Context-sensitive decision problems in groups

Steve Lakin

Thesis submitted for the degree of Doctor of Philosophy at the University of Leicester

September 2002

Abstract

The seemingly distinct areas of group theory, formal language theory and complexity theory interact in an important way when one considers decision problems in groups, such as the question of whether a word in the generators of the group represents the identity or not. In general, these problems are known to be undecidable. Much work has been done on the solvability of these problems in certain groups, however less has been done on the resource bounds needed to solve them, in particular with regard to space considerations. The focus of the work presented here is that of groups with (deterministic) context-sensitive decision problems, that is those that have such problems decidable in (deterministic) linear space. A classification of such groups (similarly to the way that the groups with, for example, regular or context-free word problem, have been classified) seems untenable at present. However, we present a series of interesting results with regard to such groups, with the intention that this will lead to a better understanding of this area. Amongst these results, we emphasise the difficulty of the conjugacy problem by showing that a group may have unsolvable conjugacy problem, even if it has a subgroup of finite index with context-sensitive conjugacy problem. Our main result eliminates the previously-considered possibility that the groups with context-sensitive word problem could be classified as the set of groups which are subgroups of automatic groups, by constructing a group with context-sensitive word problem which is not a subgroup of an automatic group. We also consider a range of other issues in this area, in an attempt to increase the understanding of the sort of groups under consideration.

Acknowledgements

I would like to thank my supervisor, Professor Rick Thomas, for all of his help, guidance, inspiration and advice throughout.

I would also like to thank my examiners, Professor Derek Holt, and Professor Iain Stewart, for many constructive suggestions which improved this thesis.

In addition, I would like to thank many other members of the Department of Mathematics and Computer Science at the University of Leicester (both staff and postgraduate students) for their help, and for creating a friendly and welcoming environment in which to work.

I am extremely grateful to the Engineering and Physical Sciences Research Council (EPSRC) for funding this research.

Finally, I would particularly like to thank my family and friends for their constant encouragement and support.

Contents

Chapter 1

Introduction

1.1 Background and motivation

The idea of a decision problem in a group, for example the word or conjugacy problem, has been around for some time. These problems developed in the early twentieth century, as topology and combinatorial group theory began to interact via the connection of the fundamental group. The first real instance of such a problem being stated was by Tietze in 1908 (see [58]), when he posed the question of deciding whether or not two groups are isomorphic, which became known as the *isomorphism problem.* In 1910, in the paper [16], Dehn first introduced the idea of the *word problem.* Dehn was a topologist and was interested in the question of when two knots are the same, and this led to the question of when two words in a group represent the same element. It was he who first specifically stated the three decision problems that occupy much of this area - the word problem, the conjugacy problem, and the isomorphism problem, all of which have strong topological, as well as group-theoretic, connections.

This interaction between topology and group theory led to the first instances of the word problem being solvable being geometrical examples. For example, Dehn showed that the fundamental groups of closed orientable surfaces have solvable word problem, and in 1926 Artin showed, in [4], that the braid groups also have solvable word problem. Perhaps the most important progress, though, came in 1932 when Magnus showed in [38] that all one-relator groups have solvable word problem. Magnus' proof is long and complicated, but this was one of the first really broad classes of groups to be shown to have solvable word problem.

Around this time the idea of computation was beginning to come to the fore, through the work of Gödel in the famous paper [22], and then through the development of the fundamental model of computation, the *Turing machine*, introduced by Turing in [59]. It had been widely suspected from the beginning that these problems were not solvable in every case, and the development of these computational ideas led to a great deal of work on the solvabilty and unsolvability of these problems. However, it was not until 1955 that Novikov finally showed in [46] that the word problem was unsolvable in general. Soon after, Adian, in [1], and Rabin, in [49], independently showed that the isomorphism problem was unsolvable, and this led Markov in the 1958 paper [40] to show the unsolvability of the homeomorphism problem in topology, returning the area to its original roots.

The issue of solvability and unsolvability is fundamental, but of course there are many more questions to ask about the algorithms to decide a problem. We are not only interested in whether or not a problem is solvable, but also, if it is, on the actual resource bounds of the algorithm, in terms of the time needed and the amount of storage space required. A natural question is to ask what we can say about groups with decision problems lying in some particular complexity class. The first real progress in this area came in

1971, when Anisimov noted in [2] that the class of groups with regular word problem is precisely the class of finite groups. Following from this, in the early 1980s, Muller and Schupp proved, in [44], that the class of groups with context-free word problem is precisely the class of virtually free accessible groups, Dunwoody then, in [17], removed the need for the word 'accessible' by showing that all finitely presented groups are accessible.

Noting that a corollary of a famous result of Higman in [26] is that the groups with recursively enumerable word problem are precisely those groups which can be embedded in finitely presented groups, the final step in the Chomsky hierarchy is to consider the context-sensitive languages, that is those that can be decided in non-deterministic linear space, and this is our particular area of interest. We are still some way off a 'classification' result as we have obtained for other groups in the hierarchy, indeed when asked the question some years ago, of when we will have a classification of groups with context-sensitive word problem, Paul Schupp replied 'not this century'. Perhaps the passing of the new millennium has improved our chances but we are still a long way from achieving this!

Our main area of interest then, is in groups with context-sensitive decision problems, and with the results presented here we hope to increase the understanding of such groups, and hopefully move another rung up the ladder towards a classification.

1.2 Structure of the thesis

The structure of the remainder of the thesis is as follows. We begin in Chapter 2 by making the necessary definitions and introducing some preliminary, basic results to illustrate the area of interest. We then move on, in Chapters

3 and 4, to ask what happens when we try to combine two groups with (deterministic) context-sensitive word problem, or extend a single such group, in some way. This culminates in us showing that the property of a group having (deterministic) context-sensitive word problem is closed under taking extensions of finite index, but that this situation is not true for the conjugacy problem.

In Chapter 5, we consider embeddings of groups with context-sensitive word problem, and we show that any group with (deterministic) context-sensitive word problem can be embedded in a two-generator group with (deterministic) context-sensitive word problem, and we also make a conjecture regarding embeddings. We then move on to perhaps the main focus of the thesis, in Chapter 6. Here, we show that the class of groups with context-sensitive word problem cannot be classified as the set of groups which are subgroups of automatic groups, by producing a suitable group which cannot be a subgroup of an automatic group. This construction also allows us to produce groups without a context-sensitive word problem.

Chapter 7 is devoted to a brief study of other decision problems, in particular the reduced and irreducible word problems, where we show that the property of a group having context-sensitive reduced, or irreducible, word problem is equivalent to having context-sensitive word problem.

Finally, before we give our conclusions and final comments, in Chapter 8 we provide a wide series of groups with context-sensitive decision problems.

Chapter 2

Preliminary definitions and results

We give the necessary preliminaries in great detail, since we wish to make this thesis accessible to readers interested in both formal language theory and group theory.

2.1 Definitions

2.1.1 Formal language theory

We begin with some ideas from computation and formal language theory. An *alphabet* X is simply a finite set of symbols. A *word* over X is a string of symbols in X, and we denote by X^* the set of all words over X, including the *empty word* which will we denote by λ. X^+ denotes the set of all non-empty words over X. The *length* of a word w over X is the number of symbols it contains, and we use the notation $l(w)$ to denote the length of w. A *language* L is a subset of X^* and a *class* (or *family*) of languages $\mathcal{C}$ is simply a collection of languages.

We can combine words together by *concatenation*. Hence, by the word $\alpha\beta$, we mean the word obtained by writing the symbols of α followed by the symbols of β.

Suppose we are given a language L over an alphabet X. A set of *rewrite rules* R for L is a set of rules of the form $\alpha \to \beta$ for words α, β over X. The basic idea is that, given a word $w = w_1\alpha w_2$ in L, the rewrite rule $\alpha \to \beta$ allows us to replace the subword α by β, 'reducing' w to the word $w' = w_1\beta w_2$. We write $u \to v$ to mean that there is a single application of a rule in R reducing u to v, and we write $u \xrightarrow{*} v$ if there is a sequence of words $u = u_1, u_2, \ldots.., u_m = v$ where $u_i \to u_{i+1}$ for all $1 \leq i \leq m-1$.

A word may reduce to some *normal form* where no more reductions are possible. This normal form may depend on the choice of rewrites at each stage (there may be more than one possibility) and hence may not be unique, but if every normal form is unique then we say that the system is *confluent*. A rewriting system is said to be *strongly normalising* if there are no infinite sequences of reductions. Hence if we have a confluent strongly-normalising system, then every word has a unique normal form.

Our model of computation throughout will be the standard multitape Turing machine as defined in [31]. Formally, a Turing machine is a quintuple

$$M = (K, \Sigma, \delta, s, F)$$

where K is a finite set of *states*, $s \in K$ is the *initial state* and $F \subseteq K$ is the set of *final* (or *halt*) *states*, some of which are defined as *accept states* and the others as *reject* (or *non-accept*) *states*. Σ is a finite set of symbols called the *alphabet* of M, including symbols $\sqcup$ (the *blank*) and $\Box$ (the *start symbol*), and

$$\delta : K \times \Sigma^n \to K \times \Sigma^n \times \{\leftarrow, \to, -\}$$

(for some n) is the *transition function.* We can, for convenience, consider the states to be of the form $S_1 \times ..... \times S_m$ where the S_i are the *state components.* This is only a notational convenience, but can be useful in circumstances where our transition we wish to define is dependent on some particular aspect of our computation so far. For example, we will use this in Section 2.4.2.

The idea, of course, is that we have a read-only *input tape* and finitely many *worktapes*, each with a cursor to indicate the cell on each tape that we are currently scanning. The point of the start symbol is that it indicates a point on each tape which we cannot move past, and we start at the cell immediately following it, which effectively gives us one-way infinite tapes. At each step of the computation, we read the symbol currently being scanned on the input tape. Depending on this symbol, and our current state, we may move our cursors left or right, or stay where we are, on each tape. We may also rewrite the symbol now being scanned on each worktape (we do not write on the input tape), and we may change state. We terminate when we reach an accept or reject state, of course we may never terminate.

A non-deterministic Turing machine is defined in exactly the same way, but at each step we have a finite *choice* of next moves, hence the non-determinism.

We are interested in finding algorithms to solve questions about our input word. A language L is *decided* by a Turing machine if there exists an algorithm which, given any string in L, terminates in an accept state and, given a string not in L, terminates in a reject state. If a Turing machine M decides a language we may denote the language by $L(M)$ and we say that L is *recursive.* If there is an algorithm which terminates in an accept state for a string in L, but runs for ever for a string not in L, then L is said to be *recursively enumerable.* Clearly a recursive language is recursively enumerable.

A *(finite state) automaton* is a Turing machine with a read-only input tape on which we can only move right, and no worktapes. A language recognised by a finite state automaton is said to be *regular*.

We can formally describe the operation of a Turing machine during its computation. A *configuration* (or *instantaneous description*) of a k-tape Turing machine (that is, a Turing machine with an input tape and $k-1$ worktapes) consists of a $(2k+1)$-tuple $(q, u_1, v_1, ..., u_k, v_k)$ where q is the current state, and we currently have the word $u_i v_i$ written on the i'th tape, with our cursor pointing at the last symbol of u_i. Note that u_i and v_i may contain blanks.

We can describe Turing machines using the *standard description.* Suppose we are given a Turing machine $M = (K, \Sigma, \delta, s, F)$. We assume that the alphabet and states are denoted by integers, where $\Sigma = \{1, ..., |\Sigma|\}$ and $K = \{|\Sigma|+1, ..., |\Sigma|+|K|\}$. We may then further assume that the numbers $|\Sigma|+|K|+1, ...|\Sigma|+|K|+6$ encode the special symbols of the Turing machine, namely $\rightarrow, \leftarrow, -, h$, 'yes' and 'no', which are used to describe which way to move on a tape, whether to halt, and whether a final state is an accept or reject state. We encode these integers as binary, making all the symbols the same length by introducing leading zeros if necessary to each binary encoding.

This idea allows us to write a description of a Turing machine, suitable for use in some other machine. The basic idea is that we begin by writing $|\Sigma|$ and $|K|$ in binary (separated by some dividing symbol). We then write a description of δ, defining the transitions for each possible state and set of symbols on each tape read (again we use dividing symbols to separate the different transitions). To simulate M, it is simply a matter of scrolling through this description searching for the required transition. See [47] for full details of this simulation.

We are interested not only in whether or not a language is decidable, but, if it is, also in the constraints on time (the number of steps made until we reach a final state) and space (the length of worktapes used in the computation) required to decide it. Usually, of course, these will be a function of the length of the input word n. So, for example, a language is decided in time $O(n)$ if, given any word of length n, we can determine in $O(n)$ steps whether or not it lies in L.

We are particularly interested here in the *context-sensitive* languages, which are defined as those languages decided by a nondeterministic Turing machine with space bound $O(n)$. We have an obvious definition of a *deterministic context-sensitive* language where the Turing machine accepting the language is deterministic. It is unknown whether these two classes of languages coincide, but note that obviously the deterministic context-sensitive languages are contained inside the context-sensitive languages.

Let us note a simple, but extremely useful, lemma with regard to context-sensitive languages.

Lemma 2.1.1 *Suppose we have a language L over a finite alphabet A. Then there exists a context-sensitive algorithm to decide L if and only if (for an input word of length n) the number of occurrences of any symbol of A, in any word in the computation, is $O(n)$.*

Proof Let $A = \{a_1, ...a_{|A|}\}$. If our algorithm is context-sensitive, then there is a linear bound on the length of any word in the computation. Obviously this is also a bound on the number of occurrences of a symbol that could appear in any word. Conversely, suppose that for an input word of length n, the number of occurrences of symbol a_i in any word in the computation is bounded by $k_i n$. Then the maximum total length of a word is bounded by

$(k_1 + ... + k_{|A|})n$, and hence we still have a linear bound for the length of any word, and we have a context-sensitive procedure. $\square$

2.1.2 Groups and generators

A *semigroup* S is simply a set closed under an associative binary operation $\circ$, normally we abbreviate $s \circ t$ as st. A *monoid* M is a semigroup with a unique element 1, such that $1m = m1 = m$ for all $m \in M$, and a *group* G is a monoid with the additional property that for all $g \in G$, there exists an 'inverse' element $g^{-1} \in G$ with $gg^{-1} = g^{-1}g = 1$. Clearly the inverse of g^{-1} is g.

Let G be a group and X a set of symbols, and suppose ϕ is a map from X to G. ϕ extends naturally to a homomorphism $\overline{\phi} : X^* \to G$ where we have

$$\overline{\phi}(x_1.....x_n) = \phi(x_1).....\phi(x_n).$$

If this homomorphism is surjective then we say X *generates* G as a monoid and call X a *monoid generating set* for G. We can define X^{-1} as a formal set of symbols $\{x^{-1} : x \in X\}$ with cardinality $|X|$, and define $\phi(x^{-1}) = \phi(x)^{-1}$. Then if $X \cup X^{-1}$ is a monoid generating set for G, we call X a *group generating set* for G and say X generates G as a group. Of course, a monoid generating set is a group generating set, and any group generating set can be made into a monoid generating set simply by adding the inverses, and so we can generally blur the distinction between them, and simply refer to a 'generating set'. We shall usually assume we have a monoid generating set, since this is generally notationally convenient, however we will be careful to specify if we have a group or monoid generating set, if there is any possibility of confusion. We use the notation $G = \langle X \rangle$ to mean that X is a generating set for G.

Effectively, of course, what we are doing is to associate an element of G to each element of X - we can identify $x \in X$ with $\phi(x) \in G$. We are therefore simply considering X as a set of elements of G, and we will generally do this without comment.

We may define a *presentation* of a group G by $G = \langle X : R \rangle$, where X is a set of generators for G and R is a set of *relations* which define the structure of our group, where each relation is of the form $u_i = v_i$ for words u_i, v_i over X. Here we introduce the notation $u = v$ to mean that u and v represent the same element of the group, and use $u \equiv v$ to mean that u and v are identical as words. As an example, we have the group

$$B(1,2) = \langle a, b : b^{-1}ab = a^2 \rangle,$$

so we have generators a and b, and we impose the constraint that $b^{-1}ab = a^2$ in our group. This group is called a *Baumslag-Solitar* group. The groups given by $G = \langle X : \emptyset \rangle$ are called the *free groups.*

Note that in a group presentation, we simply assume that the inverses are present, and so our words in our relations are in fact words over $X \cup X^{-1}$.

Given a relation $u = v$, we could write this as $uv^{-1} = 1$, and hence we may also consider our presentation to be given in the form $G = \langle X : R \rangle$ where R now represents a set of *relators*, that is words over X equivalent to the identity in G. Generally, however, we will consider relations rather than relators (one reason being that this concept generalises naturally to semigroups, where we may not have an identity element).

A more group-theoretic way of looking at all of this, is that a presentation of a group $G = \langle X : R \rangle$ (where R is a set of relators) defines G as the quotient of the free group F, generated by X, by the normal closure of R, so we may write $G = F/R$.

We are concerned here only with groups where X is finite, in which case we say that G is *finitely generated*, and we shall assume without comment that all of our groups are finitely generated. If R is also finite then we say that G is *finitely presented*.

Suppose we have a group with group generating set $X = \{x_1, ...x_m\}$. A word $u = u_1.....u_r$ over $X \cup X^{-1}$ is said to be *reduced* if it does not contain a substring $x_i x_i{}^{-1}$ or $x_i{}^{-1} x_i$ for any i, and we say that u is *cyclically reduced* if, in addition, u_r is not equal to the inverse of u_1. We formally define the inverse of the word u to be the word $u^{-1} \equiv u_r{}^{-1}.....u_1{}^{-1}$.

2.1.3 Decision problems

Suppose X is a monoid generating set for a group G. Let $\phi : X^* \to G$ be the natural homomorphism. The *word problem* of G with respect to X is formally defined as

$$W_X(G) = \phi^{-1}(1),$$

that is the set of words over X^* equivalent to the identity in G. Equivalently, we can phrase the word problem as the question of whether a given word represents the identity, which is the approach we will take.

In a group G, two elements $g_1, g_2 \in G$ are defined to be *conjugate* if there exists $h \in G$ with $h^{-1} g_1 h = g_2$, in which case we write $g_1 \sim g_2$. If we wish to stress the group under consideration then we write $g_1 {\sim_G} g_2$. The idea of the conjugacy problem is that it is the question of whether two given words are conjugate in G, or equivalently the set of all conjugate pairs of words. The situation with regard to precisely defining the conjugacy problem is somewhat more difficult, however, since we are now considering pairs or words and we need to ensure that we do have monoid generating sets for pairs of words over our group G. Probably the easiest way to achieve this is via the following construction. Suppose X generates G. We have the natural surjective homomorphism $\phi : X^* \to G$ which extends naturally to a surjective homomorphism $\phi_1 : X^* \times X^* \to G \times G$ (where we define $\phi_1(u_1, u_2) = (\phi(u_1), \phi(u_2))$ for $u_1, u_2 \in X^*$). Then $X^* \times X^*$ is naturally generated by the set of pairs

$$X_1 = \{(x, \lambda) : x \in X\} \cup \{(\lambda, x) : x \in X\}.$$

with the obvious homomorphism $\rho : {X_1}^* \to X^* \times X^*$, where

$$\rho((u_1, v_1)....(u_m, v_m)) = (u_1....u_m, v_1....v_m).$$

This gives us a monoid generating set for $G \times G$, and we can formally define the *conjugacy problem* of G with respect to X to be

$$C_X(G) = \rho^{-1}\phi_1^{-1}(U)$$

where

$$U = \{(g_1, g_2) \in G \times G : g_1 \sim g_2\}.$$

Having defined this formally, we shall, from now on, simply consider the conjugacy problem as the question of whether or not two words of G are conjugate.

Note that if we have an algorithm to solve the conjugacy problem (that is, we can determine whether or not a pair of words are conjugate in G), then this algorithm also allows us to solve the word problem, since we simply test to see if a given word u is conjugate to 1 (if so, then there exists v such that $v^{-1}uv = 1$, and hence $u = vv^{-1} = 1$). Hence the conjugacy problem is in some sense 'at least as hard as' the word problem in any group. In fact it is strictly harder, since there are groups with solvable word problem but unsolvable conjugacy problem. Note that we can also phrase the conjugacy problem as the question of whether, given two words u_1 and u_2, there exists v such that $u_1 v {u_2}^{-1} v^{-1} = 1$, which relates naturally to the word problem.

As a comment here, let us note that for any group G, any conjugate of a word u lying in $W_X(G)$ also lies in $W_X(G)$ (since if $u = 1$ then $v^{-1}uv = v^{-1}v = 1$) and similarly for the conjugacy problem. This allows us to conjugate a given word if we wish, to produce a word in some simpler form.

Note also, that in a group the word problem is essentially equivalent to the alternative question of asking whether or not two words are equivalent in the group, since we have $u = v$ if and only if $uv^{-1} = 1$. This is of particular relevance with regard to semigroups, when we may not have an identity element present, and in semigroups we define the word problem to be the set of pairs of words (u, v) representing equivalent elements of the group, or the question of whether two words represent equivalent elements. Similarly, the conjugacy problem in semigroups can be defined as the set of words (u, v) such that there exists a word w with $uw = wv$.

Another natural decision problem to consider is the *generalised word problem* which we shall denote by $GW_X(G)$. Essentially, this is the question of whether, given a group G and some subgroup H, we can determine whether or not a word over the generators of G lies in H. The formal definition is that we take a group G with generating set X. Then the generalised word problem with respect to X is the set of all tuples of words $(u_1, ..., u_m)$ where u_1 lies in the subgroup generated by $u_2, ..., u_m$. Again, we can consider this as the question of whether a given word lies in the subgroup generated by a finite set of given words.

However, we shall usually be interested in this thesis in the interpretation of the generalised word problem as deciding membership of a specific subgroup. The *generalised word problem with respect to* H, that is the question of whether a word lies in some specified subgroup H, will be denoted by $GW_X(G, H)$.

We should again note that the generalised word problem is intrinsically harder than the word problem, since the word problem is the particular case of the generalised word problem where our subgroup represents the trivial subgroup, and there exist groups with solvable word problem but unsolvable generalised word problem.

We may also wish to consider what we shall define as the *effective generalised word problem*, where not only do we wish to verify membership of the given subgroup, but also determine which particular element of the subgroup the word represents, and produce a representative in the generators of our subgroup for the word in question, if it does indeed lie in the subgroup.

The generalised word problem with respect to a subgroup is a particular example of what we shall call the *occurrence problem*[1]. Suppose S is merely some subset of a group G, where G is generated by a finite set X. Then along entirely similar lines, we can consider the question of whether or not a given word in the generators of G lies in S, which is the occurrence problem of G with respect to S and X. We will denote this by $O_X(G, S)$.

These are the decision problems we are particularly interested in, since we consider them as the most fundamental questions to ask about a group. Of course, there are other interesting decision problems one could ask. For example, we could consider the *isomorphism problem* (the question of whether two presentations present the same group), the *power problem* (determining whether or not a given word in a group is a power of another word) and the *order problem* (determining the order of a given word), however clearly there are many possible decision problems we could ask about a group and it is

[1]Notation sometimes varies in the literature, and some authors use 'occurrence problem' and 'generalised word problem' interchangeably. However we feel the notation we use is the most appropriate for our purpose.

$$\begin{tikzcd} X^* \arrow[dr, "\phi"] \arrow[d, "\theta"'] & \\ Y^* \arrow[r, "\psi"'] & M \end{tikzcd}$$

Figure 2.1: The homomorphisms in Lemma 2.2.1

impossible to investigate them all.

2.2 Independence of generating sets

A great deal of work has been done on the solvability of these decision problems in groups, that is whether there exist algorithms to decide the problems. Our area of interest here is to investigate the resource bounds required to solve such problems, if they are indeed solvable. It would be a problem if the property of a decision problem (for example the word problem) lying in a particular class of languages (for example the context-sensitive languages) depended on the choice of generating set, since many of the proofs we will provide depend on us choosing a 'convenient' generating set for the group in question. Fortunately we have the following lemma and subsequent corollaries.

Lemma 2.2.1 *Suppose that X and Y are alphabets, M is a monoid, and $\phi : X^* \to M$ and $\psi : Y^* \to M$ are surjective homomorphisms. Then there exists a homomorphism $\theta : X^* \to Y^*$ such that $\phi = \psi\theta$ (see Figure 2.1).*

Proof We follow the proof as in [25]. For each $x \in X$ we have $\phi(x) = m$ for some $m \in M$. Now, ψ is surjective and hence $\psi^{-1}(m)$ is non-empty. So we may take some $y \in \psi^{-1}(m)$, and set $\theta(x) = y$. This gives us a well-defined map from X to Y which we can extend in the natural fashion to a homomorphism from X^* to Y^*. Clearly we have $\phi = \psi\theta$ as required. $\square$

With this lemma behind us, we can prove the results we require. Firstly we consider the word problem.

Corollary 2.2.2 *Let $G = \langle X \rangle = \langle Y \rangle$, with X and Y finite, and suppose $\mathcal{C}$ is a class of languages closed under inverse homomorphism. Then*

$$W_X(G) \in \mathcal{C} \Leftrightarrow W_Y(G) \in \mathcal{C}$$

where $W_X(G)$ denotes the word problem of G with respect to X, and similarly for $W_Y(G)$.

Proof There exist surjective homomorphisms $\phi : X^* \to G$ and $\psi : Y^* \to G$ since X and Y generate G, and hence there exists a homomorphism θ such that $\phi = \psi\theta$ as in Lemma 2.2.1. Suppose $W_Y(G) \in \mathcal{C}$. Then

$$W_X(G) = \phi^{-1}(1) = \theta^{-1}\psi^{-1}(1) = \theta^{-1}(W_Y(G)) \in \mathcal{C}$$

since $W_Y(G) \in \mathcal{C}$, and $\mathcal{C}$ is closed under inverse homomorphism. Hence $W_Y(G) \in \mathcal{C} \Rightarrow W_X(G) \in \mathcal{C}$. Repeating the above argument with X and Y interchanged, we have the reverse implication, and hence

$$W_Y(G) \in \mathcal{C} \Leftrightarrow W_X(G) \in \mathcal{C}$$

as required. $\square$

Corollary 2.2.3 *Let $G = \langle X \rangle = \langle Y \rangle$ and suppose $\mathcal{C}$ is a class of languages closed under inverse homomorphism. Then*

$$C_X(G) \in \mathcal{C} \Leftrightarrow C_Y(G) \in \mathcal{C}$$

where $C_X(G)$ denotes the conjugacy problem of G with respect to X, and similarly for $C_Y(G)$.

$$\begin{tikzcd}
{X_1}^* \arrow[r, "\rho"] \arrow[d, "\theta"'] & X^* \times X^* \arrow[dr, "\phi"] & \\
 & & G \times G \\
{Y_1}^* \arrow[r, "\tau"'] & Y^* \times Y^* \arrow[ur, "\psi"'] &
\end{tikzcd}$$

Figure 2.2: The homomorphisms in Corollary 2.2.3

Proof We have surjective homomorphisms

$$\phi : X^* \times X^* \to G \times G$$

$$\psi : Y^* \times Y^* \to G \times G$$

and we define ${X_1}^*$ and ${Y_1}^*$ as in the discussion of the definition of the conjugacy problem in Section 2.1.3. Let

$$\rho : {X_1}^* \to X^* \times X^*$$

$$\tau : {Y_1}^* \to Y^* \times Y^*$$

be the natural homomorphisms. Then the maps

$$\phi\rho : {X_1}^* \to G \times G$$

$$\psi\tau : {Y_1}^* \to G \times G$$

are both surjective homomorphisms, as the composition of surjective homomorphisms. Then, by Lemma 2.2.1, there exists a homomorphism $\theta : {X_1}^* \to {Y_1}^*$ such that $\phi\rho = \psi\tau\theta$ and we have the situation illustrated in Figure 2.2.

Suppose $C_Y(G) \in \mathcal{C}$. Then by letting

$$U = \{(g_1, g_2) \in G \times G : g_1 \sim g_2\}$$

we have that

$$C_X(G) = \rho^{-1}\phi^{-1}(U) = \theta^{-1}\tau^{-1}\psi^{-1}(U) = \theta^{-1}(C_Y(G)) \in \mathcal{C}$$

since $C_Y(G) \in \mathcal{C}$ and $\mathcal{C}$ is closed under inverse homomorphism. Hence $C_Y(G) \in \mathcal{C} \Rightarrow C_X(G) \in \mathcal{C}$. Repeating the argument with X and Y interchanged, we have the reverse implication, and hence

$$C_Y(G) \in \mathcal{C} \Leftrightarrow C_X(G) \in \mathcal{C}$$

as required. $\square$

It is worth noting in Corollary 2.2.2, that we could replace '1' by any element or subset of G, and the proof would follow entirely similarly. Thus in particular, we have the following corollaries.

Corollary 2.2.4 *Let $G = \langle X \rangle = \langle Y \rangle$ and suppose $\mathcal{C}$ is a class of languages closed under inverse homomorphism. Let H be a subgroup of G. Then*

$$GW_X(G,H) \in \mathcal{C} \Leftrightarrow GW_Y(G,H) \in \mathcal{C}$$

where $GW_X(G,H)$ denotes the generalised word problem of G with respect to H and X, and similarly for $GW_Y(G,H)$.

Corollary 2.2.5 *Let $G = \langle X \rangle = \langle Y \rangle$ and suppose $\mathcal{C}$ is a class of languages closed under inverse homomorphism. Let S be a subset of G. Then*

$$O_X(G,S) \in \mathcal{C} \Leftrightarrow O_Y(G,S) \in \mathcal{C}$$

where $O_X(G,S)$ denotes the occurrence problem of G with respect to S and X, and similarly for $O_Y(G,S)$.

Of course, the first of these follows anyway from the second, but we prefer to keep the problems separate.

Almost all the classes of languages we would wish to consider (in particular the (deterministic) context-sensitive languages) have the property of being closed under inverse homomorphism, and hence these are crucial observations. We need not worry about the generating set we choose, if we can show that the problem under consideration (for example the word problem) lies in a class of languages $\mathcal{C}$ for some generating set, then it does for all generating sets, so we are at liberty to choose a particularly convenient generating set. We will exploit this fact frequently without further mention.

In particular, we may omit reference to the particular generating set in question and simply talk of the word problem $W(G)$, the conjugacy problem $C(G)$, and so on.

We are often interested in subgroups of a given group. The following lemma is extremely useful with regard to the word problem.

Lemma 2.2.6 *Let $\mathcal{C}$ be a class of languages closed under inverse homomorphism, and closed under intersection with regular languages. Let G and H be finitely-generated groups, and suppose $H \leq G$. If $W(G) \in \mathcal{C}$, then $W(H) \in \mathcal{C}$.*

Proof Suppose $W(G) \in \mathcal{C}$. We are at liberty to choose a convenient generating set for G by Corollary 2.2.2. So let us choose a generating set

$$X = \{h_1, ..., h_m, g_1, ..., g_n\}$$

where $Y = \{h_1, ..., h_m\}$ is a generating set for the subgroup H. It is obvious that $W_Y(H) = W_X(G) \cap Y^*$, since H is a subgroup of G. Since $\mathcal{C}$ is closed under intersection with regular languages, and Y^* is obviously regular, then $W(H) \in \mathcal{C}$. $\square$

This lemma can be a useful tool in showing that certain classes of groups have word problem lying in a particular class of languages, by exhibiting them as a subgroup of a group in that particular class.

In particular, we note that a finitely-generated subgroup of a group with (deterministic) context-sensitive word problem also has (deterministic) context-sensitive word problem, since the (deterministic) context-sensitive languages satisfy this condition.

Unfortunately, this result does not carry over to the situation with regard to the conjugacy problem. We cannot deduce, in the same way, that $C_Y(H) = C_X(G) \cap Y^*$, since words of H may be conjugate in the whole of G, but not in H. In fact, it is known that there are groups with solvable conjugacy problem (and hence the conjugacy problem lies in the class of recursive languages, which is closed under inverse homomorphism and intersection with regular languages), which have subgroups with unsolvable conjugacy problem (see for example [15]). This illustrates the important point that the conjugacy problem is intrinsically 'harder' than the word problem, and we will see further illustrations of this later.

2.3 Some comments on complexity classes

We are interested particularly in the context-sensitive languages. Let us make some comments on the position of these languages in the Chomsky, and space-time, hierachies and make some useful observations.

2.3.1 The Chomsky hierarchy

We have considered so far the context-sensitive languages to be those decidable in linear space, and we have taken this as our definition of context-sensitive. This is the most natural approach to take in terms of solvability and computation, and will be the approach we take throughout this thesis. However, it is possible to view these languages from an alternative viewpoint.

The material here is all well-known and standard, and covered in a myriad of textbooks - see for example [31] or [34].

Formally, a *phrase-structured grammar* is a quadtuple

$$G = (N, \Sigma, S, P)$$

where N is a finite set of *nonterminals*, Σ is a finite set of *terminals* with $N \cap \Sigma = \emptyset$, $S \in N$ is the *start symbol*, and

$$P \subset (N \cup \Sigma)^{+} \times (N \cup \Sigma)^{*}$$

is a finite set of *productions* (so P can be considered as a set of rewriting rules $\alpha \to \beta$).

The idea is that the language generated by a phrase-structured grammar is the language of words that can be obtained, starting from the start symbol S, by applying a sequence of rewrites in P, and ending with a string of terminals.

Suppose we place some restrictions on the productions in P. A *regular grammar* is a phrase-structure grammar where every production is of the form $A \to w$ or $A \to wB$, where $A, B \in N$ and $w \in \Sigma^{*}$. A *context-free grammar* is a phrase-structure grammar where every production is of the form $A \to \beta$ for $A \in N$, so we have no restriction on the right-hand side. Finally, a *context-sensitive grammar* is a phrase-structure grammar where every production is of the form $\alpha \to \beta$ with $|\alpha| \leq |\beta|$. We should note that, according to this definition, the empty word cannot lie in any language generated by a context-sensitive grammar. However, it is easily shown that given any language L generated by a context-sensitive grammar, L can also be generated by a context-sensitive grammar where the start symbol S does not appear on the right hand side of any production. Then we can allow ourselves to add the exceptional production $S \to \lambda$, and still consider this as a context-sensitive grammar, and this is what is usually done. Hence we assume that the empty word can lie in a language generated by a context-sensitive grammar.

We note in passing at this point, that the terminology 'context-sensitive' comes from this approach via grammars. In the context-free grammars, our productions have only a single non-terminal on the left-hand side, and so we can always replace this symbol regardless of the 'context' in which it appears in a word. Whereas, it is well-known that there exists a normal form of the productions in a context-sensitive grammar, where each production is of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 B \alpha_2$ for $A, B \in N$. Hence A can only be replaced in the 'context' of α_1 and α_2, and thus this grammar is 'context-sensitive'.

The correspondence between the languages generated by these grammars, and computational machines, is well-known. We can define regular languages either as those languages generated by regular grammars, or (as we defined earlier) as those accepted via finite-state automata. Similarly, context-free languages can either be defined via context-free grammars, or via *pushdown automata*, which are effectively non-deterministic finite-state automata with an additional 'stack' - we shall not go into details here as it does not serve our purpose. The context-sensitive languages are those generated by context-sensitive grammars, or those recognised by a non-deterministic Turing machine working in linear space, as we have already defined them. Finally, those languages generated by phrase-structure grammars in general are the recursively enumerable languages.

It is easily shown that the recursively enumerable languages strictly contain the context-sensitive languages, which strictly contain the context-free languages, which strictly contain the regular languages. Thus, we have a hierarchy of languages, each of which can be expressed via grammars or computational machines. This hierarchy is known as the *Chomsky hierarchy*.

One of the reasons we have discussed this, is because the groups with word problem lying in each level of the Chomsky hierarchy have been classified, with the exception of the context-sensitive languages. As we noted in the introduction, the groups with regular word problem are precisely the finite groups (see [2]), those groups with context-free word problem are precisely the virtually free groups (see [44]), and those with recursively enumerable word problem are precisely those that can be embedded into finitely presented groups (see [26]). Thus we are left only to classify the groups with context-sensitive word problem in the Chomsky hierarchy.

2.3.2 The space-time hierarchy

Let us now return to considering the context-sensitive languages as those solvable in linear space, and see how they fit into the general space-time hierarchy.

To begin with, we define what we mean by a *proper complexity function*. Suppose we have a function f from the non-negative integers to the non-negative integers which is non-decreasing, that is $f(n+1) \geq f(n)$ for all n. Suppose further that there is a multitape Turing machine T such that, for any integer n and any input word u of length n, T writes the string $\sqcap^{f(n)}$ on the last worktape, halting after $O(n + f(n))$ steps and using $O(f(n))$ space. Then we call f a proper complexity function.

The class of proper complexity functions is extremely wide and includes all the non-decreasing functions which one would class as 'reasonable' for the study of complexity, that is it only excludes functions which, in some sense, can be said to be 'artificial'. Given a proper complexity function $f(n)$, we can then consider the languages that are decidable within the time or space bounds specified by $f(n)$. We use the standard notation $\mathbf{DTIME}(f(n))$, $\mathbf{NTIME}(f(n))$, $\mathbf{DSPACE}(f(n))$ and $\mathbf{NSPACE}(f(n))$ for the deterministic and non-deterministic time and space complexity classes respectively. So, for example, $\mathbf{DTIME}(f(n))$ contains precisely those languages which are decidable in deterministic time $f(n)$. It is well known that for any 'reasonable' complexity function $f(n)$, we have that $\mathbf{DTIME}(f(n))$=$\mathbf{DTIME}(cf(n))$ for any constant $c > 0$, and similarly for the other classes, and so we generally disregard constants and use the usual $O(f(n))$ notation. Hence, in particular, we have that the context-sensitive languages are denoted by $\mathbf{NSPACE}(n)$.

At first thought, the fact that an algorithm operates in certain space bounds does not necessarily say much about the time bounds in which it operates, since we may continually re-use space despite requiring a huge amount of steps. However, there is plenty that we can actually say.

First of all let us make the obvious comment that, since any deterministic algorithm is trivially non-deterministic with only one choice at each step, $\mathbf{DTIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$ and $\mathbf{DSPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$. It is also obvious that we have $\mathbf{DTIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n))$ and $\mathbf{NTIME}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ since we only write at most one symbol on each worktape at each step, so certainly if we have $O(f(n))$ steps then the length of our worktapes are $O(f(n))$.

In fact we have the following result, as proved in [47].

Theorem 2.3.1 *For any proper complexity function $f(n)$ and number $k > 1$, we have that*

$$\boldsymbol{DTIME}(f(n)) \subseteq \boldsymbol{NTIME}(f(n)) \subseteq \boldsymbol{DSPACE}(f(n)) \subseteq$$
$$\boldsymbol{NSPACE}(f(n)) \subseteq \boldsymbol{DTIME}(k^{\log n+f(n)}).$$

In particular, with regard to the context-sensitive languages, note that any language which can be decided in (non-deterministic) linear time can be decided by a (deterministic) context-sensitive algorithm, and hence if we were able to show a group had decision problem solvable in non-deterministic linear time, then this would immediately imply that it had a deterministic context-sensitive decision problem.

Similarly, if we have a context-sensitive algorithm, then we know that we can find a deterministic algorithm that will work in time $O(k^{\log n+n})$ for any number $k>1$.

In actual fact, we have the following further result courtesy of [30].

Theorem 2.3.2 *For any proper complexity function $f(n)$, we have that*

$$\boldsymbol{DTIME}(f(n)\log f(n)) \subseteq \boldsymbol{DSPACE}(f(n)).$$

In particular, when applied to the context-sensitive languages, this gives us the immediate corollary that $\mathbf{DTIME}(n\log n) \subseteq \mathbf{DSPACE}(n)$. And hence if we can show that a group has decision problem solvable with a deterministic algorithm in $O(n\log n)$ steps, then it must have deterministic context-sensitive decision problem.

2.3.3 Determinism versus non-determinism?

Almost everything that we will discuss in the thesis with regard to decision problems allows us to replace 'context-sensitive' by 'deterministic context-sensitive', that is all of our algorithms are deterministic. The question of whether 'deterministic context-sensitive' is equivalent to 'context-sensitive' in general is still unknown.

Open Question 2.3.3 *Does $\boldsymbol{DSPACE}(n) = \boldsymbol{NSPACE}(n)$? That is, is deterministic linear space equivalent to non-deterministic linear space?*

This is still unknown, but note that we do know that non-deterministic polynomial space is equivalent to deterministic polynomial space, see for example [31].

It would be extremely interesting to see if, at least in the restricted class of (say) word problems in groups, we do have the situation that a group with context-sensitive word problem also has deterministic context-sensitive word problem, even if this is not true for context-sensitive languages in general. The motivation behind this is that groups with context-free word problem also have deterministic context-free word problem as proved in [45].

Open Question 2.3.4 *Does every group with context-sensitive word problem also have deterministic context-sensitive word problem?*

Of course, a positive solution to the $\mathbf{DSPACE}(n) \stackrel{?}{=} \mathbf{NSPACE}(n)$ question would answer this question in the affirmative.

2.3.4 A brief note on groups without context-sensitive decision problems

In general, it is extremely difficult to prove that a group does *not* have a context-sensitive decision problem (excepting the obvious cases where the decision problem is known to be unsolvable). Of course, it is not enough to exhibit an algorithm that requires more than linear space, since there is no reason why some other algorithm, hitherto unthought of, would not solve the problem. Occasionally one comes across problems where one can give a lower bound on the amount of space required, of course if we found a group with decision problem where, for example, there was a lower bound of $O(n \log n)$ on the space required then we could obviously immediately say that this cannot be context-sensitive.

Sometimes it may be easier to consider the time required for a computation. Note, as we commented before, that the time required may be much greater than a linear bound, but this does not preclude the calculation from being context-sensitive, since we may require very little storage space and can continually write over the words currently being stored. We do have the following interesting result though.

Lemma 2.3.5 *Suppose that there is no algorithm to decide a language L in deterministic time $O(c^n)$ for some $c > 1$. Then L is not context-sensitive.*

Proof This follows from the final inclusion in Theorem 2.3.1. We shall not prove this inclusion, but essentially the idea of the proof is to note that given any algorithm, we have a similar algorithm where every configuration occurs at most once - a deterministic algorithm can never return to a configuration or it would loop for ever, and if we have a non-deterministic algorithm then we simply ignore the choice that leads us to loop. We can then count the total possible number of configurations to give the required result. In this particular case, $f(n) = n$, and so we have $\mathbf{NSPACE}(n) \subseteq \mathbf{DTIME}(k^{\log n + n})$ for any constant $k > 1$. If the algorithm was context-sensitive then from this we would know that it requires only

$$O(k^{\log n+n}) = O(k^{\log n}k^n) = O(nk^n) = O(c^n)$$

steps (for some c), which contradicts the assumption of the lemma. □

Hence if we can show that a decision problem in a group has a lower bound on the time required that exceeds this exponential bound, then we can immediately prove it not to be context-sensitive. Of course, again it is generally extremely difficult to prove lower bounds, but this could be a useful observation.

2.4 Techniques used in context-sensitive Turing machines

We will wish to use many similar techniques in our exposition when we consider using Turing machines to solve problems in groups. To save unnecessary wordiness and repeated descriptions, let us describe some simple techniques which can be exploited when dealing with context-sensitive computation of problems in groups. We can then simply describe a particular action rather than describing the mechanism of the Turing machine. We will use these techniques withour further comment in our description of algorithms in the text and we will simply talk of, for example, 'check every subword' or 'delete a particular subword' without describing the actual technique in the Turing machine.

2.4.1 Re-using space

To make an obvious observation, note that we are free to re-use space as much as we like, since we can simply clear a worktape after we have finished a computation. Hence, if we have a long series of computations to perform, where we do not need to store the result of each, then we can easily perform these over and over again on the same tape.

2.4.2 Recognising symbols

Suppose we have an input string w. We are often concerned with products of groups, so first of all we certainly wish to know which group a given symbol belongs to. To achieve this, we write the generators of each group in turn on a tape separated by the blank symbol. For example, suppose we are considering some string of words over $X \cup Y \cup Z$ and we wish to know whether a symbol t lies in X, Y or Z. We have the elements of X, followed by a blank, followed by the elements of Y, followed by a blank, followed by the elements of Z, all written on a tape. We have a state component S that can take three values : S_x, S_y or S_z. We start in state component S_x and scroll along the tape until we find the symbol that we are looking for, changing state to S_y if we pass the first blank and then to S_z if we pass the second blank. Then the state we finish in tells us which set t lies in. This idea involves a tape of constant length and so certainly never affects our complexity results.

Also in groups, an elementary thing we may wish to do is, given a symbol, to know its inverse symbol. There are various ways to do this, probably the easiest is to assume that we have all the inverses contained in our generating set (so we have a monoid generating set), and then write these generators on a tape, in the form $g_1{g_1}^{-1}....g_n{g_n}^{-1}$. To find the inverse of an element we have a state component S taking values s_1 or s_2, and we simply scroll along the tape, changing state from s_1 to s_2, or vice versa, at each step. If we are in state s_1 when we find the required symbol, then we move one step forward to find the inverse, if we are in state s_2 then we move one step back.

2.4.3 Marking symbols and subwords

We will often want to 'mark' a symbol in a word in order to return to it later in the computation. To achieve this, we use a tape containing a single non-blank marker symbol. We always move along this tape simultaneously with the input tape, that is both heads always move together. We start by always moving the marker symbol along as we go (by deleting it and rewriting it when we move). When we come to the symbol we wish to mark, we stop moving the marker symbol and just move along the tape. When we wish to return to our marked point we simply scroll along the tape until we find it again and our input head is again pointing at our 'marked' symbol.

A crucial thing we will wish to do is mark out a particular subword. We can achieve this by taking a tape with two markers, the left-hand one of which can never pass the right-hand one. We define the subword marked by defining it to start in the cell marked by the first marker and ending in the cell preceding the last marker (we use the preceding cell to allow us to consider subwords of length 1).

In particular, one thing we may wish to do is investigate every subword of our input word. To do this, we start by keeping the first marker fixed, marking the first symbol, and scroll the second marker along the tape, allowing us to mark, in turn, every subword starting from the first symbol (a subword starting from the first symbol is called a *prefix*). Then, of course, we scroll the second marker back until it adjoins the first marker, advance them both one step and scroll the second marker forward again, allowing us to mark every subword starting from the second symbol. We continue in this vein and hence can investigate every possible subword.

The most important issue here is the advantage of context-sensitive algorithms in allowing us to investigate subwords. Algorithms bounded by, for example, logspace do not have this advantage since the subwords may be of linear length. This is a crucial point and one we will exploit many times.

2.4.4 Adding to and deleting from the input word

One thing we often wish to do is to make alterations to the input word, and keep changing it, for example by adding or deleting a subword. We cannot write on the input tape, but our procedure is to copy the word onto one of two worktapes. We have a state component S which takes one of two values indicating which of the two worktapes our word under consideration is written on. If we wish to, say, delete a subword, then we write the word with the subword deleted onto the other worktape, clear our current worktape, and switch states. So we can continue making changes to the word as often as we like with these two tapes.

We have not yet defined how to add or delete a subword. To add a subword to a particular point, we simply mark the point with a marker, copy the word up to that point, write the subword we wish to add and then write the rest of the word. To delete a subword, suppose we have the subword marked. Then we read along the word from the start, copying up to the beginning of the subword, then scrolling through without copying until we reach the second marker, and then copying the remainder of the word.

These ideas can all be implemented in context-sensitive space with the exception of adding a subword, since if there are no bounds on the length of the subword to add then we may exceed a linear bound, so we must be very careful when adding subwords that we stay within a linear bound.

Note as a final comment that this means when dealing with context-sensitive languages, we are free to delete trivial strings in a word such as aa^{-1}.

2.4.5 Incorporating other Turing machines

We should also note that we can incorporate any Turing machine T_1 into a Turing machine T_2. What we mean by this, is that suppose T_1 consists of an input tape and k worktapes. Then in T_2 we use $k+1$ worktapes to model T_1, where the first tape models the input tape of T_1 and the remaining k tapes model the worktapes of T_1. If during our calculation we wish to run the algorithm represented by T_1 on some word, then we can simply use these worktapes to model this algorithm. Clearly, if both of these machines operate in linear space, then so does T_2 with T_1 incorporated.

2.4.6 Sequential algorithms

We can also run one context-sensitive algorithm after another sequentially, each operating on the word produced by the previous algorithm, provided that the number of algorithms we wish to run is bounded by some constant number k (that is, not a function of the length of the input word). Suppose the i'th algorithm runs in space $c_i m$ on an input word of length m. Take a word of length n. Then the first algorithm runs in space $c_1 n$, the second in space $c_2(c_1 n)$, the third in space $c_3(c_2 c_1 n)$ and so on. Thus overall we operate in space $c_k c_{k-1} \ldots\ldots c_1 n$, which is clearly still a linear bound, and this is therefore a context-sensitive procedure.

2.5 Some simple examples

Finally, let us briefly indicate by the way of one or two simple examples, exactly what we mean by a decision problem in a group being context-sensitive, and the sort of approaches we can take.

Firstly we note some well-known results regarding free groups. Recall that a word in a free group is *reduced* if there is no occurrence of 1 (the symbol representing the identity), or xx^{-1} or $x^{-1}x$ (for some x), in the word.

Lemma 2.5.1 *Given a word u, there is a deterministic context-sensitive procedure to compute a reduced word equivalent to u.*

Proof We simply remove all occurrences of 1, and then successively remove all substrings of the form xx^{-1} or $x^{-1}x$ (for some x), and repeat this procedure until no such substrings remain, which leaves us with a word in reduced form. Any application of this decreases the length of the word, and so this is obviously a deterministic context-sensitive procedure. $\square$

We also have the following well-known result (see, for example, [37]).

Lemma 2.5.2 *In a free group F, two reduced words are equivalent in F if and only if they are identical as words.*

Hence, the following result is almost immediate.

Lemma 2.5.3 *Suppose G is a finitely-generated free group. Then G has deterministic context-sensitive word problem.*

Proof Given a word u, we can find a reduced word u' equivalent to u, by Lemma 2.5.1. The reduced word equivalent to the identity element is clearly the empty word. Hence, by Lemma 2.5.2, u is equivalent to the identity element if and only if u' is empty, which is trivial to check. Clearly this is a context-sensitive procedure since our word in question is never lengthened, and it is clearly deterministic. $\square$

We now turn to the conjugacy problem.

Lemma 2.5.4 *Suppose G is a finitely-generated free group. Then G has deterministic context-sensitive conjugacy problem.*

Proof Given a word u, we can easily calculate a cyclically reduced word representing a conjugate of u. We first freely reduce u, and then check to see if the first symbol is the inverse of the last symbol. If so, we conjugate by this symbol, which has the effect of removing them both. We continue doing this until we are left with a cyclically reduced conjugate of u.

We noted earlier that if $u \sim v$, then any conjugate of u is also conjugate to any conjugate of v. Hence, if we are given u and v as input words, we can produce cyclically reduced conjugates of each word, and thus we can assume that u and v are cyclically reduced.

Now, from [37], if u and v are cyclically reduced words, then they are conjugate if and only if u is a cyclic permutation of v. But we can clearly check this in linear space simply by comparing all possible cyclic permutations of u with v. Note that this requires only one worktape to write the permutations, since we can simply clear the tape and reuse the space every time we check a new permutation. If we find a permutation of u equal to v, then we accept the pair, otherwise if we exhaust all permutations, then we reject the words.

This is clearly a deterministic context-sensitive procedure. $\square$

Now, we consider finite groups.

Lemma 2.5.5 *Suppose that G is a finite group. Then G has deterministic context-sensitive word problem.*

Proof By Lemma 2.2.2 we are free to choose any generating set we like for G. So, since G is finite, let us choose a presentation where we take every element of G to be a generator, and our relators are given simply by the multiplication table for G. We can write this information on one tape of our Turing machine - of course this tape is of constant length so does not affect our complexity considerations. This tape acts as a 'lookup table', by which we mean that given a pair of symbols, we can look up which symbol we produce when we take their product.

Our algorithm to solve the word problem is then simple. Given an input word u, we start from the left-hand end of u, and work our way through u.

We calculate the symbol produced by the product of the first two elements, and then calculate the symbol produced by the product of this symbol with the next element, and so on. Clearly the only storage space we ever need is to store the current symbol, and so this is obviously a deterministic context-sensitive procedure.

A word is accepted if and only the final symbol produced is the symbol representing the identity, and so our algorithm indeed solves the word problem. □

Lemma 2.5.6 *Suppose that G is a finite group. Then G has deterministic context-sensitive conjugacy problem.*

Proof The procedure is similar to the above. Again we calculate the single symbols representing each word. Suppose these symbols are a and b. Then we simply calculate, for each c in our group, whether or not $c^{-1}ac = b$. This is a terminating procedure since our group is finite, and is clearly deterministic context-sensitive and solves the conjugacy problem. □

Let us look next at the generalised word problem, and consider a simple example.

Lemma 2.5.7 *Suppose that F is a free group, and that u is a fixed word of F. Then the generalised word problem of F with respect to the cyclic subgroup $\langle u \rangle$ is a deterministic context-sensitive problem.*

Proof Suppose we are given u and an input word v. We can of course start by freely reducing u and v, so we simply assume that they are freely reduced.

If v is the empty word then of course we accept it (since it represents the identity which lies in $\langle u \rangle$). So let us assume that v is non-empty.

Firstly, suppose that u is cyclically irreducible. Thus there is no cancellation when we consider words of the form u^n, and hence u^n is freely reduced. Then v lies in $\langle u \rangle$ if and only if v is of the form u^n or u^{-n} (for $n \geq 1$), since freely reduced words represent unique elements of a free group, by Lemma 2.5.2. Therefore, we simply scroll through v and see if it is of precisely this form - if so we accept it, otherwise we reject it.

We are left with the case where u is cyclically reducible. We can then determine w such that $u = w^{-1}u_1w$ with u_1 cyclically irreducible - this is easy to do simply by determining the largest prefix w of u such that w^{-1} is a suffix of u. Then v lies in $\langle u \rangle$ if and only if v is of the form $w^{-1}u_1^n w$ or $w^{-1}u_1^{-n}w$ (for $n \geq 1$), and again this is easy to verify.

It is clear that this procedure never increases the length of the input word and thus the algorithm is deterministic context-sensitive as required. □

The following result is somewhat similar.

Lemma 2.5.8 *Given a word u in a free group F, then there is a deterministic context-sensitive procedure to produce a word u_0 such that $u = u_0^c$ where c is maximal.*

Proof We assume u is freely reduced. If u is cyclically reduced, then we simply check each prefix of u in turn, until we find the smallest prefix u_0 for which u is a concatenation of some number of u_0 - of course u_0 may be equal to u. From Lemma 2.5.2, this must be the u_0 we seek.

Otherwise, if u is not cyclically reduced, we deduce the largest prefix α of u such that $u = \alpha^{-1}u'\alpha$, and so u' is cyclically reduced. Finding the appropriate u'_0 as in the previous paragraph, then we take $u_0 = \alpha^{-1}u'_0\alpha$.

This procedure clearly operates in linear space since all the words in question are shorter than the original word. □

The next result is a little less simple than the previous results, and may appear somewhat convoluted as an illustrative example - however we choose this example deliberately as the result will be of use later!

Lemma 2.5.9 *Let g be a word of length n, and suppose that u, v, and f are words in a free group F, all of length $O(n)$. Then there is a procedure, operating in space $O(n)$, to determine the occurrence problem of g in the set $S = \{u^i f v^j : i, j \in \mathbb{Z}\}$.*[2] *Suppose $u = (u_0)^{c_1}$ and $v = (v_0)^{c_2}$, where c_1 and c_2 are maximal. Then the values of i and j can be uniquely determined, and the appropriate words u^i and v^j can be written in space $O(n)$, unless $f^{-1}u_0 f = {v_0}^{\pm 1}$, in which case there are infinitely many solutions.*

Proof Obviously, we can start by freely reducing each of our words. Note that, by Lemma 2.5.2, two reduced words are equivalent in F if and only if they are identical as words.

We firstly compute the words u_0 and v_0 as in Lemma 2.5.8. If $f^{-1}u_0 f = {v_0}^{\pm 1}$, then it is immediate that g lies in S if and only if g lies in the set $\{(u_0)^{ic_1 \pm jc_2} f : i, j \in \mathbb{Z}\}$, which is easy to verify in linear space, using Lemma 2.5.7, and then the Euclidean algorithm. In this case, if there is any solution, then there are obviously infinitely many solutions.

Otherwise, we proceed via a series of cases, and reductions of the problem. Firstly, we show that we can assume that u and v are cyclically reduced. Suppose $u = \alpha^{-1} u_R \alpha$, and $v = \beta^{-1} v_R \beta$, where u_R and u_R are cyclically reduced. Then, for any i, j, we have

$$u^i f v^j = \alpha^{-1}(u_R)^i \alpha f \beta^{-1}(v_R)^j \beta.$$

Therefore, the problem is equivalent to deciding membership of $\alpha g \beta^{-1}$ in $\{(u_R)^i(\alpha f \beta^{-1})(v_R)^j : i, j \in \mathbb{Z}\}$. Each of these words are obviously of length

[2]Of course, S is considered as a set of elements of F.

$O(n)$, and so we have reduced the problem to the case where u and v are cyclically reduced. This does not affect the values of i and j.

Now, suppose that f begins with an occurrence of u. Then $f = uf_1$, for some f_1, and hence g is equal to $u^p f v^q$ (for some p, q) if and only if $g = u^{p+1} f_1 v^q$. Obviously, then, this question is equivalent to determining membership of g inside the set $\{u^i f_1 v^j : i, j \in \mathbb{Z}\}$. Hence, we can eliminate any occurrences of u at the beginning of f. Similarly, of course, we can eliminate occurrences of u^{-1} at the beginning of f, or $v^{\pm 1}$ at the end of f. Therefore, we can assume that f does not begin with $u^{\pm 1}$, or end with $v^{\pm 1}$. We can easily store how many occurrences of $u^{\pm 1}$ or $v^{\pm 1}$ we eliminate here, and add them to the value of i and j we eventually determine, to give us the total power of u and v we require.

We will assume that f is non-empty, since if f is empty then we are testing membership of $\{u^i v^j : i, j \in \mathbb{Z}\}$, and this case will be covered as part of our proof. The major problems occur when considering cancellation of the f with the powers of u and v. Suppose there was no cancellation at all. By this, of course, we mean that the last symbol of u is not the inverse of the first symbol of f, and similarly the first symbol of v is not the inverse of the last symbol of f. But then, in writing any word $u^r f v^s$, there is no cancellation anywhere in the word (since u and v are cyclically reduced). Hence, any word $u^r f v^s$ is reduced. Therefore, by Lemma 2.5.2, we can merely check to see if g is precisely of this form, which is trivial. It is obvious that the values of i and j are uniquely given by this procedure, and u^i and v^j are of length no greater than the length of g, which is of length $O(n)$.

Otherwise, we can write f in the form $f = f_u f' f_v$, where $u = u' {f_u}^{-1}$, and $v = {f_v}^{-1} v'$. We choose f' minimal, so that there is no cancellation between u' and f', and no cancellation between f' and v'. Note that u' and v' cannot be empty, by our assumption above that f does not begin with $u^{\pm 1}$, or end with $v^{\pm 1}$.

Suppose f' is non-empty. Then g lies in the set $\{u^i f v^j : i, j \in \mathbb{Z}\}$ if and only g lies in the set $\{u^i(u'f'v')v^j : i, j \in \mathbb{Z}\}$. But then there is no cancellation in any of the words in this set, and the procedure is exactly as above, to simply check that g is of this precise form, and to determine i and j.

Otherwise, f' is empty. Hence, f must completely cancel, by which, of course, we mean that our words must be of the form $u = u_1u_2$, $v = v_1v_2$ and $f = {u_2}^{-1}{v_1}^{-1}$. Let γ be the reduced form of u_1v_2. Then g lies in the set $\{u^i f v^j : i, j \in \mathbb{Z}\}$ if and only g lies in the set $\{u^i \gamma v^j : i, j \in \mathbb{Z}\}$. If γ is non-empty then, again, there is no cancellation in any of these words, and we have a similar case to the above, and again we can determine i and j.

The only remaining case is where γ is empty. In this case, we are simply testing whether or not g is a member of the set $\{u^i v^j : i, j \in \mathbb{Z}\}$. Unless $u_0 = (v_0)^{\pm 1}$, we do not get complete cancellation, and this is easily solved exactly in the methods above. Otherwise, if $u = (u_0)^{c_1}$ and $v = (u_0)^{c_2}$, then we have reduced to the condition above, where we are simply testing membership of $\{(u_0)^{ic_1 \pm jc_2} : i, j \in \mathbb{Z}\}$. Hence we have considered every possible case. Note than in all of the cases bar the exceptional case, we are simply testing words for equivalence in a free group, and so i and j are uniquely determined by Lemma 2.5.2. Finally considering the bounds on the space required, note that the only time that we can ever lengthen a word is in assuming that u and v are cyclically reduced at the start, which is still linearly bounded, and is only done once. The remainder of the tests are simple tests of equivalence of g to some word. Therefore, this is obviously a linearly-bounded procedure. $\square$

Sometimes in our work we will want to deduce some 'given' words from some initial word. Suppose we are given an input word of length n, from which other words are deducible. When considering context-sensitive procedures it is clearly then imperative that we can deduce these words from our initial word with only a linear bound on their length, in order for our whole procedure to be linearly bounded.

As a particular example, given some word w of length n in a free group F, suppose we can deduce words g, u, w and f from w, which are of length $O(n)$ and lie in F. Then we can solve the above problem for these words. This is the application of this result that we will see later, and we can then consider this a context-sensitive procedure in the length of w.

The examples we have given above are relatively simple examples, but they serve to illustrate the sort of problems we will wish to consider. We will consider many more examples of groups with context-sensitive decision problems later.

Chapter 3

Products of groups with context-sensitive decision problems

We now move on to consider some possible products of groups with (deterministic) context-sensitive decision problems, and exhibit a series of closure properties that these classes of groups possess.

We should make a brief comment on the notation in these results. Clearly, it is at least as strong a statement to say that an algorithm is 'deterministic context-sensitive' as it is to say that it is 'context-sensitive', and hence when we give explicit examples of algorithms for particular groups (for example with the free, or finite, groups earlier) we simply refer to the algorithms as being 'deterministic context-sensitive'.

In the following results, however, we are concerned with closure properties of groups with either context-sensitive, or deterministic context-sensitive, decision problems. Thus, when we use the notation '(deterministic) context-sensitive' throughout a statement or proof, we mean that the result holds whether we read the phrase as ‘context-sensitive’ throughout, or read it as ‘deterministic context-sensitive’ throughout. The bracketing of the word ‘deterministic’ prevents unnecessary duplication by writing out each theorem or result twice.

3.1 Products of groups

Suppose $G = \langle X : R \rangle$ and $H = \langle Y : S \rangle$ are groups with $X \cap Y = \emptyset$. The *free product* of G and H is defined as the group

$$G * H = \langle X \cup Y : R \cup S \rangle.$$

In a similar vein, the *direct product* of G and H is defined as the group

$$G \times H = \langle X \cup Y : R \cup S \cup \{xy = yx : x \in X, y \in Y\} \rangle,$$

that is we insist that every element of G commutes with every element of H. Note that this is equivalent to saying that our group $G \times H$ consists of ordered pairs (g, h), with $g \in G$ and $h \in H$.

Suppose we have subgroups $K \leq G$ and $L \leq H$, and that there exists an isomorphism $\phi : K \to L$. We may assume that our generating sets X and Y are given by

$$X = \{k_1, ..., k_p, g_1, ..., g_r\}, Y = \{l_1, ..., l_p, h_1, ..., h_s\}$$

where

$$K = \langle k_1, ..., k_p \rangle, L = \langle l_1, ..., l_p \rangle$$

with each $g_i \notin K$, $h_j \notin L$, and $\phi(k_i) = l_i$, for all i. Then the *amalgamated free product* of G and H, with respect to K and L, is defined as the group

$$G *_{K,L} H = \langle X \cup Y : R \cup S \cup \{k_i = l_i : 1 \leq i \leq p\} \rangle.$$

If K and L are central in G and H respectively (that is, they commute with the whole of their respective group), then we define the *central product* of G and H, with respect to K and L, to be the group

$$G \times_{K,L} H = \langle X \cup Y : R \cup S \cup \{k_i = l_i : 1 \leq i \leq p\} \cup \{xy = yx : x \in X, y \in Y\}\rangle.$$

Note that the notation $G *_{K,\phi} H$ and $G \times_{K,\phi} H$ is sometimes used in preference to the above notation, however usually we shall use the notation defined above in order to make the identified subgroups explicit, and we assume that the groups are given in a form to make the definition of ϕ obvious (since we have freedom to choose generating sets).

Note, also, that since we are free to choose any generating set we like, having done this amalgamation, we can omit the generators l_i and the relations $k_i = l_i$, and substitute l_i for k_i everywhere in our relations. This essentially considers K as a common subgroup of G and H, and we can write $G \times_K H$ and $G *_K H$. This viewpoint will often simplify our calculations somewhat.

Note, of course, that if K is the trivial subgroup, then $G \times_K H \cong G \times H$ and $G *_K H \cong G * H$.

3.2 Direct products

It is instructive, as an example of the use of a Turing machine to solve a decision problem in a product of groups, to provide a direct, detailed analysis of the solution of the word and conjugacy problem for the simple case of a direct product, although these results will follow from our work on central products later.

A fundamental tool that we shall use is being able to represent a word u in some *normal form*, that is to produce a word u' (equivalent to u in G) in some canonical form, which is easier to solve our problem for (and hence solve the problem for u). We can then show that the entire procedure (that is, producing the normal form, and then solving the problem for this equivalent form) is context-sensitive. In the case of direct products the normal form is particularly easy to define. Suppose the generators for G are $g_1, ..., g_k$ and the generators for H are $h_1, ..., h_l$ (considered as monoid generating sets). Any word u in $G \times H$ is equivalent to a word $u' = u_g u_h$, with $u_g \in G$ and $u_h \in H$. We obtain u_g by simply writing down the g_i contained inside u in turn, and similarly for u_h. For example, $h_2 g_1 g_3 h_1 g_2 h_2$ is equivalent to $g_1 g_3 g_2 h_2 h_1 h_2$.

Theorem 3.2.1 *Let G and H be groups, and suppose G and H both have (deterministic) context-sensitive word problem. Then the direct product $G \times H$ also has (deterministic) context-sensitive word problem.*

Proof Let $G = \langle g_1, ..., g_k \rangle$ and $H = \langle h_1, ..., h_l \rangle$. We use a Turing machine which incorporates the context-sensitive Turing machines for the word problem of G and H (which we know exist since G and H have context-sensitive word problem).

Now, any input word u is equivalent to a word $u_g u_h$ as above. But no relator (except the commutators) involves elements from both G and H. Hence u is equivalent to the identity if and only if u_g and u_h are both equivalent to the identity.

To solve the word problem, we start by reading in the word, one symbol by one, from left to right, looking up which group the symbol belongs to, and then storing the g_i in sequence on the tape representing the input tape for the Turing machine for the word problem for G, and the h_i on the corresponding tape for H.

Suppose u is of length n. After reading all of u, we have the appropriate words u_g and u_h on the tapes representing the input tapes for the Turing machines for the word problem of G and H. We can then simply use these context-sensitive algorithms to determine whether or not u_g and u_h do indeed both equal the identity, and hence whether our input word equals the identity as required.

The Turing machine for the word problem for G has tapes bounded in length by $k_1 n_1$ (where n_1 is the length of the input) for some k_1, since the word problem is context-sensitive, and similarly the tapes for the word problem solver for H are bounded in length by some $k_2 n_2$, for input of length n_2. But clearly n_1 and n_2 are no greater than n and hence all tapes are bounded in length by $\max(k_1, k_2)n$, which is a linear bound, and so this procedure is context-sensitive as required.

Note finally that if G and H have deterministic context-sensitive word problem, then this algorithm is entirely deterministic. $\square$

Let us now turn to the conjugacy problem.

Theorem 3.2.2 *Let G and H be groups, and suppose G and H both have (deterministic) context-sensitive conjugacy problem. Then the direct product $G \times H$ also has (deterministic) context-sensitive conjugacy problem.*

Proof Suppose u and v are our input words. Then u and v are equivalent in G to words $u' = u_g u_h$ and $v' = v_g v_h$ as before. By definition, u is conjugate to v if and only if there exists $w \in G \times H$ such that $w^{-1}uwv^{-1} = 1$, and of course we may restrict our search to a w of the form $w = w_g w_h$. Then w conjugates u and v if and only if

$$w_h{}^{-1} w_g{}^{-1} u_g u_h w_g w_h v_h{}^{-1} v_g{}^{-1} = 1,$$

which is equivalent to

$$w_g{}^{-1} u_g w_g v_g{}^{-1} w_h{}^{-1} u_h w_h v_h{}^{-1} = 1.$$

This word is equivalent to the identity if and only if both ${w_g}^{-1}u_g w_g {v_g}^{-1} = 1$ and ${w_h}^{-1}u_h w_h {v_h}^{-1} = 1$. Hence, u is conjugate to v if and only if $u_g \sim_G v_g$ and $u_h \sim_H v_h$.

So, our algorithm is actually very similar to the word problem algorithm. Again we use a Turing machine incorporating the Turing machines for the conjugacy problem of G and H. We have u followed by v (separated by a blank) on the input tape, and as before, we read in these words symbol by symbol, creating the u_g and u_h on the tapes representing the input tapes for the conjugacy problem solver for G and H respectively. When we read the blank, we write a blank on both these tapes, and proceed similarly for v_g and v_g. Then we simply apply the relevant algorithms to see if $u_g \sim_G v_g$ and $u_h \sim_H v_h$ as required.

As before, this is clearly a context-sensitive procedure and so we are done. Again note that if G and H have deterministic context-sensitive conjugacy problem, then this algorithm is entirely deterministic. □

3.3 Free and amalgamated free products

Let us now consider amalgamated free products, which of course generalise free products. Suppose we have groups G and H with isomorphic subgroups $K = \langle k_1, ..., k_r \rangle$ and $L = \langle l_1, ..., l_r \rangle$, where we have the obvious isomorphism $\phi : K \to L$ given by the natural extension of the map which sends each k_i to the respective l_i. Let us take monoid generating sets $\langle k_1, ..., k_r, g_1, ..., g_s \rangle$ and $\langle l_1, ..., l_r, h_1, ..., h_t \rangle$ for G and H respectively, where the g_i and h_j do not lie in K and L respectively (of course, products of these elements may lie in K or L). We consider the amalgamated free product $G *_{K,L} H$. As above, we can eliminate the generators of L, and consider K as a 'common subgroup' of G and H.

Following the ideas of [42] and [39], we define a word $u \in G*_K H$ to be in *reduced form* if $u = u_K u_1 u_2 .... u_n$, where u_K is a (possibly empty) string in K, $u_i \notin K$ for all $i \geq 1$, and u_i and u_{i+1} do not lie in the same factor for all $i \geq 1$. The u_i are said to be *syllables* of the word. The presence of the subword u_K here allows us to consider words in K without changing our definition, simply by having $n = 0$. However, if $n \geq 1$ then we can incorporate u_K into u_1 and simply consider $u = u_1 u_2 .... u_n$ which is perhaps more natural. We define a word $u \in G*_K H$ to be *cyclically reduced* if it is in a reduced form, and u_1 and u_n are not in the same factor (excepting the case where $n = 1$).

Every word is equivalent to a reduced word, and it is well-known that if $u_K u_1 u_2 .... u_m$ and $v_K v_1 v_2 .... v_n$ are reduced words equivalent to the same word u, then we must have $m = n$. Hence, we can talk of the *syllable length* of a word, with respect to the amalgamated free product, as the number of syllables in any reduced word equivalent to u (note that this will be 0 if u lies in K). We say that a word is *short* with respect to the free product with amalgamation if it has syllable length at most 1, and that the word is *long* otherwise.

We have the following important lemma and theorem, for proofs see, for example, [39].

Lemma 3.3.1 *Let G and H be as above. Suppose $u \in G*_K H$ is in reduced form as above. If $n \geq 1$, then u does not represent the identity.*

Theorem 3.3.2 (Solitar's Theorem) *Every element of $G*_K H$ is conjugate to a cyclically reduced element of $G*_K H$. In addition, suppose that u is a cyclically reduced element of $G*_K H$. Then*

- *if u is conjugate to an element of K, then u lies in some factor,*
- *if u is conjugate to an element v in some factor, but not in a conjugate of K, then u and v lie in the same factor and are conjugate therein,*
- *if u is conjugate to a cyclically reduced element $v = v_1....v_m$, with $m \geq 2$ and v_i in a different factor to v_{i+1} for all i, then u can be obtained by cyclically permuting $v_1....v_m$ and conjugating by an element of K.*

Let G, H and K be as above. We cannot come up with such a wide-ranging statement for the amalgamated free product as we did for the direct product. The problem is that we need to be able to determine whether or not a string of symbols actually represents an element of K, that is, determine the effective generalised word problem for G and H with respect to K. For example, suppose we had a word $u = u_g u_h$ where u_g is a string of symbols in the g_i and u_h is a string of symbols in the h_i. We may have a situation where neither u_g nor u_h represents the identity, but u_g is equivalent to some element $k \in K$ and u_h is equivalent to k^{-1}, and hence u is equivalent to the identity. Therefore, we cannot use the ideas from above. However, we can produce at least a partial result. Firstly, of course, we need to be able to reduce words.

Lemma 3.3.3 *Suppose G and H are groups with isomorphic subgroups K and L respectively, and suppose that there exists a (deterministic) context-sensitive procedure to decide the effective generalised word problem for K in G, and similarly for L in H. Then, for any word u in $G *_{K,L} H$, there exists a (deterministic) context-sensitive procedure to produce a reduced word equivalent to u.*

Proof As usual, we may eliminate L and consider K as a common subgroup of G and H. Let $G = \langle k_1, ..., k_r, g_1, ..., g_s\rangle$ and $H = \langle k_1, ..., k_r, h_1, ..., h_t\rangle$

as usual, where the k_i generate K. Of course, we incorporate the Turing machines to solve the word problem for G and H, and the effective generalised word problem for K in G and H, into our Turing machine.

Let our input word be $u \equiv v_1....v_n$, where each v_i is a string in no more than one of either the k_i, the g_i or the h_i, and v_i is not a string in the same symbols as v_{i+1} for all i. We can consider each of these v_i.

If some v_j is a string in the k_i, then we test it in the word problem solver for G, say, and if it is indeed equivalent to the identity, then we delete it and continue our algorithm on the shorter word thus obtained (redefining the v_i if necessary).

If v_j is a string in the g_i or h_i, then we use our algorithm for the effective generalised word problem in the appropriate group, to see if it lies in K. If so, then we produce a representative in the k_i, and replace v_j by this representative, and continue our algorithm on this word, again redefining the v_i if necessary.

We continue in this fashion until we either terminate with the empty word (which is obviously the reduced form corresponding to the identity), or a word in K (in which case the word is of the form u_K and is therefore of reduced form), or we cannot make any more deletions or substitutions. In this case, we are left with a word in reduced form (any subwords in the k_i can be incorporated into a factor to give us a reduced form).

Hence, this procedure has produced a reduced form for u. However, the word is not always being made shorter in length, since when we replace a subword by a representative in K, then this may increase the length of the subword by a linear factor C (since the procedures for the effective generalised word problem are context-sensitive). However, in our algorithm, we never replace a string of k_i by a non-empty word, so in particular, we certainly never replace it by a string containing any g_i or h_i, and so the number of 'replacements' we may ever have to make for a given word is finite, bounded by the number of g_i and h_i in the original word. Hence, the length of any word under consideration can never exceed Cn where n is the length of the input word.

So we will indeed eventually terminate, since there are only finitely many 'replacements' to make, and deletions of subwords strictly reduce the length of the word, and our algorithm is clearly therefore a context-sensitive procedure.

Note, finally, that if G and H have deterministic context-sensitive effective generalised word problem with respect to K and L, then this algorithm is entirely deterministic. □

We then have the following result as a simple corollary.

Theorem 3.3.4 *Let G, H, K and L be groups as above. Suppose G and H both have (deterministic) context-sensitive word problem, and suppose in addition that there exists a (deterministic) context-sensitive algorithm to decide the effective generalised word problem for G with respect to K, and for H with respect to L. Then the amalgamated free product $G *_{K,L} H$ also has (deterministic) context-sensitive word problem.*

Proof We produce the reduced form, u', of our input word u, via the linearly bounded algorithm of Lemma 3.3.3. By Lemma 3.3.1, u cannot represent the identity unless u' lies in one of the factors. If it does, then we check to see if it is indeed equivalent to the identity in the appropriate group, using the (deterministic) context-sensitive algorithm for that group. □

As a corollary to Theorem 3.3.4 we have the following.

Corollary 3.3.5 *Let G and H be groups, and suppose G and H have (deterministic) context-sensitive word problem. Then the free product $G * H$ also has (deterministic) context-sensitive word problem.*

Proof This follows immediately from the previous result, taking $K = 1$, our (deterministic) context-sensitive algorithm for determining if a word u lies in K is simply to run the word problem solver on u. □

In general, it is a difficult question to ask which groups with (deterministic) context-sensitive word problem also have (deterministic) context-sensitive effective generalised word problem with respect to the appropriate amalgamated subgroup. However, we certainly have the following lemma.

Lemma 3.3.6 *Let G be a group with (deterministic) context-sensitive word problem, and suppose K is a finite subgroup of G. Then G has (deterministic) context-sensitive effective generalised word problem with respect to K.*

Proof Let us enumerate the elements of K as words $k_1, ..., k_n$, each representing an element of K. Suppose we are given a word u. We systematically test the words $u^{-1}k_i$ in the word problem solver for G. If we accept any of these words, say $u^{-1}k_p = 1$, then u lies in K with representative k_p. If none of the words $u^{-1}k_i$ are accepted by the word problem solver then u cannot equal any of the k_i and hence does not lie in K.

Since the enumeration of the elements of K merely gives us finitely many fixed representatives, which do not affect our complexity considerations, then this algorithm clearly operates in linear space. If the word problem for G is deterministic context-sensitive then this procedure is also deterministic. □

Given this, we can immediately deduce the following corollary.

Corollary 3.3.7 *Suppose G and H are groups with isomorphic subgroups K and L respectively, where K and L are finite. If G and H have (deterministic) context-sensitive word problem, then the amalgamated free product $G *_{K,L} H$ also has (deterministic) context-sensitive word problem.*

Proof Immediate from Lemma 3.3.4 and Lemma 3.3.6. □

Let us now move on to the conjugacy problem. Unfortunately, the situation here is somewhat more restrictive, in that it is possible to have two groups with (deterministic) context-sensitive conjugacy problem and (deterministic) context-sensitive effective generalised word problem with respect to an amalgamated subgroup, yet the amalgamated free product with respect to this subgroup does not have (deterministic) context-sensitive conjugacy problem. An example of this occurs in the second half of the proof of the main theorem of [15]. However, we do have the following result.

Theorem 3.3.8 *Let G, H, K and L be groups as before. Suppose G and H both have (deterministic) context-sensitive conjugacy problem, and suppose the amalgamated subgroups K and L are finite. Then the amalgamated free product $G *_{K,L} H$ also has (deterministic) context-sensitive conjugacy problem.*

Proof As usual, we start by eliminating the l_i and considering K as a common subgroup.

Suppose u and v are our input words, written on the input tape separated by a blank. We can cyclically reduce u and v, firstly by putting them in reduced form (as in Lemma 3.3.3) and then conjugating if necessary (replacing u or v by a conjugate does not, of course, affect the property of conjugacy between them). Hence we can assume u and v are cyclically reduced and appeal to Theorem 3.3.2.

Suppose u is empty. Then if v is empty, we accept u and v as conjugate and if v is non-empty we reject them, and of course similarly if v is empty. Alternatively, suppose u lies in one of the factors. Then, by Theorem 3.3.2, v must also lie in the same factor. So we check to see if this is true, and if not then we reject u and v. If it is true, then we use the (deterministic) context-sensitive algorithm for the conjugacy problem in the appropriate group to determine whether or not they are conjugate, and accept and reject accordingly. Of course, we have an entirely similar procedure if v lies in one of the factors.

Finally, we are left with the case where $u = x_1.....x_n$ and $v = y_1.....y_m$, where $n, m \geq 2$. By Theorem 3.3.2 we must have $m = n$, so if this is not the case, then we reject the words. If $m = n$, then we know, by Theorem 3.3.2, that u and v are conjugate if and only if we can obtain a word equivalent to u by first of all taking some cyclic permutation of the y_i, and then conjugating by an element of K. Enumerating the finite number of elements of K, we consider each permutation of the y_i, and each possible conjugation by an element of K, and see if we obtain a word equivalent to u. Clearly there are only a finite number of words to check.

This algorithm is clearly context-sensitive and, as usual, if our routines are deterministic then this procedure is entirely deterministic. □

In general, it seems to be a difficult question to decide whether or not this sort of result holds for given groups.

As a corollary to Theorem 3.3.8 we have the following.

Corollary 3.3.9 *Let G and H be groups, and suppose G and H have (deterministic) context-sensitive conjugacy problem. Then the free product $G * H$ also has (deterministic) context-sensitive conjugacy problem.*

Proof This follows immediately from the previous result, taking $K = 1$. $\square$

3.4 Central products

Let us now move on to look at central products. Suppose we have groups G and H with central subgroups $K = \langle k_1, ..., k_r \rangle$ and $L = \langle l_1, ..., l_r \rangle$, where K and L are isomorphic via the obvious isomorphism which sends each k_i to the corresponding l_i. Again, let us take monoid generating sets $\langle k_1, ..., k_r, g_1, ..., g_s \rangle$ and $\langle l_1, ..., l_r, h_1, ..., h_t \rangle$ for G and H respectively, where the g_i and h_j do not lie in K and L respectively.

Theorem 3.4.1 *Let G, H, K and L be groups with generating sets as above. Suppose G and H both have (deterministic) context-sensitive word problem, and suppose that G has (deterministic) context-sensitive effective generalised word problem with respect to K, and similarly for H with respect to L. Then the central product $G \times_{K,L} H$ also has (deterministic) context-sensitive word problem.*

Proof As usual, we begin by eliminating the l_i and considering K as a common subgroup of our two words.

The algorithm is extremely similar to the algorithm we gave for direct products (Theorem 3.2.2). An input word u is equivalent to a word $u_k u_g u_h$, where u_k is a string in the k_i, and similarly for u_g and u_h. As usual, we read through our input word and build up the u_k, u_g and u_h as we read in the symbols.

This word can only be equivalent to the identity if u_g and u_h both lie in K, since there are no relators involving the g_i and h_i apart from the commutators. Hence, we test u_g and u_h for membership of K, and if either does not lie in K we reject the word. If both lie in K, then we replace them by their representatives and test the resulting word (which is a word in the k_i) in the word problem solver for G, say.

This procedure is clearly context-sensitive, and if the algorithms in the hypothesis are deterministic, then this procedure is determinstic. $\square$

Let us now move on to look at the conjugacy problem. As we might expect we have a similar result to the case in amalgamated free products.

Theorem 3.4.2 *Let G, H, K and L be groups with generating sets as above. Suppose G and H both have (deterministic) context-sensitive conjugacy problem and suppose K and L are finite. Then the central product $G \times_{K,L} H$ also has (deterministic) context-sensitive conjugacy problem.*

Proof As usual, we eliminate the l_i and consider K as a common subgroup. Suppose our input words are u and v, which we write in the equivalent forms $u_k u_g u_h$ and $v_k v_g v_h$. We seek a conjugating element w such that $w^{-1}uw = v$. It is enough to search for w of the form $w_g w_h$, since any component w_k would be central, and hence cancel in $w^{-1}uw$. Then, u and v are conjugate via w if and only if

$$w^{-1}uwv^{-1} = {w_h}^{-1}{w_g}^{-1}u_k u_g u_h w_g w_h {v_h}^{-1}{v_g}^{-1}{v_k}^{-1} = 1,$$

and hence we have

$$u_k v_k {w_g}^{-1} u_g w_g {v_g}^{-1} {w_h}^{-1} u_h w_h {v_h}^{-1} = 1.$$

This word certainly can not equal the identity unless both ${w_g}^{-1}u_g w_g {v_g}^{-1}$ and ${w_h}^{-1}u_h w_h {v_h}^{-1}$ lie in K. Enumerate the elements of K by $\kappa_1, ..., \kappa_n$. Now, ${w_g}^{-1}u_g w_g {v_g}^{-1} = \kappa_i$ if and only if ${w_g}^{-1}u_g w_g = \kappa_i v_g$. So, if such a w_g

exists it conjugates u_g and $\kappa_i v_g$, for some i. Similarly, if such a w_h exists, it conjugates u_h and $\kappa_j v_h$, for some j.

Hence, our algorithm is as follows. We systematically run through each pair of elements (κ_i, κ_j) in K. We test the words u_g and $\kappa_i v_g$ in the conjugacy problem algorithm for G, and the words u_h and $\kappa_j v_h$ in the conjugacy problem algorithm for H. If both of these accept for some pair (κ_i, κ_j), then we have a possible solution, and we test the word $u_k v_k \kappa_i \kappa_j$ for equivalence to the identity (using Lemma 2.5.5, since K is finite). If this word is indeed equivalent to the identity, then we have found a conjugating element, and we accept the words. Otherwise, we continue the algorithm until we have considered all possible pairs. If we terminate without acceptance, then we reject u and v.

This algorithm is context-sensitive, and if the algorithms in the hypothesis are deterministic then this procedure is determinstic. $\square$

Chapter 4

Subgroups and extensions of groups with context-sensitive decision problems

The discussion in the previous chapter involved the situation where we had two groups with context-sensitive decision problem, and we tried to combine them together in some way. Perhaps even more fundamentally, we could ask what happens if we tried to extend a single group with context-sensitive decision problem, in some way. Let us begin with the fundamental idea of HNN and Britton extensions, which have a close association with amalmagated free products.

4.1 HNN extensions and Britton extensions

We consider a common, and extremely important, extension of a group, which was first defined by Higman, Neumann and Neumann (hence the name HNN extension) in [27].

Suppose we have a group G, and suppose that G has isomorphic subgroups $H = \langle h_1, ..., h_n \rangle$ and $K = \langle k_1, ..., k_n \rangle$. These need not be distinct, but let us take our set of generators for G to be

$$X = \{h_1, ..., h_n, k_1, ...k_n, g_1, ...g_m\}.$$

Then if $G = \langle X : R \rangle$, we define the *HNN extension* of G with respect to H and K to be

$$G^*{}_{H,K,t} = \langle X \cup \{t\} : R \cup \{t^{-1}h_i t = k_i : 1 \leq i \leq n\} \rangle$$

where $t \notin X$, so t essentially conjugates H and K. We call t a *stable letter* for this group.

We can use the notation $\{G, t : t^{-1}Ht = K\}$ as shorthand for a presentation of the HNN extension of G with respect to t, and associated subgroups H and K.

More generally we could consider the case where we have more than one stable letter. The exposition in the coming section here is courtesy of [42]. We say that a group $G^*{}_{H,K,T}$ is a *Britton extension* of G with respect to the subgroups H and K, and stable letters $T = \{t_i\}$ (where $1 \leq i \leq r$ for some r), if it is of the form

$$G^*{}_{H,K,T} = \langle X \cup \{t_i\} : R \cup \{{t_j}^{-1}h_i t_k = k_i : 1 \leq i \leq n, \text{ some } t_j, t_k \in T\} \rangle.$$

We use the term T-symbols to refer to the set $\{{t_i}^{\pm 1}\}$. If $t_j = t_k$ in the free group obtained by setting all the letters of X equal to 1, then we say that t_j and t_k are *equivalent*.

Note that the terminology used here varies from author to author. The terms 'HNN extension' and 'Britton extension' are generally interchangeable. We use the terms in the way we do here, simply to distinguish between the case with a single stable letter (which is perhaps the more widely-used meaning of 'HNN extension') and the case with several stable letters.

It is not immediately obvious that these constructions do not collapse the group. However, as was proved in [27], for any group G with subgroups H and K, G embeds in its HNN extension with respect to H and K, and similarly for more general Britton extensions.

We define a word $u \in {G^*}_{H,K,T}$ to be *T-reduced* if it is not equivalent in the group to any word with fewer T-symbols. The important concept here is the idea of 'pinching' subwords to produce a word in T-reduced form. Suppose we have a subword ${t_j}^{-1}ut_k$ where t_j is equivalent to t_k, and suppose we can deduce the fact that ${t_j}^{-1}ut_k = v$. Then we can replace ${t_j}^{-1}ut_k$ by v, thus producing a word with two fewer T-symbols. This procedure is called a 'pinch' of the T-symbols. We have the following lemma, for a proof see, for example, [42].

Lemma 4.1.1 *Suppose w is a word in ${G^*}_{H,K,T}$. Then w is T-reduced if and only if no T-symbols can be pinched out of w.*

We stressed here the need to be able to 'deduce' a relation ${t_j}^{-1}ut_k = v$. This is actually extremely simple. Following the exposition in Chapter II of [42], if t_j and t_k are distinct then we only have such a relation if u is equivalent to some u_1 where we have the relation ${t_j}^{-1}u_1t_k = v$ in our set of defining relations. If $t_j = t_k = t$, then we have $t^{-1}ut = v$ if and only if u is equivalent to a word $u_1.....u_m$, and v is equivalent to a word $v_1.....v_m$, where we have $t^{-1}u_it = v_i$ for all $1 \leq i \leq m$, since then we have

$$t^{-1}ut = t^{-1}u_1.....u_mt = t^{-1}u_1tt^{-1}.....tt^{-1}u_mt = v_1....v_m = v.$$

A subword in G to which a pinching may be applied is called *pinchable*.

So provided we are able to determine for an arbitrary string of symbols whether or not they represent an appropriate word, we always have an effective procedure to reduce a given word.

We have the following fundamental lemma, for a proof see for example [37].

Lemma 4.1.2 (Britton's Lemma) *Suppose $G^*{}_{H,K,T}$ is a Britton extension of G with respect to subgroups H and K. If $u \in G^*{}_{H,K,T}$ is a T-reduced word $u_0 {t_{j_1}}^{\pm i_1} u_1 \ldots\ldots {t_{j_n}}^{\pm i_n} u_n$ (where each t_{j_i} is a T-symbol, and each u_i does not contain a t-symbol), and $n \geq 1$, then u is not equivalent to the identity.*

We note that an essentially equivalent formulation of Britton's Lemma asserts that a T-reduced word cannot lie in G if it contains a T-symbol, and we shall use the notation Britton's Lemma for either formulation.

4.1.1 The word problem

If G has solvable word problem, and we have a procedure to determine whether or not a word of G would allow pinching, we have a procedure to solve the word problem, since we just perform pinches until we have reduced the word, and apply Britton's Lemma - if the word contains a T-symbol then we can conclude that the word cannot represent the identity, otherwise we just solve the word problem in G. Of course, there is no reason at all why this need be a context-sensitive procedure, even if G has context-sensitive word problem.

To give an indication of the sort of problems we may encounter, consider the Baumslag-Solitar group $B(1, 2) = \langle t, a : t^{-1}at = a^2 \rangle$. This is an HNN-extension of the group $G = \langle a \rangle$ with respect to the subgroups $H = \langle a \rangle$, and $K = \langle a^2 \rangle$. But in this group, $t^{-i}at^i = a^{2^i}$ and we have an exponential increase in the length of the word. Hence, we need to be very careful when dealing with these sort of relations (although this group does actually have deterministic context-sensitive word problem, as we shall note later).

The problem in this sort of situation is that the subgroups H and K have non-trivial intersection. In the case where $H \cap K = \{1\}$, then the situation is more favourable.

Lemma 4.1.3 *Suppose G is a group with isomorphic subgroups H and K, where $H \cap K = \{1\}$, and suppose G has (deterministic) context-sensitive word problem, and (deterministic) context-sensitive effective generalised word problem with respect to H and K. Let T be a set of stable letters. Then there is a (deterministic) context-sensitive procedure to produce a T-reduced form of any word in a Britton extension $G^*_{H,K,T}$ of G.*

Proof Let us take generators $\{h_1, ..., h_r\}$ and $\{k_1, ..., k_r\}$ for H and K respectively, and extend in the natural way to a generating set

$$X = \{h_1, ..., h_r, k_1, ..., k_r, g_1, ..., g_t\}$$

for G. Hence, if G is given by the presentation $\langle X : R\rangle$, then we have the usual presentation

$$G^*{}_{H,K,T} = \langle X \cup \{T\} : R \cup \{{t_j}^{-1} h_i t_k = k_i : 1 \leq i \leq r, \text{ some } t_j, t_k \in T\}\rangle$$

for a Britton extension of G with respect to H and K and stable letters T.

In this presentation, since $H \cap K = \{1\}$, we can assume that the h_i and k_i are distinct symbols.

Suppose we are given some input word u, which we can assume is freely reduced. If u contains no T-symbol, then it is obviously T-reduced. So, suppose we have a word containing an occurrence of a T-symbol.

Now, by assumption, G has (deterministic) context-sensitive word problem, and so we can certainly test all substrings not containing a T-symbol, and remove them if they are equivalent to the identity. Also, we know by assumption that we have (deterministic) context-sensitive procedures to reduce any subword in G to a word in the h_i or k_i if it lies in H or K respectively (of course, by assumption, it cannot lie in both, unless it is equivalent to the identity).

We can perform these procedures in turn, repeating as many times as necessary, until we can perform no more such operations. Note that this is a terminating procedure, since deletions shorten the word, and we only 'reduce' words containing at least one g_i, and so every reduction to H or K reduces the number of g_i in the word. Hence, we have a new word u' which is certainly linear in the length of the original word u (because our procedures are linearly bounded), and contains no trivial strings of G, or strings of words of G containing at least one g_i which lie in K or H.

We can now test to see whether this word is T-reduced. We simply check to see if, anywhere in the word, we have a pinchable subword (that is, a subword of the form ${t_i}^{-1}u_h t_i$, or $t_i u_k {t_i}^{-1}$, for some t_i, or a subword that is just a relator involving a T-symbol). If not, then the word is T-reduced, by definition.

Otherwise, we have such a subword, and we perform the pinching. In this pinching, each k_i is replaced by the corresponding h_i, or vice versa, and we remove a pair of T-symbols. Hence the word has been shortened.

We then repeat this whole procedure with our new word, until eventually we must reach a reduced word (note that this procedure must eventually terminate, since we have not added any extra g_i with any pinching, and so we can only repeat a finite number of times).

Clearly, the entire procedure is terminating, and operates in linear space, and hence is (deterministic) context-sensitive. □

Hence we have the following as a simple corollary.

Theorem 4.1.4 *Suppose G is a group with isomorphic subgroups H and K, where $H \cap K = \{1\}$, and suppose G has (deterministic) context-sensitive word problem and (deterministic) context-sensitive effective generalised word problem with respect to H and K. Let T be a set of stable letters. Then a Britton extension $G^*_{H,K,T}$ of G also has (deterministic) context-sensitive word problem.*

Proof Simply produce the T-reduced form of a word as in Lemma 4.1.3. If this contains a T-symbol, then we reject it, otherwise we test it for equivalence to the identity in G. □

This technique does not extend to the case where H and K have non-trivial intersection, since we cannot necessarily take our generating set to consist of distinct symbols, without being forced to consider additional relations. However, it is certainly possible in some circumstances that we can deduce a similar result when $H \cap K \neq \{1\}$. For example, consider again the Baumslag-Solitar group $B(1, 2)$. As we have commented, it is an HNN extension of the group $G = \langle a \rangle$ with respect to the subgroups $\langle a \rangle$ and $\langle a^2 \rangle$, and we will note later that it has deterministic context-sensitive word problem.

Now, G is a free group, so it has (deterministic) context-sensitive word problem by Lemma 2.5.3. Hence, we have an example of an HNN extension of a group preserving the property of having (deterministic) context-sensitive word problem, even though the specified subgroups have non-trivial intersection.

Note that in this instance we have (deterministic) context-sensitive procedures to decide the effective generalised word problem with respect to the subgroups in question (calculating whether a given word is equivalent to an even power of a is a simple question of binary arithmetic).

4.1.2 A brief note on the conjugacy problem

In general, the conjugacy problem appears to be a difficult area. It is shown in [36] that it is possible to have a group G with solvable conjugacy problem, but some HNN extension of G has unsolvable conjugacy problem (although the group given there is not finitely presented, and the question is left open for finitely presented groups).

Let $G^*_{H,K,T}$ be a Britton extension of a group G with respect to the stable letters T. Suppose $w \in G^*_{H,K,T}$. Then we say that w is *T-cyclically reduced* if every cyclic permutation of w is T-reduced. It is obvious that any word is conjugate to a T-cyclically reduced word, and in particular, is conjugate to a T-cyclically reduced word beginning with a T-symbol (since we simply conjugate to permute the word around).

Given a word w in $G^*_{H,K,T}$, the *T-projection* of w is simply the word obtained by removing all symbols not in T. Two words are said to be *T-parallel* if their T-projections are identical up to symbols being equivalent (that is, their T-projections are of the same length and symbols in corresponding positions are equivalent). We can also say that two words u and v are *T-circumparallel* if u and v' are parallel for some cyclic permutation v' of v.

We have the following result courtesy of [42].

Lemma 4.1.5 (Collins' Lemma) *Let $G^*_{H,K,T}$ be a Britton extension of a group G with respect to the stable letters T. Suppose u and v are T-cyclically* *reduced words, and suppose that u begins with ${t_j}^\epsilon$ where $\epsilon = \pm 1$. Then u and v are conjugate in $G^*_{H,K,T}$ if and only if there exists some cyclic permutation v' of v satisfying the following conditions.*

- *u is T-parallel to v',*
- *v' begins with ${t_k}^\epsilon$ (where t_j and t_k are equivalent),*
- *there exists some word x in G such that $u = x^{-1}v'x$, where x is a pinchable element of H if $\epsilon = 1$, and x is a pinchable element of K if $\epsilon = -1$.*

Note that we have an entirely equivalent form of this lemma when u ends with some ${t_j}^\epsilon$. Of course, since we are considering the conjugacy problem, we can always conjugate a given word u to ensure that it either begins or ends with a suitable ${t_j}^\epsilon$.

This lemma, on its own, says nothing about the possibilities for context-sensitive algorithms since we have no bound on the possible conjugating element x. However, in certain cases, it may be possible to bound the length of x and thus exhibit an argument for a Britton extension having context-sensitive conjugacy problem.

4.1.3 Some simple pinching lemmas

As a brief point, let us note a couple of very simple lemmas with regard to HNN extensions and pinchings. Given a presentation for an HNN extension, it is generally extremely difficult to determine if the process of performing pinchings to produce a reduced word is linearly bounded. Often in many cases where it intuitively might appear to be so, there is actually a quadratic or even steeper increase in the length of the word. For example, consider the following presentation:

$$G = \langle a, b, t : t^{-1}at = ab, t^{-1}bt = b \rangle$$

which forms an HNN extension of the free group on two generators, with the associated subgroups being the whole group. Thus, the effective generalised word problem for the associated subgroups is trivial - we simply freely reduce a word.

Intuitively, these sort of relations might appear to be amenable to a linear space algorithm, but consider a word such as $t^{-n}a^n t^n$. Applying the pinchings gives us successively $t^{n-1}(ab)^n t^n$, $t^{n-2}(ab^2)^n t^{n-2}$, and so on up to $(ab^n)^n$, which is of length $(n+1)n$ and hence is of length quadratic in the length of the original word, which is $3n$.

An obvious example of when things work out satisfactorily is when none of the pinchings alter the length of a word.

Lemma 4.1.6 *Suppose we have a group G^*, which is a Britton extension of a group G with stable letters T, with respect to the subgroups H and K. Suppose further that every pinching relation is of the form ${t_{j_1}}^{-1}ut_{j_2} = v$, where both $t_{j_1}, t_{j_2} \in T$, u and v are words in G, and $|u| = |v|$, and suppose there is a non-length-increasing procedure to decide the effective generalised word problem for H and K (with the given set of generators in the pinching relations) in G. Then there is a deterministic context-sensitive procedure to T-reduce a word in G^*.*

Proof We simply apply Britton's Lemma and perform the required pinchings, reducing any word to an appropriate word in the subgroups if necessary. No procedure ever increases the length of the word and so we are done. $\square$

The insistence that the procedure for the effective generalised word problem is non-length increasing (rather than just being linear) is important -

note that if (for example) the procedure doubled the length of a subword, then applying it many times could lead to a superlinear blow-up in the length of the word. However, provided we choose our generating sets carefully, this should not be a problem.

This lemma covers the particular case where every non-T symbol appears the same number of times on both sides of every relation. In general, if this is not the case, then we have to be very careful to track the number of occurrences of a symbol. However, there is one useful, non-obvious, case in which things do work out satisfactorily. Although the following statement looks rather lengthy, the important point is that a symbol only appears on one side, and in only finitely many, of the defining relations.

Lemma 4.1.7 *Suppose we have a group G^* which is a Britton extension of a group G with stable letters T, with respect to the subgroups H and K. Suppose further that every pinching relation is of the form ${t_{j_1}}^{-1}u_it_{j_2} = v_i$, where both $t_{j_1}, t_{j_2} \in T$, and the symbol x either never appears inside a u_i, or never appears inside a v_i, and also appears in only finitely many relations. Suppose, in addition, that there is a linearly bounded procedure to decide the effective generalised word problem for H and K (with the given set of generators in the pinching relations) in G which never increases the number of x in any word already containing an x. Then there is a linear bound on the number of x that can appear in any word of the computation to reduce any a word.*

Proof Without loss of generality, assume that all of our relations are of the form ${t_{j_1}}^{-1}u_it_{j_2} = v_i$, where both $t_{j_1}, t_{j_2} \in T$, and no u_i contains an occurrence of the symbol x (so x only appears on the right-hand side of our relations).

Suppose the maximum number of occurrences of x in any of the v_i is k. Given a word u of length n, the maximum number of x that can be added to

u in the reduction process via pinchings is bounded by kn (since any pinching on a word containing an x eliminates this x, and there are no more than n pinchings that could be done).

Similarly, the procedure for the effective generalised word problem can add at most cn extra occurrences of x (for some c) by our assumptions that this procedure is context-sensitive, and does not increase the number of x in a word already containing an x.

Finally, there are at most n occurrences of x in the original word, and thus the total number of x in any step of the calculation is bounded by $(k+c+1)n$ and we are done. $\square$

Clearly, used in combination with Lemma 2.1.1, this can be a useful observation, and we will see this later.

4.2 The word problem and extensions of finite index

Suppose we have a group H, contained inside a subgroup G, where $[G : H]$ is finite, that is H is of *finite index* in G. We wish to show that the property of having (deterministic) context-sensitive word problem is closed under taking such an extension. Firstly, let us consider this situation with a restriction on the subgroup H.

Lemma 4.2.1 *Suppose G and H are groups where $H \lhd G$ (that is, H is a normal subgroup of G), and $[G : H] = s$ for some integer s. Then G has (deterministic) context-sensitive word problem if and only if H has (deterministic) context-sensitive word problem.*

Proof Firstly, if G has (deterministic) context-sensitive word problem, then so does H by Lemma 2.2.6.

Conversely, take a generating set $Y = \{h_1, ..., h_r\}$ for H, and extend this to a generating set $Z = \{h_1, ..., h_r, g_1, ...g_s\}$ for G, where $X = \{g_1, ..., g_s\}$ is a set of coset representatives for G with respect to H. The informal idea is that, since $H \lhd G$, we can form the quotient group G/H (which is finite by assumption), and then we find a coset representative for our word u. If this representative is the identity, then u lies in H, and so we can check u for equivalence to the identity, since H has (deterministic) context-sensitive word problem. If the representative is not the identity, then u cannot represent the identity and we are done.

So, let us spell out the details of this procedure. There are only finitely many elements of X and Y, and so there are only finitely many elements ${g_i}^{-1}h_jg_i$. Since $H \lhd G$, each of these elements lies in H. Hence, we can take a fixed list of words $v_i \in Y^*$ representing each of these elements, and we fix $K_1 = \max |v_i|$. Similarly, there are only finitely many elements g_ig_j. Each of these is a word of G, and so can be written in the form gw, where $g \in X$, and $w \in Y^*$, since we chose X to be a set of coset representatives for G with respect to H. So, again, we can take a fixed list of words g_kw_l representing each of these elements, and we fix $K_2 = \max |w_l|$.

Suppose $a, b \in X$ and $\alpha, \beta \in Y^*$. Let $u = a\alpha b\beta$ be of length n. Suppose $\alpha \equiv h_1.....h_p$. Then

$$b^{-1}\alpha b = b^{-1}h_1bb^{-1}h_2....b^{-1}h_pb = v_{i_1}....v_{i_p}$$

for some i_j. Therefore,

$$u = ab(b^{-1}\alpha b)\beta = g_kw_lv_{i_1}....v_{i_p}\beta$$

for some g_k, w_l, v_{i_k}. Now, the length of this equivalent word is clearly bounded by $1+K_2+K_1|\alpha|+|\beta|$ which is certainly no more than $(2+K_1+K_2)n$ and so this is a deterministic context-sensitive procedure to produce the equivalent word.

Inductively, it is then clear that we can write any word $u = a_1\alpha_1.....a_m\alpha_m$ (with $a_i \in X, \alpha_i \in Y^*$) in the form $g_k\gamma$, with $g_k \in X$ and $\gamma \in Y^*$, with the length of this equivalent word bounded by

$$(m-1) + K_2(m-1) + K_1(|\alpha_1| + ... + |\alpha_{m-1}|) + |a_m|$$

which is again certainly less than $(2 + K_1 + K_2)n$, where u is of length n.

Hence, given any input word u, we first check to see if it contains any occurrences of the g_i, since if not then it lies in H and we check to see if it equals the identity.

Otherwise, we have that u is of the form $g_{i_1}\alpha_1....g_{i_m}\alpha_m$ (conjugating if necessary to bring g_{i_1} to the front). From the above result, this is equivalent to a word $u' = g_r\eta$ for some $g_r \in X$ and $\eta \in Y^*$, where the length of this word is linearly bounded in the length of u.

And hence u is equivalent to the identity if and only if $g_j = 1$ and η is equivalent to the identity in H, which we can check using the (deterministic) context-sensitive procedure for the word problem in H. This procedure is clearly (deterministic) context-sensitive and hence we are done. □

From this result we can deduce the required result for general subgroups of finite index.

Theorem 4.2.2 *Let G and H be groups, where $H \leq G$, and $[G : H] = s$ for some integer s. Then G has (deterministic) context-sensitive word problem if and only if H has (deterministic) context-sensitive word problem.*

Proof If G has (deterministic) context-sensitive word problem, then so does H by Lemma 2.2.6.

Conversely, from standard group theory (see for example [52]), given a group G with a subgroup H of finite index, there exists a finitely generated normal subgroup N of G, with $N \leq H \leq G$ and $[G : N]$ finite. Then N has (deterministic) context-sensitive word problem (since it is a subgroup of H) by Lemma 2.2.6, and hence we can use Lemma 4.2.1 to conclude that G has context-sensitive word problem. □

4.3 The conjugacy problem and extensions of finite index

We have shown in the previous section that the property of a group having context-sensitive word problem is preserved under taking extensions of finite index. The obvious next step is to ask about the conjugacy problem. It is already known that, in contrast to the situation for the word problem, the property of having solvable conjugacy problem is not preserved under taking extensions of finite index (see [15]), indeed the extension given there is of index 2. Here we strengthen this result, and show the somewhat remarkable result that there exists a group with deterministic context-sensitive conjugacy problem which is a subgroup of index 2 of a group with unsolvable conjugacy problem.

The group that we will use to demonstrate this result is that of [23], and our proof is essentially the proof there. However, we take a rather different slant, choosing to use a set-theoretic approach rather than the approach in terms of ideals used therein, since this fits more naturally with our work here, and we consider that it is much easier to consider sets of words, rather than ideals, for the approach we wish to take. The majority of the work is done in showing that the preliminary work in producing a canonical form of a word, and the preliminary lemmas, can all be performed in linear space, since the approach in [23] most certainly does not use linear space. We do not choose to give proofs of some results from [23], where they have no bearing on the complexity of the procedure. Although these results can relatively easily be adapted to our approach here, we feel that this makes our proof unnecessarily long.

4.3.1 Defining the groups

Let us define the groups that will form the basis of our result. The groups are precisely those of [23], but we need to analyse the structure slightly further.

It is well known (originally proved in [41]) that, given two free groups X and Y of rank N, (where $N \geq 2$) there exists a finitely generated subgroup R of $X \times Y$ such that the generalised word problem of $X \times Y$ with respect to R is unsolvable. Suppose $N = 2$, and consider a suitable subgroup R. Let M be the rank of R (that is, the number of generators in a minimal generating set for R - we will assume this a monoid generating set for convenience).

Throughout this section, the notation G_{CS} will refer to the following group (the abbreviation CS refers to context-sensitive conjugacy problem).

We take as generators of G_{CS} the symbols

$$a, b, x_i, x_i{}', y_i, y_i{}', z_j, t : 1 \leq i \leq 2, 1 \leq j \leq M$$

which are subject to the relations

$$a^2 = b^2 = 1, ab = ba,$$

$$x_i x_k' = x_k' x_i, x_i y_k = y_k x_i, x_i y_k' = y_k' x_i, x_i' y_k = y_k x_i', x_i' y_k{}' = y_k' x_i', y_i y_k' = y_k' y_i,$$

$$bx'_i = x'_i b, by'_i = y'_i b,$$

$$x_i{}^{-1} a x_i = (x'_i)^{-1} a x'_i, y_i{}^{-1} a y_i = (y'_i)^{-1} a y'_i,$$

$$z_j a = a z_j, z_j b = b z_j,$$

$$z_j x'_i = x'_i z_j, z_j y'_i = y'_i z_j,$$

$$t x_i = x_i t, t y_i = y_i t, t z_j = z_j t, tb = bt$$

$$t^{-1} a t = ab.$$

At first glance, this may seem a complicated group presentation, but in fact most of the relations are simple commuting relations. The initial cause for concern appears from the relation $t^{-1}at = ab$, a similar relation to which we have already noted (at the start of Section 4.1.3) can cause problems with quadracity. However, we can overcome this problem.

Let X be the free group on the x_i, and similarly we can define Y, X', Y', Z and T to be the free groups on the y_i, x'_i, y'_i, z_j and t respectively. Exactly as in [23], we can define A to be the (non finitely-presented) group which is generated by the free product of the cyclic groups of order 2, which are of the form $\langle w^{-1}aw \rangle$, for every w which is a word of $X \times Y$. Similarly we define B to be the (non finitely-presented) group which is generated by the direct product of the cyclic groups of order 2, which are of the form $\langle w^{-1}bw \rangle$, again for every w which is a word of $X \times Y$. Let C be the group $((X \times Y) * Z) \times ((X' \times Y') * T)$. Then we have the following lemma, courtesy of [23]. We do not feel the need to replicate the proof here.

Lemma 4.3.1 *G_{CS} is isomorphic to a semidirect product $(A \times B) \rtimes C$.*

Proof See [23]. $\square$

This helps us to understand the structure of G_{CS}, but let us look at this in more detail. Usually, we will use notation such as w to denote a word in

$X \times Y$, and the dashed notation w' to denote a word in $X' \times Y'$. For a word w', let $\overline{w'}$ denote the word obtained by replacing each x_i' by x_i, each y_i' by y_i, and then reversing the word. We define $\overline{w}$ for a word w in the x_i and y_i similarly. Note that $\overline{\overline{w'}} = w'$, and that for a single symbol x_i', $\overline{x_i'}$ is simply the symbol x_i, and similarly for the other symbols. Also note that for any two words w_1, w_2, then $\overline{w_1 w_2} = \overline{w}_2 \overline{w}_1$. We will also use notation such as u_Z for a word in Z, and similarly for other symbols. This sort of notation will be used without comment unless it is explicitly stated otherwise. Note that since $a^2 = 1$, we do not need to consider words in the a, since such words must be either empty, or equivalent to a, and similarly for b.

Lemma 4.3.2 *In G_{CS}, the following relations are satisfied for any words w and w':*

(i) $(ww')^{-1}a(ww') = (\overline{w'}w)^{-1}a(\overline{w'}w)$,

(ii) $(ww')^{-1}b(ww') = w^{-1}bw$,

(iii) $(ww')^{-1}u_Z(ww') = w^{-1}u_Z w$,

(iv) $(ww')^{-1}u_T(ww') = (w')^{-1}u_T(w')$.

Proof Let $w' = w_{i_1}'....w_{i_s}'$. Then

$$(ww')^{-1}a(ww') = (w_{i_1}'....w_{i_s}')^{-1}w^{-1}aw(w_{i_1}'....w_{i_s}')$$

$$= w^{-1}(w_{i_1}'....w_{i_s}')^{-1}a(w_{i_1}'....w_{i_s}')w = w^{-1}(w_{i_s}')^{-1}....(w_{i_1}')^{-1}aw_{i_1}'....w_{i_s}'w$$

$$= w^{-1}(w_{i_s}')^{-1}....(\overline{w_{i_1}'})^{-1}a\overline{w_{i_1}'}....w_{i_s}'w$$

by applying either the relation $x_i{}^{-1}ax_i = (x_i')^{-1}ax_i'$, or the relation $y_i{}^{-1}ay_i = (y_i')^{-1}ay_i'$. We can now use the commuting relations of our group G_{CS} to show immediately that this is equivalent to

$$w^{-1}(\overline{w_{i_1}'})^{-1}(w_{i_s}')^{-1}....(w_{i_2}')^{-1}aw_{i_2}'....w_{i_s}'\overline{w_{i_1}'}w$$

and, continuing in this vein, we obtain the word

$$w^{-1}(\overline{w'_{i_1}})^{-1}....(\overline{w'_{i_s}})^{-1}a\overline{w'_{i_s}}....\overline{w'_{i_1}}w = w^{-1}(\overline{w'_{i_1}....w'_{i_s}})^{-1}a(\overline{w'_{i_1}....w'_{i_s}})w$$

$$= (\overline{w'}w)^{-1}a(\overline{w'}w)$$

which gives us (i).

The remaining relations (ii), (iii) and (iv) follow trivially from the commuting relations in G_{CS}. $\square$

Lemma 4.3.3 *In G_{CS}, the following relations are satisfied for any words w_1, w_2, w', u_Z, and any integer k:*

(i) $(w_1^{-1}bw_1)(w_2^{-1}aw_2) = (w_2^{-1}aw_2)(w_1^{-1}bw_1)$,

(ii) $(w_1^{-1}u_Zw_1)(w_2^{-1}aw_2) = (w_2^{-1}aw_2)(w_1^{-1}u_Zw_1)$,

(iii) $(w_1^{-1}bw_1)(w_2^{-1}bw_2) = (w_2^{-1}bw_2)(w_1^{-1}bw_1)$,

(iv) $(w_1^{-1}u_Zw_1)(w_2^{-1}bw_2) = (w_2^{-1}bw_2)(w_1^{-1}u_Zw_1)$,

(v) $((w')^{-1}t^kw')(w_2^{-1}bw_2) = (w_2^{-1}bw_2)((w')^{-1}t^kw')$,

(vi) $((w')^{-1}t^kw')(w_2^{-1}u_Zw_2) = (w_2^{-1}u_Zw_2)((w')^{-1}t^kw')$,

(vii) $((w')^{-1}t^kw')(w_2^{-1}aw_2) =$

- $(w_2^{-1}aw_2)((w')^{-1}t^kw')$ *(for k even),*
- $(w_2^{-1}aw_2)((w')^{-1}t^kw')(((\overline{w'})^{-1}w_2)^{-1}b((\overline{w'})^{-1}w_2))$ *(for k odd).*

Proof We will use Lemma 4.3.2 and the commuting relations of G_{CS}. We have

$$(w_1^{-1}bw_1)(w_2^{-1}aw_2) = w_1^{-1}b(w_1w_2^{-1}aw_2w_1^{-1})w_1$$

$$= w_1^{-1}b(w_2w_1^{-1})^{-1}a(w_2w_1^{-1})w_1 = w_1^{-1}b(\overline{w_2w_1^{-1}})^{-1}a(\overline{w_2w_1^{-1}})w_1$$

by Lemma 4.3.2(i). But then, since b commutes with the x_i', the y_i', and a, this is equivalent to

$$w_1^{-1}(\overline{w_2w_1^{-1}})^{-1}a(\overline{w_2w_1^{-1}})bw_1.$$

Again using Lemma 4.3.2(i), this is equivalent to

$$w_1^{-1}(w_2w_1^{-1})^{-1}a(w_2w_1^{-1})bw_1 = (w_2^{-1}aw_2)(w_1^{-1}bw_1)$$

which gives us the required result for (i). The result for (ii) follows in exactly the same way, since the z_i also commute with the x_i', the y_i', and a.

For (iii), we consider the word $t^{-1}(w_1^{-1}aw_1)(w_2^{-1}bw_2)t$. On the one hand we have the following.

$$t^{-1}(w_1^{-1}aw_1)(w_2^{-1}bw_2)t = (w_1^{-1}abw_1)t^{-1}(w_2^{-1}bw_2)t$$

$$= (w_1^{-1}aw_1)(w_1^{-1}bw_1)(w_2^{-1}bw_2).$$

But on the other hand,

$$t^{-1}(w_1^{-1}aw_1)(w_2^{-1}bw_2)t = t^{-1}(w_2^{-1}bw_2)(w_1^{-1}aw_1)t$$

from (i), and this is equivalent to

$$(w_2^{-1}bw_2)(w_1^{-1}abw_1) = (w_1^{-1}aw_1)(w_2^{-1}bw_2)(w_1^{-1}bw_1)$$

again using (i). And hence,

$$(w_1^{-1}aw_1)(w_1^{-1}bw_1)(w_2^{-1}bw_2) = (w_1^{-1}aw_1)(w_2^{-1}bw_2)(w_1^{-1}bw_1)$$

which gives us

$$(w_1^{-1}bw_1)(w_2^{-1}bw_2) = (w_2^{-1}bw_2)(w_1^{-1}bw_1)$$

as required for (iii). An entirely similar approach, instead considering the word $t^{-1}(w_2^{-1}aw_2)(w_1^{-1}u_Z w_1)t$ and using part (ii), gives us (iv).

Both (v) and (vi) are trivial using the commuting relations in G_{CS}, which leaves only (vii). Consider the word $((w')^{-1}t^k w')(w_2^{-1}aw_2)$. Rearranging, and using Lemma 4.3.2(i), we have

$$((w')^{-1}t^k w')(w_2^{-1}aw_2) = (w')^{-1}t^k(w_2(w')^{-1})^{-1}a(w_2(w')^{-1})w'$$

$$= (w')^{-1}t^k((\overline{w'})^{-1}w_2)^{-1}a((\overline{w'})^{-1}w_2)w'.$$

Using the relation $t^{-1}at = ab$, and the commuting relations, this word is equivalent to

$$(w')^{-1}((\overline{w'})^{-1}w_2)^{-1}ab^k((\overline{w'})^{-1}w_2)t^k w'$$

$$= (w')^{-1}((\overline{w'})^{-1}w_2)^{-1}a((\overline{w'})^{-1}w_2)((\overline{w'})^{-1}w_2)^{-1}b^k((\overline{w'})^{-1}w_2)t^k w'.$$

We can now use Lemma 4.3.2(i) to simplify this word to

$$(w')^{-1}(w_2(w')^{-1})^{-1}a(w_2(w')^{-1})((\overline{w'})^{-1}w_2)^{-1}b^k((\overline{w'})^{-1}w_2)t^k w'$$

and then applying the commuting relations, and part (v), we obtain

$$(w_2^{-1}aw_2)((w')^{-1}t^k w')(((\overline{w'})^{-1}w_2)^{-1}b^k((\overline{w'})^{-1}w_2)).$$

But since $b^2 = 1$, if k is even then the whole of the final factor cancels, and if k is odd, then $b^k = b$, which gives us the two cases in (vii). $\square$

Given G_{CS}, we can now define our second group, G_U. Recall that we chose M to be the rank of a subgroup R of $X \times Y$ such that the generalised word problem of $X \times Y$ with respect to R is undecidable. Suppose R is generated by $r_1, ..., r_M$. We can now consider the group G_U (where U refers to an unsolvable conjugacy problem), where G_U is the group given by the generators and relations of G_{CS}, together with the generator ψ, and relations

$\psi^{-1}z_i\psi = z_i b r_i{}^{-1} b r_i$ for $1 \leq i \leq M$, the relations $\psi^{-1}\kappa\psi = \kappa$ for all other symbols κ, and the relation $\psi^2 = 1$.

Lemma 4.3.4 *G_{CS} is a subgroup of index 2 of G_U.*

Proof See [23]. □

The fact that G_U has unsolvable conjugacy problem is courtesy of [23], and we again see no reason to replicate the proof here.

Theorem 4.3.5 *G_U has unsolvable conjugacy problem.*

Proof See [23]. □

4.3.2 The set-theoretic approach and preliminary lemmas

Having defined the groups that will form the basis of our result, let us now set down the set-theoretic approach that we will take, and give some preliminary lemmas. It is here that the majority of the work is done, in order to emphasise the linearly bounded nature of our procedure.

Suppose $\mathcal{B} = \{\beta_1....\beta_m\}$ is a finite set of words of $X \times Y$. In the standard way, we can then denote the word $\beta_1^{-1}b\beta_1....\beta_m^{-1}b\beta_m$ by

$$\prod_{\beta \in \mathcal{B}} \beta^{-1}b\beta.$$

For any word g of $X \times Y$, we will use the notation $\mathcal{B}g$ to denote the set of words $\{\beta_i g : 1 \leq i \leq m\}$.

We will assume that any word is freely reduced, and any word in the direct product $X \times Y$ is of the form $w_X w_Y$ with $w_X \in X$, $w_Y \in Y$, since this can always be done, simply by rearranging the symbols.

Given two sets $\mathcal{B}_1$ and $\mathcal{B}_2$, the *symmetric difference* $\mathcal{B}_1 \triangle \mathcal{B}_2$ is defined as the set $(\mathcal{B}_1 \cup \mathcal{B}_2) - (\mathcal{B}_1 \cap \mathcal{B}_2)$.

We note a few simple points with regard to the symmetric difference. Obviously, for any set $\mathcal{B}$, we have

$$\mathcal{B} \triangle \mathcal{B} = \emptyset,$$

$$\mathcal{B} \triangle \emptyset = \mathcal{B}.$$

Also, we note the commutativity and associativity of the symmetric difference. For example, for any word u, and sets $\mathcal{B}_1$ and $\mathcal{B}_2$, we have that

$$\mathcal{B}_1 u \triangle \mathcal{B}_2 u = (\mathcal{B}_1 \triangle \mathcal{B}_2) u = (\mathcal{B}_2 \triangle \mathcal{B}_1) u.$$

In our exposition in this chapter, we will often consider sets such as the set

$$\mathcal{B}_p = \{u^i : 0 \leq i \leq (p-1)\}.$$

Note that then, in this sort of case, we have

$$\mathcal{B}_p \triangle \mathcal{B}_p u = \{1, u^p\}$$

since all other elements are repeated, and hence cancel in the symmetric difference.

There are many occasions when some sort of reformulation, using the symmetric difference, can help us towards a result. We will generally make such reformulations without further explanation, where it is clear that we are just using the symmetric difference to rewrite the set.

Now, with regard to our group G_{CS}, note that we have

$$\prod_{\beta_1 \in \mathcal{B}_1} \beta_1^{-1} b \beta_1 \prod_{\beta_2 \in \mathcal{B}_2} \beta_2^{-1} b \beta_2 = \prod_{\beta_3 \in \mathcal{B}_3} \beta_3^{-1} b \beta_3,$$

where $\mathcal{B}_3 = \mathcal{B}_1 \triangle \mathcal{B}_2$, since the conjugates all commute by Lemma 4.3.3, and since $b^2 = 1$, any 'duplicated' conjugates which occur in both $\mathcal{B}_1$ and $\mathcal{B}_2$ simply cancel, and can be ignored.

We will use this set-theoretic notation only for conjugates of b, since this stresses that the conjugates of b commute, and hence our conjugating words can be given in any order.

We wish to be able to consider any element of G_{CS} in some equivalent canonical form.

Lemma 4.3.6 *Any word w of G_{CS} is essentially uniquely equivalent (by which we mean that $\alpha_i \neq \alpha_{i+1}$ for any i) to a word of the form*

$$(\alpha_1^{-1} a \alpha_1) \ldots (\alpha_m^{-1} a \alpha_m) \left(\prod_{\beta \in \mathcal{B}} \beta^{-1} b \beta \right) \nu_1 \nu_2$$

where each α_i is a word of $X \times Y$, $\mathcal{B}$ is a set of words of $X \times Y$, ν_1 is a word of $(X \times Y) * Z$, and ν_2 is a word of $(X' \times Y') * T$. In addition, each α_i, each element of $\mathcal{B}$, ν_1 and ν_2 are all of length no greater than $|w|$, and can be explicitly determined from the original word.

Proof The existence, and the essential uniqueness, of an equivalent word in the given form follows immediately from Lemma 4.3.1. Hence, we merely need to ensure that we can determine each 'component' of the word in space bounded by $|w|$. We will produce explicity an appropriate word.

We start of course by freely reducing w. Hence, we can assume that

$$w = w_1 \xi_1 \ldots w_r \xi_r$$

where each ξ_i is either the symbol a, the symbol b, a word of Z, or a word of T, and each w_i is a word of $(X \times Y) \times (X' \times Y')$. We can rearrange the symbols of w to ensure that each w_i is a word of the form $w_X w_Y w_{X'} w_{Y'}$, with

w_X in X, and similarly for the other words. This word is clearly equivalent to

$$(w_1\xi_1 {w_1}^{-1})((w_1w_2)\xi_1(w_1w_2)^{-1})....((w_1w_2....w_r)\xi_t(w_1w_2....w_r)^{-1})w_1....w_r$$

and hence we can express w as a product of conjugates, followed by some word. Note that we cannot write this word down in linear space, but we can obviously deduce each of the strings $w_1....w_j$, since each one is obtained by taking the prefix of w which consists of the symbols of w up to ξ_j, and then projecting this word onto $(X \times Y) \times (X' \times Y')$ - that is, we just write down in sequence all the occurrences of x_i, y_i, x'_i and y'_i until we reach ξ_j, and then rearrange these symbols. Note that, by our definition of a conjugating element, the actual conjugating element in each case is then obtained by inverting the word $w_1....w_j$ thus obtained.

Using Lemma 4.3.2, we can now reduce each of the conjugating elements to words in $X' \times Y'$ (if the corresponding ξ_i is a word in T), or words in $X \times Y$ otherwise. Note that this procedure never increases the length of any of the conjugates.

We can now use Lemma 4.3.3 to rearrange all these conjugates, so that the conjugates of a come first, followed by the conjugates of b, followed by the conjugates of Z, followed by the conjugates of T. In doing this, note that this adds additional conjugates of b to our word.

It is clear that the conjugates of a remain unchanged. Hence, if $\xi_{r_1}, ..., \xi_{r_m}$ are the ξ_i which consist of a, then the conjugates of a are precisely the words $\alpha_1, ..., \alpha_m$, where each α_i is obtained by writing down the word $w_1...w_{r_i}$, rearranging the symbols, inverting the word, and then performing the transformation in Lemma 4.3.2 (which does not change the length of the word). Hence, each of these α_i is of length no greater than $|w|$, and can be explicitly determined.

The conjugates of T and Z are also unchanged. For any word u, let $u_{X,Y}$ be the word obtained by projecting onto $X \times Y$, that is simply writing the X-symbols and Y-symbols of u in order, and similarly for $u_{X',Y'}$.

Suppose $\xi_{s_1}, ..., \xi_{s_l}$ are the ξ_i which consist of Z-symbols. Then the conjugating elements of these ξ_i are precisely the words $\gamma_1, ..., \gamma_l$, where each γ_i is obtained by writing down the word $w_1...w_{s_i}$, eliminating the x_i' and y_i', and then inverting. It is clear that this gives us that the word ν_1 is equivalent to

$$(\gamma_1^{-1}\xi_{s_1}\gamma_1)(\gamma_2^{-1}\xi_{s_2}\gamma_2)....(\gamma_l^{-1}\xi_{s_l}\gamma_l)(w_1....w_r)_{X,Y}$$

which in turn gives us

$$(w_1....w_{s_1})_{X,Y}\xi_{s_1}....(w_{s_{l-1}+1}...w_{s_l})_{X,Y}\xi_{s_l}(w_{s_l+1}...w_r)_{X,Y}$$

and so ν_1 is obtained simply by writing the X-symbols, Y-symbols, and Z-symbols of w in order, and so is obviously of length no greater than $|w|$.

An entirely similar argument applies to ν_2, which can be obtained by writing the X'-symbols, Y'-symbols, and T-symbols of w in order, and so is also obviously of length no greater than $|w|$.

We are left, finally, to consider the b, and to form our set $\mathcal{B}$. Obviously, we have the conjugating words that already exist. If $\xi_{t_1}, ..., \xi_{t_k}$ are the ξ_i which consist of b, then the conjugating words of b already present are precisely the words $\beta_1, ..., \beta_m$, where each β_i is obtained by writing down the word $w_1...w_{t_i}$, and removing the x_i' and y_i', and inverting. We are left to consider the additional conjugates of b which can be obtained in our rearrangement of the word.

To find these, we simply need to consider, in turn, every conjugate of T which appears earlier in the word than a conjugate of a. It is easy to consider every such pair of conjugates in turn, simply by examing every possible pair of ξ_i. The number of such additional terms is certainly bounded by $|w|^2$ as a crude upper bound, which can be stored in linear space in binary

If the power of t in some ξ_i is even, then we need do nothing. Otherwise, suppose ξ_p is an odd power of t, and we have the corresponding conjugate $\mu_p^{-1}t^{2i_p+1}\mu_p$, where $\mu_p \in X' \times Y'$. For every conjugate $\alpha_q^{-1}a\alpha_q$ appearing later in the word, we obtain, from Lemma 4.3.3, an additional term

$$((\overline{\mu_p})^{-1}\alpha_q)^{-1}b((\overline{\mu_p})^{-1}\alpha_q).$$

Suppose that $w_1....w_p$ is given by the word $(\delta_1)_{X,Y}(\delta_1)_{X',Y'}$, in the usual way. Then $\mu_p = ((\delta_1)_{X',Y'})^{-1}$, by Lemma 4.3.2 (remember that we need to invert our word to gain the conjugating word). Since $q > p$, we can write $w_1....w_q = (\delta_1)_{X,Y}(\delta_1)_{X',Y'}(\delta_2)_{X,Y}(\delta_2)_{X',Y'}$. Then, by Lemma 4.3.2,

$$\alpha_q = \overline{((\delta_1\delta_2)_{X',Y'})^{-1}}((\delta_1\delta_2)_{X,Y})^{-1}.$$

Hence, using Lemma 4.3.3, the additional conjugating word of b that we obtain is given by

$$(\overline{(\delta_1)_{X',Y'}})\overline{((\delta_1\delta_2)_{X',Y'})^{-1}}((\delta_1\delta_2)_{X,Y})^{-1}.$$

But since $\overline{w_1w_2} = \overline{w}_2\overline{w}_1$ for any two words, then we have that

$$\overline{((\delta_1\delta_2)_{X',Y'})^{-1}} = \overline{((\delta_2)_{X',Y'})^{-1}((\delta_1)_{X',Y'})^{-1}} = \left(\overline{((\delta_1)_{X',Y'})^{-1}}\right)\left(\overline{((\delta_2)_{X',Y'})^{-1}}\right),$$

and hence this leaves us simply with the conjugating word

$$\overline{((\delta_2)_{X',Y'})^{-1}}((\delta_1\delta_2)_{X,Y})^{-1}.$$

Hence, we can write down the additional conjugate $\overline{\mu_p}\alpha_q^{-1}$ of b simply by writing the occurrences of X' and Y' in $w_{p+1}....w_q$, inverting the word, and performing the usual transformation. Finally, we then write down the X and

Y symbols in $w_1....w_q$, and invert the word. It is obvious that the number of symbols here is no greater than the length of the word, since each symbol is considered at most once. Hence, each additional conjugate is of length no greater than $|w|$, and can be explicitly given. Note that the total number of occurrences of b in our equivalent word can be explicitly given, and is certainly bounded by $|w| + |w|^2$ (counting the original occurrences, plus any additional ones) which can be stored, using binary, in linear space. It is clear that we can then determine, and write down, every element of $\mathcal{B}$ in turn in linear space, which concludes the proof. □

The important point of all this is that, although we cannot write down the canonical form of a given word in linear space, we can encode the canonical form in such a way that we can store it, since we can deduce all the properties of the word from our original word in linear space. By this, essentially, we mean that given w, we can determine any particular component of the canonical form in linear space. For example, if we wish to consider the set $\mathcal{B}$, then we can consider every element of it in turn in linear space, simply by deleting a word once it has been considered, and reusing the space, over and over again. We will now use this fact without comment in the following exposition.

In actual fact, when considering general sets of words, there are also instances where even a word which is not linearly bounded in length can still be considered. For example, we can 'store' the word w^m, where w is of length $O(n)$, and m is an integer of size $O(2^n)$, by storing w on one tape, and storing m in binary on another tape, even though the word w^m may be much longer than linear in length. Provided we are careful, we can then do simple operations on such words too.

With this in mind we define the concept of a linearly expressible set.

Suppose that u is some input word, of length n, and suppose that $\mathcal{B}$ is a finite set of words of size $O(2^n)$, deducible in turn from our input word u. Suppose further that there exists a constant c (that is, independent of the length of our input word), such that every element of $\mathcal{B}$ can be formed by the concatenation of no more than c words, where each word is of the form w^m, where w is of length $O(n)$ and m is an integer of size $O(2^n)$ (of course, if $m = 1$ then the word is linearly bounded anyway). Then we say that $\mathcal{B}$ is *linearly expressible* with respect to u. The bound on the number of elements of our set is necessary since we will wish to enumerate the elements in linear space. Often, our input word will be obvious (in the main proof which follows, the input is always the two words we take as our initial input, which we wish to test for conjugacy), and so in this case we can omit reference to u. The idea behind this, obviously, is that we can store any linearly expressible word in linear space. The constant c is crucial, as without this, there is no bound on the number of tapes possibly needed to store the components of our words.

Note the following simple lemma.

Lemma 4.3.7 *Suppose we are given any two linearly expressible words u, v of $X \times Y$, with length bounded by n. Then there is a procedure, operating in space $O(n)$, to determine whether or not $u = v$ in $X \times Y$.*

Proof By assumption, u and v are words over $X \times Y$. Hence, to test for equivalence, we merely need to freely reduce each component of u and v, and then by Lemma 2.5.2, we need to test only that the words remaining in each component are identical.

To prove that we can freely reduce a word in linear space, we proceed inductively. Suppose we have a linearly expressible word $w_1^{m_1} w_2^{m_2}$, where we assume each w_i is cyclically reduced. We wish to simulate the usual free cancellation in $X \times Y$. We begin by projecting w_1 and w_2 onto X. We then freely reduce the word thus obtained. If (for example) the whole of w_2 completely cancels in this reduction, then we reduce the value of m_2 by one, and repeat our procedure on this new word. We continue until either no more cancellation can be performed (that is, there is either no cancellation, or only part of a word cancels), or one of the m_i reaches zero, in which case we stop. We then have an equivalent word stored in the form $w_1^i w' w_2^j$ where no more cancellation can occur. This word in X is clearly freely reduced and still linearly expressible. Inductively we can then freely reduce the X-component of any linearly expressible word, and the resulting word is still linearly expressible.

We can then verify whether or not the resulting X-components of u and v are identical, and similarly for the Y-components, and we are done. $\square$

The following lemma is now a simple corollary.

Lemma 4.3.8 *Suppose u is a word of $X \times Y$ of length n, and suppose $\mathcal{A}$ and $\mathcal{B}$ are linearly expressible sets of words of $X \times Y$ with respect to u. Then the set $\mathcal{A} \triangle \mathcal{B}u$ is linearly expressible.*

Proof If $\mathcal{B}$ is linearly expressible, then so is $\mathcal{B}u$. Hence we simply write down each element of $\mathcal{A}$, and then each element of $\mathcal{B}u$, all of which are linearly expressible. For every word v we write down, we consider in turn every other element, and use Lemma 4.3.7 to check if a word equivalent to v appears elsewhere in $\mathcal{A}$ or $\mathcal{B}u$ - if it does then we ignore it. $\square$

Now, given any word g of $X \times Y$, we will wish to choose a transversal (a set of coset representatives) for $X \times Y$ with respect to the cyclic subgroup $\langle g \rangle$. From this, for any word w, we can associate with w a word w_g, which satisfies

$w = w_g g^k$ for some k. The actual choice of transversal is not particuarly important - all that is important is that we do have a transversal, and that the word thus obtained is linearly expressible.

First of all let us consider how to do this in the free group X. The intuitive idea is simply to cancel powers of g from the end of any word w, but we need to take into account a slight ambiguity if we merely follow this simple procedure.

Lemma 4.3.9 *Given a word g of X, of length n, and a word w of X which is linearly expressible with respect to g, then there is a procedure, operating in space $O(n)$, to determine a linearly expressible word w_g such that $w = w_g g^k$ for some k. Furthermore, if w_1 and w_2 lie in the same coset with respect to $\langle g \rangle$, and $(w_1)_g$ and $(w_2)_g$ are the words thus obtained, then $(w_1)_g = (w_2)_g$.*

Proof We assume that w is freely reduced, and we begin by removing any occurrences of g or g^{-1} from the end of w, using the method of Lemma 4.3.8 to perform this elimination. Suppose this leaves us with the word $\widetilde{w}$, which does not end in an occurrence of g or g^{-1}, but still lies in the same coset as w, and is still linearly expressible.

This does not necessarily give us a unique coset representative, since if $g = g_1 g_2$ (as freely reduced words), and w is equivalent to the word $\widetilde{w} g_1^{-1} g^k$, where no further powers of g can be extracted from $\widetilde{w} g_1^{-1}$, then w is also equivalent to $\widetilde{w} g_2 g^{k-1}$, and no further powers of g can be extracted from $\widetilde{w} g_2$. In this situation we will simply decide to choose $\widetilde{w} g_2 g^{k-1}$.

Hence, we reduce our word w by eliminating powers of g from the end, and we are left with a word $\widetilde{w}$. We then check to see if $\widetilde{w}$ ends in an occurrence of g_1^{-1} for some proper prefix g_1 of g. If so, then we replace $\widetilde{w}$ by $\widetilde{w} g_2$, where g_2 is the suffix of g obtained by eliminating g_1.

We then set w_g to be the word thus obtained. By forcing one choice in the only case where there can be any ambiguity, it is quite clear from this definition that if w_1 and w_2 lie in the same coset with respect to $\langle g \rangle$, then $(w_1)_g = (w_2)_g$, and we are done. Finally, it is immediately obvious that the word w_g that remains is linearly expressible. $\Box$

We can now move on to the direct product $X \times Y$. The idea is similar to the result for free groups, but there is an additional problem with the direct product, since there can be conflict between the requirements for the components in X and Y when we come to decide which possibility to choose in the case of any ambiguity. For example, if $w = x^{-1}y$ and $g = xy$, should we multiply by g to eliminate the x, or multiply by g^{-1} to eliminate the y, or simply leave the word alone? We resolve this problem by always giving priority to X. Remembering that all we need is some transversal that can be constructively determined, we can therefore define an appropriate word w_g in this case too.

Lemma 4.3.10 *Given a word g of $X \times Y$, of length n, and a word w of $X \times Y$ which is linearly expressible with respect to g, then there is a procedure, operating in space $O(n)$, to determine a linearly expressible word w_g such that $w = w_g g^k$ for some k. Furthermore, if w_1 and w_2 lie in the same coset with respect to $\langle g \rangle$, and $(w_1)_g = (w_2)_g$ are the words thus obtained, then $(w_1)_g = (w_2)_g$.*

Proof Suppose $w = w_X w_Y$ and $g = g_X g_Y$ in the obvious way. Using Lemma 4.3.9, we determine a word $(w_X)_{g_X}$ such that $w_X = (w_X)_{g_X}(g_X)^{k_X}$. Note that the value of k_X can be explicitly determined by our algorithm, and is certainly of length no greater than the length of w_X. Then, we take our word to be $w_g = (w_X)_{g_X} w_Y (g_Y)^{-k_X}$ (so that $w = w_g g^{k_X}$), which we freely reduce if necessary. This is obviously linearly expressible, and since the Y-component is determined explicitly from the X-component, which is uniquely determined, it is clear that this word satisfies the required conditions. $\Box$

Given a set $\mathcal{B}$, let $\mathcal{B}_g$ be the list of elements β_g, for $\beta \in \mathcal{B}$. Note that this list may contain repeated elements. The following lemma and its subsequent corollary will be crucial.

Lemma 4.3.11 *Suppose that g is a word of $X \times Y$, of length n, and suppose that $\mathcal{B}$ is a linearly expressible set, with respect to g. Then $\mathcal{B}$ is equivalent to the set $\mathcal{A} \triangle \mathcal{A}g$ (for some $\mathcal{A}$) if and only if any word w_g in $\mathcal{B}_g$ appears an even number of times in $\mathcal{B}_g$.*

Proof Firstly, suppose that $\{w_g g^{i_1}, ..., w_g g^{i_r}\}$ are the words of $\mathcal{B}$ corresponding to some w_g, and suppose r is even. We assume, without loss of generality, that $i_1 < i_2 < .... < i_r$. Consider the set

$$\mathcal{A}_{w_g} = \{w_g g^{i_1}, w_g g^{i_1+1} ...., w_g g^{i_2-1}, w_g g^{i_3}, w_g g^{i_3+1}, ..., w_g g^{i_4-1}, ...$$

$$...., w_g g^{i_{r-1}}, w_g g^{i_{r-1}+1}, ..., w_g g^{i_r-1}\}$$

which we note is well-defined, since r is even. Then

$$\mathcal{A}_{w_g} \triangle \mathcal{A}_{w_g} g = \{w_g g^{i_1}, w_g g^{i_2}, ..., w_g g^{i_r}\}$$

since all other elements are repeated in the union, and hence do not lie in the symmetric difference. Taking

$$\mathcal{A} = \bigcup_{w_g \in \mathcal{B}_g} \mathcal{A}_{w_g}$$

gives us the required result in one direction.

Conversely, suppose that $\mathcal{B} = \mathcal{A} \triangle \mathcal{A}g$ for some $\mathcal{A}$. It is obvious, by definition, that $w_g g^i \in \mathcal{A}$ if and only if $w_g g^{i+1} \in \mathcal{A}g$. Hence, for any w_g, every time w_g appears in $\mathcal{A}_g$, w_g also appears in $(\mathcal{A}g)_g$. Thus, it appears an even number of times when we take the two sets together. Any removal of identical elements when considering $\mathcal{A} \triangle \mathcal{A}g$ removes two occurrences of w_g, and so, however many removals we make, we still have an even number of occurrences of w_g in $(\mathcal{A} \triangle \mathcal{A}g)_g$. Since $\mathcal{B}_g = (\mathcal{A} \triangle \mathcal{A}g)_g$, w_g occurs an even number of times in $\mathcal{B}_g$, and the claim is proved. $\square$

Corollary 4.3.12 *Suppose that g is a word of $X \times Y$, of length n, and suppose that $\mathcal{B}$ is a linearly expressible set, with respect to g. Then we can determine, in space $O(n)$, whether or not $\mathcal{B}$ is equivalent to the set $\mathcal{A} \triangle \mathcal{A}g$ for some set $\mathcal{A}$ of words of $X \times Y$.*

Proof We use Lemma 4.3.11. We consider each element of $\mathcal{B}$ in turn. For each element w, we find the word w_g by Lemma 4.3.10. Having found w_g, we then set a counter to an initial value of 1. We then consider every other element w_i of $\mathcal{B}$ in turn, finding the corresponding $(w_i)_g$, and alternating our counter between 0 and 1 every time we determine that $(w_i)_g = w_g$.

We accept the set if and only if, for every word in $\mathcal{B}$, the corresponding counter gives 0. Since every word is linearly expressible, and this algorithm considers at most two words at any given time, it clearly operates in linear space, and is obviously deterministic. $\square$

Note the following simple lemma with regard to this sort of expression of sets, which will be extremely useful in our main proof.

Lemma 4.3.13 *Given a word g, suppose a set $\mathcal{A}$ can be expressed as $\mathcal{C} \triangle \mathcal{C}g^k$ for some $\mathcal{C}$, and $k \geq 1$. Then $\mathcal{A}$ can be expressed as $\mathcal{C}' \triangle \mathcal{C}'g$ for some $\mathcal{C}'$.*

Proof Simply note that $\mathcal{C} \triangle \mathcal{C}g^k$ is equal to

$$\mathcal{C} \triangle \mathcal{C}g \triangle \mathcal{C}g \triangle .... \triangle \mathcal{C}g^{k-1} \triangle \mathcal{C}g^{k-1} \triangle \mathcal{C}g^k$$

$$= (\mathcal{C}\triangle\mathcal{C}g\triangle....\triangle\mathcal{C}g^{k-1})\triangle(\mathcal{C}\triangle\mathcal{C}g\triangle....\triangle\mathcal{C}g^{k-1})g$$

which, taking $\mathcal{C}' = \mathcal{C}\triangle\mathcal{C}g\triangle....\triangle\mathcal{C}g^{k-1}$, gives the required result. $\Box$

The following lemmas are technical, but will prove to be useful.

Lemma 4.3.14 *Let $\mathcal{B}$ be a set of words of $X \times Y$, and suppose g and h are some words of $X \times Y$, where $gh = hg$ and $\langle g \rangle \cap \langle h \rangle = 1$. Then $\mathcal{B} = \mathcal{A}\triangle\mathcal{A}g$ for some set $\mathcal{A}$ if and only if $\mathcal{B}\triangle\mathcal{B}h = \mathcal{C}\triangle\mathcal{C}g$ for some set $\mathcal{C}$.*

Proof Suppose $\mathcal{B} = \mathcal{A}\triangle\mathcal{A}g$ for some set $\mathcal{A}$. Then

$$\mathcal{B}\triangle\mathcal{B}h = (\mathcal{A}\triangle\mathcal{A}g)\triangle(\mathcal{A}\triangle\mathcal{A}g)h = (\mathcal{A}\triangle\mathcal{A}h)\triangle(\mathcal{A}\triangle\mathcal{A}h)g$$

since $gh = hg$. Setting $\mathcal{C} = (\mathcal{A}\triangle\mathcal{A}h)$ gives us the required result in one direction.

Conversely, suppose that $\mathcal{B}\triangle\mathcal{B}h = \mathcal{C}\triangle\mathcal{C}g$ for some set $\mathcal{C}$. We will proceed by contradiction, so suppose that $\mathcal{B} \neq \mathcal{A}\triangle\mathcal{A}g$ for any set $\mathcal{A}$.

By Lemma 4.3.11, $\mathcal{B}_g$ must contain a member which appears an odd number of times in $\mathcal{B}_g$. In fact, we may assume that every member of $\mathcal{B}_g$ appears an odd number of times in $\mathcal{B}_g$, since if some member β_g appears an even number of times, then we can consider the set $\mathcal{B}'$ obtained by removing every γ in $\mathcal{B}$ such that $\gamma_g = \beta_g$. We must still have $\mathcal{B}'\triangle\mathcal{B}'h = \mathcal{C}'\triangle\mathcal{C}'g$ for some set $\mathcal{C}'$, by Lemma 4.3.11 (since every transversal representative still appears an even number of times). However, we still also have $\mathcal{B}' \neq \mathcal{A}\triangle\mathcal{A}g$ for any set $\mathcal{A}$, again by Lemma 4.3.11. Hence, progressively removing all such members, we can assume that every member of $\mathcal{B}_g$ appears an odd number of times in $\mathcal{B}_g$, simply by considering $\mathcal{B}'$ instead.

Hence, every member β_g appears an odd number of times in $\mathcal{B}_g$, but must appear an even number of times in $\mathcal{B}\triangle\mathcal{B}h$, since $\mathcal{B}\triangle\mathcal{B}h = \mathcal{C}\triangle\mathcal{C}g$. Hence

β_g must also appear in $(\mathcal{B}h)_g$. Considering every such β_g, we must have that (counting each element only once), the $(\mathcal{B}h)_g$ can be obtained simply by permuting the $\mathcal{B}_g$ in some way. Repeating this permutation obviously gives us the $(\mathcal{B}h^2)_g$, and so on. If m is the order of this permutation, then applying it m times means that $\mathcal{B}_g = (\mathcal{B}h^m)_g$, and so we have $h^m = g^q$ for some q, which is impossible by the assumption that $\langle g\rangle \cap \langle h\rangle \neq 1$.

Hence, we must have $\mathcal{B} = \mathcal{A}\triangle\mathcal{A}g$ for some set $\mathcal{A}$, which concludes the proof. $\square$

The reason for the hypotheses in the following seemingly rather technical lemma, will be apparent later.

Lemma 4.3.15 *Suppose that g is a word of $X \times Y$, of length n, and suppose that M is a subset of the centraliser of g in $X \times Y$, where there exists an effective procedure to determine, for a linearly expressible word w, whether w is a member of the set $\{mg^k : m \in M, k \in \mathbb{Z}\}$, where this procedure operates in space $O(n)$, and suppose that $m_1(m_2)^{-1} \notin \langle g\rangle$ for any $m_1 \neq m_2$ in M. Suppose further that $\mathcal{A}$ and $\mathcal{B}$ are linearly expressible sets of words of $X \times Y$ with respect to g. Then there exists a procedure, operating in space $O(n)$, to decide whether or not there exists a word u of M such that $\mathcal{A}\triangle\mathcal{B}u = \mathcal{C}\triangle\mathcal{C}g$ for some set of words $\mathcal{C}$ of $X \times Y$.*

Proof Firstly, use Corollary 4.3.12 to test whether or not $\mathcal{B} = \mathcal{D}\triangle\mathcal{D}g$ for some $\mathcal{D}$. If so, then we claim that for any u in M, $\mathcal{A}\triangle\mathcal{B}u = \mathcal{C}\triangle\mathcal{C}g$ for some $\mathcal{C}$ if and only if $\mathcal{A} = \mathcal{E}\triangle\mathcal{E}g$ for some $\mathcal{E}$. To prove this claim, suppose firstly that $\mathcal{A} = \mathcal{E}\triangle\mathcal{E}g$. Then for any u in M,

$$\mathcal{A}\triangle\mathcal{B}u = (\mathcal{E}\triangle\mathcal{E}g)\triangle(\mathcal{D}\triangle\mathcal{D}g)u$$

$$= (\mathcal{E}\triangle\mathcal{D}u)\triangle(\mathcal{E}\triangle\mathcal{D}u)g$$

since g commutes with u by definition. Setting $\mathcal{C} = \mathcal{E}\triangle\mathcal{D}u$ gives us the required result in one direction.

Conversely, suppose $\mathcal{A}\triangle\mathcal{B}u = \mathcal{C}\triangle\mathcal{C}g$ for some u in M. Then

$$\mathcal{A}\triangle(\mathcal{D}\triangle\mathcal{D}g)u = \mathcal{C}\triangle\mathcal{C}g$$

which gives us immediately

$$\mathcal{A} = (\mathcal{C}\triangle\mathcal{C}g)\triangle(\mathcal{D}\triangle\mathcal{D}g)u = (\mathcal{C}\triangle\mathcal{D}u)\triangle(\mathcal{C}\triangle\mathcal{D}u)g$$

and, setting $\mathcal{E} = \mathcal{C}\triangle\mathcal{D}u$, the claim is proved.

So, if $\mathcal{B} = \mathcal{D}\triangle\mathcal{D}g$ for some $\mathcal{D}$, then we simply have to test whether or not $\mathcal{A} = \mathcal{E}\triangle\mathcal{E}g$ for some $\mathcal{E}$, again using Corollary 4.3.12. If so, then any u lying in M can be taken.

Hence, we are left only with the case where $\mathcal{B}$ cannot be written in the form $\mathcal{D}\triangle\mathcal{D}g$ for any $\mathcal{D}$. In this case, using Lemma 4.3.11, some β_g in $\mathcal{B}_g$ appears an odd number of times in $\mathcal{B}_g$. We can assume without loss of generality that β_1 in $\mathcal{B}$ is such that $(\beta_1)_g$ appears an odd number of times in $\mathcal{B}_g$.

Consider the equation $((\beta_1)_g)^{-1}(\alpha_j)_g = ug^k$ for any α_j in $\mathcal{A}$. By assumption, since clearly $((\beta_1)_g)^{-1}(\alpha_j)_g$ is linearly expressible, we can determine u and k such that this equation holds. Also, these values must be unique, since if $((\beta_1)_g)^{-1}(\alpha_j)_g = u_1g^{k_1}$ and $((\beta_1)_g)^{-1}(\alpha_j)_g = u_2g^{k_2}$, then $u_1(u_2)^{-1} = g^{k_1-k_2}$, which contradicts our assumption on M, unless $u_1 = u_2$ and $k_1 = k_2$.

Hence, from this, we can determine a unique u_j (if it exists) satisfying $((\beta_1)_g)^{-1}(\alpha_j)_g = u_jg^k$ for each α_j. We now claim that, for any word $u_r \in M$, then $\mathcal{A}\triangle\mathcal{B}u_r$ cannot be equivalent to $\mathcal{C}\triangle\mathcal{C}g$ unless u_r is equal to one of the u_j that we can determine above. To prove this, consider the set $\mathcal{A}\triangle\mathcal{B}u_r$. Included in this set is the set of elements lying in $\mathcal{B}u_r$ of the form $(\beta_1)_gg^lu_r$, each of which is equivalent to the word $(\beta_1)_gu_rg^l$. This subset contains an odd number of elements, by our definition of β_1. But there can be no element of this form $(\beta_1)_g u_r g^l$ lying in $\mathcal{A}$, since if there was, then we would have $(\alpha_j)_g g^{l_1} = (\beta_1)_g u_r g^l$, which gives us $((\beta_1)_g)^{-1}(\alpha_j)_g = u_r g^{l-l_1}$, contradicting our assumption that u_r is not one of the u_j determined above. Hence, if u_r is not one of the u_j determined above, then there is an odd number of occurrences of $(\beta_1)_g$ in $(\mathcal{A}\triangle\mathcal{B}(u_r))_g$, and so by Lemma 4.3.11, it cannot be of the form $\mathcal{C}\triangle\mathcal{C}g$, as required. Hence, our algorithm works as follows. We first of all determine a suitable $(\beta_1)_g$ simply by checking each of the $(\beta_i)_g$ in turn until one is found. For each α_j in $\mathcal{A}$, we then solve the equation $((\beta_1)_g)^{-1}\alpha_j = u_j g^k$ for u_j, which gives us a unique solution, where u_j is linearly expressible, by assumption. For each α_j, we then need to check whether or not this particular $\mathcal{A}\triangle\mathcal{B}u_j$ can be written in the form $\mathcal{C}\triangle\mathcal{C}g$, which we can do in linear space by Lemma 4.3.8 and Corollary 4.3.12, and this concludes the proof $\square$

We obviously need to consider under what conditions on M the hypotheses of Lemma 4.3.15 hold. We certainly have the following cases.

Corollary 4.3.16 *Suppose that g is a word of $X \times Y$, of length n, and suppose that M is a cyclic subset of the centraliser of g in $X \times Y$, where $M = \langle h_0 \rangle$, and $M \cap \langle g \rangle = 1$. Suppose further that $\mathcal{A}$ and $\mathcal{B}$ are linearly expressible sets of words of $X \times Y$, with respect to g. Then there exists a procedure, operating in space $O(n)$, to decide whether or not there exists an integer q such that $\mathcal{A}\triangle\mathcal{B}(h_0)^q = \mathcal{C}\triangle\mathcal{C}g$ for some set of words $\mathcal{C}$ of $X \times Y$.*

Proof Any word of M is of the form $(h_0)^q$. It is immediate that, for any $q_1 \neq q_2$, the element $(h_0)^{q_1}((h_0)^{q_2})^{-1} = (h_0)^{q_1-q_2}$ cannot lie in $\langle g \rangle$ by our assumption on M. If we wish to solve the equation $u = (h_0)^i g^j$, then this has a unique solution by applying Lemma 2.5.9 to each component of u in X

and Y (since the case where there are infinitely many solutions cannot arise in both cases, since $\langle h_0 \rangle \cap \langle g \rangle = 1$), and this solution is linearly bounded, and hence we have the conditions of Lemma 4.3.15. $\square$

Corollary 4.3.17 *Suppose that g is a word of $X \times Y$, of length n, and suppose that M is the centraliser of g in $X \times Y$. Suppose further that $\mathcal{A}$ and $\mathcal{B}$ are linearly expressible sets of words of $X \times Y$, with respect to g. Then there exists a procedure, operating in space $O(n)$, to decide whether or not there exists m_g in M_g such that $\mathcal{A} \triangle \mathcal{B} m_g = \mathcal{C} \triangle \mathcal{C} g$ for some set of words $\mathcal{C}$ of $X \times Y$.*

Proof Given any linearly expressible word w, we can both find w_g, and determine k such that $w = w_g g^k$, from Lemma 4.3.10. Also, if $(m_1)_g((m_2)_g)^{-1} = g^l$, for some non-zero l, then we have $(m_1)_g = (m_2)_g g^l$, which is an obvious contradiction, or we could eliminate more occurrences of g from $(m_1)_g$. Hence we have the conditions of Lemma 4.3.15. $\square$

By Lemma 4.3.11, if m_g satisfies the equation $\mathcal{A} \triangle \mathcal{B} m_g = \mathcal{C} \triangle \mathcal{C} g$ for some set of words $\mathcal{C}$ of $X \times Y$, then so does $m_g g^k$ for any k, since $m_g = (m_g g^k)_g$, and any solution must be of this form. Hence, if this equation is solvable, then from Lemma 4.3.15, either any word in the centraliser of g is a solution, or every solution is of the form $m_g g^k$ (for some k), for a finite set of possibilities m_g.

In a similar vein to Lemma 4.3.15, we have the following.

Lemma 4.3.18 *Suppose that v is a word of $X \times Y$, where v is of the form g^q, and is of length n. Suppose further that $\mathcal{A}$ and $\mathcal{B}$ are linearly expressible sets of words of $X \times Y$, with respect to v. Then there exists a procedure,* *operating in space $O(n)$, to decide whether or not there exists an integer p such that $\mathcal{A}\triangle\mathcal{B}g^p = \mathcal{C}\triangle\mathcal{C}g^q$ for some set of words $\mathcal{C}$ of $X \times Y$.*

Proof We claim that $\mathcal{A}\triangle\mathcal{B}g^p = \mathcal{C}\triangle\mathcal{C}g^q$ for some set $\mathcal{C}$, if and only if it is also the case that $\mathcal{A}\triangle\mathcal{B}g^{p\pm q} = \mathcal{D}\triangle\mathcal{D}g^q$, for some set $\mathcal{D}$. To prove this, we firstly note that

$$\mathcal{B}g^p\triangle\mathcal{B}g^{p+q} = \mathcal{B}g^p\triangle(\mathcal{B}g^p)g^q,$$

and also that

$$\mathcal{B}g^p\triangle\mathcal{B}g^{p-q} = (\mathcal{B}g^{p-q})g^q\triangle\mathcal{B}g^{p-q} = \mathcal{B}g^{p-q}\triangle(\mathcal{B}g^{p-q})g^q.$$

Hence, in either case, $\mathcal{B}g^p\triangle\mathcal{B}g^{p\pm q} = \mathcal{E}\triangle\mathcal{E}g^q$ for some $\mathcal{E}$. Given this, we simply note that if $\mathcal{A}\triangle\mathcal{B}g^p = \mathcal{C}\triangle\mathcal{C}g^q$, then

$$\mathcal{A}\triangle\mathcal{B}g^{p\pm q} = \mathcal{A}\triangle(\mathcal{B}g^p\triangle\mathcal{B}g^p)\triangle\mathcal{B}g^{p\pm q}$$

(where the notation $p \pm q$ is assumed to mean either $p + q$ throughout, or $p - q$ throughout)

$$= (\mathcal{A}\triangle\mathcal{B}g^p)\triangle(\mathcal{B}g^p\triangle\mathcal{B}g^{p\pm q})$$

$$= (\mathcal{C}\triangle\mathcal{C}g^q)\triangle(\mathcal{E}\triangle\mathcal{E}g^q)$$

$$= (\mathcal{C}\triangle\mathcal{E})\triangle(\mathcal{C}\triangle\mathcal{E})g^q$$

as required, taking $\mathcal{D} = \mathcal{C}\triangle\mathcal{E}$. Entirely similarly, in the converse direction, if $\mathcal{A}\triangle\mathcal{B}g^{p\pm q} = \mathcal{D}\triangle\mathcal{D}g^q$, then

$$\mathcal{A}\triangle\mathcal{B}g^p = \mathcal{A}\triangle(\mathcal{B}g^{p\pm q}\triangle\mathcal{B}g^{p\pm q})\triangle\mathcal{B}g^p$$

$$= (\mathcal{A}\triangle\mathcal{B}g^{p\pm q})\triangle(\mathcal{B}g^p\triangle\mathcal{B}g^{p\pm q})$$

$$= (\mathcal{D}\triangle\mathcal{D}g^q)\triangle(\mathcal{E}\triangle\mathcal{E}g^q)$$

$$= (\mathcal{D}\triangle\mathcal{E})\triangle(\mathcal{D}\triangle\mathcal{E})g^q,$$

and the claim is proved,taking $\mathcal{C} = \mathcal{D} \triangle \mathcal{E}$. Hence, if the equality holds for some p, then it also holds for $p \pm iq$ for any integer i. It is therefore enough to simply test whether or not the equation $\mathcal{A} \triangle \mathcal{B} g^p = \mathcal{C} \triangle \mathcal{C} g^q$ holds for those p satisfying $0 \leq p < q$, which we can do in linear space by Lemma 4.3.8 and Corollary 4.3.12, since $|g^p| \leq |g^q| = O(n)$. □

We give a brief lemma regarding products of free groups. It is well-known, see for example [39], that the centraliser of a word in a free group is a cyclic subgroup (unless the word is empty, in which case the whole group is the centraliser). If $f = (f_0)^c$ for c maximal, then the centraliser of f is simply $\langle f_0 \rangle$. Given this, we have the following.

Lemma 4.3.19 *Suppose w is a freely-reduced word of $(X \times Y) * Z$. Then the centraliser of w is cyclic if and only if w contains a Z-symbol. Similarly, if w is a freely-reduced word of $(X' \times Y') * T$, then the centraliser of w is cyclic if and only if w contains a T-symbol.*

Proof This is a well-known result (again see [39] for example), and so we simply sketch the proof. If w contains a Z-symbol, then let $w = w_0^q$ for q maximal. Then the centraliser of w is simply the set $\{{w_0}^i : i \in \mathbb{Z}\}$ which is obviously cyclic. Otherwise, suppose $w = ((w_0)_X)^{q_1}((w_0)_Y)^{q_2}$ with q_1 and q_2 maximal. Then the centraliser of w is the set $\{((w_0)_X)^i((w_0)_Y)^j : i, j \in \mathbb{Z}\}$ which is obviously non-cyclic. The second part follows entirely similarly. □

The following lemma will also be useful.

Lemma 4.3.20 *Suppose F is a free group, and let f be a word of F of length n. Suppose h, g_1 and g_2 are words of F of length $O(n)$. Then the equation $h^q g_1 w = g_2$, for w some word such that $wf = fw$, has either zero or one* *solutions for q, or any q satisfies the equation. If there is a unique solution, then the word h^q can be written in space $O(n)$.*

Proof Suppose that $h^{q_1}g_1w_1 = g_2$, and $h^{q_2}g_1w_2 = g_2$ are two such solutions, where $q_1 \neq q_2$. Then $(g_1)^{-1}h^{q_1-q_2}g_1 = w_2(w_1)^{-1}$, which commutes with f since w_1 and w_2 both commute with f. Since we are in a free group, and hence the centraliser of f is either cyclic or the whole group, we must have that $(g_1)^{-1}hg_1$ also commutes with f. Let $w_0 = (g_1)^{-1}hg_1$. Then, for any q, we have

$$h^q g_1 = g_1(w_0)^q = g_2(g_2)^{-1}g_1(w_0)^q,$$

simply by inserting a trivial word. But

$$(g_2)^{-1}g_1 = ((w_1)^{-1}(g_1)^{-1}h^{-q_1})g_1 = (w_1)^{-1}(w_0)^{-q_1}$$

which commutes with f, since both terms commute with f. Hence, we have the equation

$$h^q g_1((w_0)^{-q}(g_1)^{-1}g_2) = g_2,$$

where $(w_0)^{-q}(g_1)^{-1}g_2$ commutes with f, which means that any value of q can also solve this equation. Hence if there is more than one solution, then any value of q is a solution.

Now, if f is empty, then the centraliser of f is the whole of F, and obviously we can then have infinitely many solutions. Otherwise, the centraliser of f is cyclic, equal to $\langle f_0 \rangle$, say, and we can solve whether or not $h^q g_1(f_0)^p = g_2$ by Lemma 2.5.9. If there is a unique solution, then by Lemma 2.5.9, h^q can be written in space $O(n)$. $\square$

4.3.3 Reducing the conjugacy problem

Having established these lemmas, we can now move on to consider the conjugacy problem in G_{CS}. The majority of the work has now been done. The remainder of this chapter is now essentially the main proof of [23], adapted to suit our approach here, and stressing the linearly bounded nature of our adapted procedure. We begin by reducing the problem somewhat.

Suppose that u_1 and u_2 are words of G_{CS}. By Lemma 4.3.6, suppose that

$$u_1 = (v_1^{-1}av_1)....(v_{i_1}^{-1}av_{i_1})\left(\prod_{\beta_1\in\mathcal{B}_1}\beta_1^{-1}b\beta_1\right)\mu_1\mu_2$$

and

$$u_2 = (w_1^{-1}aw_1)....(w_{i_2}^{-1}aw_{i_2})\left(\prod_{\beta_2\in\mathcal{B}_2}\beta_2^{-1}b\beta_2\right)\nu_1\nu_2.$$

Lemma 4.3.21 *If u_1 and u_2 are conjugate in G_{CS}, then $\mu_1\mu_2$ and $\nu_1\nu_2$ are conjugate in $((X \times Y) * Z) \times ((X' \times Y') * T)$.*

Proof Suppose $(u_3)^{-1}u_1u_3 = u_2$ for some word u_3. Suppose that ω_1 is the projection of u_3 onto $(X \times Y) * Z$, and ω_2 is the projection of u_3 onto $(X' \times Y') * T$. Considering the projections of both sides of the equation $(u_3)^{-1}u_1u_3 = u_2$, by the uniqueness of the canonical form of Lemma 4.3.6, we must have $(\omega_1)^{-1}\mu_1\omega_1 = \nu_1$ and $(\omega_2)^{-1}\mu_2\omega_2 = \nu_2$ as required. □

Lemma 4.3.22 $((X \times Y) * Z) \times ((X' \times Y') * T)$ *has deterministic context-sensitive conjugacy problem. In addition, if two words α_1 and α_2 are conjugate, then there exists a conjugating element of length no greater than* $\max(|\alpha_1|, |\alpha_2|)$.

Proof By using repeated applications of Lemma 2.5.4, Lemma 3.2.2, and Lemma 3.3.9, it is immediate that $((X \times Y) * Z) \times ((X' \times Y') * T)$ has deterministic context-sensitive conjugacy problem. Suppose $f = f_1^{-1} f_2 f_1$ and $h = h_1^{-1} h_2 h_1$ are words in a free group, where f_2 and h_2 are cyclically reduced. Then, by Lemma 2.5.4, a conjugating element of f and h must be of the form $f_1^{-1} f_3^{-1} h_1$, where f_3 is a prefix of f_2 (so its conjugacy action permutes f_2), which is obviously of length no greater than the length of the larger of the two words. In a free product, we have a similar result by Theorem 3.3.2 (where the amalgamated subgroup is obviously the trivial subgroup), since again we must cyclically reduce and permute the words. Finally, in a direct product, the conjugating word is obtained by taking a conjugating word in each factor, and hence if these are no larger than the words in the factors, then they are no larger than the words in the direct product. Repeatedly applying these results gives us the required result. □

Hence, we can test to see whether or not $\mu_1\mu_2$ and $\nu_1\nu_2$ are conjugate in $((X \times Y) * Z) \times ((X' \times Y') * T)$. If so, we can immediately write down an element ω where $\omega^{-1}\mu_1\mu_2\omega = \nu_1\nu_2$ from Lemma 4.3.22 (if not, of course we can immediately reject the words). Therefore, given u_1 and u_2, we can conjugate u_1 by ω. This resulting word is of length no greater than $3\max(|u_1|, |u_2|)$ (since ω is bounded in length by $\max(|u_1|, |u_2|)$, and so we are still working in linear space. Hence we can still find an equivalent form for this word in the usual way. We can assume, therefore, that we have done this if necessary, and so we may assume that our words u_1 and u_2 are of the form

$$u_1 = (v_1^{-1} a v_1) \ldots . (v_{i_1}^{-1} a v_{i_1}) \left(\prod_{\beta_1 \in \mathcal{B}_1} \beta_1^{-1} b \beta_1 \right) \mu_1 \mu_2$$

and

$$u_2 = (w_1^{-1} a w_1) \ldots . (w_{i_2}^{-1} a w_{i_2}) \left(\prod_{\beta_2 \in \mathcal{B}_2} \beta_2^{-1} b \beta_2 \right) \mu_1 \mu_2 .$$

The following two lemmas are technical lemmas with regard to conjugacy in G_{CS}. Since they do not in any way affect our complexity considerations, then we choose to omit the proofs, and refer to [23].

Lemma 4.3.23 *Given u_1 and u_2 in the above form, if u_1 and u_2 are conjugate in G_{CS}, then $i_1 = i_2$.*

Proof See [23]. □

We can also simplify our potential conjugating words by assuming that there are no conjugates of a present.

Lemma 4.3.24 *Given u_1 and u_2 in the above form, if u_1 and u_2 are conjugate in G_{CS}, then they are conjugate via an element of the form*

$$\left(\prod_{\beta_3 \in \mathcal{B}_3} \beta_3^{-1} b \beta_3 \right) \eta_1 \eta_2$$

where $\eta_1 \mu_1 = \mu_1 \eta_1$ and $\eta_2 \mu_2 = \mu_2 \eta_2$.

Proof See [23]. □

Given this, then the path we must follow towards determining the conjugacy of our two elements, depends on the elements that could commute with the μ_i. Obviously, Lemma 4.3.19 will be useful here.

4.3.4 Solving the conjugacy problem

We can now move on to the proof that the conjugacy problem of G_{CS} is a deterministic context-sensitive problem. Again, this is essentially the main proof of [23], adapted to suit our approach here, and stressing the linearly bounded nature of our adapted procedure. Recall that we are given words

$$u_1 = (v_1^{-1} a v_1) \ldots . (v_{i_1}^{-1} a v_{i_1}) \left(\prod_{\beta_1 \in \mathcal{B}_1} \beta_1^{-1} b \beta_1 \right) \mu_1 \mu_2$$

$$u_2 = (w_1^{-1}aw_1)....(w_{i_1}^{-1}aw_{i_1})\left(\prod_{\beta_2\in\mathcal{B}_2}\beta_2^{-1}b\beta_2\right)\mu_1\mu_2,$$

and we are looking for a conjugating word u_3 (and so $u_1u_3 = u_3u_2$), which is of the form

$$\left(\prod_{\beta_3\in\mathcal{B}_3}\beta_3^{-1}b\beta_3\right)\eta_1\eta_2.$$

We have a series of cases to consider, depending on whether the centralisers of μ_1 and μ_2 are cyclic, and the value of i_1. Firstly suppose that i_1 is zero.

Lemma 4.3.25 *If i_1 is zero, then there is a deterministic context-sensitive procedure to determine whether or not u_1 and u_2 are conjugate in G_{CS}.*

Proof In this case, there are no instances of a, and hence we do not need to worry about any additional conjugates of b that could be obtained in rewriting a word. Writing $u_1u_3 = u_3u_2$, we obtain

$$\left(\prod_{\beta_1\in\mathcal{B}_1}\beta_1^{-1}b\beta_1\right)\mu_1\mu_2\left(\prod_{\beta_1\in\mathcal{B}_3}\beta_3^{-1}b\beta_3\right)\eta_1\eta_2$$

$$= \left(\prod_{\beta_3\in\mathcal{B}_3}\beta_3^{-1}b\beta_3\right)\eta_1\eta_2\left(\prod_{\beta_2\in\mathcal{B}_2}\beta_2^{-1}b\beta_2\right)\mu_1\mu_2.$$

Suppose firstly that μ_1 does not contain an occurrence of a Z-symbol, that is, it consists only of symbols in $X \times Y$. Then, by Lemma 4.3.19, η_1 does not consist of any Z-symbol either. Hence, we obtain

$$\left(\prod_{\beta_1\in\mathcal{B}_1}\beta_1^{-1}b\beta_1\right)\left(\prod_{\beta_1\in\mathcal{B}_3}(\beta_3\mu_1^{-1})^{-1}b\beta_3\mu_1^{-1}\right)\mu_1\mu_2\eta_1\eta_2$$

$$= \left(\prod_{\beta_3\in\mathcal{B}_3}\beta_3^{-1}b\beta_3\right)\left(\prod_{\beta_2\in\mathcal{B}_2}(\beta_2\eta_1^{-1})^{-1}b\beta_2\eta_1^{-1}\right)\mu_1\mu_2\eta_1\eta_2.$$

Considering the conjugates of b, and rearranging slightly, we are left with the equation

$$\mathcal{B}_1 \triangle \mathcal{B}_2 \eta_1^{-1} = \mathcal{B}_3 \triangle \mathcal{B}_3 \mu_1^{-1}.$$

But, by Lemma 4.3.11, if $\eta_1 = \eta(\mu_1)^i$ satisfies this equation, then so does η (since the tranversal representatives are the same). Hence, if $\mathcal{M}$ is the centraliser of μ_1, we can assume that η_1 lies in $\mathcal{M}_{\mu_1}$, and we can solve this equation in linear space by Corollary 4.3.17.

Now suppose that μ_1 contains an occurrence of a Z-symbol, and that $\mu_1 = (\gamma_1\zeta_1....\gamma_m\zeta_m)^{k_1}$, where each γ_i is a word in $X \times Y$, each ζ_i is a word in Z, and k_1 is maximal. By Lemma 4.3.19, we must have $\eta_1 = (\gamma_1\zeta_1....\gamma_m\zeta_m)^p$ for some p. Let $\gamma_0 = \gamma_1....\gamma_m$. Then, obviously,

$$\mu_1 \left(\prod_{\beta_3 \in \mathcal{B}_3} \beta_3^{-1} b \beta_3 \right) = \left(\prod_{\beta_3 \in \mathcal{B}_3} (\beta_3 \gamma_0^{-k_1})^{-1} b \beta_3 \gamma_0^{-k_1} \right) \mu_1,$$

$$\eta_1 \left(\prod_{\beta_2 \in \mathcal{B}_2} \beta_2^{-1} b \beta_2 \right) = \left(\prod_{\beta_2 \in \mathcal{B}_2} (\beta_2 \gamma_0^{-p})^{-1} b \beta_2 \gamma_0^{-p} \right) \eta_1.$$

Hence, we have to solve the equation $\mathcal{B}_1 \triangle \mathcal{B}_2 \gamma_0^{-p} = \mathcal{B}_3 \triangle \mathcal{B}_3 \gamma_0^{-k_1}$, which can be solved in linear space by Lemma 4.3.18, and we are done. □

This has covered the case where i_1 is equal to zero, and so we had no occurrences of a anywhere in our words. We now assume that $i_1 \geq 1$. Therefore, when writing $u_1 u_3 = u_3 u_2$, we are looking to solve the equation

$$(v_1^{-1} a v_1)....(v_{i_1}^{-1} a v_{i_1}) \left(\prod_{\beta_1 \in \mathcal{B}_1} \beta_1^{-1} b \beta_1 \right) \mu_1 \mu_2 \left(\prod_{\beta_3 \in \mathcal{B}_3} \beta_3^{-1} b \beta_3 \right) \eta_1 \eta_2$$

$$= \left(\prod_{\beta_3 \in \mathcal{B}_3} \beta_3^{-1} b \beta_3 \right) \eta_1 \eta_2 (w^{-1} a w_1)....(w_{i_1}^{-1} a w_{i_1}) \left(\prod_{\beta_2 \in \mathcal{B}_2} \beta_2^{-1} b \beta_2 \right) \mu_1 \mu_2.$$

There are four cases to consider. We begin with the simplest two cases, where there are no T-symbols to consider. In this situation, again, there are no additional conjugates of b to consider in any rearrangement of our words. However, when considering equality, we must take into account conjugates of a, as well as conjugates of b.

Lemma 4.3.26 *Suppose that μ_1 is a word in $X \times Y$, and that μ_2 is a word in $X' \times Y'$. Then there is a deterministic context-sensitive procedure to determine whether or not u_1 and u_2 are conjugate in G_{CS}.*

Proof Exactly as in Lemma 4.3.25, we need to solve the equation

$$\mathcal{B}_1 \triangle \mathcal{B}_2 \eta_1^{-1} = \mathcal{B}_3 \triangle \mathcal{B}_3 \mu_1^{-1},$$

which can again be done by Corollary 4.3.17, assuming that a solution lies in $\mathcal{M}_{\mu_1}$,where $\mathcal{M}$ is the centraliser of μ_1. However, previously we were only concerned with the solvability of this equation. We now need to consider the impact of any solution on the conjugates of a as well.

From Corollary 4.3.17, the equation above is either unsolvable, solvable for only a finite number of words of M_{μ_1} (which are linearly expressible), or solvable for every η_1. Obviously, we can immediately reject the words if the equation is unsolvable.

Otherwise, using Lemma 4.3.6 to consider the conjugates of a that appear, and equating, we obtain $v_i = (\eta_1)^{-1} w_i (\overline{\eta_2})^{-1}$ for every $1 \leq i \leq i_1$. Now, we can consider $\eta_1 = (\eta_1)_X (\eta_1)_Y$, $\eta_2 = (\eta_2)_{X'} (\eta_2)_{Y'}$, $v_i = (v_i)_X (v_i)_Y$ and $w_i = (w_i)_X (w_i)_Y$ in the usual way. Then, splitting our equation into two, and rearranging, we obtain

$$(\overline{(\eta_2)_{X'}})(v_i)_X (\eta_1)_X = (w_i)_X,$$

$$(\overline{(\eta_2)_{Y'}})(v_i)_Y (\eta_1)_Y = (w_i)_Y.$$

If the equation $\mathcal{B}_1 \triangle \mathcal{B}_2 \eta_1^{-1} = \mathcal{B}_3 \triangle \mathcal{B}_3 \mu_1^{-1}$ had been determined to be solvable for every η_1, then of course we can forget about it, and we simply need to solve these two equations. But, in this case, if $(\mu_1)_X = (\delta_1)^{c_1}$, for c_1 maximal, then by Lemma 4.3.19, $(\eta_1)_X = (\delta_1)^p$ for some p. Similarly, if $(\mu_2)_{X'} = (\delta_2)^{c_2}$, for c_2 maximal, then $(\eta_2)_{X'} = (\delta_2)^q$ for some q. Hence, the equation $(\overline{(\eta_2)_{X'}})(v_i)_X(\eta_1)_X = (w_i)_X$ becomes

$$(\overline{\delta_2})^q(v_i)_X(\delta_1)^p = (w_i)_X$$

which is solvable in linear space by Lemma 2.5.9. An entirely similar argument for the equation in Y gives us the required result.

Finally, we consider the case where $\mathcal{B}_1 \triangle \mathcal{B}_2 \eta_1^{-1} = \mathcal{B}_1 \triangle \mathcal{B}_2 \mu_1^{-1}$ is only solvable for a finite number of elements of M_{μ_1}, which are linearly expressible. Take each of these elements, m_j, say, in turn. Then any solution involving m_j must be of the form $m_j((\mu_1)_X)^r$ for some r. Hence, we are searching for an integer r such that

$$(\overline{(\eta_2)_{X'}})(v_i)_X(m_j)_X((\mu_1)_X)^r = (w_i)_X,$$

and similarly for Y. These equations can be solved in linear space by Lemma 4.3.20. If the same solution satisfies both of these equations for all i, then we are done, otherwise we repeat our algorithm with the next m_j. If we cannot solve the equations for any m_j, then we reject the words. □

We now consider what happens when μ_1 contains a Z-symbol.

Lemma 4.3.27 *Suppose μ_1 contains a Z-symbol, but that μ_2 is a word in $X' \times Y'$. Then there is a deterministic context-sensitive procedure to determine whether or not u_1 and u_2 are conjugate in G_{CS}.*

Proof Exactly as in Lemma 4.3.25, we take $\mu_1 = (\gamma_1\zeta_1....\gamma_m\zeta_m)^{k_1}$, where k_1 is maximal, and then note that η_1 must be of the form $(\gamma_1\zeta_1....\gamma_m\zeta_m)^p$. Again, letting $\gamma_0 = \gamma_1....\gamma_m$, we obtain, equating conjugates of b,

$$\mathcal{B}_1 \triangle \mathcal{B}_2 \gamma_0^{-p} = \mathcal{B}_3 \triangle \mathcal{B}_3 \gamma_0^{-k_1}.$$

Consider next the conjugates of a. We must have $v_i = (\overline{\eta_2})^{-1} w_i {\gamma_0}^{-p}$, or, rearranging, $\eta_2 v_i {\gamma_0}^p = w_i$, for every $1 \leq i \leq i_1$. Splitting these equations into equations in X and Y in the usual way, where $\gamma_0 = (\gamma_0)_X(\gamma_0)_Y$, $\eta_2 = (\eta_2)_{X'}(\eta_2)_{Y'}$, $v_i = (v_i)_X(v_i)_Y$ and $w_i = (w_i)_X(w_i)_Y$, we obtain the two equations

$$\overline{(\eta_2)_{X'}}(v_i)_X((\gamma_0)_X)^p = (w_i)_X$$

$$\overline{(\eta_2)_{Y'}}(v_i)_Y((\gamma_0)_Y)^p = (w_i)_Y.$$

Note that the powers p must be equal here.

From Lemma 4.3.20, each of these two sets of equations each have either zero or one solution, or are satisfied for every p. If there exists a single value satisfying one set of equations (which can be determined in linear space), then we check that this value also satisfies the other set of equations. If so, then the lengths of $((\gamma_0)_X)^p$ and $((\gamma_0)_Y)^p$ are linearly bounded by Lemma 4.3.20, and hence so is the length of γ_0^p. Therefore, we can verify whether or not $\mathcal{B}_1 \triangle \mathcal{B}_2 \gamma_0^{-p} = \mathcal{B}_3 \triangle \mathcal{B}_3 \gamma_0^{-k_1}$ for this particular p, by Lemma 4.3.8 and Corollary 4.3.12.

If the two equations hold for any p, then we can immediately apply Lemma 4.3.18 to determine if there is a solution to $\mathcal{B}_1 \triangle \mathcal{B}_2 \gamma_0^{-p} = \mathcal{B}_3 \triangle \mathcal{B}_3 \gamma_0^{-k_1}$ in linear space, and we are done. □

The remaining two cases are a little more complicated, due to the fact that if μ_2 contains a T-symbol, then there are additional conjugates of b to consider when we try to form our canonical form for a word. Let us first look at the case where μ_1 lies in $X \times Y$. Firstly let us note a simple lemma.

Lemma 4.3.28 *There is a deterministic context-sensitive procedure to determine, given two words σ_1 and σ_2 of $X \times Y$, whether or not $\langle \sigma_1 \rangle \cap \langle \sigma_2 \rangle = 1$.*

Proof Writing $\sigma_1 = ((\rho_1)_X)^{c_1}((\rho_1)_Y)^{c_2}$ and $\sigma_2 = ((\rho_2)_X)^{d_1}((\rho_2)_Y)^{d_2}$, where c_1, c_2, d_1, and d_2 are maximal, then $\langle\sigma_1\rangle \cap \langle\sigma_2\rangle \neq 1$ if and only if $((\rho_1)_X) = ((\rho_2)_X)^{\pm 1}$ and $((\rho_1)_Y) = ((\rho_2)_Y)^{\pm 1}$, and $c_1 d_2 = d_1 c_2$ (we omit this standard proof, see for example [39]). Clearly, the size of both $c_1 d_2$ and $d_1 c_2$ is at most quadratic in the length of the input, which can be stored in linear space using binary. Note that this means that we have a word σ such that $\sigma = (\sigma_1)^{k_1} = (\sigma_2)^{k_2}$, and that σ is linearly expressible. □

Lemma 4.3.29 *Suppose μ_1 is a word in $X \times Y$, but μ_2 contains a T-symbol. Then there is a deterministic context-sensitive procedure to determine whether or not u_1 and u_2 are conjugate in G_{CS}.*

Proof This time, η_2 is a word of $(X' \times Y') * T$, but η_1 is a word of $(X \times Y)$. In the usual way, by Lemma 4.3.19, if $\mu_2 = (\delta_1 t^{j_1} \ldots\delta_s t^{j_s})^c$, where $|c|$ is maximal, then η_2 must be of the form $(\delta_1 t^{j_1} \ldots\delta_s t^{j_s})^q$. We use $|c|$ here, so that we can assume that $q \geq 0$, since we can take the inverse of $\delta_1 t^{j_1} \ldots\delta_s t^{j_s}$ if necessary. Given this, let $\delta_0 = \overline{\delta_1 \ldots\delta_s}$.

Considering $u_1 u_3 = u_3 u_2$, it is clear from Lemma 4.3.6 that, equating conjugates of a in this equation, we must have $v_i = (\delta_0)^{-q} w_i (\eta_1)^{-1}$ for every $1 \leq i \leq i_1$, or equivalently, ${\delta_0}^q v_i \eta_1 = w_i$.

Now, we can consider $\delta_0 = (\delta_0)_X(\delta_0)_Y$, $\eta_1 = (\eta_1)_X(\eta_1)_Y$, $v_i = (v_i)_X(v_i)_Y$ and $w_i = (w_i)_X(w_i)_Y$ in the usual way. Then this equation gives us the two sets of equations, for every $1 \leq i \leq i_1$,

$$((\delta_0)_X)^q (v_i)_X (\eta_1)_X = (w_i)_X,$$

$$((\delta_0)_Y)^q (v_i)_Y (\eta_1)_Y = (w_i)_Y.$$

By Lemma 4.3.20, we can determine in linear space whether each of these equations either have zero solutions, one solution, or are satisfied for any q.

Hence, the whole system of equations has either zero solutions, one solution, or are satisfied for any q, since if any two of the equations are determined to have unique solutions, then we check that these two solutions are in fact identical. If they have no solution then obviously we reject the words.

Otherwise, we now consider the conjugates of b. Rearranging our conjugating equation, using the commutativity of the conjugates of a and b, we have

$$\left(\prod_{\beta_1\in\mathcal{B}_1}\beta_1^{-1}b\beta_1\right)\left(\prod_{\beta_2\in\mathcal{B}_2}(\beta_2\eta_1^{-1})^{-1}b\beta_2\eta_1^{-1}\right)(\eta_1\eta_2)^{-1}(v_1^{-1}av_1)..(v_{i_1}^{-1}av_{i_1})\mu_1\mu_2 =$$

$$\left(\prod_{\beta_3\in\mathcal{B}_3}\beta_3^{-1}b\beta_3\right)\left(\prod_{\beta_3\in\mathcal{B}_3}(\beta_3\mu_1^{-1})^{-1}b\beta_3\mu_1^{-1}\right)(w^{-1}aw_1)..(w_{i_1}^{-1}aw_{i_1})\mu_1\mu_2(\eta_1\eta_2)^{-1}.$$

We choose this slightly unusual formulation, involving $(\eta_1\eta_2)^{-1}$, simply to avoid having to work with inverses later on.

We now have to consider the additional conjugates of b obtained when rearranging the left hand side. Recall that $\eta_2 = (\delta_1 t^{j_1}....\delta_s t^{j_s})^q$, (for some q). We are concerned only with those j_i such that j_i is odd. Suppose that $j_{k_1}, ..., j_{k_m}$ are those j_i in question. Then it is clear from Lemma 4.3.6 that we obtain an additional conjugating element of b,

$$(\overline{\delta_1....\delta_{k_l}})(\overline{\delta_1....\delta_s})^{q_1}v_k$$

for any $1 \leq l \leq m$, $0 \leq q_i \leq (q-1)$, $1 \leq k \leq i_1$. Let $\mathcal{C}$ be this set of words. Since we defined $\delta_0 = \overline{\delta_1....\delta_s}$, then we have that

$$\mathcal{C} = \{(\overline{\delta_1....\delta_{k_l}})(\delta_0)^{q_1}v_k : 1 \leq l \leq m, 0 \leq q_i \leq (q-1), 1 \leq k \leq i_1\}.$$

Hence, we are looking for a set $\mathcal{B}_3$ satisfying

$$\mathcal{B}_1\triangle\mathcal{B}_2(\eta_1)^{-1}\triangle\mathcal{C} = \mathcal{B}_3\triangle\mathcal{B}_3(\mu_1)^{-1}.$$

Of course, we need to know the value of q to be able to form $\mathcal{C}$, and we note that this is a linearly expressible set, provided that $q = O(2^n)$.

Suppose we decided that the two sets of equations

$$((\delta_0)_X)^q(v_i)_X(\eta_1)_X = (w_i)_X,$$

$$((\delta_0)_Y)^q(v_i)_Y(\eta_1)_Y = (w_i)_Y$$

are solvable for only one q, which we have found. Note that we have that $(\delta_0)^q$ is linearly expressible from Lemma 4.3.20. We can then test the equation $\mathcal{B}_1\triangle\mathcal{B}_2(\eta_1)^{-1}\triangle\mathcal{C} = \mathcal{B}_3\triangle\mathcal{B}_3(\mu_1)^{-1}$ for this particular q, using Corollary 4.3.12, since all the sets in question are linearly expressible.

Otherwise, they are solvable for any q, and thus we need concern ourselves only with the equation $\mathcal{B}_1\triangle\mathcal{B}_2(\eta_1)^{-1}\triangle\mathcal{C} = \mathcal{B}_3\triangle\mathcal{B}_3(\mu_1)^{-1}$. Note that since they are solvable for any q, we can therefore combine the equations, and we can conclude that

$$(\delta_0)^q v_i \eta_1 = w_i$$

is also solvable for any value of q. Specifically, if $q = 0$, then we obtain

$$(v_i)^{-1}w_i = (v_1)^{-1}w_1$$

for any value of i, since both sides are equal to η_1. Similarly, using this result, and taking $q = 1$, we obtain

$$(v_i)^{-1}\delta_0 v_i = (v_1)^{-1}\delta_0 v_1$$

for any value of i.

Hence, we can assign $(v_i)^{-1}\delta_0 v_i$ to a word u_0, which is obviously still linearly bounded. Then, any word of $\mathcal{C}$, which by definition is of the form $(\overline{\delta_1....\delta_{k_l}})(\delta_0)^{q_1}v_k$, can be written in the form $(\overline{\delta_1....\delta_{k_l}})v_k(u_0)^{q_1}$. Hence we can consider the equation

$$\mathcal{B}_1\triangle\mathcal{B}_2(\eta_1)^{-1}\triangle\mathcal{C}'\mathcal{U}_q = \mathcal{B}_3\triangle\mathcal{B}_3(\mu_1)^{-1}$$

where we define the sets $\mathcal{C}'$ and $\mathcal{U}_q$ as

$$\mathcal{C}' = \{(\overline{\delta_1....\delta_{k_l}})v_k : 1 \leq m, 1 \leq k \leq i_1\},$$

$$\mathcal{U}_q = \{{u_0}^j : 0 \leq j \leq q-1\}.$$

We let $\omega = (w_1)^{-1}v_1$, and hence we wish to solve the equation

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\omega(u_0)^q) \triangle \mathcal{C}'\mathcal{U}_q = \mathcal{B}_3 \triangle \mathcal{B}_3(\mu_1)^{-1}$$

(since $(\eta_1)^{-1} = (w_1)^{-1}v_1(u_0)^{-q}$). This may seem a strange thing to do, since it appears to complicate our equation, but our intention is either to be able to reduce the problem to a small number of possible solutions, or to be able to apply Lemma 4.3.15.

We now use Lemma 4.3.28 to test whether or not $\langle u_0 \rangle \cap \langle \mu_1 \rangle = 1$. There are two possible cases.

Suppose, firstly, that we determine that $\langle u_0 \rangle \cap \langle \mu_1 \rangle \neq 1$. In this case, using Lemma 4.3.28, we can write a number s (in binary) in linear space such that $\langle u_0 \rangle^s \in \langle \mu_1 \rangle$. Note that this obviously implies that the word ${u_0}^{s_0}$ is linearly expressible for any $0 \leq s_0 \leq s$. We now claim that the equation

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\omega(u_0)^q) \triangle \mathcal{C}'\mathcal{U}_q = \mathcal{B}_3 \triangle \mathcal{B}_3(\mu_1)^{-1}$$

is solvable for q if and only if it is also solvable for $q - 2s$. To prove this claim, suppose firstly that the equation is solvable for q, that is we have

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\omega(u_0)^q) \triangle \mathcal{C}_1'\mathcal{U}_q = \mathcal{B}_3 \triangle \mathcal{B}_3(\mu_1)^{-1}.$$

Now, it is trivial that we have

$$\mathcal{B}_2(\omega(u_0)^{q-2s}) = \mathcal{B}_2(\omega(u_0)^q) \triangle \mathcal{B}_2(\omega(u_0)^{q-2s})(u_0)^{2s} \triangle \mathcal{B}_2(\omega(u_0)^{q-2s})$$

and also, by the definition of $\mathcal{U}_i$, it is immediately clear that

$$\mathcal{U}_{q-2s} = \mathcal{U}_q \triangle \mathcal{U}_{2s}(u_0)^{q-2s}$$

$$= \mathcal{U}_q \triangle (\mathcal{U}_s \triangle \mathcal{U}_s (u_0)^s)(u_0)^{q-2s}.$$

Putting all of this together, and using our assumption on the equation for q, we obtain the fact that $\mathcal{B}_1 \triangle \mathcal{B}_2(\omega(u_0)^{q-2s}) \triangle \mathcal{C}'\mathcal{U}_{q-2s}$ is equal to

$$(\mathcal{B}_3 \triangle \mathcal{B}_3 \mu_1^{-1}) \triangle \left(\mathcal{B}_2 \omega(u_0)^{q-2s} \triangle \mathcal{B}_2 \omega(u_0)^{q-2s}(u_0)^{2s}\right) \triangle \mathcal{C}'(u_0)^{q-2s}(\mathcal{U}_s \triangle \mathcal{U}_s(u_0)^s).$$

But since by assumption, $(u_0)^s$ is a power of μ_1, then we are left with three terms, each of which is of the form $\mathcal{G} \triangle \mathcal{G} {\mu_1}^k$, and hence the whole term is of this form. But then, by Lemma 4.3.13, it must also be expressible in the form $\mathcal{G}_1 \triangle \mathcal{G}_1 \mu_1$, which shows that the equation is also solvable for $q - 2s$. The converse argument (where the equation is solvable for $q - 2s$) is entirely similar, and this proves our claim. Given this, we therefore need to test only the cases where $0 \le q \le 2s$, which we can do by Corollary 4.3.12, since all the words in our sets are clearly linearly expressible.

The only remaining case is where we determine, from Lemma 4.3.28, that $\langle u_0 \rangle \cap \langle \mu_1 \rangle = 1$. If so, then we can apply Lemma 4.3.14 to deduce that the equation $\mathcal{B}_1 \triangle \mathcal{B}_2(\omega(u_0)^q) \triangle \mathcal{C}'\mathcal{U}_q = \mathcal{B}_3 \triangle \mathcal{B}_3(\mu_1)^{-1}$ holds if and only if

$$(\mathcal{B}_1 \triangle \mathcal{B}_1 u_0) \triangle \left(\mathcal{B}_2(\omega(u_0)^q) \triangle \mathcal{B}_2(\omega(u_0)^q) u_0\right) \triangle (\mathcal{C}'\mathcal{U}_q \triangle \mathcal{C}'\mathcal{U}_q u_0)$$

$$= \mathcal{B}_4 \triangle \mathcal{B}_4(\mu_1)^{-1}$$

for some set $\mathcal{B}_4$. It is clear that $\mathcal{U}_q \triangle \mathcal{U}_q u_0 = \{1, (u_0)^q\}$, since all other terms appear in both sets, and hence cancel. Hence,

$$\mathcal{C}'\mathcal{U}_q \triangle \mathcal{C}'\mathcal{U}_q u_0 = \mathcal{C}' \triangle \mathcal{C}'(u_0)^q.$$

Therefore, our equation reduces to

$$(\mathcal{B}_1 \triangle \mathcal{B}_1 u_0 \triangle \mathcal{C}') \triangle (\mathcal{B}_2 \omega \triangle \mathcal{B}_2(\omega u_0) \triangle \mathcal{C}')(u_0)^q = \mathcal{B}_4 \triangle \mathcal{B}_4(\mu_1)^{-1}$$

and this equation can be solved in linear space by Corollary 4.3.16, since all the sets in question are clearly linearly expressible by Lemma 4.3.8. □

The only remaining case is when μ_1 contains Z-symbols, and μ_2 contains T-symbols.

Lemma 4.3.30 *Suppose that μ_1 contains a Z-symbol, and μ_2 contains a T-symbol. Then there is a deterministic context-sensitive procedure to determine whether or not u_1 and u_2 are conjugate in G_{CS}.*

Proof This time, in the conjugating word we are looking for, η_2 is a word of $(X' \times Y') * T$, and η_1 is a word of $(X \times Y) * Z$. As usual, if $\mu_2 = (\delta_1 t^{j_1} .... \delta_s t^{j_s})^c$, where $|c|$ is maximal, then η_2 must be of the form $(\delta_1 t^{j_1} .... \delta_s t^{j_s})^q$, and we let $\delta_0 = \overline{\delta_1 .... \delta_s}$. Similarly, if $\mu_1 = (\epsilon_1 \zeta_1 .... \epsilon_r \zeta_r)^d$, with d maximal, then η_1 must be of the form $(\epsilon_1 \zeta_1 .... \epsilon_r \zeta_r)^p$, and we let $\epsilon_0 = \epsilon_1 .... \epsilon_r$. As in Lemma 4.3.29, we can assume that $q \geq 0$.

In a similar vein to the previous cases, equating conjugates of a in our conjugacy equation, we must have $(\delta_0)^q v_i (\epsilon_0)^p = w_i$. As usual, we consider $\delta_0 = (\delta_0)_X (\delta_0)_Y$, $\epsilon_0 = (\epsilon_0)_X (\epsilon_0)_Y$, $\eta_1 = (\eta_1)_X (\eta_1)_Y$, $v_i = (v_i)_X (v_i)_Y$ and $w_i = (w_i)_X (w_i)_Y$. Then, splitting in the usual way, this equation gives us the two sets of equations, for every $1 \leq i \leq i_1$,

$$(\delta_0)_X^q (v_i)_X (\epsilon_0)_X^p = (w_i)_X,$$

$$(\delta_0)_Y^q (v_i)_Y (\epsilon_0)_Y^p = (w_i)_Y.$$

We can solve these equations by Lemma 2.5.9, and hence there are either zero or exactly one common solution, or infinitely many solutions. From Lemma 2.5.9, there are only infinitely many solutions if

$$(v_i)_X^{-1} \delta_X (v_i)_X = (\epsilon_X)^{\pm 1},$$

$$(v_i)_Y^{-1} \delta_Y (v_i)_Y = (\epsilon_Y)^{\pm 1},$$

where $\delta_0 = (\delta_X)^{m_1} (\delta_Y)^{m_2}$ and $\epsilon_0 = (\epsilon_X)^{m_3} (\epsilon_Y)^{m_4}$, where every m_i is maximal.

If they have no solution then obviously we reject the words. Otherwise, we now consider the conjugates of b. Proceeding similarly to Lemma 4.3.29, we obtain the equation

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\epsilon_0)^{-p} \triangle \mathcal{C} = \mathcal{B}_3 \triangle \mathcal{B}_3(\epsilon_0)^{-d},$$

where $\mathcal{C}$ is defined exactly as in Lemma 4.3.29, namely

$$\mathcal{C} = \{(\overline{\delta_1 \ldots . \delta_{k_l}})(\delta_0)^{q_1} v_k : 1 \leq l \leq m, 0 \leq q_i \leq (q-1), 1 \leq k \leq i_1\}.$$

Suppose we decided that the two sets of equations

$$(\delta_0)_X^q (v_i)_X (\epsilon_0)_X^p = (w_i)_X,$$

$$(\delta_0)_Y^q (v_i)_Y (\epsilon_0)_Y^p = (w_i)_Y$$

are solvable for a unique p and q, which we note that we can determine in linear space by Lemma 2.5.9. We can then test the equation

$$\mathcal{B}_1 \triangle \mathcal{B}_2 \epsilon_0^{-p} \triangle \mathcal{C} = \mathcal{B}_3 \triangle \mathcal{B}_3 \epsilon_0^{-d}$$

for this particular p and q, using Corollary 4.3.12, since all the terms involved are obviously linearly expressible.

The only remaining case is where, for every i,

$$(v_i)_X^{-1} \delta_X (v_i)_X = (\epsilon_X)^{\lambda_1},$$

$$(v_i)_Y^{-1} \delta_Y (v_i)_Y = (\epsilon_Y)^{\lambda_2},$$

where $\lambda_1 = \pm 1$, $\lambda_2 = \pm 1$. Then the equation $(\delta_0)_X^q (v_i)_X (\epsilon_0)_X^p = (w_i)_X$ can be rewritten as

$$(\delta_X)^{m_1 q} (v_i)_X (\epsilon_X)^{m_3 p} = (w_i)_X,$$

or, using the equation above,

$$(\epsilon_X)^{m_3 p + \lambda_1 m_1 q} = (v_i)_X^{-1} (w_i)_X.$$

Similarly, we obtain

$$(\epsilon_Y)^{m_4p+\lambda_2m_2q} = (v_i)_Y^{-1}(w_i)_Y.$$

But we can solve these equations for $m_3p + \lambda_1m_1q$ and $m_4p + \lambda_2m_2q$ in linear space by Lemma 2.5.7. If we have no solution, then we reject the words. Otherwise, suppose we have determined that we have the solutions $m_3p + \lambda_1m_1q = r_1$ and $m_4p + \lambda_2m_2q = r_2$, where we note that the size of r_1 and r_2 are certainly of size no greater than the length of $(v_i)^{-1}w_i$, from Lemma 2.5.7, and so are obviously linearly bounded. Provided $\lambda_1m_1m_4 \neq \lambda_2m_2m_3$, then these equations have exactly one real solution, namely

$$p = (\lambda_2m_2r_1 - \lambda_1m_1r_2)/(\lambda_2m_2m_3 - \lambda_1m_1m_4)$$

$$q = (r_2m_3 - r_1m_4)/(\lambda_2m_2m_3 - \lambda_1m_1m_4).$$

But it is simple to determine whether these are integers, and indeed can be done in linear space, using simple binary arithmetic, since all the terms are at most quadratic in the length of the input. If we find integral p and q, then we can check to see if the equation

$$\mathcal{B}_1\triangle\mathcal{B}_2(\epsilon_0)^{-p}\triangle\mathcal{C} = \mathcal{B}_3\triangle\mathcal{B}_3(\epsilon_0)^{-d}$$

holds by Corollary 4.3.12, since all the terms are linearly expressible.

Hence, the only remaining case we have to consider, is the situation where $\lambda_1m_1m_4 = \lambda_2m_2m_3$. We claim that if p and q satisfy our equations, where $q > 2m_3d$, then so do the values $p + 2\lambda_1m_1d$ and $q - 2m_3d$ (recall that d is the maximal value such that $\mu_1 = (\epsilon_1\zeta_1....\epsilon_r\zeta_r)^d$). Note that we chose m_3 and d to be maximal,so they are certainly positive, and hence $2m_3d$ is positive. If this is true, then we need only check our equations for $0 \leq q \leq 2m_3d$, and the corresponding value of p (which, given q, can be determined from either the equation $m_3p + \lambda_1 m_1 q = r_1$, or the equation $m_4 p + \lambda_2 m_2 q = r_2$), which we can do by Corollary 4.3.12, since every term will obviously be linearly expressible, and we are done.

To prove the claim, we firstly consider the equation

$$(\delta_0)_X^q (v_i)_X (\epsilon_0)_X^p = (w_i)_X.$$

Note, that since we have $(v_i)_X^{-1}\delta_X(v_i)_X = (\epsilon_X)^{\lambda_1}$, $(\delta_0)_X = (\delta_X)^{m_1}$, and $(\epsilon_0)_X = (\epsilon_X)^{m_3}$, then this equation gives us

$$(v_i)_X(\epsilon_X)^{\lambda_1 m_1 q + m_3 p} = (w_i)_X.$$

Now, we can deduce that

$$(\delta_0)_X^{q-2m_3d}(v_i)_X(\epsilon_0)_X^{p+2\lambda_1 m_1 d}$$

$$= (v_i)_X(\epsilon_X)^{\lambda_1 m_1(q-2m_3d)+m_3(p+2\lambda_1 m_1 d)}$$

$$= (v_i)_X(\epsilon_X)^{\lambda_1 m_1 q + m_3 p} = (w_i)_X$$

and hence if this equation holds for p and q, it also holds for $p + 2\lambda_1 m_1 d$ and $q - 2m_3 d$.

Secondly, consider the equation

$$((\delta_0)_Y)^q (v_i)_Y ((\epsilon_0)_Y)^p = (w_i)_Y.$$

Similarly to the above, noting that $(v_i)_Y^{-1}\delta_Y(v_i)_Y = (\epsilon_Y)^{\lambda_2}$, $(\delta_0)_Y = (\delta_Y)^{m_2}$, and $(\epsilon_0)_Y = (\epsilon_Y)^{m_4}$, we obtain

$$(v_i)_Y(\epsilon_Y)^{\lambda_2 m_2 q + m_4 p} = (w_i)_Y.$$

But then, we have

$$((\delta_0)_Y)^{q-2m_3d}(v_i)_Y((\epsilon_0)_Y)^{p+2\lambda_1 m_1 d} = (v_i)_Y(\epsilon_Y)^{\lambda_2 m_2(q-2m_3d)+m_4(p+2\lambda_1 m_1 d)}.$$

But since $\lambda_1 m_1 m_4 = \lambda_2 m_2 m_3$ by assumption, this gives us simply

$$(v_i)_Y(\epsilon_Y)^{\lambda_2 m_2 q + m_4 p} = (w_i)_Y$$

and hence, again, if this equation holds for p and q, it also holds for $p+2\lambda_1 m_1 d$ and $q - 2m_3 d$.

We are left therefore, only to consider the equation

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\epsilon_0)^{-p} \triangle \mathcal{C} = \mathcal{B}_3 \triangle \mathcal{B}_3(\epsilon_0)^{-d},$$

which we assume holds (for some $\mathcal{B}_3$) for some p and q. Since we have written $\delta_0 = (\delta_X)^{m_1}(\delta_Y)^{m_2}$ and $\epsilon_0 = (\epsilon_X)^{m_3}(\epsilon_Y)^{m_4}$, we can follow a similar approach to Lemma 4.3.29, to rewrite this equation as

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\epsilon_X)^{-pm_3}(\epsilon_Y)^{-pm_4} \triangle \mathcal{C}'\mathcal{E}_q = \mathcal{B}_3 \triangle \mathcal{B}_3(\epsilon_X)^{-dm_3}(\epsilon_Y)^{-dm_4},$$

where we define $\mathcal{C}'$ exactly as in Lemma 4.3.29, and we define

$$\mathcal{E}_q = \{(\epsilon_X)^{\lambda_1 m_1 q_i}(\epsilon_Y)^{\lambda_2 m_2 q_i} : 0 \leq i \leq q-1\}.$$

Assuming this holds for p and q, now consider the expression

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\epsilon_X)^{-m_3(p+2\lambda_1 m_1 d)}(\epsilon_Y)^{-m_3(p+2\lambda_1 m_1 d)} \triangle \mathcal{C}'\mathcal{E}_{q-2m_3 d}.$$

To complete the proof, we need to show that this can be written in the form $\mathcal{S} \triangle \mathcal{S}(\epsilon_X)^{-dm_3}(\epsilon_Y)^{-dm_4}$ for some $\mathcal{S}$. For notational convenience, let us define

$$\mathcal{R}_1 = \left(\mathcal{B}_2(\epsilon_X)^{-pm_3}(\epsilon_Y)^{-pm_4}\right).$$

Then, clearly,

$$\begin{aligned}&\mathcal{B}_2(\epsilon_X)^{-m_3(p+2\lambda_1 m_1 d)}(\epsilon_Y)^{-m_4(p+2\lambda_1 m_1 d)} \\ &= \mathcal{R}_1 \triangle \mathcal{R}_1 \triangle \mathcal{R}_1(\epsilon_X)^{-2\lambda_1 m_1 m_3 d}(\epsilon_Y)^{-2\lambda_1 m_4 m_1 d},\end{aligned}$$

including some triviality. Again, for notational convenience, let us now define

$$\mathcal{R}_2 = \mathcal{C}'(\epsilon_X)^{\lambda_1 m_1(q-2m_3d)}(\epsilon_Y)^{\lambda_2 m_2(q-2m_3d)}.$$

Then, we clearly have

$$\mathcal{C}'\mathcal{E}_{q-2m_3d} = \mathcal{C}'\mathcal{E}_q \triangle \mathcal{R}_2\mathcal{E}_{2m_3d}.$$

Combining all of this, we obtain three terms, namely

$$\mathcal{T}_1 = \mathcal{B}_1 \triangle \mathcal{R}_1 \triangle \mathcal{C}'\mathcal{E}_q,$$

$$\mathcal{T}_2 = \mathcal{R}_1 \triangle \mathcal{R}_1(\epsilon_X)^{-2\lambda_1 m_1 m_3 d}(\epsilon_Y)^{-2\lambda_1 m_4 m_1 d},$$

$$\mathcal{T}_3 = \mathcal{R}_2\mathcal{E}_{2m_3d},$$

where these terms combine to give us

$$\mathcal{B}_1 \triangle \mathcal{B}_2(\epsilon_X)^{-m_3(p+2\lambda_1 m_1 d)}(\epsilon_Y)^{-m_3(p+2\lambda_1 m_1 d)} \triangle \mathcal{C}'\mathcal{E}_{q-2m_3d} = \mathcal{T}_1 \triangle \mathcal{T}_2 \triangle \mathcal{T}_3.$$

But, $\mathcal{T}_1$, is by assumption, equal to $\mathcal{B}_3 \triangle \mathcal{B}_3(\epsilon_X)^{-dm_3}(\epsilon_Y)^{-dm_4}$. It is obvious that $\mathcal{T}_2$ can be written in the required form by Lemma 4.3.13. Finally, since $\lambda_1 m_1 m_4 = \lambda_2 m_2 m_3$, we clearly have

$$\mathcal{E}_{2m_3d} = \mathcal{E}_{m_3d} \triangle \mathcal{E}_{m_3d}((\epsilon_X)^{\lambda_1 m_1 m_3 d}(\epsilon_Y)^{\lambda_2 m_2 m_3 d})$$

$$= \mathcal{E}_{m_3d} \triangle \mathcal{E}_{m_3d}((\epsilon_X)^{\lambda_1 m_1 m_3 d}(\epsilon_Y)^{\lambda_2 m_1 m_4 d}),$$

and hence the final term is also of the required form, by Lemma 4.3.13, and thus so is the whole word when we combine them together, and we have proved the claim, and hence we are done. □

4.3.5 The main theorem

We can now put all these results together to give us the required result for G_{CS}.

Lemma 4.3.31 *G_{CS} has deterministic context-sensitive conjugacy problem.*

Proof We have considered every possibility for μ_1 and μ_2 in Lemma 4.3.25, Lemma 4.3.26, Lemma 4.3.27, Lemma 4.3.29, and Lemma 4.3.30, and hence we can always determine whether or not a conjugating element exists. In every case, our procedure is deterministic, and linearly bounded. □

Finally, then, we have the main result.

Theorem 4.3.32 *There exists a group with deterministic context-sensitive conjugacy problem, which is a subgroup of index 2 of a group with unsolvable conjugacy problem.*

Proof Take the groups G_{CS} and G_U as defined above. □

This result serves to demonstrate just how difficult the conjugacy problem can be. Even by taking a relatively small extension of a group, of index 2, and even if the group has the comparatively strong condition of having context-sensitive conjugacy problem, there is no bound on the complexity of the conjugacy problem in the larger group, and indeed can be unsolvable, as we have shown. This is in sharp contrast to the situation for the word problem, and shows again the difficulty of the conjugacy problem.

Chapter 5

Embedding a group with context-sensitive word problem

In this brief chapter, let us consider possibilities for embedding groups with context-sensitive word problem into other groups. An *embedding* of a group H into a group G is simply an injection from H to G, and so if H embeds into G, then we may consider H to be a subgroup of G.

5.1 Embedding into two-generator groups

Let us show that we can embed any finitely generated group with (deterministic) context-sensitive word problem into a group with only two generators with (deterministic) context-sensitive word problem. The basis of the construction is the one used on p188 of [37], but we require some substantial expansion, and additional techniques to illustrate the context-sensitive nature of our procedure. Given a word u, we define the *α-exponent sum* of u to be $|u|_\alpha - |u|_{\alpha^{-1}}$, that is the sum of the exponents of all the occurrences of α in the word. We also define the *k'th partial α-exponent sum* of u to be the

a-exponent sum of the prefix of u of length k (where, for a word of length n, a prefix of length m for $m > n$ is defined simply to be the whole word). We will give a sequence of lemmas, from which the final result will then follow easily.

We start with a free group $F = \langle a, b : \emptyset \rangle$. Consider the subgroup K_1 of F generated by the elements $Y = \{b^{-i}ab^i : i \geq 0\}$.

Now take a group $H = \langle X : R \rangle$, where $X = \{h_1, ..., h_d\}$, and suppose H has context-sensitive word problem. We form the direct product $L = H \times F$. By Lemma 3.2.1 and Lemma 2.5.3, L also has context-sensitive word problem.

Consider the subgroup K_2 of $H \times F$ generated by the elements $Z = \{c_i a^{-i} b a^i : i \geq 0\}$, where $c_i = h_i$ for $1 \leq i \leq d$ and $c_i = \lambda$ otherwise. We do not feel the need to prove the following result, since it does not affect our complexity considerations.

Lemma 5.1.1 *K_1 and K_2 are freely generated by their generators, and hence K_1 and K_2 are isomorphic.*

Proof See page 188 of [37]. □

Lemma 5.1.2 *There is a deterministic context-sensitive procedure to solve the generalised word problem of F with respect to K_1.*

Proof We claim that a freely-reduced word in F lies in K_1 if and only if the b-exponent sum is zero, and all the partial b-exponent sums are non-positive.

In one direction, suppose u is a word in K_1. Since there are no relations between a and b, apart from the trivial ones, u must be formed by a concatenation of words of the form $b^{-i}ab^i$ or $b^{-i}a^{-1}b^i$, possibly followed by some free reduction. But it is then immediately evident that any word in K_1 has b-exponent sum zero, and all the partial b-exponent sums are non-positive (since any concatenation of these words has both of these properties, and free reduction does not affect either of these properties).

Conversely, suppose we have a word u such that the b-exponent sum of u is zero, and all the partial b-exponent sums are non-positive. We can write u in the form $b^{i_1}a^{j_1}b^{i_2}a^{j_2}.....b^{i_m}a^{j_m}b^{i_{m+1}}$ (where i_1 and/or i_{m+1} may be zero). This is equivalent to the word

$$b^{i_1}a^{j_1}b^{-i_1}b^{i_1+i_2}a^{j_2}b^{-(i_1+i_2)}.....b^{i_1+...+i_m}a^{j_m}b^{-(i_1+...+i_m)}$$

(since $i_1+.....+i_{m+1}=0$). However, the partial b-exponent sums are necessarily non-positive, and so this consists of a product of terms of the form $b^{-p}a^qb^p$ with $p \geq 0$, which are products of generators of K_1, since $b^{-p}a^qb^p = (b^{-p}ab^p)^q$. Hence u must be an element of K_1.

Thus, to determine membership of K_1, we simply need to check the associated b-exponent sums, and this procedure can obviously be done in linear space. □

Lemma 5.1.3 *There is a context-sensitive procedure to solve the generalised word problem of L with respect to K_2.*

Proof Suppose we have an input word u. Firstly, since u is a word in L, we rearrange the symbols in u so that it is of the form $u = u_Hu_F$, with u_H containing only symbols in the h_i, and u_F containing only a and b (and of course their inverses).

There are two parts to the checking of u. Firstly, we need to check whether u_F lies in the subgroup generated by $\{a^{-i}ba^i : i \geq 0\}$. We can achieve this simply by following the proof of Lemma 5.1.2, interchanging a and b. Hence we merely need to check the appropriate a-exponent sums. If it does not lie in this subgroup, then we reject u immediately.

So, suppose u_F does indeed lie in this subgroup. We can write u_F in the form $a^{-i_1}b^{j_1}a^{-i_2}b^{j_2}.....a^{-i_m}b^{j_m}a^{-i_{m+1}}$ (where i_1 and/or i_{m+1} may be zero). This is equivalent to the word

$$a^{-i_1}b^{j_1}a^{i_1}a^{-(i_1+i_2)}b^{j_2}a^{(i_1+i_2)}.....a^{-(i_1+...+i_m)}b^{j_m}a^{(i_1+...+i_m)}$$

(since $i_1 + ... + i_{m+1} = 0$), where each of the sums $i_1 + ... + i_k$ must be positive.

Hence, if u lies in K_2, then u_H must be equivalent to the word

$$\gamma = (c_{i_1})^{j_1}....(c_{i_1+....+i_m})^{j_m}.$$

We can easily calculate each $i_1 + ... + i_k$, and thus write down γ, which is of length no greater than $l(u_F)$. Hence, we can test $u_H\gamma^{-1}$ in the context-sensitive word problem solver for H. Note that the length of this word is certainly no greater than the length of u, and hence we have a context-sensitive procedure and we are done. □

We can form the HNN extension of L,

$$G = \langle L, t : t^{-1}b^{-i}ab^{i}t = c_i a^{-i}ba^{i}\}.$$

G will be the group we are looking for. Note that $K_1 \cap K_2 \neq \{1\}$. For example, the element $b^{-2}a^{-(d+1)}ba^{d+1}b$ lies in both subgroups, since it is equivalent to $(b^{-2}ab^2)^{-(d+1)}(b^{-1}a^{n+1}b)$, which lies in K_1, and equivalent to $b^{-2}(a^{-(d+1)}ba^{d+1})b$, which lies in K_2. Hence we cannot use Theorem 4.1.4 here to determine that G has a context-sensitive word problem. Thus, we will give a direct algorithm.

Lemma 5.1.4 *G has context-sensitive word problem.*

Proof Suppose we are given an input word u of length n. If u contains no occurrence of $t^{\pm 1}$, then we test u for equivalence to the identity in L (which we know from above is a context-sensitive procedure) and we are done.

Otherwise, we search for a subword of the form $t^{-1}k_1t$ or tk_2t^{-1} for k_i in K_i (we know that testing membership of these subgroups is a context-sensitive procedure from above). If no such subword exists, then, by Britton's Lemma, the word cannot be equivalent to the identity and we are done. Otherwise, we can use the relations $t^{-1}b^{-i}ab^it = c_ia^{-i}ba^i$. If we find a subword $t^{-1}k_1t$, with k_1 in K_1, then we replace this with the word obtained by interchanging the a and b, and adding in the appropriate h_i, as above. Similarly, if we find a subword $t^{-1}k_2t$, with k_2 in K_2, then we replace this with the word obtained by interchanging the a and b, and removing any h_i that occur.

We continue in this vein until we either eliminate all occurrences of $t^{\pm 1}$ (in which case we are left with a word in L and we simply test this word for equivalence to the identity in L), or we can perform no more pinchings, and then the word cannot possibly be equivalent to the identity, and so we reject it.

It is clear that this algorithm performs the required task. Finally, we need to demonstrate that it is indeed a context-sensitive procedure. We will use Lemma 2.1.1 to demonstrate that the length of the word in question as we perform pinches is linearly bounded, then since all computations on the word (for example testing the word problem in L, or calculating the appropriate c_1, are (deterministic) context-sensitive procedures, we are done.

Let us take an input word u of length n, and suppose that we are some way through the algorithm on the word, and we have a word v. Let T represent the $t^{\pm 1}$, and similarly for A and B, representing the $a^{\pm 1}$ and $b^{\pm 1}$ respectively. The number of T-symbols in the word clearly cannot increase, since any pinching removes two occurrences of T-symbols, and hence $|v|_T \leq |u|_T \leq n$. Also, no operation - apart from free reduction which obviously decreases the number of A-symbols or B-symbols - ever alters the total number of A-symbols and B-symbols (since we just interchange them) in a word, and hence $|v|_A + |v|_B \leq |u|_A + |u|_B \leq n$.

Finally, then, we need to consider the ${h_i}^{\pm 1}$. However, the ${h_i}^{\pm 1}$ appear only on the right hand side of finitely many of our pinching relations, and since our procedures to decide the effective generalised word problem are linearly bounded, and do not increase the number of h_i in any word already containing an h_i, we can use Lemma 4.1.7 to linearly bound the number of occurrences of the ${h_i}^{\pm 1}$ in any word.

Hence, by Lemma 2.1.1, this is a context-sensitive procedure, since there is a linear bound on the number of occurrences of any symbol. □

We can now deduce the required result.

Theorem 5.1.5 *Suppose H is a group with (deterministic) context-sensitive word problem. Then H can be embedded into a two-generator group with (deterministic) context-sensitive word problem.*

Proof We embed H into the group G as in the construction above, where G has context-sensitive word problem. Since we have the relations

$$t^{-1}b^{-i}ab^{i}t = h_i a^{-i}ba^{i}$$

for $1 \leq i \leq d$, we can eliminate the h_i from this generating set. Also, we have the relation $b = t^{-1}at$, and so we can also eliminate b. Hence, G can be given by a presentation involving only two generators a and t, which still has context-sensitive word problem by Lemma 2.2.2.

Finally note that all the procedures above are deterministic, provided that H has deterministic context-sensitive word problem. $\square$

5.2 Higman's result

Without doubt, the most famous and remarkable result concerning embeddings of groups is the result of Higman from [26].

Let $G = \langle X : R \rangle$ be a finitely generated group, and suppose that the set of relations R is a recursively enumerable set. Then we say that G is *recursively presented*. The famous result of [26] asserts the following.

Theorem 5.2.1 *A finitely generated group G can be embedded in a finitely presented group if and only if G is recursively presented.*

The obvious question to ask, is what can one say about embeddings of groups with context-sensitive word problem? Clearly, any group with context-sensitive word problem is recursively presented (for example, take as our relators every word equivalent to the identity, since this a context-sensitive set, it is clearly recursively enumerable), and hence any group with context-sensitive word problem can be embedded in a finitely presented group. But there is no reason why this group need also have context-sensitive word problem. However, we have the following conjecture.

Conjecture 5.2.2 *Given any finitely generated group G with (deterministic) context-sensitive word problem, G can be embedded in a finitely presented group with (deterministic) context-sensitive word problem.*

One of the reasons behind us making this conjecture, is that the result is true in semigroups, that is any finitely generated semigroup S with (deterministic) context-sensitive word problem can be embedded in a finitely presented semigroup H with (deterministic) context-sensitive word problem. This follows from the following result of Birget in [7].

Theorem 5.2.3 *Suppose S is a finitely generated semigroup with solvable word problem. Then S can be embedded in a finitely presented semigroup H such that there is a length-contracting, deterministic linear time reduction from the word problem of H to the word problem of S.*

In particular, then, since any procedure operating in linear time certainly operates in linear space, and since the reduction procedure is length-contracting, we certainly have an algorithm operating in linear space for the word problem of H, since we just apply the reduction and then solve the word problem in S.

The situation for groups is much more difficult. There is no reason, even if the semigroup in question is actually a group, that the finitely presented semigroup in which it embeds need be a group. Some work on the embedding for groups has been done in [8] and [53]. These papers are extremely long and technical, however they do seem to imply (personal communication with the authors) that the situation holds for polynomial space, that is that every group with polynomial space word problem embeds in a finitely presented group with polynomial space word problem. However, for now, the linear space issue remains an open question.

Chapter 6

The word problem of subgroups of automatic groups

We now consider the concept of automatic groups, which have attracted a great deal of attention.

6.1 Automatic groups

Automatic groups have been studied in some detail in recent years, and an algorithm for solving the word problem in time $O(n^2)$ was given in [18]. Here we make this algorithm precise, in the sense that we give the details of its implementation on a Turing machine. It is worth noting that, with time bounds, we may be dependent on the number of worktapes in our Turing machine. For example, it may be possible to implement an algorithm in time $O(n^2)$ using two worktapes, but not using just one tape.

6.1.1 Definition

Let us first define an automatic group.

Definition 6.1.1 *Suppose G is a group. Then an* automatic structure *for G consists of an alphabet A representing a monoid generating set for G, a finite state automaton W over A, and a collection of finite state automata M_x over (A, A) (for $x \in A \cup \{\lambda\}$) which satisfy*

- *The map $\pi : L(W) \to G$ is surjective (so every element of G has at least one representative in $L(W)$), where $L(W)$ represents the language accepted by the automaton W.*

- *For all $x \in A \cup \{\lambda\}$, we have $(w_1, w_2) \in L(M_x)$ if and only if the group elements represented by $w_1 x$ and w_2 are equal, and w_1 and w_2 are both in the language $L(W)$.*

If these conditions are satisfied then we say that G is automatic*.*

It should be noted that to read in pairs of words over A on an automaton we introduce a *padding symbol* \$ to the alphabet. Suppose we wish to read in the pair (α, β), with $\alpha \equiv a_1a_2.....a_n$ and $\beta \equiv b_1b_2.....b_m$ with $a_i, b_i \in A$, and suppose $n > m$. Then we read in pairs

$$(a_1, b_1), (a_2, b_2), ..., (a_m, b_m), (a_{m+1}, \$), ..., (a_n, \$)$$

and obviously we have a similar definition for the case when $m > n$. Of course, we need no padding symbol when $m = n$. This construction enables us to consider pairs of words over the alphabet A.

The basic idea behind all this is simple. We merely, given a word over A, wish to check if it is in our acceptable language $L(W)$, and given a pair of words in $L(W)$, wish to be able to check with a finite state automaton whether the words they represent as elements of G either represent the same element of G, or differ only by the multiplication of a generator. We call the automaton W the *word acceptor*, the automaton M_λ the *equality checker*, and the remaining automata M_x the *multipliers*.

We may assume, by [18], that this is an automatic structure with uniqueness (that is each element of the group has precisely one representative in the accepted language L), and that all of the automata in our structure are normalised, that is we have no dead states (states that can never be reached from the initial state).

The *Cayley graph* for a group (with respect to a group generating set X) is a graph where each element of the group is represented by a vertex and two vertices v_1, v_2 are joined by an edge from v_1 to v_2 if $v_1 = v_2 x$ for some $x \in X$. We have the notion of the *distance* between two elements, g_1 and g_2, in G, which we define as the length of the shortest path between them in the Cayley graph, denoted by $d(g_1, g_2)$. We can then define the *length* of an element g to be $l(g) = d(1, g)$. For a word w, we let $\widehat{w}$ denote the path labelled by w in the Cayley graph of G.

The following lemma characterises automatic groups.

Lemma 6.1.2 *Let G be a group with monoid generating set A, and W a finite state automaton over A with the natural map $\pi : L(W) \to G$ surjective. Then G is automatic with automatic structure (A, W) if and only if there exists a number d such that, for any two words w_1 and w_2 accepted by W such that $w_1 x$ represents the same element of G as w_2 for some $x \in A \cup \{\lambda\}$ (ie the vertices of the Cayley graph are of distance of no more than 1 apart), then the paths $\widehat{w_1}$ and $\widehat{w_2}$ are a uniform distance less than d apart in the Cayley graph.*

Proof See for example [18]. $\square$

6.1.2 The word problem of automatic groups

Let us now consider the word problem of automatic groups.

Theorem 6.1.3 *Let G be an automatic group. Then G has word problem solvable in time $O(n^2)$ on a deterministic Turing machine with three work-tapes.*

Proof We expand on the proof of [18].

Suppose G has an automatic structure (A, W), which we may assume is a normalised automatic structure with uniqueness, and with multiplier automaton M_x for x in A (note that we do not require an equality checker, since we have an automatic structure with uniqueness).

We will begin by writing a description of all the automata in our automatic structure. We use a similar approach to the standard description of Turing machines to write a description of all these automata on a tape. For a given group, obviously, this tape is of constant length, and hence does not affect our complexity considerations. Suppose the length of this tape is D. We can then give an algorithm purely in terms of the automata in our structure, then using this tape to simulate the methods in the automata when we consider the complexity of our algorithm. Let C be a bound on the number of steps required to simulate one move in any of our automata, or write down a state description. The actual value of C (and, in some sense, a precise definition of what we mean by simulating a move) is of course irrelevant, all that really matters is that it is some constant value.

Having written the description of the automata, we can now proceed to the description of our main algorithm. What we wish to do is, given an input word w, find a word in our accepted language $L(W)$ equivalent in G to w. Then, since we have an automatic structure with uniqueness, we merely need to check if this is the word in $L(W)$ representing the identity (which we will denote by $e \in A^*$).

Suppose $w \equiv a_1......a_n$. We start by reading in a_1 on our input tape. The aim is to follow a path in M_{a_1}, where the first components of our pairs of words labelling the edges are $e\$\$\$.....$, until we reach an accept state. Reading the second components of this path as we follow it gives us a word w_1 in L representing $ea_1 = a_1$. We can then repeat this process in M_{a_2} to find a word w_2 in L with $w_2 = w_1a_2 = a_1a_2$. We continue like this until we find $w_n = a_1.....a_n \equiv w$.

Given this informal idea, it remains to determine exactly how to find this path in each case. We will use the remaining two worktapes to perform this procedure, one of which will be used to store the word in question, and one of which will be used to determine an appropriate path.

Suppose we have a word w_{i-1} written on the second worktape. We write on the third tape a sequence of sets of states in M_{a_i} that could possibly be reached if we read in this word. To do this, we start by writing down the list of all states that can be reached from the start state (that is, those states that are connected to the start state via an edge with first component labelled as the first symbol of w_{i-1}). Note that we only need to include each state once in the set. Once we have written all possible states, we write the symbol we are considering (so that we have it for reference later once we have deleted our old word) and then some marker symbol. We then consider each of these states, and write down all the states that can then be reached from these states when reading in the second symbol of w_{i-1} (again recording each state only once), and continue in this vein, until we have read all of w_{i-1}. We then continue, writing all the states that can be reached by reading a \$ in the first component, until we reach an accept state, at which point we stop. This procedure must terminate, the reason being that there are no dead states since the automaton is normalised, and once we have read in all of the first word and are reading the $ in the first component, then we can have no more loops (or this loop would represent the identity which we could follow as many times as we liked and would contradict the uniqueness of the automatic structure). And hence we are certain to reach an accept state eventually.

To determine our word, informally all we have to do is work 'backwards' through this tape to write the word. However, we have to be slightly careful in how we go about writing the word to ensure our complexity bounds are met. There are several ways to do this, and we will give one method here.

We firstly clear the second worktape since we have no further need for the old word, and move to the start of this tape. We scroll backwards through the third tape, advancing the head on the second tape every time we pass a marker symbol indicating the end of the description of a new set of states. This positions the head on the second worktape at the end of where our word should be written. We then scroll back to the end of the third worktape, and work backwards through the states. We find a state that can lead to the accept state, with the appropriate first component on the corresponding edge (recall that we noted this first component at the end of our set of states), and write down the appropriate symbol, and continue in this vein until we reach the start of both tapes. Note that it does not matter which choice of 'previous' state we make - the point is that, considering the i'th state, no matter which previous state we choose, we know we can reach the $(i-1)$'th state and there is a path from the this state to the i'th state, and so we can choose any preceding state we like (as long as it is connected to the current state, which we of course check). Hence we simply choose the first state we encounter which satisfies our requirements. Finally we simply clear the third worktape, ready to continue our whole algorithm on the new word.

There are, of course, many other ways of calculating the required word but this seems to be one of the more efficient ways since it leaves the word in the correct form on our second worktape, without any need to reverse it or alter it in any other way.

Given this algorithm, it merely remains to show that it has time complexity $O(n^2)$. We define N to be a number greater than the number of states in any of our automata in the automatic structure.

To find w_i from any word w_{i-1} cannot take more than $|w_{i-1}| + N$ steps in an automaton, since we have to read in all of w_{i-1}, and then we can have no further loops, and hence we can have no more than N \$'s to read before we reach an accept state. Recall that we defined C as a bound on the number of steps in our Turing machine to simulate any one move, where C is a constant. Here, we wish to simulate up to N^N moves at once (since there could be up to N states in any set of 'possible' states, which could have edges to up to N other states), and write down all the corresponding states. Although this is a large amount of calculations to consider, the number of steps is still bounded by a constant value. The precise value of this constant is unimportant so we choose to omit the technicalities of the calculation, and simply let C' be a bound for this number of steps. Then, to write the set of 'possible' states on the third worktape takes no more than $C'(|w_{i-1}| + N) + 2$ steps (since we have to write the symbol in question, and the marker symbol).

We then scroll backwards and forward through this tape, then backwards again, to write the appropriate word, and then forward and backwards again to clear the tape - it is quite clear that this whole process therefore takes no more than $6(C'(|w_{i-1}| + N) + 2)$ steps, taking into account the six times we pass along the tape.

Therefore, to find w_1 takes no more than $6(C'(1+N)+2)$ steps, finding w_2 takes no more than $6(C'(|w_1|+N)+2) \leq 6(C'(1+2N)+2)$ steps, and similarly all of the w_i can be found from w_{i-1} in no more than than $6(C'(1+iN)+2)$ steps on our Turing machine. At the conclusion of our algorithm, we test to see if we have the outcome e, in $|e|$ steps.

Thus the total number of steps to find our equivalent word to w is bounded by

$$|e| + \sum_{i=1}^{n} 6(C'(1+iN)+2)$$

$$= |e| + 6(C'+2)n + \frac{6C'Nn(n+1)}{2} = O(n^2)$$

as required. □

Corollary 6.1.4 *Let G be an automatic group. Then G has (deterministic) context-sensitive word problem.*

Proof The algorithm as given above is entirely context-sensitive, since the tape storing the state information is of constant length, and the longest word w_i that we use is bounded in length by $1+nN$, and hence both of the other worktapes are linearly bounded, as required. Finally, this procedure is entirely deterministic. □

Note that this result will also follow directly from the results in [54], which we will discuss later.

An important point to note with regard to this algorithm is that the time and space bounds are satisfied at the same time.

Corollary 6.1.5 *The word problem for automatic groups is solvable with an algorithm working (with three worktapes) in space $O(n)$ and time $O(n^2)$ simultaneously.*

Proof Immediate from the above algorithm. □

We have considered here the word problem. The situation with regard to the conjugacy problem is still unclear.

Open Question 6.1.6 *Is the conjugacy problem for automatic groups solvable? If so, what complexity bounds can be put on it?*

Note that we are able to say that the conjugacy problem in the (possibly) smaller class of *biautomatic groups* is solvable (though the algorithm given is multiply exponential in space), for a discussion of biautomatic groups and a proof of this result see [18].

At this point it would be an natural question to ask whether or not the automatic groups characterise the class of groups with word problem solvable in time $O(n^2)$ on a deterministic Turing machine, and especially from our point of view, to ask about characterising the class of groups with context-sensitive word problem. As we will show, the answer to both of these questions is negative.

6.2 The Heisenberg group

The *Heisenberg group* is given by the group presentation

$$H = \{a, b, c : abc = ba, ac = ca, bc = cb\}.$$

The following lemma is well known.

Lemma 6.2.1 *The Heisenberg group H is not automatic.*

Proof See for example [18]. $\square$

However, as the following result shows, this group has word problem solvable in time $O(n^2)$, and hence the automatic groups do not categorise the class of groups with word problem solvable in time $O(n^2)$.

Lemma 6.2.2 *The Heisenberg group H has word problem solvable in deterministic time $O(n^2)$.*

Proof Firstly note that from the relation $ba = abc$, we can deduce the following relations:

$$ba = abc, b^{-1}a = ab^{-1}c^{-1}, ba^{-1} = a^{-1}bc^{-1}, b^{-1}a^{-1} = a^{-1}b^{-1}c.$$

Given these relations, it is clear that any word u in the Heisenberg group can be written in the form $u = a^i b^j c^k$ where

- a^i is equivalent to the word obtained by projecting u onto $\langle a \rangle$ (that is, the word obtained by simply writing down each occurrence of a or a^{-1} in u),
- b^j is equivalent to the word obtained by projecting u onto $\langle b \rangle$,
- c^k is equivalent to the word $\gamma_0\gamma_1 \ldots \gamma_m$, where m is given by $|u|_b$, γ_0 is equivalent to the word obtained by projecting u onto $\langle c \rangle$, and each γ_i (for $i \geq 1$) is equivalent to the word obtained by projecting the suffix of u, starting from the i'th occurrence of $b^{\pm 1}$, onto $\langle a \rangle$, and then replacing each a by c, and each a^{-1} by c^{-1}, if the i'th occurrence of $b^{\pm 1}$ is b, and replacing each a by c^{-1}, and each a^{-1} by c otherwise.

This equivalent form of u is immediately evident when one considers rearranging u by 'pushing' all the b and c to the end of u. Whenever a $b^{\pm 1}$ 'passes' an $a^{\pm 1}$ it adds in an extra $c^{\pm 1}$.

Thus, our algorithm works by first of all writing down on three worktapes (one for each symbol) the projections onto $\langle a \rangle$, $\langle b \rangle$ and $\langle c \rangle$. For example, to get the projection onto $\langle a \rangle$ we simply read through the input word. Every time we read an a we consider the word on our first worktape, and add an a to it if the word consists of a's so far (or is empty), and remove an a^{-1} if it consists of a^{-1}'s (since we have cancellation between a and a^{-1}), and of course similarly when we read an a^{-1}, and similarly for the other symbols. This allows us to store efficiently the equivalent word to each projection.

We then return to the start of each tape, and now add in the additional $c^{\pm 1}$. To do this, we simply scroll through the input word, and every time we read a $b^{\pm 1}$, we mark the position on the word, scroll to the end and add the appropriate $c^{\pm 1}$ to the third worktape (simply by adding the appropriate $c^{\pm 1}$ every time we read an $a^{\pm 1}$, and similarly for a^{-1}), and then return to our marked position.

At the conclusion of this procedure, we have the appropriate equivalent form to u written on the three worktapes, and then to solve the word problem we merely check to see whether or not these tapes are empty.

The number of steps needed for this algorithm is clearly bounded by

$$1 + 2n + 2\sum_{i=1}^{n} i$$

(where the $2n$ occurs due to forming the initial projections and returning to the start of the word, the sum occurs due to considering each suffix and returning to the marked point, and the 1 is the final step checking if the tapes are empty). This is obviously a quadratic bound and we are done.

Note that this algorithm is entirely deterministic. □

The algorithm as given above is not linearly bounded, since the number of c that we wish to store may be $O(n^2)$ which is obviously not linear in space. However we can easily modify the algorithm to show that the word problem for the Heisenberg group is indeed deterministic context-sensitive[1].

Corollary 6.2.3 *The Heisenberg group has deterministic context-sensitive word problem.*

Proof Simply use the above technique, but store the appropriate word on the third worktape as a binary number representing the power of c that we are currently working with. □

Of course, the reason that we did not implement this binary method of storage in the original algorithm, is due to the fact that the computation of adding or subtracting one to a binary word of length m is of order $O(m)$ (whereas just adding or removing a symbol takes just one step) and so we no longer have an $O(n^2)$ algorithm.

6.3 Subgroups of automatic groups

6.3.1 The word problem of subgroups of automatic groups

The Heisenberg group gives an example of a non-automatic group with word problem solvable in time $O(n^2)$ and also a non-automatic group with (deter-

[1]In fact, the word problem for the Heisenberg group is *real-time*, and we shall comment on this class of languages later.

ministic) context-sensitive word problem. Let us now move on a stage and ask about subgroups of automatic groups.

Lemma 6.3.1 *Suppose G is an automatic group, and $H \leq G$. Then the word problem of H is solvable by a deterministic algorithm operating simultaneously in space $O(n)$ and time $O(n^2)$, with three worktapes.*

Proof The class of languages decidable by an deterministic algorithm operating in both space $O(n)$ and time $O(n^2)$ with three worktapes is closed under inverse homomorphism and intersection with regular languages (this is a simple observation), and hence this result follows directly from Lemma 2.2.6 and Lemma 6.1.5. $\square$

This opens up our field of view somewhat, in lieu of the fact that subgroups of automatic groups need not themselves be automatic. It has been an open question as to whether or not subgroups of automatic groups may provide the classification of groups with context-sensitive word problem that we seek. We will proceed to eliminate this possibility.

6.3.2 Small cancellation theory

The idea of small cancellation theory will play an important part in our approach. We start with a free group F, which has group generating set $X = \{x_1, ...x_m\}$. Let R be a subset of F, that is a set of words over $X \cup X^{-1}$. Suppose R satisfies the following conditions.

- Every element of R is cyclically reduced,
- For every u in R, u^{-1} also lies in R,
- For every u in R, every cyclic permutation of u also lies in R.

Such an R is said to be *symmetrised.*

Suppose that $u_1 \equiv st_1$ and $u_2 \equiv st_2$ are distinct elements of R. Then s is said to be a *piece* (relative to R). Note that since s is cancelled in the product ${u_1}^{-1}u_2$, and R is symmetrised, a piece is simply a subword of an element of R which is cancelled in the product of two (non-inverse) elements of R. The fundamental idea of small cancellation theory is that pieces are small parts of elements of R. Formally, we define this as follows.

Definition 6.3.2 *Suppose R is a symmetrised subset of a free group F as above. Then R satisfies the* small cancellation condition $C'(\tau)$ *if, for any $u = st$ in R where s is a piece, we have $|s| < \tau|u|$.*

We will be particularly interested in groups given by a presentation $G = \langle X : R\rangle$ where R satisfies $C'(\frac{1}{6})$, which are known as *sixth-groups.*

6.3.3 Some preliminary lemmas

Let us set up the necessary preliminary lemmas for our result, from which our final conclusion will follow easily.

Suppose $\mathcal{F}$ is a class of proper complexity functions. We use the notation $\mathbf{NSPACE}(\mathcal{F})$ and $\mathbf{DSPACE}(\mathcal{F})$ to denote complexity classes, where a language L lies in $\mathbf{NSPACE}(\mathcal{F})$ if and only if L lies in $\mathbf{NSPACE}(f(n))$ for some $f(n) \in \mathcal{F}$. We will assume that $\mathcal{F}$ satisfies the following condition.

Condition 6.3.3 *$\mathcal{F}$ is a class of complexity functions satisfying the properties that*

- *$f(n) \geq cn$ (some constant c) for every $f(n) \in \mathcal{F}$,*
- *if $f(n) \in \mathcal{F}$, then $g(n) \in \mathcal{F}$, where $g(n) = f(2n)$.*

For example, the class of non-constant polynomial functions satisfies this condition, as does the class of linear functions $f(n) = c_1 n + c_2$ (c_1, c_2 constants).

Lemma 6.3.4 *Let $\Sigma = \{a_1, ..., a_r\}$ be a finite alphabet, and suppose that θ is a permutation of Σ such that $\theta^2 = 1$. For a word $u = a_{i_1} ..... a_{i_m}$, we define, in the natural way, $\overline{\theta}(u) = \theta(a_1) ..... \theta(a_m)$. Suppose further that $K \subseteq \Sigma^*$ where $K \in$ **NSPACE**$(\mathcal{F})$, for some class of functions satisfying Condition 6.3.3. Then defining S to be the set of rewrite rules*

$$S = \{a\theta(a) \to \lambda, \theta(a)a \to \lambda : a \in \Sigma\} \cup \{u \to \overline{\theta}(v) : uv \in K, |u| > |v|\},$$

we have that

$$L = \{\alpha \in \Sigma^* : \alpha \xrightarrow{*} \lambda\} \in \textbf{\textit{NSPACE}}(\mathcal{F}).$$

Proof Take some word $u \in \Sigma^*$. If $u \in L$ then some sequence of reductions leads to the empty word. Hence, we begin with u and our non-deterministic algorithm works as follows. At each stage, we choose some reduction to make - we either remove a substring of the form $a\theta(a)$ or $\theta(a)a$, or we perform the following:

- Choose some subword α of u,
- Choose some word β with $|\beta| < |\alpha|$,
- Verify that the word $\alpha\beta \in K$,
- Replace α by $\theta(\beta)$.

We continue performing reductions until we either reduce to the empty word, or there are no more possible reductions to be made. Note that this algorithm is clearly terminating since every reduction strictly reduces the length of the word and hence the number of possible steps is bounded by the length of the original word. Also note that there only a finite number of choices at any point, since there are a finite number of subwords of any word, and a finite number of possible words of length strictly less than a given word.

It is clear from the definition that if $u \in L$ then some sequence of choices leads to the empty word, and hence we accept u, and if $u \notin L$ then no such sequence of choices leads to the empty word and we reject u. Hence this algorithm certainly decides L. It remains to consider the space bounds of this algorithm.

Now, $K \in \mathbf{NSPACE}(\mathcal{F})$, and so K is decidable in non-deterministic space $f(n)$ for some $f(n) \in \mathcal{F}$. Since clearly $|\alpha\beta| < 2|n|$ (α has length at most n and $|\beta| < |\alpha|$) then verifying membership of K can be done in space at most $f(2n) \in \mathcal{F}$. Every possible rewrite strictly reduces the length of the word, and so it is clear that we can perform any rewrite in linear space. Choosing a subword and some shorter word can also obviously be done in linear space. Hence, since $\mathcal{F}$ satisfies Condition 6.3.3, and therefore contains no sublinear functions, it is immediate that L lies in $\mathbf{NSPACE}(\mathcal{F})$ as required. $\square$

This result concerns non-deterministic languages. In fact, with a condition on the rewrite rules, we can prove a similar result for deterministic languages, which will enable us to produce a stronger final result.

Corollary 6.3.5 *Suppose we have the situation in Lemma 6.3.4, but suppose that $K \in \boldsymbol{DSPACE}(\mathcal{F})$, and the set of rewrite rules S is confluent on L. Then $L \in \boldsymbol{DSPACE}(\mathcal{F})$.*

Proof The basis of the algorithm is as before, but we need to make it deterministic. The crucial point to notice is that since our rewriting system is confluent on L, and clearly terminates, we have a confluent strongly-normalising system on L and hence every word has the unique normal form λ. Thus if a word reduces to the empty word under some choice of sequences of rewrites, it does under every possible sequence of rewrites. The system need not be confluent on words not in L - the point being that a word not in L must rewrite to some normal form (since the algorithm is terminating) that cannot be the empty word (or it would lie in L), and since all words in L have the unique normal form λ, once we reach some non-empty normal form then we know that the word cannot lie in L.

Hence we can use some deterministic procedure to choose which rewrite to perform at each stage on a word u. Thus our algorithm works as follows. At each stage, we remove all substrings of the form $a\theta(a)$ or $\theta(a)a$, and then:

- Consider every possible subword α of u in turn,
- For each subword, consider every possible word β with $|\beta| < |\alpha|$,
- If we find α, β such that $\alpha\beta \in K$, replace α by $\theta(\beta)$ and start the algorithm again.

So this algorithm allows us to continue performing deterministic rewrites until we terminate either with the empty word (in which case $u \in L$), or with some non-empty word (in which case $u \notin L$), so it clearly recognises L. Also, exactly as above, it is clear that L lies in $\mathbf{DSPACE}(\mathcal{F})$. $\square$

The crux behind our construction will be small cancellation theory. Suppose, as usual, we have a free group F with generating set X, and we have R as a symmetrised subset of F. For u a word of F, we use the notation $u > cR$ to mean that there is some $r \in R$ with $r \equiv uv$ and $|u| > c|r|$.

The following well-known result is of fundamental importance.

Theorem 6.3.6 (Greendlinger) *Let F be a free group with generating set X, and R a symmetrised subset of F, and suppose $G = F/N$ where N is the normal closure of R in F. Let $u \in N$ be a non-trivial, cyclically reduced word, and suppose R satisfies $C'(\frac{1}{6})$. Then either*

- *$u \in R$,*

or some cyclic permutation of u contains one of the following:

- *two disjoint subwords, each $> \frac{5}{6}R$,*
- *three disjoint subwords, each $> \frac{2}{3}R$,*
- *four disjoint subwords, two $> \frac{2}{3}R$ and two $> \frac{1}{2}R$,*
- *five disjoint subwords, four $> \frac{1}{2}R$ and one $> \frac{2}{3}R$,*
- *six disjoint subwords, each $> \frac{1}{2}R$.*

Proof See [37]. □

This has the following corollary, convenient for our purposes here.

Corollary 6.3.7 *We take F, R and N as above, with R satisfying $C'(\frac{1}{6})$. Suppose $u \in N$. Then we can reduce u to λ with a finite sequence of steps of the form*

- *removing a substring of the form $x^{-1}x$ or xx^{-1},*
- *replacing a substring α by β^{-1} where $|\beta| < |\alpha|$ and $\alpha\beta \in R$.*

Proof Take $u \in N$. We can assume that u is cyclically reduced. Hence, from Greendlinger's Theorem, we either have $u \in R$ (in which case we replace u by λ as an application of the third rule and we are done), or u contains more than half of a relator in R (this follows from each of the cases in Greendlinger's Theorem, since there are at least two disjoint subwords in every case, then even without any cyclic permutation of the word, there must exist at least one suitable subword in our word), in which case we again apply the third rule. This reduced word still lies in N (since it still represents the same element of the group, namely the identity) and is strictly shorter than u, and hence proceeding inductively we must eventually terminate with λ.

It is clear that this algorithm need take only a finite number of steps since it is length-reducing. □

Now, suppose that we have an alphabet $X = \{x_1, \ldots.., x_k\}$ (in fact, we can take $k = 2$, but we consider general k for ease of construction) and suppose $L \subseteq X^*$, where $L \in \mathbf{DSPACE}(\mathcal{F})$, for some $\mathcal{F}$ satisfying Condition 6.3.3. Let $A = \{x_1, \ldots.., x_k, a_1, \ldots.., a_{12}\}$, and define $R' = \{a_1\alpha \ldots.. a_{12}\alpha : \alpha \in L\}$. Then we may define R to be the set of cyclically reduced words formed from R' by taking closure under inverses and cyclic permutations. Let $\Sigma = A \cup A^{-1}$. Then we may define a permutation $\theta : \Sigma \to \Sigma$ where θ sends each symbol to its corresponding inverse symbol. Obviously, $\theta^2 = 1$.

Lemma 6.3.8 *R is symmetrised and satisfies $C'(\frac{1}{6})$.*

Proof It is clear from the definition that R is symmetrised. Note that a maximal piece p of R is of the form $\beta a_i \gamma$ or $\gamma^{-1} a_i{}^{-1} \beta^{-1}$ where $1 \leq i \leq 12$, β is a suffix to a word α in L and γ a prefix to the same α. Hence for any piece p, we certainly have $|p| \leq 1 + 2|\alpha|$.

So, suppose a piece p occurs in a cyclic permutation of a word $u = a_1\alpha.....a_{12}\alpha$, or its inverse. Then we have

$$\begin{aligned} |p| &\leq 1 + 2|\alpha| \\ &= \frac{1}{6}(6 + 12|\alpha|) \\ &< \frac{1}{6}(12 + 12|\alpha|) \\ &= \frac{1}{6}(|u|). \end{aligned}$$

Hence any piece of R is of length strictly less than a sixth of any word of which it is a part, and hence R satisfies $C'(\frac{1}{6})$ as required. □

Now consider the group $G = \langle A : R \rangle$, and let W be the word problem of this group.

Lemma 6.3.9 *$W \in$ **DSPACE**$(\mathcal{F})$.*

Proof By Corollary 6.3.7, a word in W can be reduced to the empty word by a sequence of rewrites either of the form $xx^{-1} \to \lambda$, $x^{-1}x \to \lambda$, or $\alpha \to \beta^{-1}$ where $|\beta| < |\alpha|$ and $\alpha\beta \in R$. But these rewrites are precisely the rewrites in Lemma 6.3.4 and Corollary 6.3.5 (note that since R is symmetrised the rewrite rules specified there are sufficient) and hence W is equivalent to the language L defined therein. The only issue we need to resolve is the question of whether these rewrite rules are confluent on W. But this is a simple matter of induction. There are no rewrites to perform on the empty word, and so our base step is trivial. Take some word $u \in W$. Whichever rewrite we choose, we either remove a word equivalent to the identity, or replace a word by a shorter word representing the same element of the group. And hence our reduced word u' represents the same element of the group and therefore lies in W. By induction, u' has a unique normal form λ, and hence whichever sequence of choices of rewrites we make on u, we always terminate with λ and thus our rewrite system is confluent on W. Hence, by Corollary 6.3.5, we immediately deduce that $W \in \mathbf{DSPACE}(\mathcal{F})$ as required. $\Box$

Lemma 6.3.10 *If u is some word over A, then*

$$u \in L \Leftrightarrow a_1u.....a_{12}u \in W.$$

Proof Suppose firstly that $u \in L$. Then by the definition of R, we have $a_1u.....a_{12}u \in R \subseteq W$. Conversely, suppose $v = a_1u.....a_{12}u \in W$. From Corollary 6.3.7, we note again that we can reduce any word in W to the empty word by a sequence of moves whereby we either remove trivial substrings like xx^{-1} and $x^{-1}x$ to freely reduce v, or we replace a substring α by β^{-1}, where $|\beta| < |\alpha|$ and $\alpha\beta \in R$. The important point to notice is that when we perform a replacement like this, $|\alpha| > \frac{1}{2}|\alpha\beta|$. And hence, if v is reduced, then it must contain at least half of some element of R, which immediately forces $u \in L$. $\Box$

Corollary 6.3.11 *Suppose there is an algorithm to decide membership of W in time/space bound $g(n)$. Then there is an algorithm to decide membership of L in time/space bound $g(12n + 12)$.*

Proof This follows immediately from the previous result, since to check whether or not a word u of length n lies in L, we simply test the word $a_1u....a_{12}u$ for membership of W, and this word is of length $12n + 12$. $\Box$

6.3.4 The main theorem

Having set up these preliminary lemmas, we can now proceed towards the main result. We wish to produce a suitable language L.

We begin with a simple observation.

Lemma 6.3.12 *Suppose $s(n)$ and $t(n)$ are proper complexity functions and $t(n)$ satisfies the condition that* $\inf_{n\to\infty}\frac{t(n)}{n} = \infty$. *Suppose that there exists a language L, decidable on a deterministic multitape Turing machine M_1 with k worktapes, operating in space $s(n)$ and time $t(n)$. Then for any constants c_1 and c_2 (with $c_1, c_2 > 0$), there exists a deterministic multitape Turing machine M_2 with k worktapes to decide L, working in space $c_1 s(n)$ and time $c_2 t(n)$.*

Proof (Sketch). This is essentially the standard tape compression theorem, as given in for example [31], where we encode m symbols of M_1 into one. We merely need to note that this technique compresses both the time and space required, and hence choosing m sufficiently large enough will allow us to satisfy both bounds. □

Hence, in particular, if we have a Turing machine that decides a language in space $O(n)$ and time $O(n^2)$, then there exists a Turing machine that decides the same language in space n and time n^2. With this in mind, let us define a language L to be the set of words $\{M; x\}$, where M is the standard description of a deterministic Turing machine, with at most three worktapes, accepting x in space n and time n^2.

Lemma 6.3.13 *L is a deterministic context-sensitive language.*

Proof We will demonstrate a deterministic Turing machine with linear space bound to decide L. Suppose we are given an input $M; x$ where M is written in the standard description. We firstly of course check that M is indeed the description of a Turing machine with at most three worktapes. Assuming this, the idea is that we wish to add an extra tape to M which effectively acts as an 'alarm clock'. Given input x of length n, this tape starts by writing n^2 in binary on this extra tape, and decrements this value by one every time a move is performed on the remaining worktapes. The idea of this tape is that if it reaches 0 without us having accepted x, then we have used n^2 steps in M and hence it cannot possibly accept x in the required bounds. The language accepted by this Turing machine is clearly the same as the language accepted by M. Also note that the length of this tape never exceeds $2\log n$, which is still a linear bound.

So we will construct a Turing machine with five worktapes to decide L. This machine begins by reading in $M;x$ and then writing on its first worktape the standard description of the Turing machine M' obtained by adding the extra 'alarm clock' tape to M. Suppose M has an alphabet A of size $|A|$, which we assume does not contain the symbols 0 and 1 used for the binary arithmetic. Then the alphabet of M' has size $|A|+2$, which when written in binary is clearly of length no greater than $\log|A|+1 \leq 2\log|A|$. This is a bound on the length of the encoding of each symbol in M'. It is well known that binary arithmetic can be performed using a fixed, finite number of states, say B, and hence the number of states increases only by B, and so the length of the encoding of a state certainly increases by no more than a factor of B.

Each description of a transition in M' consists of a transition in M with an extra symbol added to represent the symbol on the extra worktape (and so could at most double the number of symbols read), and so, since the length of the encoding of the symbols is at most doubled, and the length of the encoding of the states is at most multiplied by B, then the length of a transition is increased by at most $4B$. Thus, given a Turing machine with standard description of length n_1, we can write the standard description of the Turing machine obtained by adding an alarm clock tape in length no greater than $4Bn_1 + Zn_1$ where Z is the length of the usual encoding of binary arithmetic (the factor of n_1 comes from a very crude upper bound on the length of a symbol or state). Clearly this is a linear bound in n_1, and hence if we have an input $M; x$ of length n, we certainly can write this new Turing machine in space linear in n.

Having got this far our algorithm is simple to describe. We write this description of our new Turing machine M' on the first tape. It is then a simple matter to simulate the running of M' on x on the four remaining worktapes (clearly M' has at most four worktapes). If this simulation rejects x, or exceeds space $|x|$, or the alarm clock runs out, then we reject $M; x$, else otherwise we accept it.

It is clear that this Turing machine decides L. Finally note that the description of M' is linear in n as we have shown, and the simulation of M' uses space no greater than $|x| < n$, and so this is certainly a context-sensitive procedure. □

Lemma 6.3.14 *There is no deterministic Turing machine with at most three worktapes to decide L in space $O(n)$ and time $O(n^2)$.*

Proof Suppose, on the contrary, that there exists such a machine. From Lemma 6.3.12, it follows that there exists a deterministic Turing machine M_L deciding L, working in space $\frac{n}{3}$ and time $\frac{n^2}{9}$, with at most three worktapes.

Let us define a diagonalising machine D as follows. D takes as input a string M purporting to represent a Turing machine, and accepts M if M_L rejects $M; M$, and rejects M if M_L accepts $M; M$. Suppose we have an input M of length n. Then D simply simulates the running of M_L on $M; M$, which is of length no greater than $3n$ (we assume that M is non-empty as we can immediately reject an empty input). Hence D runs in space no greater than $\frac{3n}{3} = n$ and time no greater than $\frac{(3n)^2}{9} = n^2$, and also has no more than three worktapes.

Consider the computation of D given input D. D clearly terminates on all inputs. Suppose D accepts D. Then, by definition, M_L rejects $D; D$ and hence $D; D \notin L$. Hence, by the definition of L, and since D has at most three worktapes, D does not accept itself in space n and time n^2. But D certainly runs within these bounds and always terminates, and hence since it does not accept itself, it must reject itself, that is D rejects D and we have a contradiction.

Similarly, suppose that D rejects D. Then we have that $D; D \in L$. Thus, by definition, D accepts D and hence cannot ever reject D, and again we have a contradiction.

Hence our original assumption that M_L exists must have been false, and hence there is no deterministic machine with at most three worktapes to decide L in space $O(n)$ and time $O(n^2)$, as required. □

Putting these two results together we have the following immediate corollary.

Corollary 6.3.15 *There exists a deterministic context-sensitive language L which cannot be decided on a deterministic Turing machine with at most three worktapes, operating in space $O(n)$ and time $O(n^2)$.*

Proof The language L above satisfies precisely these conditions. □

Finally we can put everything together to produce our result. We take $\mathcal{F}$ to be the class of linear functions $f(n) = c_1 n + c_2$, so that **DSPACE**$(\mathcal{F})$ is simply **DSPACE**(n), the complexity class containing the deterministic context-sensitive languages.

Theorem 6.3.16 *There exists a finitely-generated group G with deterministic context-sensitive word problem which is not a subgroup of an automatic group.*

Proof We simply take our language L of Corollary 6.3.15 (which is a deterministic context-sensitive language and therefore lies in $\mathbf{DSPACE}(n)$ by definition) and form the group $G = \langle A : R \rangle$ as above. By Lemma 6.3.9 the word problem W of G lies in $\mathbf{DSPACE}(n)$. Suppose that there exists an algorithm to decide W in space $O(n)$ and time $O(n^2)$. Then by Corollary 6.3.11 we have L decidable by an algorithm operating in space $O(12n+12) = O(n)$ and time $O((12n+12)^2) = O(n^2)$, contradicting the definition of L. Hence G has deterministic context-sensitive word problem, but does not have word problem solvable simultaneously in time $O(n^2)$ and space $O(n)$ on a deterministic Turing machine with three worktapes, and so cannot be a subgroup of an automatic group by Corollary 6.3.1. $\square$

We have shown that there exists a group with (deterministic) context-sensitive word problem which is not a subgroup of an automatic group. The group constructed is infinitely presented, since we have an infinite set of relations. This leads to the obvious question of whether we can produce a finitely presented group with context-sensitive word problem which is not a subgroup of an automatic group, and we conjecture the following (which we note is certainly true if Conjecture 5.2.2 is true).

Conjecture 6.3.17 *There exists a finitely presented group with (deterministic) context-sensitive word problem which is not a subgroup of an automatic group.*

6.4 Further issues

Let us finally consider a couple of other issues related to this result and the technique we have used.

6.4.1 Real-time and context-sensitive word problems

A *real-time* Turing machine is one which reads in a word from left to right, and makes only a bounded number of steps for each input symbol read. Groups with real-time word problem are the subject of much current investigation, for example it is known that word-hyperbolic groups have real-time word problem, see [28]. Clearly, any group with real-time word problem has linear time word problem, and hence must be context-sensitive.

The converse of this question was posed by Claas Roever (personal communication), that is, is it possible to have a group with context-sensitive word problem that does not have real-time word problem? The construction we have allows us to answer this question.

Theorem 6.4.1 *There is a group with deterministic context-sensitive word problem, which does not have real-time word problem.*

Proof By the results of [51], there exists a deterministic context-sensitive language which is not real-time. Embed this into a group exactly as in the proof of Theorem 6.3.16. □

6.4.2 Generalising the technique

The technique we used in the proof of this main result could be generalised somewhat. We were particularly interested in groups which have a context-sensitive word problem but not a quadratic-time word problem. We can easily extend this to a much more general result regarding word problems of groups. Let us take any proper class of complexity functions $\mathcal{F}$ satisfying Condition 6.3.3. First of all note that our procedure was entirely deterministic (we used Corollary 6.3.5 to show this).

Now, suppose $\mathcal{G}$ is any class of proper complexity functions, where for some $g(n) \in \mathcal{G}$, $g(n) = O(2^{f(n)})$ for every $f(n) \in \mathcal{F}$. This bound is necessary so that we can apply the 'alarm clock' technique of Lemma 6.3.13, where we need to write $g(n)$ in binary.

Then it is clear that the procedure we followed to prove the main theorem, will also work with these complexity bounds, giving us the following result.

Theorem 6.4.2 *Suppose that $\mathcal{F}$ is a class of proper complexity functions satisfying Condition 6.3.3, and that $\mathcal{G}$ is a class of proper complexity functions, where for some $g(n) \in \mathcal{G}$, $g(n) = O(2^{f(n)})$ for every $f(n) \in \mathcal{F}$. Then there exists a group with word problem solvable in **DSPACE**($\mathcal{F}$), but not solvable in **DTIME**($\mathcal{G}$) on any given number of worktapes.*

Proof Similarly to the proof of Theorem 6.3.16. □

Note that this includes classes $\mathcal{G}$ with $\mathcal{G}$ 'smaller' than $\mathcal{F}$, this is obvious when one considers that the functions in $\mathcal{F}$ are merely an upper bound.

As a specific instance of Theorem 6.4.2, this provides us with a series of groups with deterministic context-sensitive word problem, but with word problem unsolvable in deterministic time $O(n^k)$, any any given number of tapes, for any given k.

We can also extend the result in another way. Suppose L is a language decidable in **DSPACE**($\mathcal{F}$), but not decidable in **DSPACE**($\mathcal{G}$), where $\mathcal{F}$ satisfies Condition 6.3.3. We can follow the proof of Theorem 6.3.16 to construct a group with word problem solvable in **DSPACE**($\mathcal{F}$). Then, from

Lemma 6.3.11, this group cannot have word problem lying in $\mathbf{DSPACE}(\mathcal{G})$, otherwise L would be decidable in this space bound too. Hence we have the following.

Theorem 6.4.3 *Suppose there exists a language decidable in* $\boldsymbol{DSPACE}(\mathcal{F})$*, but not decidable in* $\boldsymbol{DSPACE}(\mathcal{G})$*, where F satisfies Condition 6.3.3. Then there exists a group with word problem contained in* $\boldsymbol{DSPACE}(\mathcal{F})$*, but not decidable in* $\boldsymbol{DSPACE}(\mathcal{G})$*, on this fixed number of worktapes.*

Proof Construct a group exactly as in the proof of Theorem 6.3.16. □

Since there is a 'hierarchy' of space complexity classes, we have a similar 'hierarchy' of groups. In particular, it is well-known that $\mathbf{DSPACE}(n^2)$ strictly contains $\mathbf{DSPACE}(n)$, and hence there exists a language decidable in deterministic quadratic space, but not in deterministic linear space. This allows us to deduce the following obvious corollary.

Corollary 6.4.4 *There exists a group with solvable word problem which does not have deterministic context-sensitive word problem.*

Proof Take any language L contained in $\mathbf{DSPACE}(n^2)$, but not contained in $\mathbf{DSPACE}(n)$, and construct a group exactly as in the proof of Theorem 6.3.16. □

Of course, we can combine these two 'extensions' together to produce a whole series of groups with word problem solvable in some space bound, but not solvable in some other time/space bound, which illustrates the power of our technique.

Chapter 7

The reduced and irreducible word problem

In this thesis, we have considered the word and conjugacy problems in detail, and we have also touched upon the generalised word problem. Of course, there are many decision problems that it is possible to consider in a group. As an example, let us look at the reduced and irreducible word problems.

7.1 Reduced and irreducible word problems

The reduced and irreducible word problems were introduced in [24], and discussed at length in [48].

Suppose $G = \langle X \rangle$, and let $W_X(G)$, as usual, denote the word problem of G with respect to X. Then the *reduced word problem* $R_X(G)$ is defined to be the set of non-empty words α in $W_X(G)$ such that no proper prefix of α lies in $W_X(G)$. Similarly, we define the *irreducible word problem* $I_X(G)$ to be the set of non-empty words α in $W_X(G)$ such that no proper subword of α lies in $W_X(G)$.

We need the idea of insertion closure as discussed in [32]. For words α and β we let

$$\alpha \leftarrow \beta = \{u\beta v : \alpha = uv\},$$

denote the set of possible 'insertions' of β into α. For languages L_1 and L_2 we define $L_1 \leftarrow L_2$ in the obvious fashion, namely

$$L_1 \leftarrow L_2 = \{\alpha \leftarrow \beta : \alpha \in L_1, \beta \in L_2\}.$$

Then for a language L we may naturally define

$$L_1 = L, L_2 = L \leftarrow L, L_3 = (L \leftarrow L) \leftarrow L, ...$$

and define the *insertion closure* of L to be

$$I(L) = L_1 \cup L_2 \cup L_3 \cup .....$$

We have the following result, which was proved in [48], but we give for completeness.

Lemma 7.1.1 *Let $G = \langle X \rangle$. Then,*

(a) $W_X(G) = R_X(G)^*$ where L^* represents the Kleene star of L,

(b) $W_X(G) = I(I_X(G))$ where $I(L)$ represents the insertion closure of L.

Proof It is clear that any word in $R_X(G)^*$ represents the identity since it is merely a concatenation of words representing the identity, and hence we have $R_X(G)^* \subseteq W_X(G)$. In the opposite direction, we proceed by induction. Any word of length 1 in $W_X(G)$ must lie in $R_X(G)$ since it has no proper prefixes, and hence we have our initial step. Let α be a word in $W_X(G)$ of length n. If α has no proper prefix representing the identity then it lies in $R_X(G) \subseteq R_X(G)^*$, and we are done. Otherwise it has some proper prefix representing the identity, which we denote by β. Then we can write $\alpha = \beta\gamma$

where both β and γ represent the identity (since $\gamma = \beta^{-1}\alpha$). Both β and γ have length shorter than n and hence by induction lie in $R_X(G)^*$, and hence so does α as their concatenation. Therefore $W_X(G) \subseteq R_X(G)^*$ and so we have $W_X(G) = R_X(G)^*$ as required.

The second part is similar. It is clear that $I(I_X(G)) \subseteq W_X(G)$ since all we do is insert elements representing the identity into words representing the identity, so we clearly always remain with a word representing the identity. In the other direction, again we proceed by induction, and the initial case is again trivial. Given some word α of length n in $W_X(G)$ then either it does not contain a proper subword equivalent to the identity, in which case it lies in $I(I_X(G))$ immediately, or it does contain a proper subword γ such that $\alpha = \beta\gamma\delta$ and $\gamma \in W_X(G)$. Then the word $\beta\delta$ is still in $W_X(G)$, since all we have done is remove a substring representing the identity from α, and has length less than n. Hence, by induction, $\beta\delta \in I(I_X(G))$ and hence $\alpha \in I(I_X(G))$ as it is formed as the insertion of $\gamma \in W_X(G)$ into a string in $I(I_X(G))$. Thus $W_X(G) \subseteq I(I_X(G))$ and hence we have $W_X(G) = I(I_X(G))$ as required. $\Box$

We then have the following lemma.

Lemma 7.1.2 *Let G be a group generated by a finite set X. Then the following are equivalent:*

(a) G has (deterministic) context-sensitive word problem with respect to X,

(b) G has (deterministic) context-sensitive reduced word problem with respect to X,

(c) G has (deterministic) context-sensitive irreducible word problem with respect to X.

Proof Suppose G has (deterministic) context-sensitive word problem. Let u be some word over X. Let us consider the reduced word problem. Firstly, of course, we test that u does indeed represent the identity, using the (deterministic) context-sensitive algorithm for the word problem for G. If it does not, then it certainly cannot lie in the reduced word problem and we reject it immediately. If it does, then we then simply test all possible proper prefixes of u. If any of them are equivalent to the identity, then we reject u. Otherwise, if we have tested all possible proper prefixes without finding one equivalent to the identity, then we accept u. The algorithm for the irreducible word problem is entirely similar, but instead of testing every proper prefix we test every possible subword.

These two algorithms are clearly (deterministic) context-sensitive procedures and so we have shown (a) $\Rightarrow$ (b) and (a) $\Rightarrow$ (c).

Conversely, suppose G has (deterministic) context-sensitive reduced word problem, and we are given a word u over X. From Lemma 7.1.1, we have that $W_X(G) = R_X(G)^*$. So if u represents the identity, then some prefix of u (including possibly u itself) must lie in $R_X(G)$. So we simply test each prefix, in turn, with our (deterministic) context-sensitive algorithm for the reduced word problem to see if they lie in $R_X(G)$. If we find no prefix in $R_X(G)$, then we reject the word. If we find a prefix in $R_X(G)$, then we delete this prefix and start our algorithm again on this shorter word. We must eventually terminate either with rejection, or by reducing u to the empty word, in which case we accept u.

The algorithm for the word problem, when we know we have a (deterministic) context-sensitive irreducible word problem is similar, noting that $W_X(G)$ is the insertion closure of the irreducible word problem $I_X(G)$. We simply search through all possible subwords and delete a subword if we find one in $I_X(G)$, and continue our algorithm on this shorter word. If we cannot find one then it cannot lie in the insertion closure of $I_X(G)$ and we must reject the word, and we accept the word if we eventually terminate with the empty word.

Again these algorithms are clearly (deterministic) context-sensitive and thus this proves (b) $\Rightarrow$ (a) and (c) $\Rightarrow$ (a), and shows the equivalence of the three conditions. $\square$

In the above, we were careful to specify a particular generating set, since we needed this to be able to use the results of Lemma 7.1.1. However, this result allows us to deduce that the property of having a (deterministic) context-sensitive reduced or irreducible word problem is independent of the choice of generating set.

Corollary 7.1.3 *Suppose $G = \langle X \rangle = \langle Y \rangle$. Then,*

- *if G has (deterministic) context-sensitive reduced word problem with respect to X, then G also has (deterministic) context-sensitive reduced word problem with respect to Y.*
- *if G has (deterministic) context-sensitive irreducible word problem with respect to X, then G also has (deterministic) context-sensitive irreducible word problem with respect to Y.*

Proof Suppose G has (deterministic) context-sensitive reduced word problem with respect to X. Then, by Lemma 7.1.2, G has (deterministic) context-sensitive word problem with respect to X, and thus, by Lemma 2.2.2, G has (deterministic) context-sensitive word problem with respect to Y. Hence, again using Lemma 7.1.2, G has (deterministic) context-sensitive reduced word problem with respect to Y.

The situation with regard to the irreducible word problem is proved similarly. □

Hence we are able to talk about the reduced word problem, or irreducible word problem, being (deterministic) context-sensitive, without worrying about the generating set being taken. Note, however, that this proof does require Lemma 7.1.2, which requires linear space, and hence we cannot deduce a similar result for arbitrary languages closed under inverse homomorphism.

7.2 Other decision problems

We have considered, in the above, the reduced and irreducible word problems as examples of problems that we had not considered as the 'main' problems that occupy the majority of the thesis, and we were able to show some interesting results. We will now make some very brief comments about some other well-known decision problems, which if nothing else, perhaps show why we have focussed on the problems that we have!

The *power problem* for a group presentation is the problem of determining, for two words u and v, whether or not u is a power of v (including the 0'th power, where $v^0 = 1$ for all v). However, this is really just another formulation of the generalised word problem for cyclic subgroups, and so this problem is equivalent to one of the problems which we considered as our 'main' problems.

There are also other sorts of decision problems. A closely related problem to the power problem is the *order problem*. Given a presentation of a group G, this is the problem of computing $\mathrm{ord}(w)$, where for a given word w we define $\mathrm{ord}(w)$ to be the order of w if w has finite order, or 0 otherwise.

This is a different type of decision problem to those that we have previously considered, since we are actually trying to compute the value of a function rather than a straightforward yes/no algorithm. Because of this, it is more difficult to have a context-sensitive algorithm, since unless our answer is linearly bounded in terms of the original word, of course it cannot be a context-sensitive procedure (somewhat similarly to our idea of an effective generalised word problem). For many problems like this, having to impose strong conditions to ensure such a bound exists, makes the value of detailed study rather prohibitive.

One could also ask questions about presentations of groups, not just the groups themselves. For example, one could consider the question of whether a given presentation represents the trivial group, or the *isomorphism problem*, the question of whether two presentations represent the same group. However, it seems unlikely that these sort of questions are really of any interest with regard to context-sensitive algorithms.

There are a huge number of questions that could be asked about a group and, of course, many of these would be potentially interesting with regard to context-sensitive algorithms. However, the number of possible questions is virtually limitless, and it is quite clear that the decision problems we have considered in this thesis seem to be the most important ones to ask.

Chapter 8

Groups with context-sensitive decision problems

As a final issue, it seems sensible to give a collection of classes of groups that have (deterministic) context-sensitive decision problems. From the results obtained earlier in the thesis, we can then combine these groups together, or extend them, in a variety of ways to create further groups with (deterministic) context-sensitive decision problems.

8.1 Some groups with context-sensitive decision problems

In this section we are interested in producing a comprehensive list of appropriate groups, and make no special effort to precisely define the groups given. Readers interested in a particular group should consult the appropriate reference. Note, also, that there exist overlaps between the classes of groups discussed, but again we are interested in compiling an extensive list of groups to be of interest to a wide range of mathematical interests and we feel that it is worthwhile incorporating groups if they are of special interest even if they are a subset of (or closely connected to) another set of groups in our list. We concentrate mainly on the word problem.

Firstly, we make the observation that we showed in Theorem 4.2.1 that if H is a subgroup of finite index of a group G, then G has (deterministic) context-sensitive word problem if and only if H has (deterministic) context-sensitive word problem. Given a group-theoretic property P, a group G is said to be *virtually* P if it has a subgroup of finite index with property P. So, for example, G is a virtually abelian group if it has an abelian subgroup H, of finite index in G.

Given this, for any of our groups given below defined by a particular property P (for example the free groups) we can also deduce that the virtually P groups (for example the virtually free groups) also have (deterministic) context-sensitive word problem[1].

Therefore, to avoid tedious repetition in our list of groups, we state here that this property holds and omit the word 'virtually' in our lists.

In addition, we know from Lemma 2.2.6 that if G has (deterministic) context-sensitive word problem, then so do all of its subgroups. Therefore, we can also add to our collection, any subgroups of the listed groups. Hence, again, to avoid tedious repetition, we omit the words 'subgroups of' from our list.

8.1.1 Linear groups

A group is said to be *linear* if it is isomorphic to a group of matrices over some field. In [35] the following crucial result is proved.

[1] Of course, this does not hold for the conjugacy problem

Theorem 8.1.1 *The word problem for a finitely generated linear group over a field of characteristic zero is solvable in logspace.*

This result was extended in [55] to groups of matrices over an arbitrary field.

Since, obviously, the class of groups with deterministic context-sensitive word problem contains the class of groups with word problem solvable in logspace then any group which we can exhibit as linear immediately has deterministic context-sensitive word problem.

Some examples of classes of groups known to be linear include :

- Matrix groups (obviously!),
- Free groups,
- Polycyclic groups,
- Metabelian groups,
- Abelian groups such that the maximal periodic normal subgroup has finite rank,
- Torsion-free nilpotent groups,
- Many groups from Lie algebra theory,
- Many automorphism groups, in particular the automorphism groups of certain abelian and soluble groups,
- The Baumslag-Solitar groups $B(1,q)$, $B(p,1)$, or $B(p,p)$.

References for the above results can be found in [5], [18], [35], [50], and [61]. In particular, [61] provides an excellent survey of some of the automorphism groups that arise.

It is worth making the comment that not all groups with word problem solvable in logspace are linear. A group G is said to be *residually finite* if, for every non-trivial element $g \in G$, there exists a homomorphism ϕ from G into a finite group K, such that $\phi(g) \neq 1$. It is known that linear groups are residually finite. In [60], the group

$$\langle a, b, c : b^{-1}ab = a^2, c^{-1}ac = a^2 \rangle$$

is shown to have word problem solvable in logspace.

However, it is also shown that this group is not residually finite, and therefore not linear. Hence this gives a non-linear group with word problem solvable in logspace. We also note for reference that this group gives us the following lemma.

Lemma 8.1.2 *There exists a group with deterministic context-sensitive word problem which is not residually finite.*

8.1.2 Automatic groups

In Theorem 6.1.3, we demonstrated that automatic groups have deterministic context-sensitive word problem. So let us give some examples of automatic groups as another list of groups with deterministic context-sensitive word problem.

- Finite groups,
- Biautomatic groups,
- Euclidean groups,
- Braid groups,

- Negatively curved groups,
- Geometrically finite groups,
- Hyperbolic groups,
- Coxeter groups,
- Lattices in the Lie groups $SO(n, 1)$,
- Artin groups associated with the finite Coxeter groups,
- Mapping class groups of finite (punctured) surfaces,
- The fundamental group of a geometrically finite hyperbolic group,
- Fundamental groups of compact manifolds with negative curvature,
- $\mathrm{Aut}(F_2)$,
- Amalgamated free products of finitely generated abelian groups,
- Amalgamated free products of negatively curved subgroups with a cyclic amalgamated subgroups,
- Amalgamated free products of finitely generated free groups with a finitely generated amalgamated subgroups,
- Certain small cancellation groups,
- Certain Fibonnacci groups,
- Fundamental groups of thick doodles,
- An extension of F_2 by $\mathbb{Z}$ which is not virtually a direct product,

- Torsion-free groups admitting a discrete cocompact action on a locally finite building of specified types.

For proofs that these classes of groups are automatic see [5], [6], [9], [11], [18], [19], and [20].

8.1.3 Combable groups

As another example, we discuss briefly the issue of combings in groups.

Let G be a group with generating set A, and suppose that L is a language over A such that the natural homorphism $\phi : L \to G$ is surjective, in which case we call L a *normal form* for G. As before, we define the length of an element $g \in G$ to be $l(g) = d(1, g)$ (recall our definition of distance of words via the Cayley graph).

Then we say that L is an *asynchronous combing* for G if there exists $K \in \mathbb{R}$ such that for all $u, v \in L$ with $d(\phi(u), \phi(v)) \leq 1$ we can find monotone reparametrisations of $[0, \infty)$

$$\psi_1 : t \mapsto t', \psi_2 : t \mapsto t''$$

such that for all t, we have that $d(u(t'), v(t'')) \leq K$. The informal idea is that the paths of u and v in some sense stay 'close' to each other. Let us also define a language L to be *short* if there exist μ, ϵ satisfying

$$w \in L \Rightarrow l(w) \leq \mu l(\phi(w)) + \epsilon.$$

Also, we define a function D to be a *departure function* for L if, for any word u, we have $l(\phi(v)) \geq n$ for $l(v) \geq D(n)$, where v is any subword of u.

The first result in this area was the following result, as proved in [54].

Theorem 8.1.3 *Let H be a finitely generated subgroup of a group G.*

- *If G has a short asynchronous combing, then the word problem of H is context-sensitive.*
- *If G has a short asynchronous combing with a departure function, and this combing is deterministic context-sensitive, then the word problem of H is deterministic context-sensitive.*

Since automatic groups satisfy these hypotheses, from Lemma 2.2.6 we have the following corollary which gives an alternative proof that subgroups of automatic groups have deterministic context-sensitive word problem.

Corollary 8.1.4 *Any finitely generated subgroup of an automatic group has deterministic context-sensitive word problem.*

An extension to these results can be obtained from the results of Gersten in the paper [21], where the linearly-bounded nature of the procedures used lead to the following, stronger, result.

Theorem 8.1.5 *Any combable group has context-sensitive word problem.*

Note that there is no requirement of determinism in this result.

8.1.4 Real-time algorithms

The concept of *real-time* word problems was discussed in [28] and developed further in [29]. An algorithm is *real-time* if we only move right on the input tape, and terminate when we reach the end of the word, and for each step on the input tape, we only make a bounded number of moves on each of our worktapes. An informal interpretation of this, is that a language is real-time if it is accepted by a computer reading in at a constant rate. Clearly, a real-time algorithm is context-sensitive - suppose the number of operations we can make for each step right on the input tape is bounded by m. Then, since we terminate upon reaching the end of our input word, the length of the tapes are bounded by mn for an input word of length n, and hence the procedure is context-sensitive.

In [28] and [29] the following groups are given as examples of groups with real-time, and hence context-sensitive, word problem.

- Nilpotent groups,
- Word hyperbolic groups,
- Geometrically finite hyperbolic groups.

8.1.5 Other groups

Let us make the obvious comment that any group with decision problem solvable in a subset of the context-sensitive languages has context-sensitive decision problem. This may seem obvious but it is worth bearing in mind. For example, we gave above a list of groups which are linear and hence have word problem solvable in logspace, and therefore have context-sensitive word problem. Similarly groups with regular or context-free word problem would have context-sensitive word problem - however these have already been classified as the finite and virtually free groups respectively.

As a specific example, we gave the example earlier courtesy of [60] of the group

$$\langle a, b, c : b^{-1}ab = a^2, c^{-1}ac = a^2 \rangle,$$

which is not linear but has word problem solvable in logspace and hence certainly has (deterministic) context-sensitive word problem.

We also mentioned, as a corollary to Theorem 2.3.2, that any group with word problem solvable in deterministic time $O(n \log n)$ has deterministic context-sensitive word problem. Clearly, there are many groups for which specific algorithms can be given, but we hope that the above list summarises some of the important classes of groups that are within our scope of interest.

8.1.6 The conjugacy problem

The situation with regard to the conjugacy problem being context-sensitive is much more difficult, and comparatively little is known about such groups. In many cases even where algorithms are known to solve the conjugacy problem, they are highly exponential. Of course, we made the comment earlier that an algorithm may be exponential in terms of time, but need not be in terms of space since we may require little storage space.

There are some simple examples of groups where the conjugacy problem is solvable in linear space. As we have noted in Lemma 2.5.6 and Lemma 2.5.4, finite groups and free groups have deterministic context-sensitive conjugacy problem. Also, those abelian groups with (deterministic) context-sensitive word problem are a trivial example, since if $u \sim v$ then there exists w such that $w^{-1}uw = v$, and hence $u = v$ (since the group is abelian and w cancels with w^{-1}). Hence, this reduces to the word problem.

We have given some examples in the thesis of specific groups with (deterministic) context-sensitive conjugacy problem, but actually categorising specific classes of groups appears to be a very difficult problem.

8.2 One-relator groups

One of the first classes of groups shown to have solvable word problem was the class of groups which, for obvious reasons, are known as *one-relator groups*. Let us now look at these groups in a little more detail.

8.2.1 Definition

We consider the one-relator groups, that is those groups G that may be given by a presentation $G = \langle X : R \rangle$ where R consists of a single relation $u = v$. The word problem for one-relator groups has been known for some time to be solvable (see [39]), although the complexity may be very high. However, the question of the solvability of the conjugacy problem remains open. Juhász claimed a proof in [33] but it is generally believed (even now by Juhász himself!) that this proof is not watertight, and to the mathematical community the question remains open. However, since many constructions involving one-relator groups are built up through small cancellation theory and HNN extensions, it seems that a significant number of these groups may be amenable to context-sensitive algorithms.

8.2.2 Certain one relator-groups

Let us show that with a fairly weak condition on the single relation in a one-relator group, then we have a deterministic context-sensitive procedure to solve the word problem. Suppose we can write a presentation for a group in the form

$$G = \langle a_1, ..., a_m, b_1, ...b_n : u_a = u_b \rangle$$

where u_a is a word in the a_i and u_b is a word in the b_i. All this condition really says, is that there is some presentation with one relator, where we may split the relator into two parts such that no symbols from one part is contained in the other. For example, a relator where some symbol x appears only once can obviously be decomposed like this, since we cyclically permute the relator until x appears at the start of the word, and then take $u_a = x^{-1}$, and u_b to be the remainder of the word.

Let us now show that any group of this form has deterministic context-sensitive word problem.

Lemma 8.2.1 *Any one-relator group G which has a presentation of the form $G = \langle a_1, ..., a_m, b_1, ...b_n : u_a = u_b \rangle$, where u_a is a word in the a_i and u_b is a word in the b_i, has deterministic context-sensitive word problem.*

Proof Suppose we are given a group G satisfying the hypotheses of the lemma. Let F_A and F_B be the free groups on the a_i and b_i respectively. Note that G is the amalgamated free product of F_A and F_B with respect to the cyclic subgroups $C_{u_A} = \langle u_A \rangle$ and $C_{u_B} = \langle u_B \rangle$. To determine the effective generalised word problem of F_A with respect to C_{u_A}, we can simply apply Lemma 2.5.7, and of course, similarly for the effective generalised word problem of F_B with respect to C_{u_B}.

And hence the result follows from Lemma 3.3.4 and Lemma 2.5.3. $\square$

We conjecture that a similar result holds for the conjugacy problem.

Conjecture 8.2.2 *Any one-relator group G which has a presentation of the form $G = \langle a_1, ..., a_m, b_1, ...b_n : u_a = u_b \rangle$, where u_a is a word in the a_i and u_b is a word in the b_i, has deterministic context-sensitive conjugacy problem.*

8.2.3 Surface groups

Let us now give an important class of groups satisfying the above condition - namely the class of surface groups, which we will hence show to have deterministic context-sensitive word problem. This class of groups occurs as a strong connection between the ideas of topology and surfaces, and combinatorial group theory, which we touched upon in the introduction. We shall not give any details here of the topological construction (for further information see [57]), but the important point is that there is a basic classification of all closed finite surfaces - either as an orientable surface (of some genus n), a non-orientable surface (of some genus n), or a sphere. We are interested in the fundamental groups of these surfaces, the so-called *surface groups*. We have the following presentations of the surface groups :

$$\text{orientable, genus } n : \langle a_1, b_1, ..., a_n, b_n : a_1 b_1 {a_1}^{-1} {b_1}^{-1} .... a_n b_n {a_n}^{-1} {b_n}^{-1} = 1 \rangle,$$

$$\text{non-orientable, genus } n : \langle a_1, ..., a_n : {a_1}^2 .... {a_n}^2 = 1 \rangle,$$

$$\text{sphere } : \{1\}.$$

Lemma 8.2.3 *All surface groups have deterministic context-sensitive word problem.*

Proof Any surface group is either isomorphic to the fundamental group of a sphere - in which case the group is trivial and the word problem is trivially solved - or it is a one-relator group satisying the conditions of Lemma 8.2.1. □

If Conjecture 8.2.2 is true, then all surface groups have deterministic context-sensitive conjugacy problem.

8.3 The relative difficulty of the word, conjugacy and generalised word problems

Finally, let us emphasise again the difficulty of the conjugacy problem and the generalised word problem.

We mentioned previously that the word problem is a specific case of the conjugacy problem (where one of the words is the identity) and generalised word problem (where the subgroup is the trivial subgroup). Thus, any group with deterministic context-sensitive conjugacy problem (or generalised word problem) also has deterministic context-sensitive word problem. Let us show via a series of examples how the converse is untrue and how the solvability of these problems can, in many cases, be independent of each other.

Lemma 8.3.1 *There exists a group G_1 with deterministic context-sensitive word problem, conjugacy problem, and generalised word problem.*

Proof The trivial group $\{1\}$ suffices. $\Box$

It is also possible for none of the problems to be decidable via context-sensitive algorithms.

Lemma 8.3.2 *There exists a group G_2 with unsolvable word, conjugacy, and generalised word problem.*

Proof It is well known that there are groups with unsolvable word problem (see for example p146 of [14] for a finitely-presented example), and since as we have noted, the word problem is a special case of the conjugacy problem and generalised word problem, then they must be unsolvable too. $\Box$

It is more interesting to consider groups where some of the problems are deterministic context-sensitive, and some are unsolvable.

Lemma 8.3.3 *There exists a group G_3 with deterministic context-sensitive word problem, and unsolvable generalised word problem and conjugacy problem.*

Proof From Theorem 5.2 of [43], the conjugacy and generalised word problem are unsolvable for some finitely generated subgroups of the matrix group

$SL(4, \mathbb{Z})$, since the direct product $F_n \times F_n$ of two free groups of rank n has a faithful representation in $SL(4, \mathbb{Z})$, and so the result follows from [41]. However, from Theorem 8.1.1, linear groups have deterministic context-sensitive word problem, and thus so do their subgroups from Lemma 2.2.6. $\square$

Lemma 8.3.4 *There exists a group G_4 with deterministic context-sensitive word problem and conjugacy problem, and unsolvable generalised word problem.*

Proof Let F be a free group on at least two generators, and form the group $G_4 = F \times F$. From [41], G_4 has a finitely generated subgroup L such that the generalised word problem for L in G_4 is unsolvable. However, F has deterministic context-sensitive word and conjugacy problem from Lemma 2.5.3 and Lemma 2.5.4, and thus so does G_4 from Lemma 3.2.1 and Lemma 3.2.2. $\square$

It is worth noting purely for interest's sake that L in the above lemma has unsolvable conjugacy problem (and is not finitely presented), we shall not give the proof here but my thanks go to Chuck Miller (private communication) for pointing this out.

We have not considered in this exposition, the existence of a group with deterministic context-sensitive generalised word problem (and hence word problem), but unsolvable conjugacy problem. As far as we are aware, it is still open whether there even exists a group with solvable generalised word problem, but unsolvable conjugacy problem. Hence we pose this final possibility merely as an open question.

Open Question 8.3.5 *Does there exist a group with solvable generalised word problem but unsolvable conjugacy problem? If so, does there exist a* *group with deterministic context-sensitive generalised word problem but unsolvable conjugacy problem?*

Chapter 9

Conclusions

The intention of this thesis has been to provide a coherent study of the issue of groups with context-sensitive decision problems. We have tried to keep in mind, throughout, the desire to increase understanding of these groups. Although the word problem is perhaps the most fundamental decision problem one could ask about a group, and hence makes a substantial contribution to this study, we have also endeavoured to provide a detailed insight into the conjugacy problem, and also consider other decision problems, such as the generalised word problem, and the reduced and irreducible word problems. In the course of the thesis, we have given several examples of groups with context-sensitive decision problems, and hope that the proof techniques employed for some of these will be useful for other groups. With this in mind, we have given conditions under which we can combine or extend these groups to form new groups which still have a context-sensitive decision problem.

We stressed almost from the start that the conjugacy problem is a much more difficult problem than the word problem. This is illustrated in our result that taking an extension of finite index of a group with context-sensitive word problem preserves the context-sensitive nature of the word problem, but, remarkably, if the group has context-sensitive conjugacy problem, this extension need not even preserve the solvability of the conjugacy problem, even if it is only of index 2.

Despite the fact that one might think that linear space would be a fairly prohibitive bound on calculations in groups, the class of groups that we have studied appears to be very wide. We have exhibited a large number of seemingly different 'types' of groups that have context-sensitive decision problems (in particular the word problem), including both specific groups, and general classes of groups such as the surface groups. We have also eliminated the possibility that the groups with context-sensitive word problem may all come under the banner of being subgroups of automatic groups, by illustrating a group with context-sensitive word problem which is not a subgroup of an automatic group. Given the wide nature of automatic groups (and hence even more so their subgroups!) this only serves to strengthen what a wide range of groups we are dealing with.

We have touched throughout the thesis on issues that remain to be resolved, and questions still to be answered, and clearly these are difficult questions. For example, we touched very briefly on the issue of the generalisation of these results to semigroups, when we considered embeddings. How much of the work here can be transferred to results about semigroups? We have not addressed this question in this thesis, since it requires a detailed study on its own. We can also generalise many of the results to other space complexity classes. We also have the question of determinism. All of our algorithms are deterministic, but it is still unknown whether or not deterministic linear space is equivalent to non-deterministic linear space, and even if this is not true, if it is true in the restricted case of word problems in groups (along the same lines as context-free and deterministic context-free word problems). We have also considered issues of finite presentability and effectiveness, amongst others, and we have indicated via open questions or conjectures in the text where we feel that there are important questions to be answered.

This all suggests that classifying such groups is not going to be an easy task, and there is still a great deal of work to be done. However, the first step towards any sort of classification, or for that matter any result at all, is to increase the understanding of the subject we are dealing with, and this is what we hope we have achieved in this thesis.

Bibliography

[1] S. I. Adian *The unsolvability of certain algorithmic problems in the theory of groups* Trudy Moscov. Mat. Obsc. 6 p231-298 (1957)

[2] A. V. Anisimov *Group languages* Kibernetika No.4 p18-24 (1971)

[3] M. Anshel and P. Stebe *The solvability of the conjugacy problem for certain HNN groups* Bull. Amer. Math. Soc. 80 p266-270 (1974)

[4] E. Artin *Theorie der Zöpfe* Abh. Math. Sem. Univ. Hamburg 4 p47-72 (1926)

[5] G. Baumslag *Some open problems* Summer school in group theory in Banff (Editor O. Kharlampovich) American Mathematical Society p1-9 (1996)

[6] G. Baumslag, S. M. Gersten, M. Shapiro and H. Short *Automatic groups and amalgams* J. Pure Appl. Algebra 76, No. 3, p229-316 (1991)

[7] J.-C. Birget *Time-complexity of the word problem for semigroups and the Higman embedding theorem* Internat. J. Algebra Comput. 8, No. 2, p235-294 (1998)

[8] J.-C. Birget, A. Yu. Ol'shanskii, E. Rips, M. V. Sapir *Isoperimetric functions of groups and the computational complexity of the word problem* (preprint)

[9] T. Brady *Complexes of nonpositive curvature for extensions of F_2 by $\mathbb{Z}$* Topology Appl. 63, No. 3, p267-275. (1995)

[10] C. M. Campbell, E. F. Robertson, N. Ruskuc and R. M. Thomas *Automatic semigroups* Theoret. Comp. Sci. 250, No. 1-2, p365-391 (2001)

[11] D. I. Cartwright and M. Shapiro *Hyperbolic buildings, affine buildings and automatic groups* Michigan Math. J. 42, No. 3, p511-523 (1995)

[12] D. J. Collins *Recursively enumerable degrees and the conjugacy problem* Acta. Math. 122 p115-60 (1969)

[13] D. J. Collins *The word, power and order problems in finitely presented groups* Decision problems and the Burnside problem in group theory (Editors W. W. Boone, F. B. Cannonito and R. C. Lyndon) North-Holland Publishing Company (Amsterdam-London) p401-420 (1973)

[14] D. J. Collins, R. I. Grigorchuk, P. F Kurchanov and H. Zieschang *Combinatorial Group Theory and Applications to Geometry* Springer-Verlag Berlin Heidelberg (1998)

[15] D. J. Collins and C. F. Miller III *The conjugacy problem and subgroups of finite index* Proc. London Math. Soc. (3) 34 , No. 3, p535-556 (1977)

[16] M. Dehn *Über die Topologie des dreidimensional Raumes* Math. Ann. 69 p137-168 (1910)

[17] M. J. Dunwoody *The accessibility of finitely presented groups* Inventiones Math. 81 p449-457 (1985)

[18] D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson and W. P. Thurston *Word processing in groups* Jones and Bartlett (1991)

[19] B. Farb *Automatic groups: a guided tour* Enseign. Math. (2) 38, No. 3-4, p291-313 (1991)

[20] S. M. Gersten and H. B. Short *Small cancellation theory and automatic groups* Invent. Math. 102, No. 2, p305-334 (1991)

[21] S. M. Gersten *Finiteness properties of asynchronously automatic groups* Geometric Group Theory (Editors R. Charney, M. Davis, M. Shapiro) Ohio State Univ. Math. Research Institute Publ, vol. 3, Walter de Gruyter, Berlin-New York, p121-133 (1995)

[22] K. Gödel *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I* Monatsh. f. Math. u. Phys. 38 p173-198 (1931)

[23] A. V. Goryaga and A. S. Kirkinskii *The decidability of the conjugacy problem does not carry over to finite extensions of groups* Algebra i Logika 14, No. 4, p393-406 (Russian) (1975). English translation in Algebra and Logic 14 (1975), No. 4, p240-248 (1976)

[24] R. H. Haring-Smith *Groups and simple languages* Trans. Amer. Math. Soc. 279 p337-356 (1983)

[25] T. Herbst and R. M. Thomas *Group presentations, formal languages and characterizations of one-counter groups* Theoret. Comp. Sci. 112 p187-213 (1993)

[26] G. Higman *Subgroups of finitely presented groups* Proc. Roy. Soc. Ser. A 262 p455-475 (1961)

[27] G. Higman, B. H. Neumann and H. Neumann *Embedding theorems for groups* Jour. London Math. Soc. No.24 p246-254 (1949)

[28] D. F. Holt *Word-hyperbolic groups have real-time word problem* Internat. J. Algebra Comput. Vol. 10, No. 2 p221-228 (2000)

[29] D. F. Holt and S. Rees *Solving the word problem in real-time* J. London. Math. Soc. (2) 63, No. 3, p623-639 (2001)

[30] J. E. Hopcroft, W. J. Paul and L. G. Valiant *On time versus space* J. Assoc. Comput. Mach. 24, No. 2, p332-337 (1977)

[31] J. E. Hopcroft and J. D. Ullman *Introduction to automata theory, languages and computation* Addison-Wesley (1979)

[32] M. Ito, L. Kari and G. Thierrin *Insertion and deletion closure of languages* Theoret. Comp. Sci. 183, No. 1, p3-19 (1997)

[33] A. Juhász *Solution of the conjugacy problem in one-relator groups* Algorithms and classification in combinatorial group theory (Berkeley, CA, 1989), Math. Sci. Res. Inst. Publ., 23, Springer, New York, p69-81 (1992)

[34] D. Kelley *Automata and formal languages - an introduction* Prentice-Hall (1995)

[35] R. J. Lipton and Y. Zalcstein *Word problems solvable in logspace* J. Assoc. Comput. Mach. 24 p522-526(1977)

[36] J. M. Lockhart *An HNN-extension with cyclic associated subgroups and with unsolvable conjugacy problem* Trans. Amer. Math. Soc 313, No.1 p331-345 (1989)

[37] R. C. Lyndon and P. E. Schupp *Combinatorial group theory* Springer Berlin (1977)

[38] W. Magnus *Das Identitätsproblem für Gruppen mit einer definierenden Relation (der Freiheitsatz* Math. Ann. 106 p295-307 (1932)

[39] W. Magnus, A. Karrass and D. Solitar *Combinatorial group theory* Dover Publishing New York (1976)

[40] A. A. Markov *Insolubility of the problem of homeomorphy* Proc. Internat. Congr. Math p200-306 (1958)

[41] K. A. Mihailova *The occurrence problem for direct products of groups* Dokl. Akad. Nauk SSSR 119 p1103-1105 (1958)

[42] C. F. Miller III *On group-theoretic decision problems and their classification* Princeton University Press (1971)

[43] C. F. Miller III *Decision Problems In Groups - Surveys and Reflections* Algorithms and classification in combinatorial group theory (Berkeley, CA, 1989), p 1-59, Math. Sci. Res. Inst. Publ., 23, Springer, New York (1992)

[44] D. E. Muller and P. E. Schupp *Groups, the theory of ends, and context-free languages* J. Comp. System Sci. 26 p295-310 (1983)

[45] D. E. Muller and P. E. Schupp *The theory of ends, pushdown automata and second-order logic* J. Comp. System Sci. 37 p51-75 (1985)

[46] P. S. Novikov *On the algorithmic unsolvability of the word problem in group theory* Trudy Mat. Inst. Steklov 44, p143-286 (1955)

[47] C. H. Papadimitriou *Computational Complexity* Addison-Wesley (1995)

[48] D. W. Parkes and R. M. Thomas *Reduced and irreducible word problems of groups* University of Leicester Technical Report No. 1999/4 (1999)

[49] M. O. Rabin *Recursive unsolvability of group theoretic problems* Ann. of Math. (2) No. 67 p 172-194 (1958)

[50] V. N. Remeslennikov *Representation of finitely generated metabelian groups by matrices* Algebra i Logika 8 p72-75 (1969)

[51] A. L. Rosenberg *Real-time definable languages*, Jour. Assoc. Comp. Mach. Vol 14, No. 4 p645-662 (1967)

[52] J. J. Rotman *An introduction to the theory of groups*, Fourth Edition, Springer-Verlag (New York) (1995)

[53] M. V. Sapir, J.-C. Birget, E. Rips *Isoperimetric and isodiametric functions of groups* (preprint)

[54] M. Shapiro *A note on context-sensitive languages and word problems* Internat. J. Algebra Comput. Vol.4, No.4 p493-497 (1994)

[55] H.-U. Simon *Word problems for groups and context-free recognition* Fundamentals of computation theory (Proc. Conf. Algebraic, Arith. and Categorical Methods in Comput. Theory, Berlin/Wendisch-Rietz) p417-422 (1979)

[56] J. C. Stillwell *The word problem and the isomorphism problem for groups* Bull. Amer. Math. Soc. Vol 6, No. 1, p33-56 (1982)

[57] J. C. Stillwell *Classical topology and combinatorial group theory*, Second Edition, Springer-Verlag (New York) (1993)

[58] H. Tietze *Über die topologischen Invarienten mehrdimensionaler Mannigfaltigkeiten* Monatsh. f. Math. u. Phys. 19 p1-188 (1908)

[59] A. M. Turing *On computable numbers, with an application to the Entscheidungsproblem* Proc. London Math. Soc. (2) No. 42 p230-265 (1936)

[60] S. Waack *Tape complexity of word problems* Fundamentals of computation theory (Szeged, 1981), Lecture Notes in Comput. Sci., 117, Springer Berlin-New York, p467-471 (1981)

[61] B. A. F. Wehrfritz *Infinite linear groups - an account of the group-theoretic properties of infinite groups of matrices* Springer-Verlag, New York-Heidelberg (1973)