Computational aspects of knots and knot transformations

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy

by

Rafiq Asad Muthanna Saleh

August, 2011

Abstract

In this thesis we study the computational aspects of knots and knot transformations. Most of the problems of recognising knot properties (such as planarity, unknottedness, equivalence) are known to be decidable, however for many problems their precise time or space complexity is still unknown. On the other hand, their complexity in terms of computational power of devices needed to recognise the knot properties was not studied yet. In this thesis we address this problem and provide first known bounds for some knot problems within this context. In order to estimate and characterise complexity of knot problems represented by Gauss words, we consider various tools and mathematical models including automata models over infinite alphabets, standard computational models and definability in logic.

In particular we show that the planarity problem of signed and unsigned Gauss words can be recognised by a two-way deterministic register automata. Then we translate this result in terms of classical computational models to show that these problems belong to the log-space complexity class $\mathcal{L}$. Further we consider definability questions in terms of first order logic and its extensions and show that planarity of both signed and unsigned Gauss words cannot be expressed by a formula of first-order predicate logic, while extensions of first-order logic with deterministic transitive closure operator allow to define planarity of both signed unsigned Gauss words. Following the same line of research we provide lower and upper bounds for the planarity problem of Gauss paragraphs and unknottedness.

In addition we consider knot transformations in terms of string rewriting systems and provide a refined classification of Reidemeister moves formulated as string rewriting rules for Gauss words. Then we analyse the reachability properties for each type and present some bounds on the complexity of the paths between two knot diagrams reachable by a sequence of moves of the same type. Further we consider a class of non-isomorphic knot diagrams generated by type I moves from the unknot and discover that the sequence

corresponding to the number of diagrams with respect to the number of crossings is equal to a sequence related to a class of Eulerian maps with respect to the number of edges. We then investigate the bijective mapping between the two classes of objects and as a result we present two algorithms to demonstrate the transformations from one object to the other. It is known that unknotting a knot may lead to a significant increase in number of crossings during the transformations. We consider the question of designing a set of rules that would not lead to the increase in the number of crossings during knot transformations. In particular we introduce a new set moves in this regard which can be used to substitute one of the rules of type II that increases the number of crossings. We show that such new moves coupled with Reidemeister moves can unknot all known examples of complex trivial knot diagrams without increasing number of crossings.

Acknowledgements

The Department of Computer Science at the University of Liverpool has been an excellent place to conduct research and all members of staff have been friendly and helpful. I would like to kindly thank the Department of Computer Science for funding my postgraduate studies and providing me with sufficient financial assistance to attend a number of conferences

I would like to acknowledge the contribution of my supervisors, Dr. Alexei Lisitsa and Dr. Igor Potapov, to the development of this thesis and for their continual support and guidance throughout the past four years. Their enthusiasm, constructive feedback, research ideas and always making time for discussions have made the completion of my PhD. possible and quite enjoyable.

I would like to express my gratitude to my thesis adviser, Prof. Michael Fisher, for his invaluable advice throughout my PhD. study. Also I would like to thank both of my thesis examiners, Dr. Boris Konev, and Dr. Stéphane Demri for their constructive comments and attention to my work.

Last but not least, many thanks go to all of my family and friends for their encouragement and moral support.

Contents

Chapter 1

Introduction

1.1 Background

Knot theory is the area of mathematics that studies mathematical knots and links. A knot (a link) is an embedding of a circle (several circles) in 3-dimensional Euclidean space, $\mathbb{R}^3$, considered up to a smooth deformation of the ambient [1] space. It is a well established and active area of research with strong connections to topology [44], algebra [4] and combinatorics [52].

Major problems in knot theory have algorithmic or computational nature: *equivalence problem* (how to recognise that two knots are equivalent), or *unknottedness problem* (how to recognise that a knot is a trivial one). Consideration of such problems led to fruitful interactions between knot theory and computer science. In particular, the questions of computational complexity of knot problems have been addressed in [30]. Examples of other interactions include works on formal language theory [36] and quantum computing [1; 22; 45].

Algorithmic and computational topology is a new growing branch of modern topology. Much of the recent effort has focused on classifying the inherent complexity of topological problems.

One of the founding theorems of knot theory states that any two diagrams of a given knot may be changed from one into the other by a sequence of local moves referred to as Reidemeister moves [58]. This result is crucial as it allows one to define a knot

[1] An ambient space is the space surrounding a mathematical knot(or link respectively).

invariant as an invariant of a diagram which is unchanged under Reidemeister moves. In this thesis we formalise Reidemeister moves in terms of string rewriting rules for words encoding knot diagrams and analyse the minimal number of distinct rules for each type.

Decidablity of Knot Equivalence and Unknottedness problems was demonstrated more than 40 years ago [26], but the first results on the complexity of these problems only appeared much later. Hass, Lagarias and Pippenger in [30] have shown that Unknotedness and some related problems on links can be decided in Non-deterministic Polynomial Time (NP). As an upper bound, the number of Reidemeister moves sufficient to transform a knot diagram with n crossings to a trivial knot is 2^{cn} where c=10^{11} [29] and later improved to $c = 15^4$ [64]. For the general case of equivalence of knots, a new recent upper bound based on the number of Reidemeister moves for knots and links was shown to be 2^{c^n} [13] where $c = 10^6$. As to the lower bound for unknottedness, the number of Reidemeister moves required is quadratic with respect to the number of crossings [31; 32]. Furthermore, a result on a normal form was obtained where it was shown that two knots are equivalent iff one can be obtained from the other by a sequence of type I$\uparrow$[1] moves, followed by a sequence of type II$\uparrow$ moves, followed by a sequence of type III moves, followed by sequence of type II$\downarrow$[2] moves [12]. This result raises a number of questions regarding the complexity bounds of such a sequence by investigating the complexity of each move as well as the complexity of a subset of moves. In this thesis we address such questions by evaluating the length of the path between two reachable knot diagrams by each type of Reidemeister moves separately as well as by a combination of moves.

Perhaps one of the reasons the number of Reidemeister moves is exponential can be due to the existence of some knot diagrams which require an increase in their crossings number before they can be simplified into the unknot[3]. One may ask whether introducing new moves in addition to ordinary Reidemeister moves may help with simplifying trivial knot diagrams without increasing their number of crossings during the transformation (i.e. by avoiding Reidemeister moves of types I$\uparrow$ and II$\uparrow$). Such a question was investigated in [24] where some generalised version of Reidemeister moves of types

[1] A rule with an $\uparrow$ involves increasing the number of crossings

[2] A rule with an $\downarrow$ involves decreasing the number of crossings

[3] An unknot is a trivial diagram with no crossings usually represented by a standard circle

I and II (referred to as pass moves [14]) were presented. However, a counter example was presented in the same paper where it still required an increase in the number of crossings even after considering the new added moves.

In this context we will introduce a set of moves which are more general than the moves proposed in [24] but can be seen as a restrictive case of pass moves for links described in ([14], page 67). Although these moves are restrictive, we will show that all known examples of classical complex trivial knot diagrams (including Goretiz's counterexample) can be simplified using our new moves coupled with ordinary Reidemeister moves without increasing their number of crossings.

In addition, we consider a class of knot diagrams obtained by application of Reidemeister moves of type I. Starting from an unknot we generate all possible diagrams and count the number of non-isomorphic knot diagrams with respect to the number of crossings. Surprisingly, we discover that our sequence matches the sequence corresponding to a class of unrooted Eulerian n-edge planar maps with a distinguished outer face (see the On-Line Encyclopedia of Integer Sequences [63]). Then we investigate the correspondence between the two classes of objects and present explicit algorithms for the construction of knot diagrams represented by Gauss diagrams and vice versa.

One of the earliest questions of an algorithmic nature related to knots was the question of characterisation of Gauss words [23]. With every knot one can associate a word, called a Gauss word, which is a sequence of labels for the crossings read off directly from the projection of the knot on a plane. Depending on whether the information on the orientation is present, the word can be signed or unsigned. The simple property of any Gauss word is that every label (index) in it appears twice. The converse is not true, there are the words with every label appearing twice which do not correspond to any classical planar knot diagram. The question of characterisation of "true", or planar Gauss words was posed by Gauss himself [23] and was eventually resolved by Nagy in [53]. Since then many criteria and algorithms have been proposed both for recognition of signed [7; 38] and unsigned [8; 16; 17; 37; 46; 48; 57; 60; 60; 61; 62] Gauss words. The questions of computational complexity of the proposed algorithms were rarely explicitly addressed with notable exceptions being [38] where linear time algorithm for the signed case is proposed, and in [61] where a linear time complexity for unsigned case is established and compared with earlier quadratic bounds in [57].

Most of the problems of recognising knot properties (such as planarity, unknottedness, equivalence) are known to be decidable, with different time complexity. However their complexity in terms of computational power of devices needed to recognise the knot properties was not studied yet. In this thesis we will address this problem and provide first known bounds for recognisability of knot properties in terms of various automata models over *infinite* alphabets. The infinite alphabet appeared naturally due to the fact that the number of crossings in knots is unbounded.

As an alternative and complementary approach to address the complexity of knot problems, we consider definability questions about expressibility of knot properties in terms of logic and show lower and upper bounds for the planarity and unknottedness properties.

The main goal of the proposed approach is to give a new insight on knot problems and characterise knot problems according to their computational complexity.

The results presented in this thesis were achieved by a combination of methods from knot theory, automata theory and logic. Part of these results has been published in [41; 42]

1.2 Overview of the thesis

In Chapter 2 we give an introduction to knots and their representations, adopting some notions from knot theory. We begin by showing how knots can be combinatorially encoded by finite structures, such as graphs or words. We then present a set of local diagrammatic moves known as *Reidemeister moves* used to show if two of knot diagrams are of the same type and finally describe some problems related to knots that will be studied in this thesis.

In Chapter 3 we consider the formulation of Reidemeister moves in terms of string rewriting rules and analyse their reachability properties. In Section 3.2 we formalise Reidemeister moves in terms of string rewriting rules for Gauss words and analyse the minimal number of distinct rules for each type of Reidemeister moves with respect to cyclic order and renaming of labels. As a result we show that there are two minimal rules for type I, two minimal rules for type II and eight minimal rules for type III.

In Section 3.3 we analyse the reachability properties of Reidemeister moves. We evaluate the lower and upper bounds on the number of transformations for the equiva-

lence problem of two reachable knot diagrams by type I, type II and type III individually as well as a combination of rules of types $\{I, II\}$, $\{I, III\}$ and $\{II, III\}$. We show linear lower and upper bounds for types I and II, and a quadratic lower bound for type III with respect to the number of crossings in a knot diagram while for the set of rules of types $\{I, II\}$, $\{I, III\}$ and $\{II, III\}$ we provide some plausible classes of diagrams that can be used to establish some lower bounds.

In section 3.4 we consider oriented knot diagrams (represented by Gauss diagrams) generated by application of Reidemeister moves using type I moves only. Starting from an unknot (the simplest trivial diagram) we generate all possible diagrams and count the number of non-isomorphic knot diagrams with respect to the number of crossings. We show that the number of non-isomorphic knots with n-crossings is equal to the number of unrooted Eulerian n-edge maps with a distinguished outer face in the plane. Then we investigate the bijective mappings between the two classes of objects and provide two explicit algorithms to demonstrate the construction of Gauss diagrams from maps and vice versa.

In Section 3.5 we introduce a new set of moves referred to as generalised Reidemeister moves to be used in companion with classical Reidemeister moves for the purpose of simplifying complex trivial knot diagrams into the unknot without increasing their number of crossings during the transformation. Considering all known classical examples of complex trivial knot diagrams we show that each diagram can be transformed into the unknot without exceeding number of crossings of the original diagram.

In Chapter 4 we consider several computational models to be used for the purpose of evaluating the complexity of some knot theoretic problems represented by Gauss words. In section 4.1 we describe and extend the models of automata over infinite alphabet that will be used for establishing the lower and upper bounds on recognition of knot properties.

In Section 4.2 we show the simulation of counters and pebbles by register automata and demonstrate generic results on the mutual simulations between logspace bounded classical computations (over finite alphabets) and register automata working over infinite alphabets.

In Section 4.3 we apply register automata to establish some lower and upper bounds for the recognisability of some knot properties. We show that the languages of Gauss words (signed and unsigned) are not recognisable by a non-deterministic 1-way register

automata while the same languages are recognisable by a deterministic 2-way register automata. Although register automata is one of the weakest models of automata over infinite alphabet, we show that it can recognise non-trivial properties, specifically we show that a deterministic 2-way register automaton can recognise whether two Gauss words are isomorphic (i.e. two Gauss words are isomorphic if they are equivalent upto cyclic shift and renaming of labels).

In Chapter 5 we investigate the descriptional complexity of knot theoretic problems and show upper bounds for planarity problems of signed knot diagrams represented by Gauss words and signed link diagrams represented by Gauss Paragraph (to determine whether a given signed Gauss word (paragraph) is *planar*, i.e. encodes a plane diagram of a classical knot (link) in $\mathbb{R}^3$.)

In Section 5.1 we show that the language of planar signed Gauss words can be recognised by deterministic two-way register automata by simulating the algorithm presented in [7] and in Section 5.2 we show that the language of planar signed Gauss paragraphs (a set of Gauss words representing links) can be recognised by deterministic two-way register automata simulating the recently discovered linear time algorithm in [38]. Furthermore in Section 5.3 we translate this result in the classical settings and show that the languages of planar signed Gauss words and planar signed Gauss paragraphs belong to the deterministic log-space complexity class $\mathcal{L}$.

In Chapter 6 we continue the same line of research as in the previous chapter but focussing mainly on the *unsigned case*. In Section 6.1 we provide an analysis of Cairns-Elton algorithm for planarity of unsigned Gauss words and show that it is implementable by co-non-deterministic register automata and in Section 6.2 we show that planarity of unsigned Gauss paragraph is recognisable by a linearly bounded memory automata simulating Kauffman algorithm [37]. Further we refine our results to demonstrate that the Cairns-Elton algorithm can be implemented in $\mathcal{SL}$ (symmetric logspace) and therefore in $\mathcal{L}$. As a consequence we show that planarity of unsigned Gauss words is recognisable by deterministic register automata.

An alternative and to some extent a complementary approach to the study of descriptional complexity of recognisability problems is that based on definability in some logic. In Chapter 7 we pose the questions of definability of the knot properties by their expressibility in first order logic and its extensions. We show lower and upper bounds

for definability of planarity of Gauss words and Gauss paragraphs, and the unknottedness encoded by logical structures. In Section 7.1 we introduce first-order predicate logic and its inductive extensions. Then we define two encodings for Gauss words as logical structures and describe the conditions for Hanf locality that is widely used to prove undefinability in first-order predicate logic

In Section 7.2 we show that planarity of both signed and unsigned Gauss words can not be defined by a formula of first-order predicate logic, while extensions of first-order logic with *deterministic* transitive closure operators allow to define planarity of signed and unsigned Gauss words and in Section 7.3 we show that the property of unknottedness can not be defined in FO and demonstrate an implicit upper bound by showing that it is definable in existential second order logic relying on the fact that $NP = ESO$ [21].

Chapter 2

Knots and their representations

In this chapter we give an introduction to knots and their representations adopting some notions from knot theory. We begin by showing how knots can be combinatorially encoded by finite structures, such as graphs or words. Then we present a set of local diagrammatic moves known as *Reidemeister moves* used to show if two of knot diagrams are of the same type and finally describe some problems related to knots that will be studied in this thesis.

2.1 Knots representations

A knot is defined as a simple closed curve in three-dimensional Euclidean space. There are various discrete representations for knots. One of such discrete representations is a Gauss word consisting of a sequence of symbols (labels O ("over") and U ("under") with indices and signs), which can be read off directly from a projection of the knot on a plane.

2.1.1 Knot diagrams

A knot diagram is a picture of a projection or a shadow of a knot onto a plane with the restriction that each point on the diagram is the shadow of no more than two points called a crossing.

At each crossing we create small breaks in the strand that passes underneath (as depicted in the left-hand side of Figure 2.1) to distinguish between the over-strand and

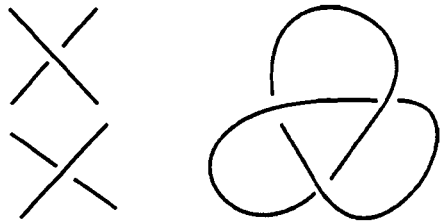

Figure 2.1: Trefoil - An example of a knot diagram

the under-strand so that the original knot can be reconstructed. The knot diagram of the trefoil knot is illustrated on the right-hand side of Figure 2.1.

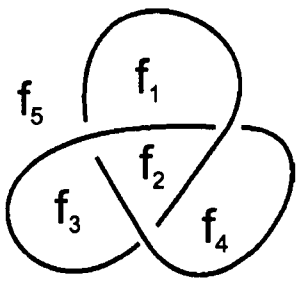

Figure 2.2: - Faces of a trefoil knot diagram

It's clear that in addition to crossings, a knot diagram contains faces (cycles) bounded by arcs (edges connecting the crossings). For example in Figure 2.2, the trefoil knot diagram contains 5 faces labelled $f_1, \cdots, f_5$. The face f_5 will be referred to as an outer face and the faces $f_1, \cdots, f_4$ will be referred to as inner faces.

Oriented knots. A choice of a direction to travel around a knot is called an orientation of the knot. An oriented knot is a knot together with a specific choice of orientation as shown in Figure 2.3. In this thesis we consider oriented knot diagrams only.

Fixing an orientation of a knot, one needs to distinguish between the two types of crossings corresponding to right-handed twists and left-handed twists depicted in Figure 2.4. To represent the difference, two opposite signs will be assigned; a + sign (for a right-handed twist) and a − sign (for a left-handed twist). One type of crossing will be called positive and the other type will be called negative. The crossing is positive if one can rotate the under-strand in a clock-wise direction so that its arrow-head is in line with the arrow head of the top-strand.

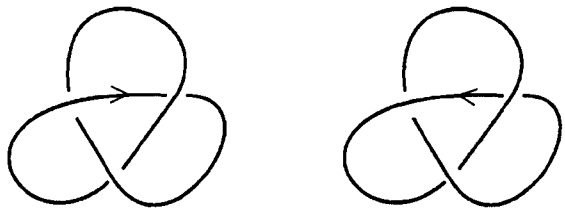

Figure 2.3: Trefoil - An example of oriented knot diagrams

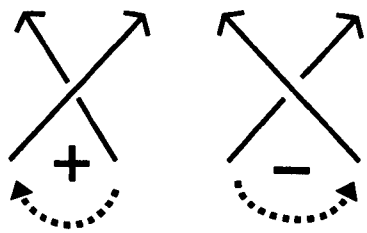

Figure 2.4: Types of crossings - positive and negative crossings

Although knot diagrams provide a clear visual impression of knots, another discrete representation in terms of strings or words would be more suitable for analysing their computational properties. One such representations is Gauss words. Gauss words considered as finite sequences of letters representing crossings for which every letter in the Gauss word appears exactly twice [23].

2.1.2 Gauss words

The Gauss word is obtained from an oriented knot diagram by first labelling each crossing with a number and indicating the sign of a crossing as in Figure 2.4. Then we choose a starting point (at any place other than a crossing) on the knot diagram and walk along the diagram following the chosen orientation. At each crossing encountered we record the name of the crossings and whether the walk takes us over it or under it until we arrive back at our starting point. For signed Gauss words, we also record whether the sign of the crossing is positive or negative (see Figure 2.5).

As to shadow Gauss words, only the labels for the crossings are recorded. The signs and information about over-crossing and under-crossing are not considered.

A Gauss word w can be described as a sequence of pairs $(a_1, b_1), \cdots, (a_{|w|}, b_{|w|})$

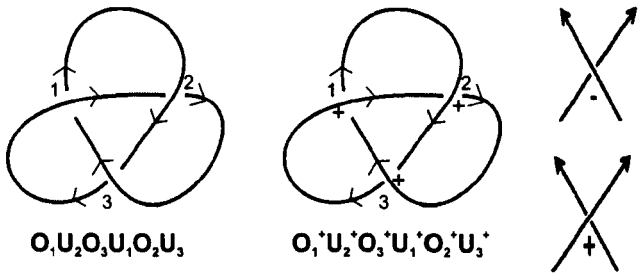

Figure 2.5: - Trefoil with its corresponding Gauss words (unsigned and signed)

where the first component consists of the labels $a_1, \cdots, a_{|w|}$ which is from a finite set ($\{U, O\}$) and the second component consists of the data values $b_1, \cdots, b_{|w|}$ taken from an infinite set ($\mathbb{N}$). This description is known as *data words* [5; 54].

In this dissertation, Gauss words (signed, unsigned and shadow) will be considered by default as cyclic words except in the context of automata where Gauss words are considered to be linear words as inputs for automata.

For a word w and a symbol d denote by $\mid w \mid_d$ the number of occurrences of d in w. As usual $\mid w \mid$ denotes the length of the word w.

Formal definitions for signed, unsigned and shadow Gauss words are given below.

Definition 2.1.1. A signed Gauss word *w is a data word over the alphabet $\Sigma \times \mathbb{N}$ where $\Sigma = \{U^+, O^+, U^-, O^-\}$, such that for every n either*

- $|w|_{(U^+,n)} = |w|_{(O^+,n)} = |w|_{(U^-,n)} = |w|_{(O^-,n)} = 0$, *or*
- $|w|_{(U^+,n)} = |w|_{(O^+,n)} = 1$ *and* $|w|_{(U^-,n)} = |w|_{(O^-,n)} = 0$, *or*
- $|w|_{(U^-,n)} = |w|_{(O^-,n)} = 1$ *and* $|w|_{(U^+,n)} = |w|_{(O^+,n)} = 0$.

The language of all signed Gauss words is denoted by L_{SGW}.

Definition 2.1.2. An unsigned Gauss word *w is a data word over the alphabet $\Sigma \times \mathbb{N}$ where $\Sigma = \{U, O\}$, such that for every $n \in \mathbb{N}$ either*

- $|w|_{(U,n)} = |w|_{(O,n)} = 0$, *or*
- $|w|_{(U,n)} = |w|_{(O,n)} = 1$.

The language of all unsigned Gauss words is denoted by L_{UGW}.

Definition 2.1.3. A shadow Gauss word w *is a word over the alphabet* $\mathbb{N}$ *(i.e. finite sequence of natural numbers) such that for every* $n \in \mathbb{N}$ *either* $|w|_n = 0$ *or* $|w|_n = 2$ *(i.e. every label in* w *should appear exactly twice).*

The language of shadow Gauss words is denoted by L_{ShGW}.

Throughout this dissertation we will use the notion of interlacement graph which is a graph associated with a shadow Gauss word, defined as follows:

Definition 2.1.4. *Given a shadow Gauss word* w*, the vertices of the interlacement graph* $G(w)$ *correspond to the labels in* w *and the edges of* $G(w)$ *are the pairs of labels* (i, j) *such that* i *and* j *are interlaced in* w *if* i *occurs once between the two occurrences of* j *and vice versa.*

For an example see Figure 2.6.

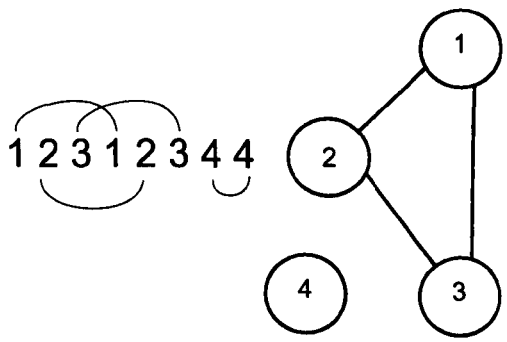

Figure 2.6: - An interlacement graph $G(w)$ for $w = 12312344$

Another useful discrete representation of knots is known as Gauss diagrams.

2.1.3 Gauss diagrams

A Gauss diagram is a diagrammatic representation of a Gauss word of the knot depicted in Figure 2.7. We form a Gauss diagram by taking an oriented circle with a basepoint chosen on the circle.

Walk along the circle marking it with the labels for the crossings in the order of Gauss word. Now connect two points of the same label by an edge from the inside of

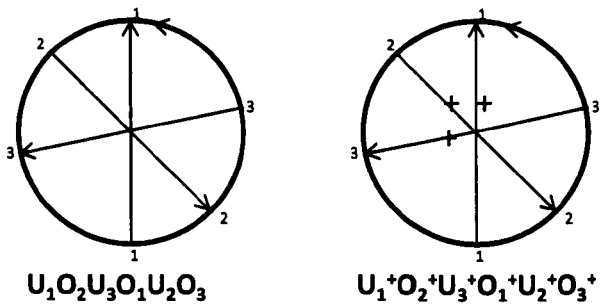

Figure 2.7: - Gauss words with its corresponding Gauss diagrams (signed and unsignd)

the circle (such an edge is referred to as a chord). Orient each chord from overcrossing site to undercrossing site. Mark each chord with a + or a − according to the sign of the corresponding crossing label in the Gauss word.

2.2 Reidemeister moves

Two knots are isotopic if one can be continuously manipulated in 3-space until it looks like the other. Reidemeister in [58] demonstrated that knot diagrams of isotopic knots can be connected by a sequence of operations. These operations are referred to as Reidemeister moves. A Reidemeister move refers to one of the following local moves on a knot diagram:

Move I. Twist and untwist in either direction as illustrated in Figure 2.8.

Figure 2.8: Type I - Type I Reidemeister moves

Move II. Move one loop completely over another as depicted in Figure 2.9.

Figure 2.9: Type II - Type II Reidemeister moves

Move III. Move a strand completely over or under a crossing as indicated in Figure 2.8.

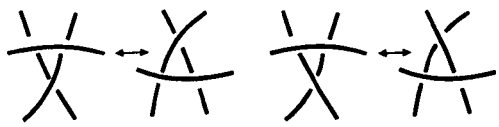

Figure 2.10: Type III - Type III Reidemeister moves

Theorem 2.2.1. *([58]) Two knot diagrams are equivalent if and only if one can be obtained from the other by a sequence of Reidemeister moves.*

Example Given two knot diagrams K_1 and K_2 illustrated in Figure 2.11, we will show that $K_1 = K_2$ by transforming K_1 into K_2 using a sequence of Reidemeister moves.

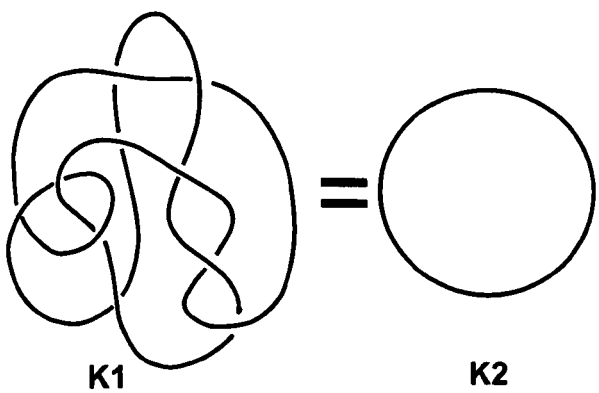

Figure 2.11: Example - K_1 and K_2 are both trivial knot diagrams

The moves shown in Figures 2.8 to 2.10 are intended to indicate local changes that are made in a larger diagram. Figure 2.12 shows a sequence of Reidemeister moves used for transforming K_1 to K_2.

2.3 Knot problems

Planarity, equivalence and unknottedness are three main computational problems in knot theory. In this section we introduce and formulate these problems.

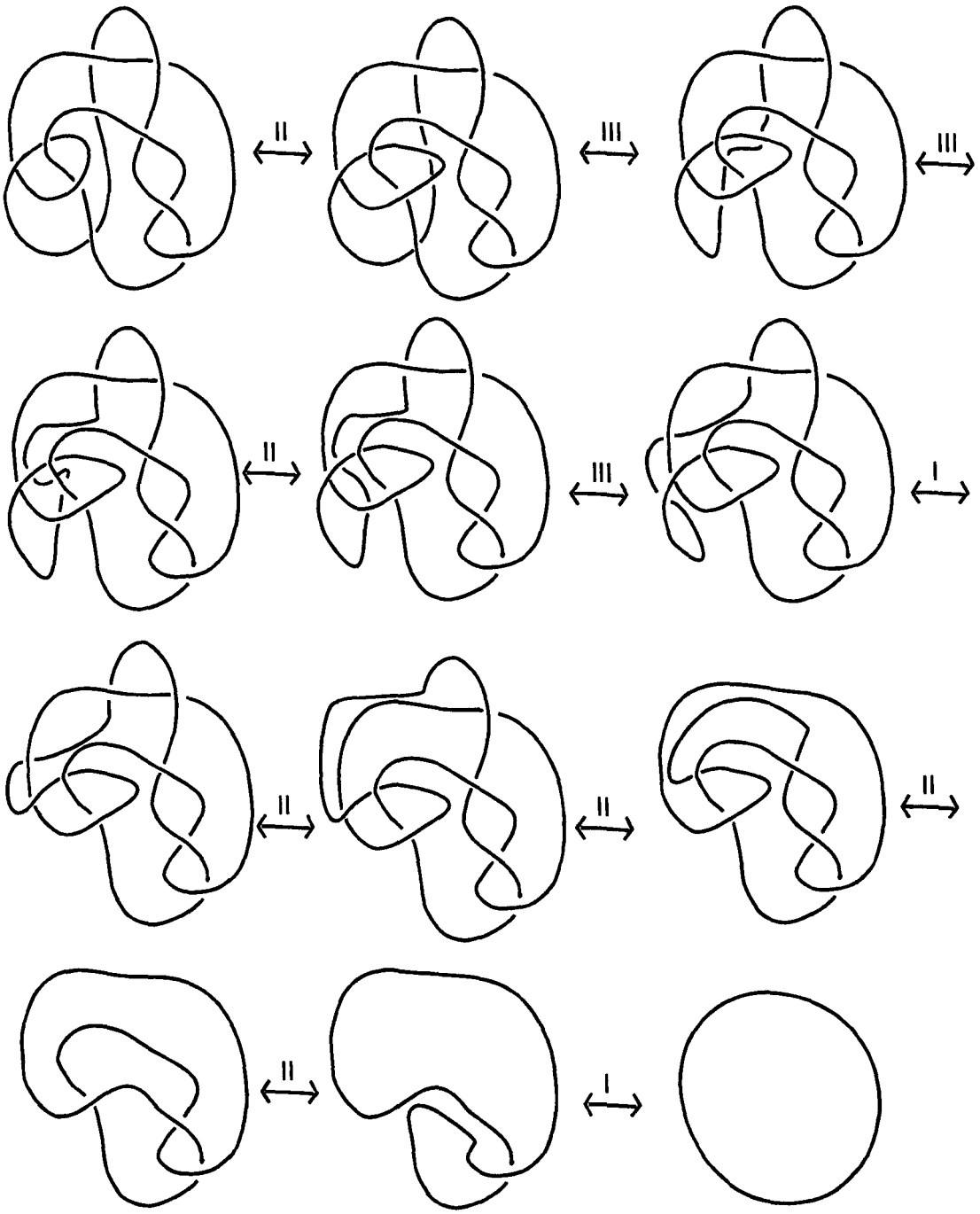

Figure 2.12: Knot transformation - Transforming K_1 into K_2 by Reidemeister moves

2.3.1 Planarity

The construction of a Gauss word is quite straightforward as it can be obtained by starting from a non-crossing point on the curve and writing down the labels of the crossings and their types of strand that occur as the curve is traversed according to the orientation of the knot until the same starting point is met for the second time. However, the inverse problem of reconstructing a knot diagram from a Gauss word is harder because the construction does not always results in a planar diagram. So if an arbitrary Gauss word does not encode a classical diagram then such Gauss word will correspond to a non-classical knot diagram, i.e. a diagram that will contain additional crossings which do not appear in the Gauss word (see Figure 2.13). Such crossings are known as virtual crossings. This observation was one of the motivations for introducing virtual knot theory [37]. A Gauss word representing a knot diagram on a plane without virtual crossings is called classical or planar. The problem of recognising planar Gauss words have been formulated by Gauss himself and recently several algorithmic solutions for both signed and unsigned cases have been proposed.

Planar signed Gauss words. The planarity problem of signed Gauss words has been studied first in [9]. Two different algorithms has been proposed by [7; 38] based on the work of Carter in [9]. These two algorithms are described in details in Chapter 5.

The planarity question for **signed Gauss words** can be formulated as follows:

Problem 1. *(Planarity of signed Gauss words)*

Instance: Given a signed Gauss word w.
Question: Does w represents a planar knot diagram?

In Section 2.1.2 we described that for every knot, one can produce a corresponding Gauss word w. However, not every Gauss word correspond to a classical knot (i.e a knot is classical if its projection can be embedded in the plane without self-intersections). For example the Gauss word $w = O_1^- O_2^- U_1^- O_3^+ U_2^- U_3^+$, does not correspond to any classical knot and may only correspond to a virtual knot (a non-planar diagram) presented in Figure 2.13 (to distinguish between classical crossing and virtual crossings, the virtual crossing are marked by a circle on the diagram).

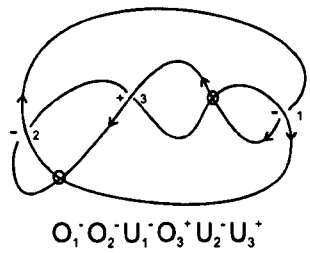

Figure 2.13: Non-planar knot diagram - An example of a non-planar knot diagram with 2 virtual crossings

There are many criteria for characterising the planarity property for knots, in particular Cairns-Elton algorithm in [7] can be applied to check the Gauss word w. Therefore it can be determined that w is non-planar because the first condition of algorithm does not hold. The algorithm presented by Cairns and Elton in [7] is discussed in details in Chapter 5, Section 5.1.1.

Planar unsigned Gauss words. The planarity problem of unsigned Gauss words has generated a lot of interest since it was posed by Gauss in [23]. As a result, several algorithmic solutions have been proposed, e.g. in [8; 16; 37; 46; 48; 57; 61; 62].

The planarity question for **unsigned Gauss words** is formulated as follows:

Problem 2. *(Planarity of unsigned Gauss words)*

Instance: Given an unsigned Gauss word w.

Question: Does there exist a choice of signs that can be assigned to w so that w represents a planar knot diagram?

2.3.2 Equivalence

The equivalence problem is a central problem in knot theory. The problem has shown to be decidable [26] and recently a new upper bound based on the number of Reidemeister moves was shown to be highly exponential with respect to the number of crossings [13].

Problem 3. *(Equivalence)*

Instance: Given two knot diagrams K_1 and K_2.

Question: Can K_1 be transformed into K_2 by a sequence of Reidemeister moves?

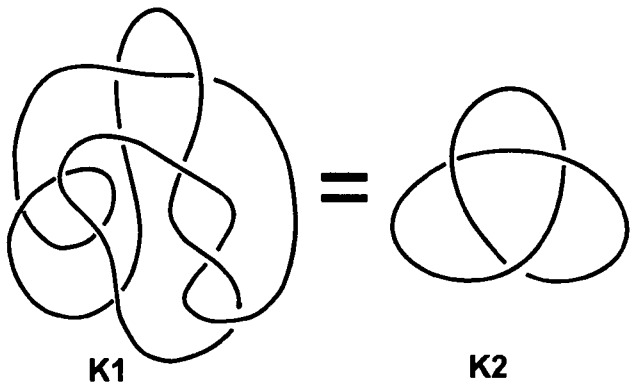

Figure 2.14: non-trivial knot diagrams - an example of two equivalent non-trivial knot diagrams

2.3.3 Unknottedness

Unknottedness (also referred to as the unknotting problem) can be seen as a special case of equivalence. The problem has been shown to be in the class NP [30] and in the same paper the authors conjectured that the unknottedness problem is contained in NP ∩ co.NP. Recently it was shown that the unknottedness problem is contained in AM ∩ co-AM [28] (i.e. this is the class of decision problems for which both "yes" and "no" answers can be verified by an Arthur-Merlin protocol). The first algorithm for the unknottedness problem was presented in [26] based on normal surface theory and since then a new algorithm based on arc-presentation theory was presented in [19]. As an upper bound for the Reidemeister moves sequence, it was shown in [29; 64] that the number of Reidemeister moves sufficient to transform a knot diagram with n-crossings to the unknot is exponential with respect to the number of crossings and as a lower bound, the number of Reidemeister moves required is quadratic with respect to the number of crossings [31; 32]. The unknottedness problem is formulated as follows:

Problem 4. *(Unknottedness)*

Instance: Given a knot diagram K

Question: Can K be transformed into an unknot by a sequence of Reidemeister moves?

A knot diagram is called *trivial* if it can be transformed by a sequence of Reidemeister moves into the unknot otherwise it is called $non - trivial$. For an example of trivial

knot diagrams see Figure 2.11.

2.4 Summary

In this chapter we gave an introduction about knots and their representations. We have described how knots can be encoded by Gauss words and similarly by Gauss diagrams in a discrete way so that problems about knots can be reduced to the questions about Gauss words or diagrams. In order to estimate and characterise complexity of knot problems represented by Gauss words or Gauss diagrams, we will use different tools and mathematical models including automata models, computational complexity and definability in logic.

Chapter 3

Knot rewriting and algorithmic problems for knots

In this chapter we consider knot transformations in terms of string rewriting systems. We begin with Section 3.5.1 where we introduce some definitions and notation to formulate the concept of knot transformations in the context of Gauss word rewriting. Then we move to Section 3.2 to consider some local diagrammatic moves presented in [58] known as Reidemeister moves which are used for the transformations between two knots of the same type. We formulate the Reidemeister moves in terms of string rewriting rules for Gauss words and analyse the minimal set of rules sufficient for rewriting. Then in Section 3.3 we investigate the reachability properties of each type and provide some lower and upper bounds on the complexity of the paths between two knot diagrams reachable by a sequence of moves of the same type.

Further in Section 3.4 we consider a class of oriented knot diagrams (represented by Gauss diagrams) generated by application of Reidemeister moves of type I only. Starting from an unknot (a diagram with no crossings) we generate all possible diagrams. We are interested in the question of how many non-isomorphic diagrams can be obtained from the unknot for a fixed number of crossings and discover that the number of non-isomorphic knot diagrams with n-crossings (where $n \geq 1$) is equal to the number of unrooted Eulerian n-edge maps in the plane. Then we investigate the bijective map between the two classes of objects and as a result we present two explicit algorithms to describe the transformations from Gauss diagrams to maps and vice versa.

Furthermore in Section 3.5 we consider the question of knot transformations by means of reduction and introduce a new set of moves that can be seen as a more generalised version of Reidemeister moves of types II and III formulated also in terms of string rewriting rules for Gauss words. It's known that transforming a knot into an unknot can lead to a significant increase in the number of crossings during the transformations, we demonstrate that our moves coupled with Reidemeister moves can unknot all known examples of complex trivial knot diagrams presented in the literature [24; 32; 47] without increasing number of crossings.

3.1 Definitions and notation

In this section we introduce some definitions needed to formulate knot rewriting.

Definition 3.1.1. *Let Σ be an alphabet, a cyclic shift s_k with $k \in \mathbb{N}$ is a function $s_k : \Sigma^* \to \Sigma^*$ such that for a word $w \in \Sigma^*$ where $w = w_1, ..., w_n$, the cyclic shift of w is defined as $s_k(w_1, ..., w_n) = w'_i, ..., w'_n$ where $w_{(i+k) \pmod n} = w'_i$ for some $i = 1, ..., n$.*

Definition 3.1.2. *Let w and w' be some Gauss words, $w \equiv^c w'$ (w is equivalent to w' up to cyclic shift) iff $|w| = |w'| = n$ such that $\exists k : 0 \leq k < n$ and $w = s_k(w')$.*

By $[w]_c$, we denote a c-equivalence classes of w.

Example. *Let $w = O_1U_2O_3U_1O_2U_3$, the following words are equivalent words to w up to cyclic shift:*

- $s_1(w) = U_2O_3U_1O_2U_3O_1$
- $s_2(w) = O_3U_1O_2U_3O_1U_2$
- $s_3(w) = U_1O_2U_3O_1U_2O_3$
- $s_4(w) = O_2U_3O_1U_2O_3U_1$
- $s_5(w) = U_3O_1U_2O_3U_1O_2$

Definition 3.1.3. *Let* $w = (a_1, b_1), (a_2, b_2), ..., (a_n, b_n)$ *where* $a \in \{O, U\}$ *and* $b \in \mathbb{N}$, $w \equiv^r w'$ *(w is equivalent to w' up to renaming of labels) iff there exists a bijective mapping r:* $\mathbb{N} \to \mathbb{N}$ *such that* $w' = (a_1, r(b_1)), (a_2, r(b_2)), ..., (a_n, r(b_n))$.

By $[w]_r$, *we denote an r-equivalence classes of w.*

Example. *Let* $w = O_1U_2O_3U_1O_2U_3$ *and* $w' = O_2U_3O_1U_2O_3U_1$. *To show that* $w \equiv^r w'$. *We check that the symbols of the first component of w (OUOUOU) have the same order the same as those in the first component of w' and there is a bijective mapping between the elements of the second component of w (123123) and the elements of the second component of w'. That is* $1 \to 2$, $2 \to 3$ *and* $3 \to 1$.

Definition 3.1.4. *Let w and w' be two Gauss words,* $w \equiv^{cr} w'$ *(w is equivalent to w' up to cyclic shift and renaming of labels) iff* $\equiv^{cr} = (\equiv^c \cup \equiv^r)^*$. *By* $[w]_{cr}$, *we denote a cr-equivalence classes of w.*

Example. *Let* $w = O_1O_2O_3U_1U_2U_3$ *and* $w' = O_2U_3U_1U_2O_3O_1$. *To illustrate that* $w \equiv^{cr} w'$. *First we apply cyclic shift to w' to obtain* $s_4(w') = O_3O_1O_2U_3U_1U_2$ *and then determine the bijective mapping between the elements of the second component of w (123123) and the elements of the second component of w' . That is* $1 \to 3$, $2 \to 1$ *and* $3 \to 2$.

In the next definitions, we define the string rewriting rules system.

Definition 3.1.5. *Let X denote a finite set of variables and* Σ *denote an alphabet such that* $\Sigma = \{(O, i) \mid i \in I\} \cup \{(U, i) \mid i \in I\}$ *where I is a finite set.*

For an alphabet Σ, *the language of all cyclic words over* Σ *is defined as* $\Sigma_c^* = \Sigma^* \backslash_{\equiv^c} = \{[w]_c | w \in \Sigma^*\}$

We define a Gauss string rewriting system T as a tuple (X, Σ_c, R), *where R is a set of rewriting rules of the form* $l \leftrightarrow r$ *such that* $l, r \in (\Sigma \cup X)_c^*$ *and* $var(l) = var(r)$ *where* $var(l)$ *(or* $var(r)$*) denotes the set of variables of in l (or in r respectively).*

Let σ denote a ground substitution defined as a function $\sigma : X \to \Sigma^*$ which can be extended homomorphically (and preserving the name) to $\sigma : (\Sigma \cup X)^* \to \Sigma^*$. In

here, we will use σ to denote a quotient mapping $\sigma : (\Sigma \cup X)_c^* \to \Sigma_c^*$. Application of a substitution σ to a word w is denoted by $w\sigma$.

Definition 3.1.6. *Let $R = \{t_1, t_2, ..., t_n\}$ denote the set of rewriting rules and let $t \in R$, a one-step rewriting relation $\Rightarrow_t \subseteq \Sigma_c^* \times \Sigma_c^*$ where $t = l \leftrightarrow r$ is defined as follows: $w \Rightarrow_t w'$ iff $\exists\sigma$ from $(\Sigma \cup X)_c^* \to \Sigma_c^*$ and $[w]_c = l\sigma$ and $[w']_c = r\sigma$ or $[w]_c = r\sigma$ and $[w']_c = l\sigma$.*

$\Rightarrow_R = \bigcup_{t \in R} \Rightarrow_t$. *The reflexive transitive closure of $\Rightarrow_R$ is denoted by $\Rightarrow_R^*$.*

3.2 Classification of Reidemeister moves as rewriting rules

In this section, we consider Reidemeister moves as rewriting rules for Gauss words. We analyse the minimal number of distinct rules for each type of Reidemeister moves. Because the symbols on a Gauss word follow some cyclic order, the Reidemeister moves will take into account orientation of the strands. So for each type, we will consider all possible orientations of the strands involved and the order in which they are visited during the traversal of the knot diagram. We will show that there are two distinct rewriting rules for type I, two distinct rewriting rules for type II and eight distinct rewriting rules for type III.

In this section our goal is to define Reidemeister moves as Gauss string rewriting rules to make the following diagram in Figure 3.1 commutative. We demonstrate that if a knot diagram K_1 is transformed into K_2 by a Reidemeister move of type i where $i \in \{I, II, III\}$ then similarly the Gauss word corresponding to K_1 ($Gw(K_1)$) can be transformed into $Gw(K_2)$ by a rewriting rule of type i.

3.2.1 Type I

Type I move involves a single strand and can be applied to any part of the knot by either introducing a "kink" (a simple loop) which in turn increases the number of crossings by one or removing the kink, thereby decreasing the number of crossings by one. The rule corresponding to type I moves which increases (or decreases respectively) the number of crossings will be denoted by $I \uparrow$ (or $I \downarrow$ respectively). We illustrate all possible variants of type I shown in Figure 3.2 that can be obtained by twisting a kink in two

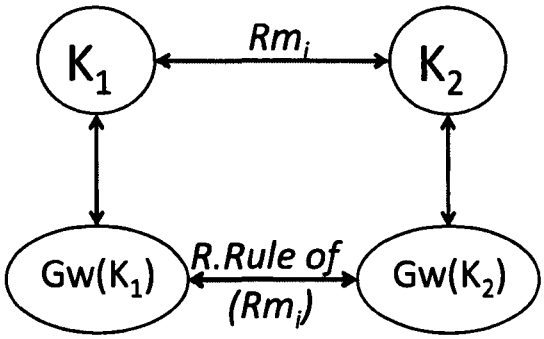

Figure 3.1: - Reidemeister moves on knot diagrams correspond to rewriting rules on Gauss words

different directions (clockwise and anti-clockwise) and assigning to a kink two opposite orientations.

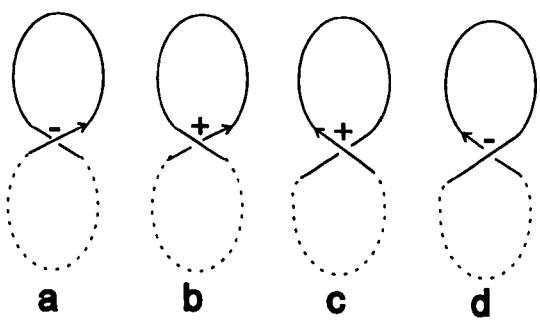

Figure 3.2: - All variants of type I

Let the set $\Omega_1 = \{a, b, c, d\}$ denote all variants of type I pictured in Figure 3.2 such that x is a subword which corresponds to the curve with dashed lines. Then $a = xO_iU_i \leftrightarrow x$, $b = xU_iO_i \leftrightarrow x$, $c = O_iU_ix \leftrightarrow x$ and $d = U_iO_ix \leftrightarrow x$.

In the following proposition, we will show that for the purpose of rewriting Gauss words[1], the following set of rules {a,b} is equivalent to the set Ω_1. Diagrammatic representations of the two rules are shown in Figure 3.3.

Proposition 3.2.1. $\Rightarrow^*_{\Omega_1} = \Rightarrow^*_{\{1.1,1.2\}}$

1.1 $xO_iU_i \leftrightarrow x$

[1]Recall from Section 2.1.2 that Gauss words are considered by default as cyclic words

1.2 $xU_iO_i \leftrightarrow x$

Proof. Consider cyclic permutations of each element in Ω_1. As a result of cyclic shift $a = xO_iU_i$, $b = xU_iO_i$, $c = xO_iU_i$ and $d = xU_iO_i$. Hence a and c are equivalent to 1.1 and b and d are equivalent 1.2 □

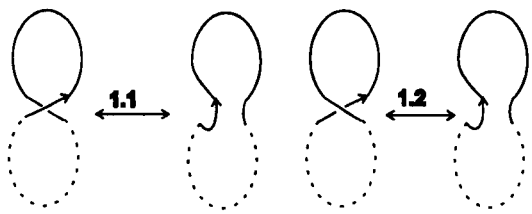

Figure 3.3: Type I - Minimal classes of type I

3.2.2 Type II

Type II moves involve the interaction between two strands such that either one strand is placed on top of the other creating two new crossings or the two strands are pulled apart removing two crossings. The rule corresponding to type II moves which increases (or decreases respectively) the number of crossings will be denoted by II↑ (or II↓ respectively). We observe that moves of Type II↑ can only be applied to two strands

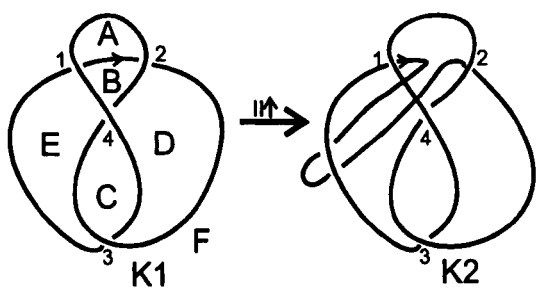

Figure 3.4: - Application of type II↑ to two strands in $K1$ that do not share a common face

which share the same face in a knot diagram otherwise there will be some intersections with other intermediate strands that may result in creating additional crossings (known as virtual crossings [37]) which will not be present in the corresponding Gauss

word. For an example, let $K1$ be a knot diagram (with labelled faces from A to F) representing the Gauss word $w = U_1U_2O_3U_4O_2O_1O_4U_3$ and let $K2$ represents the new word w' obtained from w by applying type II↑ move to U_1U_2 and U_3U_1 in w. Now considering the two diagrams $K1$ and $K2$ illustrated in Figure 3.4, it's clear that the strands in $K1$ corresponding to U_1U_2 and U_3U_1 do not share a common face between them and in order to connect them we had to cross an intermediate strand (resulting in two additional crossings) which are not taken into account in w'.

For the purpose of rewriting we only need to consider variants that correspond to oriented knot diagrams. In Figure 3.5 we present all possible variants for type II rules obtained by considering all possible orientations of the strands involved and the order in which they are visited during the traversal of the knot diagram.

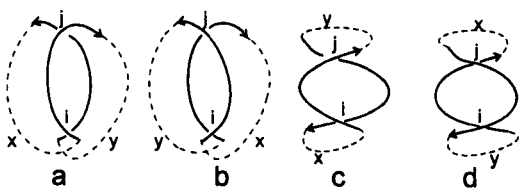

Figure 3.5: Type II - All variants of type II

Let the set $\Omega_2 = \{a, b, c, d\}$ denote all variants of type II pictured in Figure 3.5 such that x and y are subwords corresponding to the two curves with dashed lines. Then $a = xO_iO_jyU_iU_j \leftrightarrow xy$, $b = xO_iO_jyU_iU_j \leftrightarrow xy$, $c = xO_iO_jyU_jU_i \leftrightarrow xy$ and $d = xO_iO_jyU_jU_i \leftrightarrow xy$.

In the following proposition, we will show that for the purpose of rewriting Gauss words, the following set of rules {a,c} is equivalent to Ω_2. Diagrammatic representations of the two rules are shown in Figure 3.6.

Proposition 3.2.2. $\Rightarrow^*_{\Omega_2} = \Rightarrow^*_{\{2.1,2.2\}}$

2.1 $xO_iO_jyU_iU_j \leftrightarrow xy$

2.2 $xO_iO_jyU_jU_i \leftrightarrow xy$

Proof. The proof is similar to that in type I. Here, a and in b are equivalent to 2.1 and c and d are equivalent to 2.2. □

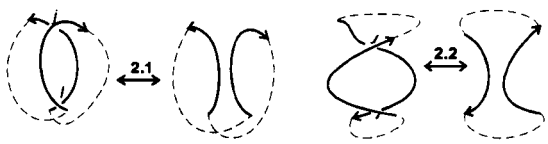

Figure 3.6: Type II - Minimal classes of type II

3.2.3 Type III

Type III moves involve the interaction between three strands with triple crossing points which share the same face; a top strand (going over two crossings), a middle strand (going over one crossing and under the other) and a bottom strand (going under two crossings). A type III move does not change the number of crossings in a knot but rather it rearranges the order of crossings.

Variants of type III If one considers all possible orientations of the involved strands and their order in which they appear in the knot. Then as shown in Figure 3.7, there are 8 possible ways in which the three strands can be oriented, plus 4 different forms in which the three strands may appear in the knot in a legal type III move and 2 different ways in which the strands can be ordered in the knot. So in total there are 64 variants corresponding to a particular move and another 64 variants corresponding to the inverse move. These are listed below in terms of rewriting rules.

1 $xO_iO_jyU_kU_jzU_iO_k \leftrightarrow xO_kO_iyU_kU_jzO_jU_i$

2 $xO_iO_jyU_kU_izO_kU_j \leftrightarrow xO_iO_jyU_jU_kzU_iO_k$

3 $xO_iO_jyU_iO_kzU_kU_j \leftrightarrow xO_kO_iyO_jU_izU_kU_j$

4 $xO_iO_jyO_kU_jzU_kU_i \leftrightarrow xO_iO_jyU_iO_kzU_jU_k$

5 $xO_iO_jyU_iO_kzU_jU_k \leftrightarrow xO_kO_iyO_jU_izU_jU_k$

6 $xO_iO_jyO_kU_izU_kU_j \leftrightarrow xO_iO_jyU_jO_kzU_iU_k$

7 $xO_iO_jyU_jU_kzU_iO_k \leftrightarrow xO_kO_iyU_jU_kzO_jU_i$

8 $xO_iO_jyU_kU_jzO_kU_i \leftrightarrow xO_iO_jyU_iU_kzU_jO_k$

9 $xO_iO_jyU_iO_kzU_jU_k \leftrightarrow xO_kO_iyO_jU_izU_jU_k$

10 $xO_iO_jyO_kU_izU_kU_j \leftrightarrow xO_iO_jyU_jO_kzU_iU_k$

11 $xO_iO_jyU_jU_kzU_iO_k \leftrightarrow xO_kO_iyU_jU_kzO_jU_i$

12 $xO_iO_jyU_kU_jzO_kU_i \leftrightarrow xO_iO_jyU_iU_kzU_jO_k$

13 $xO_iO_jyU_kU_jzU_iO_k \leftrightarrow xO_kO_iyU_kU_jzO_jU_i$

14 $xO_iO_jyU_kU_izO_kU_j \leftrightarrow xO_iO_jyU_jU_kzU_iO_k$

15 $xO_iO_jyU_iO_kzU_kU_j \leftrightarrow xO_kO_iyO_jU_izU_kU_j$

16 $xO_iO_jyO_kU_jzU_kU_i \leftrightarrow xO_iO_jyU_iO_kzU_jU_k$

17 $xO_iO_jyU_kU_jzO_kU_i \leftrightarrow xO_kO_iyU_kU_jzU_iO_j$

18 $xO_iO_jyU_iU_kzO_kU_j \leftrightarrow xO_iO_jyU_kU_jzU_iO_k$

19 $xO_iO_jyO_kU_izU_kU_j \leftrightarrow xO_kO_iyU_iO_jzU_kU_j$

20 $xO_iO_jyO_kU_jzU_iU_k \leftrightarrow xO_iO_jyU_iO_kzU_kU_j$

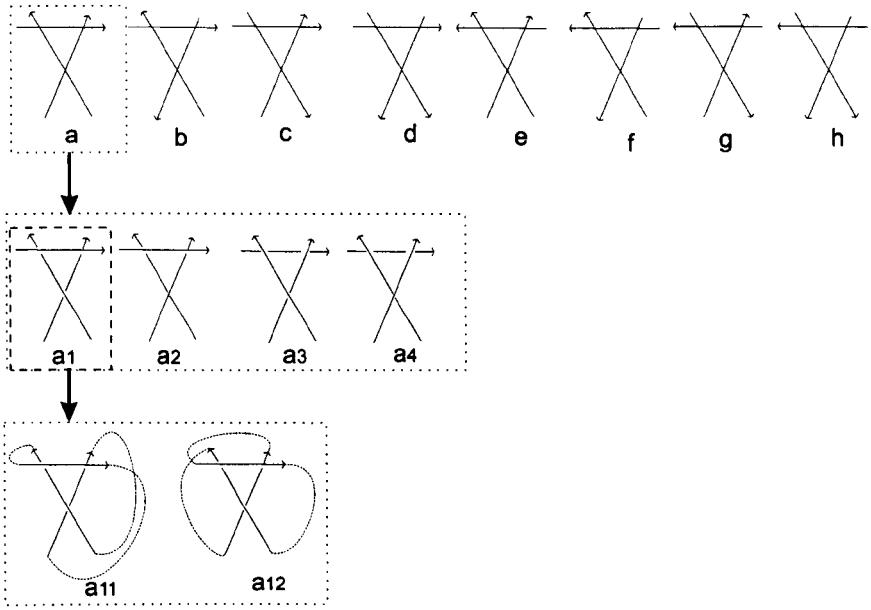

Figure 3.7: - a set of variants of type III

21 $xO_iO_jyU_jO_kzU_iU_k \leftrightarrow xO_jO_kyO_iU_jzU_iU_k$

22 $xO_iO_jyU_iO_kzU_kU_j \leftrightarrow xO_iO_jyO_kU_jzU_iU_k$

23 $xO_iO_jyU_iU_kzU_jO_k \leftrightarrow xO_jO_kyU_iU_kzO_iU_j$

24 $xO_iO_jyU_kU_jzU_iO_k \leftrightarrow xO_iO_jyU_iU_kzO_kU_j$

25 $xO_iO_jyU_jU_kzO_kU_i \leftrightarrow xO_kO_iyU_jU_kzU_iO_j$

26 $xO_iO_jyU_jU_kzO_kU_i \leftrightarrow xO_iO_jyU_kU_izU_jO_k$

27 $xO_iO_jyO_kU_izU_jU_k \leftrightarrow xO_kO_iyU_iO_jzU_jU_k$

28 $xO_iO_jyO_kU_izU_jU_k \leftrightarrow xO_iO_jyU_jO_kzU_kU_i$

29 $xO_iO_jyU_jO_kzU_kU_i \leftrightarrow xO_jO_kyO_iU_jzU_kU_i$

30 $xO_iO_jyU_jO_kzU_kU_i \leftrightarrow xO_iO_jyO_kU_izU_jU_k$

31 $xO_iO_jyU_kU_izU_jO_k \leftrightarrow xO_jO_kyU_kU_izO_iU_j$

32 $xO_iO_jyU_kU_izU_jO_k \leftrightarrow xO_iO_jyU_jU_kzO_kU_i$

33 $xO_iO_jyU_kU_izO_kU_j \leftrightarrow xO_jO_kyU_kU_izU_jO_i$

34 $xO_iO_jyU_iU_kzU_jO_k \leftrightarrow xO_iO_jyU_kU_jzO_kU_i$

35 $xO_iO_jyO_kU_jzU_kU_i \leftrightarrow xO_jO_kyU_jO_izU_kU_i$

36 $xO_iO_jyU_jO_kzU_iU_k \leftrightarrow xO_iO_jyO_kU_izU_kU_j$

37 $xO_iO_jyO_kU_jzU_iU_k \leftrightarrow xO_jO_kyU_jO_izU_iU_k$

38 $xO_iO_jyU_iO_kzU_jU_k \leftrightarrow xO_iO_jyO_kU_jzU_kU_i$

39 $xO_iO_jyU_iU_kzO_kU_j \leftrightarrow xO_jO_kyU_iU_kzU_jO_i$

40 $xO_iO_jyU_jU_kzU_iO_k \leftrightarrow xO_iO_jyU_kU_izO_kU_j$

41 $xO_iO_jyO_kU_jzU_iU_k \leftrightarrow xO_jO_kyU_jO_izU_iU_k$

42 $xO_iO_jyU_iO_kzU_jU_k \leftrightarrow xO_iO_jyO_kU_jzU_kU_i$

43 $xO_iO_jyU_iU_kzO_kU_j \leftrightarrow xO_jO_kyU_iU_kzU_jO_i$

44 $xO_iO_jyU_jU_kzU_iO_k \leftrightarrow xO_iO_jyU_kU_izO_kU_j$

45 $xO_iO_jyU_kU_izO_kU_j \leftrightarrow xO_jO_kyU_kU_izU_jO_i$

46 $xO_iO_jyU_iU_kzU_jO_k \leftrightarrow xO_iO_jyU_kU_jzO_kU_i$

47 $xO_iO_jyO_kU_jzU_kU_i \leftrightarrow xO_jO_kyU_jO_izU_kU_i$

48 $xO_iO_jyU_jO_kzU_iU_k \leftrightarrow xO_iO_jyO_kU_izU_kU_j$

49 $xO_iO_jyU_jO_kzU_kU_i \leftrightarrow xO_jO_kyO_iU_jzU_kU_i$

50 $xO_iO_jyU_jO_kzU_kU_i \leftrightarrow xO_iO_jyO_kU_izU_jU_k$

51 $xO_iO_jyU_kU_izU_jO_k \leftrightarrow xO_jO_kyU_kU_izO_iU_j$

52 $xO_iO_jyU_kU_izU_jO_k \leftrightarrow xO_iO_jyU_jU_kzO_kU_i$

53 $xO_iO_jyU_jU_kzO_kU_i \leftrightarrow xO_kO_iyU_jU_kzU_iO_j$

54 $xO_iO_jyU_jU_kzO_kU_i \leftrightarrow xO_iO_jyU_kU_izU_jO_k$

55 $xO_iO_jyO_kU_izU_jU_k \leftrightarrow xO_kO_iyU_iO_jzU_jU_k$

56 $xO_iO_jyO_kU_izU_jU_k \leftrightarrow xO_iO_jyU_jO_kzU_kU_i$

57 $xO_iO_jyU_jO_kzU_iU_k \leftrightarrow xO_jO_kyO_iU_jzU_iU_k$

58 $xO_iO_jyU_iO_kzU_kU_j \leftrightarrow xO_iO_jyO_kU_jzU_iU_k$

59 $xO_iO_jyU_iU_kzU_jO_k \leftrightarrow xO_jO_kyU_iU_kzO_iU_j$

60 $xO_iO_jyU_kU_jzU_iO_k \leftrightarrow xO_iO_jyU_iU_kzO_kU_j$

61 $xO_iO_jyU_kU_jzO_kU_i \leftrightarrow xO_kO_iyU_kU_jzU_iO_j$

62 $xO_iO_jyU_iU_kzO_kU_j \leftrightarrow xO_iO_jyU_kU_jzU_iO_k$

63 $xO_iO_jyO_kU_izU_kU_j \leftrightarrow xO_kO_iyU_iO_jzU_kU_j$

64 $xO_iO_jyO_kU_jzU_iU_k \leftrightarrow xO_iO_jyU_iO_kzU_kU_j$

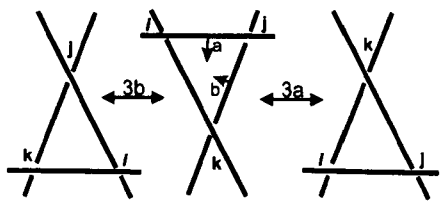

Figure 3.8: Type III - Two types of moves

In Figure 3.8, we show that there are two possible outcomes for application of type III where the first outcome corresponds to sliding the top strand from one side of a crossing to other side (denoted by 3a) and the second outcome corresponds to sliding the bottom strand to the other side of a crossing (denoted by 3b). In terms of rewriting on a Gauss words; we will consider the following example:

Example. *Let* $w = xO_iO_jyU_kU_jzO_kU_i$, w' *denote the word obtained after applying* $3a$ *to* w *and* w'' *be the word obtained after applying* $3b$ *to* w.

$$w = xO_iO_jyU_kU_jzO_kU_i \underset{\longleftarrow}{3a}_{\longrightarrow} w' = xO_iO_jyU_iU_kzU_jO_k$$

$$w = xO_iO_jyU_kU_jzO_kU_i \underset{\longleftarrow}{3b}_{\longrightarrow} w'' = xO_kO_iyU_kU_jzU_iO_j$$

Application of type III results in two outcomes depending on whether the move $3a$ or $3b$ is applied (See Figure 3.8). The effect of this on a word is as follows: If $3a$ is applied to a word w, then the positions of the symbols O_iO_j remain unchanged in

w' and likewise the positions of U_kU_j in w'' (after $3b$ is applied) but the positions of the remaining symbols are altered. We notice that the rearrangement of the remaining symbols depends on whether the symbol $k \in \{U_k, O_k\}$ appears before or after the counterpart symbols of O_iO_j (if $3a$ is applied). Application of $3a$ rearranges the symbols associated with k in the following way: If U_i (or U_j respectively) is a predecessor of k then application of a rule make U_j (or U_i respectively) to become a successor of k. For instance, consider the word $w = xO_iO_jyU_iU_kzO_kU_j$ where U_i is a predecessor of U_k and U_j is a successor of O_k. After application of $3a$ to w, we obtain a new word $w' = xO_iO_jyU_kU_jzU_iO_k$ where U_j becomes a successor of U_k and U_i becomes a predecessor of O_k.

Alternatively the rewriting procedure for type III rules can be described in terms of the interlacement of the symbols involved. Recall in Definition 2.1.4 we defined the notion of interlacement of symbols where two labels i and j are said to be interlaced in w iff the label j appears once between the two labels of i and vice versa, e.g $i, ..., j, ..., i, ...j$

The rewriting procedure can be described as follows: Let w' be the resultant word obtained from w after application of type III.

1. If $3a$ is applied then

   - Fix the positions of O_iO_j and rearrange the remaining symbols in such away that for all ij, i and j interlace in w' iff i and j do not interlace in w.

2. If $3b$ is applied then

   - Fix the positions of U_iU_j and rearrange the remaining symbols in such away that for all ij, i and j interlace in w' iff i and j do not interlace in w.

We notice that the interlacement of symbols in a Gauss word relates to the intersection of chords in a Gauss diagram (see Figure 3.9). That is, for any labels i and j in a Gauss word w, i and j interlace in w iff chord i intersects with chord j in the corresponding Gauss diagram.

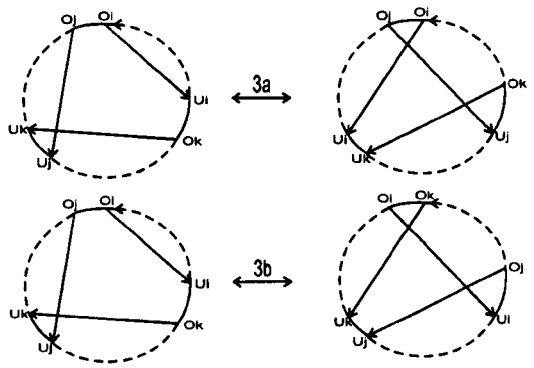

Figure 3.9: Gauss diagram - Interlacement of symbols appear in a Gauss diagram as intersection of chords

Let Ω_3 denote the set of all variants of type III. In the following proposition we will show that Ω_3 is equivalent to the following set of rules.

Proposition 3.2.3. $\Rightarrow^*_{\Omega_3} = \Rightarrow^*_{\{3.1,\ldots,3.8\}}$, *where*

3.1 $xO_iO_jyU_kU_jzU_iO_k \leftrightarrow xO_kO_iyU_kU_jzO_jU_i$

3.2 $xO_iO_jyU_kU_izO_kU_j \leftrightarrow xO_iO_jyU_jU_kzU_iO_k$

3.3 $xO_iO_jyU_iO_kzU_kU_j \leftrightarrow xO_kO_iyO_jU_izU_kU_j$

3.4 $xO_iO_jyO_kU_jzU_kU_i \leftrightarrow xO_iO_jyU_iO_kzU_jU_k$

3.5 $xO_iO_jyO_kU_izU_kU_j \leftrightarrow xO_iO_jyU_jO_kzU_iU_k$

3.6 $xO_iO_jyU_kU_jzO_kU_i \leftrightarrow xO_iO_jyU_iU_kzU_jO_k$

3.7 $xO_iO_jyU_jU_kzO_kU_i \leftrightarrow xO_kO_iyU_jU_kzU_iO_j$

3.8 $xO_iO_jyO_kU_izU_jU_k \leftrightarrow xO_kO_iyU_iO_jzU_jU_k$

Proof. Determine all possible equivalent classes in terms of cyclic shift and renaming of labels. As a result, one will obtain the following:

- $3.1 \equiv 1, 13, 18, 24, 39, 43, 60, 62$
- $3.2 \equiv 2, 7, 11, 14, 33, 40, 44, 45$

- $3.3 \equiv 3, 15, 20, 22, 37, 41, 58, 64$
- $3.4 \equiv 4, 5, 9, 16, 35, 38, 42, 47$
- $3.5 \equiv 6, 10, 19, 21, 36, 48, 57, 63$
- $3.6 \equiv 8, 12, 17, 23, 34, 46, 59, 61$
- $3.7 \equiv 25, 26, 31, 32, 51, 52, 53, 54$
- $3.8 \equiv 27, 28, 29, 30, 49, 50, 55, 56$

□

The claim below follows from Propositions 3.2.1 to 3.2.3.

Claim 1. *Reidmeister moves on a knot diagram can be formalised as the following set of rewriting rules on a Gauss word:*

1.1 $xO_iU_i \leftrightarrow x$

1.2 $xU_iO_i \leftrightarrow x$

2.1 $xO_iO_jyU_iU_j \leftrightarrow xy$

2.2 $xO_iO_jyU_jU_i \leftrightarrow xy$

3.1 $xO_iO_jyU_kU_jzU_iO_k \leftrightarrow xO_kO_iyU_kU_jzO_jU_i$

3.2 $xO_iO_jyU_kU_izO_kU_j \leftrightarrow xO_iO_jyU_jU_kzU_iO_k$

3.3 $xO_iO_jyU_iO_kzU_kU_j \leftrightarrow xO_kO_iyO_jU_izU_kU_j$

3.4 $xO_iO_jyO_kU_jzU_kU_i \leftrightarrow xO_iO_jyU_iO_kzU_jU_k$

3.5 $xO_iO_jyO_kU_izU_kU_j \leftrightarrow xO_iO_jyU_jO_kzU_iU_k$

3.6 $xO_iO_jyU_kU_jzO_kU_i \leftrightarrow xO_iO_jyU_iU_kzU_jO_k$

3.7 $xO_iO_jyU_jU_kzO_kU_i \leftrightarrow xO_kO_iyU_jU_kzU_iO_j$

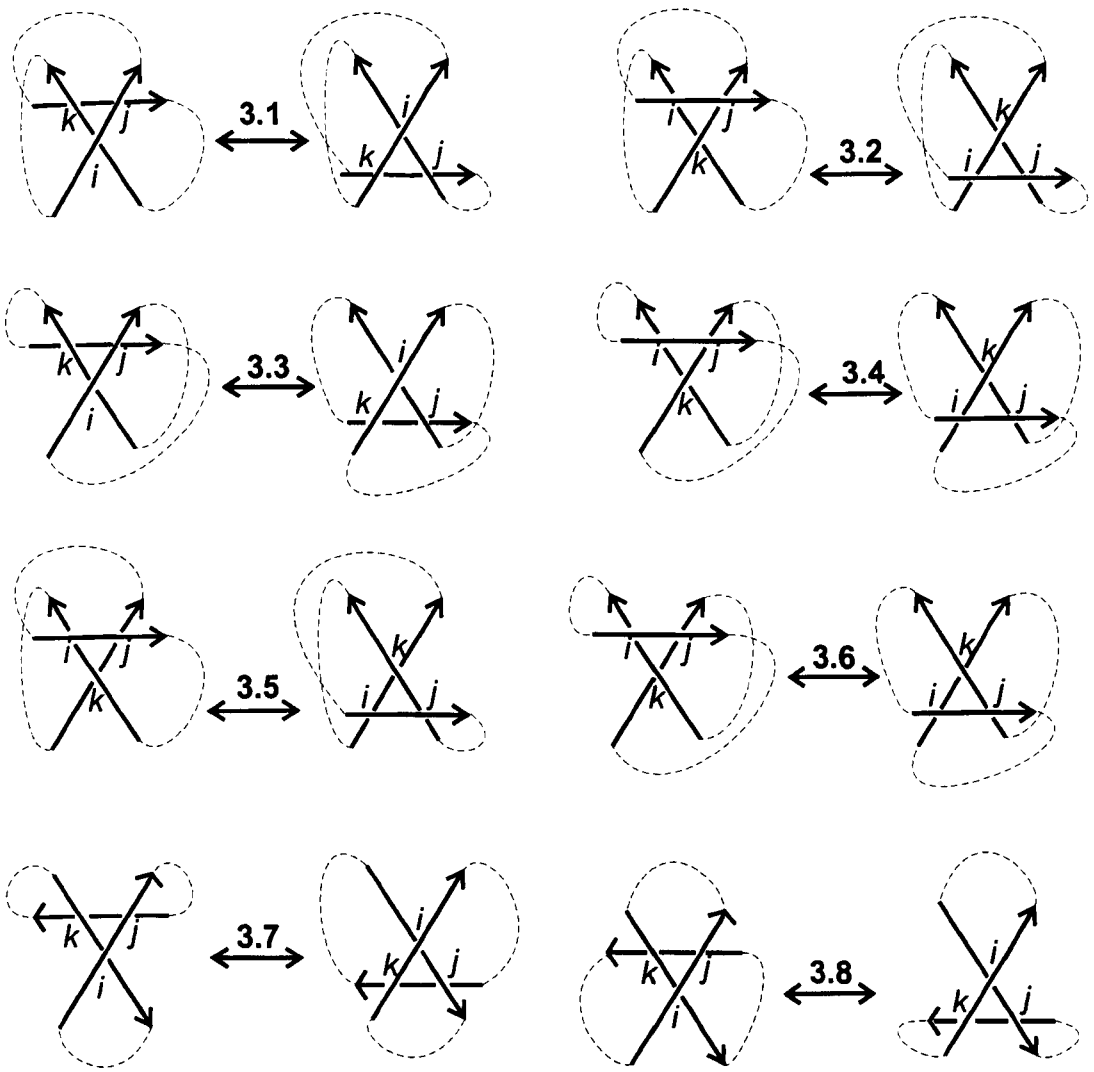

Figure 3.10: Type III - Minimal classes of type III

3.8 $xO_iO_jyO_kU_izU_jU_k \leftrightarrow xO_kO_iyU_iO_jzU_jU_k$

The above claim means that if you take a knot diagram $K1$ and perform some knot transformations in some order to obtain another diagram $K2$ then it can be presented as follows: Take a corresponding Gauss word of $K1$ and apply the above set of rewriting rules in some order to obtain another Gauss word corresponding to $K2$.

3.3 Reachability properties of Reidemeister moves

According to the Reidemeister theorem, if two knot diagrams are equivalent then there exists a finite sequence of moves of types I, II and III that can transform one knot diagram into the other. There has been some research devoted towards the estimation of the length of such sequence and in particular an initial upper bound on the length of the sequence which leads to the unknot was demonstrated to be that 2^{cn} [29] where c=10^{11} and n is the number of crossings and later improved to 2^{cn} where $c = 15^4$ [64]. The only lower bounds presented for the unknotting problem based on the sequence of transformations involving I, II and III were shown to be quadratic with respect to the number of crossings [31; 32]. For the general case of equivalence of knots as well as links a new recent upper bound was shown to be to 2^{c^n} [13] where $c = 10^6$. Furthermore, two knot diagrams are equivalent iff one can be obtained from the other by a sequence of type $I \uparrow$ moves (increase), followed by a sequence of type $II \uparrow$ moves (increase), followed by a sequence of type III moves, followed by sequence of type $II \downarrow$ moves (decrease) [12]. This is denoted by

$$I \uparrow, II \uparrow, III, II \downarrow$$

This result raises some questions about the complexity bounds for the equivalence problem based on the number of transformations, particularly for a subset of rules.

In this section we analyse the reachability properties of Reidemeister moves and provide lower and upper bound on the number of transformations for the equivalence

problem of two knot diagrams reachable by a sequence of moves of the same type. We show linear lower and upper bounds for types I and II, and a quadratic lower bound for type III with respect to the number of crossings while for a subset of rules of type $\{I, II\}$, $\{I, III\}$ and $\{II, III\}$ we provide some classes of structures useful for proving lower bounds.

3.3.1 Basic definitions

For definitions on string rewriting systems we refer the reader to Section 3.5.1.

Because application of types I and II rules can either increase or decrease the size of a Gauss word, we need to distinguish between the two different operations. So applications of type i↑ can increase the size of w whereas applications of type $i\downarrow$ decreases the size of w, where $i \in \{I, II\}$.

For a set of rewriting rules R over a finite set Σ, we define some properties on the relation $\Rightarrow_R$ induced by R. For an illustration see Figure 3.11.

Definition 3.3.1. *A relation $\Rightarrow_R$ is said to be locally confluent iff for all w, w' and w'' in Σ_c^*, if $w \Rightarrow_R w'$ and $w \Rightarrow_R w''$ then there exist w''' in Σ_c^* such that $w' \Rightarrow_R^* w'''$ and $w'' \Rightarrow_R^* w'''$ [3].*

Definition 3.3.2. *A relation $\Rightarrow_R$ is said to be (globally) confluent iff for all w,w'and w" in Σ_c^*, if $w \Rightarrow_R^* w'$ and $w \Rightarrow_R^* w''$ then there exist w''' in Σ_c^* such that $w' \Rightarrow_R^* w'''$ and $w'' \Rightarrow_R^* w'''$. This is also known as Church-Rosser property [3].*

Lemma 3.3.1.1 ([56]). *If a relation $\Rightarrow_R$ is locally confluent and has no infinite rewriting sequences ($w_0 \Rightarrow_R w_1 \Rightarrow_R \cdots \Rightarrow_R w_i \Rightarrow_R \cdots \Rightarrow_R$) then $\Rightarrow_R$ is globally confluent.*

Definition 3.3.3. *Let w be a Gauss word and $R = \{I\downarrow\}, \{II\downarrow\}, \{I\downarrow, II\downarrow\}$, then w is reducible iff there exists a word w' such that $w \Rightarrow_R^* w'$. The word w' is called R-reduct of w (denoted by $reduct_R(w)$) if w' is not reducible by $\Rightarrow_R$ respectively.*

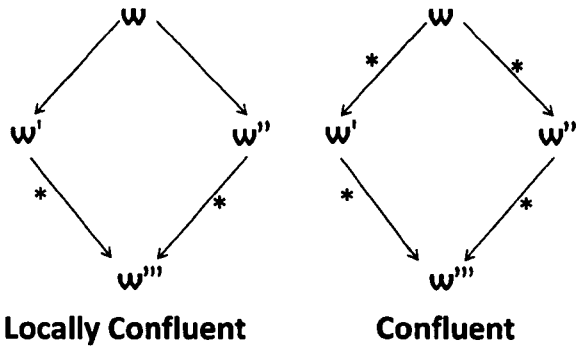

Figure 3.11: - Properties of rewriting systems

3.3.2 Reachability by type I

In this subsection, we investigate the complexity of the path between two equivalent knot diagrams represented by Gauss words which are reachable by a sequence of Reidemeister moves of type I and present lower and upper bounds on the number of transformations for transforming one diagram into the other.

In the following proposition we will state that the relation $\Rightarrow_R$ over Σ where $R = \{I\downarrow\}$ is confluent and then demonstrate an upper bound on the number of transformations between two Gauss words reachable by $\{I\}$.

Proposition 3.3.1. *Let $R = \{I\downarrow\}$, the relation $\Rightarrow_R$ over Σ is confluent.*

Proof. To prove that $\Rightarrow_R$ is confluent, we will need to show that $\Rightarrow_R$ is locally confluent and that all reduction sequences of $\Rightarrow_R$ terminate.

To show that $\Rightarrow_R$ is locally confluent, assume that $w \Rightarrow_R w'$ and $w \Rightarrow_R w''$ for some word w. Let $w = xaybz$ where $a, b \in \{O_iU_i, U_jO_j\}$ for some $i, j \geq 1$. Then $w = xaybz \Rightarrow_R xybz = w'$ and $w = xaybz \Rightarrow_R xybz = w''$. Now we have $w' \Rightarrow_R xyz$ and $w'' \Rightarrow_R xyz$ (see Figure 3.12 for an illustration). Thus local confluence holds for the relation $\Rightarrow_R$. Now it remains to show that all sequences of $\Rightarrow_R$ terminate. Let us consider any sequence $w_1 \Rightarrow_R w_2, \ldots, \Rightarrow_R w_n$, since at each step we decrease the size of resulting word, that is for any two words w_i, w_j in the sequence if $w_i \Rightarrow_R w_j$, then $|w_j| < |w_i|$; so the sequence will terminate after finite number of steps. Therefore by

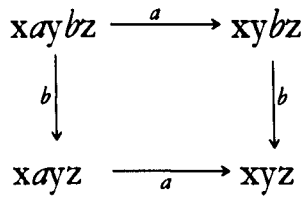

Figure 3.12: - Demonstrating the local confluence property of the relation $\Rightarrow_{\{I\downarrow\}}$

Lemma 3.3.1.1 $\Rightarrow_R$ is a confluent. □

Proposition 3.3.2. *Let* $w, w' \in \Sigma_c^*$ *and* $R = \{I\downarrow\}$, *if* $w \Rightarrow^*_{\{I\}} w'$ *then* $Reduct_R(w) = Reduct_R(w')$

Proof. Suppose that $w \Rightarrow^*_R w'$. Then $w \Rightarrow^*_R Reduct_R(w)$ and $w' \Rightarrow^*_R Reduct_R(w')$. It follows that $w \Rightarrow^*_R Reduct_R(w')$. By Proposition 3.3.1 $Reduct_R(w) = Reduct_R(w')$. □

Corollary 3.3.1. *If* $w \Rightarrow^*_I w'$ *then* $w \Rightarrow^*_{\{I\downarrow\}} Reduct(w') \Rightarrow^*_{\{I\uparrow\}} w'$

In the above Corollary we show that given two Gauss words w and w', if there exists a path reaching w' from w via the sequence I^* then there exists another path from which w' is reachable using the sequence $I\downarrow^*$ and $I\uparrow^*$.

To compute the upper bound we count the number of steps taken to transform w into w' via the sequence $I\downarrow^* I\uparrow^*$.

Proposition 3.3.3. *Given two Gauss words* w *and* w' *where* $|w| = 2n$ *and* $|w'| = 2m$, *if* $w \Rightarrow^*_I w'$ *then the total number of transformations sufficient to rewrite* w *to* w' *is at most* $n + m$

Proof. This is the total number of transformations in the sequence

$$w \Rightarrow_{\{I\downarrow\}} w_i, \ldots, \Rightarrow_{\{I\downarrow\}} Reduct_{\{I\downarrow\}}(w') \Rightarrow_{\{I\uparrow\}} w_j, \ldots, \Rightarrow_{\{I\uparrow\}} w'$$

obtained from Corollary 3.3.2. Since type I can increase or decrease the size of a Gauss by ± 2, then the number of transformations sufficient to reach $Reduct_{\{I\downarrow\}}(w')$ from w is no more than n and no more than m to reach w' from $Reduct_{\{I\downarrow\}}(w')$. □

For computing the lower bound, we consider two classes of diagrams A and B in Figure 3.13. Let w represent A_n and w' represent B_m where $w = U_iO_j, \ldots, O_nU_n, \ldots, U_jO_i$ and $w' = U_iO_i, \ldots, U_mO_m$. For the purpose of proving the lower bound, we associate

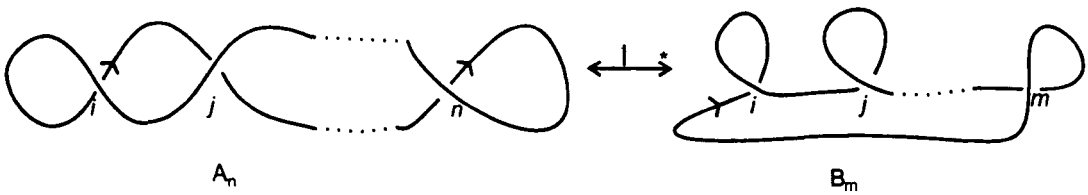

Figure 3.13: - Reachability by Reidemeister move of Type I

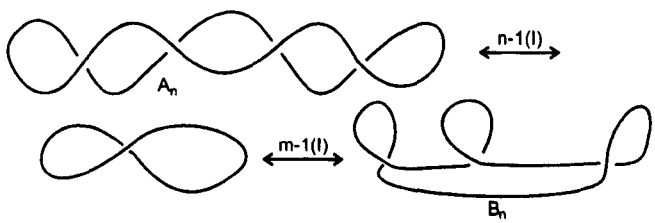

Figure 3.14: - Diagrams A_n and B_m for $n = 4$ and $m = 3$

an integer vector with a Gauss word defined below.

Definition 3.3.4. *Given a Gauss word w, we associate a non-negative integer vector $S(w) = \langle x, y \rangle$ with w where x denote the number of adjacent pairs of OU and UO in w and y denote the number of adjacent pairs of UU and OO in w.*

Example *Given $w = U_1U_2U_3U_4O_4O_3O_2O_1$ and $w' = U_1O_1U_2O_2U_3O_3U_4O_4$ (note that w and w' are assumed to be cyclic words). Let S_1 and S_2 be two vectors associated with w and w' respectively. Then $S_1 = \langle 2, 6 \rangle$ and $S_2 = \langle 8, 0 \rangle$.*

Since the values of a vector are defined in terms of the first component of w (a sequence of Os and Us) then application of type $I \uparrow$ move will correspond to the addition of two symbols of the form UO or OU and type $I \downarrow$ will correspond to the deletion of the symbols UO or OU. In the next proposition we will show how of application of type I can affect the values of a vector $S(w)$.

Proposition 3.3.4. *For Gauss words w and w' the following holds:*

1. *If $w \Rightarrow_{I\uparrow} w'$ then either $S(w') = S(w) + \langle 2, 0\rangle$ or $S(w') = S(w) + \langle 0, 2\rangle$*
2. *If $w \Rightarrow_{I\downarrow} w'$ then either $S(w') = S(w) - \langle 2, 0\rangle$ or $S(w') = S(w) - \langle 0, 2\rangle$*

Proof. Suppose that $w \Rightarrow_{I\uparrow} w'$. The values of $S(w')$ depend on where the symbols UO or OU are inserted in w. There are eight cases below one needs to consider:

- $w = OOx$, $w' = OUOOx$ and $S(w') = S(w) + \langle 2, 0\rangle$.
- $w = UUx$, $w' = UUOUx$ and $S(w') = S(w) + \langle 2, 0\rangle$.
- $w = UOx$, $w' = UUOOx$ and $S(w') = S(w) + \langle 0, 2\rangle$.
- $w = OUx$, $w' = OUOUx$ and $S(w') = S(w) + \langle 2, 0\rangle$.
- $w = OOx$, $w' = OOUOx$ and $S(w') = S(w) + \langle 2, 0\rangle$.
- $w = UUx$, $w' = UOUUx$ and $S(w') = S(w) + \langle 2, 0\rangle$.
- $w = UOx$, $w' = UOUOx$ and $S(w') = S(w) + \langle 2, 0\rangle$.
- $w = OUx$, $w' = OOUUx$ and $S(w') = S(w) + \langle 0, 2\rangle$.

Now suppose that $w \Rightarrow_{I\downarrow} w'$. There are also eight cases to be considered for application of type $I\downarrow$ moves.

- $w = OUOOx$, $w' = OOx$ and $S(w') = S(w) - \langle 2, 0\rangle$.
- $w = UUOUx$, $w' = UUx$ and $S(w') = S(w) - \langle 2, 0\rangle$.
- $w = UUOOx$, $w' = UOx$ and $S(w') = S(w) - \langle 0, 2\rangle$.
- $w = OUOUx$, $w' = OUx$ and $S(w') = S(w) - \langle 2, 0\rangle$.
- $w = OOUOx$, $w' = OOx$ and $S(w') = S(w) - \langle 2, 0\rangle$.
- $w = UOUUx$, $w' = UUx$ and $S(w') = S(w) - \langle 2, 0\rangle$.

- $w = UOUOx$, $w' = UOx$ and $S(w') = S(w) - \langle 2, 0 \rangle$.
- $w = OOUUx$, $w' = OUx$ and $S(w') = S(w) - \langle 0, 2 \rangle$.

□

Next we will show that the number of transformations required to rewrite w to w' is at least linear with respect to the number of crossing labels in w.

Theorem 3.3.1. *Let $w = U_i, \ldots, U_n O_n, \ldots, O_i$ and $w' = U_i O_i, \ldots, U_m O_m$ where $|w| = 2n$ and $|w'| = 2m$, then $w \Rightarrow_I^* w'$ and the total number of transformations required to rewrite w to w' is at least $n+m-2$*

Proof. Let $S(w)$ and $S(w')$ be the vectors associated with w and w' respectively. By Definition 3.3.4. $S(w) = \langle 2, 2(n-1) \rangle$ and $S(w') = \langle 2m, 0 \rangle$. By Proposition 3.3.4 application of type $I \downarrow$ moves to w can only reduce either the value of first component or the value of the second component of $S(w)$ by 2 and application of type $I \uparrow$ moves can only increase either the value of first component or the value of the second component of $S(w)$ by 2. Therefore to transform w to w', we will need to use at least $n-1$ applications of type $I \downarrow$ moves to reduce the value of first component of $S(w)$ from $2(n-1)$ to 0 and at least $m-1$ applications of type $I \uparrow$ moves to increase the value of second component of $S(w)$ from 1 to $2m$. □

3.3.3 Reachability by type II

In this subsection, we investigate the complexity of the path between two equivalent knot diagrams (encoded by Gauss words) which are reachable by a sequence of Reidemeister moves of type II and present some upper and lower bounds based on the number of transformations for transforming one diagram into the other with respect to the number of crossings. In the following proposition we show that the relation $\Rightarrow_{\{II\downarrow\}}$ is confluent and then use this property to derive the reachability path between two Gauss words reachable by a sequence of type II moves.

Proposition 3.3.5. *Let $R = \{II \downarrow\}$, the relation $\Rightarrow_R$ over Σ is confluent.*

Proof. To prove that $\Rightarrow_R$ is confluent, we will need to show that $\Rightarrow_R$ is locally confluent and that all reduction sequences of $\Rightarrow_R$ terminate.

Assume that $w \Rightarrow_R w'$ and $w \Rightarrow_R w''$ for some word w. Let $w = xabycdz$ where $a = O_iO_j$, $b = U_kU_m$, $c = U_iU_j$ or $c = U_jU_i$ and $d = O_kO_m$ or $d = O_mO_k$ for some $i, j, k, m \geq 1$ and $i < j < k < m$. Then $w = xabycdz \Rightarrow_R xbydz = w'$ and $w = xabycdz \Rightarrow_R xaycz = w''$. Now we have $w' \Rightarrow_R xyz$ and $w'' \Rightarrow_R xyz$. Thus local confluence holds.

To show that $\Rightarrow_R$ terminate. Let us consider a sequence $w_1 \Rightarrow_R w_2, \ldots, \Rightarrow_R w_n$, notice that is for any two words w_i, w_j in the sequence if $w_i \Rightarrow_R w_j$, then $|w_j| < |w_i|$; so the sequence does terminate after finite number of steps. Therefore by Lemma 3.3.1.1 $\Rightarrow_R$ is a confluent. □

Proposition 3.3.6. *Let $w, w' \in \Sigma_c^*$ and $R = \{II \downarrow\}$, if $w \Rightarrow^*_{\{II\}} w'$ then $Reduct_R(w) = Reduct_R(w')$*

Proof. The proof uses an argument similar to the argument in the proof of Proposition 3.3.2. □

Corollary 3.3.2. *If $w \Rightarrow^*_{II} w'$ then $w \Rightarrow^*_{\{II\downarrow\}} Reduct(w') \Rightarrow^*_{\{II\uparrow\}} w'$*

In the above Corollary we show that given two Gauss words w and w', if there exists a path reaching w' from w via the sequence II^* then there exists another path from which w' is reachable without exceeding the number of crossings using the sequence II↓*II↑*

To compute the upper bound we count the number of steps taken to transform w into w' via the sequence $II \downarrow^* II \uparrow^*$.

Proposition 3.3.7. *Given two Gauss words w and w' where $|w| = 2n$ and $|w'| = 2m$, if $w \Rightarrow^*_{II} w'$ then the total number of transformations sufficient to rewrite w to w' is at most $\frac{n+m}{2}$.*

Proof. This is the total number of transformations in the sequence

$$w \Rightarrow_{\{I\downarrow\}} w_i, \ldots, \Rightarrow_{\{I\downarrow\}} Reduct(w') \Rightarrow_{\{I\uparrow\}} w_j, \ldots, \Rightarrow_{\{I\uparrow\}} w'$$

obtained from Corollary 3.3.2. □

For computing the lower bound, we consider two classes of diagrams A and B illustrated in Figure 3.15. Let w represents diagram A_n and w' represents B_n where $w = U_i, \ldots, U_n O_i \ldots, O_n$ and $w' = U_i, \ldots, U_m O_m, \ldots, O_i$ such that $n = m \equiv 0 \bmod 1$.

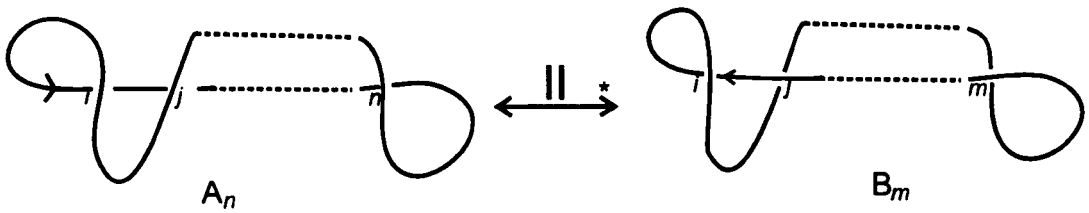

Figure 3.15: - Reachability by Reidemeister move of Type II

We consider second component of a Gauss word only (a sequence of natural numbers for the crossing labels referred to as a shadow word) and associate an interlacement graph with a shadow word in the following way: Create a node for each label $i \in \{1 \ldots n\}$ in w and connect two nodes i, j by an edge iff i occurs only once between the two appearances of j in w. For an example see Figure 2.6 in Chapter 2.

To estimate the lower bound, we define a vector based on the number of edges and nodes of the interlacment graph associated with a shadow word.

Definition 3.3.5. *Let w be a Gauss word and $G(w)$ be an interlacement graph associated with w, then $S(G_w) = \langle x, y \rangle$ is a vector associated with $G(w)$ where x denotes the number of nodes of $G(w)$ and y denotes the number of edges of $G(w)$.*

Proposition 3.3.8. *For Gauss words w and w', the following hold:*

1. *If $w \Rightarrow_{II\uparrow} w'$ then $S(G_{w'}) = S(G_w) + \langle 2, y \rangle$ for $y = 0, \ldots, 2n+1$*
2. *If $w \Rightarrow_{II\downarrow} w'$ then $S(G_{w'}) = S(G_w) - \langle 2, y \rangle$ for $y = 0, \ldots, 2n-3$*

Proof. Suppose that $w \Rightarrow_{I\uparrow} w'$. Then application of type $I\uparrow$ move to w can increase the number of nodes of $S(G_{w'})$ by 2 and the number of edges of $S(G_{w'})$ can change depending on where the labels ij and ji or ij and ij are inserted in w. So we have the following cases to consider:

- $w = 1 \cdots n1 \cdots n$ $w' = ij1 \cdots n1 \cdots nji$ and $S(G_{w'}) = S(G_w) + \langle 2, 0 \rangle$.
- $w = 1 \cdots n1 \cdots k \cdots n$ $w' = ij1 \cdots kij \cdots n1 \cdots k \cdots n$ and $S(G_{w'}) = S(G_w) + \langle 2, 2k+1 \rangle$.
- $w = 1 \cdots n1 \cdots n$ $w' = ij1 \cdots nij1 \cdots n$ and $S(G_{w'}) = S(G_w) + \langle 2, 2n+1 \rangle$.

Now suppose that $w \Rightarrow_{I\downarrow} w'$. Then the number of nodes can decrease by 2 and the number of edges can decrease as follows:

- $w = ij1 \cdots n1 \cdots nji$ $w' = 1 \cdots n1 \cdots n$ and $S(G_{w'}) = S(G_w) - \langle 2, 0 \rangle$.
- $w = ij1 \cdots kij \cdots n1 \cdots k \cdots n$ $w' = 1 \cdots n1 \cdots k \cdots n$ and $S(G_{w'}) = S(G_w) + \langle 2, 2k-3 \rangle$.
- $w = ij1 \cdots nij1 \cdots n$ $w' = 1 \cdots n1 \cdots n$ and $S(G_{w'}) = S(G_w) + \langle 2, 2n-3 \rangle$.

□

Theorem 3.3.2. *Let $w = U_i \cdots U_n O_i \cdots O_n$ and $w' = U_i \cdots U_n O_n \cdots O_i$ where $n, m \equiv 1 \bmod 2$, then $w \Rightarrow_{II}^{*} w'$ and the total number of transformations of type II required to rewrite w to w' is at least $\frac{n+m}{2} - 1$*

Proof. Let $S(G_w)$ and $S(G_{w'})$ be the vectors associated with w and w' respectively as defined in Definition 3.3.5. Then $S(G_w) = \langle n, \frac{n(n-1)}{2} \rangle$ and $S(G_{w'}) = \langle m, 0 \rangle$. By Proposition 3.3.8 application of type $II \downarrow$ moves to w can reduce either the number of nodes in $S(G_w)$ by 2 or the number of nodes by 2 and the number of edges by at most $2n-3$ while application of type $II \uparrow$ moves can increase either the number of nodes by 2 or the number of nodes by 2 and the number of edges by at most $2n+1$.

To calculate the minimal number of steps required to reduce number of edges in $S(G_w)$ from $\frac{n(n-1)}{2}$ to 0, we do the following computations. Let k denote the number of steps where for each step $i = 0, \ldots, k-1$, the number of nodes is reduced by $n-2i$ and the number of edges is reduced by a maximal number $2(n-2i)-3$. Then we have the following equation.

$$\frac{n(n-1)}{2} = 2(n-0) - 3 - 2(n-2) - 3 \ldots - 2(n-(k-1)) - 3$$

To find the minimal number of such steps we compute k by rewriting the equation in a closed form and using the quadratic formula to solve it.

$$\frac{n(n-1)}{2} = 2(n-0) - 3 - 2(n-2) - 3 \ldots - 2(n-(k-1)) - 3$$

$$\frac{n(n-1)}{2} = k(2n-3) - 2k(k-1)$$

$$n(n-1) = 2(k(2n-3) - 2k(k-1))$$

$$n(n-1) = (4n-2)k - 4k^2$$

$$4k^2 - (4n-2)k + n(n-1) = 0$$

$$k = \frac{4n-2-\sqrt{(-4n-2)^2 - 4 \times 4 \times (n^2-n)}}{2 \times 4} = \frac{n-1}{2}$$

$$k = \frac{4n-2+\sqrt{(-4n-2)^2 - 4 \times 4 \times (n^2-n)}}{2 \times 4} = \frac{n}{2}$$

Next we will show that no matter how type $II\downarrow$ is applied we still need to have at least $\frac{n-1}{2}$ applications. To do this we define some local property to demonstrate that applications of $II\downarrow II\uparrow$ is no better than applications of $II\uparrow II\downarrow$.

Let $II\downarrow^{max}$ denote the maximal number of edges that can be removed by $II\downarrow$ and let $w \Rightarrow_{II\downarrow^{max}} w'$ and $w \Rightarrow_{II\downarrow} w''$ then the value of y' in $S(G_{w'})$ is less or equal to the value of y'' in $S(G_{w''})$ and similarly if $w \Rightarrow_{II\downarrow^{max}} w' \Rightarrow_{II\uparrow} w''$ and $w \Rightarrow_{II\uparrow} w''' \Rightarrow_{II\downarrow^{max}} w''''$ then the value of y'' in $S(G_{w''})$ is less or equal to the value of y'''' in $S(G_{w''''})$. Now we can use this property globally to rearrange the sequence of applications of types II by sorting all applications of type $II\downarrow$ followed by all applications of type $II\uparrow$.

So far we have computed the minimal number of applications of type $II\downarrow$ moves required to reduce the number of edges of $S(G_w)$ from $\frac{n(n-1)}{2}$ to 0. Now it remains to compute the number of applications of type $II\uparrow$ moves needed to increase the number of nodes of $S(G_w)$ from 1 to m. Let $l \geq \frac{(n-1)}{2}$ denote the number of applications of type $II\downarrow$, and p denote the number of applications of type $II\uparrow$, to compute p we need

to solve the following equation:

$$\begin{aligned} n + 2p - 2l &= m \\ n + 2p - 2(\frac{n-1}{2}) &\geq m \\ 2p + 1 &\geq m \\ p &\geq \frac{m-1}{2} \end{aligned}$$

Therefore to transform w to w', we need to use at least $\frac{n-1}{2}$ applications of type $I \downarrow$ moves to reduce the number of edges of $S(G_w)$ from $\frac{n(n-1)}{2}$ to 0 and at least $\frac{m-1}{2}$ applications of type $I \uparrow$ moves to increase the number of nodes of $S(G_w)$ from 1 to m. □

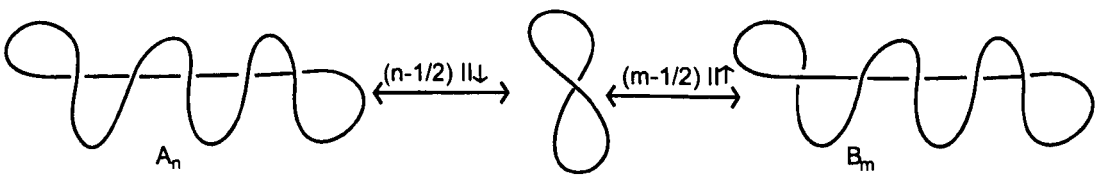

Figure 3.16: - Diagrams A_n and B_m for $n = 5$ and $m = 5$

3.3.4 Reachability by type III

In this subsection, we investigate the complexity of the path between two equivalent knot diagrams which are reachable by a sequence of Reidemeister moves of type III and present a quadratic lower bound on the number of transformation needed to transform one diagram into the other with respect to the number of crossings. The complexity of Reidemeister moves of type III has been studied in [10] where some constant lower bound for the number of moves of type III was presented by using extended n-colorings of knot diagrams in the plane.

As to the upper bound, one can obtain a trivial exponential upper bound with respect to the number of symbols of a Gauss word by computing its permutation. This because transformations involving type III does neither increase of decrease the number

of crossings. However, a precise upper bound is still to be studied.

It is a known fact in knot theory that the *writhe* is a signed knot diagram invariant [14] under Reidemeister moves of types II and III. The writhe is defined as the sum of signs of all crossings in a signed knot diagram. The value assigned to each crossing is 1 if a crossing is positive and -1 if a crossing is negative. So from this fact we derive the following proposition:

Proposition 3.3.9. *[14] Let w and w' be signed Gauss words representing two knot diagrams and $Wr(w)$ denote the writhe of a Gauss word w, if $w \Rightarrow^{*}_{III} w'$ then $Wr(w) = Wr(w')$*

The fact that the converse does not hold can be shown by providing two knot diagrams that are not reachable from each other by type III but have the same *writhe*. This is illustrated in Figure 3.17. It is easy to see that type III is not applicable to any of the diagrams.

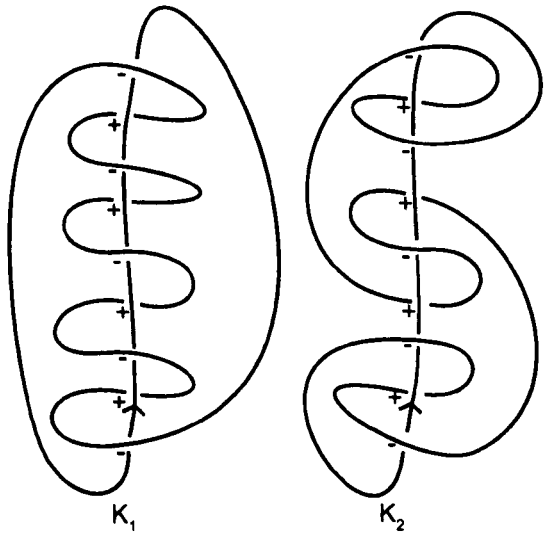

Figure 3.17: - Two different knots with the same writhe ($Wr(K_1) = Wr(K_2) = -1$)

To estimate the lower bounds for type III, we consider two classes A and B of knot diagrams where for any knot diagram $A_k \in A$, the number of crossings of A_k and B_k is $3k$ for some $k \geq 1$ such that A_k and B_k are constructed in the same form as in Figure

3.18.

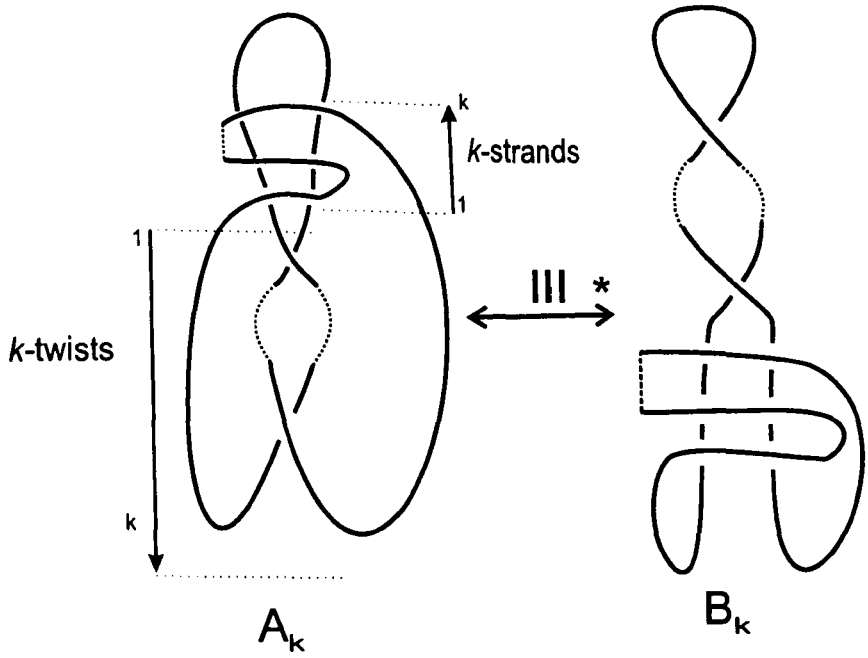

Figure 3.18: - Diagrams A_k and B_k with $3k$ crossings

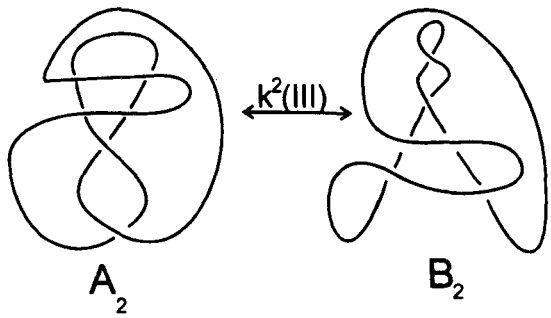

Figure 3.19: - Diagrams A_k and B_k for $k = 2$

Proposition 3.3.10. *Given two knot diagrams A_k and B_k with n crossings where $n = 3k$ and $k \geq 1$, if B_k is reachable from A_k by a sequence of Reidemeister moves of type III then the number of moves required to transform A_k to B_k is at least $(\frac{n}{3})^2$*

Proof. A_k consists of k twists and k horizontal strands. Application of Type III moves to diagram A_k is limited as it can only be applied to the k-strands by moving them

down (or down and then up) through the crossings corresponding to the k twists. To have the minimal steps we need to move the k-strands down through each twist. Since application of type III corresponds to moving one strand through a single crossing, then we would require at least k^2 steps to move k strands through k crossings. Hence $k^2 = (\frac{n}{3})^2$ □

3.3.5 Combination of Reidemeister moves types

In this section we analyse the reachability properties of Reidemeister moves for a combination set of rules and present linear upper bounds on the number of moves of type $\{I, II\}$ and quadratic lower bounds on the number of moves of type $\{I, III\}$ to reach one diagram from the other. In addition we provide some plausible classes of structures which can be used to prove lower bounds for types $\{I, II\}$ and $\{II, III\}$.

3.3.5.1 Reachability by types I and II

In this subsection, we investigate the complexity of the path between two equivalent knot diagrams which are reachable by a sequence of Reidemeister moves of type $\{I, II\}$ and demonstrate a linear upper on the number of transformations between two Gauss words reachable by $\{I, II\}$. Then we present two classes of structures which can be used to prove the lower bound and we expect that the number of moves of types $\{I, II\}$ required to transform one diagram into the other is linear with respect to the number of crossings.

In the following proposition we will show that $\Rightarrow_R$ is confluent where $R = \{I\downarrow, II\downarrow\}$ and then use this property to derive the reachability path between two Gauss words reachable by a sequence of type $\{I, II\}$.

Proposition 3.3.11. *Let $R = \{I\downarrow, II\downarrow\}$, the relation $\Rightarrow_R$ over Σ is confluent.*

Proof. Assume that $w \Rightarrow_R w'$ and $w \Rightarrow_R w''$ for some word w. There are three cases to be considered for R; $R = \{I\downarrow\}$, $R = \{II\downarrow\}$ and $R = \{I\downarrow, II\downarrow\}$. By Proposition 3.3.1 the relation $\Rightarrow_{\{I\downarrow\}}$ is confluent and by Proposition 3.3.5 the relation

$\Rightarrow_{\{II\downarrow\}}$ is confluent. Now it remains to show for $R = \{I\downarrow, II\downarrow\}$, the relation $\Rightarrow_R$ is confluent. Let $w = xabycz$ where $a = O_iO_j$, $c = U_iU_j$ or $c = U_jU_i$ and $b = O_kU_k$ or $b = U_kO_k$ for some $i, j, k \geq 1$ and $i < j < k$. Then $w = xabycz \Rightarrow_R xaycz = w'$ and $w = xabycz \Rightarrow_R xbyz = w''$. Now we have $w' \Rightarrow_R xyz$ and $w'' \Rightarrow_R xyz$. Thus local confluence holds.

To show that $\Rightarrow_R$ terminate. Let us consider a sequence $w_1 \Rightarrow_R w_2, \ldots, \Rightarrow_R w_n$, notice that is for any two words w_i, w_j in the sequence if $w_i \Rightarrow_R w_j$, then $|w_j| < |w_i|$; so the sequence does terminate after finite number of steps. Therefore by Lemma 3.3.1.1 $\Rightarrow_R$ is a confluent. □

Proposition 3.3.12. *Let $w, w' \in \Sigma_c^*$ and $R = \{I\downarrow, II\downarrow\}$, if $w \Rightarrow^*_{\{I,II\}} w'$ then $Reduct_R(w) = Reduct_R(w')$*

Proof. The proof uses the same argument as in the proof of Proposition 3.3.2. □

Corollary 3.3.3. *If $w \Rightarrow^*_{\{I,II\}} w'$ then $w \Rightarrow^*_{\{I\downarrow,II\downarrow\}} Reduct(w') \Rightarrow^*_{\{I\uparrow,II\uparrow\}} w'$*

To compute the upper bound we count the number of steps taken to transform w into w' via the sequence $\Rightarrow^*_{\{I\downarrow,II\downarrow\}}, \Rightarrow^*_{\{I\uparrow,II\uparrow\}}$.

Proposition 3.3.13. *Given two Gauss words w and w' where $|w| = 2n$ and $|w'| = 2m$, if $w \Rightarrow^*_{I,II} w'$ then the number of moves sufficient to rewrite w as w' is at most $n+m-1$.*

Proof. This is the total number of transformations in the sequence

$$w \Rightarrow_{\{I\downarrow,II\downarrow\}} w_i, \ldots, \Rightarrow_{\{I\downarrow,II\downarrow\}} Reduct_R(w') \Rightarrow_{\{I\uparrow,II\uparrow\}} w_j, \ldots, \Rightarrow_{\{I\uparrow,II\uparrow\}} w'$$

obtained from Corollary 3.3.3. Here, we assume that type II is applied once. □

For computing the lower bound, we consider two classes of diagrams A and B (where B can be seen as mirror images of diagrams in A) for which Reidemeister moves of type I and II are applicable. The sequence of diagrams of A and B follow the form illustrated in Figure 3.20 where both A_k and B_k have $3k$ crossings for some $k \geq 1$. For an example where $k = 2$, see Figure 3.21.

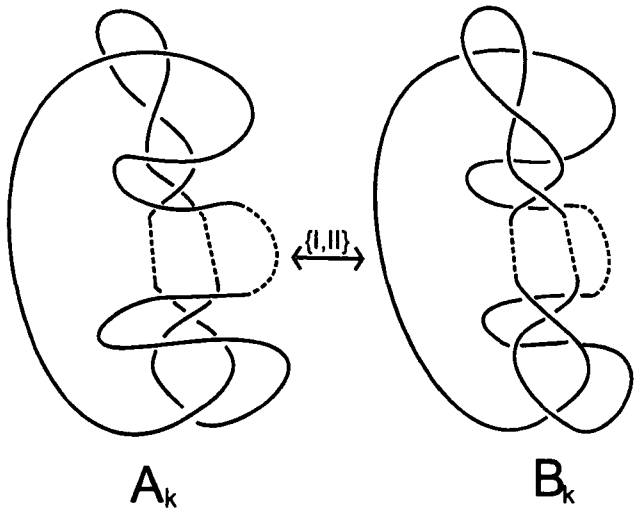

Figure 3.20: - An instance of two knot diagrams A_k and B_k with $3k$-crossings reachable by rules of type I and II

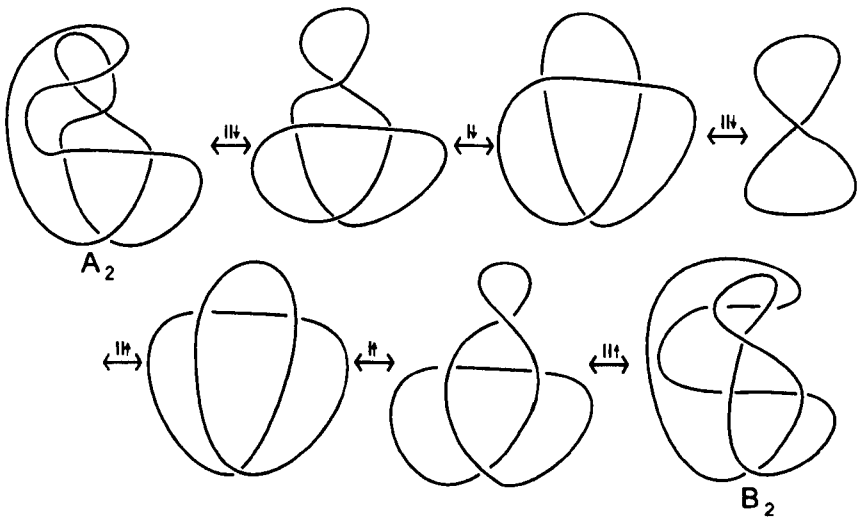

Figure 3.21: - Transformation of A_k into B_k by types I and II where $k = 2$

Conjecture 3.3.1. *Given two knot diagrams $A_k \in A$ and $B_k \in B$ with n-crossings where $n = 3k$ for some $k \geq 1$, if B_k is reachable from A_k by a sequence of Reidemeister moves of types $\{I, II\}$ then the number of moves required to transform A_k to B_k is at least $\frac{4(n+m)}{3} - 2$.*

Comment. Notice that A_k is a mirror image of B_k obtained by inverting the crossings of A_k from over-crossing to under-crossing and vice versa. We believe that the minimal way is to reduce A_k and B_k is to a common diagram. So the only common diagram reachable from A_k and B_k by Reidemeister moves of types $II \downarrow$ and $I \downarrow$ is a diagram with one crossing. Since A_k has $3k$ crossings we will require at least k applications of type $II \downarrow$ moves and $k-1$-applications of type $I \downarrow$ moves to reduce the number of crossing A_k to 1 and then to reach B_k will need at least k applications of type $II \uparrow$ and $k-1$ applications of type $I \uparrow$ to increase the number of crossings from 1 to $3k$. So in total the number of steps required is at least $2(k-1) + 2k = 4k - 2$. Now substituting k by $\frac{n}{3}$, we have $\frac{4(n+m)}{3} - 2$.

Essentially to make it a full proof we need to show that any other way will lead to a longer sequence. We believe it's true but we don't have the full proof at the moment.

3.3.5.2 Reachability by types I and III

In this subsection we present two classes of diagrams A and B illustrated in Figure 3.22 such that for any diagram $A_k \in A$, A_k has $3k$ crossings and any diagram $B_k \in B$ B_k has $2k$ crossings for $k \geq 1$. Using the proposed classes of diagrams we will show that the number Reidemeister moves of types I and III required to transform A_k into B_k is at least quadratic with respect to the number of crossings.

Proposition 3.3.14. *Given a knot diagram $A_k \in A$ and $B_k \in B$ where $k \geq 1$, if B_k is reachable from A_k by a sequence of Reidemeister moves of types $\{I, III\}$ then the number of moves required to transform A_k to B_k is at least $k^2 + k$*

Proof. To transform A_k to B_k, we need to undo all twists in A_k to reduce the number of crossings from $3k$ to $2k$. Application of type III move does not affect the number of

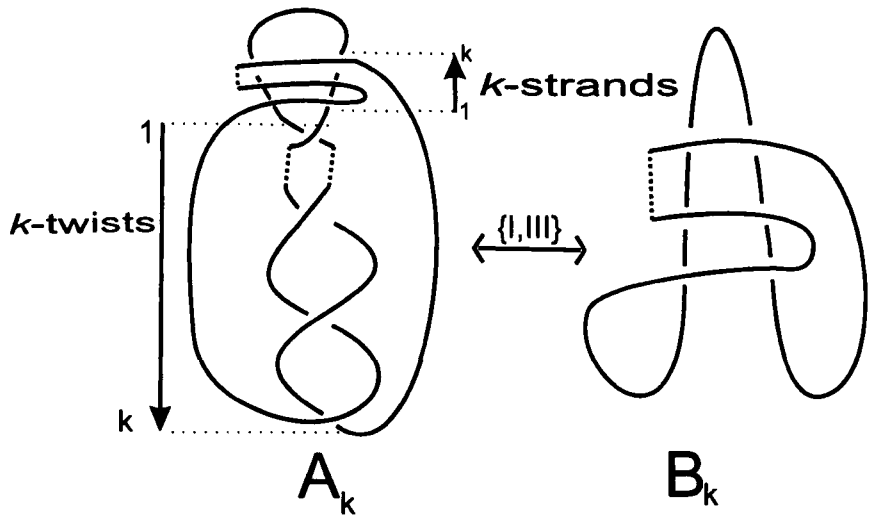

Figure 3.22: - Diagram A_k with $3k$ crossings and Diagram B_k with $2k$

crossings and application of type I move can either increase of decrease the number of crossings by 1.

Notice that any sequence involving type I↑ moves does not help in reducing applications of type III to move the k-strands down through the crossings and allow us undo the k- twists in A_k. This is because application of type I↑ moves only creates new loops on a single strand and as a result of this it will increase the number of crossings in which in turn increases the number of transformations.

Therefore the shortest way to reduce the number of crossings in A_k is to apply type $I\downarrow$. The only way to make type $I\downarrow$ applicable is by moving the k-strands down through the crossing. Since there are k-twists and k-horizontal strands, to move the k-strands down through each crossing, we would require at least k^2 moves of type III and k moves of type $I\downarrow$ to undo the twists. □

3.3.5.3 Reachability by types II and III

In this section, we consider a class of diagrams presented in [31] which were used to demonstrate lower bound on the number of Reidemeister moves of types $\{I, II, III\}$ for the unknottedness problem. We modify such class of diagrams as depicted in Figure

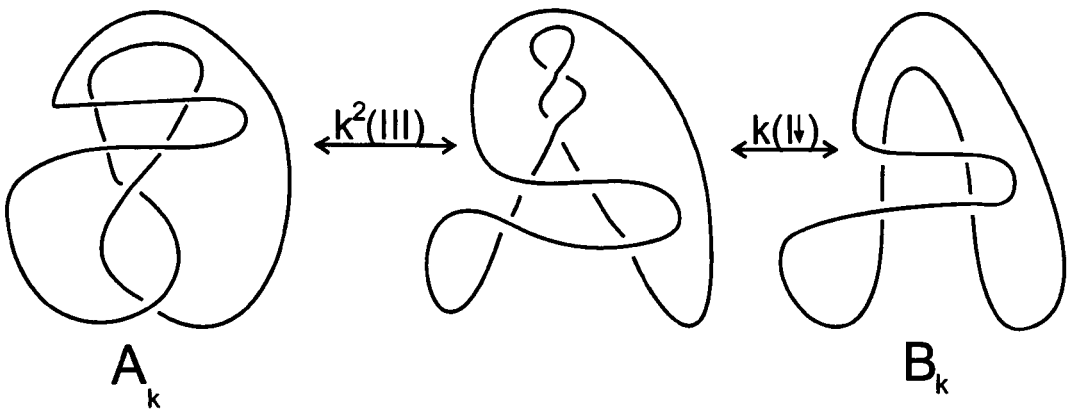

Figure 3.23: - Transformation of A_k into B_k by types I and III where $k = 2$

3.24 to make it more complex and propose to use it for proving the lower bound on the number of Reidemeister moves of types II and III required for the equivalence of two knot problems. We expect that the number of moves is quadratic with respect to the number of crossings. Let A and B be two classes of diagrams such that for any diagram $A_k \in A$, the number of crossings of A_k is $19k - 5$ and for any diagram $B_k \in B$ the number of crossings of B_k is $k + 3$ for some $k \geq 1$ and A_k has the same form as Figure 3.24 and B_k has the same form as Figure 3.25.

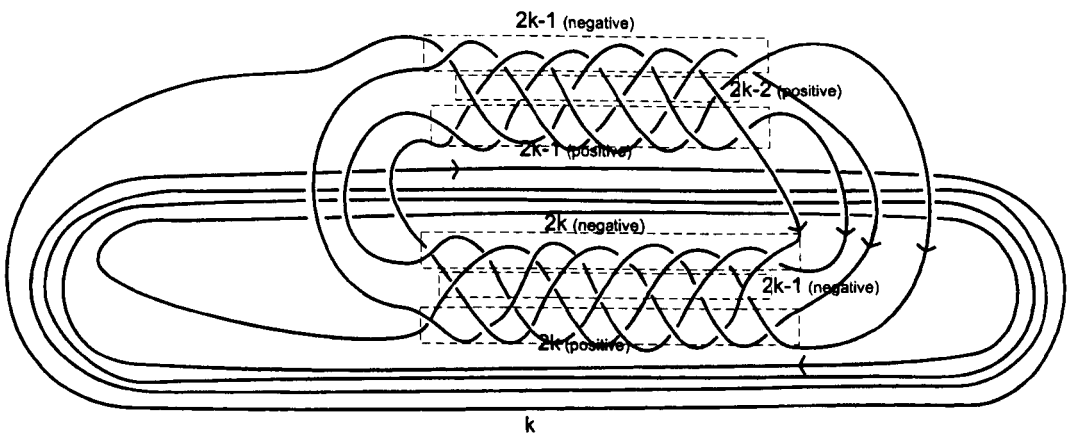

Figure 3.24: - Diagram A_k for $k = 4$

Conjecture 3.3.2. *Given two knot diagrams A_k and B_k where n=19k-5 $k \geq 1$, if B_k*

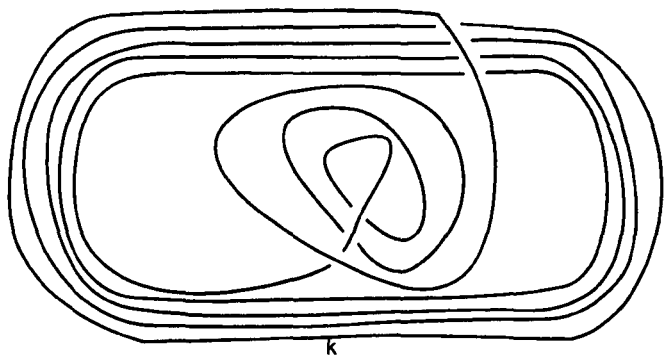

Figure 3.25: - Diagram B_k for $k = 4$

is reachable from A_k by a sequence of Reidemeister moves of types $\{II, II\}$ then the number of moves required to transform A_k to B_k is at least $6k^2 + 4k$

Comment. To transform A_k to B_k, we need reduce the number of crossings of A_k from $19k - 5$ to $k + 3$. Applications of type III does not affect the number of crossings whereas type II can affect the number of crossings by ± 2. Before type $II \downarrow$ move can applied we need to apply type III move a number of times, so we expect that at least $k(6k - 4)$ moves of type III and $8k$ moves of type $II \downarrow$ are needed. For an example consider Figure 3.26 for the transformations of A_4 to B_4. We believe this is the minimal way because type $II \downarrow$ is applied here while applications of type $II \uparrow$ moves will only make it worse.

3.4 Non-isomorphic knot diagrams generated by applications of type I

In this section we demonstrate the connection between knot diagrams generated by Reidemeister moves of type I and unrooted Eulerian n-edge maps. Let us consider oriented knot diagrams in the plane represented by Gauss diagrams. Starting from an unknot (a diagram with no crossings) we apply type I moves to generate all possible diagrams. We are interested in the question of how many non-isomorphic diagrams

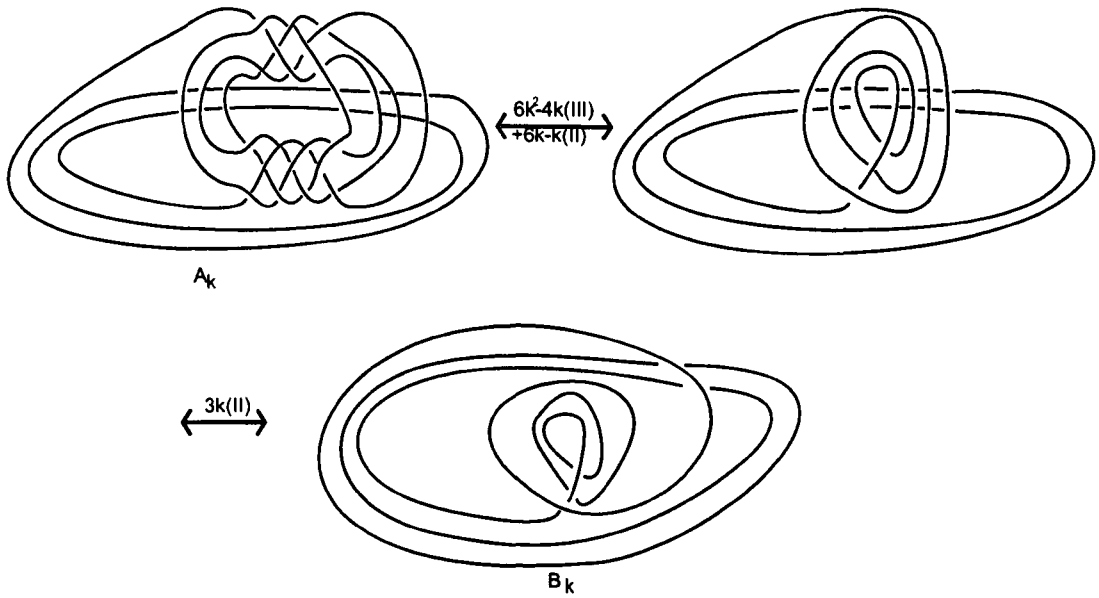

Figure 3.26: - Transformation of A_k into B_k by types II and III where $k = 2$

can be obtained from the unknot. To conduct this experimentation we developed a Java program and discovered that the sequence of non-isomorphic knot diagrams with n-crossings we obtained matches the sequence of unrooted Eulerian n-edge maps in the plane with a distinguished outer face (see the On-Line Encyclopedia of Integer Sequences [63]). Further we were looking for the bijective map between the two classes of objects and as a result we found two explicit algorithms to describe the transformations from Gauss diagrams to maps and vice versa.

A 2-cell embedding is an embedding in which every face is homeomorphic to an open disk (i.e. if each of the faces is a simply connected region).

Definition 3.4.1. *[43] A map is a 2-cell embedding of undirected connected graph, loops and parallel edges allowed, on an unbounded surface. If the surface is a sphere then the map is a planar map; if the surface is an infinite plane then the map is a plane map and one of its faces is distinguished as an outside face.*

Definition 3.4.2. *[43] A homeomorphism between two maps on orientable surfaces is a bicontinuous bijection between their embedding surfaces that takes the vertices, edges and faces of one map into the vertices, edges and faces of the other; in the case of plane*

maps, it also takes the outside face of one map into the outside face of the other.

An unrooted map is an equivalence class of maps under orientation-preserving homeomorphism.

Definition 3.4.3. *A Gauss diagram is an oriented circle with oriented chords. The orientation of the circle corresponds to the orientation of the knot diagram and the orientation of a chord denotes over-crossing to under-crossing passes. The region inside the circle will be called an inner face and the region in the outside of the circle will be called an outer face.*

3.4.1 Gauss diagrams to maps

In this subsection we will describe the first part of the correspondence by presenting an algorithm for the transformation of a Gauss diagram to a map. There are two main parts of the algorithm. The first part is to construct a set of cycles by adding new edges from outside the circle and the second part is to join all cycles together to make the corresponding map.

Let U_i denote a point in the Gauss diagram with an incoming chord and O_i denote a point with an outgoing chord where i denotes the label of the chord.

First we add a new edge from the outer face to connect two points on the circle if either there are two consecutive (in counter-clock wise direction) points on the circle of the form U_iO_i and U_iO_j or $U_i \cdot S \cdot O_i$ and $U_i \cdot S \cdot O_j$ where S denote a set of connected points that separates U_i from O_i or U_i from O_j. Then we join two cycles together if there are two consecutive points of the form U_iU_j or O_iU_j such that the first point belong to the first cycle and the second points belong to the second cycle. Further we merge two points that belong to the same cycle if there are two consecutive points of form U_iO_i or O_iU_i. Finally we merge the points of any two cycles that were joined by an edge from the inner face.

At every step of this algorithm there are many choices of how connections are created. However the outcome of any choice is equivalent up to orientation-preserving homeomorphism as defined in Definition 3.4.2. This means the algorithm will produce

a unique outcome up to this equivalence.

3.4.1.1 Algorithm 1

1. For all non-connected points U followed by O in a Gauss diagram

   (a) Connect in anti-clock wise direction from an outer face a non-connected U with the nearest non-connected O separated by connected vertices only.

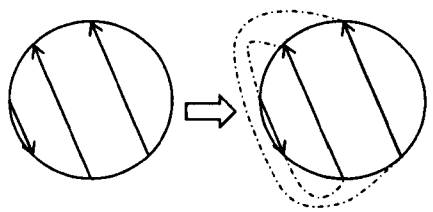

Figure 3.27: Step 1 - Connecting new edges between points on the circle from the outer face

2. For any two consecutive points (in counter-clock wise direction) of the form U_iU_j or O_iU_j, if the first point belong to cycle C_i and the second point belong to cycle C_j then create a new edge from the inner face to join the two points together.

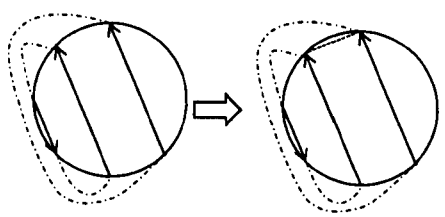

Figure 3.28: Step 2 - Connecting new edges between cycles from the inner face

3. For each cycle, if there are two adjacent points of form U_iO_i or O_iU_i then merge the two points into one.

4. For any two cycles C_i and C_j, if C_i is connected to C_j then remove the edge joining the points of C_i and C_j and glue the two points together.

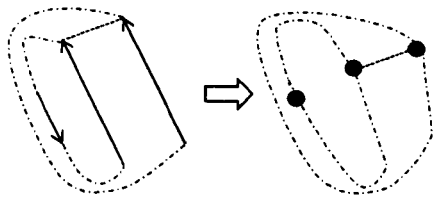

Figure 3.29: Step 3 - Merging two points in a cycle

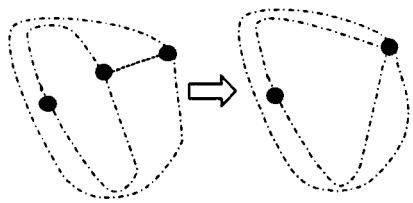

Figure 3.30: Step 4 - Removing edges between two cycles created in step 2 and gluing the two points joining the cycles

Proposition 3.4.1. *Algorithm 1 transforms a Gauss diagram with n non-crossing chords into an unrooted Eulerian map with n-edges where* $n > 0$.

Proof. Application of Reidemeister moves of type I↑ correspond to adding a chord (with two different possible directions) only to a free segment on the oriented circle (intersection of chords is not allowed). This is illustrated in Figure 3.31.

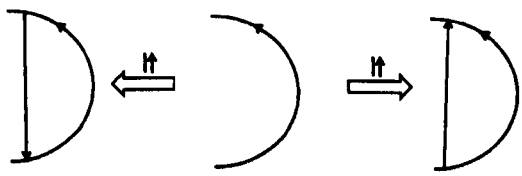

Figure 3.31: - Application of type I↑ move on a Gauss diagram

We will consider the proof by induction. To show that the base step holds, we consider all possible Gauss diagrams with one chord and apply the steps of *algorithm* 1 to each Gauss diagram in a sequential way to show that the corresponding map has only one edge. Since there are two variants of type I↑ moves, we consider each case separately. After applying *algorithm* 1, it is clear that each case results in a map with one edge as illustrated in Figure 3.32.

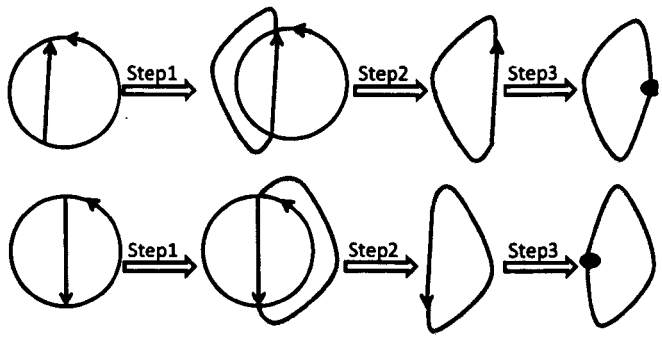

Figure 3.32: Base step - Possible Gauss diagrams with one chord

Let us assume that a Gauss diagram G with n chords can be transformed by algorithm 1 into a map with n edges. We will show that if we add a new chord to G, the number of edges of the corresponding map should also increase by one. The segment of the circle with dotted lines is assumed to contain $n-1$ non-crossing chords. To add a chord, we need to consider the two different directions of the new added chord, two different directions of the existing adjacent chord and two different possibilities of placing a chord either in a sequential order (after another existing chord) or in parallel (between the two end of an existing chord). Such consideration give rise to the following eight cases:

1. Adding a new chord ($\rightarrow$) immediately after another chord with the same direction (see Figure 3.34).

2. Adding a new chord ($\rightarrow$) immediately after another chord with opposite direction (see Figure 3.35).

3. Adding a new chord ($\rightarrow$) between the end-points of another chord with the same direction (see Figure 3.36).

4. Adding a new chord ($\rightarrow$) between the end-points of another chord with opposite direction (see Figure 3.37).

5. Adding a new chord ($\leftarrow$) immediately after another chord with opposite direction (see Figure 3.38).

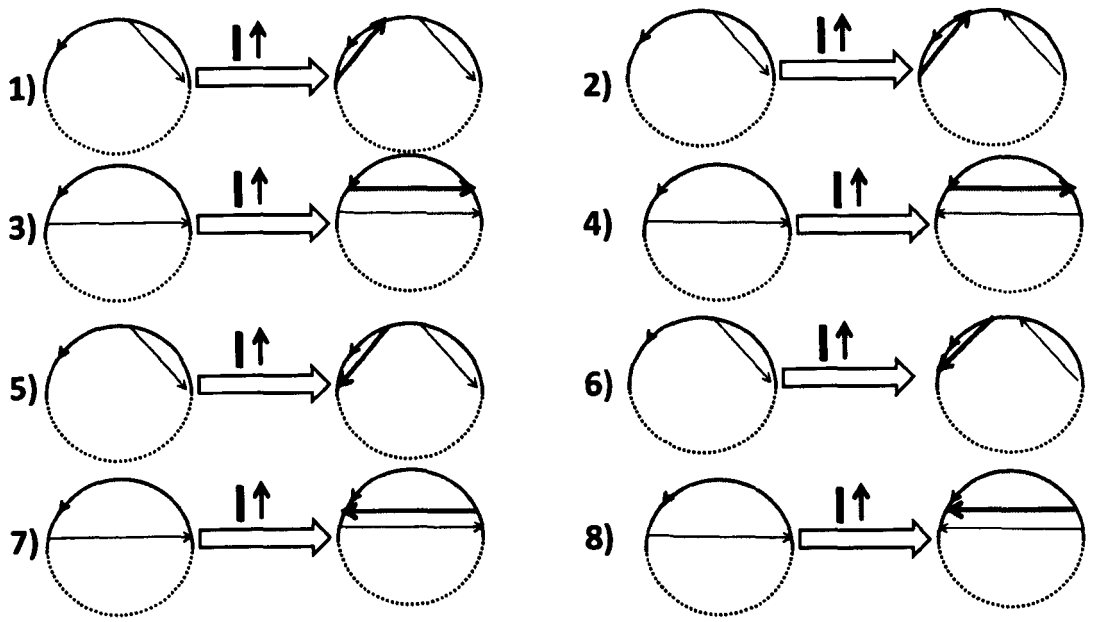

Figure 3.33: - all possible cases for adding a new chord

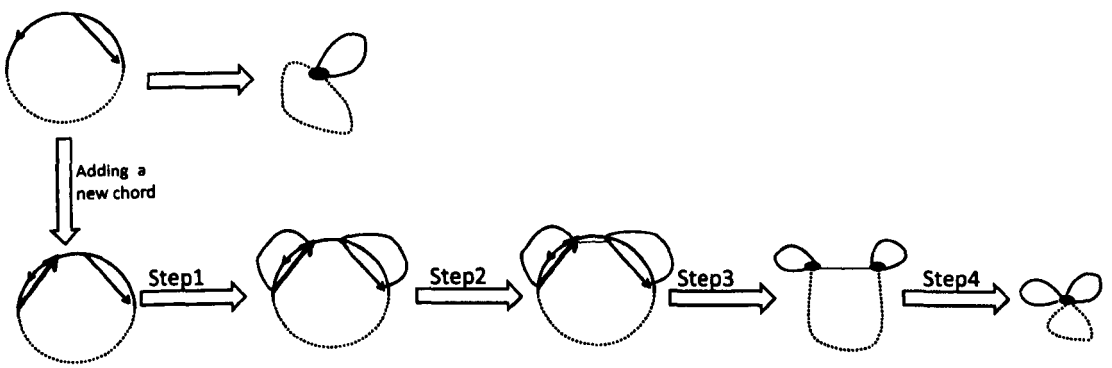

Figure 3.34: case 1 - Adding a new chord ($\rightarrow$) immediately after another chord with the same direction

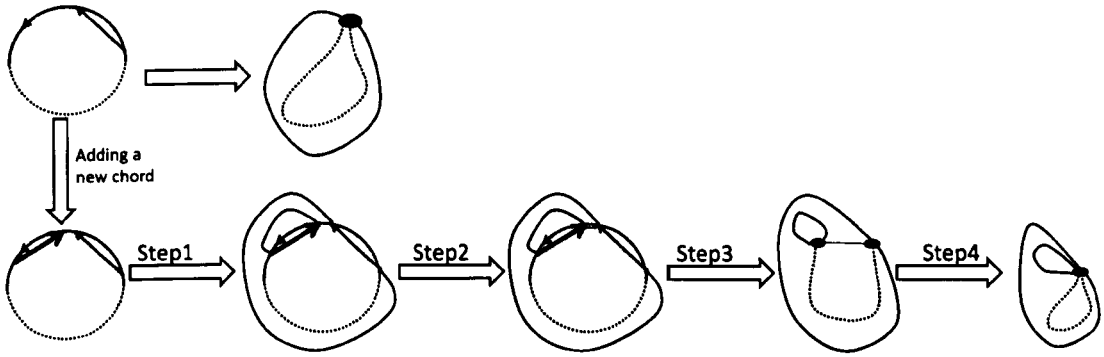

Figure 3.35: case 2 - Adding a new chord (→) immediately after another chord with opposite direction

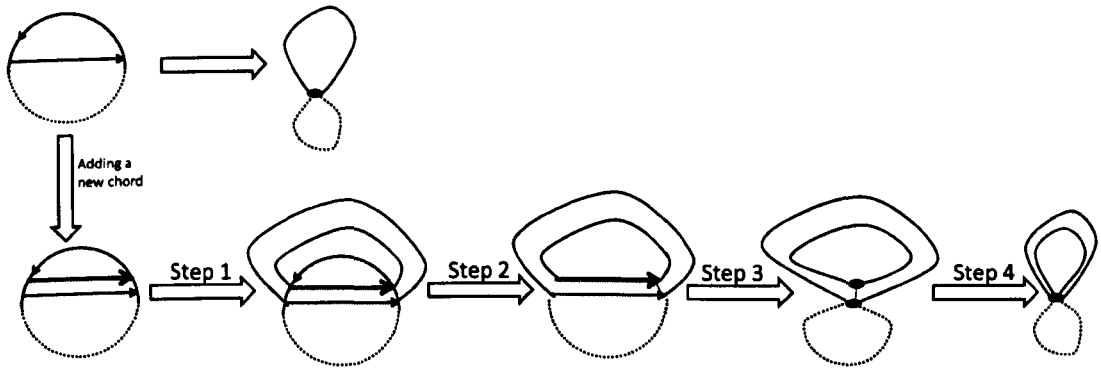

Figure 3.36: case 3 - Adding a new chord (→) between the end-points of another chord with the same direction

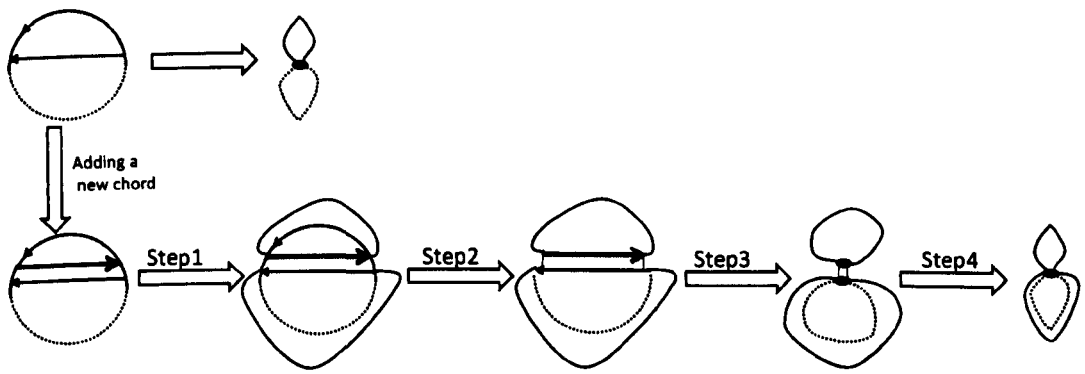

Figure 3.37: case 4 - Adding a new chord (→) between the end-points of another chord with opposite direction

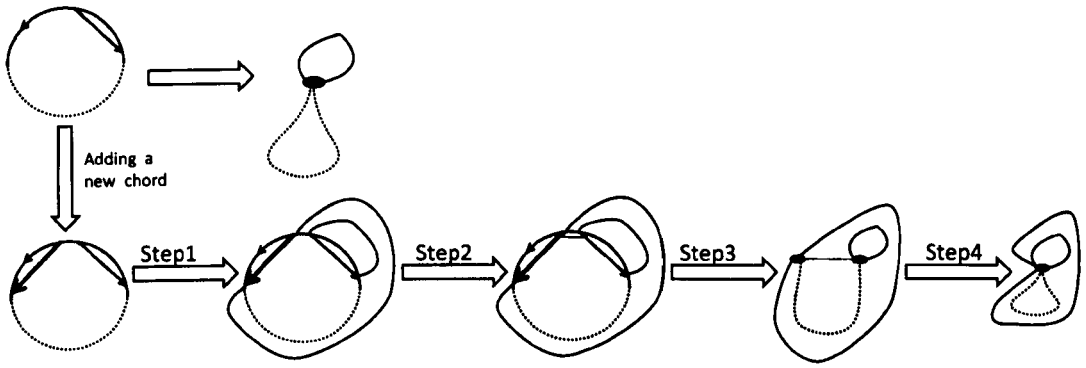

Figure 3.38: case 5 - Adding a new chord (←) immediately after another chord with opposite direction

6. Adding a new chord (←) immediately after another chord with the same direction (see Figure 3.39)

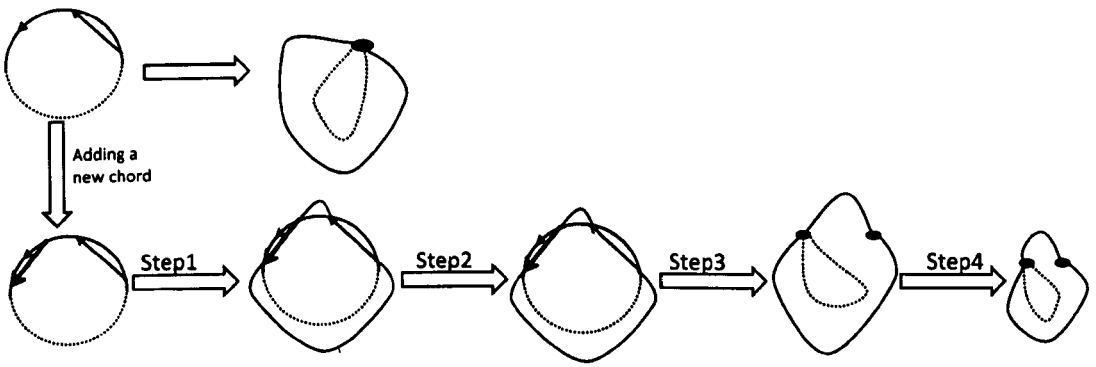

Figure 3.39: case 6 - Adding a new chord (←) immediately after another chord with the same direction

7. Adding a new chord (←) between the end-points of another chord with the same direction (see Figure 3.40)

8. Adding a new chord (←) between the end-points of another chord with opposite direction (see Figure 3.41).

In each case, we add a chord to a Gauss diagram G with n chords and apply the steps of algorithm 1 sequentially to transform G into a map. The number of edges of the map obtained coincide to the number of chords of G. The transformation as

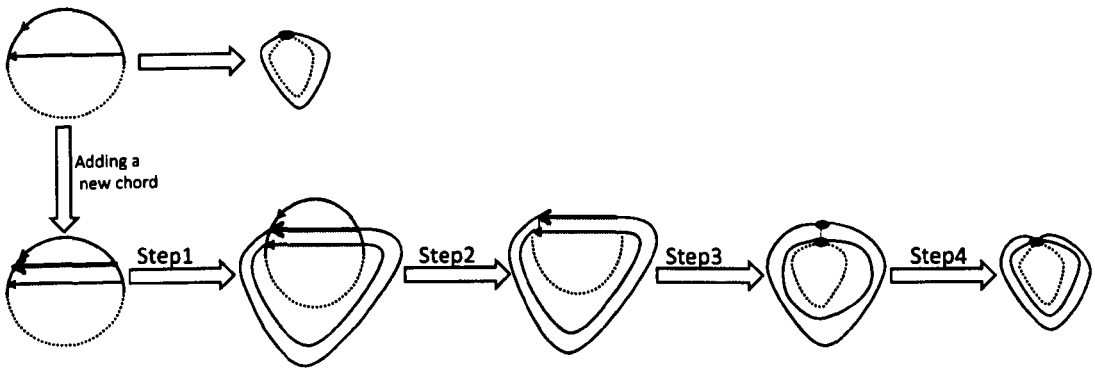

Figure 3.40: case 7 - Adding a new chord (←) between the end-points of another chord with the same direction

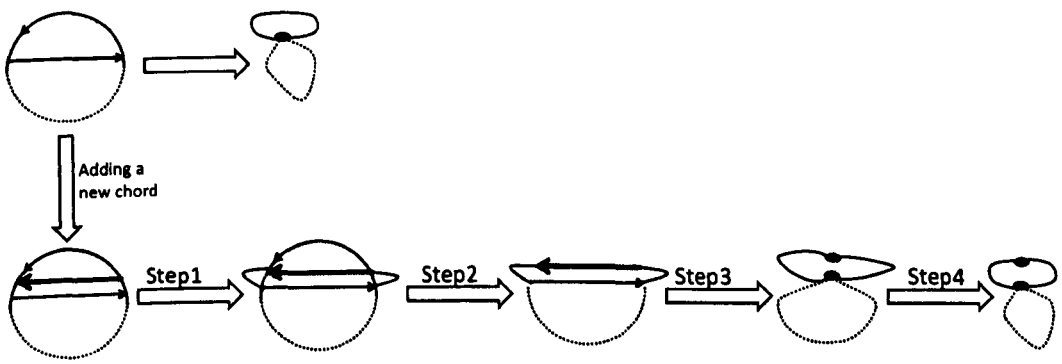

Figure 3.41: case 8 - Adding a new chord (←) between the end-points of another chord with opposite direction

illustrated in Figures 3.36 to 3.38 does not change the whole map except for a local part where the new edge is added. Thus the statement is true for $n+1$.

□

For a Gauss diagram x, by $x \to f_m(x)$ we will denote the correspondence map of x constructed using *Algorithm* 1.

3.4.2 Maps to Gauss diagrams

In this subsection, we present an algorithm to describe the second part of the correspondence concerning the transformation of an Eulerian map in the plane into a Gauss diagram. The first part of the algorithm is to construct an oriented circle and the second part is to construct oriented chords.

Definition 3.4.4. *Let G denote an Eulerian map, E denote a set of edges of G and V denotes a set of points of G. A cycle is a sequence of edges $S \subseteq E$ connecting a set of points such that the starting point and the ending point are the same. An inner face of a cycle c_i is a region in the plane that is surrounded by c_i. An outer face of c_i is a region that surrounds the boundary of c_i.*

We begin by partitioning the map into disjoint set of oriented cycles such that if two cycles share the same point then we make a copy of that point and attach one copy for each cycle then connect the two points by a new edge with a dashed line to join the two cycles together. Here, we assume that all points have distinct labels $(i, i_1, ..., i_n)$ and denote by D the set of directed edges with dashed lines. The number of edges with dashed lines incident to a point in any cycle is at most four (i.e. two edges (an incoming and an outgoing) from inside the cycle and two other edges from outside the cycle).

For example, if there are a set of cycles $C = \{c_1, ..., c_n\}$ where $n \geq 1$ glued to point i then we separate the cycles and arrange them in the plane from right to left with respect to the same order in which they appear in the map. We connect the points $i_1, ..., i_n$ which were previously glued to point $i \in c_i$ by edges in D as follows: If the cycles are

glued to point i from the inner face of cycle c_i and either no cycles linked to point j from inner face or the cycles linked to point j from the inner face are disjoint from those linked to point i from the inner face then connect $d(i,1), d(1,2), ..., d(n-1,n), d(n,j)$ where $d \in D$ and $i, j \in c_i$. Otherwise connect $d(i,1), d(1,2), ..., d(n-1,n), d(n,i)$ where $d \in D$ and $i \in c_i$ such that point 1 belongs to the first cycle c_1 which is located on right-hand side of point i and point n belongs to the last cycle which is located on the left-hand side of point i.

The cycles are arranged in this way to allow us obtain the order of chords on the circle.

We obtain an oriented circle by marking a starting point in the outer face and visiting each edge twice according to a set of rules formulated in step 3 of the algorithm. Once each edge has been visited twice then we return to our starting point. Finally we obtain the chords by removing all edges that were connected from the outside of the circle and all edges with dashed lines.

3.4.2.1 Algorithm 2

1. Orient all cycles in counter-clock wise direction.

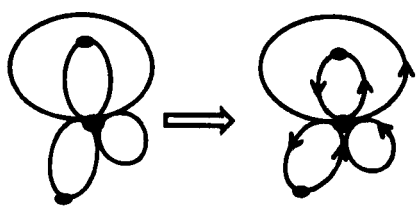

Figure 3.42: Step 1 - Orient all cycles in counter-clock wise direction

2. Partition the map into disjoint set of cycles such that if there are a set of cycles $C = \{c_1, ..., c_n\}$ where $n \geq 1$ glued to point i then

   (a) If the cycles $c_1, \cdots, c_n$ are glued to a point i from the inner face of cycle c_i where $i \in c_i$ AND If either there are no cycles glued to point j from the inner face of c_i where $i, j \in c_i$ OR there exists a set of cycles $c'_1, \cdots, c'_m$

where $m \geq 1$ glued to points j from the inner face of c_i AND $c'_1, \cdots, c'_m$ are disjoint from the cycles glued to point i then

i. Attach a new point $k \in \{i_1, \cdots, i_n\}$ to each cycle $c_k \in C$

ii. Arrange the cycles $c_1, \cdots, c_n$ in the plane inducing the same cyclic order imposed on the edges incident to point i in the map such that c_1 is the first cycle attached to point i and c_n is the last cycle attached to point i.

iii. Connect $d(i, i_1), d(i_1, i_2), \cdots, d(i_{n-1}, i_), d(i_n, j)$ where $d \in D$ and $i, j \in c_i$.

(b) Else

- Apply steps 2(a)i and 2(a)ii
- Connect $d(i, i_1), d(i_1, i_2), \cdots, d(i_{n-1}, i_n), d(i_n, i)$ where $d \in D$

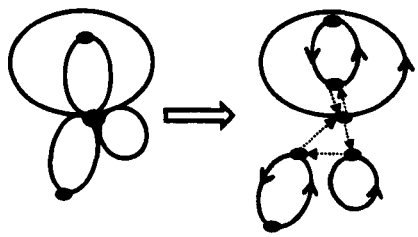

Figure 3.43: Step 2 - Partitioning the map into disjoint set of oriented cycles joined by dashed line edges

3. Mark an arbitrary point in the outer face and walk along each edge twice until the same initial point is reached for the second time according to the following rules:

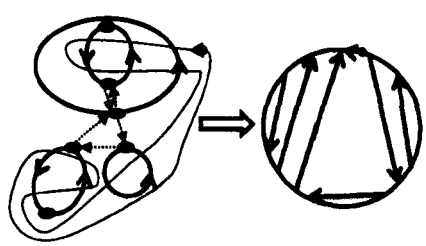

Figure 3.44: Step 3 - Traversing along edges to obtain a Gauss diagram

(a) If one crosses an edge e_{ij} from an outer face of cycle c_i where $e_{ij} \in c_i$ then

i. If there exists an edge $d_{ik} \in D$ such that k is connected to i from the inner face of c_i, then

- Move to the edge e_{km} where $e_{km} \in E$

ii. Else move to e_{jn}.

(b) If one crosses an edge e_{ij} from an inner face of some cycle c_i where $e_{ij} \in c_i$ then

i. If there exists an edge $d_{ik} \in D$ such that k is connected to i from the outer face of c_i, then

- Move to the edge e_{km} where $e_{km} \in E$

ii. Else move to e_{ij}.

4. Orient the circle in counter-clock direction and remove all edges in the outer face of the circle and all other edges with dashed lines.

Proposition 3.4.2. *Algorithm 2 transforms an unrooted Eulerian map with n-edges into a Gauss diagram with n non-crossing chords where* $n > 0$.

Proof. The proof is by induction on the number of edges of a map. For the base case, we consider an Eulerian map with one edge and show that the outcome is a corresponding Gauss diagram with one chord. In Figure 3.45 we apply algorithm 2 to a map with one edge and illustrate the transformations involved in each step to produce the expected Gauss diagram with one chord.

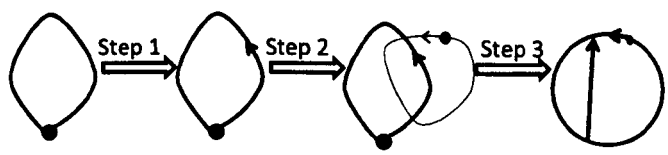

Figure 3.45: Base step - Transforming A Gauss diagram into a map

For the inductive step, we will assume that it holds for any map with n edges and demonstrate that if we add a new edge, the number of chords of the corresponding Gauss

diagram increases by one. To add a new edge, we either place a point on an existing edge (to split it into two edges) or add a new loop to an existing point. There are only two possibilities of where the new loop can be appear on the diagram (either attached to a point from an inner face or an outer face). We consider all possible cases below. In each case, we consider a map with n edges and its corresponding Gauss diagram (the bottom part of the map with dotted lines is assumed to contain $n-1$ edges). After adding a point or a loop, we follow the steps of the algorithm by partitioning the map into disjoint set of oriented cycles and then introducing new directed edges (with dashed lines) to join the cycles. Finally, we obtain a Gauss diagram by walking along all edges according to the rules formulated in step 3:

1. Adding a loop to an existing point from an inner face as illustrated in Figure 3.46.

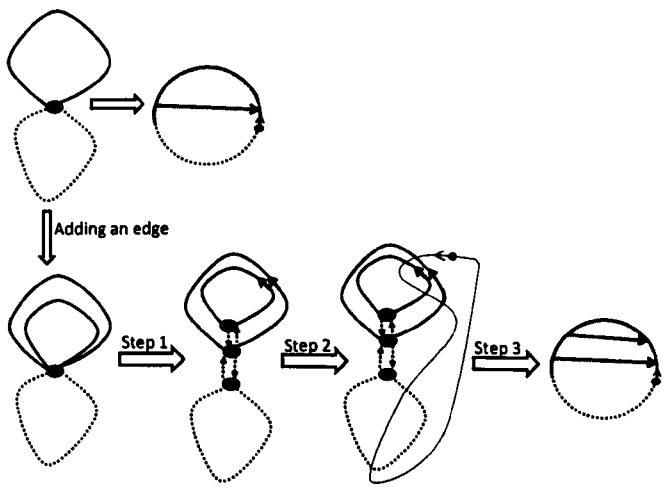

Figure 3.46: case 1 - Adding a new loop to a point from an inner face

2. Adding a loop to an existing point from an outer face as illustrated in Figure 3.47.

3. Adding a point to an existing edge

The transformation as illustrated in Figures 3.46 to 3.48 is done locally and does not

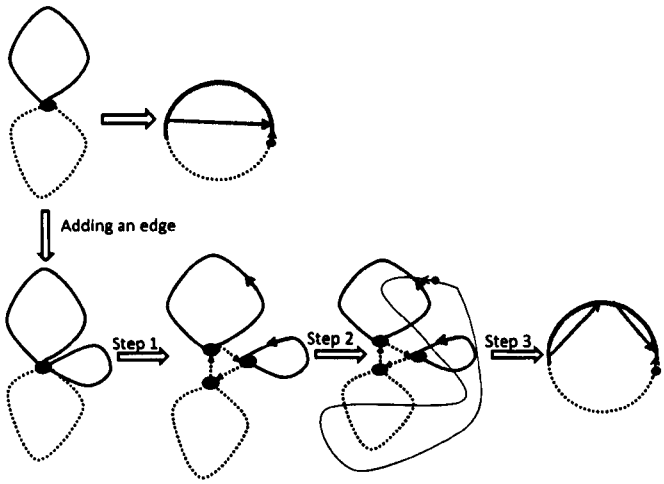

Figure 3.47: case 2 - Adding a new loop to a point from an outer face

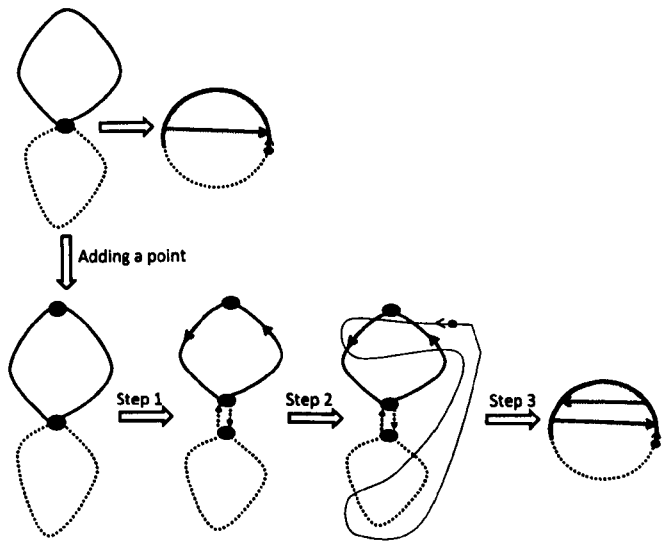

Figure 3.48: case 3 - Adding a point to an existing edge

change other parts of the Gauss diagram. The number of chords of the corresponding Gauss diagram is also increased by one when either we add a loop or a point. □

For a map y, by $y \rightarrow f_k(y)$ we will denote the correspondence Gauss diagram of y obtained by *algorithm* 2.

Conjecture 3.4.1. *For any Gauss diagram x and any map y, if $x \rightarrow f_m(x)$ and $y \rightarrow f_k(y)$ then $x = f_k((f_m(x))$ and $y = f_m(f_k(y))$.*

Comment. Because both algorithms are deterministic and produce a unique output for each distinct input we believe that this conjecture is true but still needs some formal proof.

3.5 Generalised Reidemeister moves

In general the upper bound for transforming a trivial knot diagram into the unknot diagram based on Reidemeister moves has shown to be exponential with respect to the number of crossings [31]. One of the reasons the number of Reidemeister moves is exponential is possibly due to the existence of some knots which require an increase in their crossings number before they can be simplified into the unknot. One may ask whether introducing new moves in addition to ordinary Reidemeister moves may help with simplifying trivial knots without increasing their number of crossings during the transformation (i.e. by avoiding Reidemeister moves of types I↑ and II↑). Such a question was investigated in [24] where some generalised version of Reidemeister moves (illustrated in Figure 3.49 taken from [25]) also referred to as pass moves [14]) particularly for types I and II were presented. However, a counter example (see Figure 3.57) was presented in the same paper where it still required an increase in number of crossings even after considering the new added moves in Figure 3.49.

In this section, we introduce a new set of moves depicted in Figure 3.51 that will be referred to as generalised Reidemeister moves (GR). Our proposed moves are more general than [24] but can be seen as a restrictive case of pass moves for links shown

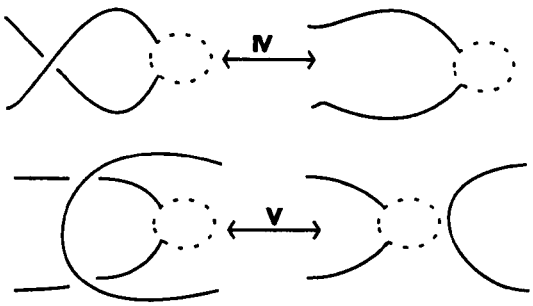

Figure 3.49: - types IV and V pass moves [25]

in [[14], page 67]. Although these moves are restrictive, we will show in Proposition 3.5.1 that all known examples of classical complex trivial knot diagrams that we found in the literature can be simplified using our new set of moves coupled with ordinary Reidemeister moves without increasing their number of crossings. In particular, our

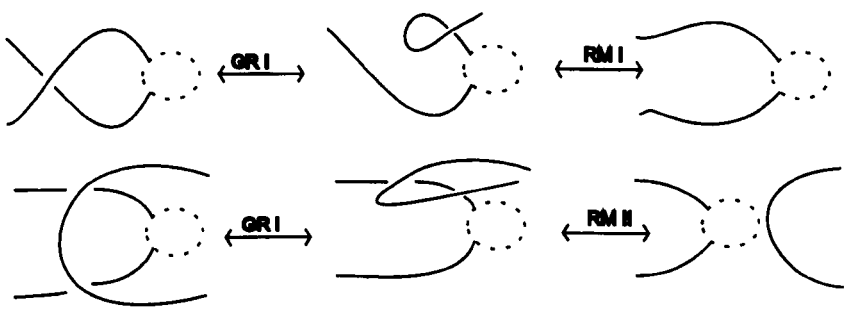

Figure 3.50: - Transformation of Goeritz moves by Generalised Reidemeister moves of type I

generalisation as demonstrated in Figure 3.50 captures the two moves presented by Goeritz by *GRI*. Extending *GRI* (with two strands entering the circle and two other strands emanating out of the circle), we obtain *GRIII* which is a more generalised form of an ordinary type III move. Similarly Reidemeister move of type II is extended by *GRII*. The dotted circle denotes a local fragment of the knot.

The introduction of generalised moves can lead to an improvement in the overall complexity bounds for the unknotting problem if the transformation is restricted to knot diagram simplification. This in turn reduces the overall number of Reidemeister moves required for the transformation by avoiding types I↑ and II↑ (increase) particularly

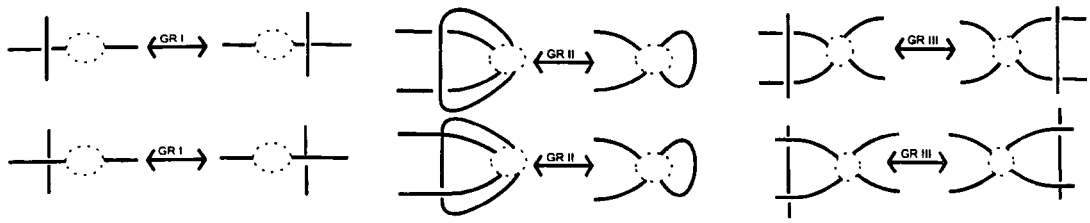

Figure 3.51: - Generalised Reidemeister moves

when either types I↓ and II↓ (decrease) or type III become unavailable. The difficulty with application of type II↑ as we noted in Section 3.2.2 is that it can not be applied arbitrarily for Gauss words because information about faces of the knot diagram is required to maintain the planarity of the resultant diagram. In addition to faces, we need to determine which of the strands if selected would result in decreasing number of crossings of the diagram . In both case such information is not available for Gauss words (with an exception to signed Gauss word where there is a procedure for determining faces [38]).

Let $RM = \{RMI, RMII, RMIII\}$ denote a set of Reidemeister moves and $GR = \{GRI, GRII, GRIII\}$ denote a set of generalised Reidemeister moves, we will show in the next proposition that if a given knot diagram K_1 can be transformed by GR into another knot diagram K_2 then likewise the same knot diagram can be transformed via RM.

Proposition 3.5.1. *Given two knot diagrams K_1 and K_2, if $K_1 \xrightarrow{GR} K_2$ then $K_1 \xrightarrow{RM} K_2$*

Proof. The proof consists of showing that each move in GR can be replaced by a sequence of RM. In each GR move we require to pass a strand consisting of either one crossing (for the case of GRI) or two crossings with the same type (either two-over-crossings or two-under-crossings for the case of $GRII$ and $GRIII$) through a sequence of crossings belonging to a fragment (or a partition) of the knot diagram (marked by a dotted circle in Figure 3.51) to reach some position located outside the dotted circle. We will show that the sequence of RM of types II↑ III and II↓ can be used to substitute any GR move.

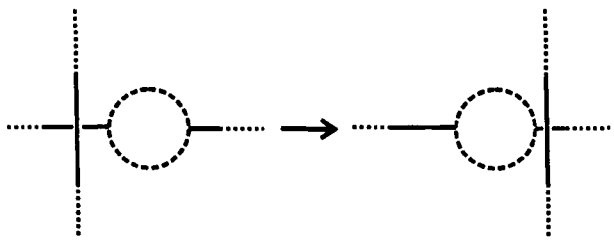

Figure 3.52: - Moving a strand over a fragment of a knot diagram

Let us consider the diagram in Figure 3.52 for which the configuration on the left denote the precondition of the rule and the configuration on the right denote the post-condition of the rule and let F denote the set of outer faces of fragment S where a face f of a fragment S is called an inner face iff all of the arcs (edges) surrounding f belong to S and otherwise it is called an outer face iff some of the arcs (edges) surrounding f belong to S.

Starting with Figure 3.53 we show how a strand can passed through a single crossing using a sequence of RM of types II↑, III and II↓.

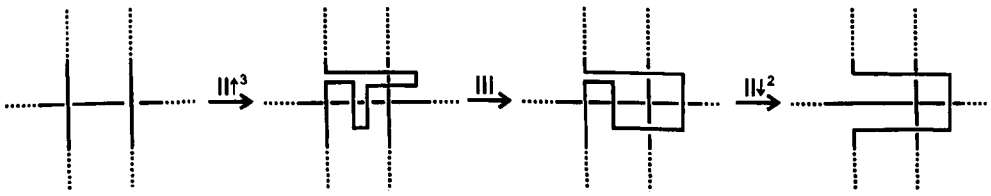

Figure 3.53: - Passing a strand through a single crossing using a sequence of Reidemeister moves of types II and III

More generally for the cases of GRI and $GRII$ where it's required to pass a strand through a sequence of crossings we apply the same sequence to move through each crossing in a breadth first search manner by visiting neighbouring faces as follows:

1. For each face belonging to the fragment

   (a) Mark all edges with II↑ as illustrated in Figure 3.54.

   (b) Encircle the crossing points by applying moves of type III and II↓

(c) Move to the next neighbouring faces until one of the outer faces of the fragment is reached.

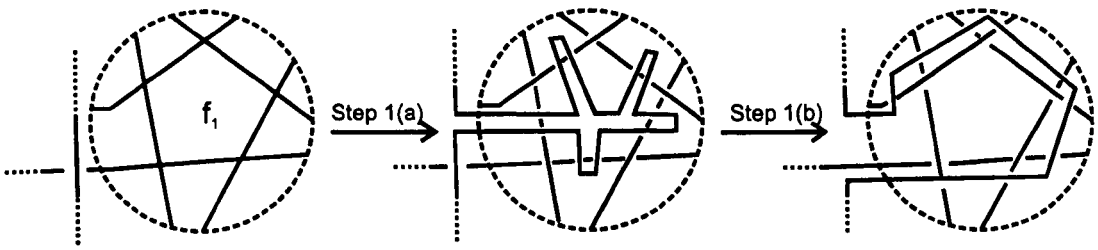

Figure 3.54: - Passing a strand through a face f_1 of some fragment subsuming all edges and crossing points of f_1 using steps 1(a) and 1(b)

2. If the outer face is reached then apply II↓.

Since the fragment of a knot contains finitely many faces and crossings therefore the procedure is guaranteed to terminate.

□

In the next proposition, we consider all known classical examples of complex trivial knot diagrams which can be unknotted using classical Reidemeister moves only by first increasing number of crossings of the original diagram (i.e. where the original diagram does neither admit types I↑ and II↓ nor type III and therefore the only applicable move is type II↑). We will show that each diagram can be transformed into the unknot without exceeding number of crossing of original diagram by substituting types II↑ with generalised Reidemeister moves.

Proposition 3.5.2. *The following knot diagrams in Figure 3.55 can be transformed into the unknot diagram without increasing number of crossings.*

Proof. Apply a generalised move when either types II↓ and I↓ or type III are not available. To avoid increasing number of crossings during transformation we substitute Reidemeister moves of type II↑ with a more generalised move depicted in Figure 3.51.

See Figures 3.56 to 3.61 for an illustration of the transformations of diagrams $A, \cdots, D$ to the unknot respectively. □

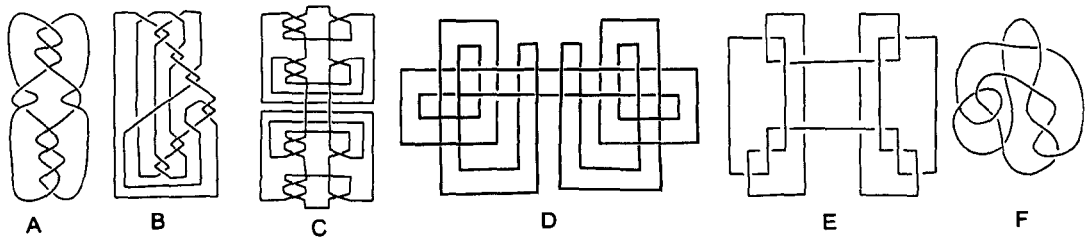

Figure 3.55: Complex trivial diagrams - A and B([24]); C ([32]); D and E ([47]); F (KnotPlot)

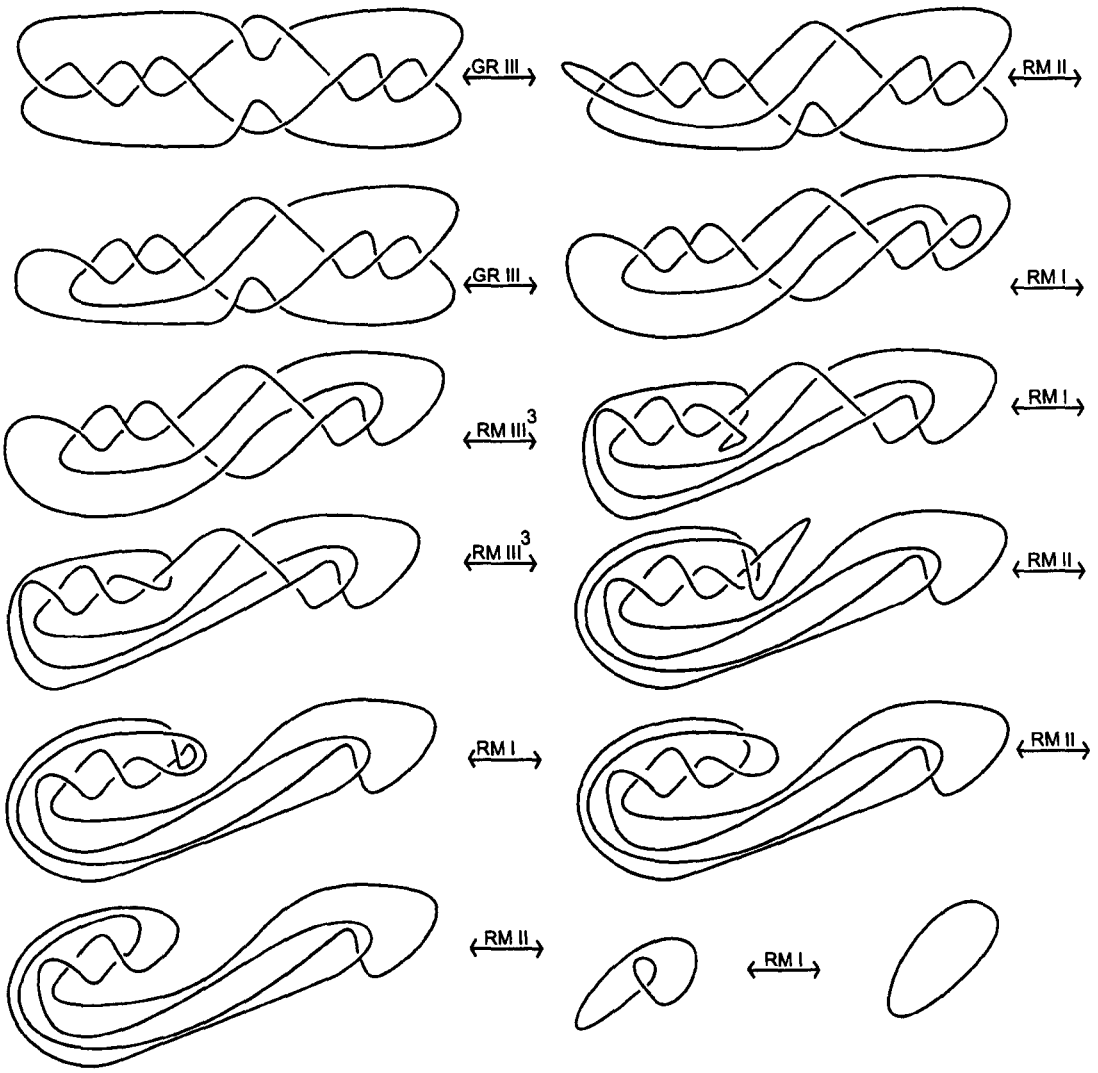

Figure 3.56: A - taken from [24] - Transformation of diagram A into the unknot without increasing number of crossings

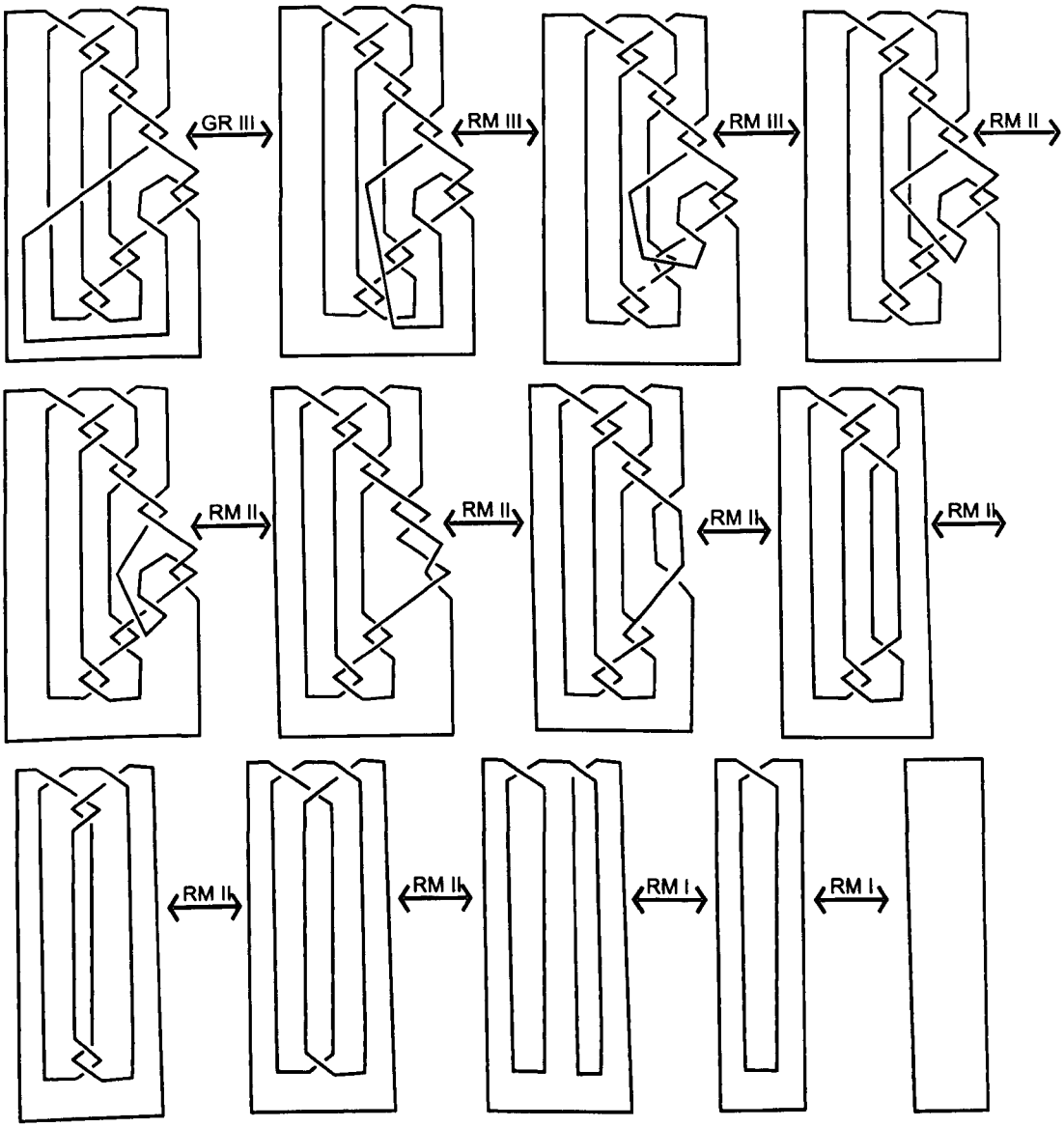

Figure 3.57: B- taken from [24] - Transformation of diagram B into the unknot without increasing number of crossings

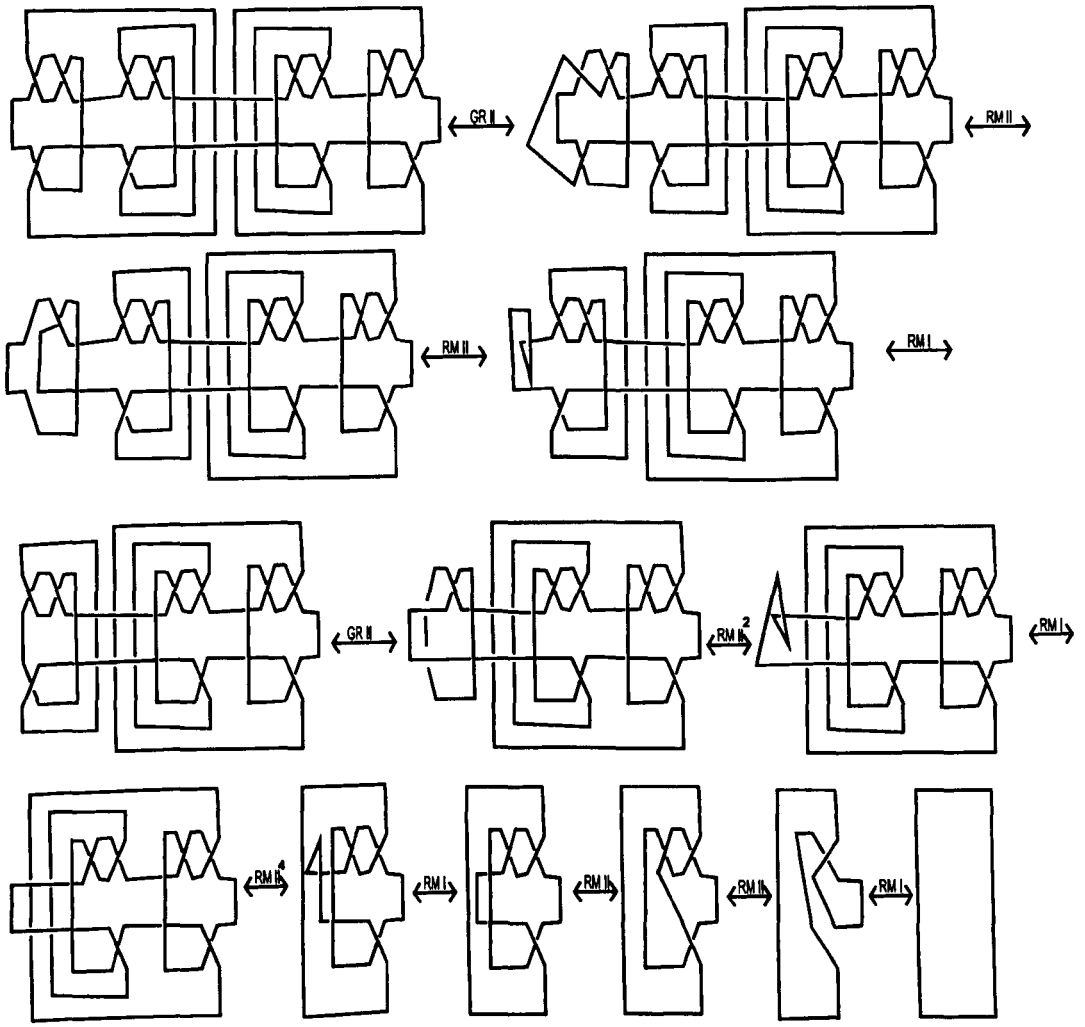

Figure 3.58: C- taken from [32] - Transformation of diagram C into the unknot without increasing number of crossings

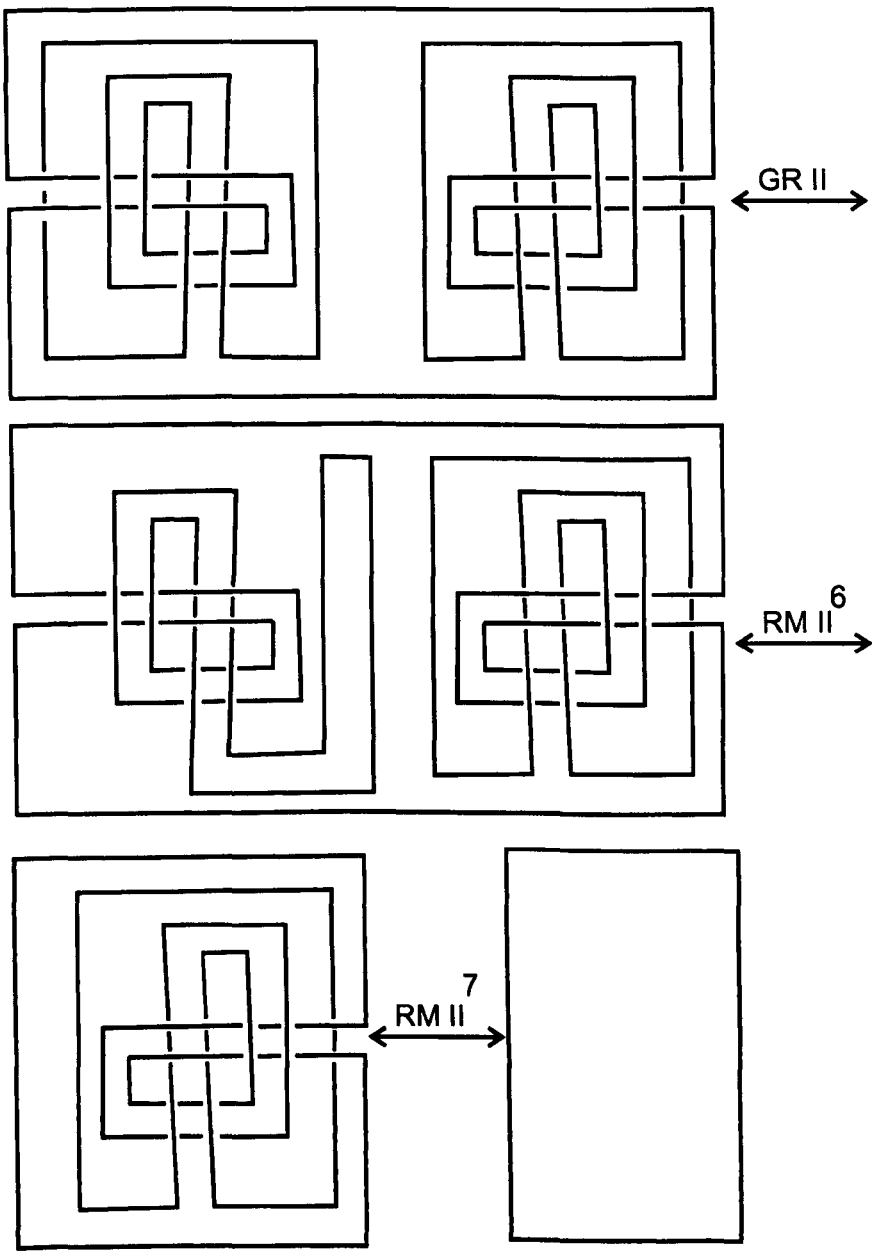

Figure 3.59: D- taken from [47] - Transformation of diagram D into the unknot without increasing number of crossings

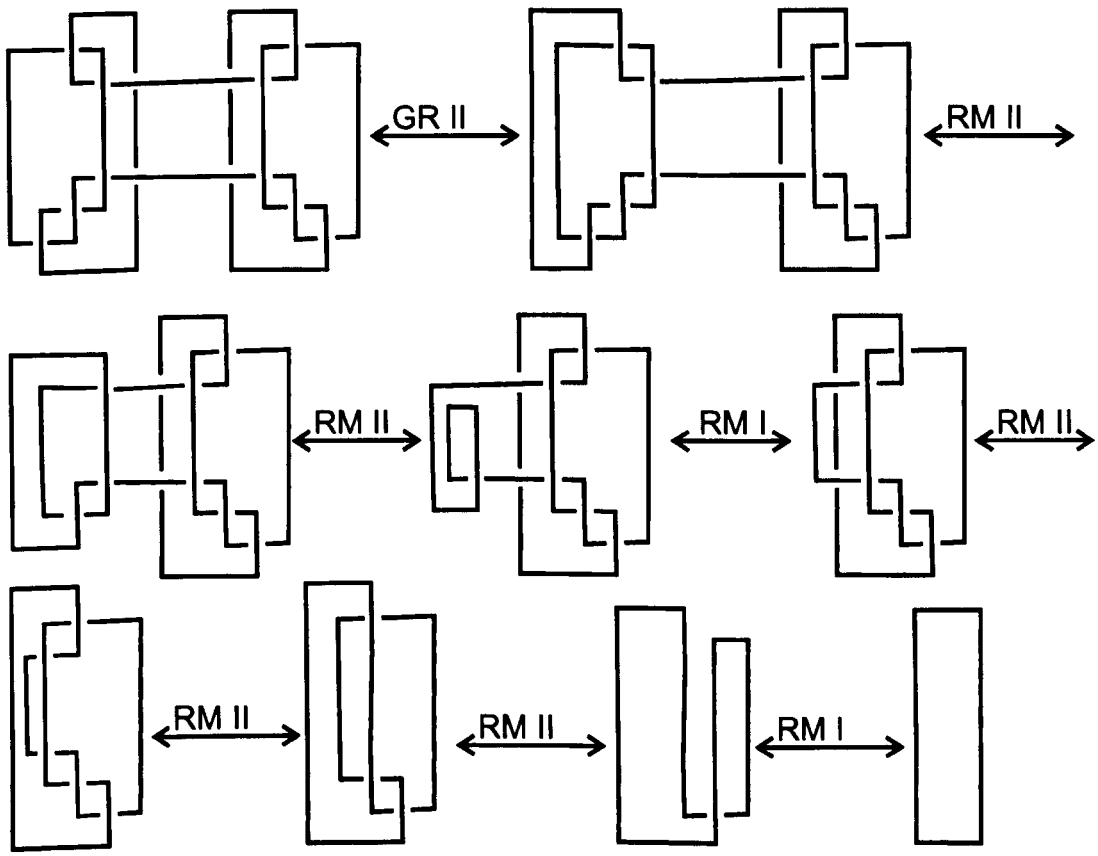

Figure 3.60: E- taken from [47] - Transformation of diagram E into the unknot without increasing number of crossings

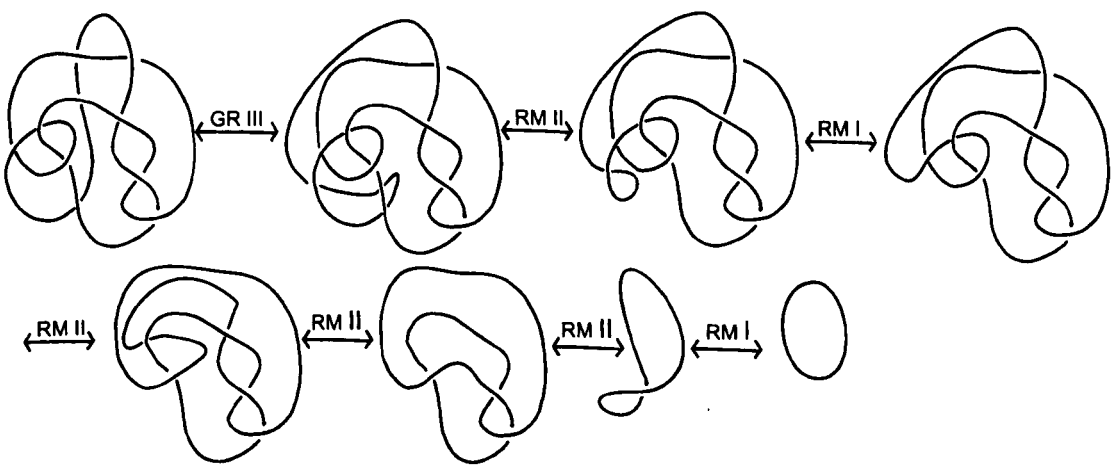

Figure 3.61: F- taken from KnotPlot - Transformation of diagram F into the unknot without increasing number of crossings

3.5.1 Generalised Reidemeister moves as rewriting rules

In this subsection, we consider formalising generalised Reidemeister moves in terms of string rewriting rules for Gauss words. In a similar way to classical Reidemeister moves, we take into account all possible orientations of the strands involved and all possible ways in which they can be connected (or ordered) on the knot diagram. We present one rule for *GRI*, two rules for *GRII* and 14 rules for type III. The rules are formulated as conditional rewriting rules and not in the form as defined in Section and also we use abbreviations like i to denote either O_i or U_i.

Definition 3.5.1. *A Gauss word s is self-contained word if for every symbol O_i in s there exists a corresponding symbol U_i in s and vice versa.*

Type I

A Gauss words admit GR I, if it contains a subword s that is a self-contained word and one of the symbols U_i or O_i adjacent to s for some $i \in \mathbb{N}$. For the purpose of simplification the adjacent symbol to s will be denoted by i to capture both cases (U_i and O_i) since it will not affect the definition of a self-contained word.

Definition 3.5.2. *Let w be a Gauss word, w admits GR I iff $w = xis$ or $w = xsi$ where s is self-contained.*

Let w be a Gauss word that admits GR of type I, application of GR I allows us to move a symbol i adjacent to a self-contained subword s from the beginning of s to the end of s and vice versa. To capture this effect we formulate the following rule for GR I.

GR1.1 $xis \leftrightarrow xsi$

Type II

Definition 3.5.3. *Let w be a Gauss word and $s = p \cdot q$ be a self-contained subword of w, w admits GR II iff either $w = x\bar{i}pw_aq\bar{j}$ or $w = x\bar{j}pw_aq\bar{i}$ where $w_a = O_iO_j$ or $w_a = U_iU_j$ and $\bar{i}, \bar{j}$ denote the counterpart symbols of w_a.*

Unlike GR I and GR III, application of GR II can affect the size of w by ± 4. In the following rules the concatenation of p and q denoted by $p \cdot q$ forms a self-contained word.

GR2.1 $x\bar{i}pw_aq\bar{j} \leftrightarrow xpq$

GR2.2 $x\bar{j}pw_aq\bar{i} \leftrightarrow xpq$

Type III

In the following definition we introduce some notation to describe the relation between two symbols in a Gauss word for the purpose of simplifying the various cases used in Definition 3.5.5.

Definition 3.5.4. *Let x, y be some symbols in a Gauss word w*

- *We denote by $P_{x,y} = \{(x,y)|xy \in w\}$ the set of pairs (x,y) where x is a predecessor of y (i.e. x appears to be the first symbol on the left-hand side of y). The set $S_{x,y} = P_{y,x}$ denotes the set of pairs (x,y) where x is a successor of y and the set $A_{x,y}$ is defined as either $A_{x,y} = P_{x,y}$ or $A_{x,y} = P_{y,x}$*
- *The symbol $\bar{x}$ denotes the counterpart of x where $x = O_i$ iff $\bar{x} = U_i$ and vice versa.*

Definition 3.5.5. *Let w be a Gauss word and $s = p \cdot q$ be a self-contained subword of w, w admits GR III iff $w = xw_ayw_bzw_c$ where*

- *$w_a = O_iO_j$ or $w_a = U_iU_j$ and*
  - *$w_b \in A_{\bar{i},p}$ and $w_c \in A_{\bar{j},q}$ or $w_c \in A_{\bar{i},p}$ and $w_b \in A_{\bar{j},q}$ or*
  - *$w_b \in A_{\bar{j},p}$ and $w_c \in A_{\bar{i},q}$ or $w_c \in A_{\bar{j},p}$ and $w_b \in A_{\bar{i},q}$ or*
  - *$w_b = \bar{i}p\bar{j}$ and $w_c = q$ or $w_c = \bar{i}p\bar{j}$ and $w_b = q$ or*
  - *$w_b = \bar{j}p\bar{i}$ and $w_c = q$ or $w_c = \bar{j}p\bar{i}$ and $w_b = q$*

Let w be a Gauss word admitting GR III, we denote by p_1 and p_n (and q_1 and q_n respectively) the first symbol and last symbol of p (and q respectively). The rewriting

of w by GR III depends on whether the pair (p_1, p_n) interlace with the pair (q_1, q_n) in w or not such that either $p_1 = \bar{q_1}$ (the first symbol in p is equal to the counterpart of the first symbol in q) and $p_n = \bar{q_n}$ or $p_1 = \bar{q_n}$ and $p_n = \bar{q_1}$. The third case captures the case when either $p_1 \neq \bar{q_1}$ and $p_n \neq \bar{q_n}$ or $p_1 \neq \bar{q_n}$ and $p_n \neq \bar{q_1}$.

The rules 3.1 to 3.4 correspond to the first case where the symbols connecting to the counterpart symbols of w_a holds and the rules 3.5 to 3.8 correspond to the second case where the interlacement property does not hold. The rules 3.9 and 3.14 correspond to the third case

GR3.1 $w_a x\bar{i}pyq\bar{j}z \leftrightarrow w_a xp\bar{j}y\bar{i}qz$

GR3.2 $w_a xp\bar{i}y\bar{j}qz \leftrightarrow w_a x\bar{j}pyq\bar{i}z$

GR3.3 $w_a xpy\bar{j}q\bar{i}z \leftrightarrow w_a x\bar{j}p\bar{i}yqz$

GR3.4 $w_a xpy\bar{i}q\bar{j}z \leftrightarrow w_a x\bar{i}p\bar{j}yqz$

GR3.5 $w_a x\bar{i}py\bar{j}qz \leftrightarrow w_a xp\bar{j}yq\bar{i}z$

GR3.6 $w_a xp\bar{i}yq\bar{j}z \leftrightarrow w_a x\bar{j}py\bar{i}qz$

GR3.7 $w_a xpy\bar{i}q\bar{j}z \leftrightarrow w_a x\bar{j}p\bar{i}yqz$

GR3.8 $w_a xpy\bar{j}q\bar{i}z \leftrightarrow w_a x\bar{i}p\bar{j}yqz$

GR3.9 $w_a xp\bar{i}y\bar{j}qz \leftrightarrow w_a x\bar{i}pyq\bar{j}z$

GR3.10 $w_a xp\bar{i}yq\bar{j}z \leftrightarrow w_a x\bar{i}py\bar{j}qz$

GR3.11 $w_a xp\bar{j}y\bar{i}qz \leftrightarrow w_a x\bar{j}pyq\bar{i}z$

GR3.12 $w_a xp\bar{j}yq\bar{i}z \leftrightarrow w_a x\bar{j}py\bar{i}qz$

GR3.13 $w_a x\bar{i}p\bar{j}yqz \leftrightarrow w_a xpy\bar{j}q\bar{i}z$

GR3.14 $w_a x\bar{j}p\bar{i}yqz \leftrightarrow w_a xpy\bar{i}q\bar{j}z$

3.6 Summary

In this chapter we considered local diagrammatic moves called Reidemeister moves used for the transformations between two knots of the same type. We provided a formalisation of Reidemeister moves in terms of string rewriting systems and analysed the minimal set of rules sufficient for rewriting. Then we analysed the reachability problem of Gauss words in terms of string rewriting rules and provided some lower and upper bounds based on the number of transformations needed to reach one word from the other by considering a set of rules of the same type as well as using a combination set of rules of different types.

Further we considered oriented knot diagrams (represented by Gauss diagrams) generated by application of Reidemeister moves of type I only and counted the number of non-isomorphic knot diagrams with n-crossings (where $n \geq 1$). We discovered that our sequence corresponds to the number of unrooted Eulerian n-edge maps in the plane and provided explicit algorithms to describe the construction between maps and Gauss diagrams.

Furthermore we introduced a new set of moves that can be seen as a more generalised version of Reidemeister moves of types II and III formulated also in terms of string rewriting rules for Gauss words. We considered all known examples of complex trivial knot diagrams that can only be reduced to the trivial knot diagram by first increasing the number of crossings of the original diagram and shown that our generalised moves can be used to substitute application of Reidemeister moves of type II$\uparrow$ and therefore simplifying the transformational process into a reduction one. Finally we formulated some open questions left as conjectures for future work.

Chapter 4

Computational models

In this chapter we introduce and analyse some computational models over infinite alphabets for the purpose of evaluating the complexity of some knot theoretic problems represented by Gauss words.

Since knots can have arbitrarily many crossings then Gauss words will be considered as strings over an infinite (unbounded) alphabet. In section 4.1, we describe and extend the models of automata over infinite alphabet that will be used for establishing the lower and upper bounds on recognition of knot properties.

In Section 4.2 we examine the capability of a register automata to handle some useful operations required for checking knot properties and show that over a class of all languages of our interest register automata can simulate the behaviour of a counter machine and that of a pebble automata. Then we demonstrate generic results on the mutual simulations between logspace bounded classical computations (over finite alphabets) and register automata working over infinite alphabets.

In Section 4.3 we apply register automata to establish some lower and upper bounds for the recognisability of some knot properties. We show that the languages of Gauss words (signed and unsigned) are not recognisable by non-deterministic 1-way register automata but they are recognisable by deterministic 2-way register automata. We also show that non-trivial properties such as checking the equivalence of two Gauss words in terms of cyclic shift and renaming of labels referred to as isomorphic Gauss words

problem can be recognised by a deterministic 2-way register automata.

4.1 Automata over infinite alphabets

Let D be an infinite set called an *alphabet*. A word, or a string over D, or shortly, D-word or D-string, is a finite sequence $d_1, \ldots, d_n$, where $d_i \in D$, $i = 1, \ldots, n$. A language over D (a D-language) is a set of D-words. For a word w and a symbol d, denote by $\mid w \mid_d$ the number of occurrences of d in w. As usual $\mid w \mid$ denotes the length of the word w. A language L over an infinite alphabet D is called *n-bounded* if there is a constant $n \in \mathbb{N}$ such that for any $w \in L$ and for any $d \in D$ one has $\mid w \mid_d \leq n$. All languages considered in this thesis are bounded.

The language of shadow Gauss words L over D is *2-bounded* due to the double occurrence of the symbols of d in w, i.e. $\mid w \mid_d = 2$. The language of nested words [2] can also be viewed as a language of $2 -$ *bounded* data words.

4.1.1 Words and data words

In previous works on the computational models on infinite alphabets it has been acknowledged that in many situations it is natural to consider infinite alphabets as the subsets of $\Sigma \times \Delta$ where Σ is a *finite* set and Δ is an infinite set. Thus, the symbol here is an ordered pair (a, b). The words over such alphabets are called *data words*. In the definition of automata over data words, it is sensible to assume that when an automaton reads a symbol (a, b) it has a direct access to both components of the pair. For this purpose, the form of transition rules in automata can be adapted to include one extra argument on the left-hand sides as defined in Definition 4.1.3.

For the language of Gauss words, we consider one of the weakest models of automata over infinite alphabet called register automata, introduced in [35] and studied further in [55]. This model is quite restrictive in a way that it can only store unique values from the infinite alphabet in registers. This restriction was inconvenient for designing automata to check Gauss word properties.

In the next section, we describe how to overcome this restriction.

4.1.2 Register automata

Register automata are finite state machines equipped with a finite number of memory cells called registers which may hold values from an infinite alphabet. There are several variants of register automata which have different computational power. These are deterministic, nondeterministic, one-way and two-way register automata. Below, we describe the formal definition of a two-way k-register automaton taken from [55].

Definition 4.1.1 ([55]). *A (non-deterministic) two-way k-register automaton over an infinite alphabet D is a tuple (Q, q_0, F, τ_0, P) where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, $\tau_0 : \{1,...,k\} \to D \cup \{\triangleright, \triangleleft\}$ is the initial register assignment and P is a finite set of transitions of the forms:*

1) $(i, q) \to (q', d)$ (If a current state is q and the observed symbol on the tape equals to a value in register i then enter the state q' and move along the string according to the specified direction d where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

2) $q \to (q', i, d)$ (If a current state is q and the observed symbol on the tape does not equal to any value held in registers then enter the state q', copy the current symbol to a specified register i and move along the string according to the specified direction d where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

Given a D-word w delimited by symbols $\triangleright$,$\triangleleft$ on the input tape, an automaton starts in a state q_0 and in the position of the first letter of w and applies non-deterministically any applicable rules. A configuration of an automaton on w is a tuple $[j, q, \tau]$ where j denote a position of a letter of w or the position of either $\triangleright$ or $\triangleleft$, $q \in Q$ and $\tau : \{1, ..., k\} \to D \cup \{\triangleright, \triangleleft\}$. As usual, if automaton is able ever to reach a state $q \in F$, it accepts the word, otherwise the word is rejected. The set of all accepted words forms a language recognisable by an automaton.

An automaton is *deterministic* if for all rules there are no two rules with the same left-hand-side and different right-hand-side so that for any configuration at most one transition applies.

We modify the above definition by allowing more general transition rules which will accept replication of the same value in different registers. This does not affect the computational power of the model (see Lemma 4.1.2.1 below) but makes the design of such automata for various recognition problems much more natural and easier. Similar modifications (in more general setting) have appeared in [15; 18].

We define *modified* two-way k-register automata by adding to the definition above two extra types of transition rules:

Definition 4.1.2. *[41; 55] A non-deterministic two-way k-register automaton over an infinite alphabet D is a tuple* (Q, q_0, F, τ_0, P) *where* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $F \subseteq Q$ *is the set of final states,* $\tau_0 : \{1,...,k\} \to D \cup \{\triangleright, \triangleleft\}$ *is the initial register assignment and* P *is a finite set of transitions:*

1) $(i, q) \to (q', d)$ (If a current state is q and the observed symbol on the tape equals to a value in the register i then enter the state q' and move along the string according to the specified direction d where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$)

2) $q \to (q', i, d)$ (If a current state is q and the observed symbol on the tape does not equal to any value held in registers then enter the state q', copy the current symbol to a specified register i and move along the string according to the specified direction d where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

3) $(i, q) \to (q', j, d)$ (If a current state is q and the observed symbol equals to a value in the register i then enter the state q', copy the current symbol to a register j and move along the string according to the specified direction d where $i, j \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

4) $q \to (q', d)$ (If a current state is q and the observed symbol does not equal to any

value held in registers then enter the state q' and move along the string according to the specified direction d where $q, q' \in Q$ and $d \in \{stay, left, right\}$).

Deterministic and co-non-deterministic register automata as well as the language accepted by automata are defined in the standard way.

In the following lemma we show that the new added rules does not affect the computational power of original model.

Lemma 4.1.2.1. *The models of original register automaton and modified register automaton over an infinite alphabet are equivalent.*

Proof. Let M_1 be a register automaton over an infinite alphabet and M_2 be a modified model of register automaton over an infinite alphabet. Since M_2 contains all rules of M_1 (i.e. M_2 is a generalisation of M_1). Then any computations on M_1 can be simulated by M_2 in a straight forward way. That is the rules of M_1 coincide with the first two rules of M_2. Now to prove that the converse statement holds we will show that the two extra types of rules of M_2 can be simulated by M_1.

The rule $(i, q) \rightarrow (q', j, d)$ of type 3 of M_2 can be simulated in M_1 by using the following construction.

The state of the registers of M_2, that is a sequence of not necessarily different values $R = [r_1, r_2, \ldots, r_k]$ is represented in the simulating automaton as a pair:

- the set of unique values $U = \{r_1, r_2, \ldots, r_{k+1}\}$, and
- the surjective mapping $\phi : \{1, \ldots, k\} \rightarrow U$

The content of U is kept in the registers and since the mapping ϕ is finite, it can be kept in the finite state control. Now it is straightforward to simulate the effects of type 3, in terms of pairs U, ϕ as follows:

The rule $(i, q) \rightarrow (q', j, d)$ of type 3 is replaced by $(\phi(i), [q, \phi_x]) \rightarrow ([q', \phi_y], \phi_y(j), d)$ where x and y correspond to the current index of a mapping at states q and q' respectively. We update ϕ_x by changing $\phi(j)$ to be equal to $\phi(i)$.

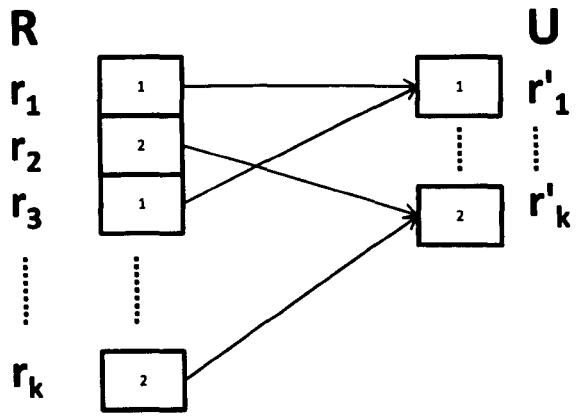

Figure 4.1: - Simulation of rule 3 in M_2 by M_1

The rule $q \to (q', d)$ of type 4 in M_2 can be simulated by adding one extra dummy register $k+1$ and replacement of rules of M_2 of the form $q \to (q', d)$ by either $(k+1, q) \to (q', d)$ or $q \to (q', k+1, d)$. If a value of the register $k+1$ is equal to the observed symbol then the rule $(k+1, q) \to (q', d)$ is applicable otherwise the rule $q \to (q', k+1, d)$ is applicable.

In terms of the data structure proposed for the simulation of rule 3, the rule of type 4 can implemented in the following way:

Let l be a register in M_2 where $l \in \{1, \ldots, k\}$ such that $\phi(l) = k+1$, if a value of the register $k+1$ is equal to the observed symbol then the rule $(\phi_x(l), [q, \phi_x]) \to ([q', \phi_y], d)$ is applicable otherwise the rule $[q, \phi_x] \to ([q', \phi_y], \phi_y(l), d)$ is applicable.

□

In Definition 4.1.2 we defined register automata as a model of computation for languages over an infinite alphabet. Another option is to define register automata as a model of computation for data words. A data word is a finite sequence of $\Sigma \times \mathbb{N}$, where Σ is a finite set of labels and $\mathbb{N}$ is an infinite set data values.

In the context of Gauss words, the following definition is more convenient and thus it will be used throughout this thesis.

Definition 4.1.3. *A non-deterministic two-way k-register automaton A over an alphabet D where* $D = \Sigma \times \mathbb{N}$ *is a tuple* (Q, q_0, F, τ_0, P) *where* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $F \subseteq Q$ *is the set of final states,* $\tau_0 : \{1,...,k\} \to D \cup \{\triangleright, \triangleleft\}$ *is the initial register assignment and* P *is a finite set of transitions:*

1) $(s, i, q) \to (q', d)$ (If a current state is q and first component of the observed symbol on the tape is s and second component of the observed symbol equals to a value in the register i then enter the state q' and move along the string according to the specified direction d where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$)

2) $(s, q) \to (q', i, d)$ (If a current state is q and first component of the observed symbol on the tape is s and second component of the observed symbol does not equal to any value held in registers then enter the state q', copy the current symbol to a specified register i and move along the string according to the specified direction d, where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

3) $(s, i, q) \to (q', j, d)$ (If a current state is q and first component of the observed symbol on the tape is s and second component of the observed symbol equals to a value in the register i then enter the state q', copy the current symbol to a register j and move along the string according to the specified direction d where $i, j \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

4) $(s, q) \to (q', d)$ (If a current state is q and first component of the observed symbol on the tape is s and second component of the observed symbol does not equal to any value held in registers then enter the state q' and move along the string according to the specified direction d, where $q, q' \in Q$ and $d \in \{stay, left, right\}$).

Given a data word $w = w_i \dots w_n$ where $w_i = (a_i, b_i)$ with $a_i \in \Sigma$ and $b_i \in \mathbb{N}$, a configuration of A on $\triangleright\ w\ \triangleleft$ is a tuple $[j, q, \tau]$ where $0 \leq j \leq n+1$ is a current position of the head in the word, $q \in Q$ is the current state, and $\tau : \{1, \cdots, k\} \to D \cup \{\triangleright, \triangleleft\}$ is the current register assignment. The initial configuration is $[1, q_0, \tau_0]$. A configuration $[j, q, \tau]$ with $q \in F$ is accepting. As usual, if automaton is able ever to reach a state

$q \in F$, it accepts the word, otherwise the word is rejected. The set of all accepted words forms a language recognisable by an automaton.

4.1.3 Pebble automata

As an alternative model of automata over infinite alphabet, *pebble automata*(PA) were introduced in [50] and further studied in [55]. We follow the definitions in [55]. In this model, instead of registers, finite state machines are equipped with the finite set of pebbles which can be placed on the input string and later lifted following the *stack* discipline. That means pebbles are numbered from 1 to k and pebble $i+1$ can only be placed when pebble i has already been placed on the string and vice-versa, pebble i can only be lifted if $i+1$ is not on the string. Further assumption is that the pebble with the highest number acts as a head, so an automaton has an access to the symbol of the string under such a pebble and to the information on which other pebbles are located at the same position. The transition of pebble automata depends on the following: the current state, the pebbles placed on the current position of the head, and the equality type of the data values under the placed pebbles. The effect of the transition is the change of a state, movement of the head and, possibly, removal of the head pebble, or placement of the new pebble. A new pebble is placed at the position of the most recent pebble and pebble $i-1$ becomes the current head when pebble i is removed.

As usual acceptance of a word is defined as reachability of one of the final states.

Definition 4.1.4 ([55]). *A nondeterministic two-way k-pebble automaton A over an infinite alphabet D is a tuple (Q, q_0, F, T) where*

- *Q, $q_0 \in Q$ and $F \subset Q$ are a finite set of states, the initial state, and the set of final states, respectively; and*
- *T is a finite set of transitions of the form $\alpha \to \beta$*
  - *α is of the form (i, s, P, V, q) or (i, P, V, q), where $i \in \{1, \cdots, k\}$, $s \in D \cup \{\triangleright, \triangleleft\}$, $P, V \subseteq \{1, \cdots, i-1\}$ and*

- β *is of the form* (q, d) *with* $q \in Q$ *and* $d \in \{$*stay, left, right, place-new-pebble, lift-current-pebble*$\}$.

Given a word $w = d_1...d_n \in D^*$, a configuration of A on $\triangleright\ w\ \triangleleft$ is a triple $[i, q, \theta]$ where $i \in \{1, \cdots, k\}$, $\theta : \{1, \cdots, i\} \to \{0, 1, \cdots, n, n+1\}$. The function θ defines the position of the pebbles and is called pebble assignment. The initial configuration $[1, q_0, \theta_0]$, with $\theta_0(1) = 1$. A configuration $[i, q, \theta]$ with $q \in F$ is called an accepting configuration.

Let $d_0 = \triangleright$, $d_{n+1} = \triangleleft$ and $d_i = a_i \in w$. A transition $(i, s, P, V, p) \to \beta$ applies to a configuration $[j, q, \theta]$ if

1. $i = j$ and $p = q$,
2. $V = \{l < i | d_{\theta(l)} = d_{\theta(i)}\}$,
3. $P = \{l < i | \theta(l) = \theta(i)\}$, and
4. $d_{\theta(i)} = s$.

A transition $(i, P, V, q) \to \beta$ applies to a configuration $[j, q, \theta]$ if the first three conditions hold and no transition of the form $(i, s, P, V, q) \to \beta$ applies to $[j, q, \theta]$.

Intuitively, a transition $(i, s, P, V, p) \to \beta$ applies to a configuration if pebble i is the current head, p is the current state, V is the set of pebbles that see the same symbol as the top pebble, P is the set of pebbles that sit at the same position as the top pebble, and the current symbol seen by the top pebble is s.

4.1.4 Linearly bounded memory automata

In all models above the input can be thought of as given on the input tape which can only be read, but not written on. Linearly bounded memory automata (LBMA) are an extension of register automata with the input tape. The automaton can read and write in the tape cells symbols of an infinite alphabet. The input is given on the initial part of the tape and for the input size n, the size of the tape is assumed to be $O(n)$,

i.e. linearly bounded. Types of rules of LBMA include all types of rules of (modified) RA and additional rules allowing us to write on the tape. For every form $L \rightarrow (\ldots)$ of rules of the (modified) RA model the following is a form of rule for LBA: $L \rightarrow (\ldots, l)$, where $l \in \{1, \ldots k\}$. The effect of application of the latter is the same as of the former, plus the automaton *writes the content of the register* l in the current position on the tape before possible head movement.

Definition 4.1.5. *A deterministic linearly bounded memory automaton over an infinite alphabet D is a tuple* (Q, q_0, F, τ_0, P) *where* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $F \subseteq Q$ *is the set of final states,* $\tau_0 : \{1,\ldots,k\} \rightarrow D \cup \{\triangleright, \triangleleft\}$ *is the initial register assignment and* P *is a finite set of transitions:*

1) $(i, q) \rightarrow (q', d)$ (If a current state is q and the observed symbol on the tape equals to a value in the register i then enter the state q' and move along the string according to the specified direction d where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$)

2) $q \rightarrow (q', i, d)$ (If a current state is q and the observed symbol on the tape does not equal to any value held in registers then enter the state q', copy the current symbol to a specified register i and move along the string according to the specified direction d, where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

3) $(i, q) \rightarrow (q', j, d)$ (If a current state is q and the observed symbol equals to a value in the register i then enter the state q', copy the current symbol to a register j and move along the string according to the specified direction d where $i, j \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).

4) $q \rightarrow (q', d)$ (If a current state is q and the observed symbol does not equal to any value held in registers then enter the state q' and move along the string according to the specified direction d, where $q, q' \in Q$ and $d \in \{stay, left, right\}$).

5) $(i, q) \rightarrow (q', d, l)$ (If a current state is q and the observed symbol on the tape equals to a value in the register i then enter the state q', move along the string according

to the specified direction d where $i \in \{1,..,k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$ and write the content of register l in the current position on the tape).

6) $q \rightarrow (q', i, d, l)$ (If a current state is q and the observed symbol on the tape does not equal to any value held in registers then enter the state q', copy the current symbol to a specified register i, move along the string according to the specified direction d, where $i \in \{1,..,k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$ and write the content of register l in the current position on the tape).

7) $(i, q) \rightarrow (q', j, d, l)$ (If a current state is q and the observed symbol equals to a value in the register i then enter the state q', copy the current symbol to a register j, move along the string according to the specified direction d where $i, j \in \{1,..,k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$ and write the content of register l in the current position on the tape).

8) $q \rightarrow (q', d, l)$ (If a current state is q and the observed symbol does not equal to any value held in registers then enter the state q', move along the string according to the specified direction d, where $q, q' \in Q$ and $d \in \{stay, left, right\}$ and write the content of register l in the current position on the tape).

Given a word w delimited by symbols $\triangleright, \triangleleft$ on the input tape, A configuration of an automaton on w is a tuple $[j, q, \tau, \rho]$ where j denote a position of a letter of w or the position of either $\triangleright$ or $\triangleleft$, $q \in Q$, $\tau : \{1, ..., k\} \rightarrow D \cup \{\triangleright, \triangleleft\}$ and ρ is the current content of the tape. The function $\rho : \mathbb{N} \mapsto D \cup \{\triangleright, \triangleleft\}$ is a partial function with domain of ρ being a fragment $[1, \cdots, m]$ of $\mathbb{N}$. The initial configuration $[0, q_0, \tau_0, \rho_0]$ with $\rho_0(0) = \triangleright$, $\rho_0(1) = w_1 (1 \leq i \leq n)$ and $\rho_0(n+1) = \triangleleft$. A configuration $[i, q, \tau, \rho]$ with $q \in F$ is called an accepting configuration.

As usual, if automaton is able ever to reach a state $q \in F$, it accepts the word, otherwise the word is rejected. The set of all accepted words forms a language recognisable by an automaton.

4.1.5 Turing Machine

A deterministic Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \tau_0, q_0, \#, F)$ where Q is a set of states, Σ is the input alphabet, Γ is the tape alphabet, τ_0 is the partial transition function, $\# \in \Gamma$ is a symbol called blank, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of final states.

The transition function for the Turing machine is given by $\tau_0 : Q \to Q \times \Gamma \times \{L, R\}$. This means when the machine is in a given state (Q) and reads a given symbol (Γ) from the tape, it replaces the symbol on the tape with some other symbol (Γ), goes to some other state (Q), and moves the tape head one square left (L) or right (R).

A configuration of a Turing machine requires the state the Turing machine is in, the contents of the tape, and the position of the tape head on the tape. This is written as a string of the form $x_i...x_j q_m x_k...x_l$ where the x's are the symbols on the tape, q_m is the current state, and the tape head is on the square containing x_k (the symbol immediately following q_m).

4.2 Register automata and classical complexity

In this section we examine the capability of a register automata to handle some useful operations required for checking knot properties. We show that over a class of *bounded languages* including all languages of our interest register automata can simulate the behaviour of a counter machine and that of a pebble automata. Further in Sections 4.2.3 and 4.2.4 we demonstrate generic results on the mutual simulations between logspace bounded classical computations (over finite alphabets) and register automata working over infinite alphabets.

4.2.1 Simulation of counters by register automata

In this Subsection we explain how a two-way register automata with k registers on the input with t distinct symbols can simulate k counters bounded by t.

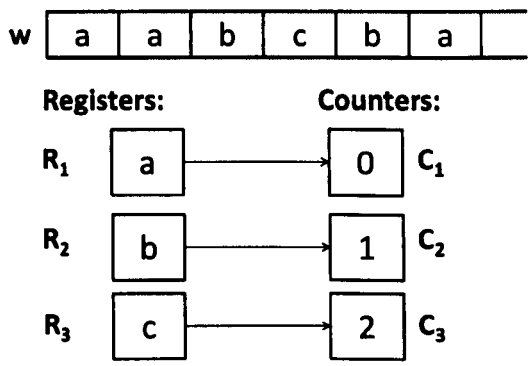

Figure 4.2: - Modelling of counters by registers in a register automaton model

Let us assume that a word on an input tape has at least t distinct symbols. The value of a counter stored in a register i will corresponds to the number of distinct symbols from the beginning of the word till the position of the first appearance of symbol stored in the register (for for an illustration, see Figure 4.2). Then we can increase (decrease) the value by updating the register with the next (previous) symbol on the string that will appear for the first time. Counter i is equal to zero if the value stored in the register i is the first symbol on the input tape. If we use k registers then we can store k counters bounded by t, where t is a number of distinct symbols on the input tape.

4.2.2 Simulation of pebble automata by register automata

As expressive power concerned, in general pebble automata are incomparable with register automata [55]. We will show, however, in the following proposition that over a class of *bounded languages*, including all languages of our interest, PA can be effectively simulated by RA.

Proposition 4.2.1. *If an n-bounded language over infinite alphabet D can be recognised by a k-pebble automaton, then it is also recognised by a $k+2$-register automaton.*

Proof. Let $PA = (Q, q_0, F, T)$ be a pebble automaton, we will first show how to represent the configuration $[i, q, \theta]$ of PA where $i \in \{1, \cdots, k\}$, $q \in Q$ and $\theta : \{1, \cdots, i\} \rightarrow$

$\{0, 1, \cdots, |w| + 1\}$, on the input word $w = d_1, \cdots, d_{|w|}$ by a configuration of a register automaton. Then we will show how to simulate the transition rules of PA on RA.

For the purpose of modelling PA, we define a register automaton $RA = (\tilde{Q}, \tilde{q_0}, \tilde{F}, \tau_0, \tilde{T})$ where its finite set of states $\tilde{Q} = Q^{aux} \times Q \times Q_n^k$ where Q^{aux} is a finite set of auxiliary states (for controlling the executions of general subroutines needed to check certain conditions on the transition rules), $Q_n^i = \{0, 1, \cdots, n\}$ (for keeping a reference to the number of previous occurrences of the symbol stored in register i) and n is a constant of the bounded language. The number of registers of RA is the number of pebbles of PA. Before we complete the definition of RA, let us explain the representation of the configuration of PA in a configuration of RA.

Let $\gamma = [i, q, \theta]$ be a configuration of PA on the input word w. Then the configuration $[j, q', \tau]$ of RA encodes γ if:

- $q' = (s, q, t_1, \cdots, t_k) \in \tilde{Q}$
- $\tau(l) = d_{\theta(l)}$, *for* $1 \leq l \leq i$
- $t_l = |\{r | r < \theta(l) \text{ and } d_r = d_{\theta(l)}\}| + 1$, *for* $1 \leq l \leq i$
- $t_l = 0$, *for* $i < l \leq k$

It easy to see that the definition of the notion of the encoding does not depend on j. To explain the above encoding the position of pebble i is determined as follows:

Suppose that pebble i is placed on the symbol $x \in D$ from the word $u \cdot x \cdot v$, where $u, v \in D^*$. Then the position of this pebble can be uniquely represented by a pair (x, t_x), where x is a symbol marked by pebble i and $t_x = |u|_x$, i.e. the number of occurrences of a symbol x in a word u (that is to the left of x). Thus, in order to simulate a pebble we need one register (for storing x) and a finite number of states $0, \cdots, n$ (for keeping t_x in a state space) since t_x is bounded by a constant n.

If RA is in state $\tilde{q}$ where $\tilde{q} = (s, q, t_1, \cdots, t_k) \in \tilde{Q}$ then placing a pebble i on a symbol x will correspond to storing x in register i and updating the states of t_i accordingly and lifting a pebble i will correspond to updating the state t_i to 0. Under

this encoding the top most pebble on a input correspond to the rightmost state among $t_1, \cdots, t_k$ with a non-zero value.

The initial configuration $\gamma_0 = [1, q_0, \theta_0]$ with $\theta_0(1) = 1$ of PA on the input word w is represented by the configuration $[1, \tilde{q_0}, \tau_0]$ of RA where $\tau_0(1) = d_1$, $\tau_0(k+1) = \triangleright$, $\tau_0(k+2) = \triangleleft$ and $\tilde{q_0} = (s, q_0, 1, 0, \cdots, 0)$. $\tilde{F}$ is the set of accepting states of RA where $\tilde{F} = \{(s, q, t_1, \cdots, t_k) | (s, q, t_1, \cdots, t_k) \in \tilde{Q} \text{ and } q \in F\}$.

Next we will explain how a transition rule of PA can modelled by the transition rules $\tilde{T}$ of RA. To model the effect of transitions rule transition rule $(i, P, V, p) \rightarrow (q, d)$ of PA (meaning, if pebble i is the current head, p is the current state, P is the set of pebbles that sit at the same position as the top pebble, V is the set of pebbles that see the same symbol as the top pebble, and the current symbol seen by the top pebble is x), RA should be able to execute the transition rules $\tilde{T}$ by checking the conditions of the rule described in first four steps and perform the updates in steps 5 and 6 as follows:

1. i is the current head, that is $t_l = 0$, *for* $i < l \leq k$ and *for* $1 \leq l \leq i$ where $0 < t_l \leq n$.

2. $P = \{l | l < i \text{ and } \tau(l) = \tau(i) \text{ and } t_l = t_i\}$, *for* $1 \leq l < i$

3. $V = \{l | l < i \text{ and } \tau(l) = \tau(i)\}$, *for* $1 \leq l < i$

4. $\tilde{p} = (s, p, t_1, \cdots, t_k) \in \tilde{Q}$ p is a state of the modelled PA, that is a current state of RA $(s_0, p, t_1, \cdots, t_k) \in \tilde{Q}$ for some $s_0, t_1 \cdots, t_k$

5. $\tilde{q} = (s_0, q, t'_1, \cdots, t'_k) \in \tilde{Q}$, the state of RA should become $(s'q, t'_1, \cdots, t'_k)$ for some $t'_1, \cdots, t'_k$

6. $d = \{stay, left, right, place{-}new{-}pebble, lift{-}current{-}pebble\}$ can be modelled as follows

   (a) $d = left$ then $\tau(i) = d_{\theta(i-1)}$.

   (b) if $d = right$ then $\tau(i) = d_{\theta(i+1)}$

(c) if $d = stay$ then $\tau(i) = d_{\theta(i)}$

(d) if $d = place - new - pebble$ then $t_{i+1} = t_l$ where $t_l = 0$, $for\ i < l \leq k$ and $0 < t_l \leq n\ for\ 1 \leq l \leq i$.

(e) if $d = lift - current - pebble$ then $t_{i-1} = t_l$ where $t_l = 0$, $for\ i < l \leq k$ and $0 < t_l \leq n\ for\ 1 \leq l \leq i$.

So far we have shown that how to model a configuration and transition rules of PA by RA. Now to show that RA accepts the same language as PA, we model an accepting run for PA by RA depicted in Figure 4.3 as follows:

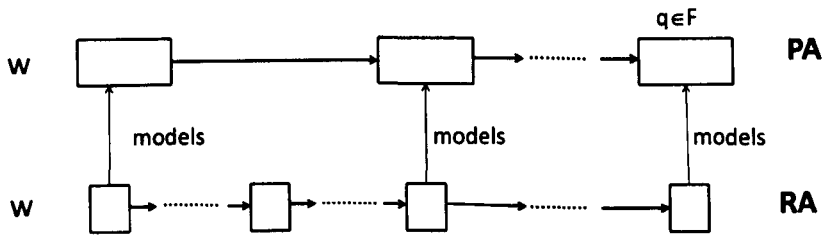

Figure 4.3: - Modelling an accepting run for PA by RA

Starting from an initial configuration, we launch a sequence of intermediate subroutines to obtain a new configuration that models the configuration in PA which resulted from application of one transition rule to initial configuration. We do the same as in the previous step for each other transition until the final configuration to modelled is reached, and if the sequence is an accepting run for PA then there is an acceptance sequence for RA and vice versa.

□

In the next section we will show that if a language L over an infinite alphabet is recognisable by register automata then the encoding of L over a finite alphabet can be recognised by a Turing machine with log-space memory. This results is quite interesting as it will allows to translate our results on recognisability by automata in the classical settings.

Definition 4.2.1. *We define the following classes of languages in terms of their recognisability by the appropriate computational device.*

- *DRA is the class of languages over an infinite alphabet recognisable by a deterministic register automata.*
- *Co-NRA is the class of languages over an infinite alphabet recognisable by a Co-non-deterministic register automata.*
- $\mathcal{L}$ *is the class of languages recognisable by a deterministic Turing machine in log-space memory.*
- $\mathcal{NL}$ *is the class of languages recognisable by a non-deterministic Turing machine in log-space memory*
- $Co-\mathcal{NL}$ *is the class of languages whose complements are in* $\mathcal{NL}$.

4.2.3 RA to $\mathcal{L}$

We will show that if a language L over the alphabet $\Sigma \times D$ is acceptable by two-way register automata then the encoding of L over a finite alphabet can be accepted by a Turing machine in log space memory. Let us first define the encoding of L over a finite alphabet as follows.

Let L be a language over the alphabet $\Sigma \times D$. For any pair of symbols (x, y) of a word $w \in L$ we denote by $ord_w(y)$ (or $ord(y)$ if w is understood) the number of distinct symbols of the second component in w to the left of the first occurrence of y in w. So, for example, if $w = (O,a)(O,c)(U,c)(U,b)(U,a)(O,b)$ then $ord_w(a) = 0$, $ord_w(c) = 1$ and $ord_w(b) = 3$. Let $\lceil x \rceil : \mathbb{R}^* \rightarrow \mathbb{N}$ where x is rounded up to the nearest natural number, we define $\tau : \Sigma \rightarrow \{0,1\}^k$ as a mapping from the letters of finite alphabet to their binary encoding where $k = \lceil log(|\Sigma|) \rceil$.

If ϕ is a mapping from natural numbers into their binary encoding, $\phi(i) : \mathbb{N} \rightarrow \{0,1\}^*$, then for any word $w = (a_1, b_1) \cdots (a_n, b_n) \in L$ where $a_i \in \Sigma$ and $b_i \in D$ for each $i \in \{1, \cdots, n\}$, the mapping $\psi(w) : (\Sigma \times D)^* \rightarrow \{0,1,\#\}^*$ is an encoding of a

word w, where each binary encoding of (a_i, b_i) is separated by a special symbol #:

$$\psi(w) = \#\tau(a_1)\phi(ord(b_1))\#\tau(a_2)\phi(ord(b_2))\#, ..., \#\tau(a_n)\phi(ord(b_n)).$$

Thus a language $L_{finite} = \{\psi(w)|w \in L\}$ is an encoding of L over the finite alphabet $\{0, 1, \#\}$ and L_{binary} is a natural encoding of L_{finite} over $\{0, 1\}$ alphabet, where $0 \to 00$, $1 \to 01$ and $\# \to 11$.

Proposition 4.2.2. *If L is recognisable by a finite register automata then L_{finite} is recognisable by a Turing machine in* log *space memory.*

Proof. Let L denotes the language accepted by a finite register automaton A with r registers. Let us show that the language L_{finite} is recognisable by a Turing machine M in log space memory. Let $w = (a_1, b_1) \cdots (a_n, b_n) \in L$ and $|w| = n$, then w consists of no more then n different symbols, so the length of the binary encoding of a_i is no more than $\lceil \log(|\Sigma|) \rceil$ and the length of the binary encoding of b_i is no more than $\log n$, i.e. $|\tau(a_i)| \leq k$ where $k = \lceil log(|\Sigma|) \rceil$ and $|\phi(ord(b_i))| \leq \log n$. We design a Turing machine M that can mimic all the computations of A by keeping the value of registers of A on its working memory tape. The operation of storing a symbol x from the infinite alphabet in a register in A can be simulated in M by storing the finite encoding of $ord(x)$ on the working memory. The storing procedure in M is organised by fixing the length of the binary encoding of each symbol to $\log n$ and adding a # at the end of each binary word to distinguish between the values held in the registers. The finite state control in M will correspond to the finite state control in A and the content of r registers in A will be stored on the working memory tape in M, which require $r \log(n + k)$ cells and r is a constant. The only two operation of register automaton are: to store a symbol in a register and to compare the register value with a symbol on a tape will not require any extra space apart from $r \log(n + k)$ cells. □

4.2.4 $\mathcal{L}$ to RA

For a data word $w = (a_1, b_1), \cdots, (a_{|w|}, b_{|w|})$ over the alphabet $\Sigma \times D$, we define its variability $v(w)$ as the number of distinct symbols of the second component (from D) in w. For a language L and an integer function $f(n)$ we say that variability of L is of the order $f(n)$ iff $min_{w \in L, |w|=n} v(w) \geq f(n)$, i.e. $f(n)$ is a lower bound of variabilities of words of length n in L.

Lemma 4.2.4.1. *Computations of a Turing Machine on a work tape T of size $c \cdot log(k)$ over a binary alphabet can be simulated by Register Automata model on an input string S, where $v(S)=k$.*

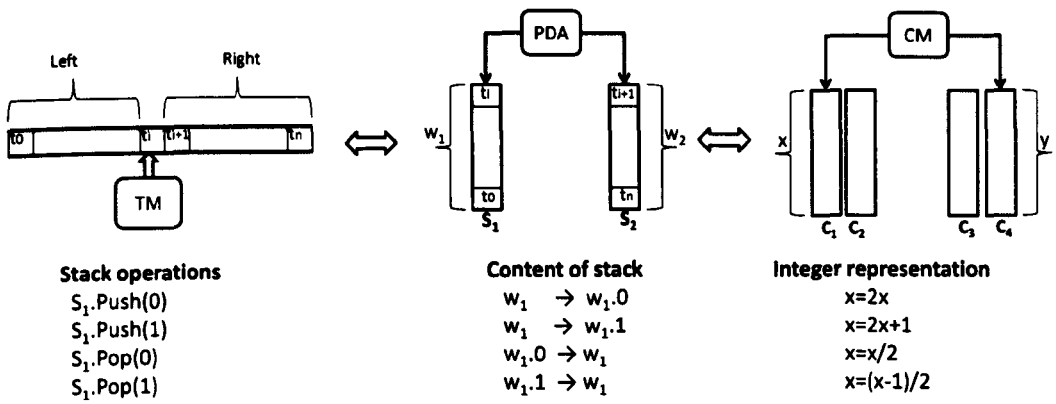

Figure 4.4: The schema for simulating the operations of a Turing machine (TM) by a two pushdown stacks and representing a binary word on a stack by a counter machine (CM) as an integer -

Proof. Assume that a head H is on a work tape T at the position $T(i)$. First all operations under the tape head such as rewriting, checking current symbol on a tape and head moves can be simulated by operations on two pushdown stacks, where first stack keeps the front part of T: $T(0) \ldots T(i)$ and the second part of T, is stored in the second pushdown stack with $T(i+1)$ on the top [51]. Also it is easy to see that a binary word on a stack can be represented by an integer x, where empty stack corresponds to 1 value, push a 0 onto the top of a stack corresponds to $x \mapsto 2x$ and pushing a 1 corresponds to $x \mapsto 2x + 1$; checking the top symbol can be done by checking

divisibility by 2 and popping off a 1 or 0 can be done by operations $x \mapsto (x-1)/2$ or $x \mapsto x/2$ respectively(see Figure4.4 for an illustration). Further we notice that the register automata on an input with k distinct symbols can simulate any finite number of counters of size k^c for any constant c with the following main operations: increment, decrement, multiplication by 2, division by 2 and zero testing. Storing a symbol x in a register represents a value $ord(x)+1$ of a counter. The implementation of all required operations is straightforward albeit tedious. The operation of increment (decrement) by one can be implemented by moving forward to the first occurrence of the next (previous) distinct symbol on an input string [41]. Testing of a counter to be equal to 1 corresponds to checking whether a stored symbol appear as a first symbol of an input. Also it is well known that operation of multiplication (or division) by 2 can be implemented by a finite number of extra counters with increment and decrement operations and zero testing (or testing to be equal to 1) [51]. Thus, all operations for integer representations of pushdown stacks can be implemented by a register automaton. So since the virtual counters can store a value of size k^c and simulate two pushdown stacks of size $log(k^c)$ we have that the length of a work tape T which is simulated by register automaton on an input with k distinct symbols is bounded by $log(k^c)$. □

Theorem 4.2.1. *Given a language L with a variability of the order $f(n)$, if a finite projection L_{binary} of L belongs to SPACE(log(f(n))) then L can be recognised by a register automata.*

Proof. The binary code of each symbol in $w \in L$ is of a logarithmic size from the number of distinct symbols in w. So for any word of size n over an infinite alphabet, the length of the binary code for each symbol is no more than $log(n)$. Since the number of distinct symbols in any word $w \in L$ is at least $f(n)$, thus by the construction described above in Subsection 4.2.4.1 we can simulate virtual tape over a binary alphabet of a length $c \cdot log(f(n))$ for any integer constant c. Therefore register automaton can take any symbol y from an infinite alphabet on an input string and convert it into a finite binary representation in the virtual work tape over a binary alphabet, by counting the

number of previously appeared distinct symbols from the beginning of an input string and storing it on a virtual tape, i.e. converting y into a binary representation of $ord(y)$. Also by the same construction described in Subsection 4.2.4.1 we have that any extra memory of size $O(log(f(n))$ can be implemented by a finite number of registers on a language L with a variability of the order $f(n)$. So any computations over language L_{binary} that requires SPACE($log(f(n))$) can be implemented by a register automaton on a language L with a variability of the order $f(n)$. □

4.3 Automata and Gauss words

In this section we will show that for the purpose of Gauss words, a *1-way register automaton* even when equipped with the power of non-determinism is weak as it fails to recognise the characteristic property of a Gauss word (that is, for every symbol O_i in an input word there exists a corresponding symbol U_i and vice versa). However, considering a stronger variant we show that the languages of Gauss words whether signed or unsigned as well as non-trivial properties related to Gauss words such as the equivalence of two Gauss words in terms of cyclic shift and renaming of labels are recognisable by a *2-way deterministic register automata.*

4.3.1 The language of Gauss words

In the following propositions we show that the languages of signed and unsigned Gauss words are not recognisable by a non-deterministic 1-way register automata but they can be recognised by a deterministic 2-way register automata.

Proposition 4.3.1. *The languages of unsigned Gauss words (L_{UGW}) and signed Gauss words (L_{SGW}) are not recognisable by non-deterministic one-way register automata.*

Proof. We show the argument only for the case of L_{UGW}. With obvious modifications it works for L_{SGW} as well. The argument is not new and was used e.g. in [5] to show non-recognisability of some data languages by one-way register automata. Assume that

language L is recognisable by some one-way register automaton A with n registers. Consider the word

$$w = (U,1)(U,2)\dots(U,n+1)(O,1)(O,2)\dots(O,n+1) \in L.$$

The automaton A accepts this word. After reading first $n+1$ positions, there is at least one index value $i \in \{1,\dots,n+1\}$ which does not appear in any register of A. That means that automaton A also accepts a word $w' \notin L$ obtained from w by replacing (U,i) with $(U,i+1)$. This contradicts the assumption on A. □

Proposition 4.3.2. *The languages L_{UGW} and L_{SGW} are recognisable by deterministic 2-way register automata.*

Proof. We explain only the construction of a 2-way deterministic register automaton A which recognises L_{UGW}. With obvious modifications the automaton can be adapted to the case of L_{SGW}. Let w be a data word $(a_1,b_1)...(a_n,b_n)$ such that $a \in \Sigma = \{U,O\}$ and $b \in \mathbb{N}$. The automaton A reads the first symbol b_i and stores the value of b_i in some register, then it moves right then left along the word to compare the current symbol (a_j,b_j) with the value of b_i held in some register. If the symbol (a_j,b_j) where $b_i = b_j$ and $a_i \neq a_j$ is found and there are no further occurrences of b_i, then the automaton A moves right along the word and checks the next symbol. If the next symbol is equal to the end symbol then A moves to an accepting state. □

The main property of Gauss words that we checked in Proposition 4.3.2 is that for each U_i in w there is a unique corresponding O_i such that i occurs twice in w. To determine if a Gauss paragraph[1] is correct, we check the same property. Since Gauss paragraphs posses the same property as Gauss words, then by Proposition 4.3.2 the language of Gauss paragraphs (signed and unsigned) can also be recognised by a 2-way deterministic register automaton.

[1] A Gauss paragraph is disjoint set of Gauss words representing a link diagram, a formal definition is given in Definition 5.2.2 and Definition 6.2.1 for the cases of signed and unsigned respectively

4.3.2 Isomorphic Gauss words

In this section, we investigate the recognition problem of isomorphic Gauss words. That is Gauss words which are equivalent up to cyclic shift and renaming of labels.

The definitions for the equivalence of Gauss words in terms of cyclic shift and renaming of labels are described in Chapter 3, Section 3.5.1. Let u and v be two Gauss words, u and v are called isomorphic Gauss words iff $u \equiv^{cr} v$ (i.e. u and v are equivalent up to cyclic shift and renaming of labels).

Definition 4.3.1. *Let $\Sigma_{\#} = \{U, O\} \cup \{\#\}$, the language of isomorphic Gauss words L is defined as $L = \{u(\#, \alpha)v | u \equiv^{cr} v, (\#, \alpha) \in \Sigma_{\#} \times \mathbb{N}\}$.*

In the following proposition we show that the recognisability problem of isomorphic Gauss words is implementable by a two-way deterministic register automaton.

Proposition 4.3.3. *The language of isomorphic Gauss words can be recognised by 2-way deterministic register automata.*

Proof. Let $w = u\#_\alpha v$ where u and v are Gauss words separated by a α character. To check that the words u and v are equivalent up to cyclic shift and renaming of labels. The automaton required will need first to find all cyclic words of u and then compare each word with the word v. To check all cyclic words of u, the automaton will go through each symbol in u until the symbol α is found. It will record the first symbol of each cyclic word in a register and then move right until α is found. Once it reaches the symbol α the automaton will move back to the start symbol and then right until it reaches the symbol which marks the beginning of that cyclic word. Now to check the next cyclic word the automaton moves a step to the right, stores the new symbol in the register and then traverses the word u in the same way. If the next symbol is α then the automaton will terminate (uses the fact that no symbol repeats). To compare the symbols of each cyclic word in u with the word v, the automaton will check two conditions: it will check that the first component (O or U) of each symbol of the cyclic word is in the same order as that in v and that the number of symbols between each pair of the same labels is also the same.

To check the order of first component of each symbol of the cyclic word, the automaton starts at the beginning of each cyclic word and keeps information about the first component of the symbol and its label (data value) in some specified registers. The same information about the word v will be kept in other specified registers. The information about first component of each symbol will be used for comparison and the information about labels will be used to identify the position in which to return to later before checking next symbol.

For each cyclic word in u, if the first component of the symbol is matched with its corresponding symbol in v then the automaton checks next symbol and the information about the previous symbol that were held in the registers will be overwritten with new information about current symbol. This process is continued until all symbols of the cyclic word have been checked. However if there is a mismatch then the automaton will check next cyclic word otherwise it will move back to the beginning of the current cyclic word and then check the number of symbols between each pair of the same label.

To check the number of symbols between each pair of the same label, the automaton will require the use of counters. The counters can be implemented as described in Section 4.2.1. The first counter will be used to identify the position of the first occurrence of each label. The value of this counter will be set to 0 at the beginning and it will be increased by 1 for each new label encountered for the first time whilst moving right along the word and decrease by 1 otherwise. The second counter will be used to count the number of symbols between a pair of the same label. The value of this counter will be set to 0 at the position of the first occurrence of each label and it will be increased by one for each encountered symbol in between the pair of the two labels. The automaton will go through each label i in the cyclic word and each label j in v by moving to the first occurrence of each label and then it will move right incrementing the value of the second counter by 1 for each encountered symbol until the second occurrence of the same label is reached. If the values of the counter for each label i in the cyclic word corresponds to that in v then the automaton moves to an accepting state otherwise it will check the next cyclic shift.

□

4.4 Summary

In this chapter we considered automata over infinite alphabets for the purpose of studying complexity of problems related to knots and demonstrated generic results on the mutual simulations between log-space bounded classical computations (over finite alphabets) and register automata working over infinite alphabets. our characterisation of languages recognisable by register automata is more general than one proposed in [55]. Non-trivial lower bounds for some knot problems are unknown and weak automata models are plausible candidates here to try. In opposite direction, knot theory provides a rich supply of natural problems formulated in terms of languages over infinite alphabets, and that, one may expect, will influence the development of the theory of such languages and related computational models.

In the next chapter we will apply automata over infinite alphabet to investigate the computational complexity for the recognition problem of planar signed Gauss words and Gauss paragraphs.

Chapter 5

Signed Planarity

In this chapter we investigate the descriptional complexity of knot theoretic problems and show upper bounds for the planarity problems of signed knot diagrams represented by signed Gauss words and signed link diagrams represented by signed Gauss Paragraph. For establishing the upper and lower bounds on recognition of knot properties, we study these problems in a context of automata models over an infinite alphabet.

The central problem that we study in this chapter is to determine whether a given signed Gauss word (paragraph) is *planar*, i.e. encodes a plane diagram of a classical knot (link) in $\mathbb{R}^3$.

In Section 5.1 we investigate the complexity of planarity problem of signed knot diagrams represented by signed Gauss words and demonstrate an upper bound by showing that it can be recognised by a deterministic two-way register automata simulating the algorithm presented in [7] and in Section 5.2 we consider the planarity problem of signed link diagrams represented by Gauss paragraphs and show that the language of planar signed Gauss paragraphs can be recognised by deterministic two-way register automata simulating the recently discovered linear time algorithm in [38].

Further in Section 5.3 we translate our results obtained in Sections 5.1 and 5.2 in terms of classical complexity and show that languages of planar signed Gauss words and planar signed Gauss paragraphs belong to the complexity log-space class $\mathcal{L}$.

5.1 Planarity of signed Gauss words

Every knot can be represented by a signed Gauss word, but not every signed Gauss word represents a classical knot in $\mathbb{R}^3$. For example, any attempt to reconstruct a knot diagram from the Gauss word $O_1^- O_2^- U_1^- O_3^+ U_2^- U_3^+$ inevitably leads to new (*virtual*) crossings which are not present in the Gauss word, see Figure 5.1.

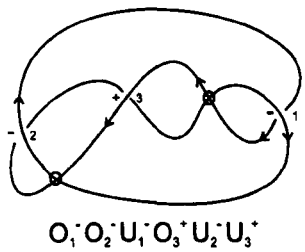

Figure 5.1: Virtual knot diagram - An example of a virtual (non-planar) knot diagram with 2 virtual crossings

The ambiguity in reconstructing the curve from the Gauss word depends on the choice of whether one curve crosses the other from right-to-left or left-to-right at a crossing. Such problem was resolved in [9] by labelling the letters in the Gauss word with signs leading to signed Gauss words. Based on the work of Carter in [9], Elton and Cairn presented an algorithm in [7] that deals with the planarity problem of signed Gauss words.

5.1.1 Cairns-Elton algorithm

The algorithm was presented in [7] to deal with the planarity problem of signed Gauss words. The encoding used to describe knots is different from the standard Gauss words, i.e. information about under-crossing and over-crossing is omitted and two opposite signs are associated with the pair of labels representing the same crossing. However, this encoding can be converted into standard Gauss words and vice versa.

To describe the implementation of the algorithm, we use the modified version of the algorithm presented in [8, Theorem 1'] and adopt the same notation.

5.1.1.1 Notation

Given a set $S = \{a_1, ..., a_k, a_1^{-1}, ..., a_k^{-1}\}$, an abstract Gauss word w is defined as a permutation of S. An abstract Gauss word can encode a knot diagram in the following way: label the crossings with letters $a_1, ..., a_k$. Choose an orientation for the curve and start traversing the curve from an arbitrary point x until the same point is reached for the second time. At each crossing a_i record a_i^{-1} if one passes through the crossing point a_i and the curve that one crosses is travelling from right to left or record a_i^{+1} otherwise. The positive superscripts are omitted.

Definition 5.1.1. *The value $\alpha_i(w)$ is a sum of the values of the superscripts (signs) of the set S_i (where S_i denote the letters appeared between the two occurrences of the letters a_i and a_i^{-1})* (mod 2).

Definition 5.1.2. *Let $\bar{S}_i = S_i \cup \{a_i, a_i^{-1}\}$ and set S_j^{-1} denote set S_j but with signs reversed. The value of $\beta_{i,j}(w)$ is a sum of the values of the superscripts of the letters in the intersection of $\bar{S}_i$ and S_j^{-1} taken modulo 2, for all $i, j \in \{1, ..., k\}$.*

Notice that the value attributed to each sign is ± 1. So the contribution it makes to the sum of values of $\alpha_i(w)$ or $\beta_{ij}(w)$ is 1 (mod 2) which is in fact similar to counting the number of letters, i.e. if the total number of letters between any pair a_i and a_i^{-1} is 0 (mod 2) then the sum of signs of the letters is also 0 (mod 2) and vice versa.

5.1.1.2 Implementation

Given the word w with $2k$ labels, the algorithm proceeds in two stages.

1. For all $i \in w$, check if $\alpha_i(w) = 0$ (mod 2) where $i = \{1, ..., k\}$.

2. For all $i, j \in w$, check if $\beta_{i,j}(w) = 0$ (mod 2) where $i, j = \{1, ..., k\}$.

If conditions 1 and 2 are satisfied then the algorithm returns "the word w is planar", otherwise if any of the conditions is not satisfied, the algorithm returns "the word w is non-planar".

The complexity of the algorithm has not been established neither in terms of time nor space. However, In terms of computational power of devices, we will demonstrate an upper bound by showing that the property expressed in this algorithm is recognisable by a deterministic two-way register automata.

Theorem 5.1.1. *The language of planar signed Gauss words can be recognised by a 2-way deterministic register automata.*

Proof. To check the value of $\alpha_i(w)$, the automaton keeps in the register the current index i of each positive letter a_i and goes through each element between a_i and a_i^{-1}. It will keep a count of the number of letters (mod 2) between a_i and a_i^{-1} in finite state control by alternating between odd and even states. If a_i^{-1} is reached and state is odd then the automaton halts, otherwise it moves to the right and checks the value of $\alpha_i(w)$ for the next positive letter a_i. If for all a_i the parity of $\alpha_i(w)$ is even the automaton proceeds to check the second condition.

To check the value of $\beta_{ij}(w)$, the automaton needs two registers to store the current indices of each a_i and a_j. Again the counting is done in finite state control by alternating between odd and even states iff for each symbol $a_k \in \bar{S}_i$, the inverse of a_k (a_k^{-1}) belongs to the set S_j (i.e this is equivalent to checking elements of the set S_j^{-1}). If the automaton reaches a_i^{-1} and state is odd then it terminates, otherwise it will continue to the check the next value. If for all $a_i, a_j \in w$, the parity of $\beta_{ij}(w)$ is even then the automaton moves to an accepting state.

□

Due to the fact that a knot diagram is defined as a link with one component, the planarity problem of signed Gauss words can be seen in general as a restriction case of the planarity problem of signed Gauss paragraphs. The above algorithm therefore is not applicable for links represented by Gauss paragraphs due to the fact that the algorithm was designed to deal with single Gauss words rather than with a set of Gauss words (for the case of links). However in the next section, we will consider a different algorithm designed for checking the planarity of signed Gauss paragraphs presented in

[38].

5.2 Planarity of signed Gauss paragraphs

The planarity problem of links represented by Gauss paragraphs is formulated in a similar way to the planarity problem of knot diagrams represented by Gauss words. The only difference is that for links the input is a set of cyclic words usually known as Gauss paragraph. Before we discuss the algorithm, we will begin with some definitions to define links and their encodings.

Definition 5.2.1. *A link is a smooth embedding (image) of several disjoint circles in* $\mathbb{R}^3$. *Each knot representing the image of one of these circles is called a link component.*

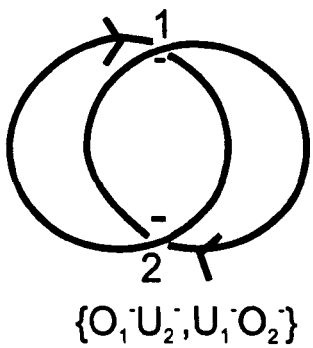

Figure 5.2: - An oriented link diagram with its corresponding Gauss paragraph

A link can be represented by a Gauss paragraph by encoding a link diagram in a combinatorial way. We associate a word to each component circle in the diagram. We record the crossing points with signs according to their order in the circle by travelling around the circle exactly once. The set of resulting words form a Gauss paragraph (see Figure 5.2). Any link diagram can be encoded by several Gauss paragraphs since there are five choices that can be made:

- Choosing a base point of each circle.

- Choosing an orientation of each circle.
- Relabelling of the crossings.
- Changing the signs for crossings.
- Permuting the order of the component circles.

To define signed Gauss paragraphs formally, we use the same notations as for the definitions of Gauss words.

Definition 5.2.2. *A signed Gauss paragraph W over the alphabet $\Sigma = \{O^+, O^-, U^+, U^-\} \times \mathbb{N}$ is a set of data words $\{w_1, ..., w_k\}$ where $k \geq 1$ such that $|w_1| + |w_2| \ldots + |w_k|$ is even and for every $i \in \mathbb{N}$ either*

- $|W|_{(U^+,i)} = |W|_{(O^+,i)} = |W|_{(U^-,i)} = |W|_{(O^-,i)} = 0$, *or*
- $|W|_{(U^+,i)} = |W|_{(O^+,i)} = 1$ *and* $|W|_{(U^-,i)} = |W|_{(O^-,i)} = 0$, *or*
- $|W|_{(U^-,i)} = |W|_{(O^-,i)} = 1$ *and* $|W|_{(U^+,i)} = |W|_{(O^+,i)} = 0$.

5.2.1 Kurlin algorithm

The main idea of the algorithm is to find the least genus[1] of the surface containing a knot diagram without virtual crossings encoded by a given Gauss paragraph. For this purpose the Euler characteristic χ [49] of the *combinatorial Carter surface* (a 2-cell complex[2] constructed from a Gauss paragraph where vertices correspond to the crossing labels, edges correspond to two consecutive labels and faces are found by successive left turns on each crossing as illustrated in Figure 5.4) [9] associated to the Gauss paragraph is computed as the number of faces (cycles) minus the number of edges plus the number of vertices. The Gauss paragraph includes the information required to reconstruct the Carter surface. We first derive a diagram (a 4-regular graph) in the

[1]This is the minimum number of handles that must be added to the sphere in order to embed the knot on the surface. The genus g is related to the Euler characteristics χ by the formula $\chi = 2 - 2g$.

[2]This is a topological space with cell structure consisting of 0-cells corresponding to vertices, 1-cells corresponding to edges and 2-cells corresponding to faces.

plane (not necessary embeddable in the plane) where the number of edges is the length of the Gauss paragraph, the number of vertices is the number of crossings (half the length of the Gauss paragraph) and the number of faces can be found by implementing traversal rules described in Definition 5.2.3. The Carter surface is obtained by gluing all faces of the diagram together.

5.2.1.1 Notation

Let $W = \{w_1, ..., w_k\}$ be a Gauss paragraph of length $2n$. That is $|w_1| + ... + |w_k| = 2n$. We associate a Carter surface $M\{w_1, ..., w_k\}$ with W as follows: Take n vertices labelled by $1, ..., n$ and connect vertices i, j by an edge with a mark (a,b) or (b,a) if one of the cyclic words $w_1, ..., w_k$ contains the ordered pair ab or ba of successive letters respectively, for some $a \in \{O_i^+, U_i^+, O_i^-, U_i^-, \}$, $b \in \{O_j^+, U_j^+, O_j^-, U_j^-, \}$ where $i, j \in \{1, ..., n\}$. We traverse a face in the resulting graph by travelling along an edge (a, b) encoding the direction of the path by $(a, b)_R$ if the letter a precedes b in the Gauss paragraph and by $(a, b)_L$ otherwise. After passing an edge we choose the next edge based on a set of traversal rules defined in Definition 5.2.3.

Example. *Let $W = \{O_1^- U_2^+, U_1^- O_2^+\}$ be a Gauss paragraph representing the link diagram in Figure 5.3. Then W generates the graph with 2 vertices connected by 4 edges $(O_1^- U_2^+), (U_2^+ O_1^-), (U_1^- O_2^+), (O_2^+ U_1^-)$ and 2 faces bounded by the edges $(O_1^- U_2^+)_R (O_2^+ U_1^-)_R (O_1^- U_2^+)_L (O_2^+ U_1^-)_L$ and $(O_1^- U_2^+)_L (O_2^+ U_1^-)_R (O_1^- U_2^+)_R (O_2^+ U_1^-)_L$.*

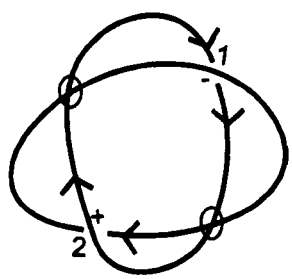

Figure 5.3: - A non-planar link diagram with two virtual crossings

5.2.1.2 Traversal rules

The traversal of a face containing a crossing i in the link diagram can be done by choosing an initial direction and turning left at each consecutive visited crossing starting from i. This global property of "turning left" can be defined by a deterministic set of traversal rules that take into account only local property of the current crossing and some finite information about the previously visited one. In general we have 8 cases since there are 2 types of crossings (positive and negative) and 4 directions from which we can approach each crossing, see Figure 5.4. From a topological point of view, we

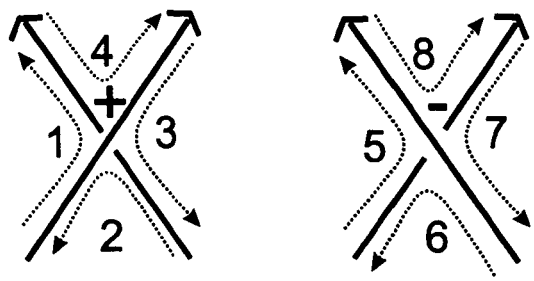

Figure 5.4: - Geometric interpretation of traversal rules for selecting faces

take the graph G defined by a Gauss paragraph and associate faces to cycles always turning left at every vertex, which leads to a surface. If the resulting surface is a 2-dimensional sphere whose Euler characteristic is 2 then the given Gauss paragraph is planar since we have embedded the graph into the plane. We follow the interpretation of the local rules for selecting cycles defined in [38], but present them here in slightly different notation, which is more appropriate for the design of a register automaton. A register automaton that is observing a current symbol S, needs to choose a correct symbol corresponding to the next crossing after turning (geometrically) to the left on a link diagram. In fact on the Gauss paragraph, it will correspond to finding S' that is the counterpart of S and then choosing a symbol which is either a left or right neighbour of S'.

For example, if S is O_i (U_i, respectively) with any sign $\varepsilon = \pm$ then we choose a neighbour of U_i (O_i, respectively) with same sign ε in the Gauss paragraph. Geometrically, taking the right neighbour in the Gauss paragraph is equivalent to going in the

direction of the orientation of the corresponding link diagram, while taking the left neighbour means moving in the direction opposite to the orientation. In order to define whether we need a neighbour from the left or from the right side we need to know the current type of the crossing which is S and the information about the previous choice of direction, i.e. whether S was chosen as a left or a right symbol.

Definition 5.2.3. *We define eight rules in the form* $(D, S) \Rightarrow (S', D')$, *where* $D, D' \in \{R, L\}$ *and* $S, S' \in \{U, O\} \times \mathbb{N} \times \{+, -\}$. *Each rule can be read as follows: if the current symbol* S *is reached via direction* D *then find* S' *(counterpart of* S*) and move one step in the specified direction.*

1. $(R,\ O_i^+) \Rightarrow (U_i^+,\ R)$
2. $(R,\ U_i^+) \Rightarrow (O_i^+,\ L)$
3. $(L,\ O_i^+) \Rightarrow (U_i^+,\ L)$
4. $(L,\ U_i^+) \Rightarrow (O_i^+,\ R)$
5. $(R,\ U_i^-) \Rightarrow (O_i^-,\ R)$
6. $(R,\ O_i^-) \Rightarrow (U_i^-,\ L)$
7. $(L,\ U_i^-) \Rightarrow (O_i^-,\ L)$
8. $(L,\ O_i^-) \Rightarrow (U_i^-,\ R)$

The first rule $(\mathrm{R}, O_i^+) \rightarrow (U_i^+, \mathrm{R})$ correspond to a curved dashed line labelled with a 1 in the first left-hand-side picture of Figure 5.4. The remaining 7 rules correspond to other 7 curved dashed lines with arrows showing 'the left turn'. The sign of a crossing does not change, while any overcrossing is replaced by an undercrossing and vice versa.

To make the rules reflect the geometric interpretation for traversing a face on a diagram by always turning left at each visited crossing, we need to draw a distinction between the search for locating the counterpart of some symbol and the move to the right (or to the left respectively) after locating the counterpart. That is, to locate the counterpart of a symbol, one can go through all symbols in the Gauss paragraph whereas for the second part, one can only move within the word that the counterpart symbol belongs to. So to identify the right-symbol of the last symbol or the left-symbol of the first symbol of some particular word, we need associate some extra symbols to mark the beginning and end of each word and two other symbols to mark the beginning and end of the Gauss paragraph.

Definition 5.2.4. *Let* $\Gamma = \{\triangleright, \triangleleft, \#, *\}$ *and* $W = \{w_1, w_2, ..., w_n\}$ *be a Gauss paragraph where* $n \geq 1$. *We obtain a new word* $w = \# \triangleright\ w_1\ \triangleleft\triangleright\ w_2\ \triangleleft, ..., \triangleright\ w_n\ \triangleleft\ *$ *by concatenating the set of words in* W *such that each word* $w_i \in W$ *is associated with the symbols* $\triangleright, \triangleleft \in \Gamma$.

Lemma 5.2.1.1. *A two-way deterministic register automaton can traverse all faces containing a crossing* i *in the Carter surface associated with a link diagram.*

Proof. We follow the rules defined in Definition 5.2.3. We can design a register automaton that keeps the finite information about its previous choice of direction (Right or Left) in its state space and chooses the Right or Left symbol of S' after observing the symbol S. It can also keep records on which rule was applied to the starting symbol S and will terminate the traversal of a face if the same rule will be applied for S again. The fact of the repetition corresponds to the completion of a cyclic path. In order to traverse all faces which are adjacent to a crossing i, we need to start from two different initial conditions associated with labels (O_i or U_i) and two different initial direction (Left or Right). □

Lemma 5.2.1.2. *Two-way deterministic register automata can compute the Euler characteristic of the Carter surface associated to a signed Gauss paragraph according to the construction described in Section 4.2.1.*

Proof. To compute the Euler characteristic we count the numbers of edges and vertices in the graph G represented by a signed Gauss paragraph. Geometrically G is the underlying graph of the link diagram encoded by the Gauss paragraph and all its vertices (crossings of the diagram) have degree 4 only. The number of vertices in G is the number of distinct symbols, while the number of edges is twice as much. Both values can be counted in a straightforward way. The number of faces attached to the graph in the combinatorial Carter surface can be counted by traversing G in the following way. The automaton goes sequentially through the list of vertices. For each vertex i it traverses (as described in Lemma 5.2.1.1) all adjacent faces and increases the counter by one for every face F not containing vertices with indices less than i. Also

the automaton counts how many times the crossing i is met during the traversal of faces adjacent to i. As soon as the value reaches 4 the automaton starts the traversal for the next crossing. The computation of the Euler characteristic χ is done by counting the values for edges, vertices and faces in individual counters and then by subtracting number of edges from the sum of the numbers of vertices and faces. Since the number of each value in counters is bounded by the number of distinct symbols, the computation can be done by the two-way deterministic register automaton. □

Theorem 5.2.1. *The language of planar signed Gauss paragraphs can be recognised by two-way deterministic register automata.*

Proof. Compute the Euler characteristics by the two-way deterministic register automaton. If the Euler characteristics χ is equal to 2, i.e. the combinatorial Carter surface is a sphere, then a signed Gauss word is planar [9; 38]. □

5.3 Complexity bounds of signed planarity

We have shown that in Chapter 4 Lemma 4.2.2 that if a language L over the infinite alphabet D is acceptable by two-way register automata then the encoding of L over a finite alphabet can be accepted by a Turing machine in log-space memory. As a consequence we have the following corollaries providing first known results in terms of classical space complexity for the planarity problems of signed Gauss words and signed Gauss paragraphs.

Corollary 5.3.1. *The language of planar signed Gauss words is in $\mathcal{L}$.*

Corollary 5.3.2. *The language of planar signed Gauss paragraphs is in $\mathcal{L}$.*

5.4 Summary

We have applied automata over infinite alphabets for studying complexity of problems related to knots. We have shown that the languages of planar signed Gauss words and

signed Gauss paragraphs can be recognised by deterministic two-way register automata. Therefore this result is final in the sense that the power of non-deterministic one-way register automata is not even enough to recognise whether an input is a Gauss word.

We have also shown that planarity of signed Gauss words and signed Gauss paragraphs can be recognised in deterministic logarithmic space on classical computational models. In the next chapter, we consider the languages of planar unsigned Gauss words and unsigned Gauss paragraphs.

Chapter 6

Unsigned Planarity

In this chapter we investigate the complexity of knot theoretic problems and show upper bounds for planarity problem of unsigned knot and link diagrams.

A knot can be encoded by a string of symbols O_i's (over-crossing i) and U_i's (under-crossing i) where i is a label for some crossing such that each crossing has a unique label. The double occurrence sequence of labels was first described by Gauss in [23] and this string of symbols is known as a *Gauss word.*

The question of characterisation of "true", or planar Gauss words was posed by Gauss himself [23] and was eventually resolved by Nagy in [53]. Since then there has been proposed many criteria and algorithms both for recognition of signed [7; 38] and unsigned [8; 16; 17; 37; 46; 48; 57; 60; 60; 61; 62] Gauss words. The questions of computational complexity of the proposed algorithms were rarely explicitly addressed with notable exceptions being [38] where linear time algorithm for the signed case is proposed, and in [61] where a linear time complexity for unsigned case is established and compared with earlier quadratic bounds in [57].

In Chapter 5 we proposed to evaluate the complexity of problems of recognising knot properties in terms of the computational power of devices needed to recognise the properties. Following the proposal we demonstrated lower and upper bounds for recognisability of knot properties in terms of various automata models over *infinite* alphabets. The infinite alphabet appeared naturally due to the fact that the num-

ber of crossings in knots is unbounded. The main property addressed was planarity problem for signed Gauss words and signed Gauss paragraphs. We have shown that languages of planar signed Gauss words and signed Gauss paragraphs can be recognised by deterministic register automata working over infinite alphabets.

In this chapter, we continue this line of research focussing mainly on the *unsigned case*. Our contribution is as follows:

1. In Section 6.1 we provide an analysis of Cairns-Elton algorithm for planarity of unsigned Gauss words and show that it is implementable by co-non-deterministic register automata.

2. In Section 6.2 we show that planarity of unsigned Gauss paragraph is recognisable by a linearly bounded memory automata simulating Kauffman algorithm [37].

3. In Section 6.3 we further show that Cairns-Elton algorithm is implementable in $\mathcal{SL}$ (symmetric logspace) and therefore in $\mathcal{L}$ [59]. It follows that planarity of unsigned Gauss words is recognisable by deterministic register automata, refuting the conjecture from [41].

6.1 Planarity of unsigned Gauss words

In this section we address the question of recognising planarity of Gauss words by simulating an algorithm presented by Cairns and Elton in [8] and present an upper bound for the planarity problem of unsigned Gauss words by showing that it is recognisable by a co-non-deterministic register automata.

The fact that every knot can be represented by a Gauss word follows directly from the constructive definition of a Gauss word and the fact that the converse does not hold is illustrated in Figure 6.1. Such an observation was one of the motivations for introducing virtual knot theory [37]. A Gauss word which represents a classical knot diagram, that is a diagram embeddable into a plane without virtual crossings, is called classical or planar.

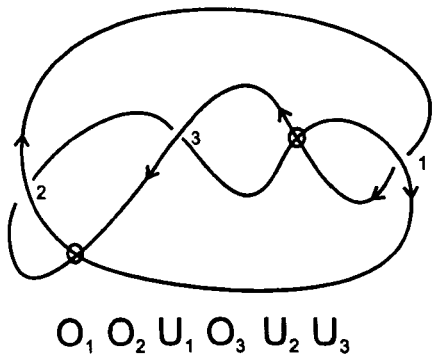

Figure 6.1: Non-planar knot diagram

6.1.1 Extended version of Cairns-Elton algorithm

Cairns and Elton provided a combinatorial algorithm in [8] which is an extension of the previous algorithm presented in [7] for signed planarity which we considered in Section 5.1. An extra condition is added to check that the parity of the sum of values (assigned to edges) is even for every closed path in the interlacement graph constructed from the Gauss word.

6.1.1.1 Notation

We will begin with some definitions to describe the main steps of the algorithm.

Definition 6.1.1. *For a Gauss word w, denote by $\alpha_i(w)$ the number of symbols that occur in w in cyclic order between the symbols U_i and O_i, taken modulo 2.*

Notice that, due to the fact that every label appears twice in a Gauss word, in the above definition one can swap U_i and O_i, so the definition of $\alpha_i(w)$ will not be affected.

Definition 6.1.2. *A signing s is a mapping $s : \mathbb{N} \to \{+, -\}$.*

Definition 6.1.3. *For an unsigned Gauss word w and a signing s, a signed word w^s is obtained from w by replacing all symbols (U,i) with $(U^{s(i)}, i)$ and (O,i) with $(O^{s(i)}, i)$. Let S_i denote the subset of symbols that occur in w^s in cyclic order between, either the symbols U_i^+ and O_i^+, or O_i^- and U_i^-. Let $\bar{S}_i$ denote $\{U_i^{s(i)}, O_i^{s(i)}\} \cup S_i$ and S_i^{-1} denote the set S_i after swapping Us with Os, that is $S_i^{-1} = \{(U^{s(j)}, j) | (O^{s(j)}, j) \in S_i\}$*

$\cup\{(O^{s(j)}, j) | (U^{s(j)}, j) \in S_i\}$. *Then* $\beta_{ij}(w^s)$ *is the number of elements in the intersection of* $\bar{S}_i$ *and* S_j^{-1} *taken modulo 2 (i.e.* $\beta_{ij}(w^s) = |\bar{S}_i \cap S_j^{-1}| \pmod 2$*).*

Notice that $\beta_{ij}(w^s)$ depends on signs $s(i)$ and $s(j)$ but not on $s(k)$ for $k \neq i, j$.

Given an unsigned Gauss word w, we associate an interlacement graph $G(w)$ as defined in Definition 2.1.4 where the vertices of $G(w)$ are labels in a shadow projection $sp(w)$ (natural numbers) and the edges of $G(w)$ are the pairs of labels (i, j) such that i and j are interlaced in $sp(w)$.

Example. *Given* $w = U_1O_3U_4U_2O_1U_5O_2U_3O_5O_4$, *the interlacement graph* $G(w)$ *of* w *is shown in figure 4. Let* $i = 1$, $j = 2$, $s(1) = +$ *and* $s(2) = +$. *Then* $\alpha_1(w) = |\{O_3, U_4, U_2\}| = 3 \equiv 1 \pmod 2$, *and* $\beta_{12}(w^s) = |\bar{S}_1 \cap S_2^{-1}| = |\{U_1^{s(1)}, O_3^{s(3)}, U_4^{s(4)}, U_2^{s(2)}, O_1^{s(1)}\} \cap \{U_1^{s(1)}, O_5^{s(5)}\}| = |\{U_1^{s(1)}\}| \equiv 1 \pmod 2$

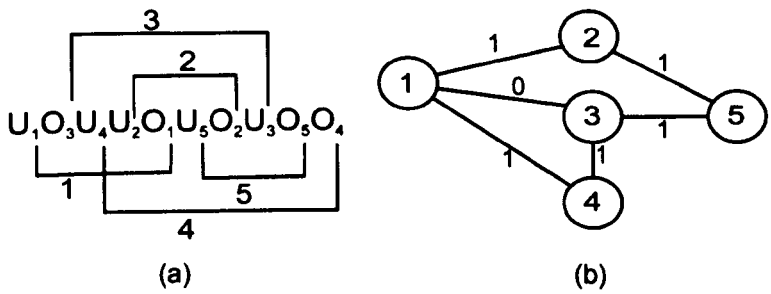

Fig. 4. *Non-planar Gauss word* w *and its corresponding interlacement graph* $G(w)$ *with edges* (i, j) *labelled by* $\beta_{ij}(w^s)$

For a signed word w^s and for each edge e_{ij} in $G(w)$, we assign the number $\beta_{ij}(w^s) \in \mathbb{Z}_2$. According to [8] that assignment defines a $\mathbb{Z}_2$ 1-cochain[1] $B(w^s)$ and the property that the Cairns-Elton algorithm checks is whether this co-chain is closed. For the purpose of this paper we need only characterisation of the closedness of $B(w^s)$ in terms of notions we have already introduced: $B(w^s)$ is closed if and only if for every closed path P in $G(w)$, the sum of the numbers $\beta_{ij}(w^s) \equiv 0 \pmod 2$ for each edge $(ij) \in P$.

[1]This is an algebraic means of representing the relationship between the cocycles and coboundaries in various dimensions of some space.

The closed path is in fact a simple cycle with no repeated vertices other than starting and ending vertices.
The Propositions 6.1.1 and 6.1.2 provide with the properties crucial for the efficient implementation of the Cairns-Elton algorithm. Also as an easy consequence of Proposition 6.1.1, we formulate Lemma 6.1.1.1.

Proposition 6.1.1. *[8, page 139] $\beta_{ij}(w^s)$ does not depend on s whenever i and j do not interlace.*

Lemma 6.1.1.1. *Let $G(w)$ denote the interlacement graph of the Gauss word w and $N(v_i)$ denote the set of vertices connected to v_i, then $\beta_{ij}(w) = |N(v_i) \cap N(v_j)| \pmod 2$ whenever i and j do not interlace in w.*

Proposition 6.1.2. *[8, Lemma 1] The condition $B(w^s)$ to be closed depends on w but not on s.*

For *all positive* signing s, that is $s(i) = +$ for all $i \in \mathbb{N}$, we denote $B(w^s)$ by $B(w)$ and $\beta_{ij}(w^s)$ by $\beta_{ij}(w)$.

6.1.1.2 Implementation

Given an unsigned Gauss word w, the algorithm proceeds by checking that

1. For all $i \in w$, $\alpha_i(w) = 0$.
2. For all $i, j \in w$, $\beta_{i,j}(w) = 0$ whenever i and j do not interlace.
3. $B(w)$ is closed.

If conditions 1,2 and 3 are satisfied then the algorithm returns "the word w is planar", otherwise if any of the conditions is not satisfied, the algorithm returns "the word w is non-planar".

We will show in this subsection that the checking of first two conditions of the above algorithm is implementable by a deterministic register automata whereas the checking of the third condition is implementable by a co-non-deterministic register

automata (that is the negation of this condition is checkable by a non-deterministic register automata).

First, we refine the above description of the algorithm and present it in more details. Given an unsigned Gauss word w, the algorithm proceeds in three stages.

I The input word is checked on whether the number of neighbours of v_i is odd for some v_i in $G(w)$. If "yes", the algorithm stops with the result "the input word is non-planar", otherwise the algorithm proceeds to the second stage.

II The input word is checked on whether $\beta_{ij}(w)$ is odd for some pair of vertices (v_i, v_j) in $G(w)$ that are not connected by an edge. If "yes", the algorithm stops with the result "the input word is non-planar", otherwise the algorithm proceeds to the third stage.

III For all positive signing s, the input word is checked on whether there exists a cycle in $G(w)$ such that the sum of $\beta_{ij}(w^s)$ assigned to its edges e_{ij} is odd. If "yes", the algorithm stops with the result "the input word is non-planar", otherwise the algorithm stops with the result "the input word is planar".

Theorem 6.1.1. *The language of non-planar unsigned Gauss words (UNSIGNED NONPLANARITY) can be recognised by a two-way non-deterministic register automaton.*

Proof. The proof is divided into two parts. In the first part we show that the first two conditions can be implemented by a deterministic register automata and in the second part we show the third condition can be implemented by a non-deterministic register automata. Let w be an unsigned Gauss word and $G(w)$ be the interlacement graph of w. Denote by $N(v_i)$ the set of all neighbours of a vertex $v_i \in G(w)$.

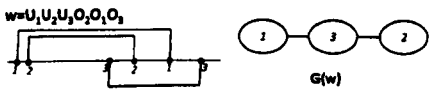

Figure 6.2: Example - $N(v_1) = \{3\}$, $N(v_2) = \{3\}$ and $N(v_3) = \{1, 2\}$

Part1 For the first condition, the automaton checks the parity of the number of neighbours of each vertex $v_i \in G(w)$. It will store in the register the first occurrence of the label i in w which corresponds to vertex v_i and store the parity of $|N(v_i)|$ in finite state control of the automaton. Checking the parity of $|N(v_i)|$ corresponds to checking the parity of the number of symbols between the two occurrences of i in w. So the automaton goes through each symbol j in between the pair of the labels i and i^{-1} in w (where i^{-1} represents the second occurrence of i in w) and on the first occurrence of j it moves first to an odd state and then alternates between odd and even states for any further occurrences. If i^{-1} is reached and the current state is odd then it moves to an accepting state. Otherwise if, for all vertices $v_i \in G(w)$, the parity of $|N(v_i)|$ is even then it checks condition (2).

For the second condition, we use Lemma 6.1.1.1 where the automaton is required to check the number of common neighbours ($|N(v_i) \cap N(v_j)|$) for any pair of vertices (v_i, v_j) that are not connected by an edge in $G(w)$. Two vertices (v_i, v_j) are connected by an edge in $G(w)$ if the label i appears between the two occurrence of j in w only once and vice versa. Otherwise the two vertices (v_i, v_j) are not connected by an edge in $G(w)$ if j appears twice between the two occurrences of i in w or does not appear at all between the two occurrences of i in w .

To verify that a vertex v_i is not connected to any vertex v_j, the automaton stores in the registers the first occurrence of i and the first occurrence of each j and then it checks whether there is an even number (either 2 or 0) of occurrences of each j in between i and i^{-1}. If the number of occurrences of j in between i and i^{-1} is even then it stores the symbol k in the register (which occurs in between i and i^{-1}) and compares it with the symbols in between j and j^{-1}. The parity of $|N(v_i) \cap N(v_j)|$ is stored in finite state control of the automaton. If there is a match, it will move first to an odd state and then alternate between odd and even states for any further matches. If i^{-1} is reached and current state is odd, the automaton moves to an accepting state. Otherwise if, for all pairs of vertices $v_i, v_j \in G(w)$ that are not connected by an edge, the parity of $|N(v_i) \cap N(v_j)|$ is even then it checks condition 3.

Part2 For checking Condition 3, we assume that s is all positive signing. First, we show how to check if two vertices v_i and v_j are connected by an edge and how to compute their $\beta_{ij}(w^s)$ value. Then we show how to sum up the $\beta_{ij}(w^s)$ values online during the traversal of a cycle. To verify that two vertices v_i and v_j are connected by an edge in $G(w)$, the automaton keeps a copy of i and j in the registers and checks that there is only one occurrence of j in between U_i and O_i. Now to calculate the value of $\beta_{ij}(w^s)$ for all positive signing s, the automaton moves its head to find the symbol U_j then compares the counterpart of each symbol k in between U_j and O_j (notice that all such counterparts form the set S_j^{-1}) with the symbols in the set $\bar{S}_i(U_i, \ldots, O_i)$. If there is a match it will move first to an odd state and then alternate between odd and even states for any further matches until O_j is reached. Finally to traverse a cycle in $G(w)$, the automaton non-deterministically chooses a vertex v_i and moves along chosen edge. During the traversal, it sums up the $\beta_{ij}(w^s)$ values of each visited edge by incrementing the counter by 1 (mod 2) only if the value of $\beta_{ij}(w^s)$ is odd, and continue updating the counter until the same vertex v_i is met for the second time. If v_i is met for the second time and the value of the counter is odd then the automaton moves to an accepting state. □

Corollary 6.1.1. *The language of planar unsigned Gauss words can be recognised by two-way co-non-deterministic register automata.*

In the next section we investigate the complexity of the planarity problem of links encoded by unsigned Gauss paragraphs.

6.2 Planarity of unsigned Gauss paragraphs

The algorithms in [] designed for recognising planarity of unsigned Gauss paragraphs have quite non-trivial properties in contrast with properties for recognising planarity of unsigned Gauss words. One may ask whether we can reduce the planarity question of unsigned Gauss paragraphs to the planarity question of unsigned Gauss words. The Cairns-Elton algorithm described in previous section works only for knot diagrams rep-

resented by Gauss words and in general does not work for links. To make it applicable for links, one has to transform a link diagram into a knot diagram while preserving the planarity of the link diagram. For such transformation, a method was presented in [38] designed for transforming signed Gauss paragraphs into signed Gauss words. In this section we will modify this method to make it applicable for the unsigned Gauss paragraphs with the aim of applying the Cairns-Elton algorithm. However will show that such transformational method requires more than finite memory and as an alternative we consider the implementation of Kauffman algorithm introduced in [37] designed for checking the planarity of Gauss paragraphs.

Before we discuss the transformation procedure we will first give a definition of unsigned Gauss paragraphs. We use a similar notation to the signed case in previous chapter.

Definition 6.2.1. *An unsigned Gauss paragraph W over the alphabet $\Sigma = \{O, U\} \times \mathbb{N}$ is a set of data words $\{w_1, ..., w_k\}$ where $k \geq 1$ such that $|w_1| + |w_2| \ldots + |w_k|$ is even and for every $i \in \mathbb{N}$ either*

- $|W|_{(U,i)} = |W|_{(O,i)} = 0$ *or*
- $|W|_{(U,i)} = |W|_{(O,i)} = 1$

6.2.1 Gauss paragraphs to Gauss words

In this section we present a modified version of the method presented in [38] for transforming a signed Gauss paragraph into a signed Gauss word adapted for the unsigned case with the aim of applying the Cairns-Elton algorithm for checking the planarity problem of unsigned Gauss words. The reason for considering such transformation method is because the recognisability problem of planar unsigned Gauss paragraphs does not seem to be implementable with finite memory as oppose to the planarity problem of unsigned Gauss words.

The main idea of the transformation is to add a new crossing each time to two different components of a link diagram and to merge the two different components in

a way that preserves the planarity property of the link diagram until we have a single component (i.e. a knot diagram).

Given a Gauss paragraph $W = \{w_1, ..., w_k\}$ of length $2n$ ($2n = |w_1| + ... + |w_k|$) with k words, we associate a Gauss word of length $2n + 2k - 2$. The transformation procedure is done as follows:

- Take two words w_i and w_j where $w_i, w_j \in W$ and $i, j \in \{1, ..., k\}$ such that either $O_l \in w_i$ and $U_l \in w_j$ where $l \in \{1, ..., n\}$.
- Rewrite w_j cyclically so that the letter U_l in w_j is the last letter of w_j. We insert a new crossing m (where $m \notin \{1, ..., n\}$) by using the following rules:
  - $O_l \rightarrow O_l U_m$ and $U_l \rightarrow U_l O_m$
- merge the two words together by inserting w_j directly after $O_l U_m$ in w_i.
- Repeat this process until we have a single component Gauss paragraph

Since the above transformation involves rewriting and merging several words before deriving the final Gauss word, the size of the final Gauss paragraph is increased by $2k - 1$ compared to the size of the original Gauss paragraph with k words. It will obviously require more than constant memory and therefore can not be transformed by a two-way deterministic register automaton with finite memory.

As an alternative, we will consider an algorithm presented by Kauffman in [37] designed to deal with the planarity problem of unsigned Gauss paragraphs also based on rewriting of a Gauss paragraph but without increasing the size of the Gauss paragraph.

6.2.2 Kauffman algorithm

We consider an algorithm for recognition of planar unsigned Gauss paragraphs proposed by Kauffman in [37]. In fact, planarity here does not depend on first components of Gauss data words, i.e. on information whether particular crossing is under- or over-crossing. Because of that the input for the algorithm is a shadow Gauss paragraph, i.e a sequence of labels (=natural numbers), where each label in the sequence occurs

twice. We assume the labels in the input word w are $1, 2, \ldots, n$ and they first occur in w in that order. The idea of the algorithm is to rewrite the shadow Gauss paragraph as single shadow word and then check the duality property (defined below) on the final word. The geometric intuition behind this rewriting is the conversion of the link diagram into a single Jordan curve (i.e a simple closed curve which divides the plane into two partitions) by smoothing each crossings [37].

Before we describe the algorithm in details, we will need to introduce the following notations:

6.2.2.1 Notation

For the definition of a shadow Gauss word refer to Definition 2.1.3 in Chapter 2.

Definition 6.2.2. *A shadow Gauss word w is called dually paired iff the set of all labels of w can be partitioned into two subsets such that no two labels in the same subset are interlaced.*

The algorithm proceeds in three stages:

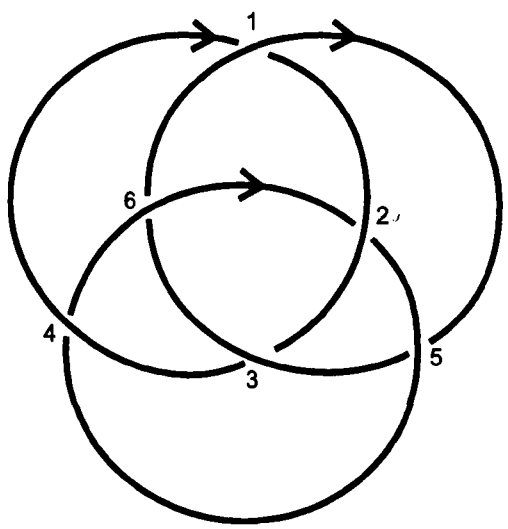

Figure 6.3: An interlacement graph of a non-planar Gauss word -

Let the set $W = \{w_1, \ldots, w_k\}$ be a Gauss paragraph where $k \geq 1$ and let $|W| = 2n$

such that $n \geq 1$.

1. If W is a single component Gauss paragraph ($k = 1$) then
   (a) For each $i = 1, ..., n \in W$
      i. check if the parity of the number of labels that occurred between the two appearances of i is even
2. For each $i = 1, 2, \ldots, n$.
   (a) If $i \in w_j$ and $i \in w_k$ where $w_j \neq w_k$ the two words are
      i. Cyclically permute w_j and w_k to make i be the first label in each word.
      ii. Attach w_k to the end of w_j
      iii. Rewrite the labels between the two occurrences of i in reverse the order.
   (b) Else rewrite the labels between the two occurrences of i in reverse order.
3. Let w^* be the resulting word. w^* is checked on whether it is a *dually paired* word.

6.2.2.2 Implementation

Let the set $W = \{w_1, \ldots, w_k\}$ be a Gauss paragraph where $k \geq 1$ and let $|W| = 2n$ such that $n \geq 1$. The three stages of the algorithm are described below in more details.

1. If $k = 1$, then the input word is checked on whether there is *even* number of labels in between two appearances of any label. If "yes" the algorithm proceeds to the second stage, if "no" the algorithm stops with the result "no, the input word is non-planar".
2. Starting with $i = 1$, the order of labels occurrence between two occurrences of i is reversed if the pair of label belongs to the same word. Otherwise we permute the two words to make i the first label in each word then we join them together and rewrite the labels between the two occurrences of i in reverse order. The process is repeated successively using $i = 1, 2, \ldots, n$. Let w^* is a resulting word.

3. w^* is checked on whether it is a *dually paired* word. If "yes" the algorithm stops with the result "yes, the input word is planar". If "no" the algorithm stops with the result "no, the input word is non-planar".

Returning to the question on lower and upper bounds for the planarity of unsigned Gauss paragraphs problem one may notice that the first stage of the Kauffman algorithm is obviously implementable on the deterministic register automata. The second stage looks problematic, for to implement reversing, the finite memory appears to be insufficient while the third stage is implementable by a non-deterministic two-way register automata as shown in Proposition 6.2.1.

The next proposition will be used later to show upper bound for planarity of unsigned Gauss paragraphs problem.

Proposition 6.2.1. *The shadow Gauss language of not dually paired words is recognisable by a non-deterministic two-ways register automaton.*

Proof. Consider an interlacement graph G_w associated with a shadow Gauss word w . It is straightforward to verify that w is dually paired iff G_w is a bipartite graph. The graph is bipartite iff it does not contain the cycle of the *odd* size. Required register automaton given a word w simulates non-deterministic traversal of the graph G_w. At the beginning it picks up non-deterministically a vertex i of G_w by moving its head to the first occurrence of the label i in w, stores the label i in the register and starts the traversal of the graph moving along the edges of G_w. The *parity* of the length of the path is stored in the finite state control of the automaton. If during traversal the automaton arrives at the vertex stored in the register it checks the parity of the path covered so far and if it is odd the word is accepted. □

In order to get a better upper bound one may try to extend the register automata with pebbles. However, in Chapter 4 Proposition 4.2.1, we showed that over Gauss words register automata are capable to model effects of adding any finite number of pebbles.

Theorem 6.2.1. *Planarity of unsigned Gauss paragraphs is recognisable by deterministic linearly bounded memory automata (LBMA).*

Proof. The proof consists in showing that the Kauffman algorithm is implementable on LBMA.

Indeed, the first stage of the algorithm is implementable with the finite memory. Also, it is straightforward to implement the second stage (reversing) on the linear memory. Proposition 6.2.1 states that the third stage of the algorithm, that is the search for the cycles of odd size on the graph associated with the Gauss word, can be done non-deterministically using only the finite memory. Notice, that if the cycle of odd size exists in the graph, then necessarily the odd cycle of the size no more than n also exists, where n is the length of the input word. Deterministic automaton may iterate then over all paths of the length up to n and check the odd cycle condition. The linear order on the input Gauss word induces the linear order on the vertices of the associated graph. It is clear that this order as well as the relation "next" with respect to the order are computable using only finitely many registers. The linear order on vertices is extended lexicoraphically on paths in the graph. Deterministic automaton iterates over paths along this order. No more than $O(n)$ memory is needed. □

6.3 Complexity bounds of unsigned planarity

In this section we refine the complexity bounds of planarity recognition and show that the language of planar unsigned Gauss words is in the complexity class $\mathcal{L}$.

In Corollary 6.1.1 we showed that the language of planar unsigned Gauss words can be recognised by a co-non-deterministic register automata and in Chapter 4 Proposition 4.2.2 we showed that if a language is recognisable by register automata then the same language can be recognised by a Turing machine with log space memory. As a result of this we have the following corollary.

Corollary 6.3.1. *The language of planar Gauss words is in* **co-**$\mathcal{NL}$ *and therefore is in* $\mathcal{NL}$ *[34].*

Next we refine the complexity bounds of planarity recognition by showing that the language of planar unsigned Gauss words is in $\mathcal{L}$.

We have shown in Theorem 6.1.1 that planarity of Gauss words is recognisable by co-nondeterministic register automata and therefore belongs to the complexity class **co-**$\mathcal{NL}$ ($= \mathcal{NL}$). Using simple arguments one can show that in terms of classical complexity classes, the result can be refined further, that is the language of planar unsigned Gauss words belongs to $\mathcal{L}$ (deterministic logspace). Indeed, the only place when one needs nondeterminism in the proof of Theorem 6.1.1 is in checking condition 3 where the search for the cycles in the interlacement graph bearing odd sum of labels. It is routine to check that the rest of the algorithm can be easily implemented in $\mathcal{L}$. It follows then that the language of non-planar unsigned Gauss words is logspace reducible to the problem from the following proposition and therefore is in $\mathcal{L}$.

Proposition 6.3.1. *The following problem is in $\mathcal{L}$ (deterministic logspace)*

GIVEN: *An undirected graph G with every edge labelled by* 0 *or* 1

QUESTION: *Is there any cycle C in G that the sum of labels of all edges in C is odd?*

Proof. The search for the required cycle can be done nondeterministically in logspace by guessing next edge in the cycle and computing the parity of the sum of labels online. Since the graph is undirected, the search can be implemented in symmetric logspace [39]. By [59] $\mathcal{SL} = \mathcal{L}$. Since $\mathcal{L}$ is closed under complementation, it follows that the language of planar unsigned Gauss words is in $\mathcal{L}$. □

Corollary 6.3.2. . *The language of planar Gauss words can be recognised by a two-way deterministic register automata.*

The above corollary follows from Proposition 6.3.1 and Theorem 4.2.1 in Chapter 4.1.

6.4 Summary

In this chapter we have investigated the complexity of planarity of knot diagrams represented by unsigned Gauss words and planarity of link diagrams represented unsigned Gauss paragraphs. We have shown that planarity of unsigned Gauss words can be recognised in deterministic logarithmic space on classical computational models and by deterministic register automata over infinite alphabets while planarity of unsigned Gauss paragraphs is recognisable by a linearly bounded memory automata. To demonstrate the results for planarity of unsigned Gauss words we have used generic mutual simulation between both computations models for the languages of bounded variability. Notice that unlike the case of signed Gauss words discussed in Chapter 5 we do not provide explicit deterministic logspace bounded decision procedure for planarity of unsigned Gauss words and rather refer to the general reduction of $S\mathcal{L}$ to $\mathcal{L}$ [59]. An explicit deterministic logspace algorithm for the later case as well as the comparison of its complexity with the algorithm(s) for signed case is a topic for further work.

Chapter 7

Definability of knot properties

In Chapters 5 and 6 we proposed to evaluate the complexity of problems of recognising knot properties in terms of the computational power of devices needed to recognise the properties. Following the proposal we demonstrated lower and upper bounds for recognisability of knot properties in terms of various automata models over *infinite* alphabets. The main property addressed was *planarity* of knot diagrams.

An alternative and to some extent a complementary approach to the study of descriptional complexity of recognisability problems is that based on definability in some logic.

In general the expressive power of register automata over infinite alphabets and definability in logics are incomparable, in particular, it was shown in [55] that expressiveness of register automata indeed is incomparable with First Order logic (FO) and Monadic Second Order logic (MSO) on words over an infinite alphabet. There are some languages that are recognisable by register automata but not even definable in MSO, on the other hand there FO properties not expressible in register automata.

In this chapter we investigate the complexity of knot properties by their definability in first order logic and its extensions. We show lower and upper bounds for the planarity and the unknotting problems encoded by logical structures.

In Section 7.1 we define the syntax and semantics of first-order predicate logic and its extension with a transitive closure operator. Then we define two encodings for Gauss

words as logical structures and prove that both encodings are uniformally translatable to each other. Further we describe the conditions for Hanf locality that will be used later to prove undefinability in first-order predicate logic.

In Section 7.2, we show that planarity of both signed and unsigned Gauss words can not be defined by a formula of first-order predicate logic, while extensions of first-order logic with *deterministic* transitive closure operators allow to define planarity of signed and unsigned Gauss words and in Section 7.3 we show that the property of unknottedness can not be defined in FO and demonstrate an implicit upper bound by showing that it is definable in existential second order logic.

7.1 Preliminaries

In this section we define the syntax and semantics of first-order predicate logic and first-order predicate logic plus the transitive closure operator. We provide two encodings for the definition of Gauss words as first-order structures and show that the encoding of the two structures are translatable to each other using first-order predicate logic plus deterministic transitive closure operator (FO+DTC). Then we describe the conditions for Hanf locality widely used to prove undefinability in FO.

7.1.1 FO

First order logic (also known as predicate logic) allows expressibility of declarative sentences with quantification and predicates to express properties of objects and relations. There are two key parts of first order logic: The syntax determines which collections of symbols are legal expressions in first-order logic, and the semantics which determines the meanings behind these expressions.

7.1.1.1 Syntax

A vocabulary in first-order logic consists of three sets:

1. A set of predicate symbols P of positive arities.

2. A set of function symbols F of positive arities.

3. A set of constant symbols C (function symbols of zero arity).

There are two key types of legal expressions: terms, which intuitively represent objects, and formulas, which intuitively express statements that can be true or false.

A term t can be defined as $t ::= x|c|f(t_1, ..., t_n)$ where x denote any variable, c denote a constant and $f \in F$ is a function with an arity n such that $n > 0$.

A formula Φ can be defined as

$$\Phi ::= p(t_1, ..., t_n)|(\neg\Phi)|(\Phi \wedge \Phi)|(\Phi \vee \Phi)|(\Phi \rightarrow \Phi)|\forall x\Phi|\exists x\Phi$$

where $p \in P$ denote a set of predicate symbols with n terms such that $n \geq 1$.

A formula in FO is called a sentence (closed or bounded formula) if it has no free variables (i.e. each variable is in the scope of a corresponding quantifier). In contrast, a formula has free variables if it is not a sentence. For example $\forall x\ p(x, y)$ is not a sentence because it has a variable y which is not bounded by a quantifier and the formula $P(x) \wedge \forall x Q(x)$ has two occurrences of the variable x with the first occurrence being free and the second occurrence is bounded by a quantifier whereas the formula $\forall x \exists y\ p(x, y)$ is a sentence as each variable is within the scope of a corresponding quantifier.

7.1.1.2 Semantics

An interpretation or a structure M is: $M = (U, f_1^M, f_2^M, ..., c_1^M, c_2^M, ..., P_1^M, P_2^M, ...)$ where

- U is a non-empty set called the universe.
- $f_i^M : U^{n_i} \rightarrow U$, where n_i is the arity of the function symbol f_i
- $c_i^M \in U$
- P_i^M of an arity n_i is a subset of U^{n_i}

Each term $t(\bar{x})$ where $|x| = n$ is assigned an element $t_M(\bar{a}) \in U$, where $\bar{a} \in U^n$ is defined by the corresponding object assignment.

If Φ is a sentence, then one can define the satisfaction relation ($\models$) between structures on one hand and sentences on the other hand. For a formula ϕ, $M \models \phi$, meaning M satisfies ϕ (under some object assignment) is defined by:

1. $M \models t_1(\bar{a}) = t_2(\bar{a})$ iff $t_1^M(\bar{a}) = t_2^M(\bar{a})$
2. $M \models P(t_1(\bar{a}), \cdots, t_k(\bar{a}))$ iff $(t_1(\bar{a}), \cdots, t_k(\bar{a})) \in P^M$
3. $M \models \neg\Phi$ iff $M \not\models \Phi$
4. $M \models \Phi \wedge \Psi$ iff $M \models \Phi$ and $M \models \Psi$
5. $M \models \Phi \vee \Psi$ iff $M \models \Phi$ or $M \models \Psi$
6. $M \models \exists x\psi(\bar{a}, x)$ iff $M \models \Phi(\bar{a}, b)$ for some $b \in U$
7. $M \models \forall x\psi(\bar{a}, x)$ iff $M \models \Phi(\bar{a}, b)$ for every $b \in U$

7.1.1.3 Quantifier rank

The quantifier rank of a formula ϕ, written $qr(\phi)$ is defined inductively as follows:

- if ϕ is atomic formula then $qr(\phi) = 0$
- if $\phi = \neg\psi$ then $qr(\phi) = qr(\psi)$.
- if $\phi = \psi_1 \vee \psi_2$ or $\phi = \psi_1 \wedge \psi_2$ then $qr(\phi) = max(qr(\psi_1), qr(\psi))$.
- if $\phi = \exists x\psi$ or $\phi = \forall x\psi$ then $qr(\phi) = qr(\psi) + 1$.

7.1.1.4 Elementary equivalence (up to quantifier rank n)

For two structures A and B, we say that $A \equiv_n B$ if for any sentence ϕ with $qr(\phi) \leq n$, $A \models \phi$ if, and only if $B \models \phi$.

7.1.1.5 Definability of a class of structures in FO

Let Σ be some vocabulary and K a class of Σ-structures (that is structures which can be used to interpret any formula over vocabulary Σ). We say that the class K is definable over Σ if there is a closed Σ-formula ϕ such that for every Σ-structure S, $S \models \phi$ iff $S \in K$.

It is well-known that first-order logic has limited expressive power. In particular it lacks recursion and counting mechanism. For instance, the transitive closure (TC[1]) operator can not be expressed in FO [40] (i.e. there is no such an FO-formula $\phi_R(x, y)$ on any structure whose vocabulary is R that would define the transitive closure of R). In other words, the transitive closure operator can not be precisely characterised within the framework of FO. By adding transitive closure operators to FO, we obtain a natural family of logics with a recursion mechanism.

We adopt the following definitions of FO+TC and FO+DTC from [20].

7.1.2 FO+TC

Transitive closure logic is closed under the usual first-order operations. It's obtained by augmenting the syntax of first order logic by the following rule for building formulae: Let $\phi(\bar{x}, \bar{y})$ be a formula with variables $\bar{x} = x_1, \cdots, x_k$ and $\bar{y} = y_1, \cdots, y_k$, and let $\bar{u}$ and $\bar{v}$ be two k-tuples of terms. Then $[TC_{\bar{x},\bar{y}}\phi(\bar{x}, \bar{y})](\bar{u}, \bar{v})$ is a formula which says that the pair $(\bar{u}, \bar{v})$ is contained in the transitive closure of the binary relation on k-tuples that is defined by ϕ. In other words, $A \models [TC_{\bar{u},\bar{v}}\phi(\bar{x}, \bar{y})](\bar{a}, \bar{b})$ if and only if, there exist an $n \geq 1$ and tuples $\bar{c_0}, \cdots, \bar{c_n}$ in A such that $\bar{c_0} = \bar{u}, \bar{c_n} = \bar{v}$ and $A \models \phi(\bar{c_i}, \bar{c_{i+1}})$, for all $i < n$.

The transitive closure logic is defined by the following BNF:

$$\phi ::= p(t_1 \cdots, t_n)(\neg\Phi)|(\Phi \wedge \Phi)|(\Phi \vee \Phi)|(\Phi \rightarrow \Phi)|\forall x\Phi|\exists x\Phi|TC_{\bar{x},\bar{y}}\phi(\bar{x}, \bar{y})](\bar{t_1}, \bar{t_2})$$

where the variables in $\bar{x}\bar{y}$ are pairwise distinct and where the tuples $\bar{x}, \bar{y}, \bar{t_1}$ and $\bar{t_2}$ are

[1]The transitive closure of a relation R is defined as the smallest relation extending R that is transitive.

all of the same length, $\bar{t_1}$ and $\bar{t_2}$ being tuples of terms.

7.1.3 FO+DTC

Deterministic transitive closure logic is an interesting variant of FO+TC which defines the transitive closure of any deterministic definable relation. The syntax of FO+DTC is analogous to FO+TC, allowing us to build formulae of the form $[DTC_{\bar{x},\bar{y}}\phi(\bar{x},\bar{y})](\bar{u},\bar{v})$ for any formula $\phi(\bar{x},\bar{y})$. The semantics can be defined by the equivalence

$$[DTC_{\bar{x},\bar{y}}\phi(\bar{x},\bar{y})](\bar{u},\bar{v}) \equiv [TC_{\bar{x},\bar{y}}\phi(\bar{x},\bar{y}) \wedge \forall\bar{z}\phi(\bar{x},\bar{z}) \rightarrow \bar{y} = \bar{z}](\bar{u},\bar{v})$$

FO+TC has shown to be more expressive than FO+DTC and likewise FO+DTC is more expressive than FO [20].

7.1.4 Gauss structure

In order to address definability questions of knot properties, we will use the two variants of knot encodings, namely representations of Gauss diagrams and Gauss words by finite FO relational structures.

Definition 7.1.1. *A Gauss diagram can be described as a finite relational structure F, where $F = (V, R, S)$ and R and S are binary relations.*

- *$V = V_o \cup V_u$ where V is a finite set of points such that $V_o \cap V_u = \emptyset$ and $V_o \cup V_u = V$.*
- *(V, S) is an oriented cycle*
- *R is a bijection relation between V_o and V_u.*

We will also use a variant of the above definition called a *pointed* Gauss diagram defined as $F_c = (V, R, S, c)$ where c is constant denoting a distinct point in V. When we consider definability of properties of diagrams in the extensions of first-order logic by transitive closure operators, we may assume that pointed diagrams are linearly ordered, for linear order is definable over pointed diagrams in obvious way.

Definition 7.1.2. *A Gauss word w over the alphabet $\Sigma \times \mathbb{N}$ can be described as a finite structure $G_w = (M, <, \sim, P_O)$ where:*

- $M = \{1, ..., n\}$ *where* $|w| = n$,
- $<$ *is a binary relation "less than" on* M
- $\sim$ *is an equivalence relation where two positions are equivalent if they have the same data value (elements of the second component).*
- P_O *is a predicate symbol satisfying the label (the element of first component)* O *at some position.*

Definition 7.1.2 above can be seen as a special case of the encoding of general data words considered in [6] without the successor relation $+1$. The successor relation $+1$ was needed in [6] because the logic used was a fragment of FO which was restricted to quantification over only two variables (FO^2) whereas in the above definition we don't consider such restriction and hence the relation $+1$ can be defined in terms of the relation $<$. Another difference is the structures we consider in Definition 7.1.2 correspond to bounded Gauss words where every equivalent class of the relation $\sim$ contains two elements.

It easy to see that the encoding of the two structures are uniformly translatable to each other using FO+DTC and DTC is only needed to define linear order for the encoding of Gauss words using the relation S on the encoding of Gauss diagrams.

Let $\mathcal{F}$ denote a class of structures of all Gauss diagrams F, $\mathcal{F}_c$ denote a class of structures of all pointed Gauss diagrams F_c and $\mathcal{G}$ denote a class of structures of all Gauss words G_w.

Proposition 7.1.1. *$\mathcal{F}_c$ is uniformly FO-translatable to $\mathcal{G}$ (i.e. there are FO-formulae $\phi_S(x, y)$, $\phi_R(x, y)$ and $\phi_c(x)$ in vocabulary $(<, \sim, P_O)$ such that given a Gauss word structure $G_{gd} = (V, \leq, \sim, P_O)$ corresponding to the Gauss diagram gd, then $F_{gd} = (V, R, S, c)$ is a pointed Gauss diagram structure corresponding to the Gauss diagram*

gd. Here $S = \{(a,b) \in V | G_{gd} \models \phi_S(a,b)\}$ *and* $R = \{(a,b) \in V | G_{gd} \models \phi_R(a,b)\}$*) and* $c \in V$ *such that* $G_{gd} \models \phi_c(c)$.

Proof. To define F_{gd} in terms of G_{gd}, we consider the following FO-formulae:

- $\phi_S(x,y) = [x < y \wedge \neg\exists z(x < z \wedge z < y \wedge x \neq z \wedge y \neq z)] \vee [\forall u(u \neq x \Rightarrow u < x) \wedge \forall v(v \neq y \Rightarrow y < v)]$,
- $\phi_R(x,y) = P_O(x) \wedge \neg P_O(y) \wedge x \sim y$ and
- $\phi_c(x) = \forall z(z \neq x \Rightarrow z < x)$.

Now it's straight forward to check that these two formulae define what we need. □

From the above proposition, it follows that if some property of Gauss diagram/words is FO-definable over $\mathcal{F}_c$ then is definable over $\mathcal{G}$. By contraposition, if some property is not FO-definable over $\mathcal{G}$ then it is not definable over $\mathcal{F}_c$.

Proposition 7.1.2. $\mathcal{G}$ *is uniformly FO+DTC-translatable to* $\mathcal{F}$ *(i.e. there are FO+DTC-formulae* $\phi_<(x,y)$, $\phi_\sim(x,y)$ *and* ϕ_{P_O} *in vocabulary* (R,S,c) *such that given a Gauss diagram structure* $F_w = (V,R,S,c)$ *corresponding to the Gauss word* w *then* $G_w = (V, <, \sim, P_O)$ *is a Gauss word structure corresponding to the Gauss word* w*. Here* $<= \{(a,b) \in V | F_w \models \phi_<(a,b)\}$, $\sim= \{(a,b) \in V | F_w \models \phi_\sim(a,b)\}$ *and* $P_O = \{a \in V | F_w \models \phi_{P_O}(a)\}$*).*

Proof. Let $\phi_<(x,y) = DTC_{u,v}(S(u,v) \wedge v \neq c)(x,y)$, $\phi_\sim(x,y) = R(x,y) \vee R(y,x)$ and $\phi_{P_O}(x) = \exists y R(x,y)$

Now it's straight forward to check that these two formulae define what we need. □

From this proposition, it follows that if some property of Gauss diagram/words is FO+DTC-definable over $\mathcal{G}$ then it is FO+DTC-definable over $\mathcal{F}_c$. By contraposition, if some property is not FO+DTC-definable over $\mathcal{F}_c$ then it is not definable over $\mathcal{F}$.

The encoding of Gauss paragraphs is defined in a similar way except for the relation S which is defined as a set of oriented cycles $S = S_1, \ldots, S_n$.

7.1.5 Hanf locality

Hanf locality is used to show that a certain class of structures is not FO-definable using sufficient conditions based on the idea of locality.

Before we formulate Hanf locality criterion we will explain first the notion of neighbourhood.

Given a relational structure $M = (A, p_1, \cdots, p_n)$, we define the binary relation: $E(a_1, a_2)$ if, and only if, there is some relation R and some tuple $\bar{a}$ containing both a_1 and a_2 with $R(\bar{a})$. In this way we obtain a graph $G_M = (A, E)$ called the Gaifman graph of M. We denote by $d(a, b)$ the distance of the shortest path from a to b in G_M. The neighbourhood of a point a in M denoted by $Nbd_r^M(a)$ is defined as a substructure of M given by the set $\{b | d(a, b) \leq r\}$.

Two structures A and B are *Hanf equivalent* with radius r and threshold q denoted by $A \simeq_{r,q} B$ if for every $a \in A$ the two sets $\{a' \in A | Nbd_r^A(a) \cong Nbd_r^A(a')\}$ and $\{b \in B | Nbd_r^A(a) \cong Nbd_r^B(b)\}$ either have the same size or both have size greater than q and similarly for every $b \in B$.

Next we formulate the Hanf locality criterion. Here the symbol $\equiv_p$ denotes elementary equivalence

Theorem 7.1.1. *[27] For every vocabulary σ and every p there are $r \leq 3^p$ and $q \leq p$ such that for any $\sigma-$structures A and B if $A \simeq_{r,q} B$ then $A \equiv_p B$.*

7.1.6 Duplicated Gauss words

In this subsection we introduce a specific class of Gauss words which will be referred to as *duplicated Gauss words.* Consideration of such class of structures is useful for proving some lower bounds for planarity problem as well as unknottedness.

Definition 7.1.3. *Let $w = (a_1, b_1), \cdots, (a_{|w|}, b_{|w|})$ be a Gauss word, a shadow projection sh of w is defined as $sh(w) = b_1, \cdots, b_{|w|}$ (i.e. a sequence of elements of the second component of w).*

Definition 7.1.4. *A duplicated-Gauss word is a Gauss word w such that $sh(w) = u \cdot u$ where $u = i_1, \cdots, i_n$.*

Example Given a *duplicated*-Gauss word $w = U_1U_2U_3U_4U_5O_1O_2O_3O_4O_5$, then $sh(w) = 1234512345$. The elements of the shadow projection of w can be seen as a shadow Gauss word (i.e a sequence of natural numbers). The corresponding duplicated-Gauss diagram for the shadow projection of w is shown in Figure 7.1.

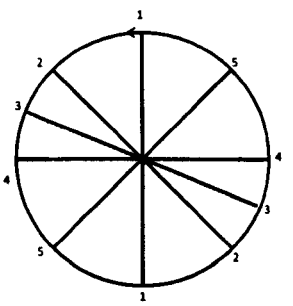

Figure 7.1: Example: - A duplicated-Gauss diagram representing the word 1234512345

7.2 Planarity

In this section we investigate the complexity of planarity problem of Gauss words and Gauss paragraphs in terms of their definability in Logic. We show as a lower bound that planarity property of Gauss words (signed and unsigned) is not definable in first-order predicate logic, while extensions of first-order logic with *deterministic* transitive closure operators allow to define planarity property of signed and unsigned Gauss words.

7.2.1 Gauss words

Proposition 7.2.1. *Given a duplicated-Gauss word w representing a knot diagram k where $|w| = 2n$ and $n \geq 1$, n is even iff k is a non-planar diagram.*

Proof. Suppose that k is a planar knot diagram and n is even. Let $\alpha_i(w)$ denote the number of labels that appeared between two occurrences of $i \in w$ where $i = \{1, ..., n\}$.

If w represents a planar knot diagram then for every label i of w, $\alpha_i(w) \equiv 0 \pmod 2$ (Gauss's necessary condition for planarity [23]). Since $w = i_1 \cdots i_n i_1 \cdots i_n$ and n is even then for any $i \in w$, $\alpha_i(w) = n - 1 \equiv 1 \pmod 2$. Therefore w does not represent a planar knot diagram, hence k must be a non-planar knot diagram. Similarly, if k is non-planar then n is even holds following the same idea. □

Theorem 7.2.1. *The planarity property of signed/unsigned Gauss words is not definable by FO over Gauss diagram structures.*

Proof. Let $\mathcal{A}$ denote a class of all planar duplicated Gauss diagrams and $\mathcal{B}$ denote a class of all non-planar duplicated Gauss diagrams. We are going to show that for all r and q there are $A \in \mathcal{A}$ and $B \in \mathcal{B}$ such that $A \simeq_{r,q} B$. This and Theorem 7.1.1 would

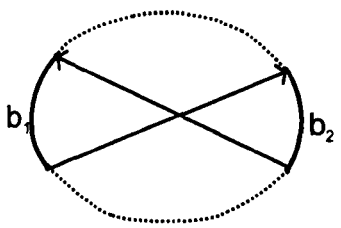

Figure 7.2: Border edges - Border edges on the circle are labelled by b_1 and b_2

imply that for all p there are $A \in \mathcal{A}$ and $B \in \mathcal{B}$ such that $A \equiv_p B$, which in turn would imply that planarity property of signed/unsigned Gauss words is not definable by FO over Gauss diagram structures.

Fix r and q, we show that for $A \in \mathcal{A}$ of size $2(2r + 3^q + 1)$ and a Gauss diagram $B \in \mathcal{B}$ of size $2(2r + 3^q)$ we have $A \simeq_{r,q} B$.

To distinguish between the different types of neighbourhoods, we consider two groups C_1 and C_2 where C_1 include the set of points that can reach the *border edges*[1] by radius r and C_2 includes the set of points that cannot reach the border edges within radius r. So we divide the points of A and B into six parts where $C_1^A = A_1 \cup A_3 \cup A_4 \cup A_6$

[1] a border edge is an edge on the circle whose points are connected by an incoming chord and an outgoing chord, for illustration see Figure 7.2. Notice all other edges on the circle are either connected to two incoming chords or two outgoing chords

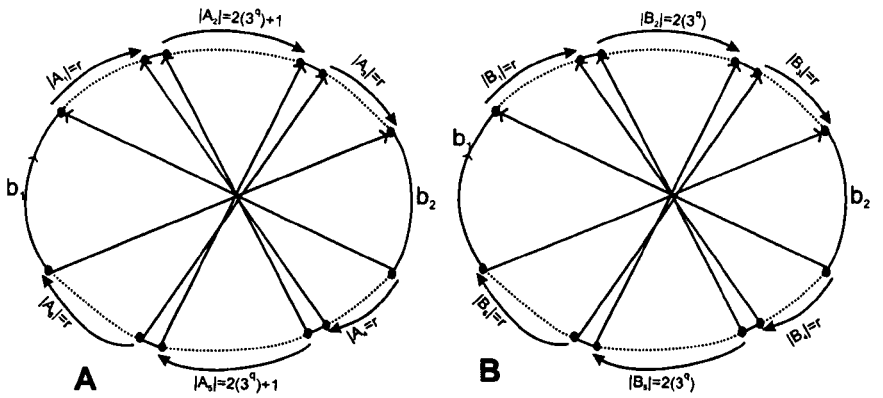

Figure 7.3: duplicated-Gauss diagrams - A represents a planar knot diagram and B represents a non-planar knot diagram

and $C_2^A = A_2 \cup A_5$ and likewise $C_1^B = B_1 \cup B_3 \cup B_4 \cup B_6$ and $B = B_1, \cdots, B_6$ $C_2^B = A_B \cup B_5$, (see Figure 7.3 for an illustration).

We will show that for each $a \in C_1^A$, the number of isomorphic neighbourhoods of each type in C_1^A within A is the same as the number isomorphic neighbourhoods in C_1^B within B.

To capture all neighbourhoods of the set of points in C_1^A and in C_1^B respectively, we expand the set of points close to the border by considering all points within radius $2r$. Now if we consider the two substructures on both diagrams, we get the same number of points and the substructures containing such points are also identical. From this it follows that if you take any point of radius r the number of isomorphic neighbourhoods of radius r within A is the same as the number of isomorphic neighbourhoods of radius r within B.

For all other points not C_1^A (or C_1^B respectively), we will show that their number of isomorphic neighbourhoods is greater than q. This is true because the number of points in C_2^A (or in C_2^B respectively) is exponential with respect to q and the set of points in C_2^A (or in C_2^B respectively) generate the same type of neighbourhood due to the regular structure maintained by both diagrams.

□

In Chapter 4 Proposition 4.2.2 we showed that if a language is recognisable by reg-

ister automata then the same language under some natural encoding can be recognised by a Turing machine with log-space memory. Then in Chapters 5 and 6 we showed that the language of planar (signed and unsigned) Gauss words is recognisable by a deterministic register automata. As a consequence it follows that the languages of planar Gauss words are definable by FO+DTC.

Theorem 7.2.2. *The language of planar (signed and unsigned) Gauss words is definable in first order logic plus deterministic transitive closure (FO+DTC).*

Proof. The language of planar signed Gauss words is in DRA Theorem 5.1.1. The language of planar unsigned Gauss words is in DRA Corollary . By Proposition 4.2.2 $DRA \subseteq \mathcal{L}$. Since $\mathcal{L}$=FO+DTC (over ordered structures) [33] then the languages of planar signed and unsigned Gauss words are definable in FO+DTC. □

7.2.2 Gauss paragraphs

The question of definability of planarity of Gauss paragraph has already been addressed in the recent work by B. Courcelle [11].

Before we formulate the result from [11] and discuss its relation to our work, we would like to recall the main results on relationships between the computational models we consider in this chapter and definability in logic. The computational power of classical finite automata over finite alphabets is characterised precisely in terms of definability in *Monadic Second Order Logic (MSO)*, the extension of *First-Order Logic (FO)* with quantification over sets. The classical theorem of Trakhtenbrot and Elgot at al. from 1950s states that the languages recognisable by finite state automata (regular languages) are exactly those definable in MSO. The languages definable in FO constitute an important class of star-free regular languages. For the case of automata over infinite alphabets the situation is much more intricate, and in general, register automata are orthogonal to logically defined classes. In [55] authors compared definability in MSO^*, suitably defined variant of MSO which allows to define the properties of data words, with recognisability by register and pebble automata. In particular they have shown

that MSO^* is as least as powerful as *one-way non-deterministic* register automata, but incomparable with *two-way deterministic* and *non-deterministic* automata. Pebble automata behave much better and recognisability by all their natural variants is covered by definability in MSO^*.

In [11] B. Courcelle proves the following theorem, which we reformulate in the terms we have adopted in the thesis.

A genus of a surface is a topologically invariant property of a surface defined as the largest number of nonintersecting simple closed curves that can be drawn on the surface without separating it [65].

Theorem 7.2.3. *[11] For every genus g, it is definable by an MSO formula the property of a Gauss paragraph to be a code of the self-intersecting closing curve, embeddable in a surface of the genus g.*

Corollary 7.2.1. *The planarity of unsigned Gauss paragraphs is definable in MSO.*

The encoding of Gauss paragraph by relational structures and logic MSO used in [11] are different from the encoding of data words and logic MSO^* from [55], but insignificantly. It is straightforward to show that over Gauss paragraphs the notion of definability is the same for both cases.

For the planarity case of signed Gauss paragraphs our result on DRA recognisability is incomparable with the above MSO-definability result on unsigned Gauss paragraphs. In fact we will show that planarity of signed Gauss paragraphs is definable in FO+DTC.

Theorem 7.2.4. *The language of planar signed Gauss paragraphs is definable in first order logic plus deterministic transitive closure (FO+ DTC).*

Proof. The language of planar unsigned Gauss paragraphs is recognisable by a DRA by Theorem 5.2.1. By Proposition 4.2.2 $DRA \subseteq \mathcal{L}$. Since $\mathcal{L} = FO + DTC$ (over ordered structures) [33] then it follows that the language of planar signed Gauss paragraphs is definable in FO+ DTC. □

7.3 Unknottedness

Before we can show that the property of unknottedness is not definable in FO, we need to consider two classes of structures A and B such that A represents a non-trivial knot and B represents a trivial knot. We will construct non-trivial knot diagrams that belong to A by starting with a simple non-trivial knot diagram (trefoil) and then adding even number of crossings in such a way so that the original structure of the trefoil becomes hidden (see Figure 7.4). For trivial knot diagrams that belong to B, we do the same but we start with a trivial knot diagram with triple-point crossings that has similar order of crossings as trefoil (see Figure 7.5).

Definition 7.3.1. *Let C be a class of duplicated-Gauss words such that for any $w \in D$, $|w| = 6k$ where $k \in \mathbb{N}$ and $k \equiv 1(Mod2)$.*

Proposition 7.3.1. *Given $w \in C$ and an odd integer $k \geq 1$, if the projection of the first component (i.e. sequence of Os and Us) of w is $O^kU^kO^kU^kO^kU^k$ then w is non-trivial.*

Proof. Consider the knot diagram encoded by w for $k = 1$, clearly this is a projection of the trefoil knot (illustrated on the left-hand side picture of Figure 7.4), which is of course a non-trivial knot diagram. For any odd integer k where $k > 1$, the number of crossings added correspond to k applications of Reidemeister moves of type II↑. Since the knot type is invariant under Reidemeister moves, it will not be changed. □

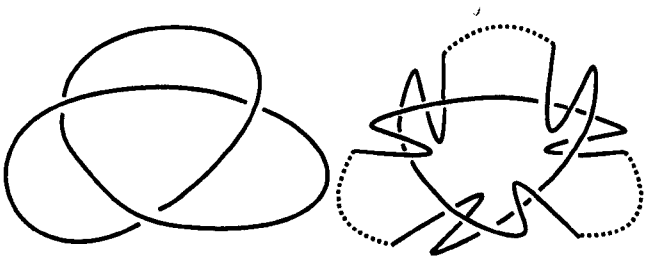

Figure 7.4: - A class of non-trivial knot diagrams

Proposition 7.3.2. *Given $w \in C$ and an odd integer $k \geq 1$, if the projection of the first component of w is $O^{k+1}U^{k-1}O^kU^{k+1}O^{k-1}U^k$ then w is trivial.*

Proof. The arguments are similar to those in the proof of Proposition 7.3.1 but the knot diagram encoded by w is trivial, see Figure 7.5. □

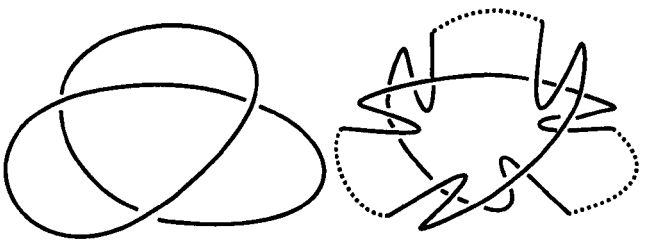

Figure 7.5: - A class of trivial knot diagrams

Theorem 7.3.1. *The property of unknottedness is not definable by FO over Gauss diagram structures.*

Proof. Let $\mathcal{A}$ denote a class of all non-trivial duplicated Gauss diagrams and $\mathcal{B}$ denote a class of all trivial duplicated Gauss diagrams. We are going to show that for all r and q there are $A \in \mathcal{A}$ and $B \in \mathcal{B}$ such that $A \simeq_{r,q} B$.

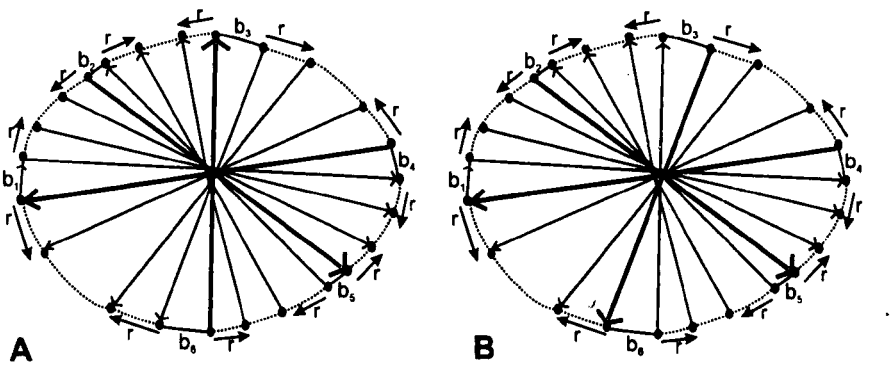

Figure 7.6: duplicated-Gauss diagrams - A represents a non-trivial knot diagram and B represents a trivial knot diagram

This and Theorem 7.1.1 would imply that for all p there are $A \in \mathcal{A}$ and $B \in \mathcal{B}$ such that $A \equiv_p B$, which in turn would imply the property of unknottedness is not definable by FO over Gauss diagram structure.

Fix r and q, we show that for $A \in \mathcal{A}$ of size $6(2r + 2(3^q)) + 2$ and a Gauss diagram $B \in \mathcal{B}$ of the same size we have $A \simeq_{r,q} B$.

To distinguish between the different types of neighbourhoods, we use an argument similar to that in the proof of Theorem 7.2.1 (that is by categorising the points of the diagram into two groups; one group contain the set of points whose neighbourhood include border edges and the other group contain the set of points whose neighbourhood does not include border edges). There are six border edges in each diagram labelled by $b1 \cdots, b_6$, all r-points close to the border will satisfy the first disjunct condition of Hanf-equivalent by generating a bounded number of isomorphic neighbourhoods of the same size and all other points that cannot reach any of the border edges will satisfy the second disjunct condition of Hanf-equivalent by generating an exponential number of isomorphic neighbourhoods of the same type with respect to q. □

From the computational complexity side, the unknotting problem had being shown to be in NP [30] and in $AM \cap Co - AM$ [28]. As a results of this, we obtain an upper bound for the unknotting problem in terms of definability in existential second order logic.

Proposition 7.3.3. *The unknotting problem is definable in existential second order logic (ESO)*

Proof. The unknotting problem was shown to be in NP [30]. Since NP = ESO [21] then the unknotting problem is in ESO. □

7.4 Summary

We have investigated the descriptive complexity of some knot properties and demonstrated new lower and upper bound for the planarity problem of Gauss words and the unknotting problem. We have shown that planarity of both signed and unsigned Gauss words cannot be expressed by a formula of first-order predicate logic, while extensions of first-order logic with deterministic transitive closure operator allow to define planarity of both signed and unsigned Gauss words. Similarly for Gauss paragraphs we have shown that planarity of signed Gauss paragraphs is definable in FO+DTC while planarity of unsigned Gauss paragraphs is definable in MSO [11].

Referring to the classical Fagin's results [21] we have shown that unknottedness is definable in ESO. Explicit definitions of planarity and unknottedness are of some interest and are left for future work.

Chapter 8

Conclusion and Further work

In this thesis we explored a wide range of computational problems and properties of knots. We were primarily concerned with algorithmic, computational and logical aspects of knots, and analysis of knot theoretic properties by theoretical computer science methods. In the introductory chapter we showed that many knot theoretic problems have decision algorithms but have not been investigated in terms of their complexity status. In this work we partially cover unexplored aspects of these problems and provide a number of new results related to reachability problems for Reidemeister moves, relation of Gauss diagrams to Eulerian maps, descriptional and descriptive complexity of classical knot problems.

In Chapter 2 we described various finite representations of knots as discrete structures and provided a formal definition for the encoding of knots by Gauss words. We described how problems about knots can be reduced to the questions about Gauss words and described a list of knot problems to be investigated in this thesis.

In Chapter 3 we considered knot transformations in terms of string rewriting systems and provided a refined classification of Reidemeister moves formulated as string rewriting rules for Gauss words. Then we analysed the reachability properties for each type and presented some lower and upper bounds on the complexity of the paths between two knot diagrams reachable by a sequence of moves of the same type while for a combination set of moves of different type we provided some plausible classes of

structures that can be used for proving lower bounds.

Further we considered a class of non-isomorphic knot diagrams generated by type I moves from the unknot and discovered that the sequence corresponding to the number of non-isomorphic diagrams with respect to the number of crossings is equal to a sequence related to a class of unrooted Eulerian maps with respect to the number of edges. We investigated the bijective mapping between the two classes of objects and presented two explicit algorithms to demonstrate the transformations from one object to the other.

Furthermore we considered the question of knot transformations by reduction and introduced a new set moves which can be used to substitute one of the rules of type II that increases the number of crossings. We demonstrated that our new moves coupled with Reidemeister moves can unknot all known examples of complex trivial knot diagrams without increasing number of crossings. Finally we formulated some open questions left as conjectures for future work.

In Chapter 4 we considered different models of automata over infinite alphabets for studying complexity of problems related to knots. In particular, we considered register automata which is one of the weakest models of automata over infinite alphabets. Although register automata is considered weak, we showed that over a class of bounded languages it can effectively simulate counters and pebbles and log-space bounded computations under some proper encoding. Then we demonstrated that over a class of languages with bounded variability the computation done by register automata can be mimicked by a Turing machine using log-space memory. Further we considered different variants of register automata and presented lower and upper bounds for the recognition problem of Gauss words, particularly we showed that the language of Gauss words cannot be recognised by a 1-way non-deterministic register automaton but it is recognisable by using a 2-way deterministic register automaton. Additionally we showed that register automaton can even recognise non-trivial properties, specifically we have shown that the isomorphic Gauss words problem is recognisable by a 2-way deterministic register automaton.

More generally, the automata based approach opens perspectives for studying more

complex knot problems, like unknotedness or equivalence. Non-trivial lower bounds for such problems are unknown and weak automata models are plausible candidates here to try. In opposite direction, knot theory provides a rich supply of natural problems formulated in terms of languages over infinite alphabets, and that, one may expect, will influence the development of the theory of such languages and related computational models.

In chapter 5 we have applied automata over infinite alphabets for studying complexity of planarity problem for signed Gauss words and signed Gauss paragraphs. We considered the implementation of two different algorithms and shown that the languages of planar signed Gauss words and signed Gauss paragraphs can be recognised by deterministic two-way register automata. Then in terms of classical computational models we showed that planarity of signed Gauss words and signed Gauss paragraphs can be recognised in deterministic Turing machine using logarithmic space memory. The result is final in the sense that the power of non-deterministic one-way register automata is not even enough to recognise whether an input is a Gauss word.

In chapter 6 we investigated the complexity of the planarity problem for unsigned Gauss words and unsigned Gauss paragraphs. We have shown that planarity of unsigned Gauss words can be recognised in deterministic logarithmic space on classical computational models and by deterministic two-way register automata over infinite alphabets while for the planarity problem of unsigned Gauss paragraphs we demonstrated an upper bounds in terms automata with linearly bounded memory. To demonstrate these results we have used generic mutual simulation between both computations models for the languages of bounded variability. Notice that unlike the case of signed Gauss words discussed in Chapter 5 we did not provide explicit deterministic log-space bounded decision procedure for planarity of unsigned Gauss words and rather refer to the general reduction of $\mathcal{SL}$ to $\mathcal{L}$ [59]. An explicit deterministic log-space algorithm for the later case as well as the comparison of its complexity with the algorithm(s) for signed case is a topic for further work.

In Chapter 7 we have investigated the complexity of some knot properties in terms

of their logical expressions and demonstrated new lower and upper bound for the planarity problem of signed and unsigned Gauss words as well as the unknotting problem. We have shown that planarity of both signed and unsigned Gauss words cannot be expressed by a formula of first-order predicate logic, while extensions of first-order logic with deterministic transitive closure operator allow to define planarity of both signed unsigned Gauss words. Similarly for Gauss paragraphs we demonstrated an upper bound and shown that planarity of signed Gauss paragraphs is definable in FO+DTC while planarity of unsigned Gauss paragraphs is definable in MSO [11]. Referring to the classical Fagin's results [21] we have shown that unknottedness is definable in ESO. Explicit definitions of planarity and unknottedness are of some interest and a potential area for further study in the future.

There is much more work to be carried out in relation to computational problems for knots and links. Clearly determining better complexity bounds for equivalence and unknottedness are central open problems in this field. We expect that new approaches and methods presented in this thesis will later help with further investigations of the above mentioned problems.

References

[1] S. Abramsky. Temperley-Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics. *Mathematics of Quantum Computation and Quantum Technology*, 2007. 1

[2] Rajeev Alur, Kousha Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 467–481. Springer Berlin / Heidelberg, 2004. 85

[3] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge Univ Pr, 1999. 35

[4] D. Bar-Natan, T.T.Q. Le, D.P. Thurston, et al. Two applications of elementary knot theory to lie algebras and vassiliev invariants. *Geometry and Topology*, 7(1):1–31, 2003. 1

[5] H. Bjorklund and T. Schwentick. On Notions of Regularity for Data Languages. *Lectures Notes in Computer Science*, 4639:88, 2007. 11, 104

[6] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. *Logic in Computer Science, Symposium on*, pages 7–16, 2006. 143

[7] G. Cairns and D.M. Elton. The planarity problem for signed Gauss words. *Journal*

of Knot Theory and its Ramifications, 2(4):359–367, 1993. 3, 6, 16, 17, 109, 110, 121, 123

[8] G. Cairns and D.M. Elton. The Planarity Problem II. *Journal of Knot Theory and its Ramifications*, 5:137–144, 1996. 3, 17, 110, 121, 122, 123, 124, 125

[9] J.S. Carter. Classifying immersed curves. In *Proc. Amer. Math. Soc*, volume 111, pages 281–287, 1991. 16, 110, 114, 119

[10] J.S. Carter, M. Elhamdadi, M. Saito, and S. Satoh. A lower bound for the number of Reidemeister moves of type III. *Topology and its Applications*, 153(15):2788–2794, 2006. 45

[11] B Courcelle. Diagonal walks on plane graphs and local duality. Available at http://www.labri.u-bordeaux.fr/perso/courcell/Textes/DiagonalWalks-Submitted(2006).pdf, March 2006. 149, 150, 153, 158

[12] A. Coward. Ordering the Reidemeister moves of a classical knot. *Algebraic & Geometric Topology*, 6:659–671, 2006. 2, 34

[13] A. Coward and M. Lackenby. An upper bound on reidemeister moves. *Arxiv preprint arXiv:1104.1882*, 2011. 2, 17, 34

[14] P.R. Cromwell. *Knots and links*. Cambridge Univ Pr, 2004. 3, 46, 70, 71

[15] C. David. Mots et données infinies. *Master's thesis, LIAFA*, 2004. 87

[16] H. De Fraysseix and P. Ossona de Mendez. On a characterization of Gauss codes. *Discrete and Computational Geometry*, 22(2):287–295, 1999. 3, 17, 121

[17] M. Dehn. Uber kombinatorische topologie. *Acta Math*, 67:123–168, 1936. 3, 121

[18] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic (TOCL)*, 10(3):1–30, 2009. 87

[19] IA Dynnikov. Arc-presentations of links: monotonic simplification. *Fund. Math*, 190(29-76):89, 2006. 18

[20] H.D. Ebbinghaus and J. Flum. *Finite model theory.* Springer Verlag, 2005. 141, 142

[21] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *American Mathematical Society*, pages 43–74, 1974. 7, 153, 154, 158

[22] R. Freivalds. Knot Theory, Jones Polynomial and Quantum Computing. *Lectures Notes in Computer Science*, 3618:15, 2005. 1

[23] C.F. Gauss. Werke. *Band 8. Teubner*, 1900. 3, 10, 17, 121, 147

[24] L. Goeritz. Bemerkungen zur knotentheorie. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 10, pages 201–210. Springer, 1934. 2, 3, 21, 70, 75, 76

[25] L. Goeritz. Bemerkungen zur knotentheorie (remarks on knot theory), 1934. Partial English translation available at http://f2.org/maths/kt/goeritz1934.html, lastchecked on 28 Feb 2011. 70, 71

[26] W. Haken. Theorie der Normalfachen, ein Isotopiekriterium fur den Kreisknoten. *Journal of Acta Mathematica*, 105, 1961. 2, 17, 18

[27] W. Hanf. Model-theoretic methods in the study of elementary logic. *The Theory of Models*, pages 132–145, 1965. 145

[28] M. Hara, S. Tani, and M. Yamamoto. Unknotting is in am ∩ co-am. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 359–364. Society for Industrial and Applied Mathematics, 2005. 18, 153

[29] J. Hass and J.C. Lagarias. The number of Reidemeister moves needed for unknotting. *Journal of the American Mathematical Society*, 14(2):399–428, 2001. 2, 18, 34

[30] J. Hass, J.C. Lagarias, and N. Pippenger. The computational complexity of knot and link problems. *Journal of the ACM (JACM)*, 46(2):185–211, 1999. 1, 2, 18, 153

[31] J. Hass and T. Nowik. Unknot diagrams requiring a quadratic number of Reidemeister moves to untangle. *Discrete and Computational Geometry*, 44(1):91–95, 2010. 2, 18, 34, 52, 70

[32] C. Hayashi. A lower bound for the number of Reidemeister moves for unknotting. *Journal of Knot Theory and its Ramifications*, 15(3):313, 2006. 2, 18, 21, 34, 75, 77

[33] N. Immerman. Expressibility as a complexity measure: results and directions. In *Second Structure in Complexity Conference*, pages 194–202, 1987. 149, 150

[34] N. Immerman. Nondeterministic space is closed under complementation. In *Structure in Complexity Theory Conference, 1988. Proceedings., Third Annual*, pages 112–115, 1988. 134

[35] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. 85

[36] J. Kari and V. Niemi. Morphic Images of Gauss Codes. In *Developments in Language Theory*, pages 144–156, 1993. 1

[37] L.H. Kauffman. Virtual knot theory. *Arxiv Preprint Math. GT/*, 9811028, 1998. 3, 6, 16, 17, 25, 121, 122, 129, 130, 131

[38] V. Kurlin. Gauss paragraphs of classical links and a characterization of virtual link groups. *Mathematical Proceedings Cambridge Phil. Soc.*, 145:129–140; 2008. 3, 6, 16, 72, 109, 113, 116, 119, 121, 129

[39] H. Lewis and C. Papadimitriou. Symmetric space-bounded computation (extended abstract). *Automata, Languages and Programming*, pages 374–384, 1980. 135

[40] L. Libkin. *Elements of finite model theory.* Springer Verlag, 2004. 141

[41] A. Lisitsa, I. Potapov, and R. Saleh. Automata on Gauss Words. In *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications*, pages 505–517. Springer, 2009. 4, 87, 103, 122

[42] A. Lisitsa, I. Potapov, and R. Saleh. Planarity of knots, register automata and logspace computability. *Language and Automata Theory and Applications*, pages 366–377, 2011. 4

[43] V.A. Liskovets and T.R. Walsh. Counting unrooted maps on the plane. *Advances in Applied Mathematics*, 36(4):364–387, 2006. 55

[44] C. Livingston. *Knot theory*, volume 24. The Mathematical Association of America, 1993. 1

[45] S.J. Lomonaco Jr and L.H. Kauffman. Topological Quantum Computing and the Jones Polynomial. *Arxiv Preprint Quant-ph/*, 0605004, 2006. 1

[46] L. Lovász and M.L. Marx. A forbidden substructure characterization of Gauss codes. *Bull. Amer. Math. Soc.*, 82(1), 1976. 3, 17, 121

[47] V. Manturov. *Knot theory.* CRC Press, 2004. 21, 75, 78, 79

[48] V.O. Manturov. A proof of Vassiliev's conjecture on the planarity of singular links. *Izvestiya: Mathematics*, 69(5):1025–1033, 2005. 3, 17, 121

[49] W.S. Massey. *A basic course in algebraic topology.* Springer, 1991. 114

[50] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1):66–97, 2003. 91

[51] M.L. Minsky. *Computation: finite and infinite machines.* 1967. 102, 103

[52] Kunio Murasugi. *Knot theory and its applications.* Birkhauser, 1996. 1

[53] J.S. Nagy. Über ein topologisches Problem von Gauss. *Mathematische Zeitschrift*, 26(1):579–592, 1927. 3, 121

[54] F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. *Mathematical Foundations of Computer Science 2001*, pages 560–572, 2001. 11

[55] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic (TOCL)*, 5(3):403–435, 2004. 85, 86, 87, 91, 96, 108, 137, 149, 150

[56] M.H.A. Newman. On theories with a combinatorial definition of" equivalence". *The Annals of Mathematics*, 43(2):223–243, 1942. 35

[57] RC Read and P. Rosenstiehl. On the Gauss crossing problem. In *Colloq. Math. Soc. Janos Bolyai*, volume 18, pages 843–876, 1976. 3, 17, 121

[58] K. Reidemeister. Elementare Begründung der Knotentheorie. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 5, pages 24–32. Springer, 1927. 1, 13, 14, 20

[59] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008. 122, 135, 136, 157

[60] P. Rosenstiehl. Solution algebrique du probleme de Gauss sur la permutation des points dintersection dune ou plusieurs courbes fermees du plan. *CR Acad. Sci. Paris Ser. AB*, 283, 1976. 3, 121

[61] P. Rosenstiehl and R.E. Tarjan. Gauss codes, planar Hamiltonian graphs, and stack-sortable permutations. *Journal of algorithms*, 5(3):375–390, 1984. 3, 17, 121

[62] B. Shtylla, L. Traldi, and L. Zulli. On the realization of double occurrence words. *Discrete Mathematics*, 309(6):1769–1773, 2009. 3, 17, 121

[63] N.J.A. Sloane et al. The on-line encyclopedia of integer sequences, 2011. *Published electronically at http://oeis.org/A103939*, 8. 3, 55

[64] C.H.O. Suh. A short proof of the hass-lagarias theorem. 2008. 2, 18, 34

[65] E.W. Weisstein. *CRC concise encyclopedia of mathematics*. CRC Pr I Llc, 2003. 150