Automata on Gauss Words

Alexei Lisitsa, Igor Potapov, and Rafiq Saleh

Department of Computer Science,
University of Liverpool, Ashton Building,
Ashton St, Liverpool L69 3BX, U.K.

Abstract. In this paper we investigate the computational complexity of knot theoretic problems and show upper and lower bounds for planarity problem of signed and unsigned knot diagrams represented by Gauss words. Due to the fact the number of crossing in knots is unbounded, the Gauss words of knot diagrams are strings over infinite (unbounded) alphabet. For establishing the lower and upper bounds on recognition of knot properties we study these problems in a context of automata models over infinite alphabet.

1 Introduction

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. In this paper we investigate the computational complexity of knot theoretic problems and show upper and lower bounds for planarity problem of signed and unsigned knot diagrams. The main goal of 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 paper were achieved by a combination of methods from knot theory, automata theory and computational complexity.

Knot theory is the area of topology 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 an ambient space. It is well established and exciting area of mathematical research with strong connections with topology, algebra and combinatorics. Examples of interactons between knot theory and computer science include works on formal language theory [1], quantum computing [2–4] and computational complexity [5].

Knots can be described in various ways, including various discrete representations. For example, a common method of describing a knot is a planar diagram called a knot diagram. A knot diagram is a projection of the knot onto a plane, where at each crossing we must indicate which section is "over" and which is "under", so as to be able to recreate the original knot. As an additional information we can also add a label ("'+" or "-") to each of the crossing for representing orientation of its strands.

A knot diagram can be encoded as a string of symbols O_i's (over) and U_i's (under) also known as *Gauss word*. The procedure of writing a Gauss word can be described as follows: Starting from a base point on the circle, write down the labels of the crossings in the counterclockwise direction, e.g. the trefoil K can

be defined by a Gauss word $U_1{}^+O_2{}^+U_3{}^+O_1{}^+U_2{}^+O_3{}^+$, where indices indicate an (arbitrary) order of the crossings in the knot diagram and signs stand for orientation of each crossing. Likewise, links can be represented by *several* Gauss words - one for each component of the link.

As you can see the construction of the Gauss word is quite straightforward by reading visited crossings travelling along a circle. The inverse problem of constructing a knot from some strings of symbols from the set of O_i's and U_i's is harder and it is not always possible. It may happen that some Gauss words would not correspond to any classical (planar) diagrams. In such case we say that a Gauss word corresponds to a non-planar knot where any of its diagram should contain virtual crossings (i.e. which are not listed in the Gauss word). Most of the problems of recognising knots properties (such as virtuality, 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 paper we address this problem and provide first known bounds for some knot problems in this context.

The central problem which we are studying in this paper is to determine whether a given Gauss word corresponds to planar or non-planar knot. Due to the fact the number of crossing in knots is unbounded, the Gauss words of knot diagrams are strings over infinite (unbounded) alphabet. In this context we cannot estimate computational complexity in terms of classical models over finite alphabets and need to consider a new hierarchy of languages and models over infinite alphabet. Such models were recently introduced in [6, 7].

In Section 2 we describe and extend the models of automata over infinite alphabet that we used for establishing the lower and upper bounds on recognition of knot properties. Then in Section 3 we show that the language of planar (non-planar) signed Gauss words can be recognised by deterministic two-way register automata by simulation of recently discovered linear time algorithm proposed in [8]. Due to the fact that the algorithm presented in [8] allows to check planarity property not only for knots but also for links we think that the proposed idea of recognising planarity by register automata can be extended for links after some minor modification. The result is final in a 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 also conjecture that planarity problem for unsigned Gauss words is harder than the the same problem for signed Gauss words and cannot be solved by register or k-pebble automata over infinite alphabet. For the case of unsigned Gauss words we provide the upper bound by showing that planarity can be checked by deterministic linearly bounded memory automata.

2 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 $d_1, \dots d_n$ where $d_i \in D$, $i = 1, \dots, n$. A language over D (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 and only if there is a constant $n \in N$ such that for any $w \in L$ and for any $d \in D \mid w \mid_d \leq n$. All languages we consider are bounded languages.

2.1 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. It is one of the weakest models of automata over infinite alphabets introduced in [6] and studied further in [7].

Definition 1 ([7]). *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 the register i then enter the state q' and move along the string according to specified direction d;)
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 d 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 d and applies non-deterministically any applicable rules. 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 in each configuration at most one transition applies.

For the purpose of this paper we modify the definition of register automata from [7] by allowing more general transition rules that allows replication of the same value in different registers. This does not affect the computational power of the model (see Lemma 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 [9, 10].

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

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 specified direction d;
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 specified direction d, where $i, j \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$.

Lemma 1. *The models of original register automaton and modified register automaton over an infinite alphabet are equivalent.*

Proof. We show that two extra types of rules of the modified model can be simulated by the original automata. The rule of type 4 is simulated by adding one extra dummy register $k+1$ and replacement of rules of modified automata of the form $q \to (q', d)$ by a pair of rules $(k+1, q) \to (q'd)$ and $q \to (q', k+1, d)$. 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. Also we replace the rule $q \to (q', i, d)$ of type 2 in the modified automata model by a pair of rules; original type 2 rule $q \to (q', i, d)$ and type 3 rule $(k+1, q) \to (q'd)$.

The rule of type 3 $(i, q) \to (q', j, d)$ of the modified model that allows storing the same value in different registers, can be simulated in original model by using the following construction.

The state of the registers of the modified automaton, that is a sequence of not necessarily different values $R = [r_1, r_2, \ldots, r_{k+1}]$ 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 $\phi : \{1, \ldots, k+1\} \to U$

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 all possible types of rules, including the type 3, in terms of pairs U, ϕ. We omit obvious details. □

Pebble Automata As an alternative model of automata over infinite alphabet, *pebble automata*(PA) was introduced in [11] and further studied in [7]. We follow the definitions in [7]. 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, the pebbles that see the same symbol as the top pebble. 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. As usual acceptance of a word is defined as reachability of one of the final states.

As expressive power concerned, in general pebble automata are incomparable with register automata [7]. We will show, however, in the Section 3 that over a class of *bounded languages*, including all languages of our interest, PA can be effectively simulated by RA.

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 automata (LBMA) is an extension of register automata with the input tape. The automaton can read and write in the tape cells the 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 to write on the tape. For every form $L \to (\ldots)$ of rules of the (modified) RA model the following is a form of rule for LBA: $L \to (\ldots, i)$, where $i \in \{1, \ldots k\}$. The effect of application of the latter is the same as of the former, plus the automaton *writes the content of the register* i in the current position on the tape before possible head movement.

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* [12]. 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 can be adapted to include one extra argument on the left-hand sides. For example, the rule $(c, i, q) \to (q', d)$ is read as "if an automaton is in a state q and observes the symbol (c, a) and a is the content of the register i then the automaton can change the state to q' and move the head along the direction d." It should be clear now how to modify the definitions of all above models to work over data words.

3 Recognisability of Knot Properties

3.1 The Language of Gauss Words

Knots which are defined as embeddings of a circle in 3-dimensional Euclidean space can be faithfully represented by finite structures, such as graphs or words. One of such discrete representations is a Gauss code, which is a word of crossing labels O ("over") and U ("under") with appropriate indices, which can be read of the projection of the knot on the plane. Given orientation of the plane one can distinguish between left and right-handed crossings (see Figure 1) which are labelled by signs $-$ and $+$, respectively.

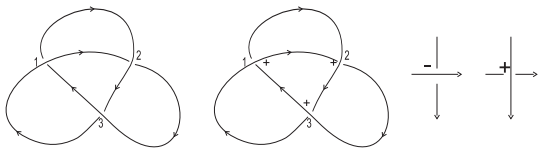

Fig. 1. Example of a knot diagram with its corresponding Gauss words(signed and unsigned)

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

Definition 3. *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* $|w|_{(U^+,n)} = |w|_{(O^+,n)} = 0$,

Definition 4. *A shadow Gauss word w is a word over alphabet $\mathbb{N}$ (i.e. finite sequence of natural numbers) such that for every $n \in \mathbb{N}$ either $|w|_n = 0$ or $|w|_n = 2$*

Shadow Gauss words can be seen as the projections of (un)signed Gauss words on the second components of their symbols. A (unsigned, signed, shadow) Gauss language is an arbitrary set of (unsigned, signed, shadow) Gauss words.

We denote the data language of all unsigned Gauss words by L_{GW} and the language of all signed Gauss words by L_{SGW}.

Proposition 1. *The languages L_{GW} 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_{GW}. 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 $b \in N = \{1, ..., n\}$. The automaton A reads the first symbol (a_i, 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 $\mathrm{b}_i = b_j$ and $\mathrm{a}_i \neq a_j$ is found and there are no further occurrences of b_i then automaton A moves right along the word and checks next symbol. If next symbol is equal to end symbol then A moves to an accepting state.

Proposition 2. *The languages L_{GW} and L_{SGW} are not recognisable by nondeterministic one-way register automata.*

Proof. We show the argument only for the case of L_{GW}. With obvious modifications it works for L_{SGW} as well. The argument is not new and was used e.g. in [12] to show non-recognizability 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 R. That means that automaton A also accepts a word $w' \notin L$ which obtained from w by replacing (U,i) with $(U,i+1)$. That is in contradiction with an assumption on A.

3.2 Planar and Non-planar Gauss Words

Every knot can be represented by a Gauss word but not every Gauss word represents a knot. For example, any attempt to reconstruct a knot diagram from the Gauss word $O_1^- O_2^- U_1^- O_3^+ U_2^- U_3^+$ will lead to new (*virtual*) crossings which are not present in the Gauss word (see Figure 2). Such an observation was one of the motivations for introducing *virtual knot theory* [13]. The Gauss word which represents a classical knot diagram, that is a diagram embeddable into a plane without virtual crossings, is called classical or *planar*. The problem of recognition of planar Gauss words have been formulated by Gauss himself and recently several algorithmic solutions for both signed and unsigned case have been proposed, e.g. in [14, 15, 13, 8].

In this section we address the question of recognisability of planarity of Gauss words by automata models and consider unsigned and signed case separately.

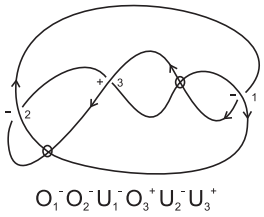

Fig. 2. Example of a virtual knot

Signed Gauss Words In this subsection we will show that a language of planar signed Gauss words can be recognised by two-way deterministic register automata. The design of automata will be based on a linear time algorithm, presented by V. Kurlin in [8]. The main idea of the algorithm is computing the least genus of the surface to which a knot diagram is embeddable without virtual crossings. For this purpose the Euler characteristics χ of the *combinatorial cell complex (Carter surface)* [16] associated with the knot diagram is computed. Computation of the Euler characteristics χ requires counting the number of faces (cycles), edges and vertices of a combinatorial cell complex (Carter surface) associated with a knot diagram. In the context of Gauss word, the number of edges corresponds to the length of the word, the number of vertices corresponds to the number of crossings (that is half the length of the word) and the number of faces can be determined by implementing a set of traversal rules.

Lemma 2. *Two-way deterministic register automaton can traverse all faces of (a complex associated with) a knot diagram which are adjacent to a crossing i.*

Proof. The traversal of adjacent face to a crossing i in the knot diagram can be done by choosing initial direction and turning left on each consecutive visited crossing staring from i. This global property of "turning left" can be defined by deterministic set of traversal rules which will take into account only local property of the current crossing and a finite information about previously visited one. In general we have 8 cases since there are two types of crossings (with a sign "+" and with a sign "-") and four directions from which we can approach each crossing, see Figure 3.

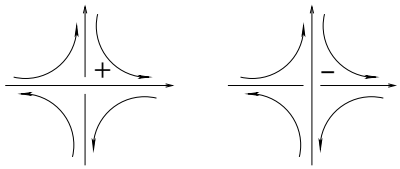

Fig. 3. Automaton moves

We follow interpretation of the local rules for selecting cycles defined in [8], 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 which correspond to the next crossing after turning (geometrically) to the left on a knot diagram. In fact on the Gauss code it will correspond to finding S' which is the counterpart of S and then choosing a symbol which is either left or a right neighbour of S'. For example, if S is O_i (U_i) then we need to choose a neighbour of U_i (O_i) in the Gauss word. In order to define whether we need a symbol 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. Now we define eight rules in the form $(D, S) \to (S', D')$ where $D, D' \in \{Right, Left\}$ and $S, S' \in \{U, O\} \times \mathbb{N} \times \{+, -\}$ where $\mathbb{N} \in \{1,2\ldots,n\}$. 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 to the specified direction.

(Right, $O_i{}^+$)$\to$ ($U_i{}^+$, Right)	(Right, $U_i{}^+$)$\to$ ($O_i{}^+$, Left)
(Left, $O_i{}^+$)$\to$ ($U_i{}^+$, Left)	(Left, $U_i{}^+$)$\to$ ($O_i{}^+$, Right)
(Right, $U_i{}^-$)$\to$ ($O_i{}^-$, Right)	(Right, $O_i{}^-$)$\to$ ($U_i{}^-$, Left)
(Left, $U_i{}^-$)$\to$ ($O_i{}^-$, Left)	(Left, $O_i{}^-$)$\to$ ($U_i{}^-$, Right)

Following above rules we can design a register automata that will keep the finite information about its previous choice of direction (Right or Left) in its state space and which will choose the Right or Left symbol of S' 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 3. *Two-way register automata with k registers on the input with t distinct symbols can simulate a counter machine with k counters bounded by t.*

Proof. Let us assume that a word on an input tape has at least t distinct symbols. The value of a counter stored in register i 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. Then we can increase (decrease) the value by looking for 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. □

Lemma 4. *Two-way deterministic register automata can compute the Euler characteristics of a signed Gauss word.*

Proof. In order to compute Euler characteristics we need to count the number of faces in a degree 4 graph G represented by a signed Gauss word, the number of edges and the number of vertices in G. The number of vertices in G is a number of distinct symbols, and the number of edges is the number of symbols in the Gauss word. Both values can be counted in a straightforward way. The number of faces can be counted by traversal of G in the following way. The automaton goes sequentially through the list of vertices. For each vertex i it traverses (as described in Lemma 2) all adjacent faces and increases a counter by one for every face F in which there are no vertices of smaller than i labels. Also the automaton counts how many times a 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 characteristics 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 1. *A language of planar signed Gauss words 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 then a signed Gauss word is planar [16, 8]. □

Unsigned Gauss Words As for signed words, an unsigned Gauss word w is called *planar* if there is a knot diagram such that w is a Gauss word read off this diagram. Before discussing the recognisability of this property by automata we briefly present an algorithm for recognition of planar unsigned Gauss word proposed by L. Kauffman in [13]. 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 word, 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 algorithm proceeds in three stages:

1. 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. 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".

To complete the algorithm description we explain what is a dually paired word [13].

Definition 5. *For a shadow Gauss word w two labels i and j are called* interlaced *in w iff $w = w_1\ i\ w_2\ j\ w_3\ i\ w_3\ j\ w_4$ or $w = w_1\ j\ w_2\ i\ w_3\ j\ w_3\ i\ w_4$.*

Definition 6. *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.*

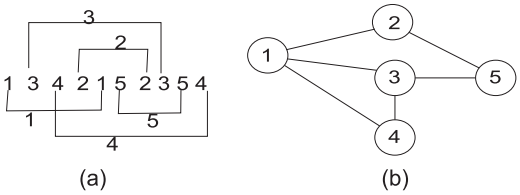

Fig. 4. Not dually paired word and the corresponding graph

An example of the shadow word which is *not* dually paired is shown in Figure 4(a).

Returning to the question on lower and upper bounds for the planarity of unsigned Gauss words 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. We do not have a proof though, neither we know a better method to check planarity, so we propose the following

Conjecture 1. Planarity of unsigned Gauss words is not recognisable by (non)-deterministic register automata.

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

Proposition 3. *The shadow Gauss language of not dually paired words is recognisable by a non-deterministic two-ways register automaton.*

Proof. Given a shadow Gauss word w define an undirected graph g_w as follows. The vertices of q_w are labels in w and there is an edge (i, j) in g_w iff i and j are interlaced. Figure 4(b) demonstrates the graph g_w for the word w shown in Figure 4(a). Now 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. □

We conjectured earlier that planarity of unsigned Gauss words is not recognisable by register automata. In order to get an upper bound one may try to extend the register automata with pebbles. However as the following proposition shows over Gauss words register automata are capable to model effects of adding any finite number of pebbles.

Proposition 4. *if an n-bounded language over infinite alphabet A can be recognised by a k-pebble automaton, then it is also recognised by a k-register automaton.*

Proof. Assume that we have to place a pebble on the symbol $x \in A$ from the word $u \cdot x \cdot v$, $u, v \in A^*$. The position of this pebble can be uniquely represented by a pair (x, l_x), where x is a symbol with a pebble and $l_x = |u|_x$, i.e. the number of occurrences of a symbol x in a word u. Thus, in order to simulate a pebble we need one register (for storing x) and a finite number of states (for keeping l_x in a state space) since l_x is bounded by a constant n. To simulate LIFO stack discipline by the register automaton one may use resister i to store a symbol which has a pebble i placed on it and the finite state structure of the automaton to store a reference to the register which corresponds to the current top of the stack. □

Theorem 2. *Planarity of unsigned Gauss words is recognisable by deterministic linearly bounded memory automata*

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 3 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. □

4 Conclusion

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

words can be recognized by deterministic two-way register automata, while for the same problem for unsigned words we demonstrated an upper bound in terms of automata with linearly bounded memory. The obvious next step is to try to establish a lower and better upper bound for the latter problem. More generally, automata based approach opens new 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 reach 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.

References

1. Kari, J., Niemi, V.: Morphic Images of Gauss Codes. In: Developments in Language Theory. (1993) 144–156
2. Abramsky, S.: Temperley-Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics. Mathematics of Quantum Computation and Quantum Technology (2007)
3. Freivalds, R.: Knot Theory, Jones Polynomial and Quantum Computing. Lectures Notes in Computer Science **3618** (2005) 15
4. Lomonaco Jr, S., Kauffman, L.: Topological Quantum Computing and the Jones Polynomial. Arxiv Preprint Quant-ph/ **0605004** (2006)
5. Hass, J., Lagarias, J., Pippenger, N.: The computational complexity of knot and link problems. Journal of the ACM (JACM) **46**(2) (1999) 185–211
6. Kaminski, M., Francez, N.: Finite-memory automata. In: Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on. (1990) 683–688
7. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic (TOCL) **5**(3) (2004) 403–435
8. Kurlin, V.: Gauss phrases realizable by classical links. Arxiv Preprint Math. GT/ **0610929** (2006)
9. David, C.: Mots et données infinies. Master's thesis, LIAFA (2004)
10. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. In: LICS. Volume 6. (2006) 17–26
11. Milo, T., Suciu, D., Vianu, V.: Typechecking for XML transformers. Journal of Computer and System Sciences **66**(1) (2003) 66–97
12. Bjorklund, H., Schwentick, T.: On Notions of Regularity for Data Languages. Lectures Notes in Computer Science **4639** (2007) 88
13. Kauffman, L.: Virtual knot theory. Arxiv Preprint Math. GT/ **9811028** (1998)
14. Cairns, G., Elton, D.: The planarity problem for signed Gauss words. Journal of Knot Theory and its Ramifications **2**(4) (1993) 359–367
15. Cairns, G., Elton, D.: The Planarity Problem II. Journal of Knot Theory and its Ramifications **5** (1996) 137–144
16. Carter, J.: Classifying immersed curves. In: Proc. Amer. Math. Soc. Volume 111. (1991) 281–287