(((q, u), \varepsilon, (A, B)), ((q_1, u), w))$$

such that  $((q, \varepsilon, (A, B)), (q_1, w)) \in \delta_G$ ; that is, there is an  $\varepsilon$ -transition from  $(q, u)$  to  $(q_1, u)$  writing  $w$  onto the pushdown stack, whenever  $((q, \varepsilon, (A, B)), (q', w)) \in \delta_G$ .

(T3) For every  $q \in Q_G, xu \in \text{Suffix}(Z)$  where  $x \in X$ , and every pair  $(A, B)$  there is a transition

$$(((q, xu), \varepsilon, (A, B)), ((q_1, u), w))$$

such that  $((q, x, (A, B)), (q_1, w)) \in \delta_G$ ; that is, there is an  $\varepsilon$ -transition from  $(q, xu)$  to  $(q_1, u)$  writing  $w$  onto the pushdown stack, whenever  $((q, x, (A, B)), (q_1, w)) \in \delta_G$ .

#### 4.4.2 Proof of Theorem [4.4.1](#)

In this section, we first provide a sequence of technical lemmata. We then present the proof of Theorem [4.4.1](#). Throughout this subsection, we shall let  $G, H, X, Y, Z, \mathcal{A}_G$  and  $\mathcal{A}_H$  as in the subsection above.

**Lemma 4.4.3.** *Let  $\omega = y_1 y_2 \cdots y_n \in Y^*$ . Recall that we have fixed strings  $w_{y_i}$  such that  $y_i =_G w_{y_i}$ . Set  $l_i$  to be the length of the string  $w_{y_i}$ .*

*Suppose  $\mathcal{A}_G$  reads  $\omega' := w_{y_1} w_{y_2} \cdots w_{y_n} = x_1 x_2 \cdots x_t$  through a chain such as*

$$\begin{aligned} (q_0, (S', (\perp))) &\sim_{\varepsilon}^* (q_1, (S', S_1)) \sim_{x_1} (q_2, (S', S_2)) \sim_{\varepsilon}^* (q_3, (S', S_3)) \\ &\sim_{x_2} (q_4, (S', S_4)) \sim_{\varepsilon}^* (q_5, (S', S_5)) \\ &\vdots \\ &\sim_{x_t} (q_{2t}, (S', S_{2t})) \sim_{\varepsilon}^* (q_{2t+1}, (S', S_{2t+1})), \end{aligned}$$

where  $q_0 \in I_G$  and  $(S', S_i)$  are stacks for  $1 \leq i \leq 2t + 1$  and some choice of check-stack  $S'$ . Then  $\mathcal{A}_H$  can read  $\omega$  through the following chain

$$\begin{aligned} ((q_0, \varepsilon), (S', (\perp))) &\sim_{\varepsilon}^* ((q_1, \varepsilon), (S', S_1)) \\ &\sim_{y_1} ((q_1, w_{y_1}), (S', S_1)) \\ &\sim_{\varepsilon}^* ((q_{2l_1+1}, \varepsilon), (S', S_{2l_1+1})) \\ &\sim_{y_2} ((q_{2l_1+1}, w_{y_2}), (S', S_{2l_1+1})) \\ &\sim_{\varepsilon}^* ((q_{2l_2+1}, \varepsilon), (S', S_{2l_2+1})) \\ &\vdots \\ &\sim_{y_n} ((q_{2l_{n-1}+1}, w_{y_n}), (S', S_{2l_{n-1}+1})) \\ &\sim_{\varepsilon}^* ((q_{2l_n+1}, \varepsilon), (S', S_{2l_n+1})). \end{aligned}$$

*Proof.* First observe that the reading-transitions only occur from states whose second component is  $\varepsilon$ . In particular the reading transitions are those in (T1).

We shall analyse the chain by which  $\mathcal{A}_G$  reads  $\omega_1$ . The automaton  $\mathcal{A}_G$  uses a sequence (possibly of length 0) of  $\varepsilon$ -transitions to move from an initial configuration

$(i_0, (S', (\perp)))$  to  $(q_1, (S', S_1))$ . Let this sequence be denoted by  $E_1$ . The automaton  $\mathcal{A}_H$  can use a sequence transitions in (T2) that is analogous to  $E_1$ ; i.e., if the  $m^{\text{th}}$  transition in the sequence used by  $\mathcal{A}_H$  is

$$(((q', \varepsilon), \varepsilon, (A', B')), ((q'', \varepsilon), w))$$

then the  $m^{\text{th}}$  transition in  $E_1$  must be  $((q', \varepsilon, (A', B')), (q'', w))$ . These transitions yield

$$((q_0, \varepsilon), (S', (\perp))) \sim_{\varepsilon}^* ((q_1, \varepsilon), (S', S_1)). \quad (15.a)$$

Now the automaton can use a transition from (T1) to read the first input letter  $y_1$  to move the state from  $(q_1, \varepsilon)$  to  $(q_1, w_{y_1})$ . Recall transitions in (T1) do not edit the stack, therefore the chain (15.a) continues as

$$\begin{aligned} ((q_0, \varepsilon), (S', (\perp))) &\sim_{\varepsilon}^* ((q_1, \varepsilon), (S', S_1)) \\ &\sim_{y_1} ((q_1, w_{y_1}), (S', S_1)). \end{aligned} \quad (15.b)$$

Now since the second component of the last state in (15.b) is not  $\varepsilon$ ,  $\mathcal{A}_H$  cannot use any transition in (T1). Therefore  $\mathcal{A}_H$  must use transitions in (T2) and (T3) to continue the chain. We shall describe which ones the automaton uses below.

Suppose  $w_{y_1} = x_1 x_2 \cdots x_{l_1}$ . Now consider the relevant part of the chain by which  $\mathcal{A}_G$  reads  $\omega_1$  below.

$$\begin{aligned} (q_0, (S', (\perp))) &\sim_{\varepsilon}^* (q_1, (S', S_1)) \sim_{x_1} (q_2, (S', S_2)) \sim_{\varepsilon}^* (q_3, (S', S_3)) \\ &\sim_{x_2} (q_4, (S', S_4)) \sim_{\varepsilon}^* (q_5, (S', S_5)) \\ &\vdots \\ &\sim_{x_{l_1}} (q_{2l_1}, (S', S_{2l_1})) \sim_{\varepsilon}^* (q_{2l_1+1}, (S', S_{2l_1+1})). \end{aligned}$$

Let the sequence of transitions of  $\mathcal{A}_G$  used to move from  $(q_1, (S', S_1))$  to  $(q_{2l_1+1}, (S', S_{2l_1+1}))$  be denoted by  $E_2$ . The automaton  $\mathcal{A}_H$  can use a sequence of  $\varepsilon$ -transitions from (T2) and (T3) that is analogous to  $E_2$ ; that is, if the  $m^{\text{th}}$  transition in the sequence used by  $\mathcal{A}_H$  is of the form

$$(((q', u), \varepsilon, (A', B')), ((q'', u), w'))$$

then the  $m^{\text{th}}$  transition in  $E_2$  is  $((q', \varepsilon, (A', B')), (q'', w'))$ , and if the  $m^{\text{th}}$  transition in the sequence used by  $\mathcal{A}_H$  is of the form

$$(((q', xu), \varepsilon, (A', B')), ((q'', u), w'))$$

then the  $m^{\text{th}}$  transition used in  $E_2$  is  $((q', x, (A', B')), (q'', w'))$ . Note that the sequence of transitions used by  $\mathcal{A}_H$  will have the same effect on the stack as

$E_2$ . We also note that this sequence must progressively reduce the length of the string in the second coordinate of the state since  $E_2$  contains reading transitions, and therefore this sequence must contain transitions from (T3) that reduce the length of the second coordinate. Therefore the second coordinate will eventually become  $\varepsilon$  using the transitions in this sequence. Therefore using this sequence of  $\varepsilon$ -transitions the chain in (15.b) continues as

$$\begin{aligned} ((q_0, \varepsilon), (S', (\perp))) &\sim_{\varepsilon}^* ((q_1, \varepsilon), (S', S_1)) \\ &\sim_{y_1} ((q_1, w_{y_1}), (S', S_1)) \\ &\sim_{\varepsilon}^* ((q_{2l_1+1}, \varepsilon), (S', S_{2l_1+1})). \end{aligned}$$

Since the second coordinate of the state in the last configuration is now  $\varepsilon$ , the automaton  $\mathcal{A}_H$  can now read another input letter. Continuing in this manner, we that  $\mathcal{A}_H$  can read  $\omega$  through the chain in the statement of the lemma.  $\blacksquare$

**Lemma 4.4.4.** *Let  $\omega = y_1 y_2 \cdots y_n \in Y^*$ . The only chains by which  $\mathcal{A}_H$  can read  $\omega$  are those in Lemma [4.4.3](#).*

*Proof.* Suppose  $\mathcal{A}_H$  reads  $\omega$  through a chain  $\mathcal{C}$  where the check-stack is  $S'_1$ . In what follows we shall determine what transitions must be applied and the order in which they can be applied concluding that if  $\mathcal{A}_H$  reads  $\omega$  through  $\mathcal{C}$  then it must be that  $\mathcal{C}$  was induced by a chain of configurations of  $\mathcal{A}_G$  in sense of Lemma [4.4.3](#).

First we observe that for any  $q_0 \in I_G$ , any state of the form  $(q_0, \varepsilon)$  is an initial state of  $\mathcal{A}_H$ . Thus an initial configuration of  $\mathcal{A}_H$  is one of the form

$$((q_0, \varepsilon), (S, (\perp)))$$

where  $q_0 \in I_G$  and choice-stack  $S$ . For the chain  $\mathcal{C}$  the choice-stack is  $S'_1$ .

The automaton  $\mathcal{A}_H$  may use a sequence of  $\varepsilon$ -transitions  $K_1$  in (T2) to move to a configuration  $C = ((q_1, \varepsilon), (S'_1, S_1))$  from which  $\mathcal{A}_H$  uses a reading-transition. There is an analogous sequence  $K'_1$  of  $\varepsilon$ -transitions in  $\mathcal{A}_G$ ; that is, suppose the  $k^{\text{th}}$  transition in  $K'_1$  is  $((q', \varepsilon, (A', B')), (q'', w))$  then the  $k^{\text{th}}$  transition in  $K_1$  is

$$(((q', \varepsilon), \varepsilon, (A', B')), ((q'', \varepsilon), w)).$$

Note transitions in  $K'_1$  have the same effect on the stack of configurations of  $\mathcal{A}_G$  as those in  $K_1$  have on the stack of configurations of  $\mathcal{A}_H$ . Thus if the sequence of transitions  $K_1$  yields

$$((q_0, \varepsilon)(S'_1, (\perp))) \sim_{\varepsilon}^* ((q_1, \varepsilon), (S'_1, S_1)) \tag{16.a}$$

then the sequence  $K'_1$  results in the following chain in  $\mathcal{A}_G$

$$(q_0, (S'_1, (\perp))) \sim_{\varepsilon}^* (q_1, (S'_1, S_1)). \tag{17.a}$$

(We note that if  $\mathcal{A}_H$  does not use a sequence of transitions as mentioned above, then we simply take  $q_1 = q_0$ .)

Now the automaton must use a transition in (T1), to read the first input letter  $y_1$  and move to the state  $(q_1, w_{y_1})$  without changing the stack and thus the chain (16.a) continues as

$$\begin{aligned} ((q_0, \varepsilon), (S'_1, (\perp))) &\sim_{\varepsilon}^* ((q_1, \varepsilon), (S'_1, S_1)) \\ &\sim_{y_1} ((q_1, w_{y_1}), (S'_1, S_1)). \end{aligned} \quad (16.b)$$

Now the automaton  $\mathcal{A}_H$  must use some sequence of  $\varepsilon$ -transitions in (T2) and (T3), as the second component of the state in the last configuration above is not  $\varepsilon$ . We note that transitions in (T3) must be used as they reduce the length of the second component of the state and input letters can only be read when the second component is of length 0. Thus the sequence of transitions used by  $\mathcal{A}_H$  progressively reduces the length of the second component of the state (except when the transitions are from (T2)).

Let  $K_2$  be a sequence of transitions that is used by  $\mathcal{A}_H$  to move from configuration  $((q_1, w_{y_1}), (S'_1, S_1))$  to a configuration  $((q_2, \varepsilon), (S'_1, S_2))$  where we shall assume that  $\mathcal{A}_H$  will read an input letter from  $((q_2, t_1, \varepsilon), (S'_1, S_2))$ . There is an analogous sequence of transitions  $K'_2$  of  $\mathcal{A}_G$ ; i.e., suppose the  $k^{\text{th}}$  transition in  $K'_2$  is  $((q', \varepsilon, (A', B')), (q'', w))$  then the  $k^{\text{th}}$  transition in  $K_2$  is of the form

$$(((q', u), \varepsilon, (A', B')), ((q'', u), w)),$$

and if the  $k^{\text{th}}$  transition in  $K'_2$  is

$$((q', x, (A', B')), (q'', w))$$

then the  $k^{\text{th}}$  transition in  $K_2$  is of the form

$$(((q', xu), \varepsilon, (A', B')), ((q'', u), w)).$$

Note the transitions in  $K'_2$  have the same effect on the stack of a configuration of  $\mathcal{A}_G$  as those of  $K_2$  do on the stack of a configuration of  $\mathcal{A}_H$ . Thus if the sequence of transitions  $K_2$  continues the chain (16.b) as

$$\begin{aligned} ((q_0, \varepsilon), (S'_1, (\perp))) &\sim_{\varepsilon}^* ((q_1, \varepsilon), (S'_1, S_1)) \\ &\sim_{y_1} ((q_1, w_{y_1}), (S'_1, S_1)) \\ &\sim_{\varepsilon}^* ((q_2, \varepsilon), (S'_1, S_2)) \end{aligned} \quad (16.c)$$

then the sequence of transitions  $K'_2$  continues the chain (17.a) as

$$\begin{aligned} (q_0, (S'_1, (\perp))) &\sim_{\varepsilon}^* (q_1, (S'_1, S_1)) \sim_{x_1} (p_1, (S'_1, S_{1,2})) \sim_{\varepsilon}^* (p_2, (S'_1, S_{2,2})) \\ &\sim_{x_2} (p_3, (S'_1, S_{3,2})) \sim_{\varepsilon}^* (p_4, (S'_1, S_{4,2})) \\ &\vdots \\ &\sim_{x_{l_1}} (q_{l_1+1}, (S'_1, S_{l_1+1})) \sim_{\varepsilon}^* (q_2, (S'_1, S_2)), \end{aligned} \quad (17.b)$$

where  $(p_i, (S'_1, S_{i,2}))$  are the configurations obtained through the transitions in  $K'_2$ .

Observe that continuing in this way we see if  $\omega$  is read by  $\mathcal{A}_H$  through a chain then  $\mathcal{A}_G$  can read  $w_{y_1}w_{y_2}\cdots w_{y_n}$  where the states in the configurations are related in the way outlined above.  $\blacksquare$

We are now ready to present the proof of Theorem [4.4.1](#).

*Proof of Theorem [4.4.1](#).* Let  $\omega = y_1y_2\cdots y_n \in Y^*$ . Recall that we have fixed strings  $w_y \in X^*$  for every  $y \in Y$  such that  $y =_G w_y$ . Observe that  $y_1y_2\cdots y_n =_G w_{y_1}w_{y_2}\cdots w_{y_n}$  therefore

$$\omega \in coWP(H, Y) \text{ if and only if } w_{y_1}w_{y_2}\cdots w_{y_n} \in coWP(G, X).$$

Suppose  $\omega \in coWP(H, Y)$  then  $w_{y_1}w_{y_2}\cdots w_{y_n} = x_1x_2\cdots x_t \in coWP(G, X)$ . Thus there exists a chain  $\mathcal{C}$  below by which  $\mathcal{A}_G$  accepts  $w_{y_1}w_{y_2}\cdots w_{y_n} \in coWP(G, X)$ ,

$$\begin{aligned} (q_0, (S', (\perp))) &\sim_{\varepsilon}^* (q_1, (S', S_1)) \sim_{x_1} (q_2, (S', S_2)) \sim_{\varepsilon}^* (q_3, (S', S_3)) \\ &\sim_{x_2} (q_4, (S', S_4)) \sim_{\varepsilon}^* (q_5, (S', S_5)) \\ &\vdots \\ &\sim_{x_t} (q_{2t}, (S', S_{2t})) \sim_{\varepsilon}^* (q_{2t+1}, (S', S_{2t+1})), \end{aligned}$$

where  $(q_{2t+1}, (S', S_{2t+1}))$  is an accept configuration, and  $S'$  is some choice of check-stack.

Now, set  $l_i$  to be the length of the string  $w_{y_i}$ . Then by Lemma [4.4.3](#), we see that  $\mathcal{A}$  reads  $\omega$  through the following chain

$$\begin{aligned} ((q_0, \varepsilon), (S', (\perp))) &\sim_{\varepsilon}^* ((q_1, \varepsilon), (S', S_1)) \\ &\sim_{y_1} ((q_1, w_{y_1}), (S', S_1)) \\ &\sim_{\varepsilon}^* ((q_{2l_1+1}, \varepsilon), (S', S_{2l_1+1})) \\ &\sim_{y_2} ((q_{2l_1+1}, w_{y_2}), (S', S_{2l_1+1})) \\ &\sim_{\varepsilon}^* ((q_{2l_2+1}, \varepsilon), (S', S_{2l_2+1})) \\ &\vdots \\ &\sim_{y_n} ((q_{2l_{n-1}+1}, w_{y_n}), (S', S_{2l_{n-1}+1})) \\ &\sim_{\varepsilon}^* ((q_{2l_n+1}, \varepsilon), (S', S_{2l_n+1})). \end{aligned}$$

Since  $(q_{2t+1}, (S', S_{2t+1}))$  is an accept configuration then it must be that  $q_{2t+1} \in F_G$ . Further,  $q_{2t+1} = q_{2l_n+1}$  therefore  $(q_{2l_n+1}, \varepsilon)$  is an accept state and therefore  $\omega$  is accepted by  $\mathcal{A}_H$ .

Suppose  $\omega \in WP(H, Y)$  then  $w_{y_1}w_{y_2} \cdots w_{y_n} \in WP(G, X)$ . Then for any choice of choice stack  $S''$  if  $\mathcal{A}_G$  reads  $w_{y_1}w_{y_2} \cdots w_{y_n}$  through a chain such as

$$\begin{aligned} (q_0, (S'', (\perp))) &\sim_{\varepsilon}^* (q_1, (S'', S_1)) \sim_{x_1} (q_2, (S'', S_2)) \sim_{\varepsilon}^* (q_3, (S'', S_3)) \\ &\sim_{x_2} (q_4, (S'', S_4)) \sim_{\varepsilon}^* (q_5, (S'', S_5)) \\ &\vdots \\ &\sim_{x_t} (q_{2t}, (S'', S_{2t})) \sim_{\varepsilon}^* (q_{2t+1}, (S'', S_{2t+1})), \end{aligned}$$

then by Lemma [4.4.3](#),  $\mathcal{A}$  reads  $\omega$  through the chain

$$\begin{aligned} ((q_0, \varepsilon), (S'', (\perp))) &\sim_{\varepsilon}^* ((q_1, \varepsilon), (S'', S_1)) \\ &\sim_{y_1} ((q_1, w_{y_1}), (S'', S_1)) \\ &\sim_{\varepsilon}^* ((q_{2l_1+1}, \varepsilon), (S'', S_{2l_1+1})) \\ &\sim_{y_2} ((q_{2l_1+1}, w_{y_2}), (S'', S_{2l_1+1})) \\ &\sim_{\varepsilon}^* ((q_{2l_2+1}, \varepsilon), (S'', S_{2l_2+1})) \\ &\vdots \\ &\sim_{y_n} ((q_{2l_{n-1}+1}, w_{y_n}), (S'', S_{2l_{n-1}+1})) \\ &\sim_{\varepsilon}^* ((q_{2l_n+1}, \varepsilon), (S'', S_{2l_n+1})). \end{aligned}$$

Since  $w_{y_1}w_{y_2} \cdots w_{y_n} \in WP(G, X)$ ,  $q_{2t+1}$  is not an accept state. Therefore as  $q_{2l_n+1} = q_{2t+1}, (q_{2l_n+1}, \varepsilon)$  is also not an accept state. Therefore  $\omega$  is not accepted. Further note that by Lemma [4.4.4](#), there is no other way to read a string  $\omega$ ; i.e., for a string  $\omega$  to be read it must be that the chain by which  $\omega$  is read is induced by a chain in  $\mathcal{A}_G$  reading  $w_{y_1}w_{y_2} \cdots w_{y_n}$  as in Lemma [4.4.3](#). Therefore if  $\omega \in WP(H, Y)$ , then  $\omega$  is not accepted by  $\mathcal{A}_H$ .  $\blacksquare$

## 4.5 Wreath Products With Virtually Free Top Groups

In this section we shall present the following theorem.

**Theorem 4.5.1.** *Let  $B$  be a co-ETOL group and let  $T$  be a finitely generated virtually free group. Then the standard restricted wreath product  $W = B \wr T$  is co-ETOL.*

We prove this theorem (including providing an intuitive explanation of our automaton) following our two-subsection structure as described in the introduction of the chapter. That is, we follow the same structure as we have done in previous



sections of this chapter. Similar to Lemma 7 of [28], our proof is based on the techniques in the proof of Theorem 10 of [27].

However, we shall first define some notation that will be helpful to us in the rest of the chapter.

**Definition 4.5.2.** Let  $X$  be a set. Let  $v \in X^*$  and let  $x \in X$  be the last letter of  $v$ . We define  $v - x$  to be the unique prefix  $u$  of  $v$  such that  $v = ux$ , and we write  $v - x = u$ . ♣

### 4.5.1 Definition of Automaton

In this section we shall define an automaton  $\mathcal{A}$  accepting the co-word problem of  $W$ .

Let  $X_B$  be a finite symmetrically closed generating set for  $B$ . Since  $B$  is co-ETOL, there is a CSPD automaton

$$\mathcal{A}_B = (Q_B, \Sigma_B, \Delta_B, \Gamma_B, \mathcal{R}_B, I_B, F_B, \perp, \delta_B)$$

accepting  $coWP(B, X_B)$ .

Let  $X_T$  be a finite symmetrically closed generating set for  $T$ , such that  $X_T$  is a union of a symmetrically closed generating set of its free subgroup of finite index and the symmetric closure of a transversal set  $T'$  which contains  $1_T$ . We shall denote the finite index free subgroup of  $T$  by  $F_n$ .

Since  $T$  is virtually free, by Theorem [2.3.8] there exists a pushdown automaton accepting  $WP(T, X_T)$  as in [2.3.2]. We shall use the automaton  $P_G^{(eW,1)}$  defined in [2.3.2] in the construction of the automaton accepting the co-word problem of  $W$ . For the purposes of this section, denote  $P_G^{(eW,1)}$  by

$$\mathcal{A}_T = (Q_T, I_T, \Sigma_T, \chi_T, \perp, \delta_T, F_T).$$

Recall that the input alphabet  $\Sigma_T = X_T$ , and as in the construction in [2.3.2]  $\chi_T$  is the generating set we shall use for  $F_n$ .

**Intuitive Summary 4.5.3.** Recall that the standard restricted wreath product  $W = B \wr T$  is a semi-direct product. Thus every element  $g \in W$  may be expressed as  $xt$  where  $t \in T$  and  $x \in \bigoplus_{t \in T} B_t$  (where  $B_t \cong B$  for every  $t \in T$ ).

The check-stack of  $\mathcal{A}$  will consist of a string  $r \in \mathcal{R}_B$  followed by the letter  $P$  some non-zero number of times. Let  $\omega$  be an input string and let  $g \in W$  be such that  $g =_W \omega$ . We may assume that  $g = xt$  as above. We note that if  $\omega \neq_W 1_W$  then either  $x$  is non-trivial in the base group, or  $t$  is non-trivial in the top group. The automaton will non-deterministically choose between two ways of processing  $\omega$ , reflecting the semi-direct product structure of  $W$ . The first way is determining

whether  $t$  is non-trivial. This is done by simulating the pushdown automaton  $\mathcal{A}_T$  reading  $\omega$  while ignoring letters from  $X_B$  (as they project trivially to the top group). We note that we may assume that the check-stack is long enough for this calculation to take place as the associated regular language allows for an arbitrarily large number of occurrences of the letter  $P$  after a string from  $\mathcal{R}_B$ .

The second way of processing  $\omega$  is by determining whether  $x$  is non-trivial in the base group. The automaton non-deterministically chooses and locates a coordinate of  $x$  to check since if  $x$  is non-trivial, then there is a coordinate at which the entry of  $x$  is non-trivial in  $B$ . The automaton chooses and locates the coordinate by simulating a modified version of  $P_T^{(W,1)}$  non-deterministically. (Note the use of non-determinism ensures that there is some run that will check the correct coordinate, if such a coordinate exists.) This is done in more detail in the loading and coordinate location stages below. We note that while simulating  $P_T^{(W,1)}$  or a modified version of it, letters from  $X_B$  are ignored as they do not impact the chosen coordinate of  $x$ . The automaton then simulates  $\mathcal{A}_B$  to read input letters that impact the chosen coordinate. We note that this construction is more complicated than simply simulating  $\mathcal{A}_B$  as upon reading a letter from  $X_T$  the coordinate under consideration changes. The coordinate under consideration returns to the chosen coordinate when the product of letters read from  $X_T$  are trivial in  $T$ . We deal with this by using  $P_T^{(W,1)}$ . (Again, while simulating  $P_T^{(W,1)}$  the automaton ignores letters from  $X_B$  as they do not have an impact on the chosen coordinate.) The automaton oscillates between simulating  $\mathcal{A}_B$  (from where its processing was paused) and  $P_T^{(W,1)}$  until the input string is read (regardless of whether  $P_T^{(W,1)}$  was left in an accept state). At the end of processing if  $\mathcal{A}_B$  was left in an accept state then  $\omega$  is accepted. Indeed if  $\omega$  is accepted through this way of processing, then there is a coordinate of  $x$  such that the only letters of  $\omega$  impacting that coordinate form a non-trivial product in  $B$ .  $\blacklozenge$

**Remark 4.5.4.** We note that at any point when  $\mathcal{A}_B$  is being simulated there are no letters from  $P_T^{(W,1)}$  on the pushdown stack, thus ensuring that the simulation of  $\mathcal{A}_B$  is correct. We also note that no transitions will be added to  $\mathcal{A}_B$  to enable a transition when the pointer on the check-stack is at  $P$ . Further, the addition of extra  $P$  letters ensures that  $P_T^{(W,1)}$  will always complete its computation and not be cut off due to insufficient length on the check-stack. At any point in the second way of processing, the pushdown stack is used to both simulate the pushdown stack of  $\mathcal{A}_B$  as well as keep track of the coordinate of  $x$  that is currently active, with the top of the stack serving one purpose or another depending on where the machine is in its processing. The simulation of  $\mathcal{A}_B$  pauses when tracking the currently active coordinate of  $x$  starts. Tracking the coordinate of  $x$  ends when the coordinate is the same as the one originally chosen, and then the simulation of  $\mathcal{A}_B$  resumes.  $\spadesuit$

Now recall that  $Q_T$  consists of the following states:

- for every element  $t$  in the transversal  $T'$  we shall have a state  $t$  representing that element. We shall refer to this subset of the state set as  $T'$  as well.
- The states  $q_A$  and  $\Psi$ .
- For ever  $t \in T'$  and  $y \in X_T$ , there is a state  $(t, y)$ . We shall call this collection of states  $K$ .
- Recall that for every  $(t, y) \in K$  we have fixed a string  $w_{ty} = x_1x_2 \cdots x_r$  (in [2.3.2](#)) such that  $ty =_G w_{ty}t'$  for some  $t' \in T'$ . There is a state representing every letter in the string  $w_{ty}$  and is further indexed by  $(t, y)$ . That is, for every  $(t, y) \in K$  there exist states named  $q_{(t,y,1)}, q_{(t,y,2)}, \dots, q_{(t,y,r)}$  where  $r$  depends only on  $(t, y)$ . We shall call this collection of states  $K_1$ .

We shall use the states in  $T'$  as well as those in  $K_1$  in multiple places in our construction below. We recommend that the reader recall the constructions of  $P_{F_n}^{(W,1)}$  and  $P_T^{(W,1)}$  in [2.3.2](#) as we rely on them rather heavily in the constructing the automaton  $\mathcal{A}$  below. However, we remind the reader of a few key points that are crucial to the construction of  $P_T^{(W,1)}$ .

1. There is a unique initial state  $1_T$ .
2. There is a unique final state  $q_A$ .
3. A string is accepted if upon reading the string we reach the state  $q_A$ . However, we can only reach the state  $q_A$  from state  $1_T$  if the stack is empty.

Since  $B$  is co-ET0L, there exists a CSPD automaton

$$\mathcal{A}_B = (Q_B, \Sigma_B, \Delta_B, \Gamma_B, \mathcal{R}_B, I_B, F_B, \perp, \delta_B)$$

accepting  $coWP(B, X_B)$ .

We shall first give an intuitive idea of how  $\mathcal{A}$  works. We then follow this with the formal definition of  $\mathcal{A}$ .

**Intuitive Idea:** The automaton  $\mathcal{A}$  works as follows.

The check-stack of  $\mathcal{A}$  will contain a string  $r \in \mathcal{R}_B$  followed by the letter  $P$  some non-zero number of times.

First the automaton makes a choice about whether it checks the base group or the top group.

If the automaton checks the top group then it processes the input string as if it was  $\mathcal{A}_T$  while ignoring letters from  $X_B$ . That is, the letters from  $X_T$  are processed the same way  $\mathcal{A}_T$  processes them, and after a letter from  $X_T$  is finished processing then a letter from  $X_B$  does not edit the stack nor change the state.

If the automaton checks the base group, then it processes the input string in three stages.

1. The loading stage,
2. the coordinate location stage, and
3. processing the remainder of the string stage.

**The Loading Stage:** In this stage the automaton non-deterministically writes a freely reduced string on the pushdown stack, and moves to a state representing a transversal element. The product of the string on the pushdown stack and the transversal element defines an element of the top group, say  $h$ . This will be the inverse of the coordinate that we will locate in the following stage.

**The Coordinate Location Stage:** In this stage, the automaton simulates  $P_T^{(W,1)}$  while ignoring the letters from  $X_B$  until it reads the shortest prefix  $\omega_0$  of the input string such that  $\varphi_{X_T}(\omega_0) = h^{-1}$ . That is, after the previous stage, we read the shortest string  $\omega_0$  so that the state is  $1_T$  and the stack is empty after processing  $\omega_0$ .

**Processing the remainder of the string stage:** In this stage, the automaton processes letters from  $X_B$  the same way  $\mathcal{A}_B$  does until a letter from  $X_T$  is read. (Letters from  $\Gamma_B$  will be treated like bottom of stack symbols.) Upon reading a letter from  $X_T$  the automaton enters a mode which simulates  $P_T^{(W,1)}$  while ignoring letters from  $X_B$  until  $P_T^{(W,1)}$  accepts. Then the stack must have returned to the way it was before the letters from  $X_T$  were read and the automaton continues processing letters from  $X_B$  in the same way  $\mathcal{A}_B$  would. This processes repeats until the string is read.

We note that if the automaton checks the top group, the pushdown stack will only contain letters from  $\chi_T$ . However, if the automaton checks the base group then then pushdown stack will contain letters from  $\Gamma_B \cup \chi_T$ .

**Formal Definition of  $\mathcal{A}$ :** We define a CSPD automaton

$$\mathcal{A} := (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$$

accepting  $coWP(W, X_G \cup X_H)$  as follows.

- The state set  $Q$  consists of the following states:
  - the states  $q_0$ , and  $q_i$ .
  - For every in  $q \in Q_T$ , there is a state  $q$  representing that element of  $Q_T$ , we shall refer to it by the same name. We shall refer to this subset of  $Q$  as  $Q_T$  as well.
  - For every  $q \in Q_T \setminus \{q_A, \Psi\}$ , there is a state  $q^c$  representing that element of  $Q_T$ . This collection of all states  $q^c$  is denoted by  $Q_T^c$ . (We note that as the symbols are different,  $Q_T^c \cap Q_T = \emptyset$ .)

- For every  $q \in Q_B$  and every  $t \in T'$ , there is a state  $(q, t)$ . We call this collection of states  $Q_1$ .
- Recall that for every  $t \in T'$ , and every  $y \in X_T$ , there is a state in  $K_1 \subseteq Q_T$  representing every letter of  $w_{ty}$  and is further indexed by  $(t, y)$ . That is, if  $w_{ty} = x_1x_2 \cdots x_r \in \chi_T^*$  then these states are  $q_{(t,y,1)}, q_{(t,y,2)}, \dots, q_{(t,y,r)}$ . For each one of these states  $q_{(t,y,i)}$  in  $K_1 \subseteq Q_T$ , there is a state  $(q, t, q_{(t,y,i)})$ , for every  $q \in Q_G, t \in T'$ , and  $y \in X_T$ .
- The input alphabet is  $\Sigma = \Sigma_B \cup \Sigma_T = X_B \cup X_T$ .
- The check-stack alphabet is  $\Delta = \Delta_B \cup \{P\}$ .
- The pushdown stack alphabet is  $\Gamma = \Gamma_B \cup \chi_T$ .
- The regular language is  $\mathcal{R} = \mathcal{R}_B\{P\}^*$ .
- The set of initial states is  $I = \{q_0\}$ .
- The set of final states is  $F = F_T \cup (F_B \times T')$ ,

and finally the transition relation  $\delta$  consists of the following transitions.

(T0)(a) For every  $q \in I_H$ , there is a transition

$$((q_0, \varepsilon, (\perp, \perp)), (q, \perp));$$

that is, there is an  $\varepsilon$ -transition from state  $q_0$  with the pushdown stack being empty to state  $q \in I_H$  without editing the stack.

(T0)(b) There is a transition

$$((q_0, \varepsilon, (\perp, \perp)), (q_t, \perp));$$

that is, there is a transition from  $q_0$  with the pushdown stack being empty to  $q_t$  without editing the stack.

(T1)(a) For every transition  $\alpha = ((q, x, y), (q', w)) \in \delta_T$  where  $q, q' \in Q_T$  and  $x \in X_T \cup \{\varepsilon\}$ , there is a transition

$$((q, x, (A, y)), (q', w))$$

for every  $A \in \Delta \cup \{\perp\}$ . That is, there is a transition from  $q$  upon reading  $x$  with the top of the pushdown stack being  $y$  with corresponding letter on the check-stack  $A$  that moves to the same state  $\alpha$  does from  $q$  upon reading  $x$ , and writes onto the pushdown stack the string that  $\alpha$  writes.

(T1)(b) For every  $q \in T' \subseteq Q_T$ , every  $x \in X_B$  and every pair  $(A, B)$  there is a transition

$$((q, x, (A, B)), (q, B));$$

that is, from a state  $q \in T' \subseteq Q_T$ , upon reading  $x \in X_B$  with top of the pushdown stack being  $B$  and corresponding letter on the check-stack being  $A$ , the automaton remains at  $q$  and does not edit the stack.

(T2)(a) For every  $z \in \chi_T$  there is a transition

$$((q_l, \varepsilon, (\perp, \perp)), (q_l, z\perp));$$

that is, there is an  $\varepsilon$ -transition from  $q_l$  with the pushdown stack being empty to  $q_l$  while adding  $z$  onto the stack, for every  $z \in \chi_T$ .

(T2)(b) For every  $z, z' \in \chi_T$  such that  $z' \neq z^{-1}$  there are transitions

1.

$$((q_l, \varepsilon, (A, z)), (q_l, z'z));$$

that is, there is an  $\varepsilon$ -transition from  $q_l$  to  $q_l$  where  $z$  is replaced with  $z'z$  at the top of the pushdown stack for every  $z' \neq z^{-1}$  and for any  $A \in \Delta$ , and

2.

$$((q_l, \varepsilon, (A, z)), (q_l, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $q_l$  to  $q_l$  that deletes  $z$  from the top of the pushdown stack for any  $z \in \chi_T$  and any  $A \in \Delta$ .

(T2)(c) For every  $t^c \in T'^c \subseteq Q_T^c$  and pair  $(A, B)$  there is a transition

$$((q_l, \varepsilon, (A, B)), (t^c, B));$$

that is, there is an  $\varepsilon$ -transition from state  $q_l$  to state  $t^c$  for every state  $t^c \in T'^c \subseteq Q_T^c$  that does not edit the stack.

(T3) (a) For every  $x \in X_B$  we have the following transitions.

(i) For any  $A \in \Delta$  and  $z \in \chi_T$  (i.e.  $z \neq \perp$ ) there is a transition

$$((1_T^c, x, (A, z)), (1_T^c, z));$$

that is, there is a transition from  $1_T^c$  to itself upon reading  $x$  when the top of the pushdown stack is  $z \in \chi_T$  (i.e. the pushdown stack is non-empty) that does not edit the stack.

- (ii) For any  $t^c \in T'^c \subseteq Q_T^c$  such that  $t^c \neq 1_T^c$ , any  $A \in \Delta \cup \{\perp\}$ , and any  $B \in \chi_T \cup \{\perp\}$  there is a transition

$$((t^c, x, (A, B)), (t^c, B));$$

that is, there is a transition from  $t^c$  to itself (whenever  $t^c \neq 1_T^c$  upon reading  $x$  that does not edit the stack.

- (b) For every  $y \in X_T$  we have the following transitions.

- (i) For any  $A \in \Delta$  and  $z \in \chi_T$  (i.e.  $z \neq \perp$ ) there is a transition

$$((1_T^c, y, (A, z)), ((1_T^c, y)^c, z));$$

that is, there is a transition from  $1_T^c$  upon reading  $y \in X_T$  when the top of the pushdown stack is  $z \in \chi_T$  (i.e. the pushdown stack is non-empty) that moves to state  $(1_T^c, y)^c$  without editing the stack.

- (ii) For any  $t^c \in T'^c \subseteq Q_T^c$  such that  $t^c \neq 1_T^c$ , any  $A \in \Delta \cup \{\perp\}$ , and any  $B \in \chi_T \cup \{\perp\}$  there is a transition

$$((t^c, y, (A, B)), ((t^c, y)^c, B));$$

that is, there is a transition from  $t^c$  to  $(t^c, y)^c$  upon reading  $y$  with  $B \in \chi_T \cup \{\perp\}$  at the top of the pushdown stack with  $A$  being the corresponding letter on the check-stack that does not edit the stack.

- (c) Recall that for every  $y \in X_T$  and  $t \in T'$ , we have fixed a string  $w_{ty} = x_1 x_2 \cdots x_r \in \chi_T^*$  such that  $ty =_T w_{ty} t'$  for some  $t' \in T'$ .

For every  $t^c \in T'^c$  and  $y \in X_T$  we have the following transitions.

- (i) For any  $z \in \chi_T \cup \{\perp\}$  and any  $A \in \Delta \cup \{\perp\}$  there is a transition

$$(((t^c, y)^c, \varepsilon, (A, z)), (q_{(t^c, y, 1)}^c, P_{F_n}^{(W,1)}(i_0, x_1, z)))$$

where  $P_{F_n}^{(W,1)}(i_0, x_1, z)$  denotes the unique string such that

$$((i_0, x_1, z), (i_0, P_{F_n}^{(W,1)}(i_0, x_1, z))) \in \delta_{F_n}^{(W,1)};$$

that is, from state  $(t^c, y)^c$  and the top of the pushdown stack being  $z$  and the corresponding check-stack symbol is  $A$  there is an  $\varepsilon$ -transition to state  $q_{(t^c, y, 1)}^c$  that places the same string on the stack as the automaton  $P_{F_n}^{(W,1)}$  for  $F_n$  did when reading the letter  $x_1$  with  $z$  at the top of its stack.

(Note that this is analogous to transitions of  $P_T^{(W,1)}$ . We also note that in the case when  $t^c = 1_T^c$ , even though the transitions account for when  $z = \perp$  there are no transitions to  $(1_T^c, y)$  if the top of the pushdown stack is empty.)

- (ii) For every  $1 \leq i \leq r - 1$ , every  $A \in \Delta \cup \{\perp\}$ , every  $z \in \chi_T \cup \{\perp\}$  there is a transition

$$((q_{(t^c, y, i)}^c, \varepsilon, (A, z)), (q_{(t^c, y, i+1)}^c, P_{F_n}^{(W,1)}(i_0, x_{i+1}, z)))$$

where  $P_{F_n}^{(W,1)}(i_0, x_{i+1}, z)$  denotes the unique string such that

$$((i_0, x_{i+1}, z), (i_0, P_{F_n}^{W,1}(i_0, x_{i+1}, z))) \in \delta_{F_n}^{(W,1)};$$

that is, at state  $q_{(t^c, y, i)}^c$  when the top letter of the stack is  $z$  there is an  $\varepsilon$ -transition to the state  $q_{(t^c, y, i+1)}^c$  that places the same string on the stack as the automaton  $P_{F_n}^{(W,1)}$  for  $F_n$  did when reading the letter  $x_{i+1}$  with  $z$  at the top of its stack.

(Note that this is analogous to transitions of  $P_T^{(W,1)}$ .)

- (iii) For any  $z \in \chi_T \cup \{\perp\}$  and  $A \in \Delta \cup \{\perp\}$  there is a transition

$$((q_{(t^c, y, r)}^c, \varepsilon, (A, z)), (t'^c, z)).$$

(Note that this is analogous to transitions of  $P_T^{(W,1)}$ .)

- (d) For any  $q \in I_B$  there is a transition

$$((1_T^c, \varepsilon, (\perp, \perp)), ((q, 1_T), \perp));$$

that is, there is an  $\varepsilon$ -transition from  $1_T^c$  to  $(q, 1_T)$  for any  $q \in I_B$ .

- (T4) For every transition

$$((q, x, (A, B)), (q', w)) \in \delta_B$$

there is a transition

$$(((q, 1_T), x, (A, B)), ((q', 1_T), w));$$

that is, there is a transition from state  $(q, 1_T)$  upon reading  $x \in X_B \cup \{\varepsilon\}$  with  $B$  on the pushdown stack and corresponding check-stack letter  $A$  to state  $(q', 1_T)$  writing the string  $w$  whenever

$$((q, x, (A, B)), (q', w)) \in \delta_B.$$

- (T5) For any  $t \in T'$ , and  $y \in X_T$  recall we have fixed a string  $w_{ty} = x_1 x_2 \cdots x_r \in \chi_T^*$  such that  $ty =_T w_{ty} t'$ .

For any  $q \in X_B, t \in T', y \in X_T$ , and  $A \in \Delta \cup \{\perp\}$  we have the following transitions.



- (a) (i) For every  $B \in \Gamma_B$  there is a transition

$$(((q, t), y, (A, B)), ((q, t, (q, t, q_{(t,y,1)})), (P_{F_n}^{(W,1)}(i_0, x_1, \perp) - \perp)B))$$

where  $(P_{F_n}^{(W,1)}(i_0, x_1, \perp) - \perp)B$  denotes the concatenation of

$$P_{F_n}^{(W,1)}(i_0, x_1, \perp) - \perp$$

and  $B$ , with  $P_{F_n}^{(W,1)}(i_0, x_1, \perp)$  denoting the unique string such that

$$((i_0, x_1, \perp), (i_0, P_{F_n}^{(W,1)}(i_0, x_1, \perp))) \in \delta_{F_n}^{(W,1)};$$

that is, from state  $(q, t)$  upon reading  $y \in X_T$  with  $B \in \Gamma_B$  at the top of the pushdown stack and  $A$  being the corresponding check-stack letter there is a transition to  $(q, t, q_{(t,y,1)})$  that adds  $P_{F_n}^{(W,1)}(i_0, x_1, \perp) - \perp$  on top of the pushdown stack.

- (ii) For every  $B \in \chi_T \cup \{\perp\}$  there is a transition

$$(((q, t), y, (A, B)), ((q, t, q_{(t,y,1)}), P_{F_n}^{(W,1)}(i_0, x_1, B)));$$

that is, from state  $(q, t)$  upon reading  $y \in X_T$  with  $B \in \chi_T \cup \{\perp\}$  at the top of the pushdown stack and  $A$  being the corresponding check-stack letter there is a transition to  $(q, t, q_{(t,y,1)})$  writing the same string on pushdown stack the automaton  $P_{F_n}^{(W,1)}$  for  $F_n$  did when reading the letter  $x_1$  with  $B$  at the top of its stack.

- (b) For every  $1 \leq i \leq r - 1$  we have the following transitions.

- (i) For every  $B \in \Gamma_B$  there is a transition

$$(((q, t, q_{(q,t,i)}), \varepsilon, (A, B)), ((q, t, q_{(t,y,i+1)}), (P_{F_n}^{(W,1)}(i_0, x_{i+1}, \perp) - \perp)B))$$

where  $(P_{F_n}^{(W,1)}(i_0, x_{i+1}, \perp) - \perp)B$  denotes the concatenation of

$$P_{F_n}^{(W,1)}(i_0, x_1, \perp) - \perp$$

and  $B$ , with  $P_{F_n}^{(W,1)}(i_0, x_{i+1}, \perp)$  denoting the unique string such that

$$((i_0, x_{i+1}, \perp), (i_0, P_{F_n}^{(W,1)}(i_0, x_{i+1}, \perp))) \in \delta_{F_n}^{(W,1)};$$

that is, from state  $(q, t, q_{(t,y,i)})$  there is an  $\varepsilon$ -transition with  $B \in \Gamma_B$  at the top of the pushdown stack and  $A$  being the corresponding check-stack letter to state  $(q, t, q_{(t,y,i+1)})$  that adds  $P_{F_n}^{(W,1)}(i_0, x_1, \perp) - \perp$  to the top of the pushdown stack.

(ii) For every  $B \in \chi_T \cup \{\perp\}$  there is a transition

$$(((q, t, q_{(t,y,i)}), \varepsilon, (A, B)), ((q, t, q_{(t,y,i+1)}), P_{F_n}^{(W,1)}(i_0, x_{i+1}, B)));$$

that is, from state  $(q, t, q_{(t,y,i)})$  there is an  $\varepsilon$ -transition with  $B \in \chi_T \cup \{\perp\}$  at the top of the pushdown stack and  $A$  being the corresponding check-stack letter to state  $(q, t, q_{(t,y,i+1)})$  writing the same string on pushdown stack the automaton  $P_{F_n}^{(W,1)}$  for  $F_n$  did when reading the letter  $x_{i+1}$  with  $B$  at the top of its stack.

(c) For any pair  $(A, B)$  here is a transition

$$(((q, t, q_{(t,y,r)}), \varepsilon, (A, B)), ((q, t'), B));$$

that is, there is a  $\varepsilon$ -transition from  $(q, t, q_{(t,y,r)})$  to  $(q, t')$  without editing the stack.

(T6) For every  $x \in X_B$ , and  $q \in Q_B$  we have the following transitions.

(a) For any  $t \in \setminus T' \setminus \{1_T\}$ , and any pair  $(A, B)$  there is a transition

$$(((q, t), x, (A, B)), ((q, t), B));$$

that is, there is a transition from the state  $(q, t)$  upon reading  $x \in X_B$  with  $B$  at the top of pushdown stack and  $A$  being the corresponding letter on the check-stack that does not change the state or the stack.

(b) For any  $z \in \chi_T$ , and  $A \in \Delta$  there is a transition

$$(((q, 1_T), x, (A, z)), ((q, 1_T), z));$$

that is, there is a transition from state  $(q, 1_T)$  upon reading  $x \in X_B$  with  $z \in \chi_T$  at the top of the pushdown stack and  $A$  being the corresponding check-stack letter that does not change the state or the stack.

## 4.5.2 Proof of Theorem 4.5.1

In this section, we will provide a sequence of technical lemmata. We then present the proof of Theorem 4.5.1. Throughout this subsection, we shall let  $B, T, X_B, X_T, \mathcal{A}_B, \mathcal{A}_T$ , and  $\mathcal{A}$  as in the subsection above. First, we shall define some notation that will be useful in this section. We note that this notation is local to this section only.

**Definition 4.5.5.** Let  $C$  be a configuration of the automaton  $\mathcal{A}$ . Further, let  $u = x_1x_2 \cdots x_l \in X_B^*$ . Suppose

$$C \sim_{x_1} C \sim_{x_2} C \sim_{x_3} \cdots \sim_{x_n} C.$$

We denote the above chain by  $C \sim_u C$ . ♣

Due to the technical nature of the lemmata in this section we, we shall give an intuitive description about how the structure of the automaton relates to a wreath product. Further, we shall give a non-formal intuitive description of the statements of the first two lemmata before stating them. This will help in understanding the statements themselves.

We shall first relate the intuitive description of the automaton  $\mathcal{A}$  to the structure of a wreath product.

Recall that the wreath product  $W = B \wr T$  is a semi-direct product (as in Definition 2.1.15) and thus an element  $w'$  of  $W$  is a pair. The structure of the automaton reflects that. The two modes corresponds to checking non-triviality in either component.

In the first mode, we check whether the entry in the second component of  $w'$  is non-trivial. The second mode, checks whether the first component of  $w'$  is non-trivial. However, the first component is a tuple therefore it is non-trivial if there is a coordinate for which the entry in that coordinate is non-trivial. We give an outline as to how this is achieved below. First the automaton loads a freely reduced string in the generators of finitely indexed free subgroup  $F_n$  of  $T$  onto the pushdown stack. Then it passes to a state representing a transversal element. This defines an element  $y$  of  $T$  as every element of  $T$  can be expressed as a product of an element in  $F_n$  with a transversal element. The element  $y$  is the inverse of the coordinate that we will check. Then the automaton moves into coordinate location stage. In this stage the automaton simulates  $P_T^{(W,1)}$  when reading letters from  $X_T$  (while not editing the stack if the input letter is from  $X_B$ ). We keep doing this until the first time we see  $\perp$  at the top of the pushdown stack and the state is  $1_T^c$ . Then the automaton simulates  $\mathcal{A}_B$  reading a string in the generators  $X_B$  until a letter of  $X_T$  is read. With every letter from  $X_T$  read the coordinate which we are checking changes, thus the letters from  $X_B$  that are read do not impact the coordinate we are interested in until we have read a string in  $X_T$  that is equal to  $1_T$  as this represents returning to the correct coordinate. This process of possibly changing coordinate (and returning to the one we are interesting) repeats until the input string is read. We note that we only processed letters from  $X_H$  using  $\mathcal{A}_H$  when we were at the right coordinate.

We are now ready to give an intuitive description of our first lemma.

**Intuitive Description Lemma 4.5.6:** The Lemma 4.5.6 describes how the first mode of processing of  $\mathcal{A}$  processes a string  $\omega = u_1y_1u_2y_2 \cdots u_ny_n$  where  $u_i \in$

$X_B^*$  and  $y_i \in X_T$  for all  $1 \leq i \leq n$ . Recall the first mode of processing checks whether the second component of the element defined by  $\omega$  is non-trivial. Further recall the top group  $T$  is virtually free. Therefore the first mode of processing is simply the processing of  $\mathcal{A}_T$ , but with two modifications as follows.

1. Since  $\mathcal{A}$  is a CSPD automaton, the transitions and configurations look slightly different to those of  $\mathcal{A}_T$ , in particular we have to account for the check stack in the configurations.
2. All letters from  $X_B$  are ignored; i.e., they have no effect on the stack and state, and therefore they do not change the configurations.

Further, to be able to access the first mode of processing we must use the  $\varepsilon$ -transition from  $q_0$  to  $1_T$ .

**Lemma 4.5.6.** *Let  $\omega = u_1 y_1 u_2 y_2 \cdots u_n y_n \in (X_B \cup X_T)^*$  be such that  $u_i \in X_B^*$  and  $y_i \in X_T$  for all  $1 \leq i \leq n$ . Set  $t_0 = 1_T$  and  $t_{i-1} y_i =_T w_{t_{i-1} y_i} t_i$  for all  $1 \leq i \leq n$ . Denote  $\rho_{X_T}$  by  $\rho$ .*

*Then  $\mathcal{A}$  reads  $\omega$  through the following chain, where we write the pushdown stack as a string instead of a tuple,*

$$\begin{aligned}
(q_0, (S, \perp)) &\sim_\varepsilon (1_T, (S, \perp)) \\
&\sim_{u_1} (1_T, (S, \perp)) \sim_{y_1} ((1_T, y_1), (S, \perp)) \sim_\varepsilon^* (t_1, (S, \rho(w_{1_T y_1}) \perp)) \\
&\sim_{u_2} (t_1, (S, \rho(w_{1_T y_1}) \perp)) \sim_{y_2} ((t_1, y_2), (S, \rho(w_{1_T y_1}) \perp)) \\
&\sim_\varepsilon^* (t_2, (S, \rho(w_{1_T y_1} w_{t_1 y_2}) \perp)) \\
&\vdots \\
&\sim_{u_n} (t_{n-1}, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp)) \\
&\sim_{y_n} ((t_{n-1}, y_n), (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp)) \\
&\sim_\varepsilon^* (t_n, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}} w_{t_{n-1} y_n}) \perp))
\end{aligned}$$

for a long-enough choice of check-stack  $S$ .

*Proof.* By our work in [2.3.2](#) we know that  $A_T$  reads  $y_1 y_2 \cdots y_n$  through the chain

$$\begin{aligned}
(1_T, \perp) &\sim_{y_1} ((1_T, y_1), \perp) \sim_\varepsilon^* (t_1, \rho(w_{1_T y_1}) \perp) \\
&\sim_{y_2} ((t_1, y_2), \rho(w_{1_T y_1}) \perp) \sim_\varepsilon^* (t_2, \rho(w_{1_T y_1} w_{t_1 y_2}) \perp) \\
&\vdots \\
&\sim_{y_n} ((t_{n-1}, y_n), \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp) \\
&\sim_\varepsilon^* (t_n, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}} w_{t_{n-1} y_n}) \perp). \tag{18.a}
\end{aligned}$$

The transitions used in the above chain, are analogous to those in (T1)(a). (The transitions in (T1)(a) are those of  $\mathcal{A}_T$  but modified to account for the check-stack.)

Let  $S$  be a long-enough choice of check-stack. Note the transitions in (T1)(b) imply that letters from  $X_B$  do not effect any configuration whose state is an element  $t \in T' \subseteq Q_T$  representing the transversal element  $t$ . Further, there are no reading-transitions reading letters from  $X_B$  from any state in  $Q_T \setminus T'$ . Therefore the only way  $\mathcal{A}$  can read an input letter from  $X_B$  at a state in  $Q_T$  then is by being at a state in  $T'$ . Therefore letters from  $X_B$  must be read from states in  $T'$ , and letters from  $X_B$  have no effect on any configuration whose state belongs to  $T'$ .

We also note that there is the transition in (T0)(a) moving the automaton from  $q_0$  to  $1_T$ , as  $1_T$  is the unique initial state of  $\mathcal{A}_T$ . The transition changes state without editing the stack. Therefore,  $\mathcal{A}$  reads  $\omega$  through a chain starting as

$$(q_0, (S, \perp)) \sim_\varepsilon (1_T, (S, \perp)). \quad (18.b)$$

Using the transitions in (T0)(a) that are analogous to those yielding (18.a) and since letters from  $X_B$  have no effect on any configuration whose state belongs to  $T'$ , (18.b) continues as

$$\begin{aligned} (q_0, (S, \perp)) &\sim_\varepsilon (1_T, (S, \perp)) \\ &\sim_{u_1} (1_T, (S, \perp)) \sim_{y_1} ((1_T, y_1), (S, \perp)) \sim_\varepsilon^* (t_1, (S, \rho(w_{1_T y_1}) \perp)) \\ &\sim_{u_2} (t_1, (S, \rho(w_{1_T y_1}) \perp)) \sim_{y_2} ((t_1, y_2), (S, \rho(w_{1_T y_1}) \perp)) \\ &\sim_\varepsilon^* (t_2, (S, \rho(w_{1_T y_1} w_{t_1 y_2}) \perp)) \\ &\vdots \\ &\sim_{u_n} (t_{n-1}, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp)) \\ &\sim_{y_n} ((t_{n-1}, y_n), (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp)) \\ &\sim_\varepsilon^* (t_n, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}} w_{t_{n-1} y_n}) \perp)). \end{aligned}$$

■

**Intuitive Description of Lemma 4.5.7** The Lemma 4.5.7 describes how the second mode of processing of  $\mathcal{A}$  processes a string  $\omega = \omega_0 u_1 v_1 u_2 v_2 \cdots u_n v_n$ . The lemma divides the processing of  $\omega$  into three categories. These are the initial chain, chains of type  $U$ , chains of type  $V$ . These chains process different parts of the string  $\omega$ .

The initial chain processes a prefix  $\omega_0$  of  $\omega$  such that  $\varphi_{X_T}(\omega) =_T h^{-1}$  for some  $h \in T$ . This is done through the transitions  $P_T^{(W,1)}$  while ignoring letters from  $X_B$ .

Chains of type  $U$  are responsible for processing the  $u_i$ , chains of type  $V$  are responsible for processing the  $v_i$ .

We note that the following lemma is long and technical, this because there is some setup involved.

**Lemma 4.5.7.** Fix  $h \in T$  and  $t \in T'$ . Let  $\omega = \sigma_1 \sigma_2 \cdots \sigma_r \in (X_B \cup X_T)^*$ . Assume there exists a prefix  $\omega_0 = u'_1 y_1 u'_2 y_2 \cdots u'_n y_n$  (where  $u'_i \in X_B^*$  and  $y_i \in X_T$ ) of  $\omega$  such that  $y_1 y_2 \cdots y_n =_T h^{-1}$ . We assume the last letter of  $\omega_0$  is in  $X_T$ . Set  $t_0 := t$  and  $t_{i-1} y_i =_T w_{t_{i-1} y_i} t_i$  for all  $1 \leq i \leq n-1$ .

Consider the following decomposition of  $\omega$

$$\omega = \omega_0 u_1 v_1 u_2 v_2 \cdots u_n v_n,$$

where

$$u_i = x_{i,1} x_{i,2} \cdots x_{i,l(u_i)} \in X_B^*$$

and  $v_i \in (X_B \cup X_T)^*$  such that the first and last letter of  $v_i$  is in  $X_T$ , for all  $1 \leq i \leq n$ . Further suppose for each  $1 \leq j \leq n-1$ , there does not exist a prefix  $v'_j$  such that  $\varphi_{X_T}(v'_j) \neq_T 1_T$ , but  $\varphi_{X_T}(v'_j) =_T 1_T$ .

Let

$$v_i = y_{i,1} u'_{i,1} y_{i,2} u'_{i,2} \cdots y_{i,l(v_i)-1} u'_{i,l(v_i)-1} y_{i,l(v_i)}$$

for all  $1 \leq i \leq n$ . Set  $t_{i,0} = 1_T$  for every  $1 \leq i \leq n$ . Further set

$$t_{i,j-1} y_{i,j} =_T w_{t_{i,j-1} y_{i,j}} t_{i,j}$$

for every  $1 \leq j \leq l(v_i)$  and every  $1 \leq i \leq n$ . Further, let  $z_{i,j} \in \chi_T$  be the first letter of  $w_{t_{i,j-1} y_{i,j}}$

Suppose there exists a choice of check-stack  $S = (a_m, a_{m-1}, \dots, a_1, \perp)$  (where  $a_m a_{m-1} \cdots a_1 \in \mathcal{R}_B$ ) such that  $\mathcal{A}_B$  reads  $u_1 u_2 \cdots u_n$  through the chain

$$\begin{aligned} (q, (S, (\perp))) &\sim_\varepsilon^* (q_{1,1}^\varepsilon, (S, S_{1,1}^\varepsilon)) \sim_{x_{1,1}} (q_{1,1}, (S, S_{1,1})) \\ &\sim_\varepsilon^* (q_{1,2}^\varepsilon, (S, S_{1,2}^\varepsilon)) \sim_{x_{1,2}} (q_{1,2}, (S, S_{1,2})) \\ &\vdots \\ &\sim_\varepsilon^* (q_{1,l(u_1)}^\varepsilon, (S, S_{1,l(u_1)}^\varepsilon)) \sim_{x_{1,l(u_1)}} (q_{1,l(u_1)}, (S, S_{1,l(u_1)})) \\ &\sim_\varepsilon^* (q_{2,1}^\varepsilon, (S, S_{2,1}^\varepsilon)) \sim_{x_{2,1}} (q_{2,1}, (S', S_{2,1})) \\ &\sim_\varepsilon^* (q_{2,2}^\varepsilon, (S, S_{2,2}^\varepsilon)) \sim_{x_{2,2}} (q_{2,2}, (S, S_{2,2})) \\ &\vdots \\ &\sim_\varepsilon^* (q_{2,l(u_2)}^\varepsilon, (S, S_{2,l(u_2)}^\varepsilon)) \sim_{x_{2,l(u_2)}} (q_{2,l(u_2)}, (S, S_{2,l(u_2)})) \\ &\vdots \\ &\sim_\varepsilon^* (q_{n,1}^\varepsilon, (S, S_{n,1}^\varepsilon)) \sim_{x_{n,1}} (q_{n,1}, (S, S_{n,1})) \\ &\sim_\varepsilon^* (q_{n,2}^\varepsilon, (S, S_{n,2}^\varepsilon)) \sim_{x_{n,2}} (q_{n,2}, (S, S_{n,2})) \\ &\vdots \\ &\sim_\varepsilon^* (q_{n,l(u_n)}^\varepsilon, (S, S_{n,l(u_n)}^\varepsilon)) \sim_{x_{n,l(u_n)}} (q_{n,l(u_n)}, (S, S_{n,l(u_n)})) \end{aligned}$$

where  $q \in I_B$ . (We note that we enumerated the states and pushdown stacks in a way to help with keeping track of what has already been read and what type of transitions have been used. That is, after  $\sim^\varepsilon$  the symbol denoting pushdown stack is decorated with an  $\varepsilon$  as a superscript, and the subscripts are enumerated in the same way the input alphabet is, as illustrated above.) Then there exists a long enough extension of  $S$

$$S' = (P, P, \dots, P, a_m, a_{m-1}, \dots, a_1, \perp)$$

such that  $\mathcal{A}$  reads  $\omega$  through a chain which we decompose into  $2n + 1$  parts that can be categorised into three categories.

The categories are as follows

1. initial chain,
2. chain of type  $U$ , and
3. chain of type  $V$ .

There is one chain in the initial chain category, which we shall call the initial chain. There are  $n$  chains of type  $U$ , and  $n$  chains of type  $V$ .

The first chain of type  $U$  starts after the initial chain ends. The  $i^{\text{th}}$  chain of type  $V$  starts after the  $i^{\text{th}}$  chain of type  $U$  ends. The  $i + 1^{\text{th}}$  chain of type  $U$  starts after the  $i^{\text{th}}$  chain of type  $V$  ends.

We list the initial chain below, as well as the  $i^{\text{th}}$  chain of type  $U$ , and the  $i^{\text{th}}$  chain of type  $V$ . Below we denote  $\rho_{\chi_T}$  by  $\rho$ .

We express the pushdown stack as a string instead of a sequence for the initial chain and chains of type  $V$ .

The initial chain is the part of the chain responsible for processing  $\omega_0$ . The initial chain can start as

$$(q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u\perp)),$$

where  $ut =_T h$ . Further if it does then it continues as follows

$$\begin{aligned} & (q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u\perp)) \\ & \sim_{u'_1} (t^c, (S', u\perp)) \sim_{y_1} ((t^c, y_1)^c, (S', u\perp)) \sim_\varepsilon^* (t_1^c, (S', \rho(uw_{t_0y_1})\perp)) \\ & \sim_{u'_2} (t_1^c, (S', \rho(uw_{t_0y_1})\perp)) \sim_{y_2} ((t_1^c, y_2)^c, (S', \rho(uw_{t_0y_1})\perp)) \\ & \sim_\varepsilon^* (t_2^c, (S', \rho(uw_{t_0y_1}w_{t_1y_2})\perp)) \\ & \vdots \\ & \sim_{u'_n} (t_{n-1}^c, (S', \rho(uw_{t_0y_1}w_{t_1y_2} \cdots w_{t_{n-2}y_{n-1}})\perp)) \\ & \sim_{y_n} ((t_{n-1}^c, y_n)^c, (S', \rho(uw_{t_0y_1}w_{t_1y_2} \cdots w_{t_{n-2}y_{n-1}})\perp)) \\ & \sim_\varepsilon^* (t_n, (S', \rho(uw_{t_0y_1}w_{t_1y_2} \cdots w_{t_{n-2}y_{n-1}}w_{t_{n-1}y_n})\perp)) = (1_T^c, (S', (\perp))) \\ & \sim_\varepsilon ((q, 1_T), (S', (\perp))). \end{aligned}$$

The  $i^{\text{th}}$  chain of type  $U$  is the part of the chain responsible for processing  $u_i = x_{i,1}x_{i,2} \cdots x_{i,l(u_i)}$ , and it is as follows

$$\begin{aligned}
((q_{i,1}^\varepsilon, 1_T)(S', S_{i,1}^\varepsilon)) &\sim_{x_{i,1}} (q_{i,1}, (S', S_{i,1})) \\
&\sim_\varepsilon^* ((q_{i,2}^\varepsilon, 1_T), (S', S_{i,2}^\varepsilon)) \sim_{x_{i,2}} ((q_{i,2}, 1_T), (S', S_{i,2})) \\
&\vdots \\
&\sim_\varepsilon^* ((q_{i,l(u_i)}^\varepsilon, 1_T), (S', S_{i,l(u_i)}^\varepsilon)) \sim_{x_{i,l(u_i)}} ((q_{i,l(u_i)}, 1_T), (S', S_{i,l(u_i)})) \\
&\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)).
\end{aligned}$$

The  $i^{\text{th}}$  chain of type  $V$  is the part of the chain responsible for processing

$$v_i = y_{i,1}u'_{i,1}y_{i,2}u'_{i,2} \cdots y_{i,l(v_i)-1}u'_{i,l(v_i)-1}y_{i,l(v_i)},$$

and it is as follows

$$\begin{aligned}
((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)) &\sim_{y_{i,1}} ((q_{i+1,1}^\varepsilon, 1_T, q(q_{i+1,1}^\varepsilon, 1_T, 1)), (S', z_{i,1}S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}})S_{i+1,1}^\varepsilon)) \\
&\sim_{u'_{i,1}} ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}})S_{i+1,1}^\varepsilon)) \\
&\sim_{y_{i,2}} ((q_{i+1,1}^\varepsilon, t_{i,1}, q(q_{i+1,1}^\varepsilon, t_{i,1}, 1)), (S', \rho(w_{1_T y_{i,1}} z_{i,2})S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,2}), (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}})S_{i+1,1}^\varepsilon)) \\
&\sim_{u'_{i,2}} ((q_{i+1,1}^\varepsilon, t_{i,2}), (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}})S_{i+1,1}^\varepsilon)) \\
&\vdots \\
&\sim_{y_{i,l(v_i)-1}} ((q_{i+1,1}^\varepsilon, t_{i,l(v_i)-2}, q(q_{i+1,1}^\varepsilon, t_{i,l(v_i)-2}, 1)), \\
&\quad (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,l(v_i)-3} y_{i,l(v_i)-2}} z_{i,l(v_i)}))S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* (q_{i+1,1}^\varepsilon, t_{i,l(v_i)-1}), \\
&\quad (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,l(v_i)-3} y_{i,l(v_i)-2}} w_{t_{i,l(v_i)-2} y_{i,l(v_i)-1}})S_{i+1,1}^\varepsilon) \\
&\sim_{u'_{i,l(v_i)-1}} (q_{i+1,1}^\varepsilon, t_{i,l(v_i)-1}), \\
&\quad (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,l(v_i)-3} y_{i,l(v_i)-2}} w_{t_{i,l(v_i)-2} y_{i,l(v_i)-1}})S_{i+1,1}^\varepsilon) \\
&\sim_{y_{i,l(v_i)}} ((q_{i+1,1}^\varepsilon, t_{i,l(v_i)-1}, q(q_{i+1,1}^\varepsilon, t_{i,l(v_i)-1}, 1)), \\
&\quad (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,l(v_i)-2} y_{i,l(v_i)-1}} z_{i,l(v_i)}))S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,l(v_i)}), \\
&\quad (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,l(v_i)-1} y_{i,l(v_i)}})S_{i+1,1}^\varepsilon)).
\end{aligned}$$

*Proof.* We shall first construct the initial chain. Let  $S'$  be as in the statement of the lemma.



First we shall prove that the chain can start in the way lemma states it does.

Observe that (T0)(b) is an  $\varepsilon$ -transition from  $q_0$  to  $q_l$  that does not edit the stack. Therefore the chain starts with

$$(q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)).$$

In the transition in (T2)(a), letters are written onto an empty stack. The transitions in (T2)(b) write  $z'z$  onto the stack where  $z' \neq z^{-1}$  if  $z$  was at the top of the pushdown stack, and can also delete the letter  $z$  from the top of the pushdown stack. Thus after using the transitions in (T2)(a) and (T2)(b), the pushdown stack will not contain two consecutive letters that are the inverse of one another. Thus these transitions write a freely reduced string onto the pushdown stack. Further, as the transitions account for all possible combinations of the top of the pushdown stack and letters that be written onto the stack then using the transitions the automaton can write any freely reduced string onto the pushdown stack. The string  $u$  in the statement is such a string and thus be written by the automaton. Now, the transitions in (T2)(c) move the state of the automaton from  $q_l$  to a state  $t^c$  without editing the stack. Further, there transitions from  $q_l$  to every state  $t^c$  for every  $t' \in T'$ . Thus the automaton can move from  $q_l$  to the state  $t^c$  in the statement of the lemma. Therefore the chain starts with

$$(q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u\perp)).$$

We shall now construct the initial chain. We shall construct the part of the chain reading  $u'_1v_1$ . The transitions in (T3)(a) show that upon reading a letter from  $X_B$  with the state being a transversal element with a superscript  $c$  as a decoration, then the state and the stack are not changed. Using (T3)(a) the chain continues

$$(q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u\perp)) \\ \sim_{u'_1} (t^c, (S', u\perp)).$$

Reading  $y_1$ , the automaton uses a transition in (T3)(b) that reads  $y_1$  with the first letter of  $u$  being at the top of the stack and moving to a state  $(t_1^c, y_1)^c$  without editing the stack. Thus the chain continues as

$$(q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u\perp)) \\ \sim_{u'_1} (t^c, (S', u\perp)) \sim_{y_1} ((t^c, y_1)^c, (S', u\perp)).$$

Now observe that the transitions in (T3)(c) are analogous to those of  $P_T^{(W,1)}$ , and the use of these transitions has the same effect on the pushdown stack as their analogues do on the stack of  $P_T^{(W,1)}$  while the state gets decorated with the superscript

c. Thus the chain continues as

$$(q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u\perp)) \\ \sim_{u'_1} (t^c, (S', u\perp)) \sim_{y_1} ((t^c, y_1)^c, (S', u\perp)) \sim_\varepsilon^* (t_1^c, (S', u\perp)).$$

We shall assume we have constructed the part of the chain reading  $u'_1 y_1 u'_2 y_2 \cdots u'_{i-1} y_{i-1}$ , and we shall now construct the part of the chain reading  $u'_i y_i$ . After reading  $u'_1 y_1 u'_2 y_2 \cdots u'_{i-1} y_{i-1}$  (and then using a sequence of  $\varepsilon$ -transitions), the configuration is

$$(t_{i-1}^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp))$$

by our assumption. The transitions in (T3)(a) show that upon reading a letter from  $X_B$  with the state being of the form  $t^c$  the automaton does not change state or edit the stack. Therefore the chain continues as

$$(t_{i-1}^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)) \sim_{u'_i} (t_{i-1}^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)).$$

Further, by (T3)(b) we see that when a letter from  $X_T$  is read from state  $t_{i-1}^c$ , the automaton moves to state  $(t_{i-1}^c, y_i)^c$  without editing the stack. The chain continues as

$$(t_{i-1}^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)) \sim_{u'_i} (t_{i-1}^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)) \\ \sim_{y_i} ((t_{i-1}^c, y_i)^c, \\ (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)).$$

Now, observe that the transitions in (T3)(c) are analogous to those of  $P_T^{(W,1)}$ , and the use of these transitions has the same effect on the pushdown stack as their analogues do on the stack of  $P_T^{(W,1)}$  while the state gets decorated with the superscript  $c$ . Thus the chain continues as

$$(t_{i-1}^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)) \sim_{u'_i} (t_{i-1}^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)) \\ \sim_{y_i} ((t_{i-1}^c, y_i)^c, \\ (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-2} y_{i-1}}) \perp)) \\ \sim_\varepsilon^* (t_i^c, (S', \rho(uw_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{i-1} y_i}) \perp)).$$

This shows the initial chain is as in the statement of the lemma up to reading  $y_n$ . Since,  $y_1 y_2 \cdots y_n =_T h^{-1}$  and thus  $h y_1 y_2 \cdots y_n =_T 1_T$  then it must be that  $t_n = 1_T$  and the stack is empty (as seen in the proof of  $\mathcal{L}(P_T^{(W,1)}) = WP(T, X_T)$  in [2.3.2](#)). Now we can use the transition in (T3)(d) from  $1_T^c$  to the state  $(q, 1_T)$  where  $q \in I_B$ . Thus we achieve the initial chain in the statement of the lemma.

We shall now construct the first chain of type  $U$  and the first chain of type  $V$ . The first chain of type  $U$  is the one responsible for reading  $u_1 = x_{1,1}x_{1,2} \cdots x_{1,l(u_1)}$ . The first chain of type of  $V$  is the one responsible for

$$v_1 = y_{1,1}u'_{1,1}y_{1,2}u'_{1,2} \cdots y_{1,l(v_1)-1}u'_{1,l(v_1)-1}y_{1,l(v_1)}.$$

We shall start with the first chain of type  $U$ . The initial chain ends with the configuration  $((q, 1_B), (S', \perp))$ . The automaton  $\mathcal{A}_B$  reads  $u_1 = x_{1,1}x_{1,2} \cdots x_{1,l(u_1)}$  through the chain

$$\begin{aligned} (q, (S, (\perp))) &\sim_{\varepsilon}^* (q_{1,1}^{\varepsilon}, (S, S_{1,1}^{\varepsilon})) \sim_{x_{1,1}} (q_{1,1}, (S, S_{1,1})) \\ &\sim_{\varepsilon}^* (q_{1,2}^{\varepsilon}, (S, S_{1,2}^{\varepsilon})) \sim_{x_{1,2}} (q_{1,2}, (S, S_{1,2})) \\ &\vdots \\ &\sim_{\varepsilon}^* (q_{1,l(u_1)}^{\varepsilon}, (S, S_{1,l(u_1)}^{\varepsilon})) \sim_{x_{1,l(u_1)}} (q_{1,l(u_1)}, (S, S_{1,l(u_1)})) \\ &\sim_{\varepsilon}^* (q_{2,1}^{\varepsilon}, (S, S_{2,1}^{\varepsilon})). \end{aligned}$$

The transitions in (T4), are analogous to those of  $\delta_B$ , in that if a transition

$$((q, x, (A, B)), (q', w)) \in \delta_B$$

then there is a transition

$$(((q, 1_T), x, (A, B)), ((q', 1_T), w))$$

in (T4). Thus the chain above induces the following chain reading  $u_1$  in  $\mathcal{A}$

$$\begin{aligned} ((q, 1_T), (S', (\perp))) &\sim_{\varepsilon}^* ((q_{1,1}^{\varepsilon}, 1_T), (S', S_{1,1}^{\varepsilon})) \sim_{x_{1,1}} ((q_{1,1}, 1_T), (S', S_{1,1})) \\ &\sim_{\varepsilon}^* ((q_{1,2}^{\varepsilon}, 1_T), (S', S_{1,2}^{\varepsilon})) \sim_{x_{1,2}} ((q_{1,2}, 1_T), (S', S_{1,2})) \\ &\vdots \\ &\sim_{\varepsilon}^* ((q_{1,l(u_1)}^{\varepsilon}, 1_T), (S', S_{1,l(u_1)}^{\varepsilon})) \sim_{x_{1,l(u_1)}} ((q_{1,l(u_1)}, 1_T), (S', S_{1,l(u_1)})) \\ &\sim_{\varepsilon}^* ((q_{2,1}^{\varepsilon}, 1_T), (S', S_{2,1}^{\varepsilon})). \end{aligned}$$

We shall now construct the first chain of type  $V$ . The first chain of type  $U$  ends with the configuration  $((q_{2,1}^{\varepsilon}, 1_T), (S', S_{2,1}^{\varepsilon}))$ . Upon reading  $y_{1,1}$ , there is a transition in (T5)(a) that moves from state  $(q_{2,1}^{\varepsilon}, 1_T)$  to state  $(q_{2,1}^{\varepsilon}, 1_T, q_{(q_{2,1}^{\varepsilon}, 1_T, 1)})$  that pushes the first letter of  $w_{1_T, y_{1,1}}$ , say  $x_1$  onto the stack. Thus the chain starts

$$((q_{2,1}^{\varepsilon}, 1_T), (S', S_{2,1}^{\varepsilon})) \sim_{y_{1,1}} ((q_{2,1}^{\varepsilon}, 1_T, q_{(q_{2,1}^{\varepsilon}, 1_T, 1)}), (S', x_1 S_{2,1}^{\varepsilon})).$$

The transitions in (T5)(b) simulate reading  $w_{1_T y_{1,1}}$  in  $P_{F_n}^{(W,1)}$ , and thus as the end of applying transitions in (T5)(b), the chain continues as

$$\begin{aligned} ((q_{2,1}^\varepsilon, 1_T), (S', S_{2,1}^\varepsilon)) &\sim_{y_{1,1}} ((q_{2,1}^\varepsilon, 1_T, q_{(q_{2,1}^\varepsilon, 1_T, 1)}), (S', x_1 S_{2,1}^\varepsilon)) \\ &\sim_\varepsilon^* ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)). \end{aligned}$$

Reading an input letter from  $X_B$  with the state being  $(q_{2,1}^\varepsilon, t_{1,1})$  the automaton uses transitions in (T6), and the state and the stack do not change. Therefore the chain above continues as

$$\begin{aligned} ((q_{2,1}^\varepsilon, 1_T), (S', S_{2,1}^\varepsilon)) &\sim_{y_{1,1}} ((q_{2,1}^\varepsilon, 1_T, q_{(q_{2,1}^\varepsilon, 1_T, 1)}), (S', x_1 S_{2,1}^\varepsilon)) \\ &\sim_\varepsilon^* ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)) \\ &\sim_{u'_{1,1}} ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)). \end{aligned}$$

We shall now assume that we have constructed the first chain of type  $V$  up to reading  $y_{1,i} u'_{1,i}$ . We shall now construct the part reading  $y_{1,i+1} u'_{1,i+1}$ . The last configuration in the part of the chain reading  $y_{1,1} u'_{1,1} y_{1,2} u'_{1,2} \cdots y_{1,i} u'_{1,i}$  is

$$((q_{2,1}^\varepsilon, t_{1,i}), (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}}) S_{2,1}^\varepsilon)).$$

Upon reading  $y_{1,i+1}$ , there is a transition in (T5)(a) that moves from state  $(q_{2,1}^\varepsilon, t_{1,i})$  to state  $(q_{2,1}^\varepsilon, t_{1,i}, q_{(q_{2,1}^\varepsilon, t_{1,i}, 1)})$  that processes the first letter of  $w_{t_{1,i} y_{1,i+1}}$ , say  $x_{i+1}$  the same way  $P_T^{(W,1)}$  if the symbol at top of its stack was the first letter of  $\rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}})$ . Therefore the chain continues

$$\begin{aligned} ((q_{2,1}^\varepsilon, 1_T), (S', S_{2,1}^\varepsilon)) &\sim_{y_{1,1}} ((q_{2,1}^\varepsilon, 1_T, q_{(q_{2,1}^\varepsilon, 1_T, 1)}), (S', x_1 S_{2,1}^\varepsilon)) \\ &\sim_\varepsilon^* ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)) \\ &\sim_{u'_{1,1}} ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)) \\ &\vdots \\ &\sim_{y_{1,i+1}} ((q_{2,1}^\varepsilon, t_{1,i}, q_{(q_{2,1}^\varepsilon, t_{1,i}, 1)}), \\ &\quad (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}} x_{i+1}) S_{2,1}^\varepsilon)). \end{aligned}$$

The transitions in (T5)(b) simulate reading  $w_{t_{1,i} y_{1,i+1}}$  in  $P_{F_n}^{(W,1)}$ , and thus as the

end of applying transitions in (T5)(b), the chain continues as

$$\begin{aligned}
((q_{2,1}^\varepsilon, 1_T), (S', S_{2,1}^\varepsilon)) &\sim_{y_{1,1}} ((q_{2,1}^\varepsilon, 1_T, q_{(q_{2,1}^\varepsilon, 1_T, 1)}), (S', x_1 S_{2,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)) \\
&\sim_{u'_{1,1}} ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)) \\
&\vdots \\
&\sim_{y_{1,i+1}} ((q_{2,1}^\varepsilon, t_{1,i}, q_{(q_{2,1}^\varepsilon, t_{1,i}, 1)}), \\
&\quad (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}} x_{i+1}) S_{2,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{2,1}^\varepsilon, t_{1,i+1}), (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}} w_{t_{1,i} y_{1,i+1}}) S_{2,1}^\varepsilon)).
\end{aligned}$$

Reading an input letter from  $X_B$  with the state being  $(q_{2,1}^\varepsilon, t_{1,i+1})$  the automaton uses transitions in (T6), and the state and the stack do not change. Therefore the chain above continues as

$$\begin{aligned}
((q_{2,1}^\varepsilon, 1_T), (S', S_{2,1}^\varepsilon)) &\sim_{y_{1,1}} ((q_{2,1}^\varepsilon, 1_T, q_{(q_{2,1}^\varepsilon, 1_T, 1)}), (S', x_1 S_{2,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)) \\
&\sim_{u'_{1,1}} ((q_{2,1}^\varepsilon, t_{1,1}), (S', \rho(w_{1_T y_{1,1}}) S_{2,1}^\varepsilon)) \\
&\vdots \\
&\sim_{y_{1,i+1}} ((q_{2,1}^\varepsilon, t_{1,i}, q_{(q_{2,1}^\varepsilon, t_{1,i}, 1)}), \\
&\quad (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}} x_{i+1}) S_{2,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{2,1}^\varepsilon, t_{1,i+1}), \\
&\quad (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}} w_{t_{1,i} y_{1,i+1}}) S_{2,1}^\varepsilon)) \\
&\sim_{u'_{1,i+1}} ((q_{2,1}^\varepsilon, t_{1,i+1}), \\
&\quad (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,i-1} y_{1,i}} w_{t_{1,i} y_{1,i+1}}) S_{2,1}^\varepsilon)).
\end{aligned}$$

Thus we see that the first chain of type  $V$  ends with the configuration

$$((q_{2,1}^\varepsilon, t_{1,l(v_1)}), (S', \rho(w_{1_T y_{1,1}} w_{t_{1,1} y_{1,2}} \cdots w_{t_{1,l(v_1)-1} y_{1,l(v_1)}}) S_{2,1}^\varepsilon)).$$

However, since  $y_{1,1} y_{1,2} \cdots y_{1,l(v_1)} =_T 1_T$  then it must be that  $t_{1,l(v_1)} = 1_T$  and the stack is empty (as seen in the proof of  $\mathcal{L}(P_T^{(W,1)}) = WP(T, X_T)$  in [2.3.2](#)). Since  $t_{1,l(v_1)} = 1_T$  the automaton can now use transitions in (T4) to continue reading  $u_2$  through the second chain of type  $U$ .

We shall now assume that we have constructed the  $k^{\text{th}}$  chain of type  $U$  and  $V$  for all  $k \leq i - 1$ .

We will first construct the  $i^{\text{th}}$  chain of type  $U$ . The  $i - 1^{\text{th}}$  chain of type  $V$  ends with the configuration

$$((q_{i,1}^\varepsilon, 1_T), (S', S_{i,1}^\varepsilon)).$$

Recall the  $i^{\text{th}}$  chain of type  $U$  is the one responsible for processing  $u_i = x_{i,1}x_{i,2} \cdots x_{i,l(u_i)}$ . Examining the chain by which  $\mathcal{A}_B$  reads  $u_1u_2 \cdots u_n$  we see that the part of that chain responsible for reading  $x_{i,1}x_{i,2} \cdots x_{i,l(u_i)}$  is the following,

$$\begin{aligned} (q_{i,1}^\varepsilon, (S, S_{i,1}^\varepsilon)) &\sim_{x_{i,1}} (q_{i,1}, (S, S_{i,1})) \\ &\sim_\varepsilon^* (q_{i,2}^\varepsilon, (S, S_{i,2}^\varepsilon)) \sim_{x_{i,2}} (q_{i,2}, (S, S_{i,2})) \\ &\vdots \\ &\sim_\varepsilon^* (q_{i,l(u_i)}^\varepsilon, (S, S_{i,l(u_i)}^\varepsilon)) \sim_{x_{i,l(u_i)}} (q_{i,l(u_i)}, (S, S_{i,l(u_i)})). \end{aligned}$$

The transitions in (T4), are analogous to those of  $\delta_B$ , in that if a transition

$$((q, x, (A, B)), (q', w)) \in \delta_B$$

then there is a transition

$$(((q, 1_T), x, (A, B)), ((q', 1_T), w))$$

in (T4). Thus the chain above induces the following chain reading  $u_i$  in  $\mathcal{A}$

$$\begin{aligned} ((q_{i,1}^\varepsilon, 1_T), (S', S_{i,1}^\varepsilon)) &\sim_{x_{i,1}} ((q_{i,1}, 1_T), (S', S_{i,1})) \\ &\sim_\varepsilon^* ((q_{i,2}^\varepsilon, 1_T), (S', S_{i,2}^\varepsilon)) \sim_{x_{i,2}} ((q_{i,2}, 1_T), (S', S_{i,2})) \\ &\vdots \\ &\sim_\varepsilon^* ((q_{i,l(u_i)}^\varepsilon, 1_T), (S', S_{i,l(u_i)}^\varepsilon)) \sim_{x_{i,l(u_i)}} ((q_{i,l(u_i)}, 1_T), (S', S_{i,l(u_i)})) \\ &\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)). \end{aligned}$$

We shall now construct the  $i^{\text{th}}$  chain of type  $V$ . From the above, we see that the last configuration of the  $i^{\text{th}}$  chain of type  $U$  is

$$((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)).$$

Recall the  $i^{\text{th}}$  chain of type  $V$  is the one responsible for processing

$$v_i = y_{i,1}u'_{i,1}y_{i,2}u'_{i,2} \cdots y_{i,l(v_i)-1}u'_{i,l(v_i)-1}y_{i,l(v_i)}.$$

Upon reading  $y_{i,1}$  there is a transition in (T5)(a) that moves from state  $(q_{i+1,1}^\varepsilon, 1_T)$  to state  $(q_{i+1,1}^\varepsilon, 1_T, q_{(q_{i+1,1}^\varepsilon, 1_T, 1)})$  that pushes the first letter of  $w_{1_T y_{i,1}}$ , say  $b_1$  onto the pushdown stack. Thus the chain starts

$$((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)) \sim_{y_{i,1}} ((q_{i+1,1}^\varepsilon, 1_T, q_{(q_{i+1,1}^\varepsilon, 1_T, 1)}), (S', b_1 S_{i+1,1}^\varepsilon)).$$

The transitions in (T5)(b) simulate reading  $w_{1_T y_{i,1}}$  in  $P_{F_n}^{(W,1)}$ , and thus at the end of applying transitions in (T5)(b), the chain continues as

$$\begin{aligned} ((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)) &\sim_{y_{i,1}} ((q_{i+1,1}^\varepsilon, 1_T, q_{(q_{i+1,1}^\varepsilon, 1_T, 1)}), (S', b_1 S_{i+1,1}^\varepsilon)) \\ &\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)). \end{aligned}$$

Reading an input letter from  $X_B$  with the state being  $(q_{i+1,1}^\varepsilon, t_{i,1})$  the automaton uses transitions in (T6), and the state and the stack do not change. Therefore the chain above continues as

$$\begin{aligned} ((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)) &\sim_{y_{i,1}} ((q_{i+1,1}^\varepsilon, 1_T, q_{(q_{i+1,1}^\varepsilon, 1_T, 1)}), (S', b_1 S_{i+1,1}^\varepsilon)) \\ &\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)) \\ &\sim_{u'_{i,1}} ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)). \end{aligned}$$

We shall now assume that we have constructed the  $i^{\text{th}}$  chain of type  $V$  up to reading  $y_{i,j} u'_{i,j}$ . We shall now construct the part reading  $y_{i,j+1} u'_{i,j+1}$ . The last configuration in the part of the chain reading  $y_{i,1} u'_{i,1} y_{i,2} u'_{i,2} \cdots y_{i,j} u'_{i,j}$  is

$$((q_{i+1}^\varepsilon, t_{i,j}), (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}}) S_{i+1,1}^\varepsilon)).$$

Upon reading  $y_{i,j+1}$  there is a transition in (T5)(a) that moves from state  $(q_{i+1}^\varepsilon, t_{i,j})$  to state  $(q_{i+1}^\varepsilon, t_{i,j}, q_{(q_{i+1}^\varepsilon, t_{i,j}, 1)})$  that processes the first letter of  $w_{t_{i,j} y_{i,j+1}}$ , say  $b_{j+1}$  the same way  $P_T^{(W,1)}$  would if it the symbol at the top of its stack was the first letter of  $\rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}})$ . Therefore the chain continues

$$\begin{aligned} ((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)) &\sim_{y_{i,1}} ((q_{i+1,1}^\varepsilon, 1_T, q_{(q_{i+1,1}^\varepsilon, 1_T, 1)}), (S', b_1 S_{i+1,1}^\varepsilon)) \\ &\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)) \\ &\sim_{u'_{i,1}} ((q_{i+1,1}^\varepsilon, t_{i,1}), (S, \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)) \\ &\vdots \\ &\sim_{y_{i,j+1}} (q_{i+1}^\varepsilon, t_{i,j}, q_{(q_{i+1}^\varepsilon, t_{i,j}, 1)}), \\ &\quad (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}} b_{j+1} S_{i+1,1}^\varepsilon)). \end{aligned}$$

The transitions in (T5)(b) simulate reading  $w_{t_{i,j} y_{i,j+1}}$ , and thus as the end of ap-

plying transitions in (T5)(b), the chain continues as

$$\begin{aligned}
((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)) &\sim_{y_{i,1}} ((q_{i+1,1}^\varepsilon, 1_T, q_{(q_{i+1,1}^\varepsilon, 1_T, 1)}), (S', b_1 S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)) \\
&\sim_{u'_{i,1}} ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)) \\
&\vdots \\
&\sim_{y_{i,j+1}} ((q_{i+1}^\varepsilon, t_{i,j}, q_{(q_{i+1}^\varepsilon, t_{i,j}, 1)}), \\
&(S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}} b_{j+1}) S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{i+1}^\varepsilon, t_{i,j}), \\
&(S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}} w_{t_{i,j} y_{i,j+1}}) S_{i+1,1}^\varepsilon)).
\end{aligned}$$

Reading an input letter from  $X_B$  with state being  $(q_{i+1}^\varepsilon, t_{i,j})$  the automaton uses transitions in (T6), and the state and the stack do not change. Therefore the chain above continues as

$$\begin{aligned}
((q_{i+1,1}^\varepsilon, 1_T), (S', S_{i+1,1}^\varepsilon)) &\sim_{y_{i,1}} ((q_{i+1,1}^\varepsilon, 1_T, q_{(q_{i+1,1}^\varepsilon, 1_T, 1)}), (S', b_1 S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)) \\
&\sim_{u'_{i,1}} ((q_{i+1,1}^\varepsilon, t_{i,1}), (S', \rho(w_{1_T y_{i,1}}) S_{i+1,1}^\varepsilon)) \\
&\vdots \\
&\sim_{y_{i,j+1}} ((q_{i+1}^\varepsilon, t_{i,j}, q_{(q_{i+1}^\varepsilon, t_{i,j}, 1)}), \\
&(S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}} b_{j+1}) S_{i+1,1}^\varepsilon)) \\
&\sim_\varepsilon^* ((q_{i+1}^\varepsilon, t_{i,j}), \\
&(S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}} w_{t_{i,j} y_{i,j+1}}) S_{i+1,1}^\varepsilon)) \\
&\sim_{u'_{i,j+1}} ((q_{i+1}^\varepsilon, t_{i,j}), \\
&(S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,j-1} y_{i,j}} w_{t_{i,j} y_{i,j+1}}) S_{i+1,1}^\varepsilon)).
\end{aligned}$$

Thus we see that the  $i^{\text{th}}$  chain of type  $V$  ends with the configuration

$$((q_{i+1,1}^\varepsilon, t_{i,l(v_i)}), (S', \rho(w_{1_T y_{i,1}} w_{t_{i,1} y_{i,2}} \cdots w_{t_{i,l(v_i)-1} y_{i,l(v_i)}}) S_{i+1,1}^\varepsilon)).$$

However, since  $y_{i,1} y_{i,2} \cdots y_{i,l(v_i)} =_T 1_T$  then it must be that  $t_{i,l(v_i)} = 1_T$  and the stack is empty (as seen in the proof of  $\mathcal{L}(P_T^{(W,1)}) = WP(T, X_T)$  in [2.3.2](#)). Since  $t_{i,l(v_i)} = 1_T$  the automaton can now use transitions in (T4) to continue reading  $u_{i+1}$  through the  $i+1^{\text{th}}$  chain of type  $U$ .  $\blacksquare$

**Lemma 4.5.8.** *Let  $\omega = \sigma_1 \sigma_2 \cdots \sigma_r$ . The only chains by which  $\mathcal{A}$  reads  $\omega$  are those in Lemma [4.5.6](#) and Lemma [4.5.7](#).*



*Proof.* We first observe that  $q_0$  is the only initial state. Further, the only transitions from  $q_0$  are those in (T0).

Observe the transitions in (T0)(a) move the state from  $q_0$  to  $q$ , where  $q \in I_T$ . However, we note that  $I_T$  consists of one state, in particular  $1_T$ . Further observe that transitions from  $1_T$  are to states in  $Q_T$ , and any transition from a state in  $Q_T$  is to a state that is also in  $Q_T$ . We see this from the transitions in (T1), as they are the only transitions from states in  $Q_T$ . Therefore if in a chain reading  $\omega$ ,  $\mathcal{A}$  first uses the transition in (T0)(a), then it must use only transitions in (T1) for the rest of the chain. We also note that the transitions in (T1)(a) are those from  $\delta_T$ , but they are modified to account for  $\mathcal{A}$  being a CSPD automaton. Therefore, a transition from  $q$  to  $q'$  upon reading  $x \in X_T$  in (T1)(a) has the same effect on the pushdown stack as its analogue in  $\delta_T$  does on the stack of  $\mathcal{A}_T$ . Thus if  $\mathcal{A}_T$  reads  $y_1 y_2 \cdots y_s$  through the chain

$$\begin{aligned} (1_T, \perp) &\sim_{y_1} ((1_T, y_1), \perp) \sim_{\varepsilon}^* (t_1, \rho(w_{1_T y_1}) \perp) \\ &\sim_{y_2} ((t_1, y_2), \rho(w_{1_T y_1}) \perp) \sim_{\varepsilon}^* (t_2, \rho(w_{1_T y_1} w_{t_1 y_2}) \perp) \\ &\vdots \\ &\sim_{y_n} ((t_{n-1}, y_n), \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp) \\ &\sim_{\varepsilon}^* (t_n, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}} w_{t_{n-1} y_n}) \perp), \end{aligned}$$

where  $t_0 = 1_T$  and  $t_{i-1} y_i =_T w_{t_{i-1} y_i} t_i$  then  $\mathcal{A}$  reads  $y_1 y_2 \cdots y_s$  through the following chain

$$\begin{aligned} (q_0, (S, \perp)) &\sim_{\varepsilon} (1_T, (S, \perp)) \\ &\sim_{y_1} ((1_T, y_1), (S, \perp)) \sim_{\varepsilon}^* (t_1, (S, \rho(w_{1_T y_1}) \perp)) \\ &\sim_{y_2} ((t_1, y_2), (S, \rho(w_{1_T y_1}) \perp)) \\ &\sim_{\varepsilon}^* (t_2, (S, \rho(w_{1_T y_1} w_{t_1 y_2}) \perp)) \\ &\vdots \\ &\sim_{y_n} ((t_{n-1}, y_n), (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp)) \\ &\sim_{\varepsilon}^* (t_n, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}} w_{t_{n-1} y_n}) \perp)) \end{aligned}$$

for a long enough choice of check-stack  $S$ . Now suppose  $\omega = u_1 y_1 u_2 y_2 \cdots u_n y_n$ , where  $u_i \in X_B^*$ . We note that the transitions in (T1)(b) read letters from  $X_B$  from a state  $t$  representing a transversal element in  $Q_T$  to state  $t$ , without editing the stack. Thus, if  $\omega$  is read through a chain where the first transition is from (T0)(a)

then the chain must be

$$\begin{aligned}
(q_0, (S, \perp)) &\sim_\varepsilon (1_T, (S, \perp)) \\
&\sim_{u_1} (1_T, (S, \perp)) \sim_{y_1} ((1_T, y_1), (S, \perp)) \sim_\varepsilon^* (t_1, (S, \rho(w_{1_T y_1}) \perp)) \\
&\sim_{u_2} (t_1, (S, \rho(w_{1_T y_1}) \perp)) \sim_{y_2} ((t_1, y_2), (S, \rho(w_{1_T y_1}) \perp)) \\
&\sim_\varepsilon^* (t_2, (S, \rho(w_{1_T y_1} w_{t_1 y_2}) \perp)) \\
&\vdots \\
&\sim_{u_n} (t_{n-1}, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp)) \\
&\sim_{y_n} ((t_{n-1}, y_n), (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}}) \perp)) \\
&\sim_\varepsilon^* (t_n, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{n-2} y_{n-1}} w_{t_{n-1} y_n}) \perp)),
\end{aligned}$$

as in Lemma [4.5.6](#)

Now we suppose the first transition that was used was not in (T0)(a). The only other transition from  $q_0$  is that in (T0)(b) to  $q_l$  without editing the stack. Before we continue, we see from the transitions that there is no state in  $Q_T$  that can be reached by a sequence of transitions from  $q_l$ . We shall suppose that the check-stack is long enough, we will explain what the check-stack must be later. We will denote the check-stack by  $S'$ .

We note that the transitions in (T2) are those from  $q_l$ , with (T2)(a) and (T2)(b) being from  $q_l$  to  $q_l$ . As transitions in (T2)(b)1. do not add a letter  $z^{-1}$  on top of  $z$  on the pushdown stack, the string written on the pushdown stack after repeated usage of transitions in (T2)(a) and (T2)(b) is freely reduced. Suppose that string is  $u$ . The only transitions from  $q_l$  to another state are those in (T2)(c). Transitions in (T2)(c) move the state from  $q_l$  to a state  $t^c$  without editing the stack. A freely reduced string  $u$  in  $\chi_T$  and the transversal element  $t$  define an element of  $y \in T$  as every element of  $T$  can be expressed as a product of an element of the free group which is of finite index in  $T$  and a transversal element. Thus the chain starts as

$$(q_0, (S', \perp)) \sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u \perp)).$$

Observe the only transitions from a state with  $c$  as a superscript are those in (T3). The  $\varepsilon$ -transition in (T3)(d) is the only transition in (T3) from a state with  $c$  as a superscript to a state without  $c$  as a superscript. We note that the transitions in (T3)(b) and (T3)(c) are analogous to transitions of  $P_T^{(W,1)}$ , and thus have the same effect. Further, the only way to use (T3)(d) is for the state to be  $1_T^c$  and the pushdown stack being empty, and there is no other transition from  $1_T^c$  with the stack being empty. As the transitions in (T3)(b) and (T3)(c) simulate  $P_T^{(W,1)}$ ,  $\mathcal{A}$  reads the letters of  $\omega$  and processes the letters from  $X_T$  the same way  $P_T^{(W,1)}$  does. That is, the automaton  $\mathcal{A}$  moves through the same configurations (while accounting for the check-stack and superscript  $c$ ). The automaton  $\mathcal{A}$  ignoring the

letters from  $X_B$  as these are read using the transitions from (T3)(a), and these transitions do not change the state or stack. This automaton processes the letters of  $\omega$  in this way until it reaches the state  $1_T^c$  and the pushdown stack becomes empty. Then the only transition that can be used is the one in (T3)(d). Thus the automaton  $\mathcal{A}$  has read the shortest prefix  $\omega_0$  of  $\omega$  such that  $ut\varphi_{X_T}(\omega_0) =_T 1_T$ . We note that since the only transition from  $1_T^c$  with the stack being empty is in (T3)(d), there cannot exist another prefix  $\omega_1$  of  $\omega_0$  such that  $ut\varphi_{X_T}(\omega_1) =_T 1_T$  that is processed in the way  $\omega_0$  is processed. Otherwise there must be a transition from  $1_T^c$  with the stack being empty to another state with  $c$  as a superscript.

Suppose  $\omega_0 = u_{0,1}y'_1u_{0,2}y'_2 \cdots u_{0,n}y'_n$  where  $u_{0,i} \in X_B^*$  and  $y'_i \in X_T$ . Further set  $t_{0,0} = t$ , and  $t_{0,i-1}y'_i =_T w_{t_{0,i-1}y'_i}t_{0,i}$ . Then by our observations in the above paragraph, the chain must continue as

$$\begin{aligned}
(q_0, (S', \perp)) &\sim_\varepsilon (q_l, (S', \perp)) \sim_\varepsilon^* (t^c, (S', u\perp)) \\
&\sim_{u_{0,1}} (t^c, (S', u\perp)) \sim_{y'_1} ((t^c, y'_1)^c, (S', u\perp)) \sim_\varepsilon^* (t_{0,1}^c, (S', \rho(uw_{t_0y'_1})\perp)) \\
&\sim_{u_{0,2}} (t_{0,1}^c, (S', \rho(uw_{t_0y'_1})\perp)) \sim_{y'_2} ((t_{0,1}^c, y'_2)^c, (S', \rho(uw_{t_0y'_1})\perp)) \\
&\sim_\varepsilon^* (t_{0,2}^c, (S', \rho(uw_{t_0y'_1}w_{t_{0,1}y'_2})\perp)) \\
&\vdots \\
&\sim_{u_{0,n}} (t_{0,n-1}^c, (S', \rho(uw_{t_0y'_1}w_{t_{0,1}y'_2} \cdots w_{t_{0,n-2}y'_{n-1}})\perp)) \\
&\sim_{y'_n} ((t_{0,n-1}^c, y'_n)^c, (S', \rho(uw_{t_0y'_1}w_{t_{0,1}y'_2} \cdots w_{t_{0,n-2}y'_{n-1}})\perp)) \\
&\sim_\varepsilon^* (t_{0,n}, (S', \rho(uw_{t_0y'_1}w_{t_{0,1}y'_2} \cdots w_{t_{0,n-2}y'_{n-1}}w_{t_{0,n-1}y'_n})\perp)) = (1_T^c, (S', (\perp))) \\
&\sim_\varepsilon ((q', 1_T), (S', (\perp))),
\end{aligned}$$

for some  $q' \in I_B$ .

We note that the above is the initial chain as in Lemma [4.5.7](#)

We shall now explore how  $\mathcal{A}$  must read the remainder of  $\omega$ . We can express  $\omega$  as follows

$$\omega = \omega_0 u_1 v_2 u_2 v_2 \cdots u_n v_n$$

where

$$u_i = x_{i,1}x_{i,2} \cdots x_{i,l(u_i)} \in X_B^*,$$

and  $v_i \in (X_B \cup X_T)^*$  such that the first and last letter of  $v_i$  is in  $X_T$  for all  $1 \leq i \leq n$ .

Suppose the current active state is  $(q, t)$  where  $q \in Q_B$  and  $t \in T'$ . Then suppose the automaton processes  $v_i$ . Upon reading an input letter from  $y \in X_T$  the automaton must use a transition in (T5)(a) as these are the only transitions that can read input letters from  $X_T$ . We then observe the rest of the transitions in (T5)(b) and (T5)(c) simulate the processing of  $y$  in  $P_T^{(W,1)}$  from state  $t$ , while treating symbols from  $\Gamma_B$  as bottom of stack symbols. Further at the end of

simulating processing  $y$  in  $P_T^{(W,1)}$  the automaton is at a state  $(q, t')$  where  $ty =_T w_{ty}t'$ . We note that these transitions must be used as there are no other transitions that can be used. From state  $(q, t')$ , if  $t' \neq 1_T$  then if an input letter from  $X_B$  is read it must be read using (T6)(a) as these are the only transitions reading letters from  $X_B$  from a state in  $Q_1$  whose second coordinate is not  $1_T$ . Similarly if  $t' = 1_T$  but the top of the pushdown stack contains a symbol from  $\chi_T$ , and an input letter from  $X_B$  is read then it must be read using (T6)(b) as there are no other transitions that can be used to read an input letter from  $X_B$  in that case. Therefore we ignore letters from  $X_B$  except when the second coordinate of the state is  $1_T$  and the pushdown stack does not contain a symbol from  $\chi_T$  at the top. That is, if  $(q, t) = (q, 1_T)$  and at the end of reading  $v_i$  the automaton is back at  $(q, 1_T)$  with the symbol at top of the pushdown stack not being from  $\chi_T$ , then it must be that  $\varphi_{X_T}(v_i) =_T 1_T$  (by our construction of  $P_T^{(W,1)}$ ). Therefore we may assume that  $\varphi_{X_T}(v_i) =_T 1_T$  for all  $i \leq n - 1$  and further that there is no prefix  $v'_i$  of  $v_i$  such that  $\varphi_{X_T}(v'_i) =_T 1_T$ .

Now recall that at the end of the initial chain above, the last configuration is  $(q', 1_T)$ . Further we note that upon reading an input letter from  $X_B$  with the second coordinate of the state being  $1_T$  and the top of the pushdown stack not being a letter from  $\chi_T$  the automaton must use transitions in (T4) as there are no other transition reading letters in of  $X_B$  in that case. We further observe that these transitions are from  $\delta_B$  but were modified to only change the first coordinate of the state. Therefore, using these transitions does not change the second coordinate of the state. So at the end of reading  $u_1$  the second coordinate of the state will still be  $1_T$ . This, together with the previous paragraph implies the automaton processes the  $u_i$  and the  $v_i$  in the following way.

1. The  $u_i$  are processed using transitions in (T4), and since these transitions come from  $\delta_B$  we see that if  $\mathcal{A}$  processes the non-contiguous substring  $u_1u_2 \cdots u_n$  of  $\omega$  through a chain (after the initial chain) then it must be that that chain can be induced from a chain that  $\mathcal{A}_B$  uses to read  $u_1u_2 \cdots u_n$ . Thus we can partition the chains into  $n$  parts in the following way. The  $i^{\text{th}}$  part of the chain of  $\mathcal{A}$  processes  $u_i$  in the same way the  $i^{\text{th}}$  part of the chain of  $\mathcal{A}_B$  processes  $u_i$ . This  $i^{\text{th}}$  part of the chain of  $\mathcal{A}$  is the  $i^{\text{th}}$  chain of type  $U$  in Lemma [4.5.7](#)
2. The  $v_i$  are processed using transitions in (T5) (while ignoring letters from  $X_B$  using (T6)), thus producing a chain of type  $V$  as in the statement of Lemma [4.5.7](#)

Now, as a chain reading  $\omega$  is induced by a chain of  $\mathcal{A}_B$  reading  $u_1u_2 \cdots u_n$ , it must be that the string  $r_1$  on the check-stack in the chain of  $\mathcal{A}_B$  reading  $u_1u_2 \cdots u_n$  is a prefix of the string  $r_2$  on the check-stack in the chain of  $\mathcal{A}$  reading  $\omega$  that we have

just described. Further, as  $\mathcal{R} = \mathcal{R}_B P^*$  then  $r_2 = r_1 P^m$  for some  $m \geq 0$  such that  $r_2$  is long enough for  $\mathcal{A}$  to process  $\omega$ .  $\blacksquare$

We are now ready to present the proof of Theorem [4.5.1](#).

*Proof of Theorem [4.5.1](#).* Let  $\omega \in (X_B \cup X_T)^*$ , and let  $w$  be the element in  $W$  such that  $w =_W \omega$ . Suppose  $\omega$  is read by  $\mathcal{A}$ . Then by Lemma [4.5.8](#) reads  $\omega$  through the chains in Lemma [4.5.6](#) and Lemma [4.5.7](#).

Suppose  $\omega \in \text{coWP}(W, X_B \cup X_T)$ . Then either the second coordinate of  $w$  is non-trivial, or the first coordinate is non-trivial.

Suppose the second coordinate of  $w$  is non-trivial. Suppose  $\omega = a_1 y_1 a_2 y_2 \cdots a_r y_r$  where  $a_i \in X_B^*$  and  $y_i \in X_T^*$  for all  $1 \leq i \leq r$ . Then the second coordinate is  $y_1 y_2 \cdots y_r$ . By Lemma [4.5.6](#)  $\mathcal{A}$  reads  $\omega$  through the following chain

$$\begin{aligned}
(q_0, (S, \perp)) &\sim_\varepsilon (1_T, (S, \perp)) \\
&\sim_{a_1} (1_T, (S, \perp)) \sim_{y_1} ((1_T, y_1), (S, \perp)) \sim_\varepsilon^* (t_1, (S, \rho(w_{1_T y_1}) \perp)) \\
&\sim_{a_2} (t_1, (S, \rho(w_{1_T y_1}) \perp)) \sim_{y_2} ((t_1, y_2), (S, \rho(w_{1_T y_1}) \perp)) \\
&\sim_\varepsilon^* (t_2, (S, \rho(w_{1_T y_1} w_{t_1 y_2}) \perp)) \\
&\vdots \\
&\sim_{a_r} (t_{r-1}, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{r-2} y_{r-1}}) \perp)) \\
&\sim_{y_r} ((t_{r-1}, y_r), (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{r-2} y_{r-1}}) \perp)) \\
&\sim_\varepsilon^* (t_r, (S, \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{r-2} y_{r-1}} w_{t_{r-1} y_r}) \perp))
\end{aligned}$$

for a long-enough choice of check-stack  $S$ . Since  $y_1 y_2 \cdots y_r \neq_T 1_T$ , either  $t_r \neq 1_T$  or  $\rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{r-2} y_{r-1}} w_{t_{r-1} y_r}) \neq \varepsilon$  as

$$y_1 y_2 \cdots y_r =_T \rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{r-2} y_{r-1}} w_{t_{r-1} y_r}) t_r.$$

If  $t_r \neq 1_T$  then  $t_r \in F_T \subseteq F$ , and thus  $\omega$  is accepted. If

$$\rho(w_{1_T y_1} w_{t_1 y_2} \cdots w_{t_{r-2} y_{r-1}} w_{t_{r-1} y_r}) \neq \varepsilon$$

while  $t_r = 1_T$  then there is a  $\varepsilon$ -transition in  $\mathcal{A}_T$  (and thus there is analogue of it in (T1)(a)) from  $1_T$  to  $\Psi \in F_T$  when the pushdown stack is non-empty, and thus  $\omega$  is accepted.

Now suppose the first coordinate is non-trivial. Recall that the first coordinate is a tuple. Since the first coordinate is non-trivial, then it must be non-trivial in a certain coordinate, say  $h \in T$ . It must be that there exists a shortest prefix  $\omega_0$  (where the last letter of  $\omega_0$  is in  $X_T$ ) of  $\omega$  such that  $\varphi_{X_T}(\omega_0) =_T h$ . We may express  $h^{-1}$  as a product of an element of  $F_n$ , say  $u$ , and a transversal element, say  $t$ .

Thus we may express  $\omega = \omega_0 u_1 v_1 u_2 v_2 \cdots u_n v_n$ , where  $u_i \in X_B^*$  and  $v_i \in (X_B \cup X_T)^*$  such that the first and last letter of  $v_i$  is in  $X_T$ , for all  $1 \leq i \leq n$ . We also suppose that  $\varphi_{X_T}(v_j) =_T 1_T$  for all  $1 \leq j \leq n-1$ . Further suppose for each  $1 \leq j \leq n-1$ , there does not exist a proper prefix  $v'_j$  of  $v_j$  such that  $\varphi_{X_T}(v'_j) =_T 1_T$ .

Since the first coordinate is non-trivial, and is non-trivial at coordinate  $h$ . Then it must be the entry in that coordinate is  $u_1 u_2 \cdots u_n$ . Since  $u_1 u_2 \cdots u_n \neq_B 1_B$  then there exists a chain  $C$  by which  $\mathcal{A}_B$  reads  $u_1 u_2 \cdots u_n$  and the last configuration in  $C$  is an accept configuration. By Lemma 4.5.7 we see  $B$  induces a chain reading  $\omega$  as in Lemma 4.5.7. We note the state of the last configuration in the  $n^{\text{th}}$  chain of type  $V$  has as its first coordinate the same state of  $Q_B$  as the first coordinate of the state of the last configuration of the  $n^{\text{th}}$  chain of type  $U$ . This first coordinate is the state in the last configuration of the chain  $C$ . That is, this state is an accept state of  $\mathcal{A}_B$ .

Therefore if  $\omega \in \text{coWP}(W, X_B \cup X_T)$  then  $\omega$  is accepted.

Conversely, suppose  $\omega' \in \text{WP}(W, X_G \cup X_T)$  then  $\omega' = W1_W$ . Therefore the first coordinate is trivial and the second coordinate is also trivial.

Suppose  $\omega' = b_1 y'_1 b_2 y'_2 \cdots b_{r_1} y'_{r_1}$  where  $b_i \in X_B^*$ , and  $y'_i \in X_T$ . The second coordinate is  $y_1 y_2 \cdots y_{r_1}$ . Since the second coordinate is trivial then  $y_1 y_2 \cdots y_{r_1} =_T 1_T$ . Thus the chain reading  $\omega$  in Lemma 4.5.6 ends with the configuration  $(1_T, (S, \perp))$  for some long enough check-stack. However, there is no  $\varepsilon$ -transition in  $\mathcal{A}_T$  from  $1_T$  with the stack being empty to an accept state of  $\mathcal{A}_T$ . Therefore using the chain in Lemma 4.5.6,  $\mathcal{A}$  cannot accept  $\omega'$ .

For the first coordinate of  $1_W$  to be trivial, it follows that the tuple consists of the identity element of  $B, 1_B$  in every coordinate. That is, regardless of what freely reduced string and transversal element we pick (and consequently prefix  $\omega'_0$  of  $\omega$ ), there is no decomposition of  $\omega'$  whereby  $\omega' = \omega'_0 u_1 v_1 u_2 v_2 \cdots u_m v_m$  with  $u_1 u_2 \cdots u_m \in \text{coWP}(B, X_B)$ , where  $u_i$  and  $v_i$  are as in the statement of Lemma 4.5.7. Therefore for every chain reading  $u_1 u_2 \cdots u_m$  in  $\mathcal{A}_B$ , the chain ends at a configuration whose state is not an accept state. Therefore the chain in  $\mathcal{A}$  induced by Lemma 4.5.7 reading  $\omega'$  also ends at a configuration whose state is not an accept state.

Therefore if  $\omega' \in \text{WP}(W, X_B \cup X_T)$  then  $\omega'$  is not accepted. ■

## 4.6 The free product $\mathbb{Z}^n * \mathbb{Z}^m$

In this section we shall prove the following theorem.

**Theorem 4.6.1.** *The free product  $\mathbb{Z}^n * \mathbb{Z}^m$  is co-ETOL.*

Similar to previous sections, we shall start by giving a high level intuitive summary of how and why the automaton works. Then we present the intuitive

idea giving a detailed intuitive account of how the automaton works, and then we formally define the automaton accepting the co-word problem of  $\mathbb{Z}^n * \mathbb{Z}^m$ . We then give a proof that the automaton does indeed accept the co-word problem of  $\mathbb{Z}^n * \mathbb{Z}^m$ . Finally we provide some examples to aid in understanding the automaton, and the reader may wish to go through the examples before reading the proof. Our construction is motivated by the construction used in proving Theorem 11 of [28].

### 4.6.1 Definition of Automaton

Let  $\mathbb{Z}^n$  be generated by the set  $\{a_1, a_2, \dots, a_n\}$ , and let  $\mathbb{Z}^m$  be generated by the set  $\{b_1, b_2, \dots, b_m\}$ . Define the set  $X_n$  to be the symmetric closure of the generating set of  $\mathbb{Z}^n$ , i.e  $X_n := \{a_1^\pm, a_2^\pm, \dots, a_n^\pm\}$ . Similarly, define the set  $Y_m := \{b_1^\pm, b_2^\pm, \dots, b_m^\pm\}$  to be the symmetric closure of the generating set of  $\mathbb{Z}^m$ .

We will also define sets  $X_n^c := \{A_1, A_2, \dots, A_n\}$  and  $Y_m^c := \{B_1, B_2, \dots, B_m\}$ . (Note that the elements of these sets are capitalised versions of the letters in the generating sets of  $\mathbb{Z}^n$  and  $\mathbb{Z}^m$ , respectively. The fact they are capitalised versions is why  $X_n^c$  and  $Y_m^c$  have the superscript  $c$ .)

For the purpose of the following intuitive summary, we describe how the automaton accepts a string. We first describe what a string on the check-stack looks like. Given a string  $\omega$  not representing the identity element of  $\mathbb{Z}^n * \mathbb{Z}^m$ , we describe how to select a string  $\nu$  in the regular language associated to the automaton, such that with  $\nu$  on the check-stack the automaton accepts  $\omega$ . Then we describe the contents of the pushdown stack at the end of processing. We also provide a figure to aid in understanding what is stored on the pushdown stack.

**Intuitive Summary 4.6.2.** We call any element of  $X_n^c$  an  $A$ -type letter, and any element of  $Y_m^c$  a  $B$ -type letter. The regular language associated to the check-stack will be a subset of  $(X_n^c \cup Y_m^c)^*$ . We shall now describe a process by which we can obtain a string on the check-stack that will be used to show that an input string representing a non-identity element of  $\mathbb{Z}^n * \mathbb{Z}^m$  is accepted.

Let  $\omega = \omega_1\omega_2 \cdots \omega_k$  be an input string with  $\omega_i$  being syllables of  $\omega$ . Let  $\bar{\omega} = \xi_1\xi_2 \cdots \xi_t$  be the normal form expression for the element of  $\mathbb{Z}^n * \mathbb{Z}^m$  representing  $\omega$ , with  $\xi_i$  being syllables of  $\bar{\omega}$ . Further suppose  $t > 0$  and thus  $\omega$  does not represent the identity element of  $\mathbb{Z}^n * \mathbb{Z}^m$ .

Since  $\bar{\omega}$  is in normal form then each syllable  $\xi_i$  is non-trivial in its respective group. Thus there exists a generator of that group with non-zero exponent sum in  $\xi_i$ . We denote this generator by  $x_i$ , for all  $1 \leq i \leq t$ . The image of the string  $x_1x_2 \cdots x_t$  under the extension of the following map

$$\begin{aligned} a_j &\mapsto A_j \text{ for all } j \in \{1, 2, \dots, n\} \\ b_l &\mapsto B_l \text{ for all } l \in \{1, 2, \dots, m\} \end{aligned}$$

is a string over  $X_n^c \cup Y_m^c$  such that if the  $j^{\text{th}}$  letter belongs to  $X_n^c$  then the  $j + 1^{\text{th}}$  letter belongs to  $Y_m^c$  (and vice versa).

Let  $\mu$  be the image of the string  $x_1^{p_1} x_2^{p_2} \dots x_t^{p_t}$  under the map above, for suitably large  $p_1, p_2, \dots, p_t$ . If the first letter of  $\mu$  is a  $B$ -type letter then append onto the beginning of  $\mu$  a prefix of the form  $A_l^{p_0}$  for some  $l \in \{1, 2, \dots, n\}$  and a suitably large  $p_0$  to form a string  $\mu'$ . Further if the last letter of  $\mu'$  is an  $A$ -type letter, append onto the end of  $\mu'$  a suffix of the form  $B_{l'}^{p_{t+1}}$  for some  $l' \in \{1, 2, \dots, m\}$  and a suitably large  $p_{t+1}$  to form a string  $\mu''$ . Finally we form a final string  $\mu^{(3)}$  by appending a long enough suffix of the form

$$A_{i_1}^{s_1} B_{j_1}^{s'_1} A_{i_2}^{s_2} B_{j_2}^{s'_2} \dots A_{i_l}^{s_l} B_{j_l}^{s'_l},$$

for some  $i_1, i_2, \dots, i_l \in \{1, 2, \dots, n\}$ ,  $j_1, j_2, \dots, j_l \in \{1, 2, \dots, m\}$ , and suitably large  $l, s_1, s'_1, s_2, s'_2, \dots, s_l, s'_l$ . It is with  $\mu^{(3)}$  on the check-stack, that the automaton will recognise that  $\omega$  does not represent the identity element of  $\mathbb{Z}^n * \mathbb{Z}^m$ .

**Remark 4.6.3.** The core of the processing will be done using the substring  $\mu$  of the check-stack, and we pick exponents to be suitably large in order to ensure that the processing will be completed. That is, the automaton will not stop (and thus reject) due to an insufficiently long string on the check stack. However,  $\mu$  is extended to  $\mu^{(3)}$  to ensure that the string is long enough to allow the processing of all syllables of  $\omega$  (regardless of whether they trivialise or not). Another reason of extending  $\mu$  to  $\mu^{(3)}$  is to ensure that the string is in our chosen regular language, which has strings of the form

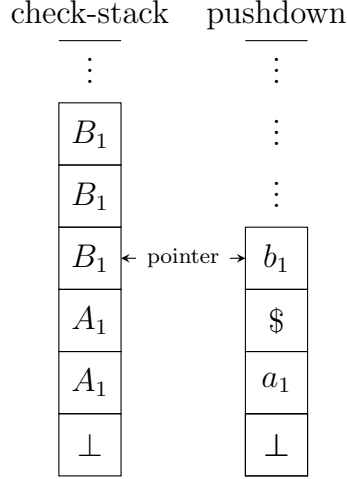
$$A_{i_1}^{s_1} B_{j_1}^{s'_1} A_{i_2}^{s_2} B_{j_2}^{s'_2} \dots A_{i_l}^{s_l} B_{j_l}^{s'_l},$$

where  $i_1, i_2, \dots, i_l \in \{1, 2, \dots, n\}$ ,  $j_1, j_2, \dots, j_l \in \{1, 2, \dots, m\}$ , and  $l, s_1, s'_1, s_2, s'_2, \dots, s_l, s'_l > 0$ . ♠

The automaton will use the pushdown stack to store the projection of each syllable  $\xi_i$  onto  $\langle x_i \rangle$ . It does so by first using bookkeeping states and padding symbols to search for the image of  $x_i$  under the map defined above. Then the automaton uses different bookkeeping states to freely reduce the projection of  $\xi_i$  onto  $(\{x_i\} \cup \{x_i\}^{-1})^*$ . Since  $\omega$  does not represent the identity of  $\mathbb{Z}^n * \mathbb{Z}^m$ , these projections will be non-trivial and thus the stack will be non-empty. The automaton will accept when the stack is non-empty at the end of processing which will happen for input strings that do not represent the identity, as we have outlined.

Below we show what the stacks look like at the end of a run processing the string  $\omega = a_1 a_1^{-1} a_2 b_1 b_1^{-1} a_2^{-1} a_1 b_1$  in  $\mathbb{Z}^2 * \mathbb{Z}$  with  $A_1 A_1 B_1 B_1 B_1$  on the check-stack. In the example below, we have chosen to denote our padding symbols with  $\$$  for simplicity.





We define a relation  $\bowtie$  between  $X_n \cup Y_m$  and  $X_n^c \cup Y_m^c$  as follows.

Let  $a \in \{a_1^\pm, a_2^\pm, \dots, a_n^\pm\}$ . Then  $a \bowtie A$  for all  $A \in \{A_1, A_2, \dots, A_n\}$ . Further let  $b \in \{b_1^\pm, b_2^\pm, \dots, b_m^\pm\}$ . Then  $b \bowtie B$  for all  $B \in \{B_1, B_2, \dots, B_m\}$ . Further, suppose  $d \in X_n \cup Y_m$  and  $D \in X_n^c \cup Y_m^c$  such that  $d \bowtie D$ . Then  $d = a_i^\pm$  or  $d = b_j^\pm$  for some  $i$  and  $j$ . If  $d = a_i^\pm$  and  $D = A_i$  then we write  $d \bowtie' D$ . Similarly if  $d = b_j^\pm$  and  $D = B_j$  then write  $d \bowtie' D$ .

We shall now give an intuitive description of the automaton  $\mathcal{A}$  accepting the  $coWP(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup Y_m)$ .

**Intuitive Idea:** The check-stack contains strings of the form

$$x_1^{i_1} y_1^{j_1} x_2^{i_2} y_2^{j_2} \dots x_t^{i_t} y_t^{j_t}$$

where  $x_l \in X_n^c$  and  $y_l \in Y_m^c$  for all  $1 \leq l \leq t$ , where  $t \geq 1$ , and  $i_1, i_2, \dots, i_t, j_1, j_2, \dots, j_t \geq 1$ . The automaton works in multiple stages, and they are as follows.

- Reading input letter stage
- Locating the correct corresponding check-stack letter stage
- Pushing input letter onto the pushdown stack stage
- Pre-check stage
- Action stage
- Clean up of the stack stage

- Check stage

**Reading input letter stage:** In this stage, the automaton simply reads an input letter  $\sigma$  and moves to the next stage while writing a  $W$  onto the pushdown stack, this letter is read regardless of the contents of the check-stack and pushdown stack.

(It maybe helpful to the reader to identify  $W$  with the word “wait”, this is because this effectively what the automaton is doing every time  $W$  is at the top of the pushdown stack. That is, the automaton must make a decision however all the information needed to make the decision is not yet available and so it is “waiting” and finding out more information to be able to make the decision.)

**Locating the correct corresponding check-stack letter stage:** Suppose the corresponding check-stack letter to  $W$  (written onto the pushdown stack in the previous stage) is  $X$ . The goal of this stage of the automaton is to locate the first check-stack letter  $Y$  that is “above”  $X$ , and  $\sigma \bowtie Y$ . (We use the word “above” here, thinking of the stacks as in the diagrams of [2.2.4.1](#).)

If  $\sigma \bowtie X$  then the automaton moves to the next stage. Otherwise, the automaton locates the first  $Y$  such that  $\sigma \bowtie Y$  where  $Y$  is above  $X$  on the check-stack. First the automaton deletes  $W$  in order to view the check-stack letter  $Z$  below  $X$ . If  $\sigma \bowtie Z$  then the automaton moves to  $q_X$ . (We view this as a rejection.) If  $\sigma$  is not  $\bowtie$  related to  $Z$  then the automaton writes  $WM$  onto the stack. We note that  $W$  is now one position higher than where it was before.

(It maybe helpful to the reader to identify  $M$  with the phrase “move on”, as if  $M$  was written the automaton must have viewed that position before via a  $W$  and does not need to test it again.)

Now the automaton can view the check-stack letter one position above  $X$ . This process repeats until the first  $Y$  on the check-stack such that  $\sigma \bowtie Y$  is located. The automaton then moves to the next stage.

**Pushing input letter onto the pushdown stack stage:** At the end of the previous stage, we have found the first check-stack letter  $Y$  such that  $\sigma \bowtie Y$  (or moved to  $q_X$ ). The automaton deletes  $W$  and writes  $\sigma$  and then moves to the next stage.

**Pre-check stage:** At this stage, the top of the pushdown stack is  $\sigma$  and the corresponding letter on the check-stack, say  $Y$  is such that  $\sigma \bowtie Y$ .

If  $\sigma \bowtie' Y$  then the automaton deletes  $\sigma$  and moves to the action stage.

Otherwise, the automaton the automaton deletes  $\sigma$  and moves to the clean up of the stack stage.

**Action stage:** At the end of the last stage, the automaton deleted  $\sigma$  from the top of the pushdown stack. If the top of the pushdown stack is  $\sigma^{-1}$  then we delete it. Otherwise, we write  $\sigma$  onto the pushdown stack.

At the end of this stage, the automaton moves to the clean up of the stack stage.

**Clean up of the stack stage:** If at this stage the top of the pushdown stack is  $M$  then the automaton deletes it. This repeats until the top of the pushdown stack is not  $M$ , then the automaton moves back to the reading input letter stage.

**Check stage:** After reading every letter in the way we just described, after the last letter has been read and the automaton moved back to the reading stage, the automaton moves to this stage. Here, if the pushdown stack is non-empty the automaton accepts the string.

We shall now give the formal definition of the automaton. We follow this with an informal description of the states. The reader may wish to take a detour there before reading the formal definition of  $\mathcal{A}$  as we link the states of the automaton to the stages discussed in the intuition above.

**Formal Definition of  $\mathcal{A}$ :** We define a CSPD automaton

$$\mathcal{A} := (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$$

accepting  $coWP(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup X_m)$  as follows.

- The state set  $Q$  consists of the following states.
  - The state  $q_r$ .  
(This is the state from which the input letters are read. This is why this state has  $r$  as a subscript.)
  - The state  $q_A$ .  
(This is the accept state, and the subscript  $A$  stands for “accept”.)
  - The state  $q_X$ .  
(This serves as a reject state.)
  - For every  $\sigma \in X_n \cup Y_m$  there is a state  $q_\sigma$
  - For every  $\sigma \in X_n \cup Y_m$  there are states  $q_{\sigma,l,1}$  and  $q_{\sigma,l,2}$ .  
(These states locate the correct entry on the check-stack. The  $l$  in the names of the state stand for the word “locate”.)
  - For every  $\sigma \in X_n \cup Y_m$  there is a state  $P.D._\sigma$ .  
(This state pushes the letter  $\sigma$  onto the pushdown stack. P.D. stands for pushdown.)
  - There is a state  $C_p$ .  
(This state checks whether the letter on the pushdown stack is  $\bowtie$  related to the corresponding letter on the check-stack. We view this as “pre-check”. Hence the subscript  $p$ .)

- For every  $\sigma \in X_n \cup Y_m$  there is a state  $A_\sigma$ .  
(This state simulates the processing of  $\sigma$ , we call this “acting” by  $\sigma$ .  
The letter  $A$  in the name of the state stands for the word “act”.)
- There is a state  $C_0$ .  
(This is referred to as the “clean-up” state.)
- There is a state  $C_1$ .  
(This is referred to as the “check state”.)
- The input alphabet is  $\Sigma = X_n \cup Y_m$ .
- The check-stack alphabet is  $\Delta = X_n^c \cup Y_m^c$ .
- The pushdown stack alphabet is  $\Gamma = X_n \cup Y_m \cup \{W, M\}$ .
- The set of initial states is  $I = \{q_r\}$ .
- The set of final states is  $F = \{q_A\}$ .
- The regular language  $\mathcal{R}$  is defined to consist of all the strings of the following form  $x_1^{i_1} y_1^{j_1} x_2^{i_2} y_2^{j_2} \dots x_t^{i_t} y_t^{j_t}$  where  $x_l \in X_n^c$  and  $y_l \in Y_m^c$  for all  $1 \leq l \leq t$ , where  $t \geq 1$ , and  $i_1, i_2, \dots, i_t, j_1, j_2, \dots, j_t \geq 1$ ,

and finally the transition relation  $\delta$  consists of the following transitions.

(T0) From  $q_r$  we have the following transitions. For every pair  $(X, Y)$  we have the following transitions.

(a) For every  $\sigma \in \Sigma$  there is a transition

$$((q_r, \sigma, (X, Y)), (q_\sigma, Y));$$

that is, there is a transition from  $q_r$  upon reading  $\sigma$  to  $q_\sigma$  for every pushdown stack letter  $Y$  and corresponding check-stack letter  $X$  that does not edit the stack.

(b) There is a transition

$$((q_r, \varepsilon, (X, Y)), (C_1, Y));$$

that is, there is an  $\varepsilon$ -transition from  $q_r$  to  $C_1$  without editing the stack.

(T1) For every  $\sigma \in \Sigma$ , from  $q_\sigma$  we have the following transitions. For every pair  $(X, Y)$  there is a transition

$$((q_\sigma, \varepsilon, (X, Y)), (q_{\sigma,l,1}, WY));$$

that is, there is an  $\varepsilon$ -transition from  $q_\sigma$  to  $q_{\sigma,l,1}$  that adds  $W$  onto the pushdown stack.

(T2) For every  $\sigma \in \Sigma$ , from  $q_{\sigma,l,1}$  we have the following transitions. For every pair  $(X, W)$  we have the following transitions.

(a) For every  $X$  such that  $\sigma \bowtie X$  there is transition

$$((q_{\sigma,l,1}, \varepsilon, (X, W)), (P.D.\sigma, W));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,1}$  to  $P.D.\sigma$  without editing the stack whenever  $\sigma \bowtie X$ .

(b) For every  $X$  such that  $\sigma$  is not  $\bowtie$  related  $X$  there is a transition

$$((q_{\sigma,l,1}, \varepsilon, (X, W)), (q_{\sigma,l,2}, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,1}$  to  $q_{\sigma,l,2}$  deleting  $W$  whenever  $\sigma$  is not  $\bowtie$  related to  $X$ .

(T3) For every  $\sigma \in \Sigma$ , from  $q_{\sigma,l,2}$  we have the following transitions. For every pair  $(X, Y)$  we have the following transitions.

(a) For every  $X$  such that  $\sigma \bowtie X$  there is a transition

$$((q_{\sigma,l,2}, \varepsilon, (X, Y)), (q_X, Y));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,2}$  to  $q_X$  without editing the stack, whenever  $\sigma \bowtie Y$ .

(b) For every  $X$  such that  $\sigma$  is not  $\bowtie$  related to  $X$  there is a transition

$$((q_{\sigma,l,2}, \varepsilon, (X, Y)), (q_{\sigma,l,1}, WMY));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,2}$  to  $q_{\sigma,l,1}$  that adds  $WM$  onto the pushdown stack whenever  $\sigma$  is not  $\bowtie$  related to  $X$ .

(T4) For every  $\sigma \in \Sigma$ , from  $P.D.\sigma$  we have the following transitions. For all pairs  $(X, W)$  such that  $\sigma \bowtie X$  there is a transition

$$((P.D.\sigma, \varepsilon, (X, W)), (C_p, \sigma));$$

that is, there is an  $\varepsilon$ -transition from  $P.D.\sigma$  to  $C_p$  that deletes  $W$  and writes  $\sigma$  instead whenever  $\sigma \bowtie X$ .

(T5) From  $C_p$  we have the following transitions. For all pairs  $(X, \sigma)$  where  $\sigma \bowtie X$  we have the following transitions.

- (a) For every  $X$  such that  $\sigma \bowtie' X$  there is a transition

$$((C_p, \varepsilon, (X, \sigma)), (A_\sigma, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $C_p$  to  $A_\sigma$  deleting  $\sigma$  whenever  $\sigma \bowtie' X$ .

- (b) For every  $X$  such that  $\sigma$  is not  $\bowtie'$  related to  $X$  there is a transition

$$((C_p, \varepsilon, (X, \sigma)), (C_0, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $C_p$  to  $C_0$  deleting  $\sigma$  whenever  $\sigma$  is not  $\bowtie'$  related to  $X$ .

- (T6) For every  $\sigma \in \Sigma$ , from  $A_\sigma$  we have the following transitions. For every  $X \in \Delta$  we have the following transitions.

- (a) There is a transition

$$((A_\sigma, \varepsilon, (X, \sigma^{-1})), (C_0, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $A_\sigma$  to  $C_0$  that deletes  $\sigma^{-1}$  if  $\sigma^{-1}$  is at the top of the pushdown stack.

- (b) For every  $Y \in \Gamma \setminus \{\sigma^{-1}\}$ , there is a transition

$$((A_\sigma, \varepsilon, (X, Y)), (C_0, \sigma Y));$$

that is, there is an  $\varepsilon$ -transition from  $A_\sigma$  to  $C_0$  that pushes  $\sigma$  onto the stack (i.e. the automaton writes  $\sigma Y$ ) whenever  $Y \neq \sigma^{-1}$ .

- (T7) From  $C_0$  we have the following transitions. For every  $X \in \Delta$  we have the following transitions.

- (a) There is a transition

$$((C_0, \varepsilon, (X, M)), (C_0, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $C_0$  to  $C_0$  that deletes  $M$  if at  $M$  is at the top of the pushdown stack.

- (b) For every  $Y \in (\Gamma \setminus \{W, M\}) \cup \{\perp\}$  there is a transition

$$((C_0, \varepsilon, (X, Y)), (q_r, Y));$$

that is, there is an  $\varepsilon$ -transition from  $C_0$  to  $q_r$  that does not edit the stack, whenever  $Y \notin \{W, M\}$ .

(T8) From  $C_1$  we have the following transitions.

(a) For every  $(X, Y) \in \Delta \times \Gamma$  there is a transition

$$((C_1, \varepsilon, (X, Y)), (q_A, Y));$$

that is, there is an  $\varepsilon$ -transition from  $C_1$  to  $q_A$  whenever the pushdown stack.

(b) There is a transition

$$((C_1, \varepsilon, (\perp, \perp)), (q_X, \perp));$$

that is, there is an  $\varepsilon$ -transition from  $C_1$  to  $q_X$  whenever the pushdown stack is empty.

Before we prove Theorem [4.6.1](#) we shall give an informal description of the states and the processes they achieve.

**Description of States:** As mentioned in the intuitive description of the machine, the automaton works in multiple stages. Here we shall specify what states make up these stages.

**Reading Stage:** The state  $q_r$  is responsible for this stage. Upon reading an input letter  $\sigma$  from  $q_r$  the automaton moves to state  $q_\sigma$ .

**Locating the correct corresponding check-stack letter stage:** The states  $q_{\sigma,l,1}$  and  $q_{\sigma,l,2}$  are responsible for this stage. The automaton moves from  $q_\sigma$  to  $q_{\sigma,l,1}$  while adding a  $W$  onto the stack. The automaton then uses the states  $q_{\sigma,l,1}$  and  $q_{\sigma,l,2}$  in the way we described in the intuitive description until the correct letter has been found or the string is immediately rejected. Then the automaton moves to the following stage by moving to the state  $P.D._\sigma$ .

**Pushing input letter onto the pushdown stack stage:** The state  $P.D._\sigma$  is responsible for this stage (for every  $\sigma \in \Sigma$ ). It deletes  $W$  and writes  $\sigma$ . The automaton then moves to the next stage by moving to the state  $C_p$ .

**Pre-check stage:** The state  $C_p$  is responsible for this stage. At the top of the pushdown stack there is an input letter, say  $\sigma$ , with the corresponding check-stack letter being  $Y$  where  $\sigma \bowtie Y$ .

If  $\sigma \bowtie' Y$  then it deletes  $\sigma$  and moves to the action stage.

Otherwise, it deletes  $\sigma$  and moves to the clean up stage.

**Action stage:** The state  $A_\sigma$  is responsible for this stage (for every  $\sigma \in \Sigma$ ). If the top of the pushdown stack is not  $\sigma^{-1}$  then we simply write  $\sigma$  onto the pushdown stack and move to the clean up of the stack stage. Otherwise, we delete  $\sigma^{-1}$  from the top of the stack, and move to the clean up of the stack stage.

**Clean up of the stack stage:** The state  $C_0$  is responsible for this stage. If the top of the pushdown stack is not  $M$  then we move back to the reading stage. Otherwise, we delete  $M$  until the top of the pushdown stack is not  $M$ .

**Check stage:** The state responsible for this is  $C_1$ . The automaton moves from  $q_r$  to  $C_1$ . From  $C_1$  there is a transition to  $q_A$  if the pushdown stack is not empty.

## 4.6.2 Proof of Theorem [4.6.1](#)

In this section, we shall prove the theorem. We do so by first explaining the process by which the automaton reads a string. It will be useful to the reader to revisit the description of the states and the intuitive description, as we shall rely on the stages we described above.

**Explanation of how a string is read:** Let  $\omega = u_1v_1u_2v_2 \cdots u_l v_l$  be a string in the generators of  $\mathbb{Z}^n * \mathbb{Z}^m$ , where  $u_i \in X_n^*$  and  $v_i \in Y_m^*$ . We shall describe how  $\mathcal{A}$  reads  $\omega$  and then we shall follow that with the proof that  $\mathcal{L}(\mathcal{A}) = \text{coWP}(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup Y_m)$ .

Recall Definition [2.1.14](#). The strings  $u_i$  and  $v_i$ , for all  $i$ , are syllables.

The automaton reads the syllable  $u_i = x_{i,1}x_{i,2} \cdots x_{i,r}$ . If  $u_i \neq \varepsilon$  then the automaton proceeds to read it in the way we described in the intuitive description; i.e., it starts by reading  $x_{i,1}$ , and then it locates the correct corresponding check-stack letter. It does so by pushing a  $W$  onto the pushdown stack via (T1). Now the automaton uses the transitions in (T2) and (T3) to find the first check-stack letter  $X_i$  such that  $x_{i,1} \bowtie X_i$  where  $X_i \in X_n^c$ . That is,  $X_i = A_j$  for some  $j \in \{1, 2, \dots, n\}$ . Now the automaton uses (T4) to replace  $W$  by the letter  $x_{i,1}$  so that  $X_i$  is the corresponding check-stack letter. If  $x_{i,1}$  is not  $\bowtie'$  related to  $X_i$  then the automaton deletes  $x_{i,1}$  and removes the letter  $M$  from the top of the stack until the top of the stack is no longer  $M$  (via a sequence of transitions in (T5)(b) and (T7)(a)). Otherwise, the automaton uses (T5)(a) and (T6). These transitions do the following. The automaton deletes  $x_{i,1}$ , and if the new top of the pushdown stack is  $x_{i,1}^{-1}$  then it deletes it. Otherwise it writes  $x_{i,1}$ . That is, let  $u$  be the maximal prefix of the pushdown stack (where the top of the pushdown stack is the beginning of the string) such that  $u \in X_n^*$  and let  $y$  be the letter after  $u$  in the pushdown stack such that  $y \in Y_m$ . Then after using (T5)(a) and (T6) the prefix  $u$  is replaced by  $\gamma_{\{a_j, a_j^{-1}\}}(ux_{i,1})$  on the pushdown stack. This is because the transition (T6)(a) freely reduces the string when inverses are on top of each other on the pushdown stack while (T6)(b) writes a generator  $\sigma$  on top of the pushdown stack if the top of the stack is not  $\sigma^{-1}$ . Further, if the top of the pushdown stack before a letter  $\sigma$  is read is either  $\sigma$  or  $\sigma^{-1}$  then by the stage that locates the correct check-stack letter it must be that either there is enough letters on that part of the



check-stack to allow processing of  $\sigma$  or the string is rejected in that run.

We shall assume that there is enough subsequent letters after  $X_i$  on the check-stack that are also equal to  $X_i$  so that the entire syllable of  $u_i$  gets processed. Otherwise by (T3)(a) the string gets rejected (since we move to state  $q_X$  and there are no transitions from  $q_X$ ). Thus we may assume that syllables get processed by in the same way as there is enough of the same check-stack letter on top of each other. Therefore after reading  $u_i$ , the maximal prefix on the pushdown stack consisting of letters in  $X_n$  will be  $\gamma_{\{a_j, a_j^{-1}\}}(uu_i)$ .

Now suppose that  $v_i =_{\mathbb{Z}^m} 1_{\mathbb{Z}^m}$ . By the above we may assume that there is a long enough substring of the check-stack consisting of the letter  $Y_i \in \{B_1, B_2, \dots, B_m\}$  after  $X_i$  for the syllable  $v_i$  to be read. Let  $Y_i = B_k$  for some  $k$ . Since  $v_i =_{\mathbb{Z}^m} 1_{\mathbb{Z}^m}$  then it is trivial in every coordinate. Therefore for all  $p \in \{1, 2, \dots, m\}$ ,  $\gamma_{\{b_p, b_p^{-1}\}}(v_i) = \varepsilon$ . In particular

$$\gamma_{\{b_k, b_k^{-1}\}}(v_i) = \varepsilon.$$

Therefore after reading  $v_i$  the pushdown stack will be the same as it was after reading  $u_i$ . Now the automaton is ready to read  $u_{i+1}$ . We note that if  $v_i =_{\mathbb{Z}^m} 1_{\mathbb{Z}^m}$  then  $u_i$  and  $u_{i+1}$  are in the same part. We observe that not only we require the number of letters of the string consisting of  $X_i$  (located previously) that are on top of each other on the check-stack to be enough so that syllable  $u_i$  can be read, we also require that the number is large enough so that every syllable in the same part can be read consecutively. Otherwise, (T3)(a) would result in the string being rejected. Reading  $u_{i+1}$  will be similar to reading  $u_i$  described above and at the end of reading  $u_{i+1}$ , the maximal prefix of the pushdown stack consisting of letters from  $X_n$  will be  $\gamma_{\{a_j, a_j^{-1}\}}(uu_iu_{i+1})$ . We are now ready to read another syllable, and the process will repeat.

There are no other choices that the automaton can make at any point during its processing. Thus the way it reads a string is unique given a choice of check-stack.

We shall now prove that given a string is read in the above way the automaton accepts exactly the co-word problem of  $\mathbb{Z}^n * \mathbb{Z}^m$ .

*Proof of Theorem 4.6.1.* Let  $\omega \in WP(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup Y_m)$ . Suppose

$$\omega = u_1v_1u_2v_2 \cdots u_tv_t$$

where  $u_i \in X_n^*$  and  $v_i \in Y_m^*$ . Since  $\omega \in WP(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup Y_m)$  either every syllable is equal to the identity in its respective group, or not all syllables are equal to the identity in their respective groups.

If every syllable is equal to the identity then

$$\gamma_{\{a_r, a_r^{-1}\}}(u_i) = \varepsilon$$

for every  $r \in \{1, 2, \dots, n\}$  and every  $i \in \{1, 2, \dots, t\}$ . Similarly,

$$\gamma_{\{b_s, b_s^{-1}\}}(v_j) = \varepsilon$$

for every  $s \in \{1, 2, \dots, m\}$  and every  $j \in \{1, 2, \dots, t\}$ .

Therefore, for any long enough choice of check-stack where there is enough letters to process every syllable the automaton will process the syllables as explained earlier. Assuming there are enough check-stack letters to process every syllable, upon reading every syllable the pushdown stack will be as follows. Either

- $\gamma_X(u) \perp$  where  $u$  can be replaced by every syllable  $u_i$ , and  $X = \{a_{i_1}, a_{i_1}^{-1}\}$  where  $i_1$  is determined by the index of the corresponding check-stack letter  $A_{i_1}$ . We note that  $A_{i_1}$  will be the first check-stack letter to appear on the check-stack. Therefore  $\gamma_X(u) = \varepsilon$  and thus the pushdown stack is empty. Alternatively, the pushdown stack will be
- $\gamma_Y(v) M^e \perp$  for some  $e > 0$  where  $v$  can be replaced by every syllable  $v_i$ , and  $Y = \{b_{j_1}, a_{j_1}^{-1}\}$  where  $j_1$  is determined by the index of the corresponding check-stack letter  $A_{j_1}$ . We note that  $A_{j_1}$  will be the first check-stack letter that is not  $A_{i_1}$ . Therefore  $\gamma_X(u) = \varepsilon$  and thus the stack will only then consist of  $M^e$ . However, the transitions in (T7)(a) will delete all occurrences of  $M$  and then stack will be empty.

Since the pushdown stack will be empty at the end of processing  $\omega$ , it will not be accepted since the transitions (T8)(a) are the only transitions to the accept state, and they can only be used if the stack is not empty.

Now we suppose that not all syllables are trivial. Suppose  $\omega = w_1 w_2 \cdots w_r$  where  $w_i$  is a syllable for every  $1 \leq i \leq r$ . Let  $i_1 \in \{1, \dots, r\}$  be the smallest index such that  $w_{i_1} \neq 1$  (where 1 is the identity of the group which the letters of  $w_{i_1}$  belong to). Then there exists a next syllable  $w_j \neq 1$ . (Otherwise  $w \in \text{coWP}(\mathbb{Z}^n * \mathbb{Z}^m, \Sigma)$ .) As every  $w_k$  is equal to the identity of its respective group for where  $i < k < j$ , we see that  $w_i$  and  $w_j$  belong to the same part and will be processed by the automaton as if they are one syllable. We note that otherwise the appropriate part of the check-stack is not long enough and the string gets rejected. That is, there exists a long enough appropriate substring of the check stack  $u_1^l$  where  $u_1$  is  $A_p$  or  $B_q$  (for some index  $p$  or  $q$ ) such that every letter of both  $w_i$  and  $w_j$  will be  $\bowtie$  related to  $u_1$ ) or the string will be rejected. If  $w_i w_j$  will be processed in the same way then either  $w_i w_j$  is equal to the identity or not. If  $w_i w_j$  is equal to the identity then we are done, since regardless of what  $u_1$  is, the pushdown stack will not contain any string corresponding to the part of  $u_1^l$  that was being used. Otherwise there must exist another syllable  $w_q$  that will also be in the same part. The automaton thus repeatedly finds these syllables, processing

them from the same part of the check-stack until the pushdown stack is empty and thus the string is rejected. Otherwise the syllables cannot be processed in the same way due to  $l$  being too small and thus the automaton will also reject the string in that instance. Therefore any string in  $WP(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup Y_m)$  never gets accepted.

Now we suppose that  $\omega' \in coWP(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup Y_m)$ . Suppose  $\omega' = w'_1 w'_2 \cdots w'_d$  where each  $w'_i$  is a syllable.

First suppose that every syllable is not equal to the identity in its respective group. As  $w_i$  is not equal to the identity for every  $i$  then there exists a generator  $\sigma_i$  for every  $w_i$  such that the sum of the exponents of  $\sigma_i$  is non-zero in  $w_i$ . If  $w_i \in X_n^*$  then  $\sigma_i \in \{a_1, a_2, \dots, a_n\}$ , and if  $w_i \in Y_m^*$  then  $\sigma_i \in \{b_1, b_2, \dots, b_m\}$ . By repeatedly picking these generators for each syllable we induce a sequence  $A'_1 B'_1 \dots A'_d B'_d$  where  $A'_1, \dots, A'_d \in \{A_1, \dots, A_n\}$  and  $B'_1, \dots, B'_d \in \{B_1, \dots, B_m\}$ . There exist lengths  $l_1, l_2, \dots, l_{2d}$  such that  $A_1^{l_1} B_1^{l_2} \dots B_k^{l_{2k}}$  is a long enough sequence on the check-stack such that  $w_1 w_2 \cdots w_d$  can be processed with the automaton.

Let  $\overline{w'_{i\sigma_i}}$  be the projection of the string  $w_i$  onto the free subgroup generated by  $\sigma_i$ . Further let  $\rho(\overline{w'_{i\sigma_i}})$  be the reverse of the free reduction of  $\overline{w'_{i\sigma_i}}$ .

By the construction of the automaton and by our choice of check-stack, the pushdown stack will consist of the product of all  $\rho(\overline{w'_{i\sigma_i}})$  (with strings consisting of the letter  $M$  in between them). Since none of these images are trivial, the stack is non-empty and the string is accepted.

Now we may suppose that there is a syllable that is equal to the identity of its respective group. Let  $\tilde{\omega}$  be the reduced form of  $\omega$ . Note that  $\omega$  is related to  $\tilde{\omega}$  by a sequence of reductions of substrings of  $\omega$  that are trivial. Further there exists a string  $r \in \mathcal{R}$  such the automaton accepts  $\tilde{\omega}$  with  $r$  on the check stack by the above. Thus there exists a string  $r' \in \mathcal{R}$  such that  $r'$  consists of the same choices of letters of as  $r$ , but  $r'$  is just longer to accommodate for the processing of  $\omega$ . For example if  $r = A_1 A_1 B_2 B_2 A_4 A_4 B_1$  then the indices used for  $r'$  do not change, but we simply have a longer substring for each letter such as  $A_1 A_1 A_1 A_1 B_2 B_2 B_2 A_4 A_4 A_4 A_4 B_1 B_1 B_1 B_1$  would be a valid choice for  $r'$ . (The string may require more letters to be appended afterwards but there is a prefix of  $r'$  where the indices used in the letters are determined by  $r$ .)

The automaton can process  $w$  and  $\tilde{w}$  with  $r'$  on the check-stack and accepts  $\tilde{w}$ . Since the reductions of substrings that are equal to the identity result in returning the stack to how it was before, regardless of the generator considered for the reduction we know that the pushdown stack upon reading  $w$  will be non-empty with  $r'$  on the check stack. Therefore  $w$  is accepted by the automaton. ■

### 4.6.3 Examples

In this section, we go through how the automaton processes two strings with different check-stacks. We focus on  $\mathbb{Z}^2 * \mathbb{Z}$ . Let  $\mathbb{Z}^2$  be generated by  $\{a_1, a_2\}$  and  $\mathbb{Z}$  be generated by  $\{b_1\}$ . Thus the check-stack letters are  $\{A_1, A_2, B_1\}$ . The two strings we consider are

1.  $\omega_1 = a_2^{-1}a_1a_2b_1a_1a_1^{-1}b_1^{-1}$ , and
2.  $\omega_2 = a_1^{-1}a_2^{-1}b_1^{-1}b_1a_2a_1$ .

We observe that  $\omega_1$  is non-trivial (since  $\omega_1 =_{\mathbb{Z}^2 * \mathbb{Z}} a_1$ ), while  $\omega_2$  is trivial.

In our descriptions below, we shall not refer to the specific states but will refer to the stages instead. For  $\omega_1$ , we shall show how the string is processed three times, with three different strings on the check-stack in order to exhibit different behaviours of the automaton.

1. Suppose the string on the check-stack is  $A_2A_2B_1B_1A_2B_1$ .

First the automaton goes into the reading stage, reading  $a_2^{-1}$ . Then it moves into locating the correct corresponding check-stack letter stage. Here,  $W$  gets written onto the stack. So the pushdown stack becomes  $W \perp$ . The letter now corresponding to  $W$  on the check-stack is  $A_2$ , and  $a_2^{-1} \bowtie A_2$ . Thus the automaton enters the Pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_2^{-1}$  gets written instead. Now the automaton moves to the pre-check stage. As  $a_2^{-1} \not\bowtie' A_2$ , the automaton deletes the top of the pushdown stack and moves to the action stage. Since the pushdown stack is now empty, the action stage writes  $a_2^{-1}$  onto the pushdown stack. So the pushdown stack becomes  $a_2^{-1} \perp$ . Now the automaton moves to the clean up of the stack stage. Since, the top of the pushdown stack is not  $M$ , the automaton moves back to the reading stage.

After reading the first letter, the pushdown stack is  $a_2^{-1} \perp$ . In this reading stage the automaton reads  $a_1$  and moves to the locating the correct corresponding check-stack letter stage. Here,  $W$  gets written onto the stack. So the pushdown stack is  $Wa_2^{-1} \perp$ . The letter now corresponding to  $W$  is  $A_2$ . Since  $a_1 \bowtie A_2$ , the automaton enters the pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_1$  gets written instead. The pushdown stack is now  $a_1a_2^{-1} \perp$ . The automaton now moves to the pre-check stage. Since  $a_1$  is not  $\bowtie'$  related to  $A_2$ , the automaton deletes  $a_1$  and moves to the clean up of the stack stage. The pushdown stack is now  $a_2^{-1} \perp$ , and since the top of the pushdown stack is not  $M$  the automaton moves back to the reading stage.

Now the automaton reads  $a_2$ . The automaton moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the pushdown stack. So the pushdown stack is  $Wa_2^{-1}\perp$ . The letter now corresponding to  $W$  is  $A_2$ . Since  $a_2 \bowtie A_2$ , the automaton enters the pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_2$  gets written instead. The pushdown stack is now  $a_2a_2^{-1}\perp$ . The automaton now moves to the pre-check stage. Since  $a_2 \bowtie' A_2$ , the automaton moves to the action stage and deletes  $a_2$ . So the top of the pushdown stack is  $a_2^{-1}$ . Therefore the action stage, deletes the top of the pushdown stack. So the pushdown stack is  $\perp$ . Now the automaton moves to the clean up of the stack stage. Since the top of the stack is not  $M$ , the automaton moves back to the reading stage.

Now the automaton reads  $b_1$  and moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the stack. So the pushdown stack is  $W\perp$ . The letter now corresponding to  $W$  is  $A_2$ . Since  $b_1$  is not  $\bowtie$  related to  $A_2$  the automaton deletes  $W$  and  $WM$  instead. Now the pushdown stack is  $WM\perp$ , and the corresponding check-stack letter to  $W$  is  $A_2$  again. Since  $b_1$  is not  $\bowtie$  related to  $A_1$ , the automaton again deletes  $W$  and writes  $WM$  instead. So the pushdown stack is  $WMM\perp$ , with the corresponding check-stack letter being  $B_1$ . Since  $b_1 \bowtie B_1$  the automaton enters the pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $b_1$  is written instead, so the pushdown stack is  $b_1MM\perp$ . The automaton now moves to the pre-check stage. Since  $b_1 \bowtie' B_1$  the automaton deletes  $b_1$  and moves to the action stage. The pushdown stack is now  $MM\perp$ , since the top of the pushdown stack is not  $b_1^{-1}$ , the automaton writes  $b_1$  and moves to the clean up of the stack stage. The pushdown stack is now  $b_1MM\perp$ . Since the top of the pushdown stack is not  $M$  the automaton moves back to the reading stage.

Now the automaton reads  $a_1$  and moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the stack. So the pushdown stack is  $Wb_1MM\perp$  with the letter on the check-stack corresponding to  $W$  being  $B_1$ . Since  $a_1$  is not  $\bowtie$  related to  $B_1$ , the automaton deletes  $W$  and writes  $WM$  thus the pushdown stack is  $Wb_1MM\perp$ . Now the corresponding check-stack letter is  $A_2$ . Since  $a_1 \bowtie A_2$  the automaton moves to the pushing input letter onto pushdown stack stage. Here  $W$  gets deleted and  $a_1$  gets written instead, and the automaton moves to the pre-check stage. The pushdown stack is now  $a_1Mb_1MM\perp$ . Since  $a_1$  is not  $\bowtie$  related to  $A_2$ , the automaton deletes the top of the stack and moves to the clean up of the stack stage. The top of the pushdown stack is now  $M$  and so the automaton deletes it, so the pushdown stack is  $b_1MM\perp$ . Since the top of the pushdown stack is no longer  $M$ , the automaton moves back to the reading stage.

Now the automaton reads  $a_1^{-1}$  with the pushdown stack being  $b_1MM\perp$ , and it moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the stack. So the pushdown stack is  $Wb_1MM\perp$ , with the corresponding check-stack letter being  $B_1$ . Since  $a_1$  is not  $\bowtie$  related to  $B_1$ , the automaton deletes  $W$  and writes  $WM$  instead. So the pushdown stack is now  $Wb_1MM\perp$ . The corresponding check-stack letter to  $W$  is now  $A_2$ . Since  $a_1^{-1} \bowtie A_2$ , the automaton moves to pushing input letter onto pushdown stack stage. The automaton deletes  $W$  and writes  $a_1^{-1}$  instead, and moves to the pre-check stage. The pushdown stack is  $a_1^{-1}Mb_1MM\perp$ , with the corresponding check-stack letter being  $A_2$ . Since  $a_1^{-1}$  is not  $\bowtie$  related to  $A_2$  the automaton deletes  $a_1^{-1}$  and moves to the clean up of the stack stage. The pushdown stack is  $Mb_1MM\perp$ . The top of the stack gets deleted, so the pushdown stack is  $b_1MM\perp$ . Since the top of the pushdown stack is no longer  $M$ , the automaton moves back to the reading stage.

Now the automaton reads  $b_1^{-1}$  with the pushdown stack being  $b_1MM\perp$ , and it moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the stack. So the pushdown stack is  $Wb_1MM\perp$ . Now the corresponding check-stack letter is  $B_1$ . Since  $b_1^{-1} \bowtie B_1$  the automaton moves to pushing input letter onto pushdown stack stage. The automaton deletes  $W$  and writes  $b_1^{-1}$  instead, and moves to the pre-check stage. Since  $b_1^{-1} \not\bowtie B_1$ , the automaton deletes the top of the stack and moves to the action stage. The pushdown stack is now  $b_1MM\perp$ . Now since the top of the stack is  $b_1$  the automaton deletes the top of the stack and moves to the clean up of the stack stage. The pushdown stack is now  $MM\perp$ . Since the top of the pushdown stack is  $M$  the automaton deletes it, so the pushdown stack is now  $M\perp$ . Again, the top of the pushdown stack is  $M$  so the automaton deletes it. Now the stack is  $\perp$ , and the automaton returns to the reading stage.

Now, since the automaton has read all the input letters it moves to the check stage. Since the stack is empty, the automaton does not accept the input string.

2. Suppose the string on the check-stack is  $A_1B_1B_1A_2A_2B_1$ .

First the automaton goes into the reading stage, where it reads  $a_2^{-1}$ . The automaton moves to locating the correct corresponding check-stack letter stage where it writes  $W$  onto the pushdown stack. So the pushdown stack becomes  $W\perp$ , with the corresponding check-stack letter being  $A_1$ . Since  $a_2^{-1} \bowtie A_1$ , the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_2^{-1}$  gets written instead. The automaton then moves to the pre-check stage. Since  $a_2^{-1}$  is not  $\bowtie$  related to

$A_1$ , the automaton deletes it and moves to the clean up of the stack stage. The pushdown stack is empty, hence the top of the stack is not  $M$  and the automaton moves back to the reading stage.

The automaton now reads  $a_1$ . The automaton moves to locating the correct check-stack letter stage where it writes  $W$  onto the pushdown stack. So the pushdown stack is  $W\perp$ , with the corresponding check-stack letter is  $A_1$ . Since  $a_1 \bowtie A_1$  the automaton moves to pushing input letter onto the pushdown stack stage. Here the automaton deletes  $W$  and writes  $a_1$ , and moves to the pre-check stage. Since  $a_1 \bowtie' A_1$ , the automaton deletes  $a_1$  from the top of the stack and moves to the action stage. Since the top of the pushdown stack is now empty, the automaton writes  $a_1$  onto the pushdown stack and moves to the clean up of the stack stage. The pushdown stack is  $a_1\perp$ , since the top of the pushdown stack is not  $M$  the automaton moves back to the reading stage.

The automaton now reads  $a_2$  and moves to locating the correct corresponding check-stack letter stage where it writes  $W$  onto the stack. The pushdown stack is now  $Wa_1\perp$ , where the corresponding check-stack letter is  $B_1$ . However, the letter below  $B_1$  is  $A_1$ , and  $a_2 \bowtie A_1$ . Thus the automaton moves to  $q_X$  rejecting the string.

3. Suppose the string on the check-stack is  $A_1A_1B_1B_1A_2B_1$ .

First the automaton goes into the reading stage, where it reads  $a_2^{-1}$ . The automaton then moves to locating the correct corresponding check-stack letter stage. Here it writes a  $W$  onto the pushdown stack. The pushdown stack is now  $W\perp$ , with the corresponding check-stack letter being  $A_1$ . Since  $a_2^{-1} \bowtie A_1$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_2^{-1}$  gets written on the stack instead. Now the automaton moves to the pre-check stage. Since  $a_2^{-1}$  is not  $\bowtie'$  related to  $A_1$ , the automaton deletes it and moves to the clean up of the stack stage. The pushdown stack is empty, in particular the top of the pushdown stack is not  $M$  and thus the automaton moves back to the reading stage.

Now the automaton reads  $a_1$  and moves to locating the correct corresponding check-stack letter stage. Here,  $W$  gets written onto the pushdown stack. The pushdown stack is  $W\perp$  with the corresponding check-stack letter being  $A_1$ . Since  $a_1 \bowtie A_1$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_1$  gets written instead. So the pushdown stack is  $a_1\perp$ . Now the automaton moves to the pre-check stage. Since  $a_1 \bowtie' A_1$  the automaton deletes  $a_1$  from the top of the stack and moves to the action stage. Since the top of the stack is not  $a_1^{-1}$  the automaton writes  $a_1$  and moves to the clean up of the stack stage. The top

of the pushdown stack is not  $M$ , and thus the automaton moves back to the reading stage.

Now the automaton reads  $a_2$ , and moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the pushdown stack. The pushdown stack is  $Wa_1\perp$ , with the corresponding check-stack letter being  $A_1$ . Since  $a_2 \bowtie A_1$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_2$  gets written instead. So the pushdown stack is  $a_2a_1\perp$ . Now the automaton moves to the pre-check stage. Since  $a_2$  is not  $\bowtie$  related to  $A_1$  the automaton deletes the top of the stack and moves to the clean up of the stack stage. The top of the pushdown stack is not  $M$  and thus the automaton moves back to the reading stage.

The automaton now reads  $b_1$ , and moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the pushdown stack. The pushdown stack is now  $Wa_1\perp$  with the corresponding check-stack letter being  $A_1$ . Since  $b_1$  is not  $\bowtie$  related to  $A_1$  the automaton deletes  $W$  and writes  $WM$  instead. So the pushdown stack is  $WMa_1\perp$  with the corresponding check-stack letter being  $B_1$ . Since  $b_1 \bowtie B_1$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $b_1$  gets written instead. So the pushdown stack is  $b_1Ma_1\perp$ . Now the automaton moves to the pre-check stage. Since  $b_1 \bowtie' B_1$  the automaton deletes  $b_1$  and moves to the action stage. The top of the pushdown stack is not  $b_1^{-1}$  and thus the automaton writes  $b_1$  onto the pushdown stack. So the pushdown stack is  $b_1Ma_1\perp$ . Then the automaton moves to the clean up of the stack stage. Since the top of the pushdown stack is not  $M$  the automaton moves back to the reading stage.

Now the automaton reads  $a_1$ , and moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the pushdown stack. The pushdown stack is now  $Wb_1Ma_1\perp$ , with the corresponding check-stack letter being  $B_1$ . Since  $a_1$  is not  $\bowtie$  related to  $B_1$ , the automaton deletes  $W$  and writes  $WM$  instead. So the pushdown stack is  $WMb_1Ma_1\perp$ , with the corresponding check-stack letter being  $A_2$ . Since  $a_1 \bowtie A_2$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_1$  gets written instead. So the pushdown stack is  $a_1Mb_1Ma_1\perp$ . Now the automaton moves to the pre-check stage. Since  $a_1$  is not  $\bowtie'$  related to  $A_2$  the automaton deletes  $a_1$  from the top of the pushdown stack and the automaton moves to the clean up of the stack stage. The pushdown stack is  $Mb_1Ma_1\perp$ . The automaton deletes the top of the pushdown stack, and so the pushdown stack is  $b_1Ma_1\perp$ . Now the automaton moves back to the reading stage.



Now the automaton reads  $a_1^{-1}$ , and moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the pushdown stack. The pushdown stack is now  $Wb_1Ma_1\perp$ , with the corresponding check-stack letter being  $B_1$ . Since  $a_1^{-1}$  is not  $\bowtie$  related to  $B_1$ , the automaton deletes  $W$  and writes  $WM$  instead. So the pushdown stack is  $Wb_1Ma_1\perp$ , with the corresponding check-stack letter being  $A_2$ . Since  $a_1^{-1} \bowtie A_2$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_1^{-1}$  gets written instead. So the pushdown stack is  $a_1^{-1}Mb_1Ma_1\perp$ . Now the automaton moves to the pre-check stage. Since  $a_1^{-1}$  is not  $\bowtie'$  related to  $A_2$  the automaton deletes  $a_1^{-1}$  from the top of the pushdown stack and the automaton moves to the clean up of the stack stage. The pushdown stack is  $Mb_1Ma_1\perp$ . The automaton deletes the top of the pushdown stack, and so the pushdown stack is  $b_1Ma_1\perp$ . Now the automaton moves back to the reading stage.

Now the automaton reads  $b_1^{-1}$ , and the moves to locating the correct corresponding check-stack letter stage. Here  $W$  gets written onto the pushdown stack. The pushdown stack is now  $Wb_1Ma_1\perp$ , with the corresponding check-stack letter being  $B_1$ . Since  $b_1^{-1} \bowtie B_1$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $b_1^{-1}$  gets written instead. So the pushdown stack is  $b_1^{-1}b_1Ma_1\perp$ , and the automaton moves to the pre-check stage. Since  $b_1^{-1} \bowtie' B_1$  the automaton deletes the top of the stack and moves to the action stage. Since the top of the stack is now  $b_1$ , the automaton deletes it and moves to the clean up of the stack stage. Now the pushdown stack is  $Ma_1\perp$ . The automaton deletes the top of the pushdown stack, so the pushdown stack becomes  $a_1\perp$ . Since the top of the pushdown stack is no longer  $M$ , the automaton moves back to the reading stage.

Now, since the automaton has read all the input letters it moves to the check stage. Since the pushdown stack is not empty, the automaton accepts the string.

We remark that in the first run of above we wanted to highlight that it is not necessary for string  $\omega \in coWP(\mathbb{Z}^n * \mathbb{Z}^m, X_n \cup X_m)$  to be accepted in every run. In the second run we wanted to highlight how a string can be rejected in the middle of processing. Finally, in the third run we gave an example of how a string is accepted.

We shall now consider  $\omega_2 = a_1^{-1}a_2^{-1}b_1^{-1}b_1a_2a_1$  when the string on the check stack is  $A_1A_1B_1B_1A_1B_1$ .

First the automaton goes into the reading stage, where it reads  $a_1^{-1}$ . The automaton the moves to locating the correct corresponding check-stack letter stage. Here it writes  $W$  onto the pushdown stack. The pushdown stack is now  $W\perp$ , with

the corresponding check-stack letter being  $A_1$ . Since  $a_1^{-1} \bowtie A_1$ , the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_1^{-1}$  gets written instead. So the pushdown stack is  $a_1^{-1}\perp$ . The automaton then moves to the pre-check stage. Since  $a_1^{-1} \bowtie' A_1$  the automaton deletes the top of the stack and moves to the action stage. Since the top of the stack is now  $\perp$ , the automaton writes  $a_1^{-1}$  onto the stack and moves to the clean up of the stack stage. Since the top of the stack is not  $M$  the automaton moves back to the reading stage.

The automaton now reads  $a_2^{-1}$  and moves to locating the correct corresponding check-stack letter stage. Here it writes  $W$  onto the pushdown stack. The pushdown stack is now  $Wa_1^{-1}\perp$ , with the corresponding check-stack letter being  $A_1$ . Since  $a_2^{-1} \bowtie A_1$ , the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_2^{-1}$  gets written instead. So the pushdown stack is  $a_2^{-1}a_1^{-1}\perp$ . The automaton then moves to the pre-check stage. Since  $a_2^{-1}$  is not  $\bowtie'$  related to  $A_1$  the automaton deletes the top of the pushdown stack and moves to the clean up of the stack stage. Since the top of the stack is not  $M$  the automaton moves back to the reading stage.

The automaton now reads  $b_1^{-1}$  and moves to locating the correct corresponding check-stack letter stage. Here it writes  $W$  onto the pushdown stack. The pushdown stack is now  $Wa_1^{-1}\perp$  with the corresponding check-stack letter being  $A_1$ . Since  $b_1^{-1}$  is not  $\bowtie$  related to  $A_1$ , the automaton deletes  $W$  and writes  $WM$  instead. Thus the pushdown stack is  $WMa_1^{-1}\perp$ . Now the corresponding check-stack letter is  $B_1$ . Since  $b_1^{-1} \bowtie B_1$  the automaton moves to pushing input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $b_1^{-1}$  gets written onto the stack instead. So the pushdown stack is  $b_1^{-1}Ma_1^{-1}\perp$ . Then the automaton moves to the pre-check stage. Since  $b_1^{-1} \bowtie' B_1$  the automaton deletes  $b_1^{-1}$  from the top of the stack and moves to the action stage. Since the top of the stack is not  $b_1$  the automaton writes  $b_1^{-1}$  and moves to the clean up of the stack stage. Now the pushdown stack is  $b_1^{-1}Ma_1^{-1}\perp$ . Since the top of the pushdown stack is not  $M$  the automaton moves back to the reading stage.

The automaton now reads  $b_1$  and moves to locating the correct corresponding check-stack letter stage. Here it writes  $W$  onto the pushdown stack. The pushdown stack is now  $Wb_1^{-1}Ma_1^{-1}\perp$  with the corresponding check-stack letter being  $B_1$ . Since  $b_1 \bowtie B_1$  the automaton moves to pushing the input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $b_1$  gets written onto the stack instead. So the pushdown stack is  $b_1b_1^{-1}Ma_1^{-1}\perp$ . Then the automaton moves to the pre-check stage. Since  $b_1 \bowtie' B_1$  the automaton deletes the top of the stack and move to the action stage. Since the top of the pushdown stack now is  $b_1^{-1}$  the automaton deletes the top of the pushdown stack and moves to the clean up of the stack stage. The pushdown stack is now  $Ma_1^{-1}\perp$ . Since the top of the pushdown stack is  $M$

the automaton deletes it. The pushdown stack is now  $a_1^{-1}\perp$ . Since the top of the pushdown stack is not  $M$ , the automaton returns to the reading stage.

The automaton now reads  $a_2$  and moves to locating the correct corresponding check-stack letter stage. Here it writes  $W$  onto the pushdown stack. The pushdown stack is now  $Wa_1^{-1}\perp$  with the corresponding check stack letter being  $A_1$ . Since  $a_2 \bowtie A_1$  the automaton moves to pushing the input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_2$  gets written onto the stack instead. So the pushdown stack is  $a_2a_1^{-1}\perp$ . The automaton then moves to the pre-check stage. Since  $a_2$  is not  $\bowtie'$  related to  $A_1$  the automaton deletes  $a_2$  from the top of the stack and moves to the clean up of the stack stage. The pushdown stack is  $a_1^{-1}\perp$ , thus the top of the pushdown stack is not  $M$  so the automaton moves back to the reading stage.

The automaton now reads  $a_1$  and moves to locating the correct corresponding check-stack letter stage. Here it writes  $W$  onto the pushdown stack. The pushdown stack is now  $Wa_1^{-1}\perp$  with the corresponding check-stack letter being  $A_1$ . Since  $a_1 \bowtie A_1$  the automaton moves to pushing the input letter onto the pushdown stack stage. Here  $W$  gets deleted and  $a_1$  gets written onto the stack instead. So the pushdown stack is  $a_1a_1^{-1}\perp$ . Now the automaton moves to the pre-check stage. Since  $a_1 \bowtie' A_1$  the automaton deletes the top of the stack. The pushdown stack is now  $a_1^{-1}\perp$ . Since the top of the pushdown stack is  $a_1^{-1}$  the automaton deletes and moves to the clean up of the stack stage. Since the stack is empty, the top of the pushdown stack is not  $M$ . Therefore the automaton moves back to the reading stage.

Now, since the automaton has read all the input letter it moves to the check stage. Since the pushdown stack is empty, the automaton does not accept the string.

## 4.7 The free product $\mathbb{Z}^n * G$ for a virtually free $G$

In this section we shall prove the following theorem.

**Theorem 4.7.1.** *Let  $G$  be a virtually free group. The free product  $\mathbb{Z}^n * G$  is co-ETOL.*

Similar to the previous section, we shall start by giving a high level intuitive summary of how and why the automaton works. Then we present the intuitive idea giving a detailed intuitive account of how the automaton works, and then we formally define the automaton accepting the co-word problem of  $\mathbb{Z}^n * G$ . We then give a proof that the automaton does indeed accept the co-word problem of  $\mathbb{Z}^n * G$ . Our construction is motivated by the construction used in proving Theorem 11 of [28].

### 4.7.1 Definition of Automaton

The automaton  $\mathcal{A}$  we define in this section works in a very similar way to the one accepting the co-word problem of  $\mathbb{Z}^n * \mathbb{Z}^m$ .

Let  $\mathbb{Z}^n$  be generated by the set  $\{a_1, a_2, \dots, a_n\}$ . Let  $X_n$  be the symmetric closure of  $\{a_1, a_2, \dots, a_n\}$ . Let  $G$  be a virtually free group. Let  $F_k$  be the free subgroup of finite index in  $G$ . As in [2.3.2](#), we shall take  $X_{F_k}$  to be the symmetric closure of the free generating set for  $F_k$ . Let  $T'$  be a right transversal for  $F_k$  in  $G$ , and we assume that  $1_G \in T'$ . Let  $T$  be the symmetric closure of  $T'$ . Observe that  $Y := X_{F_n} \cup T$  is a symmetrically closed finite generating set for  $G$ . For each  $y \in Y$  and  $t \in T$ ,  $ty \in G$  thus  $ty = f_{ty}t'$  for some  $f_{ty} \in F_k$  and  $t' \in T'$ . As in [2.3.2](#) we fix a string  $w_{ty} := x_1x_2 \cdots x_r \in X_{F_k}^*$  such that  $x_1x_2 \cdots x_r =_{F_k} f_{ty}$ .

Recall the pushdown automaton

$$P_G^{(W,1)} = (Q_G^{(W,1)}, I_G^{(W,1)}, \Sigma_G^{(W,1)}, \chi_G^{(W,1)}, \delta_G^{(W,1)}, \perp, F_G^{(W,1)})$$

defined in [2.3.2](#). We shall use a modified version of this automaton when we define the automaton  $\mathcal{A}$  accepting the co-word problem of  $\mathbb{Z}^n * G$ .

We define a set  $X_n^c := \{A_1, A_2, \dots, A_n\}$  and a set  $\{P\}$ .

For the purpose of the following intuitive summary, we describe how the automaton accepts a string. We first describe what a string on the check-stack looks like. Given a string  $\omega$  not representing the identity element of  $\mathbb{Z}^n * G$ , we describe how to select a string  $\nu$  in the regular language associated to the automaton, such that with  $\nu$  on the check-stack the automaton accepts  $\omega$ . Then we describe the contents of the pushdown stack at the end of processing.

**Intuitive Summary 4.7.2.** We call any element of  $X_n^c$  an  $A$ -type letter. The regular language associated to the check-stack will be a subset of  $(X_n^c \cup \{P\})^*$ . We shall now describe a process by which we can obtain a string on the check-stack that will be used to show that an input string representing a non-identity element of  $\mathbb{Z}^n * G$  is accepted.

Let  $\omega = \omega_1\omega_2 \cdots \omega_k$  be an input string with  $\omega_i$  being syllables of  $\omega$ . Let  $\bar{\omega} = \xi_1\xi_2 \cdots \xi_t$  be the normal form expression for the element of  $\mathbb{Z}^n * G$  representing  $\omega$ , with  $\xi_i$  being syllables of  $\bar{\omega}$ . Further suppose  $t > 0$  and thus  $\omega$  does not represent the identity element of  $\mathbb{Z}^n * G$ .

Since  $\bar{\omega}$  is in normal form then each  $\xi_i$  is non-trivial in its respective group. For each  $\xi_i \in X_n^*$  there exists a generator in  $X_n$  with non-zero exponent sum in  $\xi_i$ . We denote this generator by  $x_i$ . Observe that  $x_i$  is not defined for every  $1 \leq i \leq t$ . If  $x_i$  is not defined then we set  $x_i = P$ . The image of the string  $x_1x_2 \cdots x_t$  under the extension of the following map

$$\begin{aligned} a_j &\mapsto A_j \text{ for all } j \in \{1, 2, \dots, n\} \\ P &\mapsto P \end{aligned}$$

is a string over  $X_n^c \cup \{P\}$  such that if the  $j^{\text{th}}$  letter belongs to  $X_n^c$  then the  $j + 1^{\text{th}}$  letter is  $P$  (and vice versa).

Let  $\mu$  be the image of the string  $x_1^{p_1} x_2^{p_2} \dots x_t^{p_t}$  under the map above, for suitably large  $p_1, p_2, \dots, p_t$ . If first letter of  $\mu$  is  $P$  then append onto the beginning of  $\mu$  a prefix of the form  $A_l^{p_0}$  for some  $l \in \{1, 2, \dots, n\}$  and a suitably large  $p_0$  to form a string  $\mu'$ . Further if the last letter of  $\mu'$  is an  $A$ -type letter, append onto the end of  $\mu'$  a suffix of the form  $P^{p_{t+1}}$  for a suitably large  $p_{t+1}$  to form a string  $\mu''$ . Finally we form a final string  $\mu^{(3)}$  by appending a long enough suffix of the form

$$A_{i_1}^{s_1} P^{s'_1} A_{i_2}^{s_2} P^{s'_2} \dots A_{i_l}^{s_l} P^{s'_l},$$

for some  $i_1, i_2, \dots, i_l \in \{1, 2, \dots, n\}$  and suitably large  $l, s_1, s'_1, s_2, s'_2, \dots, s_l, s'_l$ . It is with  $\mu^{(3)}$  on the check-stack, that the automaton will recognise that  $\omega$  does not represent the identity element of  $\mathbb{Z}^n * G$ .

**Remark 4.7.3.** The core of the processing will be done using the substring  $\mu$  of the check-stack, and we pick exponents to be suitably large in order to ensure that the processing will be completed. That is; the automaton will not stop (and thus reject) due to an insufficiently long string on the check-stack. However,  $\mu$  is extended to  $\mu^{(3)}$  to ensure that the string is long enough to allow the processing of all syllables of  $\omega$  (regardless of whether they trivialise or not). Another reason of extending  $\mu$  to  $\mu^{(3)}$  is to ensure that the string is in our chosen regular language, which has strings of the form

$$A_{i_1}^{s_1} P^{s'_1} A_{i_2}^{s_2} P^{s'_2} \dots A_{i_l}^{s_l} P^{s'_l},$$

where  $i_1, i_2, \dots, i_l \in \{1, 2, \dots, n\}$  and  $l, s_1, s'_1, s_2, s'_2, \dots, s_l, s'_l > 0$ . ♠

The automaton will use the pushdown stack to store the product of the projections of each syllable  $\xi_i$  onto  $\langle x_i \rangle$  if  $\xi \in X_n^*$ , and what is written onto the stack of  $P_G^{(W,1)}$  upon reading  $\xi_i$  if  $\xi_i \in Y^*$ . It does so by first using bookkeeping states and padding symbols to search for the image of  $x_i$  under the map above. Then the automaton uses different bookkeeping states to freely reduce the projection of  $\xi_i$  onto  $(\{x_i\} \cup \{x_i^{-1}\})^*$  if  $\xi_i \in X_n^*$ . However, if  $\xi_i \in Y^*$  then the automaton simulates  $P_G^{(W,1)}$  by using bookkeeping states and recording the state of  $P_G^{(W,1)}$  reached at the end of reading  $\xi_i$  onto the stack. Since that state is always recorded, the automaton will always be able to continue processing from the same place if the next syllable of  $\omega$  it reads is trivial in its respective group. The automaton will accept when the stack is non-empty at the end of processing. Thereby, accepting strings that do not represent the identity as we have outlined. ♦

We define a relation  $\bowtie$  between  $X_n \cup (Y \cup T')$  and  $X_n^c \cup \{P\}$  as follows.

Let  $a \in \{a_1^\pm, a_2^\pm, \dots, a_n^\pm\}$ . Then  $a \bowtie A$  for all  $A \in \{A_1, A_2, \dots, A_n\}$ . Further let  $b \in Y \cup T'$ . Then  $b \bowtie P$ . Further, suppose  $d \in X_n \cup (Y \cup T')$  and  $D \in X_n^c \cup \{P\}$  such that  $d \bowtie D$ . Then  $d = a_i^\pm$  for some  $i \in \{1, 2, \dots, n\}$ , or  $d \in Y \cup T'$ . If  $d = a_i^\pm$  and  $D = A_i$  then we write  $d \bowtie' D$ . Similarly if  $d \in Y \cup T'$  and  $D = P$  then write  $d \bowtie' D$ .

We shall now give an intuitive description of the automaton  $\mathcal{A}$  accepting the  $coWP(\mathbb{Z}^n * G, X_n \cup Y)$ .

**Intuitive Idea:** The check-stack contains a string of the form

$$x_1^{i_1} P^{j_1} x_2^{i_2} P^{j_2} \dots x_t^{i_t} P^{j_t}$$

where  $x_l \in X_n^c$  for all  $1 \leq l \leq t$ , where  $t \geq 1$ , and  $i_1, i_2, \dots, i_t, j_1, j_2, \dots, j_t \geq 1$ . The automaton works in multiple stages, and they are as follows.

- Reading input letter stage
- Locating the correct corresponding check-stack letter stage
- Pushing input letter onto the pushdown stack stage for input letters from  $\mathbb{Z}^n$  / Moving to the appropriate state stage for input letters from  $G$
- Pre-check stage
- Action stage
- Clean up of the stack stage
- Check stage

**Reading input letter stage:** In this stage, the automaton simply reads an input letter  $\sigma$  and moves to the next stage while writing a  $W$  onto the pushdown stack, this letter is read regardless of the contents of the check-stack and pushdown stack.

(It maybe helpful to the reader to identify  $W$  with the word “wait”, this is because this effectively what the automaton is doing every time  $W$  is at the top of the pushdown stack. That is, the automaton must make a decision however all the information needed to make the decision is not yet available and so it is “waiting” and finding out more information to be able to make the decision.)

**Locating the correct corresponding check-stack letter stage:** Suppose the corresponding check-stack letter to  $W$  (written onto the pushdown stack in the previous stage) is  $X$ . The goal of this stage of the automaton is to locate the first check-stack letter  $Y_1$  that is “above”  $X$ , and  $\sigma \bowtie Y_1$ . (We use the word “above” here, thinking of the stacks as in the diagrams of [2.2.4.1](#).)

If  $\sigma \bowtie X$  then the automaton moves to the next stage. Otherwise, the automaton locates the first  $Y_1$  such that  $\sigma \bowtie Y_1$  where  $Y_1$  is above  $X$  on the check-stack. First the automaton deletes  $W$  in order to view the check-stack letter  $Z$  below  $X$ . If  $\sigma \bowtie Z$  then the automaton moves to  $q_X$ . (We view this as a rejection.) If  $\sigma$  is not  $\bowtie$  related to  $Z$  then the automaton writes  $WM$  onto the stack. We note that  $W$  is now one position higher than where it was before.

(It maybe helpful to the reader to identify  $M$  with the phrase “move on”, as if  $M$  was written the automaton must have viewed that position before via a  $W$  and does not need to test it again.)

Now the automaton can view the check-stack letter one position above  $X$ . This process repeats until the first  $Y_1$  on the check-stack such that  $\sigma \bowtie Y_1$  is located. The automaton then moves to the next stage.

**Pushing input letter onto the pushdown stack stage for input letters from  $\mathbb{Z}^n$ /Moving to the appropriate state stage for input letters from  $G$ :**

At the end of the previous stage, we have found the first check-stack letter  $Y_1$  such that  $\sigma \bowtie Y_1$  (or moved to  $q_X$ ).

The automaton deletes  $W$  and writes  $\sigma$  if  $\sigma \in X_n$  and then moves to the pre-check stage

If  $\sigma \in Y$  then the automaton deletes  $W$  moves to an intermediate state labelled by  $\sigma$  (which we call  $W_\sigma$ ) revealing a letter representing a transversal element  $t$  at the top of the stack (if there is one), which we delete and move to state  $(t, \sigma)$ . If there is no transversal element  $t$ , then we simply move to  $(1_G, \sigma)$ . Then the automaton moves to the action stage.

**Pre-check stage:** At this stage, the top of the pushdown stack is  $\sigma$ , if  $\sigma \in X_n$  and the corresponding letter on the check-stack, say  $Y_1$  is such that  $\sigma \bowtie Y_1$ . We assume that  $\sigma \in X_n$ , as we do not enter this stage if  $\sigma \in Y$ .

If  $\sigma \bowtie' Y$  then the automaton deletes  $\sigma$  and moves to the action stage. (We delete  $\sigma$  as we wish to view the symbol underneath it on the pushdown stack. This is crucial to be able to act correctly. Since if the symbol underneath is  $\sigma^{-1}$  then a free reduction would result in deleting  $\sigma^{-1}$ . Otherwise, we need to write  $\sigma$  onto the pushdown stack.)

Otherwise, the automaton deletes  $\sigma$  and moves to the clean up of the stack stage.

**Action stage:** At the end of the last stage, the automaton deleted  $\sigma$  from the top of the pushdown stack, if  $\sigma \in X_n$ . In that case, if the top of the pushdown stack is  $\sigma^{-1}$  then we delete it. Otherwise, we write  $\sigma$  onto the pushdown stack. Then automaton moves to clean up of the stack stage.

If  $\sigma \in Y$ , the state is  $(t, \sigma)$  for some  $t \in T'$  (as seen in moving to the appropriate state stage). The automaton now simulates the sequence of transitions the automaton  $P_G^{(W,1)}$  uses from state  $(t, \sigma)$  until it reaches a state representing

a transversal element, say  $t'$ . However, between each transition simulating the aforementioned transitions in  $P_G^{(W,1)}$  the automaton goes into a mini-check stage. This stage simply checks that the next letter on the check-stack is  $P$ . If at any point the next letter on the check-stack is not  $P$ , the automaton moves to state  $q_X$ . Otherwise, the automaton finishes simulating the transitions in  $P_G^{(W,1)}$  (which in turn simulate reading  $w_{t\sigma}$  in the automaton for  $F_k$ ) ending at a state  $t'$  representing the transversal element  $t'$  where  $t\sigma =_G w_{t\sigma}t'$ . If  $t' = 1_G$ , then the automaton does another mini-check. If there is a letter at the top of the pushdown stack from the stack alphabet  $\chi_G$  of  $P_G^{(W,1)}$  then the automaton writes  $1_G$  onto the top of the pushdown stack and moves back to the reading stage. If there is no letter at the top of the pushdown stack from the stack alphabet  $\chi_G$  of  $P_G^{(W,1)}$  and  $t' = 1_G$  then the automaton moves to the clean up of the stack stage. If  $t' \neq 1_G$  then write  $t'$  onto the pushdown stack and moves to the reading stage.

**Clean up of the stack stage:** If at this stage the top of the pushdown stack is  $M$  then the automaton deletes it. This repeats until the top of the pushdown stack is not  $M$ , then the automaton moves back to the reading input letter stage.

**Check stage:** After the last letter has been read and the automaton moved back to the reading stage, the automaton moves to this stage. Here, if the pushdown stack is non-empty the automaton accepts the string.

We shall now give the formal definition of the automaton. We follow this with an informal description of the states. The reader may wish to take a detour there before reading the formal definition of  $\mathcal{A}$  as we link the states of the automaton to the stages discussed in the intuition above.

**Formal Definition of  $\mathcal{A}$ :** Before we define the automaton  $\mathcal{A}$  accepting the co-word problem of  $\mathbb{Z}^n * G$  we ask the reader to recall the states and transitions of  $P_G^{(W,1)}$  (as in [2.3.2](#)) as we shall use them below.

We define a CSPD automaton

$$\mathcal{A} := (Q, \Sigma, \Delta, \Gamma, I, F, \mathcal{R}, \perp, \delta)$$

accepting  $coWP(\mathbb{Z}^n * G, X_n \cup Y)$  as follows.

- The state set  $Q$  consists of the following states.
  - The state  $q_r$ .  
(This is the state from which the input letters are read. This is why this state has  $r$  as a subscript.)
  - The state  $q_A$ .  
(This is the accept state, and the subscript  $A$  stands for “accept”.)



- The state  $q_X$ .  
(This serves as a reject state.)
  - For every  $\sigma \in X_n \cup Y$  there is a state  $q_\sigma$ .
  - For every  $\sigma \in X_n \cup Y$  there are states  $q_{\sigma,l,1}$  and  $q_{\sigma,l,2}$ .  
(These states locate the correct entry on the check-stack. The  $l$  in the names of the states stand for the word “locate”.)
  - For every  $\sigma \in X_n$  there is a state  $P.D._\sigma$ .  
(This state pushes the letter  $\sigma$  onto the pushdown stack. P.D. stands for pushdown.)
  - For every  $\sigma \in Y$  there is a state  $W_\sigma$ .  
(The  $W$  in  $W_\sigma$  stands for “wait”.)
  - For every  $t \in T'$ , there is a state representing  $t$ , which we call by the same name.
  - For every  $t \in T'$  and every  $y \in Y$  we have a state  $(t_s, y)$ .
  - For every state  $q_{(t,y,i)} \in Q_G^{(W,1)}$  we have a state  $q_{(t_s,y,i)}$ .
  - For every state  $q_{(t_s,y,i)}$  defined above, we have two states state  $q_{(t_s,y,i)}^{(c,1)}$  and  $q_{(t_s,y,i)}^{(c,2)}$ . There is also a state  $1_G^c$ .  
(The states  $q_{(t_s,y,i)}^{(c,1)}$  and  $q_{(t_s,y,i)}^{(c,2)}$  serves to check whether the next letter on the check-stack is  $P$ . The state  $1_G^c$  whether there is a letter from  $\chi_G^{(W,1)}$  under the  $1_G$  on the stack. The superscript  $c$  stands for “check”.)
  - There is a stack  $C_p$ .  
(This state checks whether the letter at the top the pushdown stack is  $\bowtie'$  related to the corresponding letter on the check-stack, if the letter at the top of the pushdown stack is in  $X_n$ . We view this as a “pre-check”. Hence the subscript  $p$ .)
  - For every  $\sigma \in X_n$  there is a state  $A_\sigma$ .  
(This state simulates the procesing of  $\sigma$ , we call this “acting” by  $\sigma$ . The letter  $A$  in the name of the state stands for the word “act”.)
  - There is a state  $C_0$ .  
(This is referred to as the “clean-up” state.)
  - There is a state  $C_1$ .  
(This is referred to as the “check state”.)
- The input alphabet is  $\Sigma = X_n \cup Y$ .

- The check-stack alphabet is  $\Delta = X_n^c \cup \{P\}$ .
- The pushdown stack alphabet is  $\Gamma = X_n \cup \chi_{P_G^{(W,1)}} \cup T' \cup \{W, M\}$ .
- The set of initial states is  $I = \{q_r\}$ .
- The set of final states is  $F = \{q_A\}$ .
- The regular language  $\mathcal{R}$  is defined to consist of all the strings of the following form  $x_1^{i_1} P^{j_1} x_2^{i_2} P^{j_2} \dots x_t^{i_t} P^{j_t}$  where  $x_l \in X_n^c$  for all  $1 \leq l \leq t$ , where  $t \geq 1$ , and  $i_1, i_2, \dots, i_t, j_1, j_2, \dots, j_t \geq 1$ ,

and finally the transition relation  $\delta$  consists of the following transitions.

(T0) From  $q_r$  we have the following transitions. For every pair  $(X, Y_1)$  we have the following transitions.

(a) For every  $\sigma \in \Sigma$  there is a transition

$$((q_r, \sigma, (X, Y_1)), (q_\sigma, Y_1));$$

that is, there is a transition from  $q_r$  upon reading  $\sigma$  to  $q_\sigma$  for every pushdown stack letter  $Y_1$  and corresponding check-stack letter  $X$  that does not edit the stack.

(b) There is a transition

$$((q_r, \varepsilon, (X, Y_1)), (C_1, Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $q_r$  to  $C_1$  without editing the stack.

(T1) For every  $\sigma \in \Sigma$ , from  $q_\sigma$  we have the following transitions. For every pair  $(X, Y_1)$  there is a transition

$$((q_\sigma, \varepsilon, (X, Y_1)), (q_{\sigma,l,1}, WY_1));$$

that is, there is an  $\varepsilon$ -transition from  $q_\sigma$  to  $q_{\sigma,l,1}$  that adds  $W$  onto the pushdown stack.

(T2)(a) For every  $\sigma \in X_n$ , from  $q_{\sigma,l,1}$  we have the following transitions. For every pair  $(X, W)$  we have the following transitions.

(i) For every  $X$  such that  $\sigma \bowtie X$  there is transition

$$((q_{\sigma,l,1}, \varepsilon, (X, W)), (P.D.\sigma, W));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,1}$  to  $P.D.\sigma$  without editing the stack whenever  $\sigma \bowtie X$ .

(ii) For every  $X$  such that  $\sigma$  is not  $\bowtie$  related  $X$  there is a transition

$$((q_{\sigma,l,1}, \varepsilon, (X, W)), (q_{\sigma,l,2}, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,1}$  to  $q_{\sigma,l,2}$  deleting  $W$  whenever  $\sigma$  is not  $\bowtie$  related to  $X$ .

(T2)(b) For every  $\sigma \in Y$ , from  $q_{\sigma,l,1}$  we have the following transitions. For every pair  $(X, W)$  we have the following transitions:

(i) For every  $X$  such that  $\sigma \bowtie X$  there is transition

$$((q_{\sigma,l,1}, \varepsilon, (X, W)), (W_\sigma, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,1}$  to  $W_\sigma$  deleting  $W$  from the top of the stack whenever  $\sigma \bowtie X$

(ii) For every  $X$  such that  $\sigma$  is not  $\bowtie$  related  $X$  there is a transition

$$((q_{\sigma,l,1}, \varepsilon, (X, W)), (q_{\sigma,l,2}, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,1}$  to  $q_{\sigma,l,2}$  deleting  $W$  whenever  $\sigma$  is not  $\bowtie$  related to  $X$ .

(T3) For every  $\sigma \in \Sigma$ , from  $q_{\sigma,l,2}$  we have the following transitions. For every pair  $(X, Y_1)$  we have the following transitions.

(a) For every  $X$  such that  $\sigma \bowtie X$  there is a transition

$$((q_{\sigma,l,2}, \varepsilon, (X, Y_1)), (q_X, Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,2}$  to  $q_X$  without editing the stack, whenever  $\sigma \bowtie Y_1$ .

(b) For every  $X$  such that  $\sigma$  is not  $\bowtie$  related to  $X$  there is a transition

$$((q_{\sigma,l,2}, \varepsilon, (X, Y_1)), (q_{\sigma,l,1}, WMY_1));$$

that is, there is an  $\varepsilon$ -transition from  $q_{\sigma,l,2}$  to  $q_{\sigma,l,1}$  that adds  $WM$  onto the pushdown stack whenever  $\sigma$  is not  $\bowtie$  related to  $X$ .

(T4)(a) For every  $\sigma \in X_n$ , from  $P.D._\sigma$  we have the following transitions. For all pairs  $(X, W)$  such that  $\sigma \bowtie X$  there is a transition

$$((P.D._\sigma, \varepsilon, (X, W)), (C_p, \sigma));$$

that is, there is an  $\varepsilon$ -transition from  $P.D._\sigma$  to  $C_p$  that deletes  $W$  and writes  $\sigma$  instead whenever  $\sigma \bowtie X$ .

(T4)(b) For every  $\sigma \in Y$ , from  $W_\sigma$  we have the following transitions.

(i) For all pairs  $(X, t)$  where  $t \in T'$  there is a transition

$$((W_\sigma, \varepsilon, (X, t)), ((t, \sigma), \varepsilon));$$

i.e., there is an  $\varepsilon$ -transition from  $W_\sigma$  deleting  $t \in T'$  from the top of the stack and moving to state  $(t, y)$ .

(ii) For all pairs  $(X, Y_1)$  where  $Y_1 \notin \chi_G^{(W,1)}$  there is a transition

$$((W_\sigma, \varepsilon, (X, Y_1)), ((1_G, \sigma), Y_1));$$

i.e., there is an  $\varepsilon$ -transition from  $W_\sigma$  to  $(1_G, \sigma)$  that does not edit the stack.

(T5) From  $C_p$  we have the following transitions. For all pairs  $(X, \sigma)$  where  $\sigma \bowtie X$  and  $\sigma \in X_n$  we have the following transitions.

(a) For every  $X$  such that  $\sigma \bowtie' X$  there is a transition

$$((C_p, \varepsilon, (X, \sigma)), (A_\sigma, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $C_p$  to  $A_\sigma$  deleting  $\sigma$  whenever  $\sigma \bowtie' X$ .

(b) For every  $X$  such that  $\sigma$  is not  $\bowtie'$  related to  $X$  there is a transition

$$((C_p, \varepsilon, (X, \sigma)), (C_0, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $C_p$  to  $C_0$  deleting  $\sigma$  whenever  $\sigma$  is not  $\bowtie'$  related to  $X$ .

(T6) For every  $\sigma \in X_n$ , from  $A_\sigma$  we have the following transitions. For every  $X \in \Delta$  we have the following transitions.

(a) There is a transition

$$((A_\sigma, \varepsilon, (X, \sigma^{-1})), (C_0, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $A_\sigma$  to  $C_0$  that deletes  $\sigma^{-1}$  if  $\sigma^{-1}$  is at the top of the pushdown stack.

(b) For every  $Y_1 \in \Gamma \setminus \{\sigma^{-1}\}$ , there is a transition

$$((A_\sigma, \varepsilon, (X, Y_1)), (C_0, \sigma Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $A_\sigma$  to  $C_0$  that pushes  $\sigma$  onto the stack (i.e., the automaton writes  $\sigma Y_1$ ) whenever  $Y_1 \neq \sigma^{-1}$ .

(T7) Let  $t \in T$  and  $y \in Y$ . Further let  $w_{ty} = z_1 z_2 \cdots z_r$  and  $ty =_G w_{ty} t_1$ .

(T7)(a) (i) For every pair  $(X, Y_1)$ , there is a transition

$$(((t, y), \varepsilon, (X, Y_1)), (q_{(t,y,1)}^{(c,1)}, WY_1)),$$

that is there is an  $\varepsilon$ -transition from  $(t, y)$  to  $q_{(t,y,1)}^{(c,1)}$  that pushes  $W$  onto the stack.

(ii) For every pair  $(X, W)$  such that  $y$  is not  $\bowtie$  related to  $X$  there is a transition

$$((q_{(t,y,1)}^{(c,1)}, \varepsilon, (X, W)), (q_X, W));$$

that is, there is an  $\varepsilon$ -transition from the state  $q_{(t,y,1)}^{(c,1)}$  to  $q_X$  without editing the stack when  $y$  is not  $\bowtie$  related to  $X$ .

(iii) For every  $(X, W)$  such that  $y \bowtie X$  there is a transition

$$((q_{(t,y,1)}^{(c,1)}, \varepsilon, (X, W)), (q_{(t,y,1)}^{(c,2)}, \varepsilon));$$

that is there is a  $\varepsilon$ -transition from  $q_{(t,y,1)}^{(c,1)}$  to  $q_{(t,y,1)}^{(c,2)}$  deleting  $W$  whenever  $W$  is at the top of the stack and with corresponding check-stack letter  $X$  where  $y \bowtie X$ .

(iv) For every pair  $(X, Y_1)$  such that  $Y_1 \in \chi_G^{(W,1)}$  there is a transition

$$((q_{(t,y,1)}^{(c,2)}, \varepsilon, (X, Y_1)), (q_{(t,y,1)}, P_{F_k}^{(W,1)}(i_0, z_1, Y_1)))$$

where  $P_{F_k}^{(W,1)}(i_0, z_1, Y_1)$  denotes the unique string such that

$$((i_0, z_1, Y_1), (i_0, P_{F_k}^{(W,1)}(i_0, z_1, Y_1))) \in \delta_{F_k}^{(W,1)};$$

that is, there is an  $\varepsilon$ -transition from  $q_{(t,y,1)}^{(c,2)}$  to  $q_{(t,y,1)}$  with  $Y_1$  at the top of the pushdown stack, with  $X$  being the corresponding check-stack letter writing the string that  $P_{F_k}^{(W,1)}$  writes if it read  $z_1$  with  $Y_1$  at the top of its stack.

(v) For every pair  $(X, Y_1)$  such that  $Y_1 \in X_n \cup \{M, \perp\}$  there is a transition

$$((q_{(t,y,1)}^{(c,2)}, \varepsilon, (X, Y_1)), (q_{(t,y,1)}, (P_{F_k}^{(W,1)}(i_0, z_1, \perp) - \perp)Y_1))$$

where  $(P_{F_k}^{(W,1)}(i_0, z_1, \perp) - \perp)Y_1$  denotes the concatenation of

$$P_{F_k}^{(W,1)}(i_0, z_1, \perp) - \perp$$

and  $Y_1$ , where  $P_{F_k}^{(W,1)}(i_0, z_1, \perp)$  denotes the unique string such that

$$((i_0, z_1, \perp), (i_0, P_{F_k}^{(W,1)}(i_0, z_1, \perp))) \in \delta_{F_n}^{(W,1)};$$

that is, there is an  $\varepsilon$ -transition from  $q_{(t,y,1)}^{(c,2)}$  to  $q_{(t,y,1)}$  with  $Y_1$  at the top of the pushdown stack with the corresponding check-stack letter being  $X$  writing what  $P_{F_k}^{(W,1)}$  writes if it read  $z_1$  with  $Y_1$  at the top of its stack (treating  $Y_1$  as a bottom of stack symbol).

(T7)(b) For every in  $i \in \{1, \dots, r-1\}$  we have the following transitions.

(i) For every pair  $(X, Y_1)$ , there is a transition

$$((q_{(t,y,i)}, \varepsilon, (X, Y_1)), (q_{(t,y,i)}^{(c,1)}, WY_1)),$$

that is there is an  $\varepsilon$ -transition from  $q_{(t,y,i)}$  to  $q_{(t,y,i)}^{(c,1)}$  that pushes  $W$  onto the stack.

(ii) For every pair  $(X, W)$  such that  $y$  is not  $\bowtie$  related to  $X$  there is a transition

$$((q_{(t,y,i)}^{(c,1)}, \varepsilon, (X, W)), (q_X, W));$$

that is, there is an  $\varepsilon$ -transition from the state  $q_{(t,y,i)}^{(c,1)}$  to  $q_X$  without editing the stack when  $y$  is not  $\bowtie$  related to  $X$ .

(iii) For every  $(X, W)$  such that  $y \bowtie X$  there is a transition

$$((q_{(t,y,i)}^{(c,1)}, \varepsilon, (X, W)), (q_{(t,y,i)}^{(c,2)}, \varepsilon));$$

that is there is a  $\varepsilon$ -transition from  $q_{(t,y,i)}^{(c,1)}$  to  $q_{(t,y,i)}^{(c,2)}$  deleting  $W$  whenever  $W$  is at the top of the stack and with corresponding check-stack letter  $X$  where  $y \bowtie X$ .

(iv) For every pair  $(X, Y_1)$  such that  $Y_1 \in \chi_G^{(W,1)}$  there is a transition

$$((q_{(t,y,i)}^{(c,2)}, \varepsilon, (X, Y_1)), (q_{(t,y,i+1)}, P_{F_k}^{(W,1)}(i_0, z_{i+1}, Y_1)))$$

where  $P_{F_k}^{(W,1)}(i_0, z_{i+1}, Y_1)$  denotes the unique string such that

$$((i_0, z_{i+1}, Y_1), (i_0, P_{F_k}^{(W,1)}(i_0, z_{i+1}, Y_1))) \in \delta_{F_k}^{(W,1)};$$

that is, there is an  $\varepsilon$ -transition from  $q_{(t,y,i)}^{(c,2)}$  to  $q_{(t,y,i+1)}$  with  $Y_1$  at the top of the pushdown stack, with  $X$  being the corresponding check-stack letter writing the string that  $P_{F_k}^{(W,1)}$  writes if it read  $z_{i+1}$  with  $Y_1$  at the top of its stack.

(v) For every pair  $(X, Y_1)$  such that  $Y_1 \in X_n \cup \{M, \perp\}$  there is a transition

$$((q_{(t,y,i)}^{(c,2)}, \varepsilon, (X, Y_1)), (q_{(t,y,i+1)}, (P_{F_k}^{(W,1)}(i_0, z_{i+1}, \perp) - \perp)Y_1))$$

where  $(P_{F_k}^{(W,1)}(i_0, z_{i+1}, \perp) - \perp)Y_1$  denotes the concatenation of

$$P_{F_k}^{(W,1)}(i_0, z_{i+1}, \perp) - \perp$$

and  $Y_1$ , where  $P_{F_k}^{(W,1)}(i_0, z_{i+1}, \perp)$  denotes the unique string such that

$$((i_0, z_{i+1}, \perp), (i_0, P_{F_k}^{(W,1)}(i_0, z_{i+1}, \perp))) \in \delta_{F_n}^{(W,1)};$$

that is, there is an  $\varepsilon$ -transition from  $q_{(t,y,i)}^{(c,2)}$  to  $q_{(t,y,i+1)}$  with  $Y_1$  at the top of the pushdown stack with the corresponding check-stack letter being  $X$  writing what  $P_{F_k}^{(W,1)}$  writes if it read  $z_{i+1}$  with  $Y_1$  at the top of its stack (treating  $Y_1$  as a bottom of stack symbol).

(T7)(c)

(i) For every pair  $(X, Y_1)$ , there is a transition

$$((q_{(t,y,r)}, \varepsilon, (X, Y_1)), (q_{(t,y,r)}^{(c,1)}, WY_1));$$

that is, there is an  $\varepsilon$ -transition from  $q_{(t,y,r)}$  to  $q_{(t,y,r)}^{(c,1)}$  that pushes  $W$  onto the stack.

(ii) For every pair  $(X, W)$  such that  $y$  is not  $\bowtie$  related to  $X$  there is a transition

$$((q_{(t,y,r)}^{(c,1)}, \varepsilon, (X, W)), (q_X, W));$$

that is, there is an  $\varepsilon$ -transition from the state  $q_{(t,y,r)}^{(c,1)}$  to  $q_X$  without editing the stack when  $y$  is not  $\bowtie$  related to  $X$ .

(iii) For every  $(X, W)$  such that  $y \bowtie X$  there is a transition

$$((q_{(t,y,r)}^{(c,1)}, \varepsilon, (X, W)), (q_{(t,y,r)}^{(c,2)}, \varepsilon));$$

that is, there is a  $\varepsilon$ -transition from  $q_{(t,y,r)}^{(c,1)}$  to  $q_{(t,y,r)}^{(c,2)}$  deleting  $W$  whenever  $W$  is at the top of the stack and with corresponding check-stack letter  $X$  where  $y \bowtie X$ .

(iv) For every pair  $(X, Y_1)$  such that  $Y_1 \in \chi_G^{(W,1)}$  there is a transition

$$((q_{(t,y,r)}^{(c,2)}, \varepsilon, (X, Y_1)), (t_1, t_1Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $q_{(t,y,r)}^{(c,2)}$  to  $t_1$  with  $Y_1$  at the top of the pushdown stack, with  $X$  being the corresponding check-stack letter writing  $t_1$  onto the pushdown stack.

(v) For every pair  $(X, Y_1)$  such that  $Y_1 \in X_n \cup \{M, \perp\}$  there is a transition

$$((q_{(t,y,r)}^{(c,2)}, \varepsilon, (X, Y_1)), (t_1, t_1 Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $q_{(t,y,r)}^{(c,2)}$  to  $t_1$  with  $Y_1$  at the top of the pushdown stack with the corresponding check-stack letter being  $X$  writing  $t_1$  onto the pushdown stack.

(T7)(d)

(i) From the state  $1_G$ , for every pair  $(X, 1_G)$  there is a transition

$$((1_G, \varepsilon, (X, 1_G)), (1_G^c, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $1_G$  to  $1_G^c$  that deletes the  $1_G$  from the top of the pushdown stack.

(ii) From the state  $1_G^c$ , for every pair  $(X, Y_1)$  such that  $Y_1 \in \chi_G^{(W,1)}$  there is a transition

$$((1_G^c, \varepsilon, (X, Y_1)), (q_r, 1_G Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $1_G^c$  to  $q_r$  pushing  $1_G$  onto the pushdown stack whenever the top of the pushdown stack is an element of  $\chi_G^{(W,1)}$ .

(iii) From the state  $1_G^c$ , for every pair  $(X, Y_1)$  such that  $Y_1 \notin \chi_G^{(W,1)}$  there is a transition

$$((1_G^c, \varepsilon, (X, Y_1)), (C_0, Y_1));$$

that is, there is an  $\varepsilon$ -transition from the state  $1_G^c$  to  $C_0$  without editing the stack whenever the top of the pushdown stack is not an element of  $\chi_G^{(W,1)}$ .

(T8) From  $C_0$  we have the following transitions. For every  $X \in \Delta$  we have the following transitions.

(a) There is a transition

$$((C_0, \varepsilon, (X, M)), (C_0, \varepsilon));$$

that is, there is an  $\varepsilon$ -transition from  $C_0$  to  $C_0$  that deletes  $M$  if at  $M$  is at the top of the pushdown stack.

(b) For every  $Y_1 \in (\Gamma \setminus \{W, M\}) \cup \{\perp\}$  there is a transition

$$((C_0, \varepsilon, (X, Y_1)), (q_r, Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $C_0$  to  $q_r$  that does not edit the stack, whenever  $Y_1 \notin \{W, M\}$ .



(T9) From  $C_1$  we have the following transitions.

(a) For every  $(X, Y_1) \in \Delta \times \Gamma$  there is a transition

$$((C_1, \varepsilon, (X, Y_1)), (q_A, Y_1));$$

that is, there is an  $\varepsilon$ -transition from  $C_1$  to  $q_A$  whenever the pushdown stack.

(b) There is a transition

$$((C_1, \varepsilon, (\perp, \perp)), (q_X, \perp));$$

that is, there is an  $\varepsilon$ -transition from  $C_1$  to  $q_X$  whenever the pushdown stack is empty.

Before we prove Theorem [4.7.1](#) we shall give an informal description of the states and the processes they achieve.

**Description of States:** As mentioned in the intuitive description of the machine, the automaton works in multiple stages. Here we shall specify what states make up these stages.

**Reading Stage:** The state  $q_r$  is responsible for this stage. Upon reading an input letter  $\sigma$  from  $q_r$  the automaton moves to state  $q_\sigma$ .

**Locating the correct corresponding check-stack letter stage:** The states  $q_{\sigma,l,1}$  and  $q_{\sigma,l,2}$  are responsible for this stage. The automaton moves from  $q_\sigma$  to  $q_{\sigma,l,1}$  while adding a  $W$  onto the stack. The automaton then uses the states  $q_{\sigma,l,1}$  and  $q_{\sigma,l,2}$  in the way we described in the intuitive description until the correct letter has been found or the string is immediately rejected. Then the automaton moves to the following stage by moving to the state  $P.D._\sigma$  if  $\sigma \in X_n$  and  $W_\sigma$  if  $\sigma \in Y$ .

**Pushing input letter onto the pushdown stack stage for input letters from  $\mathbb{Z}^n$ /Moving to the appropriate state stage for input letters from  $G$ :** The state  $P.D._\sigma$  is responsible for this stage (for every  $\sigma \in X_n$ ). It deletes  $W$  and writes  $\sigma$ . The automaton then moves to the next stage by moving to the state  $C_p$ .

For every  $\sigma \in Y$ , before moving to  $W_\sigma$  the automaton deletes  $W$ . From  $W_\sigma$  if the top of the stack is  $t$  (for some  $t \in T'$  then the automaton moves to state  $(t, \sigma)$ . If there is no transversal element  $t$ , then we simply move to  $(1_G, \sigma)$ . Then the automaton moves to the action stage.

**Pre-check stage:** The state  $C_p$  is responsible for this stage. At the top of the pushdown stack there is an input letter from  $X_n$ , say  $\sigma \in X_n$ , with the corresponding check-stack letter being  $Y_1$  where  $\sigma \bowtie Y_1$ .

If  $\sigma \not\bowtie Y_1$  then it deletes  $\sigma$  and moves to the action stage.

Otherwise, it deletes  $\sigma$  and moves to the clean up stage.

**Action stage:** For every  $\sigma \in X_n$  the state  $A_\sigma$  is responsible for this stage. If the top of the pushdown stack is not  $\sigma^{-1}$  then we simply write  $\sigma$  onto the pushdown stack and move to the clean up of the stack stage. Otherwise, we delete  $\sigma^{-1}$  from the top of the stack, and move to the clean up of the stack stage.

For every  $\sigma \in Y$ , the states that are responsible for this stage are  $q_{(t,y,i)}$  for every  $1 \leq i \leq |w_{ty}|$ . The states  $q_{(t,y,i)}^{(c,1)}$  and  $q_{(t,y,i)}^{(c,2)}$  are responsible for the mini-check stages, where the automaton checks that the next letter on the check-stack is  $P$ . The transitions simulated are those of  $P_G^{(W,1)}$ , and so at the end of reading  $y$  the pushdown stack of the automaton will be the same as the pushdown stack of  $P_G^{(W,1)}$  if  $y$  was read from  $t$ . Then the automaton writes the transversal element  $t_1$  onto the pushdown stack where  $ty =_G w_{ty}t_1$ . After that, the automaton makes sure there are some letters from  $\chi_G^{(W,1)}$  if  $t_1 = 1_G$  using state  $1_G^c$ . Then the automaton moves to the reading stage if the later holds, or  $t_1 \neq 1_G$ . Otherwise, it moves to the clean up of the stack stage.

**Clean up of the stack stage:** The state  $C_0$  is responsible for this stage. If the top of the pushdown stack is not  $M$  then we move back to the reading stage. Otherwise, we delete  $M$  until the top of the pushdown stack is not  $M$ .

**Check stage:** The state responsible for this is  $C_1$ . The automaton moves from  $q_r$  to  $C_1$ , and from  $C_1$  there is a transition to  $q_A$  if the pushdown stack is not empty.

## 4.7.2 Proof of Theorem [4.7.1](#)

In this section, we shall prove the theorem. We do so by first explaining the process by which the automaton reads a string. It will be useful to the reader to revisit the description of the states and the intuitive description, as we shall rely on the stages we described above.

**Explanation of how a string is read:** Let  $\omega = w_1w_2 \cdots w_t$  be a string in the generators of  $\mathbb{Z}^n * G$ , where  $w_i$  are syllables. We shall describe how  $\mathcal{A}$  reads  $\omega$  and then we shall follow that with the proof that  $\mathcal{L}(\mathcal{A}) = coWP(\mathbb{Z}^n * G, X_n \cup Y)$ .

Recall that the automaton accepting the co-word problem of  $\mathbb{Z}^n * \mathbb{Z}^m$  treated letters on the pushdown stack that were not from  $X_n$  as bottom of stack symbols when processing a syllable belonging to  $\mathbb{Z}^n$ . The automaton  $\mathcal{A}$  here, treats letters on the pushdown stack not from  $X_n$  as bottom of stack symbols when processing a syllable belonging to  $\mathbb{Z}^n$  as well. Similarly the automaton  $\mathcal{A}$  here, treats symbols on the pushdown stack that are not from  $\chi_G^{(W,1)} \cup T'$  as bottom of stack symbols when processing a syllable from  $G$ .

The automaton reads a syllable  $w_i \in X_n^*$  in exactly the same way the automaton accepting the co-word problem of  $\mathbb{Z}^n * \mathbb{Z}^m$  does. (Since the transitions concerning the generators  $X_n$  are the same.) Therefore, we may assume that during processing there are enough subsequent letters on the check-stack allowing the entire syllable to be processed in the same way, as we did in [4.6.2](#). Further, as in [4.6.2](#) upon reading a  $w_i$ , if the appropriate check-stack letter is  $A_j$ , then the automaton will freely reduce the projection  $w_i$  onto the free subgroup generated by  $a_j$ . Let that free reduction be  $u$ . Hence the top of the stack will consist of the free reduction of  $u$  and the maximal prefix of the pushdown stack with letters in  $X_n$ . This is exactly the same as it was in [4.6.2](#).

Now suppose the syllable read is  $w_j \in Y^*$ . If  $w_j \neq \varepsilon$  then the automaton proceeds to read it in the way we described in the intuitive description. That is, it reads it letter by letter where upon a letter  $x$  it moves to state  $q_x$ . Then it locates the correct corresponding check-stack letter. It does so by pushing a  $W$  onto the pushdown stack via (T1). Now the automaton uses the transitions in (T2)(b)(ii) and (T3) to find the first check-stack letter  $P$  such that  $x \bowtie P$ . Upon doing so,  $W$  is deleted from the top of the pushdown stack and moving to state  $W_x$  by (T2)(b)(i). Now the automaton views the letter at the top of the pushdown stack. If that letter is  $t$  where  $t$  represents a transversal element. Then the automaton moves to state  $(t, x)$  while deleting  $t$  by (T4)(b)(i). Otherwise, the automaton uses (T4)(b)(ii) moving to state  $(1_G, x)$ . Suppose we have reached  $(t', x)$  for some transversal element  $t'$ .

From this point onwards the automaton uses the states  $q_{(t',x,1)}, q_{(t',x,2)}, \dots, q_{(t',x,r)}$  where  $r$  is the length of  $w_{t'x}$ . These states simulate reading  $w_{t'x}$  in  $P_{F_n}^{(W,1)}$ . However, the automaton also uses states  $q_{(t',x,i)}^{(c,1)}$  and  $q_{(t',x,i)}^{(c,2)}$  for  $1 \leq i \leq r$  in order to check that the next letter on the check-stack is also  $P$ . If at any point that is not the case, the automaton immediately rejects. This is done through (T7)(a) and (T7)(b). Finally the automaton moves to a state  $t_1$  such that  $t'x =_G w_{t'x}t_1$ , this is done using (T7)(c). Here the automaton writes  $t_1$  onto the pushdown stack. Then if  $t_1 = 1_G$  the automaton uses transitions in (T7)(d) to check whether there is a letter under  $1_G$  from  $\chi_G^{(W,1)}$ , if not the automaton moves to  $C_0$ . If there was a letter under  $1_G$  from  $\chi_G^{(W,1)}$  then the automaton moves to  $q_r$  and another letter is read. At  $C_0$  the automaton deletes  $W$  off the top of the pushdown stack and then moves to  $q_r$  using the transitions in (T8). The process then repeats until the syllable is read.

Observe that if at the end of reading a syllable (or a part) using the transitions we described, and the stack does not have a symbol from  $\chi_G^{(W,1)}$  then it must be that the syllable (or part) was equal to the identity in  $G$ . This is because that is the only way the stack is empty in  $P_G^{(W,1)}$ .

There are no other choices that the automaton can make at any point during its

processing. Thus the way it reads a string is unique given a choice of check-stack.

*Proof of Theorem 4.7.1.* Let  $\omega \in WP(\mathbb{Z}^n * G, X_n \cup Y)$ . Suppose

$$\omega = u_1 v_1 u_2 v_2 \cdots u_t v_t$$

where  $u_i \in X_n^*$  and  $v_i \in Y^*$ . Since  $\omega \in WP(\mathbb{Z}^n * G, X_n \cup Y)$  either every syllable is equal to the identity in its respective group, or not all syllables are equal to the identity in their respective groups.

If every syllable is equal to the identity then

$$\gamma_{\{a_r, a_r^{-1}\}}(u_i) = \varepsilon$$

for every  $r \in \{1, 2, \dots, n\}$  and every  $i \in \{1, 2, \dots, t\}$ . Similarly, since  $v_j = 1_G$  for every  $j \in \{1, 2, \dots, t\}$  upon reading  $v_j$  using  $P_G^{(W,1)}$  the automaton  $P_G^{(W,1)}$  will return to state  $1_G$  with an empty stack.

Therefore, for any long enough choice of check-stack where there is enough letters to process every syllable the automaton will process the syllables as explained earlier. Assuming there are enough check-stack letters to process every syllable, upon reading every syllable the pushdown stack will be as follows. Either

- $\gamma_X(u) \perp$  where  $u$  can be replaced by every syllable  $u_i$ , and  $X = \{a_{i_1}, a_{i_1}^{-1}\}$  where  $i_1$  is determined by the index of the corresponding check-stack letter  $A_{i_1}$ . We note that  $A_{i_1}$  will be the first check-stack letter to appear on the check-stack. Therefore  $\gamma_X(u) = \varepsilon$  and thus the pushdown stack is empty. Alternatively, the pushdown stack will be
- $M^e \perp$  for some  $e > 0$  since  $P_G^{(W,1)}$  results in an empty stack with the state being  $1_G$  upon reading  $v_i$  for every  $i$ . However, the transitions in (T8)(a) will delete all occurrences of  $M$  and then stack will be empty.

Since the pushdown stack will be empty at the end of processing  $\omega$ , it will not be accepted since the transitions (T9)(a) are the only transitions to the accept state, and they can only be used if the stack is not empty.

Now we suppose that not all syllables are trivial. Let  $i_1 \in \{1, \dots, r\}$  be the smallest index such that  $w_{i_1} \neq 1$  (where 1 is the identity of the group which the letters of  $w_{i_1}$  belong to). Then there exists a next syllable  $w_j \neq 1$ . (Otherwise  $w \in coWP(\mathbb{Z}^n * G, \Sigma)$ .) As every  $w_k$  is equal to the identity of its respective group for  $i < k < j$ , we see that  $w_i$  and  $w_j$  belong to the same part and will be processed by the automaton as if they are one syllable. We note that otherwise the appropriate part of the check-stack is not long enough and the string gets rejected. That is, there exists a long enough appropriate substring of the check stack  $u_1^l$  where  $u$  is  $A_p$  (for some index  $p$ ) or  $P$  such that every letter of both  $w_i$  and

$w_j$  will be  $\bowtie$  related to  $u_1$ ) or the string will be rejected. If  $w_i w_j$  will be processed in the same way then either  $w_i w_j$  is equal to the identity or not. If  $w_i w_j$  is equal to the identity then we are done, since regardless of what  $u_1$  is, the pushdown stack will not contain any string corresponding to the part of  $u_1^l$  that was being used. Otherwise there must exist another syllable  $w_q$  that will also be in the same part. The automaton thus repeatedly finds these syllables, processing them from the same part of the check-stack until the pushdown stack is empty and thus the string is rejected. Otherwise the syllables cannot be processed in the same way due to  $l$  being too small and thus the automaton will also reject the string in that instance. Therefore any string in  $WP(\mathbb{Z}^n * G, X_n \cup Y)$  never gets accepted.

Now we suppose that  $\omega' \in coWP(\mathbb{Z}^n * G, X_n \cup G)$ . Suppose  $\omega' = w'_1 w'_2 \cdots w'_d$  where each  $w'_i$  is a syllable, and if  $w'_i \in X_n^*$  then  $w'_{i+1} \in Y^*$  and vice versa.

First suppose that every syllable is not equal to the identity in its respective group. As  $w_i$  is not equal to the identity for every  $i$  then there exists a generator  $\sigma_i$  for every  $w_i$  such that the sum of the exponents of  $\sigma_i$  is non-zero in  $w_i$  if  $w_i \in X_n^*$ . If  $w_i \in Y^*$  then the pushdown  $P_G^{(W,1)}$  will either not result in an empty stack or the state of  $P_G^{(W,1)}$  will not be  $1_G$ . If  $w_i \in X_n^*$  then  $\sigma_i \in \{a_1, a_2, \dots, a_n\}$ . By repeatedly picking these generators for each syllable in  $X_n$  we induce a sequence  $A'_1, A'_2, \dots, A'_d$  where  $A'_1, \dots, A'_d \in \{A_1, \dots, A_n\}$ . There exist lengths  $l_1, l_2, \dots, l_{2d}$  such that  $A_1^{l_1} P^{l_2} \dots A_d^{l_{2d-1}} P^{l_{2d}}$  is a long enough sequence on the check-stack such that  $w_1 w_2 \cdots w_d$  can be processed with the automaton.

Let  $\overline{w'_{i\sigma_i}}$  be the projection of the string  $w_i$  onto the free subgroup generated by  $\sigma_i$ , whenever  $w'_i \in X_n^*$ . Further let  $\rho(\overline{w'_{i\sigma_i}})$  be the reverse of the free reduction of  $\overline{w'_{i\sigma_i}}$ .

By the construction of the automaton and by our choice of check-stack, the pushdown stack will consist of the product of all  $\rho(\overline{w'_{i\sigma_i}})$  whenever  $w'_i \in X_n^*$ , with the product of stack that is produced by  $P_G^{(W,1)}$  upon reading  $w'_i$  whenever  $w'_i \in Y^*$  and the state that it ends on (with strings consisting of the letter  $M$  in between them). Since none of the images are trivial, and either the state is not  $1_G$  or the stack produced by  $P_G^{(W,1)}$  is non-empty upon reading a syllable that is not equal to the identity, the pushdown stack is non-empty and the string is accepted.

Now we may suppose that there is a syllable that is equal to the identity of its respective group. Let  $\tilde{\omega}$  be the reduced form of  $\omega$ . Note that  $\omega$  is related to  $\tilde{\omega}$  by a sequence of reductions of substrings of  $\omega$  that are trivial. Further there exists a string  $r \in \mathcal{R}$  such the automaton accepts  $\tilde{\omega}$  with  $r$  on the check stack by the above. Thus there exists a string  $r' \in \mathcal{R}$  such that  $r'$  consists of the same choices of letters of as  $r$ , but  $r'$  is just longer to accommodate for the processing of  $\omega$ . For example if  $r = A_1 A_1 P P A_4 A_4 P$  then the indices used for  $r'$  do not change, but we simply have a longer substring for each letter such as  $A_1 A_1 A_1 A_1 P P P A_4 A_4 A_4 A_4 P P P P$  would be a valid choice for  $r'$ . (The string may require more letters to be appended

afterwards but there is a prefix of  $r'$  where the indices used in the letters are determined by  $r$ .)

The automaton can process  $w$  and  $\tilde{w}$  with  $r'$  on the check-stack and accepts  $\tilde{w}$ . The reductions of substrings that are equal to the identity result in returning the stack to how it was before reading the substring. For strings in  $X_n^*$ , this is true regardless of the generator considered for the reduction. For strings in  $Y^*$  this is true by the way  $P_G^{(W,1)}$  is constructed. Thus we know that the pushdown stack upon reading  $w$  will be non-empty with  $r'$  on the check stack. Therefore  $w$  is accepted by the automaton. ■

# Chapter 5

## Epiregular Groups

### 5.1 Introduction

In this chapter we will introduce a new class of groups defined by a language theoretic property. We shall explore various closure properties of this class as well as distinguish it from a few well known classes of groups that are related to language theory.

This new class arose in an analysis by Jim Belk of a construction of Ville Salo [40]. Salo's construction embedded right angled Artin groups (RAAGs) into the Brin-Thompson's group  $2V$ . Further we show that the class of automatic groups is contained within the class of epiregular groups, and the Baumslag Solitar group  $BS(1, 2)$  is epiregular. Thus the class of epiregular groups is not the same as the class of  $co\mathcal{CF}$  groups nor it is equal to the class of automatic groups. This is because  $BS(1, 2)$  is known to neither be  $co\mathcal{CF}$  (as shown in [27]) nor is it automatic.

This chapter has become the basis for a paper currently in preparation joint with Collin Bleak and Luke Elliott where we explore the class of epiregular groups further.

**Definition 5.1.1.** Let  $G$  be a finitely generated group. Let  $X$  be a finite (symmetrically closed) generating set for  $G$ . We say that  $G$  is *epiregular with respect to  $X$*  if there exists a finite state automaton  $\mathcal{A}$  such that

- $\mathcal{L}(\mathcal{A}) \cap WP(G, X) = \emptyset$
- for every non-identity element,  $h$ , there exists  $w_h \in coWP(G, X)$  such that  $w_h \in \mathcal{L}(\mathcal{A})$  and  $w_h =_G h$ .



In the next section we shall show that being epiregular is a property of groups; i.e, it is independent of the generating set.

## 5.2 Well-definedness of epiregularity

**Theorem 5.2.1.** *Let  $G$  be a finitely generated group, and let  $X$  and  $Y$  be finite symmetrically closed generating sets for  $G$ . Then  $G$  is epiregular with respect to  $X$  if and only if  $G$  is epiregular with respect to  $Y$ .*

*Proof.* Suppose  $G$  is epiregular with respect to  $X$  then there exists a regular language  $L_X$  such that  $L_X \cap WP(G, X) = \emptyset$  and for every non-identity element  $h$  there exists  $w_h \in coWP(G, X)$  such that  $w_h \in L_X$  and  $w_h =_G h$ . Let  $\mathcal{A}_X = (Q_X, \Sigma_X, \delta_X, I_X, F_X)$  be a deterministic finite state automaton such that  $\mathcal{L}(\mathcal{A}_X) = L_X$ .

For every element  $x_i \in X$ , we fix a string  $w_i$  in  $w_i \in Y^*$  such that  $w_i =_G x_i$ .

$$\begin{aligned} x_1 &=_{G} y_{1,1}y_{1,2} \cdots y_{1,l(1)} && =: w_1, \\ x_2 &=_{G} y_{2,1}y_{2,2} \cdots y_{2,l(2)} && =: w_2, \\ &\cdot && \\ &\cdot && \\ &\cdot && \\ x_n &=_{G} y_{n,1}y_{n,2} \cdots y_{n,l(n)} && =: w_n. \end{aligned}$$

We define an automaton  $\mathcal{A}_Y = (Q_Y, \Sigma_Y, \delta_Y, I_Y, F_Y)$  as follows. First we set  $\Sigma_Y = Y$ ,  $I_Y = I_X$ , and  $F_Y = F_X$ . We define  $Q_Y$  and  $\delta_Y$  as follows.

For every state  $q \in Q_X$  and  $x \in \Sigma_X$  there is a unique state  $q'$  such that  $((q, x), q') \in \delta_X$ . By the above, there is a string  $w = y_1y_2 \cdots y_l$  of some length  $l$  such that  $w =_G x$ . We introduce  $l - 1$  states,  $q_{x,1}, q_{x,2}, \dots, q_{x,l-1}$  such that in  $\delta_Y$  we have the following transitions

$$\begin{aligned} &((q, y_1), q_{x,1}), \\ &((q_{x,1}, y_2), q_{x,2}), \\ &\cdot \\ &\cdot \\ &\cdot \\ &((q_{x,l-2}, y_{l-1}), q_{x,l-1}), \text{ and} \\ &((q_{x,l-1}, y_l), q'). \end{aligned}$$

We do this for every state  $q \in Q_X$  and every  $x \in \Sigma_X$ . These transitions are the only transitions in  $\delta_Y$  and  $Q_Y$  is a union of  $Q_X$  and the states  $q_{x,1}, q_{x,2}, \dots, q_{x,l-1}$  for every state  $q \in Q_X$  and every  $x \in \Sigma$ .



We observe that the edge from  $q$  to  $q'$  via the input letter  $x$  traversed in the graph of the transition diagram of  $\mathcal{A}_X$  defines a path from state  $q \in Q_X \subseteq Q_Y$  to state  $q' \in Q_X \subseteq Q_Y$  through the states  $q_{x,1}, q_{x,2}, \dots, q_{x,l-1}$  (defined above) via the string  $w$  that is traversed in the transition diagram of  $\mathcal{A}_Y$ . Conversely, a path from a state  $p \in Q_X \subseteq Q_Y$  to a state  $p' \in Q_X \subseteq Q_Y$  that does not pass through another  $p'' \in Q_X \subseteq Q_Y$  via a string  $w'$  must by definition be defined by an edge from state  $p$  to state  $p'$  via an input letter  $x'$  such that we have fixed  $w'$  such that  $w' =_G x'$ .

Therefore, for every string  $x_1x_2 \cdots x_r$  accepted by  $\mathcal{A}_X$ , there is a string  $w_1w_2 \cdots w_r$  accepted by  $\mathcal{A}_Y$  such that  $x_1x_2 \cdots x_r =_G w_1w_2 \cdots w_r$ . Further, by the paragraph above, for every string  $u$  accepted by  $\mathcal{A}_Y$  there is a string  $v$  accepted by  $\mathcal{A}_X$  and  $u =_G v$ . Thus there is no string accepted by  $\mathcal{A}_Y$  representing the identity element. Therefore  $\mathcal{A}_Y$  shows that  $G$  is epi-regular with respect to  $Y$ . ■

Let  $G$  be a group. Suppose there is a finite generating set  $X$  such that  $G$  is epi-regular with respect to  $X$ . Then by Theorem [5.2.1](#),  $G$  is epi-regular with respect to every finite generating set of  $G$ . Therefore we shall simply refer to  $G$  as an *epi-regular group*.

## 5.3 Examples

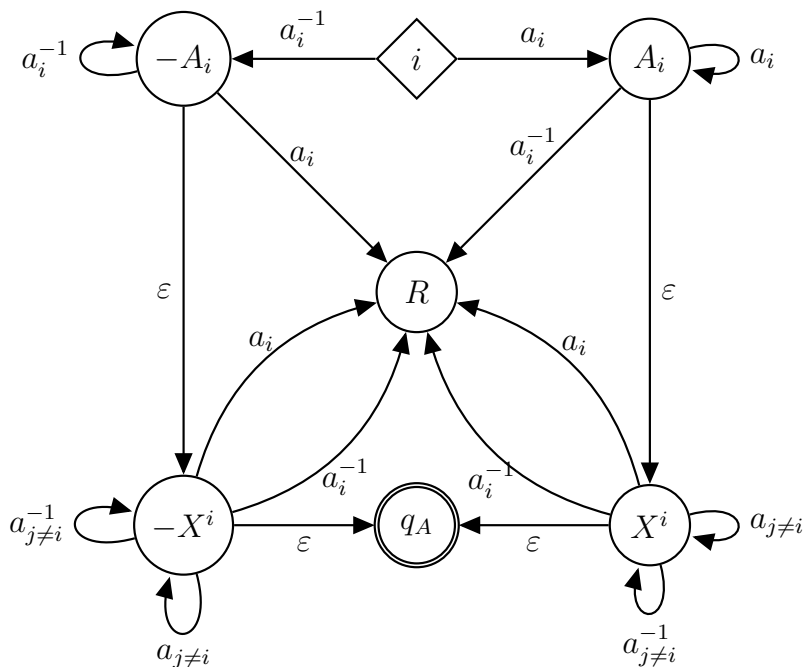
In this section, we shall show that some groups are epi-regular.

**Finite Groups** Finite groups are epi-regular. They are a very extreme case of epi-regular groups, this is due to the languages demonstrating their epi-regularity. By Anisimov's theorem (see Proposition [2.3.6](#)), there exists a finite state automaton accepting the entirety of the co-word problem. Further, finite groups are the only groups where a finite state automaton can accept all of the co-word problem. However, it is also possible to accept a language that is only finite demonstrating their epi-regularity. An example of this is a language consisting of the non-trivial elements over itself as an alphabet.

**The group  $\mathbb{Z}^n$**  The group  $\mathbb{Z}^n$  is epi-regular. We shall prove that  $\mathbb{Z}^n$  is epi-regular with respect to the standard generating set  $A := \{a_1^\pm, a_2^\pm, \dots, a_n^\pm\}$ .

Let  $i \in \{1, \dots, n\}$ . We will define automata  $\mathcal{A}_i$  such that  $\mathcal{L}_i := \mathcal{L}(\mathcal{A}_i)$  consists of strings of the form  $a_i^n w$  where  $n \in \mathbb{Z} \setminus \{0\}$  and  $w \in (A \setminus \{a_i^\pm\})^*$ . (We can think of each  $\mathcal{L}_i = \mathcal{L}_i^+ \cup \mathcal{L}_i^-$ , where  $\mathcal{L}_i^+$  consists of the subset of strings of  $\mathcal{L}_i$  that start with  $a_i$  and  $\mathcal{L}_i^-$  consists of the subset of strings of  $\mathcal{L}_i$  that start with  $a_i^{-1}$ .) Further let  $\mathcal{A} = \bigcup_{i=1}^n \mathcal{A}_i$ . This is the automaton that is required to show that  $\mathbb{Z}^n$  with respect to  $A$  is epi-regular.

Below we define  $\mathcal{A}_i$ .



As mentioned earlier, we may think of  $\mathcal{L}_i = \mathcal{L}_i^+ \cap \mathcal{L}_i^-$ , where  $\mathcal{L}_i^+$  consists of strings that do not represent the identity element that are accepted by paths reaching the accept state through the right side of the machine. Similarly  $\mathcal{L}_i^-$  consists of strings that do not represent the identity element that are accepted by paths reaching the accept state through the left side of the machine, i.e.

$$\mathcal{L}_i^+ = \{a_i^n w \mid n > 0 \text{ and } w \in (A \setminus \{a_i^\pm\})^*\} \subseteq \text{coWP}(\mathbb{Z}^n, A)$$

and

$$\mathcal{L}_i^- = \{(a_i^{-1})^n w \mid n > 0 \text{ and } w \in (A \setminus \{a_i^\pm\})^*\} \subseteq \text{coWP}(\mathbb{Z}^n, A).$$

**Proposition 5.3.1.**  $\mathbb{Z}^n$  is epiregular with respect to  $A$ .

*Proof.* We shall use  $\mathcal{A}$ . Let  $z \in \mathbb{Z}^n$ , we denote the sum of the exponents of the letter  $a_i$  in  $z$  by  $\text{exp}_i(z)$ .

Let  $g \in \mathbb{Z}^n \setminus \{1_{\mathbb{Z}^n}\}$ . Since  $g \neq 1_{\mathbb{Z}^n}$  there exists  $i \in \{1, 2, \dots, n\}$  such that  $\text{exp}_i(g) \neq 0$ . There exists a string  $w_g =_G g$  that begins with  $\text{exp}_i(g)$  occurrences of  $a_i$  and no further occurrences of  $a_i$  (if  $\text{exp}_i(g) < 0$  then there is a positive number of occurrences of  $a_i^{-1}$ ) which is accepted by  $\mathcal{A}_i$ . Thus  $w_g \in \mathcal{L}_i$ .

Let  $w \in \text{WP}(\mathbb{Z}^n, A)$ . Since  $w =_{\mathbb{Z}^n} 1_{\mathbb{Z}^n}$ , there does not exist  $j \in \{1, \dots, n\}$  such that  $\text{exp}_j(w) \neq 0$ ; i.e., for all  $j \in \{1, \dots, n\}$ ,  $\text{exp}_j(w) = 0$ . Therefore there is no  $t \in \{1, \dots, n\}$  such that  $w \in \mathcal{L}_t$ . ■

We shall now prove that some well known and well-studied classes of groups are epiregular. We first turn our attention to *automatic groups*.

### 5.3.1 Automatic groups

In this subsection, we shall prove that every automatic group is epiregular. We thank Sarah Rees for observing that fact. First we define some notions that are used in the definition of automatic groups. We do this for completeness, but we shall not use any of these concepts in the proof of Theorem 5.3.6. The reader will be able to understand the proof of Theorem 5.3.6 by reading Definition 5.3.4, as well as reading the statement of Proposition 5.3.5. For more information regarding automatic groups or any of the definitions involved, the reader may wish to consult [26] and [21].

We base our explanation of two-tape finite state automata below on Section 2.10.1 of Chapter 2 of [26].

We shall first introduce the notion of *padding*. The main objective of padding is to be able to read two strings at the same time, even if they have unequal length. We *pad* the shorter of the strings so that the resulting padded string has the same length as the longer string. We do this formally below.

**Definition 5.3.2** (Padding). Let  $A$  be an alphabet. We adjoin to  $A$  a *padding symbol*, denoted by  $\$$ , which is assumed not to lie in  $A$ , and we define  $B = A \cup \{\$\}$ . The *padded alphabet associated with  $A$*  is the set  $B$ . Let  $u, v \in A^*$ , where  $u = a_1 a_2 \cdots a_n$  and  $v = b_1 b_2 \cdots b_m$ , with each  $a_i, b_i \in A$ . We define  $(u, v)^P \in B \times B \setminus \{(\$, \$)\}$  to be the string

$$(\alpha_1, \beta_1)(\alpha_2, \beta_2) \cdots (\alpha_k, \beta_k),$$

where  $k := \max(n, m)$ , and

1.  $\alpha_i = a_i$  for  $1 \leq i \leq n$  and  $\alpha_i = \$$  for  $n < i \leq k$ ;
2.  $\beta_i = b_i$  for  $1 \leq i \leq m$  and  $\beta_i = \$$  for  $m < i \leq k$ .

We call  $(u, v)^P$  the *padded pair string corresponding to  $(u, v)$* . ♣

For example, the padded pair string corresponding to  $(hi, hello)$  is

$$(h, h)(i, e)(\$, l)(\$, l)(\$, o).$$

We are now ready to define a two-tape finite state automaton.

**Definition 5.3.3** (Two-tape finite state automaton). We say a finite state automaton  $M$  over  $A$  is a *two-tape finite state automaton* if  $M$  is a finite state automaton over the alphabet  $B \times B \setminus \{(\$, \$)\}$  such that all strings in  $\mathcal{L}(M)$  are of the form  $(u, v)^P$  for strings  $u, v \in A^*$ . ♣

We are now ready to define an automatic group as in [21].

**Definition 5.3.4** (Automatic Group). Let  $G$  be a group. An *automatic structure* on  $G$  consists of a set  $A$  that is a finite symmetrically closed generating set for  $G$ , a finite state automaton  $W$  over  $A$ , and finite state automata  $M_x$  over  $(A, A)$  for  $x \in A \cup \{\varepsilon\}$ , satisfying the following conditions.

1. For every element  $g \in G$ , there is a  $w_g \in \mathcal{W}$  such that  $w_g =_G g$ .
2. For  $x \in A \cup \{\varepsilon\}$ , the padded string corresponding to the pair  $(w_1, w_2)$  is accepted by  $M_x$  if and only if  $w_1x =_G w_2$ , and both  $w_1$  and  $w_2$  are elements of  $\mathcal{L}(W)$ .

We say  $G$  is *automatic* if there is an automatic structure on  $G$ . ♣

We will only use the language  $\mathcal{L}(W)$  in our proof of Theorem [5.3.6]. However, before we are ready to present our proof. We shall state the following proposition, which is proven in Theorem 2.5.1 of [21].

**Proposition 5.3.5.** *Let  $G$  be an automatic group as in Definition [5.3.4]. Then there exists a regular language  $L_{W'}$  such that for every group element  $g$  there exists a unique string  $w_g \in L_{W'}$  such that  $w_g =_G g$ .*

**Theorem 5.3.6.** *Every automatic group is epiregular.*

*Proof.* Let  $G$  be an automatic group as in Definition [5.3.4]. By Proposition [5.3.5], there exists a regular language  $L_{W'}$  of unique representatives for every group element of  $G$ . Let  $e$  be the unique string in  $L_{W'}$  such that  $e =_G 1_G$ . The set  $\{e\}$  is a regular language. Therefore  $L_{W'} \setminus \{e\}$  is also a regular language. By Proposition [5.3.5],  $L_{W'} \setminus \{e\}$  consists of unique representatives for every non-trivial element of  $G$ . Therefore  $G$  is epiregular. ■

We note that automatic groups have a number of interesting properties that we will not discuss here. However, it is well-known that the class of automatic groups is closed under taking free products and passing to finite index overgroups [26]. Therefore context-free groups are automatic since they are virtually free. So, context-free groups are epiregular.

**Corollary 5.3.7.** *Every context-free group is epiregular.*

In the next subsection, we turn our attention to the Baumslag-Solitar group  $BS(1, 2)$ .

### 5.3.2 Baumslag-Solitar group $BS(1, 2)$

**Theorem 5.3.8.** *The Baumslag-Solitar group  $BS(1, 2) = \langle a, b \mid bab^{-1} = a^2 \rangle$  is epiregular.*

*Proof.* In Section 2 of [14], Burillo and Elder give the following normal form. Every element can be expressed as  $b^{-i}a^kb^j$  such that

1.  $i$  and  $j$  are greater than or equal to 0, and
2. if  $i$  and  $j$  are greater than 0 then 2 does not divide  $k$ .

We note that the only expression in the stated normal form for the identity element is  $(b^{-1})^0a^0b^0$ . Thus we shall separate the expressions for non-trivial elements (i.e., all expressions excluding the one for the identity element) into four categories below.

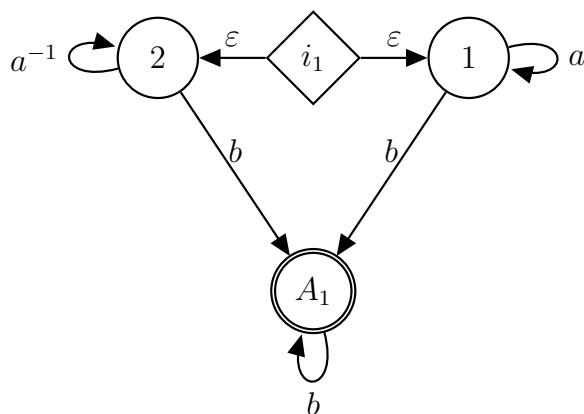
1. If  $i = 0$ , then the expression above is  $a^kb^j$  for  $j > 0$  and  $k \in \mathbb{Z}$ .
2. If  $j = 0$ , then the expression above is  $b^{-i}a^k$  for  $i > 0$  and  $k \in \mathbb{Z}$ .
3. If  $i = j = 0$ , then the expression above is  $a^k$  for  $k \in \mathbb{Z} \setminus \{0\}$ .
4. If  $i$  and  $j$  are strictly greater than 0, then the expression above is  $b^{-i}a^kb^j$  and 2 does not divide  $k$ , where  $k > 0$  or  $k < 0$ .

If we prove that each of the expressions above define a regular language then  $BS(1, 2)$  is epiregular since a finite union of regular languages is regular.

It remains to show that these languages are regular.

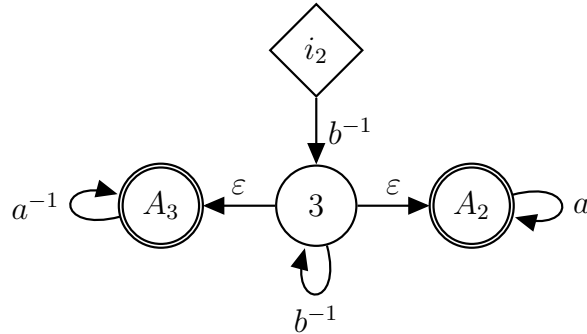
We shall draw four transition diagrams below, the automata defined by the diagrams accept exactly the expressions above in order.

The transition diagram for the first expression is the following.



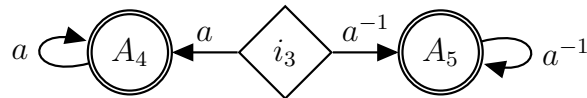
One can see that the language accepted by the above automaton is  $\{a^k b^j \mid k \in \mathbb{Z}, j > 0\}$

The transition diagram for the second expression is the following.



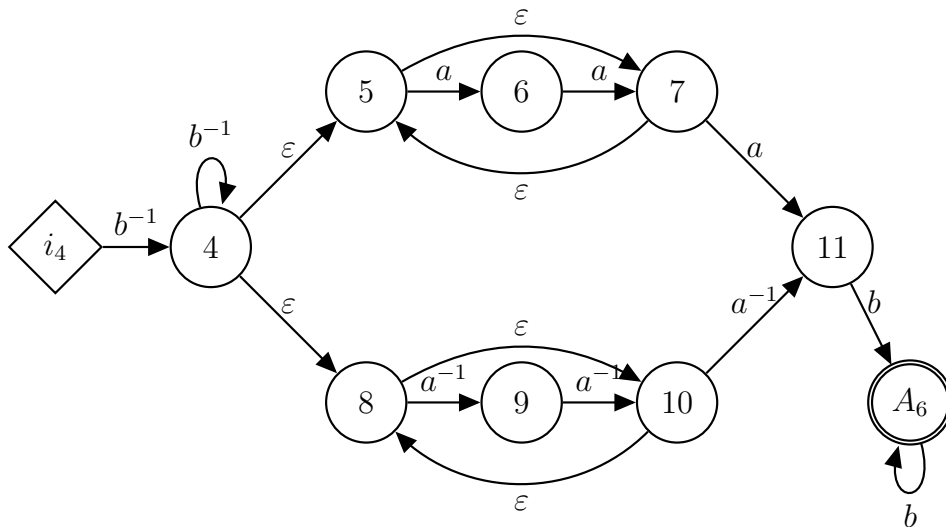
One can see that the language accepted by the above automaton is  $\{b^{-i} a^k \mid i > 0, k \in \mathbb{Z}\}$ .

The transition diagram for the third expression is the following.



One can see that the language accepted by the above automaton is  $\{a^k \mid k \in \mathbb{Z} \setminus \{0\}\}$ .

The transition diagram for the fourth expression is the following.



One can see that the language accepted by the above automaton is  $\{b^{-i} a^k b^j \mid i, j > 0, k = 1(mod 2)\}$ .

Since a finite union of regular languages is regular, the union of the above four languages demonstrates the epiregularity of  $BS(1, 2)$ . ■

The above proof technique can be easily adapted to the case of  $BS(1, n)$ . Thus we have the following corollary.

**Corollary 5.3.9.** *The Baumslag-Solitar group  $BS(1, n)$  is epiregular.*

We also conjecture that  $BS(m, n)$  is epiregular for any  $m$  and  $n$ . However, we have not yet checked this.

It is known in [13] that in general  $BS(1, n)$  is not automatic. Therefore we have the following corollary,

**Corollary 5.3.10.** *The class of epiregular groups is not equal to the class of automatic groups.*

Moreover since it is shown that  $BS(1, n)$  is not  $co\mathcal{CF}$  (for  $|n| \neq 1$ ) in [27]. We deduce the following corollary.

**Corollary 5.3.11.** *The class of epiregular groups is not equal to the class of  $co\mathcal{CF}$  groups.*

## 5.4 Closure Properties

In this section we shall prove that the class of epiregular groups is closed under passing to finite index overgroups and taking graph products of groups (see Definition 5.4.2).

**Theorem 5.4.1.** *Let  $H$  be an epiregular group. Let  $G$  be a finite index overgroup of  $H$ . Then  $G$  is epiregular.*

*Proof.* Let  $n$  be the index of  $H$  in  $G$ . Let  $X$  be a finite generating set for  $H$ . Let  $T = \{t_0, t_1, \dots, t_{n-1}\}$  be a transversal of  $H$  in  $G$ , with  $t_0 = 1_G$ .

Every element of  $G$  can be written as a product  $ht$  for some  $h \in H$  and  $t \in T$ . Therefore elements of  $G$  are either elements of  $H$ , elements of  $T$ , or a product of both.

Let  $L$  be a regular language over  $X$  demonstrating  $H$  is epiregular. We see that  $L' := L \cup (T \setminus \{1_G\}) \cup L(T \setminus \{1_G\})$  is a regular language (being the union of finitely many regular languages) over  $X \cup (T \setminus \{1_G\})$  demonstrating  $G$  is epiregular. ■

In the next subsection we prove that the class of epiregular group is closed under taking graph products.

### 5.4.1 Graph Products

First, we introduce the definition of graph products of groups as in [24].

**Definition 5.4.2.** Let  $\Gamma$  be a finite simple graph. For every  $v \in V(\Gamma)$ , let  $G_v$  be a finitely generated group presented by  $\langle X'_v \mid R'_v \rangle$ . We define the *graph product*  $G_\Gamma$  of the groups  $\{G_v \mid v \in V(\Gamma)\}$  over the graph  $\Gamma$  as the group with the following presentation

$$\langle X' \mid R' \rangle,$$

where  $X' = \bigcup_{v \in V(\Gamma)} X'_v$ , and  $R' \subseteq X'^*$  is the set of relations such that

$$R' = \left( \bigcup_{v \in V(\Gamma)} R'_v \right) \cup \{[x'_v, x'_w] \mid x'_v \in X'_v, x'_w \in X'_w, \{v, w\} \in E(\Gamma)\}.$$

We call  $G_v$  the *vertex group associated to  $v$*  (or by similar language). ♣

We note that since every  $G_v$  is finitely generated, by  $X_v$  say, then  $G_\Gamma$  is finitely generated by  $\bigcup_{v \in V(\Gamma)} X_v$ .

That is, the graph product of the groups  $\{G_v \mid v \in V(\Gamma)\}$  over the graph  $\Gamma$  is generated by all the groups  $G_v$  with the relations being a union of all the relations of the groups  $G_v$  together with relations that ensure  $[G_v, G_w] =_{G_\Gamma} 1_{G_\Gamma}$  for any  $\{v, w\} \in E(\Gamma)$ .

Before proving Theorem 5.4.10, we shall define some terminology and state some results from [25] which we shall rely on in our proof. For this purpose, we follow the exposition of [25], and redirect the reader to [25] for more details. We note that the work in [25] is based on [23].

For the remainder of this subsection, we will do the following. We shall set  $\Gamma$  be the finite graph. We will totally order the vertices of  $\Gamma$  and label them  $v_A, v_B, \dots, v_Z$  where the vertices are ordered lexicographically. (If there are more vertices, then simply choose an indexing set that is big enough with a natural ordering.) The associated vertex groups will be denoted by  $G_A, G_B, \dots, G_Z$  and their finite generating sets will be denoted by  $X_A, X_B, \dots, X_Z$ . We shall further set the union of the finite generating sets to be  $\Sigma$ .

We will use the capital letters  $I, J$ , and  $K$  to stand for arbitrary letters.

**Definition 5.4.3** (Local string and type). Let  $\omega \in \Sigma^*$ . A contiguous substring  $\omega'$  of  $\omega$  is said to be a *local string* if it is written in letters from the generators of exactly one vertex group, and there does not exist a longer contiguous substring containing  $\omega'$  that is written in letters of the generators of exactly one vertex group. The *type of a local string* is the label of the corresponding vertex. ♣

Since any string is a product of local strings we can define the *type of a string*



**Definition 5.4.4** (Type of a string). Let  $\omega = \omega_\epsilon \Sigma^*$  be a string. The product of the types of local strings of  $\omega$  is the *type* of  $\omega$ . Further, the *string of types* of  $\omega$  is the type of  $\omega$ . ♣

For example, suppose we have a string  $\omega = \omega_A \omega_C \omega_J$  where each  $\omega_I \in X_I^*$ . Then the type of  $\omega$  is  $ACJ$ .

**Definition 5.4.5** (Global Length). The *global length* of a string is the number of local strings into which it can be decomposed. ♣

We may *shuffle* local string  $\omega_I$  and  $\omega_J$  in a string if  $\omega_I$  and  $\omega_J$  are adjacent in that string, and  $\{I, J\} \in E(\Gamma)$ . That is, we may use the commuting relations defined by  $[G_I, G_J] =_{G_\Gamma} 1_{G_\Gamma}$  to obtain an equivalent string in the group when the corresponding vertices are adjacent. If in the process of shuffling, two local strings of the same type become adjacent then they amalgamate into a single local string. If it is equal to the identity of the corresponding group, then it is removed.

The process we describe above, will eventually stabilise global length. That is, it will no longer be possible to shuffle local strings of the same type to become adjacent. Then, we can choose a string of minimal type with respect to the lexicographic order.

**Definition 5.4.6.** The process outlined above, of taking a string, shuffling, amalgamating, and eventually finding a smallest representative with respect to Short-Lex ordering of types is called *pruning*. Further, a *pruning* of a  $\omega$  is a string obtained from  $\omega$  by the pruning process. ♣

We are now ready to state Proposition 3.1 of [25].

**Lemma 5.4.7** ([25], Proposition 3.1). *Let  $\omega$  and  $\omega'$  be strings in  $\Sigma^*$  such that  $\omega =_{G_\Gamma} \omega'$ . Further let  $T$  and  $T'$  be the types of prunings of  $\omega$  and  $\omega'$ , respectively. Then  $T = T'$ .*

The above result simply states that a group element uniquely determines the type of the pruning of strings representing the element. We note that the identity element is the only element with  $\epsilon$  as the type of any pruning.

In [25], the authors also give a normal form for graph products of groups (assuming one already has a normal form for every vertex group). They do this in Proposition 3.2. While we do not have such a normal form, we will state the conditions the strings must satisfy below as will make use of a construction relying on the property (O).

**Definition 5.4.8.** A string  $\omega$  is said to be *proper* if satisfies the following conditions.

- (L) Each local string is in its prescribed normal form.
- (O) If  $\omega = \cdots\omega_I\cdots\cdots\omega_J\cdots$  with  $I \geq J$  and  $\{v_I, v_J\} \in E(\Gamma)$  then there is a local string  $\omega_K$  such that  $\omega = \cdots\omega_I\cdots\omega_K\cdots\omega_J\cdots$  where  $\{v_K, v_J\} \notin E(\Gamma)$ .



The authors of [25] prove in Proposition 3.2 that the set of proper strings is a set of normal forms for  $G_\Gamma$ .

In the proof of Theorem B of [25], the authors construct a graph which they call the *admissible graph*. This graph is a finite state automaton  $\mathcal{A}_{AG}(\Gamma)$  accepting the proper strings of a graph product of cyclic groups of order 2. Further, since cyclic groups of order 2 have only one non-trivial element, a proper string corresponds exactly to the string of types of the proper string. Thus as in Theorem B of [25],  $\mathcal{A}_{AG}$  determines the types of strings that are allowed. That is, for every type of a proper string in a graph product of groups, the string can be read off by following a path in the admissible graph. Conversely, any path in the admissible graph reads the type of a proper string. In the proof below, we shall replace states of the admissible graph by automata that prove epiregularity for the vertex groups. First we shall state a few properties of the admissible graph so that our construction is clear. For the construction of the admissible graph itself, we redirect the reader to [25].

The admissible graph of a graph  $\Gamma$ ,  $\mathcal{A}_{AG}(\Gamma)$ , constructed in [25] is a finite state automaton with the following properties:

1. There is a unique initial state, this is not a final state.
2. Every non-initial state is labelled by some vertex  $v_I$  of the graph  $\Gamma$ . (The construction of the admissible graph does not guarantee a unique non-initial state for every vertex  $v_I$ .)
3. Every edge of the admissible graph from a state labelled by  $v_J \in V(\Gamma)$  to a state labelled by  $v_I \in V(\Gamma)$  has as a label the generator of  $G_I$ . (Recall the admissible graph is based on the graph product of cyclic groups of order 2 over  $\Gamma$ , hence these groups are generated by a single element.)
4. Every non-initial state is an accept state.
5. There are no  $\varepsilon$ -transitions in the admissible graph.
6. The language accepted by the admissible graph is the set of strings of types of proper strings of  $G_\Gamma$ .

Further, we extract the following statement from [25]. We do not prove this here as this covered in Proposition 3.2 as well as Theorem B of [25].

**Lemma 5.4.9.** *Let  $g \in G_\Gamma$  be a non-trivial element of  $G_\Gamma$ . Then there is a string  $\omega$  in the generators of  $G_\Gamma$  such that  $\omega =_{G_\Gamma} g$  and the type string of  $\omega$  is the type string of some proper string of the graph product of cyclic groups of order 2 over  $\Gamma$ . Further, every local string of  $\omega$  is non-trivial in its corresponding vertex group.*

We are now ready to prove the following theorem.

**Theorem 5.4.10.** *Let  $\Gamma$  be a finite graph as above. For every  $v_I \in V(\Gamma)$ , let  $G_I$  be an epi-regular group with finite generating set  $X_I$ . Let  $G_\Gamma$  be the graph product of groups  $\{G_I \mid v_I \in V(\Gamma)\}$  over the graph  $\Gamma$ . Then  $G_\Gamma$  is epi-regular.*

*Proof.* For every  $v_I \in V(\Gamma)$ , let  $\mathcal{A}_I$  be a deterministic finite state automaton demonstrating epi-regularity of  $G_I$ . (We call these  $\mathcal{A}_I$  sub-automata.) We will construct an automaton  $\mathcal{A}$  demonstrating epi-regularity of  $G_\Gamma$  by “gluing”  $\mathcal{A}_I$  onto every vertex of the admissible graph  $\mathcal{A}_{AG}(\Gamma)$  that is labelled by  $v_I$  as follows.

Let  $q$  and  $p$  be states of  $\mathcal{A}_{AG}(\Gamma)$  that are labelled by  $v_I$  and  $v_J$ , respectively. Suppose there is an edge into  $q$  from state  $p$  in  $\mathcal{A}_{AG}(\Gamma)$ . Remove that edge, and insert an edge with label  $x_I \in X_I$  from the final state of  $\mathcal{A}_J$  to the unique state  $q'$  that is reached from the initial state of  $\mathcal{A}_I$  upon reading  $x_I$ . Do this for every  $x_I \in X_I$  and all states  $p$  and  $q$  of  $\mathcal{A}_{AG}(\Gamma)$ . (We note that since there may be multiple states in  $\mathcal{A}_{AG}(\Gamma)$  labelled by the same vertex of  $\Gamma$ , we take disjoint copies of  $\mathcal{A}_I$  as needed to ensure we do not identify the states of  $\mathcal{A}_{AG}(\Gamma)$  with each other.) The initial state of this new automaton  $\mathcal{A}$  will be the initial state of  $\mathcal{A}_{AG}(\Gamma)$ . The final states consist of all the final states of every copy of  $\mathcal{A}_I$  that we have glued onto the admissible graph by the above process. (If there is an edge from a sub-automaton to another, then it is from a final state of one sub-automaton to the initial state of the other.)

We note that any path in  $\mathcal{A}$  projects onto a path in  $\mathcal{A}_{AG}(\Gamma)$  by identifying all the edges in a sub-automaton  $\mathcal{A}_I$ , and edges between a final state of a sub-automaton and the initial state of another will be edges between the vertices of  $\mathcal{A}_{AG}(\Gamma)$  that the sub-automata were glued onto. Therefore, any path in  $\mathcal{A}$  must yield the same type string as a path in  $\mathcal{A}_{AG}(\Gamma)$ .

Since the initial state is not an accept state and the identity element of  $G_\Gamma$  has type  $\varepsilon$ , by Lemma 5.4.7 the identity element of  $G_\Gamma$  is not represented by any accepted string of the above automaton. By Lemma 5.4.9, we know that every non-trivial element of  $G_\Gamma$  can be expressed as a string whose type is the type of a proper string of the graph product of cyclic groups of order 2 over  $\Gamma$ , with every local string  $\omega_I$  being non-trivial in  $G_I$ . For each such local string, there is a string  $\omega'_I$  in the language accepted by  $\mathcal{A}_I$  such that  $\omega_I =_{G_I} \omega'_I$ . Thus the automaton  $\mathcal{A}$  simply replaces the types of proper strings of the graph product of cyclic groups with these local strings  $\omega'_I$  while preserving the type string. Therefore there is a string representing every non-trivial element of  $G_\Gamma$  in  $\mathcal{L}(\mathcal{A})$ , and there is no string in  $\mathcal{L}(\mathcal{A})$  representing the identity element of  $G_\Gamma$ . Therefore  $G_\Gamma$  is epi-regular. ■

# Chapter 6

## Conclusion

There have been two main topics that we have explored in this thesis which we discuss below. The first was exploring the class of co-ETOL groups. We have done this in Chapter 4. We have proven in Theorems [4.2.1](#), [4.3.1](#), [4.4.1](#) and [4.5.1](#) that the class of co-ETOL groups is closed under the following operations:

- taking finitely many direct products,
- passing to finitely generated subgroups,
- passing to finitely indexed overgroups, and
- taking standard restricted wreath products with virtually free top groups.

Since [\[27\]](#), it is unknown whether the class of  $co\mathcal{CF}$  groups is closed under taking free products with finitely many factors. Indeed it is unknown whether  $\mathbb{Z}^2 * \mathbb{Z}$  is  $co\mathcal{CF}$ . However, in [\[28\]](#) it was shown that  $\mathbb{Z}^2 * \mathbb{Z}$  is co-indexed. We have shown that it is in fact co-ETOL in Theorem [4.6.1](#). More generally we have shown that  $\mathbb{Z}^n * \mathbb{Z}^m$  is co-ETOL. Further, we have shown  $\mathbb{Z}^n * G$  is co-ETOL for any virtually free  $G$  in Theorem [4.7.1](#). The methods we use are similar to those used in [\[28\]](#) to prove stack groups are closed in free products.

Recall that stack groups are co-indexed groups with extra restrictions on the automata accepting the co-word problem. It is possible to introduce analogous restrictions on automata proving a group is co-ETOL. This defines a subclass of co-ETOL groups. In this discussion, we shall refer to this class of groups as *PD-ish*. An area of possible future research is to study the class of PD-ish groups. We conjecture that every group currently known to be a stack group is in fact PD-ish. Further we conjecture the class of PD-ish groups is closed under all the operations mentioned above, and is also closed under taking free products with finitely many factors.

It is currently unknown whether the classes of  $co\mathcal{CF}$ , co-ETOL, co-indexed, and stack groups are all equal. It is unclear whether the class of PD-ish groups is a proper subclass of co-ETOL. It is also unclear what the relation is between the class of PD-ish groups and the classes of  $co\mathcal{CF}$ , co-indexed, and stack groups. Indeed, we believe these questions to be as difficult as analogous questions that are currently open about the classes of groups already mentioned. We note that there are currently no known sufficient criteria for a group to not be a stack group. This presents a major problem in understanding what groups cannot be stack groups. A similar problem exists for PD-ish groups. To be able to separate the classes, we believe a good place to start is finding suitably strong sufficient conditions for groups to not be in a certain class. However, this too seems to be a rather difficult problem.

Of course *Lehnert's conjecture* (stated in Chapter 1) remains open and of great interest. Moreover, we do not know whether groups such as those proven to be  $co\mathcal{CF}$  in [7] and [22] are stack groups or PD-ish. These present further areas of possible future research.

In Chapter 5 we introduced and studied a new class of groups that we named epiregular groups. We have shown that finite groups and virtually free groups are epiregular. Unlike the classes of groups that we have mentioned previously, we have proven that the class of epiregular groups is distinct from the class of  $co\mathcal{CF}$  groups. We have done this by showing that the Baumslag-Solitar group  $BS(1, 2)$  is epiregular in Theorem 5.3.8. This is sufficient as it is proven in [27] that  $BS(1, 2)$  is not  $co\mathcal{CF}$ . We have also shown that automatic groups are epiregular in Theorem 5.3.6. However, it is known in [13] that  $BS(1, 2)$  is not automatic. Therefore we have shown that the class of epiregular groups is not equal to the class of automatic groups.

In Theorems 5.4.1 and 5.4.10 we have proven that the class of epiregular groups is closed under the following operations:

- passing to finitely indexed overgroups, and
- taking a graph product of groups over a finite graph.

At the time of writing this thesis, we have not been able to find an example of a finitely generated group that is not epiregular. We are currently investigating this. Another possibility of future work is understanding the relationship between any of the classes of  $co\mathcal{CF}$ , co-ETOL, co-indexed, stack, or PD-ish groups and the class of epiregular groups. Further, it is possible to define an analogous class of epi- $\mathcal{C}$  groups where we require the language of representatives for non-trivial elements of the group to be in  $\mathcal{C}$ , for a language class  $\mathcal{C}$ . We wish to understand these classes of groups better and whether or not they are all distinct. Finally we end this chapter with a concrete question that we wish to answer in future work.

**Question 6.0.1.** Are the Higman-Thompson groups  $G_{n,r}$  epiregular?

# Bibliography

- [1] Alfred V. Aho, *Indexed grammars—an extension of context-free grammars*, J. Assoc. Comput. Mach. **15** (1968), 647–671. MR 258547
- [2] ———, *Nested stack automata*, J. Assoc. Comput. Mach. **16** (1969), 383–406. MR 267980
- [3] A. V. Anīsīmov, *The group languages*, Kibernetika (Kiev) (1971), no. 4, 18–24. MR 301981
- [4] Peter R. J. Asveld, *Controlled iteration grammars and full hyper-AFL's*, Information and Control **34** (1977), no. 3, 248–269. MR 445916
- [5] James Belk, Collin Bleak, and Francesco Matucci, *Embedding right-angled Artin groups into Brin-Thompson groups*, Math. Proc. Cambridge Philos. Soc. **169** (2020), no. 2, 225–229. MR 4138920
- [6] James Belk, Collin Bleak, Francesco Matucci, and Matthew C.B. Zaremsky, *Progress around the boone-higman conjecture*, arXiv preprint arXiv:2306.16356 (2023).
- [7] Rose Berns-Zieve, Dana Fry, Johnny Gillings, Hannah Hoganson, and Heather Mathews, *Groups with context-free co-word problem and embeddings into Thompson's group  $V$* , Topological methods in group theory, London Math. Soc. Lecture Note Ser., vol. 451, Cambridge Univ. Press, Cambridge, 2018, pp. 19–37. MR 3889099
- [8] Alex Bishop and Murray Elder, *Bounded automata groups are co-ETOL*, Language and automata theory and applications, Lecture Notes in Comput. Sci., vol. 11417, Springer, Cham, 2019, pp. 82–94. MR 3927687
- [9] Collin Bleak, Francesco Matucci, and Max Neunhöffer, *Embeddings into Thompson's group  $V$  and  $coCF$  groups*, J. Lond. Math. Soc. (2) **94** (2016), no. 2, 583–597. MR 3556455

- [10] Collin Bleak and Olga Salazar-Díaz, *Free products in R. Thompson's group V*, Trans. Amer. Math. Soc. **365** (2013), no. 11, 5967–5997. MR 3091272
- [11] William W. Boone, *Between logic and group theory*, Proceedings of the Second International Conference on the Theory of Groups (Australian Nat. Univ., Canberra, 1973), Lecture Notes in Math., vol. Vol. 372, Springer, Berlin-New York, 1974, pp. 90–102. MR 354880
- [12] William W. Boone and Graham Higman, *An algebraic characterization of groups with soluble word problem*, J. Austral. Math. Soc. **18** (1974), 41–53. MR 357625
- [13] Marcus Brazil, *Growth functions for some nonautomatic baumslag-solitar groups*, Transactions of the American Mathematical Society **342** (1994), no. 1, 137–154.
- [14] José Burillo and Murray Elder, *Metric properties of Baumslag-Solitar groups*, Internat. J. Algebra Comput. **25** (2015), no. 5, 799–811. MR 3384081
- [15] J. W. Cannon, W. J. Floyd, and W. R. Parry, *Introductory notes on Richard Thompson's groups*, Enseign. Math. (2) **42** (1996), no. 3-4, 215–256. MR 1426438
- [16] N. Chomsky, *Three models for the description of language*, IRE Transactions on Information Theory **2(3)** (September 1956), 113–124.
- [17] Laura Ciobanu, Murray Elder, and Michal Ferov, *Applications of L systems to group theory*, Internat. J. Algebra Comput. **28** (2018), no. 2, 309–329. MR 3786421
- [18] Karel Culik, II, *On some families of languages related to developmental systems*, Internat. J. Comput. Math. **4** (1974), 31–42. MR 363009
- [19] M. Dehn, *Über unendliche diskontinuierliche Gruppen*, Math. Ann. **71** (1911), no. 1, 116–144. MR 1511645
- [20] M. J. Dunwoody, *The accessibility of finitely presented groups*, Invent. Math. **81** (1985), no. 3, 449–457. MR 807066
- [21] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston, *Word processing in groups*, Jones and Bartlett Publishers, Boston, MA, 1992. MR 1161694



- [22] Daniel Farley, *Local similarity groups with context-free co-word problem*, Topological methods in group theory, London Math. Soc. Lecture Note Ser., vol. 451, Cambridge Univ. Press, Cambridge, 2018, pp. 67–91. MR 3889102
- [23] Elisabeth Ruth Green, *Graph products of groups*, Ph.D. thesis, University of Leeds, 1990.
- [24] V. S. Guba and M. V. Sapir, *Diagram groups are totally orderable*, J. Pure Appl. Algebra **205** (2006), no. 1, 48–73. MR 2193191
- [25] Susan Hermiller and John Meier, *Algorithms and geometry for graph products of groups*, J. Algebra **171** (1995), no. 1, 230–257. MR 1314099
- [26] Derek F Holt, Sarah Rees, and Claas E Röver, *Groups, languages and automata*, vol. 88, Cambridge University Press, 2017.
- [27] Derek F. Holt, Sarah Rees, Claas E. Röver, and Richard M. Thomas, *Groups with context-free co-word problem*, J. London Math. Soc. (2) **71** (2005), no. 3, 643–657. MR 2132375
- [28] Derek F. Holt and Claas E. Röver, *Groups with indexed co-word problem*, Internat. J. Algebra Comput. **16** (2006), no. 5, 985–1014. MR 2274726
- [29] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley Series in Computer Science, Addison-Wesley Publishing Co., Reading, Mass., 1979. MR 645539
- [30] Lila Kari, Grzegorz Rozenberg, and Arto Salomaa, *L systems*, Handbook of formal languages, Vol. 1, Springer, Berlin, 1997, pp. 253–328. MR 1469997
- [31] J. Lehnert, *Gruppen von quasi-automorphismen*, Ph.D. thesis, Goethe Universität, Frankfurt, 2008.
- [32] Astrid Lindenmayer, *Mathematical models for cellular interactions in development i. filaments with one-sided inputs*, Journal of Theoretical Biology **18** (April 1968), no. 3, 280–299.
- [33] ———, *Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs*, Journal of Theoretical Biology **18** (April 1968), no. 3, 300–315.
- [34] David E. Muller and Paul E. Schupp, *Groups, the theory of ends, and context-free languages*, J. Comput. System Sci. **26** (1983), no. 3, 295–310. MR 710250

- [35] ———, *The theory of ends, pushdown automata, and second-order logic*, Theoret. Comput. Sci. **37** (1985), no. 1, 51–75. MR 796313
- [36] P. S. Novikov, *On the algorithmic unsolvability of the word problem in group theory*, Izdat. Akad. Nauk SSSR, Moscow, 1955, Trudy Mat. Inst. Steklov. no. 44. MR 75197
- [37] Derek J. S. Robinson, *A course in the theory of groups*, Graduate Texts in Mathematics, vol. 80, Springer-Verlag, New York, 1993. MR 1261639
- [38] Grzegorz Rozenberg, *Extension of tabled OL-systems and languages*, Internat. J. Comput. Information Sci. **2** (1973), 311–336. MR 0413614
- [39] G. Rozenberg and A. Salomaa (eds.), *Handbook of formal languages*, Springer, 1997.
- [40] Ville Salo, *Graph and wreath products in topological full groups of full shifts*, arXiv preprint arXiv:2103.06663 (2021).
- [41] Richard J. Thompson, *Notes on three groups of homeomorphisms*, Unpublished but widely circulated handwritten notes (1970), 1–11.
- [42] Jan van Leeuwen, *Variations of a new machine model*, 17th Annual Symposium on Foundations of Computer Science (Houston, Tex., 1976), 1976, pp. 228–235. MR 0474980