Posets of twisted involutions in Coxeter groups

by

Christian Hoffmeister

A thesis submitted to the Carl-Friedrich-Gauß-Faculty
in conformity with the requirements
for the degree of Master of Science (Mathematics)

Technische Universität Braunschweig
Brunswick, Lower Saxony, Germany
October 2012

E-Mail: choffmeister@choffmeister.de

Student number: 2944344

Contents

List of Figures

1 Coxeter groups

A Coxeter group, named after Harold Scott MacDonald Coxeter, is an abstract group generated by involutions with specific relations between these generators. A simple class of Coxeter groups are the symmetry groups of regular polyhedras in the Euclidean space.

The symmetry group of the square for example can be generated by two reflections s, t, whose stabilized hyperplanes enclose an angle of $\pi/4$. In this case the map st is a rotation in the plane by $\pi/2$. So we have $s^2 = t^2 = (st)^4 = \mathrm{id}$. In fact, this reflection group is determined up to isomorphy by s, t and these three relations [7, Theorem 1.9]. Furthermore it turns out, that the finite reflection groups in the Euclidean space are precisely the finite Coxeter groups [7, Theorem 6.4].

In this chapter we compile some basic well-known facts on Coxeter groups, based on [7].

1.1 Introduction to Coxeter groups

Definition 1.1. Let $S = \{s_1, \dots, s_n\}$ be a finite set of symbols and

$$R = \{m_{ij} \in \mathbb{N} \cup \infty : 1 \leq i, j \leq n\}$$

a set numbers (or ∞) with $m_{ii} = 1$, $m_{ij} > 1$ for $i \neq j$ and $m_{ij} = m_{ji}$. Then the free represented group

$$W = \langle S \mid (s_i s_j)^{m_{ij}} \rangle$$

is called a ***Coxeter group*** and (W, S) the corrosponding ***Coxeter system***. The cardinality of S is called the ***rank*** of the Coxeter system (and the Coxeter group).

From the definiton we see, that Coxeter groups only depend on the cardinality of S and the relations between the generators in S. A common way to visualize this information are Coxeter graphs.

Definition 1.2. Let (W, S) be a Coxeter system. Create a graph by adding a vertex for each generator in S. Let $(s_i s_j)^m = 1$. In case $m = 2$ the two corrosponding vertices have no connecting edge. In case $m = 3$ they are connected by an unlabed edge. For $m > 3$ they have an connecting edge with label m. We call this graph the ***Coxeter graph*** of our Coxeter system (W, S).

Definition 1.3. Let (W, S) be a Coxeter system. For an arbitrary element $w \in W$ we call a product $s_{i_1} \cdots s_{i_n} = w$ of generators $s_{i_1} \dots s_{i_n} \in S$ an ***expression*** of w. Any expression that can be obtained from $s_{i_1} \cdots s_{i_n}$ by omitting some (or all) factors, is called a ***subexpression*** of w.

The present relations between the generators of a Coxeter group allow us to rewrite expressions. Hence an element $w \in W$ can have more than one expression. Obviously any element $w \in W$ has infinitly many expressions, since any expression $s_{i_1} \cdots s_{i_n} = w$ can be extended by applying $s_1^2 = 1$ from the right. But there must be a smallest number of generators needed to receive w. For example the neutral element e can be expressed by the empty expression. Or each generator $s_i \in S$ can be expressed by itself, but any expression with less factors (i.e. the empty expression) is unequal to s_i.

Definition 1.4. Let (W, S) be a Coxeter system and $w \in W$ an element. Then there are some (not necessarily distinct) generators $s_i \in S$ with $s_1 \cdots s_r = w$. We call r the ***expression length***. The smallest number $r \in \mathbb{N}_0$ for that w has an expression of length r is called the ***length*** of w and each expression of w, that is of minimal length, is called ***reduced expression***. The map

$$l : W \to \mathbb{N}_0$$

that maps each element in W to its length is called ***length function***.

Definition 1.5. Let (W, S) be a Coxeter system. We define

$$D_R(w) := \{s \in S : l(ws) < l(w)\}$$

as the ***right descending set*** of w. The analogue left version

$$D_L(w) := \{s \in S : l(sw) < l(w)\}$$

is called ***left descending set*** of w. Since the left descending set is not need in this paper, we will often call the right descending just ***descending set*** of w.

The next lemma yields some useful identities and relations for the length function.

Lemma 1.6. *[7, Section 5.2]. Let (W, S) be a Coxeter system, $s \in S$, $u, w \in W$ and $l : W \to \mathbb{N}$ the length function. Then*

1. $l(w) = l(w^{-1})$,
2. $l(w) = 0$ *iff* $w = e$,
3. $l(w) = 1$ *iff* $w \in S$,
4. $l(uw) \leq l(u) + l(w)$,
5. $l(uw) \geq l(u) - l(w)$ *and*
6. $l(ws) = l(w) \pm 1$.

1.2 Exchange and Deletion Condition

We now obtain a way to get a reduced expression of an arbitrary element $s_1 \cdots s_r = w \in W$.

Definition 1.7. Let (W, S) be a Coxeter system. Any element $w \in W$ that is conjugated to an generator $s \in S$ is called ***reflection***. Hence the set of all reflections in W is

$$T = \bigcup_{w \in W} wSw^{-1}.$$

Theorem 1.8 (Strong Exchange Condition)**.** *[7, Theorem 5.8]. Let (W, S) be a Coxeter system, $w \in W$ an arbitrary element and $s_1 \cdots s_r = w$ with $s_i \in S$ a not necessarily reduced expression for w. For each reflection $t \in T$ with $l(wt) < l(w)$ there exists an index i for which $wt = s_1 \cdots \hat{s}_i \cdots s_r$, where $\hat{s}_i$ means omission. In case we start from a reduced expression, then i is unique.*

The Strong Exchange Condition can be weaken, when insisting on $t \in S$ to receive the following corollary.

Corollary 1.9 (Exchange Condition)**.** *Let (W, S) be a Coxeter system, $w \in W$ an arbitrary element and $s_1 \cdots s_r = w$ with $s_i \in S$ a not necessarily reduced expression for w. For each generator $s \in S$ with $l(ws) < l(w)$ there exists an index i for which $ws = s_1 \cdots \hat{s}_i \cdots s_r$, where $\hat{s}_i$ means omission. In case we start from a reduced expression, then i is unique.*

Proof. Directly from Strong Exchange Condition. □

Remark 1.10. Note that both, Strong Exchange Condition and Exchange Condition have an analogues left-sided version

$$l(tw) < l(w) \Rightarrow tw = ts_1 \cdots s_k = s_1 \cdots \hat{s}_i \cdots s_k$$

for all reflections $t \in T$, hence for all generators $s \in S$ in particular.

Corollary 1.11 (Deletion Condition)**.** *[7, Corollary 5.8]. Let (W, S) be a Coxeter system, $w \in W$ and $w = s_1 \cdots s_r$ with $s_i \in S$ an unreduced expression of w. Then there exist two indices $i, j \in \{1, \cdots, r\}$ with $i < j$, such that $w = s_1 \cdots \hat{s}_i \cdots \hat{s}_j \cdots s_r$, where $\hat{s}_i$ and $\hat{s}_j$ mean omission.*

Proof. Since the expression is unreduced there must be an index j for that the twisted length shrinks. That means for $w' = s_1 \cdots s_{j-1}$ is $l(w's_j) < l(w')$. Using the Exchange Condition we get $w's_j = s_1 \cdots \hat{s}_i \cdots s_{j-1}$ yielding $w = s_1 \cdots \hat{s}_i \cdots \hat{s}_j \cdots s_r$. □

This corollary is called ***Deletion Condition*** and allows us to reduce expressions, i.e. to find a subexpression that is reduced. Due to the Deletion Condition any unreduced expression can be reduced by omitting an even number of generators (we just have to apply the Deletion Condition inductively).

The Strong Exchange Condition, the Exchange Condition and the Deletion Condition, are some of the most powerful tools when investigating properties of Coxeter groups. We can use the second to prove a very handy property of Coxeter groups. The intersection of two parabolic subgroups is again a parabolic subgroup.

Definition 1.12. Let (W, S) be a Coxeter system. For a subset of generators $I \subset S$ we call the subgroup $W_I \leq W$, that is generated by the elements in I with the corrosponding relations, a ***parabolic subgroup*** of W.

Lemma 1.13. *[7, Section 5.8]. Let (W, S) be a Coxeter system and $w \in W$. Let $w = s_1 \cdots s_k$ any reduced expression for w. Then $\{s_1, \ldots, s_k\} \subset S$ is independent of the particular choosen reduced expression. It only depends on w itself.*

This means, that two reduced expressions for an element $w \in W$ use exactly the same generators. A related fact, is the following lemma.

Lemma 1.14. *[7, Section 5.8]. Let (W, S) be a Coxeter system and $I, J \subset S$ two subsets of generators. Then $W_I \cap W_J = W_{I \cap J}$.*

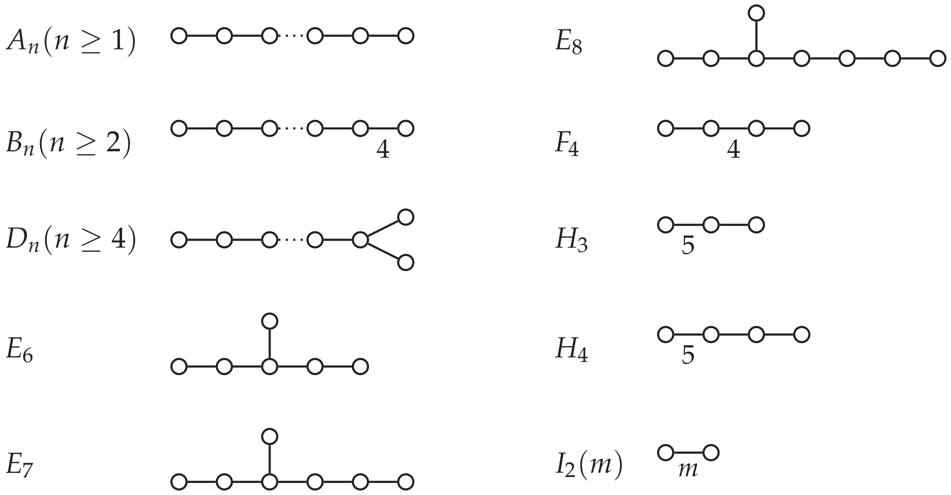

Figure 1.1: All types of irreducible finite Coxeter systems

1.3 Finite Coxeter groups

Coxeter groups can be finite and infinite. A simple example for the former category is the following. Let $S = \{s\}$. Due to definition it must be $s^2 = e$. So W is isomorph to $\mathbb{Z}_2$ and finite. An example for an infinite Coxeter group can be obtained from $S = \{s, t\}$ with $s^2 = t^2 = e$ and $(st)^\infty = e$ (so we have no relation between s and t). Obviously the element st has infinite order forcing W to be infinite. But there are infinite Coxeter groups without an ∞-relation between two generators, as well. An example for this is W obtained from $S = \{s_1, s_2, s_3\}$ with $s_1^2 = s_2^2 = s_3^2 = (s_1 s_2)^3 = (s_2 s_3)^3 = (s_3 s_1)^3 = e$. But how can one decide weather W is finite or not?

To provide a general answer to this question we fallback to a certain class of Coxeter groups, the irreducible ones.

Definition 1.15. A Coxeter system is called ***irreducible***, if the corresponding Coxeter graph is connected. Else, it is called ***reducible***.

If a Coxeter system is reducible, then its graph has more than one component and each component corrosponds to a parabolic subgroup of W.

Proposition 1.16. *[7, Proposition 6.1]. Let (W, S) be a reducible Coxeter system. Then there exists a partition of S into I, J with $(s_i s_j)^2 = e$ whenever $s_i \in I, s_j \in J$ and W is isomorph to the direct product of the two parabolic subgroups W_I and W_J.*

This proposition tells us, that an arbitray Coxeter system is finite iff its irreducible parabolic subgroups are finite. Therefore we can indeed fallback to irreducible Coxeter systems without loss of generality. If we could categorize all irreducible finite Coxeter systems, we could categorize all finite Coxeter systems. This is done by the following theorem:

Theorem 1.17. *[7, Theorem 6.4]. The irreducible finite Coxeter systems are exactly the ones in Figure 1.1.*

This allows us to decide with ease, if a given Coxeter system is finite. Take its irreducible parabolic subgroups and check, if each is of type A_n, B_n, D_n, E_6, E_7, E_8, F_4, H_3, H_4 or $I_2(m)$.

1.4 Compact hyperbolic Coxeter groups

TODO

1.5 Bruhat ordering

We now investigate ways to partially order the elements of a Coxeter group. Futhermore, this ordering should be compatible with the length function, i.e. for $w, v \in W$ we have $l(w) < l(v)$ whenever $w \leq v$.

Definition 1.18. Let M be a set. A binary relation $\leq$ is called a ***partial order*** over M, if for all $a, b, c \in M$ it satisfies the conditions

1. $a \leq a$ (***reflexivity***),
2. $a \leq b \wedge b \leq a \Rightarrow a = b$ (***antisymmetry***) and
3. $a \leq b \wedge b \leq c \Rightarrow a \leq c$ (***transitivity***).

In this case $(M, \leq)$ is called a ***poset***. If two elements $a \leq b \in M$ are immediate neighbors, i.e. there is no third element $c \in M$ with $a \leq c \leq b$ we say that b ***covers*** a. A poset is called ***graded poset*** if there is a map $\rho : M \to \mathbb{N}$ so that $\rho(b) - 1 = \rho(a)$ whenever b covers a. In this case ρ is called the ***rank function*** of the graded poset If for any two elements $a, b \in M$ there is an element $c \in M$ with $a \leq c$ and $b \leq c$, then a poset is called ***directed poset***.

Definition 1.19. Let $(M, \leq)$ be a poset and $a, b \in M$. Then we call $\{c \in M : a \leq c \leq b\}$ an ***interval*** and denote it by $[a, b]_{\leq}$. The set $\{c \in M : a < c < b\}$ is called an ***open interval*** and is denoted by $(a, b)_{\leq}$. In both cases we can omit the $\leq$, if the relation is clear from context.

Definition 1.20. Let $(M, \leq)$ be a poset. The ***Hasse diagram*** of the poset is the graph obtained in the following way: Add a vertex for each element in M. Then add a directed edge from vertex a to b whenever b covers a.

Example 1.21. Suppose we have an arbitrary set M. Then the powerset $\mathcal{P}(M)$ can be partially ordered by the subset relation, so $(\mathcal{P}(M), \subseteq)$ is a poset. Indeed this poset is always graded with the cardinality function as rank function. In Figure 1.2 we see the Hasse diagram of this poset with $M = \{x, y, z\}$.

Definition 1.22. Let (W, S) be a Coxeter system and $T = \cup_{w \in W} wSw^{-1}$ the set of all reflections in W. We write $w' \to w$ if there is a $t \in T$ with $w't = w$ and $l(w') < l(w)$. If there is a sequence $w' = w_0 \to w_1 \to \ldots \to w_m = w$ we say $w' < w$. The resulting relation $w' \leq w$ is called ***Bruhat ordering***, denoted by $\mathrm{Br}(W)$.

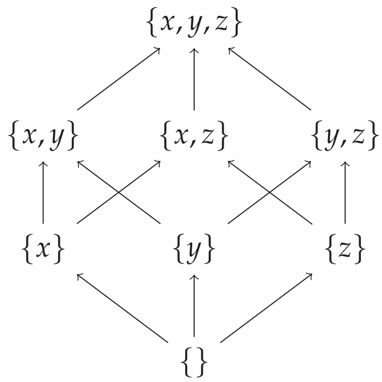

Figure 1.2: Hasse diagram of the set of all subsets of $\{x, y, z\}$ order by the subset relation

Lemma 1.23. *Let (W, S) be a Coxeter system. Then $\mathrm{Br}(W)$ is a poset.*

Proof. The Bruhat ordering is reflexive by definition. Since the elements in sequences $e \to w_1 \to w_2 \to \dots$ are strictly ascending in length, it must be antisymmetric. By concatenation of sequences we get the transitivity. $\square$

What we really want is the Bruhat ordering to be graded with the length function as rank function. By definition we already have $v < w$ iff $l(v) < l(w)$, but its not that obvious that two immediately adjacent elements differ in length by exactly 1. Beforehand let us just mention two other partial orderings, where this property is obvious by definition:

Definition 1.24. Let (W, S) be a Coxeter system. The ordering $\leq_R$ defined by $u \leq_R w$ iff $uv = w$ for some $u \in W$ with $l(u) + l(v) = l(w)$ is called the ***right weak ordering***. The left-sided version $u \leq_L w$ iff $vu = w$ is called the ***left weak ordering***.

To ensure the Bruhat ordering is graded as well, we need another characterization of the Bruhat ordering in terms of subexpressions.

Proposition 1.25. *[7, Proposition 5.9]. Let (W, S) be a Coxeter system, $u, w \in W$ with $u \leq w$ and $s \in S$. Then $us \leq w$ or $us \leq ws$ or both.*

Proof. We can reduce the proof (**TODO**why?) to the case $u \to w$, i.e. $ut = w$ for a $t \in T$ with $l(v) < l(u)$. Let $s = t$. Then $us \leq w$ and we are done. In case $s \neq t$ there are two alternatives for the lengths. We can have $l(us) = l(u) - 1$ which would mean $us \to u \to w$, so $us \leq w$.

Assume $l(us) = l(u) + 1$. For the reflection $t' = sts$ we get $(us)t' = ussts = uts = ws$. So we have $us \leq ws$ iff $l(us) < l(ws)$. Suppose this is not the case. Since we have assumed $l(us) = l(u) + 1$ any reduced expression $u = s_1 \cdots s_r$ for u yields a reduced expression $us = s_1 \cdots s_r s$ for us. With the Strong Exchange Condition we can obtain $ws = ust'$ from us by omitting one factor. This omitted factor cannot be s since $s \neq t$. This means $ws = s_1 \cdots \hat{s}_i \cdots s_r s$ and so $ws = s_1 \cdots \hat{s}_i \cdots s_r$, contradicting to our assumption $l(u) < l(w)$ $\square$

Theorem 1.26. *[7, Theorem 5.10]. Let (W, S) be a Coxeter system and $w \in W$ with any reduced expression $w = s_1 \cdots s_r$ and $s_i \in S$. Then $u \leq w$ (in the Bruhat ordering) iff u can be obtained as a subexpression of this reduced expression.*

Proof. TODO □

Corollary 1.27. *Let $u, w \in W$. Then the interval $[u, w]$ in the Bruhat order $\mathrm{Br}(W)$ is finite.*

Proof. We have $[u, w] \subseteq [e, w]$. All elements $v \in [e, w]$ can be obtained as subexpressions of one fixed reduced expression for w. Let $s_1 \dots s_k = w$ be such an reduced expression. Then there are at most 2^k many subexpressions, hence $[u, w]$ is finite. □

This characterization of the Bruhat ordering is very handy. With it and the following short lemma we will be in the position to show that $\mathrm{Br}(W)$ is graded with rank function l.

Lemma 1.28. *[7, Lemma 5.11]. Let (W, S) be a Coxeter system, $u, w \in W$ with $u < w$ and $l(w) = l(u) + 1$. In case there is a generator $s \in S$ with $u < us$ but $us \neq w$, then both $w < ws$ and $us < ws$.*

Proof. Due to Proposition 1.25 we have $us \leq w$ or $us \leq ws$. Since $l(us) = l(w)$ and $us \neq w$ the first case is impossible. So $us \leq ws$ and because of $u \neq w$ already $us < ws$. In turn, $l(w) = l(us) < l(ws)$, forcing $w < ws$. □

Proposition 1.29. *[7, Proposition 5.11]. Let (W, S) be a Coxeter system and $u < w$. Then there are elements $w_0, \dots, w_m \in W$ such that $u = w_0 < w_1 < \dots < w_m = w$ with $l(w_i) = l(w_{i-1}) + 1$ for $1 \leq i \leq m$.*

Proof. We induce on $r = l(u) + l(w)$. In case $r = 1$ we have $u = e$ and $w = s$ for an $s \in S$ and are done. Conversely suppose $r > 1$. Then there is a reduced expression $w = s_1 \cdots s_r$ for w. Lets fix this expression. Then $l(ws_r) < l(w)$. Thanks to Theorem 1.26 there must be a subexpression of w with $u = s_{i_1} \cdots s_{i_q}$ for some $i_1 < \dots < i_q$. We distinguish between two cases:

$u < us$: If $i_q = r$, then $us = s_{i_1} \cdots s_{i_q} s = s_{i_1} \cdots s_{i_{q-1}}$ which is also a subexpression of ws. This yields $u < us \leq ws < w$. Since $l(ws) < r$ there is, by induction, a sequence of the desired form. The last step from ws to w also differs in length by exactly 1, so we are done. If $i_q < r$ then u is itself already a subexpression of ws and we can again find a sequence from u to ws strictly ascending length by 1 in each step and have one last step from ws to w also increasing length by 1.

$us < u$: Then by induction we can find a sequence from us to w, say $us = w_0 < \dots < w_m = w$, where the lengths of neighbored elements differ by exactly 1. Since $w_0 s = u > us = w_0$ and $w_m s = ws < w = w_m$ there must be a smallest index $i \geq 1$, such that $w_i s < w_i$, which we choose. Suppose $w_i \neq w_{i-1} s$. We have $w_{i-1} < w_{i-1} s \neq w_i$ and due to Lemma 1.28 we get $w_i < w_i s$. This contradicts to the minimality of i. So

$w_i = w_{i-1}s$. For all $1 \leq j < i$ we have $w_j \neq w_{j-1}s$, because of $w_j < w_j s$. Again we apply Lemma 1.28 to receive $w_{j-1}s < w_j s$. Alltogether we can construct a sequence

$$u = w_0 s < w_1 s < \ldots < w_{i-1}s = w_i < w_{i+1} < \ldots w_m = w,$$

which matches our assumption. □

Corollary 1.30. *Let (W, S) be a Coxeter system and $\mathrm{Br}(W)$ the Bruhat ordering poset of W. Then $\mathrm{Br}(W)$ is graded with $l : W \to \mathbb{N}$ as rank function.*

Proof. Let $u, w \in W$ with w covering u. Then Proposition 1.29 says there is a sequence $u = w_0 < \ldots < w_m = w$ with $l(w_i) = l(w_{i-1}) + 1$ for $1 \leq i \leq m$. Since w covers u it must be $m = 1$ and so $u < w$ with $l(w) = l(u) + 1$. □

Theorem 1.31 (Lifting Property)**.** *[3, Theorem 1.1]. Let (W, S) be a Coxeter system and $v, w \in W$ with $v \leq w$. Suppose $s \in S$ with $s \in D_R(w)$. Then*

1. *$vs \leq w$,*
2. *$s \in D_R(v) \Rightarrow vs \leq ws$.*

Proof. We use the alternative subexpression characterization of the Bruhat ordering from Theorem 1.26.

1. Since $s \in D_R(w)$ there exists a reduced expression $w = s_1 \cdots s_r$ with $s_r = s$. Due to $v \leq w$ we can obtain v as a subexpression $v = s_{i_1} \cdots s_{i_q}$ from w. If $i_q = r$ then $vs = s_{i_1} \cdots s_{i_q} s = s_{i_1} \cdots s_{i_{q-1}}$ is also a subexpression of w. Else, if $i_q \neq r$ then v is a subexpression of $ws = s_1 \cdots s_{r-1}$ and so vs is again a subexpression of $w = s_1 \cdots s_{r-1} s$. In both cases we get $vs \leq w$.
2. If we additionally assume $s \in D_R(v)$ then we can always find a reduced expression $w = s_1 \cdots s_r$ with $s_r = s$ having $u = s_{i_1} \cdots s_{i_q}$ as subexpression with $s_{i_q} = s$. This yields $vs = s_{i_1} \cdots s_{i_{q-1}} \leq s_1 \cdots s_{r-1} = ws$. □

The Lifting Property seems quite innocent, but when trying to investigate facts around the Bruhat ordering it proves to be one of the key tools in many cases.

Proposition 1.32. *[2, Proposition 7]. The poset $\mathrm{Br}(W)$ is directed.*

Proof. Let $u, v \in W$. We need to find an element $w \in W$ with $u \leq w$ and $v \leq w$. For that, we induce on $r = l(u) + l(w)$. For $r = 0$ we have $u = v = e$ and can choose $w = e$. So let $r > 0$. Because of symmetry we can assume $l(u) > 0$, hence $u \neq e$ and so there is a $s \in S$ with $us < u$. By induction hypothesis there is a $w \in W$ with $us < w$ and $v < w$. Consider two cases:

$ws < w$: Then $s \in D_R(w)$ and with Lifting Property we have $u = uss \leq w$, so both $u \leq w$ and $v \leq w$.

$ws > w$: **TODO** □

Corollary 1.33. *[2, Proposition 8].*

1. *Let W be finite, then there exists an unique element $w_0 \in W$ with $w \leq w_0$ for all $w \in W$.*

2. *If W contains an element w, with $D_R(w) = S$, then W is finite and w is the unique element w_0.*

Proof. 1. Assume there are two elements $u, v \in W$ of maximal rank. Since $\mathrm{Br}(W)$ is directed, there is an element $w \in W$ with $u \leq w$ and $v \leq w$. Because $\mathrm{Br}(W)$ is graded, we have $l(w) > l(u) = l(v)$, contradicting to the maximality of u and v.

2. We want to show, that $v < w$ for all $v \in W$. For that, we induce on $r = l(v)$. If $r = 0$, then $v = e \leq w$. Let $r > 0$. Then there is a $s \in S$ with $us < u$. By induction, $us \leq w$. Since $s \in D_R(w)$, we have $uss = u \leq w$ by Lifting Property and are done with our induction. This yields $W = [e, w]$ and since by Corollary 1.27 intervals in the Bruhat order are finite, W is finite, too. □

2 Twisted involutions in Coxeter groups

In this section we focus on a certain subset of elements in Coxeter groups, the so called twisted involutions. From now on (and in the next sections) we fix some symbols to have always the same meaning (some definitions follow later):

(W, S)	A Coxeter system with generators S and elements W.
s	A generator in S.
u, v, w	A element in the Coxeter group W.
θ	A Coxeter system automorphism of (W, S) with $\theta^2 = \mathrm{id}$.
$\mathcal{I}_\theta$	The set of θ-twisted involutions of W.
$\underline{S}$	A set of symbols, $\underline{S} = \{\underline{s} : s \in S\}$.

2.1 Introduction to twisted involutions

Definition 2.1. An automorphism $\theta : W \to W$ with $\theta(S) = S$ is called a ***Coxeter system automorphism*** of (W, S). We always assume $\theta^2 = \mathrm{id}$.

Definition 2.2. Each $w \in W$ with $\theta(w) = w^{-1}$ is called a ***θ-twisted involution*** or just ***twisted involution***, if θ is clear from the context. The set of all θ-twisted involutions in W is denoted by $\mathcal{I}_\theta(W)$. Often we just omit the Coxeter group and write $\mathcal{I}_\theta$, when it is clear from the context which W is meant.

Example 2.3. Let $\theta = \mathrm{id}_W$. Then θ is an Coxeter system automorphism and the set of all id-twisted involutions coincides with the set of all ordinary involutions of W.

The next example is more helpfull, since it reveals a way to think of $\mathcal{I}_\theta$ as a generalization of ordinary Coxeter groups.

Example 2.4. Let θ be a automorphism of $W \times W$ with $\theta : (u, w) \mapsto (w, u)$. Then θ is an Coxeter system automorphism of the Coxeter system $(W \times W, S \times S)$ and the set of twisted involutions is

$$\mathcal{I}_\theta = \{(w, w^{-1}) \in W \times W : w \in W\}.$$

This yields a canonical bijection between $\mathcal{I}_\theta$ and W.

The map we define right now is of great importance to this whole paper, since it is needed to define the poset, the main thesis is about.

Definition 2.5. Let $\underline{S} := \{\underline{s} : s \in S\}$ be a set of symbols. Each element in $\underline{S}$ acts from the right on W by the following definition:

$$w\underline{s} = \begin{cases} ws & \text{if } \theta(s)ws = w, \\ \theta(s)ws & \text{else.} \end{cases}$$

This action can be extended on the whole free monoid over $\underline{S}$ by

$$w\underline{s}_1\underline{s}_2 \dots \underline{s}_k = (\dots((w\underline{s}_1)\underline{s}_2)\dots)\underline{s}_k.$$

If $w\underline{s} = \theta(s)ws$, then we say $\underline{s}$ ***acts by twisted conjugation*** on w. Else we say $\underline{s}$ ***acts by multiplication*** on w.

Note that this is no group action. For example let W be a Coxeter group with two generators s, t satisfying $\mathrm{ord}(st) = 3$ and let $\theta = \mathrm{id}$. Then $sts = tst$, but

$$e\underline{t}\underline{s} = s\underline{t}\underline{s} = tst\underline{s} = sts\underline{s} = t \neq s = tst\underline{t} = sts\underline{t} = t\underline{s}\underline{t} = e\underline{s}\underline{t}.$$

Definition 2.6. Let $k \in \mathbb{N}$ and $s_i \in S$ for all $1 \leq i \leq k$. Then an expression $e\underline{s}_1 \dots \underline{s}_k$, or just $\underline{s}_1 \dots \underline{s}_k$, is called θ***- twisted expression***. If θ is clear from the context, we omit θ and call it ***twisted expression***. A twisted expression is called ***reduced twisted expression***, if there is no $k' < k$ with $\underline{s}'_1 \dots \underline{s}'_{k'} = \underline{s}_1 \dots \underline{s}_k$.

Lemma 2.7. *[6, Lemma 3.4]. Let $w \in \mathcal{I}_\theta$ and $s \in S$. Then*

$$w\underline{s} = \begin{cases} ws & \text{if } l(\theta(s)ws) = l(w), \\ \theta(s)ws & \text{else.} \end{cases}$$

Proof. Suppose $\underline{s}$ acts by multiplication on w. Then $\theta(s)ws = w$ and so $l(\theta(s)ws) = l(w)$. Conversely, suppose $l(\theta(s)ws) = l(w)$. **TODO** $\square$

Lemma 2.8. *We have $l(ws) < l(w)$ iff $l(w\underline{s}) < l(w)$.*

Proof. Suppose $\underline{s}$ acts by multiplication on w. Then $w\underline{s} = ws$ and there is nothing to prove. So suppose $\underline{s}$ acts by twisted conjugation on w. If $l(ws) < l(w)$, then Lemma 1.6 yields $l(ws) + 1 = l(w)$. Assuming $l(w\underline{s}) = l(\theta(s)ws) = l(w)$ would imply, that $\underline{s}$ acts by multiplication on w due to Lemma 2.7, which is a contradiction. So $l(w\underline{s}) = l(\theta(s)ws) < l(w)$. Conversely, suppose $l(w\underline{s}) < l(w)$. Then Lemma 1.6 says $l(w\underline{s}) = l(\theta(s)ws) = l(w) - 2$ and so $l(ws) = l(w) - 1$. $\square$

Lemma 2.9. *For all $w \in W$ and $s \in S$ we have $w\underline{s}\underline{s} = w$.*

Proof. For $w\underline{s}$ there are two cases. Suppose $\underline{s}$ acts by multiplication on w, i.e. $\theta(s)ws = w$. For $ws\underline{s}$ there are again two possible options:

$$ws\underline{s} = \begin{cases} wss = w & \text{if } \theta(s)wss = ws, \\ \theta(s)wss = ws & \text{else.} \end{cases}$$

The second option contradicts itself.

Now suppose $\underline{s}$ acts by twisted conjugation on w. This means $\theta(s)ws \neq w$ and for $(\theta(s)ws)\underline{s}$ there are again two possible options:

$$(\theta(s)ws)\underline{s} = \begin{cases} \theta(s)wss = \theta(s)w & \text{if } \theta(s)\theta(s)wss = \theta(s)ws, \\ \theta(s)\theta(s)wss = w & \text{else.} \end{cases}$$

The first option is impossible since $\theta(s)\theta(s)wss = w$ and we have assumed $\theta(s)ws \neq w$. Hence the only possible cases yield $w\underline{s}\underline{s} = w$. $\square$

Remark 2.10. This lemma allows us to to rewrite equations of twisted expressions. For example

$$u = w\underline{s} \iff u\underline{s} = w\underline{s}\underline{s} = w.$$

This can be iterated to get

$$u = w\underline{s}_1 \dots \underline{s}_k \iff u\underline{s}_k \dots \underline{s}_1 = w.$$

Lemma 2.11. *For all θ, $w \in W$ and $s \in S$ it holds that $w \in \mathcal{I}_\theta$ iff $w\underline{s} \in \mathcal{I}_\theta$.*

Proof. Let $w \in \mathcal{I}_\theta$. For $w\underline{s}$ there are two cases. Suppose $\underline{s}$ acts by multiplication on w. Then we get

$$\theta(ws) = \theta(\theta(s)wss) = \theta^2(s)\theta(w) = sw^{-1} = (ws^{-1})^{-1} = (ws)^{-1}.$$

Suppose $\underline{s}$ acts by twisted conjugation on w. Then we get

$$\theta(\theta(s)ws) = \theta^2(s)\theta(w)\theta(s) = sw^{-1}\theta(s) = (\theta^{-1}(s)ws^{-1})^{-1} = (\theta(s)ws)^{-1}.$$

In both cases $w\underline{s} \in \mathcal{I}_\theta$.

Now let $w\underline{s} \in \mathcal{I}_\theta$. Suppose $\underline{s}$ acts by multiplication on w. Then

$$\theta(w) = \theta(\theta(s)ws) = \theta^2(s)\theta(ws) = s(ws)^{-1} = ss^{-1}w^{-1} = w^{-1}.$$

Suppose $\underline{s}$ acts by twisted conjugation on w. Then

$$\begin{aligned}\theta(w) &= \theta(\theta(s)\theta(s)wss) = \theta^2(s)\theta(\theta(s)ws)\theta(s)\\ &= s(\theta(s)ws)^{-1}\theta(s) = s(s^{-1}w^{-1}\theta(s^{-1})\theta(s) = w^{-1}.\end{aligned}$$

In both cases $w \in \mathcal{I}_\theta$. □

A remarkable property of the action from Definition 2.5 is its e-orbit. As the following lemma shows, it coincides with $\mathcal{I}_\theta$.

Lemma 2.12. *[6, Proposition 3.5]. Fix θ. Then the set of θ-twisted involutions coincides with the set of all θ-twisted expressions.*

Proof. By Lemma 2.11, each twisted expression is in $\mathcal{I}_\theta$, since $e \in \mathcal{I}_\theta$. So let $w \in \mathcal{I}_\theta$. If $l(w) = 0$, then $w = e \in \mathcal{I}_\theta$. So assume $l(w) = r > 0$ and that we have already proven, that every twisted involution $w' \in \mathcal{I}_\theta$ with $\rho(w') < r$ has a twisted expression. If w has a reduced twisted expression ending with $\underline{s}$, then w also has a reduced expression (in S) ending with s and so $l(ws) < l(w)$. With Lemma 2.8 we get $l(w\underline{s}) < l(w)$. By induction $w\underline{s}$ has a twisted expression and hence $w = (w\underline{s})\underline{s}$ has one, too. □

In the same way, we can use regular expressions to define the length of an element $w \in W$, we can use the twisted expressions to define the twisted length of an element $w \in \mathcal{I}_\theta$.

Definition 2.13. Let $\mathcal{I}_\theta$ be the set of twisted involutions. Then we define $\rho(w)$ as the smallest $k \in \mathbb{N}$ for that a twisted expression $w = \underline{s}_1 \dots \underline{s}_k$ exists. This is called the ***twisted length*** of w.

Lemma 2.14. *[5, Theorem 4.8]. The Bruhat ordering, restricted to the set of twisted involutions $\mathcal{I}_\theta$, is a graded poset with ρ as rank function. We denote this poset by $\mathrm{Br}(\mathcal{I}_\theta)$.*

We now establish many properties from Section 1 for twisted expressions and $\mathrm{Br}(\mathcal{I}_\theta)$. As seen in Example 2.4 there is a Coxeter system (W', S') and an Coxeter system automorphism θ with $\mathrm{Br}(W) \cong \mathrm{Br}(\mathcal{I}_\theta(W'))$. So the hope, that many properties can be transfered, is eligible.

Lemma 2.15. *[6, Lemma 3.8]. Let $w \in \mathcal{I}_\theta$ and $s \in S$. Then $\rho(w\underline{s}) = \rho(w) \pm 1$. In fact it is $\rho(w\underline{s}) = \rho(w) - 1$ iff $s \in D_R(w)$.*

Proof. Since $\mathrm{Br}(\mathcal{I}_\theta)$ is graded with rank function ρ and either $w\underline{s}$ covers w or w covers $w\underline{s}$ we have $\rho(w\underline{s}) = \rho(w) \pm 1$. Now suppose $w\underline{s} < w$. Then we have $\rho(w\underline{s}) < \rho(w)$ iff $w\underline{s} < w$ iff $l(w\underline{s}) < l(w)$ iff $l(ws) < l(w)$ iff $s \in D_R(w)$. ☐

Lemma 2.16 (Lifting property 2)**.** *[6, Lemma 3.9]. Let $v, w \in W$ with $v \leq w$. Suppose $s \in S$ with $s \in D_R(w)$. Then*

1. *$v\underline{s} \leq w$,*
2. *$s \in D_R(v) \Rightarrow v\underline{s} \leq w\underline{s}$.*

Proof. Whenever a relation comes from the ordinary Lifting Property, we denote it by $<_{LP}$ in this proof.

$v\underline{s} = vs \wedge w\underline{s} = ws$ Same situation as in Lifting Property.

$v\underline{s} = vs \wedge w\underline{s} = \theta(s)ws$ The first part $v\underline{s} = vs \leq_{LP} w$ is immediate. Suppose $s \in D_R(v)$. Then $vs \leq_{LP} ws \Rightarrow v = \theta(s)vs \leq ws \Rightarrow v\underline{s} = vs \leq \theta(s)ws = w\underline{s}$.

$v\underline{s} = \theta(s)vs \wedge w\underline{s} = ws$ **TODO**

$v\underline{s} = \theta(s)vs \wedge w\underline{s} = \theta(s)ws$ **TODO** ☐

Proposition 2.17 (Exchange property for twisted expressions)**.** *[6, Proposition 3.10]. Suppose $\underline{s}_1 \dots \underline{s}_k$ is a reduced twisted expression. If $\rho(\underline{s}_1 \dots \underline{s}_k\underline{s}) < k$ for some $s \in S$, then $\underline{s}_1 \dots \underline{s}_k\underline{s} = \underline{s}_1 \dots \hat{\underline{s}}_i \dots \underline{s}_k$ for some $i \in \{1, \dots, k\}$.*

Proof. Let $w = s_1 \dots \underline{s}_k$ and $v = s_1 \dots \underline{s}_k\underline{s}$. Assume $v\underline{s}_k \dots \underline{s}_{i+1}\underline{s}_i < v\underline{s}_k \dots \underline{s}_{i+1}$ for all i. Then we would get $\rho(v\underline{s}_k \dots s_1) < k - k = 0$. Hence there is an index i with $v\underline{s}_k \dots \underline{s}_{i+1}\underline{s}_i > v\underline{s}_k \dots \underline{s}_{i+1}$ and we choose i maximal with this property. Since $w > v$ we conclude by repetition of Lifting property 2, that $w\underline{s}_k \dots \underline{s}_{i+1} \geq v\underline{s}_k \dots \underline{s}_i$. By Lemma 2.15 we have $\rho(v) = k - 1$ and so $\rho(w\underline{s}_k \dots \underline{s}_{i+1}) = \rho(v\underline{s}_k \dots \underline{s}_i)$. Because $\mathrm{Br}(\mathcal{I}_\theta)$ is graded with rank function ρ, both twisted expresssions must represent the same element. Therefore we have $w\underline{s}_k \dots \underline{s}_{i+1} = v\underline{s}_k \dots \underline{s}_i$ yielding $v = w\underline{s}_k \dots \underline{s}_{i+1}\underline{s}_i \dots \underline{s}_k = \underline{s}_1\hat{\underline{s}}_i \dots \underline{s}_k$. ☐

Proposition 2.18 (Deletion property for twisted expressions)**.** *[6, Proposition 3.11]. Let $w = \underline{s}_1 \dots \underline{s}_k$ be a not reduced twisted expression. Then there are two indices $1 \leq i < j \leq k$ such that $w = \underline{s}_1 \dots \hat{\underline{s}}_i \dots \hat{\underline{s}}_j \dots \underline{s}_k$.*

Proof. Choose j minimal, so we have $\underline{s}_1 \ldots \underline{s}_j$ is not reduced. By Exchange property for twisted expressions there is an index i with $\underline{s}_1 \ldots \underline{s}_j = s_1 \ldots \hat{\underline{s}}_i \ldots \underline{s}_{j-1}$ yielding our hypothesis $w = \underline{s}_1 \ldots \underline{s}_j \ldots \underline{s}_k = \underline{s}_1 \ldots \hat{\underline{s}}_i \ldots \hat{\underline{s}}_j \ldots \underline{s}_k$. □

When applying the Exchange property for twisted expressions to a twisted expression, there is no hint which $\underline{s}_i$ can be omitted. Consider the following situation: Let $w \in \mathcal{I}_\theta$ and $w\underline{s}_1 \ldots \underline{s}_k = w\underline{t}_1 \ldots \underline{t}_k$ two reduced twisted expressions. Then in the twisted expression $w\underline{s}_1 \ldots \ldots \underline{s}_k\underline{t}_k$ we can omit the $\underline{t}_k$ and one other $\underline{s}$ by Exchange property for twisted expressions and get still the same element. It would be nice, when the second omitted $\underline{s}$ is one of the $\underline{s}_i$ in general, but unfortunately this proves to be false:

Example 2.19. Let $W = A_3$, $\theta = \mathrm{id}$ and $w = \underline{s}_3$. Then $w\underline{s}_2\underline{s}_1\underline{s}_2 = w\underline{s}_1\underline{s}_2\underline{s}_3$, but $w\underline{s}_1\underline{s}_2\underline{s}_3\underline{s}_2 \notin \{w\underline{s}_1\underline{s}_2, w\underline{s}_1\underline{s}_3, w\underline{s}_2\underline{s}_3\}$. Hence the omission cannot be choosen after the prefix w, but $w\underline{s}_1\underline{s}_2\underline{s}_3\underline{s}_2 = \underline{s}_1\underline{s}_2\underline{s}_3$ works, as guaranteed by Exchange property for twisted expressions.

2.2 Twisted weak ordering

In this section we introduce the twisted weak ordering $Wk(\theta)$ on the set $\mathcal{I}_\theta$ of θ-twisted involutions.

Definition 2.20. For $v, w \in \mathcal{I}_\theta$ we define $v \preceq w$ iff there are $\underline{s}_1, \ldots, \underline{s}_k \in \underline{S}$ with $w = v\underline{s}_1 \ldots \underline{s}_k$ and $\rho(v) = \rho(w) - k$. We call the poset $(\mathcal{I}_\theta, \preceq)$ ***twisted weak ordering***, denoted by $Wk(W, \theta)$. When the Coxeter group W is clear from the context, we just write $Wk(\theta)$.

Lemma 2.21. *The poset $Wk(\theta)$ is a graded poset with rank function ρ.*

Proof. Follows immediately from the definition of $\preceq$. □

Example 2.22. In Figure 2.1 we see the Hasse diagram of $Wk(A_4, \mathrm{id})$. Solid edges represent twisted congujations and dashed edges represent multiplications.

Lemma 2.23. *The poset $Wk(\theta)$ is a subposet of* $\mathrm{Br}(\mathcal{I}_\theta)$.

Proof. Both posets are defined on $\mathcal{I}_\theta$. Let $w, v \in \mathcal{I}_\theta$ be two twisted involutions. Assume $w \preceq v$ with $w\underline{s} = v$ for some $s \in S$. If $\underline{s}$ acts by multiplication on w, then $ws = v$ and since $s \in T$ (T the set of all reflections in W) and $l(w\underline{s}) = l(w) + 1$ we have $w \leq v$. If conversely $\underline{s}$ acts by twisted conjugation on w, then $v = \theta(s)ws = w(w^{-1}\theta(s)w)(e^{-1}se)$ and since $w^{-1}\theta(s)w, s \in T$ and $l(w\underline{s}) = l(\theta(s)w) + 1 = l(w) + 2$ we have again $w \leq v$. □

Proposition 2.24. *For all $w \in \mathcal{I}_\theta$ and $s \in S$ we have $w\underline{s} \prec w$ iff $s \in D_R(w)$ and $w\underline{s} \succ w$ iff $s \notin D_R(w)$ as well as $w\underline{s} < w$ iff $s \in D_R(w)$ and $w\underline{s} > w$ iff $s \notin D_R(w)$.*

Proof. We have $w\underline{s}\underline{s} = w$ and $\rho(w\underline{s}) = \rho(w) - 1$ iff $s \in D_R(w)$ and $\rho(w\underline{s}) = \rho(w) + 1$ iff $s \notin D_R(w)$ by Lemma 2.15. By Lemma 2.23 both statements are true for $\mathrm{Br}(\mathcal{I}_\theta)$, too. □

Definition 2.25. Let $v, w \in W$ with $\rho(w) - \rho(v) = n$. A sequence $v = w_0 \prec w_1 \prec \ldots \prec w_n = w$ is called a ***geodesic*** from v to w.

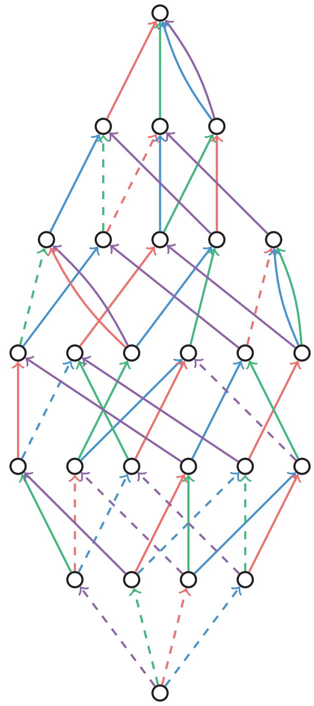

Figure 2.1: Hasse diagram of $Wk(A_4, \mathrm{id})$

Proposition 2.26. *Let $v, w \in W$ with $v \prec w$. Then all geodesics from v to w have the same count of twisted conjugated and multiplicative steps.*

Proof. Suppose we have two geodesics from v to w, where the first has n and the second m multiplicative steps. Then $l(w) + n + 2(k-n) = l(v) = l(w) + m + 2(k-m)$, hence $n = m$. □

Proposition 2.27. *Let $w \in W$ and $w\underline{s} \succ w$. Then $|\{t \in S \setminus D_R(w) : w\underline{t} = w\underline{s}\}| \in \{1, 2\}$.*

Proof. Suppose $t \in S \setminus D_R(w)$ with $w\underline{t} = w\underline{s}$. Because of the ordinary length either both $\underline{s}$ and $\underline{t}$ act by multiplication on w, or both act by twisted conjugation on w. Suppose they act by multiplication, then $ws = w\underline{s} = w\underline{t} = wt$, hence $s = t$. Conversely, assume they act by twisted conjugation. Then $\theta(s)ws = w\underline{s} = w\underline{t} = \theta(t)wt$. Because of $\theta(t)wtt = \theta(t)w = \theta(s)wst$ we have $l(\theta(s)wst) < l(\theta(s)ws)$ and so by Exchange Condition there are three possible cases

$$\theta(t)w = \theta(s)wst = \begin{cases} \theta(s)w & \Rightarrow s = t, \\ ws & \Rightarrow \theta(t) = wsw^{-1} \text{ or} \\ \theta(s)\overline{w}s & \Rightarrow w = \theta(t)\theta(s)\overline{w}s, \end{cases}$$

where $\overline{w}$ denotes a well choosen subexpression of w. The first case is trivial, the second determines t unambiguously. The third case is impossible, since by Exchange Condition and Remark 1.10 we would have a reduced expression for w beginning with $\theta(s)$ or ending with s (or both), yielding $l(\theta(s)ws) \leq l(w)$, which contradicts to $\rho(w\underline{s}) = \rho(\theta(s)ws) > \rho(w)$. Therefore, there cannot be more than two distinct $s, t \in S \setminus D_R(w)$ with $w\underline{s} = w\underline{t}$. $\square$

TODO

2.3 Residuums

Definition 2.28. Let $w \in W$ and $I \subseteq S$ be a subset of generators. Then we define

$$wC_I := \{w\underline{s}_1 \dots \underline{s}_k : k \in \mathbb{N}_0, s_i \in S\}$$

as the I***-residuum*** of w or just ***residuum***. To emphasize the size of I, say $|I| = n$, we also speak of a ***rank-n-residuum***.

Example 2.29. Let $w \in W$. Then $wC_\emptyset = \{w\}$ and $wC_S = \mathcal{I}_\theta$.

Lemma 2.30. *Let $w \in W$ and $I \subset S$. If $v \in wC_I$, then $vC_I = wC_I$.*

Proof. Suppose $v \in wC_I$. Then $v = w\underline{s}_1 \dots \underline{s}_n$ for some $s_i \in I$. Suppose $u = w\underline{t}_1 \dots \underline{t}_m \in wC_I$ is any other element in wC_I with $t_i \in I$. Then

$$u = w\underline{t}_1 \dots \underline{t}_m = (v\underline{s}_n \dots \underline{s}_1)\underline{t}_1 \dots \underline{t}_m$$

and so $u \in vC_I$. This yields $wC_I \subset vC_I$. Since $w \in vC_I$ we can swap v and w to get the other inclusion. $\square$

Corollary 2.31. *Let $v, w \in W$ and $I \subset S$. Then either $vC_I \cap wC_I = \emptyset$ or $vC_I = wC_I$.*

Proof. Immediately follows from Lemma 2.30. $\square$

Proposition 2.32. *[6, Lemma 5.6]. Let $w \in \mathcal{I}_\theta$, $I \subseteq S$ be a set of generators. Then there exists a unique element $w_0 \in wC_I$ with $w_0 \preceq w_0\underline{s}$ for all $s \in I$.*

Proof. Suppose there is no such element. Then for each $w \in wC_I$ we can find a $s \in I$ with $w' = w\underline{s} \preceq w$ and $e' \in wC_I$. By repetition of Deletion property for twisted expressions we get, that $e \in wC_I$, but e has the property, which we assumed, that no element in wC_I has. Hence there must be at least one such element. Now suppose there are two distinct elements u, v with the desired property. Note that this means, that u and w have no reduced twisted expression ending with some $\underline{s} \in I$. Let v have a reduced twisted expression $v = \underline{s}_1 \dots \underline{s}_k$. Since u and v are both in wC_I there must be a twisted v-expression for u

$$u = v\underline{s}_{k+1} \dots \underline{s}_{k+l} = \underline{s}_1 \dots \underline{s}_{k+l}$$

with $s_n \in I$ for $k + 1 \leq n \leq k + l$. This twisted expression cannot be reduced, since it ends with $\underline{s}_{k+l} \in I$. Then Deletion property for twisted expressions yields that this twisted expression contains a reduced twisted subexpression for u. It cannot end with $\underline{s}_n$ for $k + 1 \leq n \leq k + l$. Hence, it is a twisted subexpression of $\underline{s}_1 \dots \underline{s}_k = v$, too. So $u \leq v$ by Theorem 1.26. Because of symmetry we have $v \leq u$ and so $u = v$, contradicting to our assumption $u \neq v$. $\square$

Corollary 2.33. *Let $w \in \mathcal{I}_\theta$, $I \subseteq S$ be a set of generators and let $\rho_{min} := \min\{\rho(v) : v \in wC_I\}$ be the minimal twisted length within the residuum wC_I. Then there is a unique element $w_{min} \in wC_I$ with $\rho(w_{min}) = \rho_{min}$. We denote this element by* $\min(w, I)$.

Proof. The minimal rank ρ_{min} exists, since the image of ρ is in $\mathbb{N}_0$, which is well-ordered, and $wC_I \neq \emptyset$. Suppose we have an element w_{min} with $\rho(w_{min}) = \rho_{min}$. This means, that in particular all $w_{min}\underline{s}$ with $s \in I$ must be of larger twisted length, i.e. $w_{min} < w_{min}\underline{s}$ for all $s \in I$. With Proposition 2.32 this element must be unique. □

We proceed with some properties of rank-2-residuums. Our interest in these residuums stems from the fact, that their properties are needed later in Section 2.4 to construct an effective algorithm for calculating the twisted weak ordering, i.e. calculating the Hasse diagram of $Wk(\theta)$ for arbitrary Coxeter systems (W, S) and Coxeter system automorphisms θ.

Definition 2.34. Let $s, t \in S$ be two distinct generators. We define:

$$[\underline{st}]^n := \begin{cases} (\underline{st})^{\frac{n}{2}} & n \text{ even,} \\ (\underline{st})^{\frac{n-1}{2}}\underline{s} & n \text{ odd.} \end{cases}$$

This definition allows us to express rank-2-residuums differently. Suppose we have an element $w \in \mathcal{I}_\theta$ and two distinct generators $s, t \in S$. Thanks to Lemma 2.30 and Corollary 2.33 we can assume, that $w = min(w, \{s, t\})$. Then

$$wC_{\{s,t\}} = \{w\} \cup \{w[\underline{st}]^n : n \in \mathbb{N}\} \cup \{w[\underline{ts}]^n : n \in \mathbb{N}\}.$$

This encourages the following definition.

Definition 2.35. Let $w \in \mathcal{I}_\theta$ and let $s, t \in S$ be two distinct generators. Suppose $w = min(w, \{s, t\})$. Then we call $\{w[\underline{st}]^n : n \in \mathbb{N}\}$ the s-***branch*** and $\{w[\underline{ts}]^n : n \in \mathbb{N}\}$ the t-***branch*** of $wC_{\{s,t\}}$.

One question arises immediately: Are the s- and the t-branch disjoint? With the following propositions, corollaries and lemmas we will get a much better idea of the structure of rank-2-residuums and answer this question.

Proposition 2.36. *Let $w \in W$ and let $s, t \in S$ be two distinct generators. Without loss of generality suppose $w = \min(w, \{s, t\})$. If there is a $v \in wC_{\{s,t\}}$ with $v\underline{s} \prec v$ and $v\underline{t} \prec v$, then it is unique with this property in $wC_{\{s,t\}}$. Hence $wC_{\{s,t\}}$ consists of two geodesics from w to v intersecting only in these two elements. Else, the s- and t-branch are disjoint, strictly ascending in twisted length and of infinite size.*

Proof. Suppose there is a v in the s-branch with $v\underline{s} \prec v$ and $v\underline{t} \prec v$, say $v = w[\underline{st}]^n$ and n is minimal with this property. Because of the uniqueness of a minimal element from Proposition 2.32 we have $w[\underline{st}]^{m+1} \prec w[\underline{st}]^m$ for all $m \in \mathbb{N}$ with $n \leq m \leq 2n-1$. With the same argument we have $w[\underline{st}]^{2n} = w$. If no such v exists, then the s- and t-branch must be disjoint, strictly ascending in twisted length and so of infinite size. □

Proposition 2.37. *Let $w \in S$ and $s, t \in S$ be two distinct generators. If $\underline{s}$ acts by multiplications on w and $w\underline{s} \prec w$, then either $w\underline{st} \prec w\underline{s}$ or $w\underline{t} \succ w$.*

Proof. We have $\theta(s)ws = w$ and $s \in D_R(w)$, hence $w\underline{s} \leq w$. If $t \notin D_R(w)$ or $t \in D_R(w\underline{s})$, then we are done. In return suppose $t \in D_R(w)$ and $t \notin D_R(w\underline{s})$. Then we have $w\underline{t} \leq w$ and $w\underline{s} \leq w\underline{st}$. With the second part of Lifting property 2 we conclude $(w\underline{st})\underline{t} \leq w\underline{t}$. This yields

$$ws = w\underline{s} = (w\underline{st})\underline{t} \leq w\underline{t}.$$

Since $l(w) - 1 = l(w\underline{s}) \leq l(w\underline{t}) < l(w)$ we have $w\underline{t} = wt$, hence $ws \leq wt$. But $s, t \in D_R(w)$ and so $l(ws) = l(wt)$ and $ws = wt$, too. Therefore $s = t$, contradicting our assumption of two distinct generators. □

Corollary 2.38. *Let $w \in S$ and let $s, t \in S$ be two distinct generators. If w is neither $\min(w, \{s, t\})$, nor a maximal element in $wC_{\{s,t\}}$, then both $\underline{s}$ and $\underline{t}$ act by twisted conjugation on w.*

Proof. Follows immediately from Proposition 2.37. □

Lemma 2.39. *Let $s, t \in S$ be two distinct generators and $w \in S$ with $w = min(w, \{s, t\})$. Suppose $v \in wC_{\{s,t\}}$ with $v\underline{s} \prec v$ and $v\underline{t} \prec v$. Then the twisted conjugations and mulitplications are distributed axisymmetrically or pointsymmetrically like in Figure 2.2.*

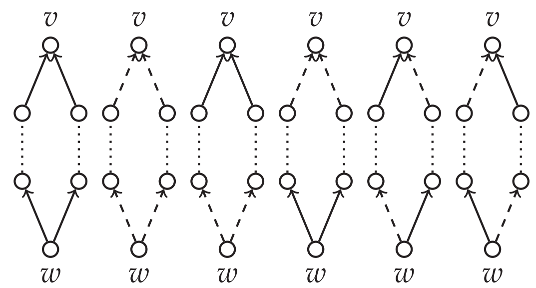

Figure 2.2: Possible distributions of twisted conjugations and multiplications in finite rank-2-residuums

Proof. If u covers w, then there are only two edges and the assumption holds. So suppose $wC_{\{s,t\}}$ contains at least four edges. Due to Corollary 2.38 the actions by multiplication can only occure next to w and v. Hence there are $2^4 = 16$ configurations possible. Proposition 2.26 wipes out ten out of the 16 configurations. The remaining are those from Figure 2.2. □

Example 2.40. In Figure 2.3 we see two Hasse diagrams of $Wk(A_4, \mathrm{id})$. The left one only contains edges with labels s_1, s_2, the middle one only edges with labels s_1, s_3 and the right one only edges with labels s_1, s_4.

Lemma 2.41. *Let $w \in S$, $s, t \in S$ be two distinct generators and $m = \mathrm{ord}(st) < \infty$. Then $|wC_{\{s,t\}}| \leq 2m$.*

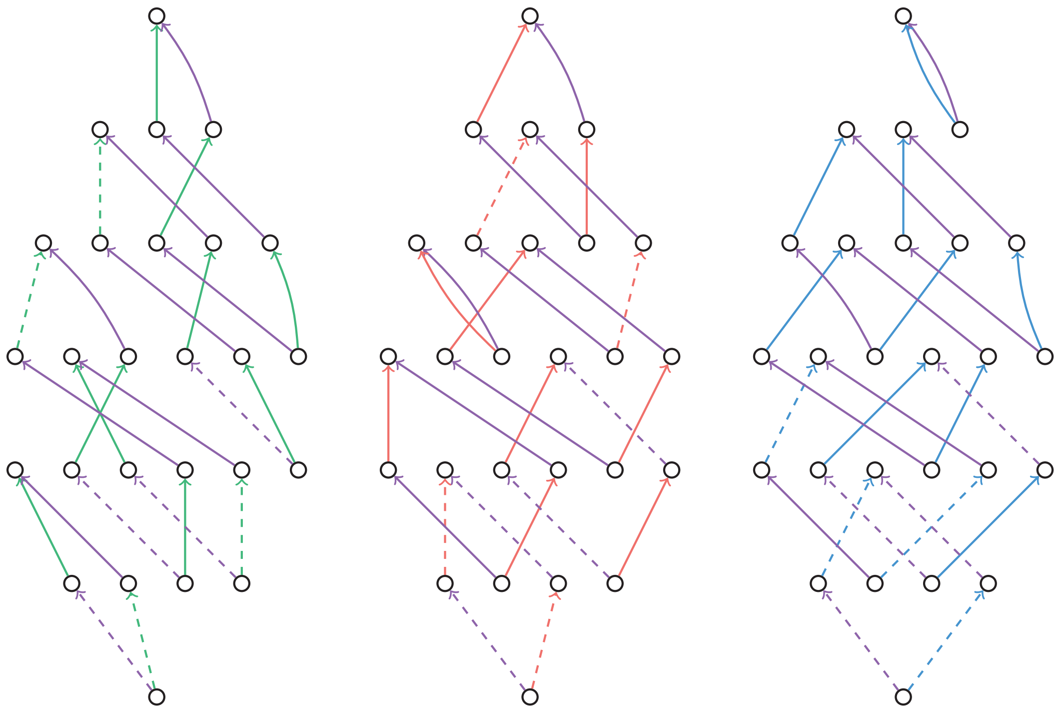

Figure 2.3: Hasse diagrams of $Wk(A_4, \mathrm{id})$ after removing s_3, s_4 edges in the left, s_2, s_4 edges in the middle and s_2, s_3 edges in the right diagram

Proof. Let w be the Wk-minimal element and v be the Wk-maximal element in our residuum. Due to Lemma 2.39 there are six different cases we have to consider. We handle the cases in Figure 2.2 from left to right:

First: We have $w(\underline{st})^m = (ts)^m w(st)^m = w$.

Second: Due to $\theta(s)w = ws$ and $\theta(t)w = wt$ we have

$$w(\underline{st})^{m/2+1} = \theta(\hat{t}(st)^{m/2-1}\hat{s})w(st)^{m/2+1} = w(st)^m = w.$$

(**TODO** Show that this situation only occurs for even m)

Third: Again we are in a case, where $\theta(s)w = ws$ and $\theta(t)w = wt$ hold. Hence we have

$$w(\underline{st})^{(m+1)/2} = \theta(\hat{t}(st)^{(m-1)/2}\hat{s})w(st)^{(m+1)/2} = w(st)^m = w.$$

(**TODO** Show that this situation only occurs for odd m)

Fourth: Analogue to the previous case, if we start from u instead of w.

Fifth: Suppose m is even. Then we have

$$w(\underline{st})^m = \theta(\underbrace{ts\cdots st}_{m-1}\,\hat{s}\,\underbrace{ts\cdots st}_{m-1}\,\hat{s})w(st)^m = \theta(\underbrace{ts\cdots s}_{m-2}\,\underbrace{s\cdots st}_{m-2})w = \ldots = w.$$

If m is odd, then we have the completely analogue situation

$$w(\underline{st})^m = \theta(\underbrace{ts\cdots ts}_{m-1}\hat{t}\,\underbrace{st\cdots st}_{m-1}\hat{s})w(st)^m = \theta(\underbrace{ts\cdots t}_{m-2}\underbrace{t\cdots st}_{m-2})w = \ldots = w.$$

Sixth: Analogue to the previous case due to symmetry.

So in all six cases we have $w(\underline{st})^k = w$ for a $k \leq \operatorname{ord}(st)$ and hence the residuum can have at most $2 \cdot \operatorname{ord}(st)$ many distinct elements. □

Proposition 2.42. *Let $w \in S$ and $s, t \in S$ be two distinct generators with $\operatorname{ord}(st) < \infty$. Suppose $k \in \mathbb{N}$ to be the smallest number with $w = w(\underline{st})^k$. Then for any $n \in \mathbb{N}$ with $w = w(\underline{st})^n$ we have $k \mid n$.*

Proof. Let $n = qk + r$ for $q \in \mathbb{N}_0$ and $r \in \{0, \ldots, k-1\}$. Then

$$w(\underline{st})^n = w(\underline{st})^{qk+r} = w((\underline{st})^k)^q(\underline{st})^r = w(\underline{st})^r.$$

For $r > 0$ we would have a contradiction to the minimality of k, hence $r = 0$, $q > 0$ and therefore $k \mid n$. □

Corollary 2.43. *Let $w \in S$ and $s, t \in S$ be two distinct generators with $w\underline{s} \neq w\underline{t}$. Suppose $w = w(\underline{st})^m = w(\underline{st})^n$. Then $\gcd(m, n) > 1$.*

Proof. Let k be the same as in Proposition 2.42. Since $w\underline{s} \neq w\underline{t}$ we have $k > 1$. Both, $k \mid n$ and $k \mid m$, hence $\gcd(m, n) \geq k > 1$. □

This constraints the possible size of rank-2-residuums.

2.4 Twisted weak ordering algorithms

Now we address the problem of calculating $Wk(\theta)$ for an arbitrary Coxeter group W, given in form of a set of generating symbols $S = \{s_1, \ldots s_n\}$ and the relations in form of $m_{ij} = \operatorname{ord}(s_i s_j)$. From this input we want to calculate the Hasse diagram, i.e. the vertex set $\mathcal{I}_\theta$ and the edges labeled with $\underline{s}$. Thanks to Lemma 2.12 the vertex set can be obtained by walking the e-orbit of the action from Definition 2.5. The only element of twisted length 0 is e. Suppose we have already calculated the Hasse diagram until the twisted length k, i.e. we know all vertices $w \in \mathcal{I}_\theta$ with $\rho(w) \leq k$ and all edges connecting two vertices u, v with $\rho(u) + 1 = \rho(v) \leq k$. Let $\rho_k := \{w \in \mathcal{I}_\theta : \rho(w) = k\}$. Then all vertices in ρ_{k+1} are of the form $w\underline{s}$ for some $w \in \rho_k, s \in S$. For each $(w, s) \in \rho_k \times S$, we calculate $w\underline{s}$. If $\rho(w\underline{s}) = k+1$ then $w \prec w\underline{s}$. To avoid having to check the twisted length we use Lemma 2.15. We already know the set $S_w \subseteq S$ of all generators yielding an edge into w. Due to the lemma we have $\rho(w\underline{s}) = k - 1$ for all $s \in S_w$ and $\rho(w\underline{s}) = k + 1$ for all $s \in S \setminus S_w$. Hence we only calculate $w\underline{s}$ for $s \in S \setminus S_w$ and know $w \prec w\underline{s}$ without checking the twisted length explicitly. The last problem to solve is the possibility of two different $(w, s), (v, t) \in \rho_k \times S$ with $w\underline{s} = v\underline{t}$. To deal with this, we have to compare a potential new twisted involution $w\underline{s}$ with each element of twisted length $k + 1$, already calculated. The concrete problem of comparing two elements in a free presented group, called ***wordproblem for groups***, will not be addressed here. We suppose, that whatever computer system is used to implement our algorithm, supplies a suitable way to do that. The only thing to note is, that solving the wordproblem is not a cheap operation. Reducing the count of element comparisions is a major demand to any algorithm, calculating $Wk(\theta)$.

The steps discussed have been compiled in to an algorithm by [1, Algorithm 2.4] and [4, Algorithm 3.1.1]. We take this as our starting point. Since the runtime is far from being optimal, we use the structural properties of rank-2-residuums from Section 2.3 to improve the algorithm. As we will show, these optimizations yield an algorithm with an asymptotical perfect runtime behavior. TWOA1 shows this algorithm.

Algorithm 2.44 (TWOA1).

procedure TwistedWeakOrderingAlgorithm1($(W,S),k_{max}$)
$V \leftarrow \{(e,0)\}$
$E \leftarrow \{\}$
for $k \leftarrow 0$ **to** k_{max} **do**
for all $(w,k_w) \in V$ **with** $k_w = k$ **do**
for all $s \in S$ **with** $\nexists(\cdot,w,s) \in E$ **do** ▷ Only for $s \notin D_R(w)$
$y \leftarrow ws$
$z \leftarrow \theta(s)y$
if $z = w$ **then**
$x \leftarrow y$
$t \leftarrow s$
else
$x \leftarrow z$
$t \leftarrow \underline{s}$
end if
$isNew \leftarrow$ **true**
for all $(w',k_{w'}) \in V$ **with** $k_{w'} = k+1$ **do** ▷ Check if x already known
if $x = w'$ **then**
$isNew \leftarrow$ **false**
end if
end for
if $isNew =$ **true then**
$V \leftarrow V \cup \{(x,k+1)\}$
end if
$E \leftarrow E \cup \{(w,x,t)\}$
end for
end for
$k \leftarrow k+1$
end for
return (V,E) ▷ The poset graph

31: **end procedure**

Note, that if W is finite, k_{max} does not have to be evaluated explicitly. When k reaches the maximal twisted length in $Wk(\theta)$, then the only vertex of twisted length k is the unique element $w_0 \in W$ of maximal ordinary length. Since $s \in D_R(w_0)$ for all $s \in S$, there is no $s' \in S$ remaining to calculate $w_0\underline{s}'$ for. This condition can be checked to terminate the algorithm without knowing k_{max} before. When W is infinite, there is no maximal element and $\mathcal{I}_\theta$ is infinite, too. In this case k_{max} is used to terminate after having calculated a finite part of $Wk(\theta)$.

Lemma 2.45. *TWOA1 is a deterministic algorithm.*

Proof. The outer loop (line 4) is strictly ascending in $k \in \{0, \ldots, k_{max}\}$ and so finite. The innermost loop (line 6) is finite since S is finite. The inner loop (line 5) is finite, since V starts as finite set and in each step there are added at most $|V| \cdot |S|$ many new vertices. Therefore the algorithm terminates. The soundness is due to the arguments at the beginning of Section 2.4. $\square$

Lemma 2.46. *Let $k \in \mathbb{N}$, $n = |\{w \in \mathcal{I}_\theta : \rho(w) \leq k\}|$ and for $0 \leq i \leq k$ let $\rho_i = |\{w \in \mathcal{I}_\theta : \rho(w) = i\}|$. Then $TWOA1 \in \mathcal{O}(n^2/k)$.*

Proof. Our algorithm has to do at least $\rho_i(\rho_i - 1)/2$ many element comparisons (line 17) for each $0 \leq i \leq k$. Set $m = \lfloor \frac{n}{k} \rfloor$. In the most optimistic case we have $\rho_i \geq m$ for all i. In practice the situation will be worse, since some ρ_i will be smaller than m (for example $\rho_0 = 1$) and so some ρ_i will be much larger than m. This optimistic case yields at least $m(m-1)/2 \cdot k$ many element comparisons. Hence regarding the most delimiting operation, the element comparison, our algorithm is in $\Omega(m^2 k) = \Omega(n^2/k)$. The element comparison at line 9 done at most $n \cdot |S|$. Other operations, like for example insertion into or searching in sets can be considered super linear, if for example sets are ordered immediately at insertion and then searching is done with binary search. So the algorithm is in $\mathcal{O}(n^2/k)$. $\square$

Corollary 2.47. *Let $k \in \mathbb{N}$ and $n = |\{w \in \mathcal{I}_\theta : \rho(w) \leq k\}|$. Then any algorithm calculating $Wk(\theta)$ is in $\Omega(n)$.*

Proof. Any algorithm must at least return $\{w \in \mathcal{I}_\theta : \rho(w) \leq k\}$ and this set is of size n. $\square$

Our goal is to improve TWOA1 so that we get an algorithm in $\mathcal{O}(n)$, i.e. an asymptotical perfect algorithm for calculating $Wk(\theta)$. As already seen the element comparison of a potential new element with all already known elements of same twisted length (line 17) is the bottleneck. Here the rank-2-residuums become key. Suppose we have a $w \in \mathcal{I}_\theta$ with $\rho(w) = k$ and $s \in S$. In TWOA1 we would now check, if $w\underline{s}$ is a new vertex, or if we already calculated it by comparing it with all already known vertices of twisted length $k+1$. Assume we have already calculated it. This means there is another twisted involution v

with $\rho(v) = k$ and another generator $t \in S$ with $v\underline{t} = w\underline{s}$. With Proposition 2.36 $w\underline{s}$ is the unique element of maximal twisted length in the rank-2-residuum $wC_{\{s,t\}}$. This yields a necessary condition for $w\underline{s}$ to be equal to a already known vertex, allowing us to replace the ineffective search all method in TWOA1, line 17.

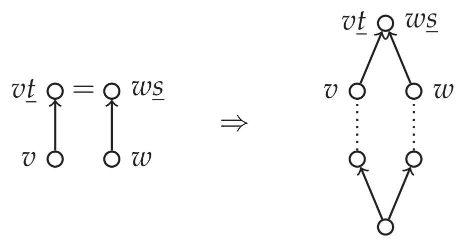

Figure 2.4: Optimization of TWOA1

Lemma 2.48. *Let $k \in \mathbb{N}$ and suppose we are in the sitation described at the beginning of Section 2.4. Let $\rho_i := \{w \in \mathcal{I}_\theta : \rho(w) = i\}$ and ρ'_{k+1} the set of the already calculated vertices with twisted length $k+1$. If $w\underline{s} \in \rho'_{k+1}$ for some $w \in \rho_k, s \in S$, say $w\underline{s} = v\underline{t}$ with $v \in \rho_k$ and $t \in S \setminus \{s\}$, then $w\underline{s} = w[\underline{ts}]^n$ for some $n \in \mathbb{N}$ with $w[\underline{ts}]^j \in \rho_0 \cup \ldots \cup \rho_k \cup \rho'_{k+1}$ for $1 \leq j \leq n$.*

Proof. The equality $w\underline{s} = w[\underline{ts}]^n$ for some $n \in \mathbb{N}$ is due to Proposition 2.36. All vertices in this rank-2-residuum except $v\underline{t}$ have a twisted length of k or lower. For $v\underline{t}$ we supposed it is already known, hence $v\underline{t} \in \rho'_{k+1}$. Therefore all vertices $w[\underline{ts}]^j$, $1 \leq j \leq n$ are in $\rho_0 \cup \ldots \cup \rho_k \cup \rho'_{k+1}$. □

This can be checked effectively. Both, w and s are fixed. Start with $M = \emptyset$. For all already known edges from or to w being labeled with $\underline{t} \in \underline{S} \setminus \{\underline{s}\}$ we do the following: Walk $w[\underline{ts}]^i$ for $i = 0, 1, \ldots$ until $\rho(w[\underline{ts}]^i) = k+1$. Note that walking in this case really means walking the graph. All involved vertices and edges have already been calculated. So there is no need for more calculations in W to find $w[\underline{ts}]^i$. By Proposition 2.36 such a path must exist (in a completely calculated graph). But we could be in the case, where the last step from $w[\underline{ts}]^{i-1}$ to $w[\underline{ts}]^i$ has not been calculated yet. If it is already calculated, then add this element to M by setting $M = M \cup \{w[\underline{ts}]^i\}$. If not, do not add it to M.

Now M contains all already known elements of twisted length $k+1$, satisfying the necessary condition from Lemma 2.48. Furthermore $|M| < |S|$. So for each pair (w, s) we have to do at most $|S| - 1$ many element comparisons the determine, if $w\underline{s}$ is new or already known, no matter how many elements of twisted length $k+1$ are already known. We can get even more information from the rank-2-residuums:

Lemma 2.49. *Let $w \in \mathcal{I}_\theta$ with $\rho(w) = k$, s, t be two distinct generators and $s \notin D_R(w)$. Suppose $n \in \mathbb{N}$ to be the smallest number for that $\rho(w[\underline{ts}]^{2n-1}) = k+1$ holds. Then:*

1. *If $n = \operatorname{ord}(st)$, then $w[\underline{ts}]^{2n-1} = w\underline{s}$.*
2. *If $n \geq 2$ and $l(w[\underline{ts}]^{2n-1}) - l(w[\underline{ts}]^{2n-2}) = 1$, then $w[\underline{ts}]^{2n-1} = w\underline{s}$.*

Proof. 1. Follows immediately from Lemma 2.41.

2. Because of the length difference the step from $w[\underline{ts}]^{2n-2}$ to $w[\underline{ts}]^{2n-1}$ is a multiplication, not a twisted conjugation, and because of $n \geq 1$ this step cannot be next to the smallest element in $wC_{\{s,t\}}$. Hence $w[\underline{ts}]^{2n-1} = w\underline{s}$ by Corollary 2.38. $\square$

Lemma 2.50. *Let $w \in \mathcal{I}_\theta$ with $\rho(w) = k$, s, t be two distinct generators and $s \notin D_R(w)$. Suppose $w[\underline{ts}]^{2n-1} = w\underline{s}$ and suppose n to be the smallest number with this property. Then $w[\underline{ts}]^{n-1}$ is the minimal element $\min(w, \{s,t\})$ and $w[\underline{ts}]^{2n-1}$ is the maximal element. Define*

$$\begin{aligned} a &= l(w\underline{s}) - l(w), \\ b &= l(w[\underline{ts}]^{n-1}) - l(w[\underline{ts}]^{n-2}), \\ c &= l(w[\underline{ts}]^{n}) - l(w[\underline{ts}]^{n-1}) \text{ and} \\ d &= l(w[\underline{ts}]^{2n-1}) - l(w[\underline{ts}]^{2n-2}). \end{aligned}$$

Note that $a, b, c, d \in \{1, 2\}$ contain the information, if edges next to the minimal and the maximal element of $wC_{\{s,t\}}$ are twisted conjugations or multiplications. Then each can be deduced from the three remaining ones with the equation $a + b = c + d$.

Proof. The minimality of $w[\underline{ts}]^{n-1}$ and the maximality of $w[\underline{ts}]^{2n-1}$ is due to Proposition 2.36. The soundness of the equation follows from the symmetric distribution of twisted conjugations and mutipliations from Lemma 2.39. $\square$

Lemma 2.51. *Let $w \in S$ and $s, t \in S$ be two distinct generators with $w\underline{s} \neq w\underline{t}$. Then the following table shows all possible $n \in \mathbb{N}, n \leq \operatorname{ord}(st)$ with $w(\underline{st})^n = w$.*

ord(st)	*2*	*3*	*4*	*5*	*6*	*7*	*8*	
n	*2*	*3*	*2,4*	*5*	*2,3,4,6*	*7*	*2,4,6,8*	*First, Fifth, Sixst*
n	*2*	–	*3*	–	*2,4*	–	*5*	*Second*
n	–	*2*	–	*3*	–	*2,4*	–	*Third, Fourth*

Proof. In each case we get a m with $w = (\underline{st})^m$ from the proof of Lemma 2.41. By Corollary 2.43 any n with this property has a non trivial divisor in common with m. $\square$

Algorithm 2.52 (TWOA2).

procedure TwistedWeakOrderingAlgorithm1($(W, S), k_{max}$)
  $V \leftarrow \{(e, 0)\}$
  $E \leftarrow \{\}$
  for $k \leftarrow 0$ **to** k_{max} **do**
    for all $(w, k_w) \in V$ **with** $k_w = k$ **do**
      for all $s \in S$ **with** $\nexists(\cdot, w, s) \in E$ **do** ▷ Only for $s \notin D_R(w)$ ▷ **TODO**
      end for
    end for
    $k \leftarrow k + 1$
  end for

    return (V, E) ▷ The poset graph
end procedure

			Timings		Element comparisons	
W	$\lvert Wk(W, \mathrm{id})\rvert$	$\rho(w_0)$	TWOA1	TWOA2	TWOA1	TWOA2
A_9	9496	25	00:02.180	00:01.372	13,531,414	42,156
A_{10}	35696	30	00:31.442	00:06.276	185,791,174	173,356
A_{11}	140152	36	11:04.241	00:29.830	2,778,111,763	737,313
E_6	892	20	00:03.044	00:00.268	85,857	2,347
E_7	10208	35	06:11.728	00:02.840	7,785,186	29,687
E_8	199952	64	–	11:03.278	–	682,227

Table 2.1: Benchmark

3 Main Thesis

Question 3.1. *Let (W, S) be a Coxeter system, $\theta : W \to W$ an automorphism of W with $\theta^2 = \mathrm{id}$ and $\theta(S) = S$, and $K \subset S$ a subset of S generating a finite subgroup of W with $\theta(K) = K$. Denote the largest element in $\langle K \rangle \leq W$ by w_K. Futhermore let $S_1, S_2, S_3 \subset S$ be three sets of generators. Define $S_{ij} = S_i \cap S_j$ and $T = S_1 \cap S_2 \cap S_3$. For which Coxeter groups W does the implication*

$$\forall\, 1 \leq i < j \leq 3 : w \in w_K C_{S_{ij}} \quad \Rightarrow \quad w \in w_K C_T \tag{3.1.1}$$

hold for any possible K, θ, S_1, S_2, S_3 and w?

The reader might wonder, why we handle with intersections of sets of generators and not just with arbitrary sets of generators. The reason for that is also the main reason, why $Wk(\theta)$ is less accessible than $\mathrm{Br}(W)$: In $Wk(\theta)$ there is the possibilty for $w\underline{s} = w\underline{t}$ for two distinct generators $s, t \in S$. Within the Hasse diagram this situation appears in form of double edges between two vertices. For example, let $W = A_3$ and θ be the Coxeter system automorphism swapping s_1 with s_3. Then we have $e\underline{s_1} = s_3 s_1 = s_1 s_3 = e\underline{s_3}$. Double edges can also occur for $\theta = \mathrm{id}$, but in this situation they cannot appear next to the neutral element e, since $\theta(s)es = e$ for all $s \in S$, hence $e\underline{s} = s \neq t = e\underline{t}$ for all $s, t \in S$ with $s \neq t$. Therefore, if we had written 3.1.1 with arbitrary sets S_{12}, S_{23}, S_{31}, then it would be false immediately for any Coxeter system automorphism, that swaps two commutating generators, as seen in Example 3.3.

The following corollary shows us, what distinguish our special configuration of sets of generators from the arbitrary configuration.

Corollary 3.2. *Let M be a set and $S_{12}, S_{23}, S_{31} \subseteq M$ three subsets. Then there are three sets $S_1, S_2, S_3 \subseteq M$ with $S_{ij} = S_i \cap S_j$ iff no element $x \in M$ is precisely in two of the sets S_{ij}.*

Proof. Let S_{12}, S_{23}, S_{31} be the pairwise intersection of three sets S_1, S_2, S_3. If an element $x \in M$ is in none or in one of the sets S_i, then it is in none of the sets S_{ij}. If it is in two of the sets S_i, say $x \in S_1, S_2$, then $x \in S_{12}$, but x is not in one of the other two S_{ij}. If x is in all three S_i, then it is in all three S_{ij}, too. Hence there is no $x \in M$, that is in precisely two of the sets S_{ij}. Conversely, suppose S_{12}, S_{23}, S_{31} to be arbitrary with the constraint, that there is no element $x \in M$ in precisely two of them. Then we can construct three sets S_1, S_2, S_3, whose pairwise intersections coincides with the sets S_{ij} by $x \in S_i \wedge x \in S_j$ iff $x \in S_{ij}$. With this construction and the previous considerations, it is clear that these S_i have the S_{ij} as pairwise intersection. Note that this construction is not unique in general, since when there is a $x \in M$, that is in none of the sets S_{ij}, then we could add it to S_1, S_2 or S_3 or just omit it without changing there pairwise intersection. □

3.1 Results in less or more specific cases

In this section we investigate some results and examples, in situations that are less or more specific than the situation from Question 3.1.

Example 3.3. Let $W = A_3$ and θ be the Coxeter system autmorphism swapping s_1 and s_3. Then $e\underline{s}_1 = e\underline{s}_3$ but $e\underline{s}_1 \notin eC_{\{s_1\}\cap\{s_1\}\cap\{s_2\}} = eC_\emptyset = \{e\}$.

Such a trivial counterexample like in Example 3.3 can not occur in the situation from Question 3.1.

Proposition 3.4. *Consider the situation from Question 3.1. Let $s \in S_{12}$ and $s \in S_{23}$. Then $s \in T$.*

Proof. We have $s \in S_1, S_2, S_3$, hence $s \in T$. □

Corollary 3.5. *Consider the situation from Question 3.1. Let $w, v \in \mathcal{I}_\theta$ with $\rho(v) - \rho(w) = 1$ and let $S_1, S_2, S_3 \subset S$ be three arbitrary sets of generators with $v \in wC_{S_{ij}}$ for $1 \leq i < j \leq 3$. Then we have $v \in wC_T$.*

Proof. Each set of S_1, S_2, S_3 must at least contain s or t, hence s or t is in at least two sets, say s is in two sets. By Proposition 3.4 s is in all three sets, hence $s \in T$ and so $v \in wC_T$. □

A hypothesis, that is much stronger than Question 3.1, reads $wC_I \cap wC_J = wC_{I\cap J}$. If this would be true, Question 3.1 could be concluded immediately. Unfortunately it proves to be false. Again, double-edges yield a simple counterexample.

Example 3.6. Let $w \in \mathcal{I}_\theta$ and s, t two distinct generators with $w\underline{s} = w\underline{t} = v$. Then $wC_{\{s\}} \cap wC_{\{t\}} = \{w, v\} \neq \{w\} = wC_\emptyset = wC_{\{s\}\cap\{t\}}$.

Proposition 3.7. *Consider the situation from Question 3.1. Suppose one set of S_1, S_2, S_3 is contained in another. Then*

$$\forall\, 1 \leq i < j \leq 3 : v \in wC_{S_{ij}} \quad \Rightarrow \quad v \in wC_T.$$

Proof. Let $S_1 \subset S_2$. Then we have $S_{12} = S_1$. By this we get the identity

$$T = S_1 \cap S_2 \cap S_3 = S_{12} \cap S_3 = S_1 \cap S_3.$$

Hence $v \in wC_T = wC_{S_{31}}$. □

A Source codes

```
Read("twistedinvolutionweakordering.gap");

CalculateTwistedWeakOrderings := function()
    local tasks, task, theta, W, matrix;

    tasks := [
        rec(system := CoxeterGroup_An(1), thetas := [ "id" ]),
        rec(system := CoxeterGroup_An(2), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(3), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(4), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(5), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(6), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(7), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(8), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(9), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_An(10), thetas := [ "id", "-id" ]),
#        rec(system := CoxeterGroup_An(11), thetas := [ "id", "-id" ]),
#        rec(system := CoxeterGroup_An(10), thetas := [ "id" ]),
#        rec(system := CoxeterGroup_An(12), thetas := [ "id", "-id" ]),
#        rec(system := CoxeterGroup_An(13), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_BCn(2), thetas := [ "id", "-id" ]),
        rec(system := CoxeterGroup_BCn(3), thetas := [ "id" ]),
#        rec(system := CoxeterGroup_BCn(4), thetas := [ "id" ]),
#        rec(system := CoxeterGroup_BCn(5), thetas := [ "id" ]),
#        rec(system := CoxeterGroup_BCn(6), thetas := [ "id" ]),
#        rec(system := CoxeterGroup_BCn(7), thetas := [ "id" ]),
        rec(system := CoxeterGroup_Dn(4), thetas := [ "id" ]),
        rec(system := CoxeterGroup_Dn(5), thetas := [ "id" ]),
        rec(system := CoxeterGroup_Dn(6), thetas := [ "id" ]),
        rec(system := CoxeterGroup_E6(), thetas := [ "id", [6,5,3,4,2,1] ]),
        rec(system := CoxeterGroup_E7(), thetas := [ "id" ]),
#        rec(system := CoxeterGroup_E8(), thetas := [ "id" ]),
        rec(system := CoxeterGroup_F4(), thetas := [ "id" ]),
        rec(system := CoxeterGroup_H3(), thetas := [ "id" ]),
        rec(system := CoxeterGroup_H4(), thetas := [ "id" ]),
#        rec(system := CoxeterGroup_I2m(3), thetas := [ "id", "-id" ]),
#        rec(system := CoxeterGroup_I2m(4), thetas := [ "id", "-id" ]),
#        rec(system := CoxeterGroup_I2m(5), thetas := [ "id", "-id" ]),
#        rec(system := CoxeterGroup_I2m(6), thetas := [ "id", "-id" ]),
    ];

    for task in tasks do
        W := task.system.group;
        matrix := task.system.matrix;

        for theta in List(task.thetas, t -> GroupAutomorphismByPermutation(W, t)) do
            Print(Name(W), " ", Name(theta), "\n");
            TwistedInvolutionWeakOrdering(StringToFilename(Concatenation(Name(W), "-",
                Name(theta))), W, matrix, theta);
        od;
    od;
end;

Benchmark := function ()
    local tasks, task, theta, W, matrix, benchmarks, b, result, startTime, endTime;

    tasks := [
        rec(system := CoxeterGroup_An(9), thetas := [ "id" ]),
```

```
        #rec(system := CoxeterGroup_An(10), thetas := [ "id" ]),
        #rec(system := CoxeterGroup_An(11), thetas := [ "id" ]),
        rec(system := CoxeterGroup_E6(), thetas := [ "id" ]),
        #rec(system := CoxeterGroup_E7(), thetas := [ "id" ]),
    ];

    benchmarks := [];

    Print("Benchmark algo 1\n");

    for task in tasks do
        W := task.system.group;
        matrix := task.system.matrix;

        for theta in List(task.thetas, t -> GroupAutomorphismByPermutation(W, t)) do
            Print(Name(W), " ", Name(theta), "\n");

            startTime := Runtime();
            result := TwistedInvolutionWeakOrdering1(fail, W, matrix, theta);
            endTime := Runtime();

            Add(benchmarks, rec(name := Name(W), algo := "TWOA1", time := StringTime(endTime - startTime), result := result));
        od;
    od;

    Print("Benchmark algo 2\n");

    for task in tasks do
        W := task.system.group;
        matrix := task.system.matrix;

        for theta in List(task.thetas, t -> GroupAutomorphismByPermutation(W, t)) do
            Print(Name(W), " ", Name(theta), "\n");

            startTime := Runtime();
            result := TwistedInvolutionWeakOrdering(fail, W, matrix, theta);
            endTime := Runtime();

            Add(benchmarks, rec(name := Name(W), algo := "TWOA2", time := StringTime(endTime - startTime), result := result));
        od;
    od;

    for b in benchmarks do
        Display(b);
    od;
end;

TestCondition := function ()
    local tasks, task, S, K, _T, T, K12, K23, K31, wK, parts, part, graph,
        resS12, resS23, resS31, resT, resDiff, i, j;

    tasks := [
#        "H_3-id",
#        "H_4-id",
#        "F_4-id",
#        "D__4_-id",
#        "D__5_-id",
#        "D__6_-id",
#        "E_6-id",
```

```
#        "E_7-id",
        "E_8-id",
#        "A__4_-id",
 #       "A__5_-id",
 #       "A__6_-id",
 #       "A__7_-id",
 #       "BC__4_-id",
 #       "BC__5_-id",
 #       "BC__6_-id",
 #       "A__8_-id",
 #       "A__9_-id",
 #       "A__10_-id",
 #       "BC__7_-id",
    ];

    for task in tasks do
        graph := TwistedInvolutionWeakOrderingPersistReadResults(task);
        S := [1..graph.data.rank];
        Print(graph.data.name, " ", graph.data.automorphism, "\n");
        i := 0;

        for K in IteratorOfCombinations(S) do
            j := 0;
            i := i + 1;

            if Length(K) <= 2 or Length(K) = Length(S) then continue; fi;
            parts := PartitionsSet(K, 3);
            wK := TwistedInvolutionWeakOrderungLongestWord(graph.vertices[1], K);

            for _T in IteratorOfCombinations(K) do
                j := j + 1;
                Print(i, " ", j, "                         \r");

                T := Union(_T, Difference(S, K));

                for part in parts do
                    K12 := part[1];
                    K23 := part[2];
                    K31 := part[3];
                    #Print("K=", K, " T=", T, " K12=", K12, " K23=", K23, " K31=", K31, "\n");

                    resS12 := TwistedInvolutionWeakOrderungResiduum(wK, Union(K12, T));
                    resS23 := TwistedInvolutionWeakOrderungResiduum(wK, Union(K23, T));
                    resS31 := TwistedInvolutionWeakOrderungResiduum(wK, Union(K31, T));
                    resT := TwistedInvolutionWeakOrderungResiduum(wK, T);

                    resDiff := Difference(Intersection(resS12, resS23, resS31), resT);

                    if Length(resDiff) > 0 then
                        Print("*** FOUND COUNTEREXAMPLE ***\n",
                            "W = ", graph.name, "\n",
                            "theta = ", graph.automorphis, "\n",
                            "S = ", S, "\n",
                            "K = ", K, "\n",
                            "wK = ", wK, "\n",
                            "T = ", T, "\n",
                            "K12 = ", K12, "\n",
                            "K23 = ", K23, "\n",
                            "K31 = ", K31, "\n",
                            "w = ", resDiff, "\n\n");
```

```
                    fi;
                od;
            od;
        od;
    od;
end;
```

```
LoadPackage("io");

Read("misc.gap");
Read("coxeter.gap");
Read("twistedinvolutionweakordering-persist.gap");

TwistedInvolutionDeduceNodeAndEdgeFromGraph := function(matrix, startNode, startLabel,
    labels)
    local rank, comb, trace, possibleEqualNodes, e, k, n;

    rank := -1/2 + Sqrt(1/4 + 2*Length(matrix)) + 1;
    possibleEqualNodes := [];

    for comb in List(Filtered(labels, label -> label <> startLabel), label -> rec(
        startNode := startNode, s := [startLabel, label], m := CoxeterMatrixEntry(
        matrix, rank, startLabel, label))) do
        trace := [];
        k := 1;
        n := comb.startNode;

        Add(trace, rec(node := n, edge := rec(label := comb.s[1], type := -1)));

        while k < comb.m do
            e := FindElement(n.inEdges, e -> e.label = comb.s[k mod 2 + 1]);
            if e = fail then break; fi;
            n := e.source;

            Add(trace, rec(node := n, edge := e));
            k := k + 1;
        od;

        while k > 0 do
            e := FindElement(n.outEdges, e -> e.label = comb.s[k mod 2 + 1]);
            if e = fail then break; fi;
            n := e.target;

            Add(trace, rec(node := n, edge := e));
            k := k - 1;
        od;

        if k <> 0 then continue; fi;

        if Length(trace) = 2*comb.m then
            return rec(result := 0, node := trace[Length(trace)].node, type := trace[
                comb.m + 1].edge.type, trace := trace);
        fi;

        if Length(trace) >= 4 then
            if trace[Length(trace) / 2 + 1].edge.type <> trace[Length(trace) / 2].edge.
                type then
                # cannot be equal
            else
                if trace[Length(trace)].edge.type = 0 then
                    return rec(result := 0, node := trace[Length(trace)].node, type :=
```

```
                        0, trace := trace);
                else
                    Add(possibleEqualNodes, trace[Length(trace)].node);
                fi;
            fi;
        else
            Add(possibleEqualNodes, trace[Length(trace)].node);
        fi;
    od;

    return rec(result := -1, possibleEqualNodes := possibleEqualNodes);
end;

# Calculates the poset Wk(theta).
TwistedInvolutionWeakOrdering := function (filename, W, matrix, theta)
    local persistInfo, maxOrder, nodes, edges, absNodeIndex, absEdgeIndex, prevNode,
        currNode, newEdge,
        label, type, deduction, startTime, endTime, S, k, i, s, x, y, n;

    persistInfo := TwistedInvolutionWeakOrderingPersistResultsInit(filename);

    S := GeneratorsOfGroup(W);
    maxOrder := Minimum([Maximum(Concatenation(matrix, [1])), 5]);
    nodes := [ [], [ rec(element := One(W), twistedLength := 0, inEdges := [], outEdges
         := [], absIndex := 1) ] ];
    edges := [ [], [] ];
    absNodeIndex := 2;
    absEdgeIndex := 1;
    k := 0;

    while Length(nodes[2]) > 0 do
        if not IsFinite(W) then
            if k > 200 or absNodeIndex > 10000 then
                break;
            fi;
        fi;

        for i in [1..Length(nodes[2])] do
            Print(k, " ", i, "         \r");

            prevNode := nodes[2][i];
            for label in Filtered([1..Length(S)], n -> Position(List(prevNode.inEdges,
                e -> e.label), n) = fail) do
                deduction := TwistedInvolutionDeduceNodeAndEdgeFromGraph(matrix,
                    prevNode, label, [1..Length(S)]);

                if deduction.result = 0 then
                    type := deduction.type;
                    currNode := deduction.node;
                elif deduction.result = 1 then
                    type := deduction.type;

                    currNode := rec(element := y, twistedLength := k + 1, inEdges :=
                        [], outEdges := [], absIndex := absNodeIndex);
                    Add(nodes[1], currNode);

                    absNodeIndex := absNodeIndex + 1;
                else
                    x := prevNode.element;
                    s := S[label];

```

```
                    type := 1;
                    y := s^theta*x*s;
                    if (CoxeterElementsCompare(x, y)) then
                        y := x * s;
                        type := 0;
                    fi;

                    currNode := FindElement(deduction.possibleEqualNodes, n ->
                        CoxeterElementsCompare(n.element, y));

                    if currNode = fail then
                        currNode := rec(element := y, twistedLength := k + 1, inEdges
                            := [], outEdges := [], absIndex := absNodeIndex);
                        Add(nodes[1], currNode);

                        absNodeIndex := absNodeIndex + 1;
                    fi;
                fi;

                newEdge := rec(source := prevNode, target := currNode, label := label,
                    type := type, absIndex := absEdgeIndex);

                Add(edges[1], newEdge);
                Add(currNode.inEdges, newEdge);
                Add(prevNode.outEdges, newEdge);

                absEdgeIndex := absEdgeIndex + 1;
            od;
        od;

        TwistedInvolutionWeakOrderingPersistResults(persistInfo, nodes[2], edges[2]);

        Add(nodes, [], 1);
        Add(edges, [], 1);
        if (Length(nodes) > maxOrder + 1) then
            for n in nodes[maxOrder + 2] do
                n.inEdges := [];
                n.outEdges := [];
            od;
            Remove(nodes, maxOrder + 2);
            Remove(edges, maxOrder + 2);
        fi;
        k := k + 1;
    od;

    TwistedInvolutionWeakOrderingPersistResultsInfo(persistInfo, W, matrix, theta,
        absNodeIndex - 1, k - 1);
    TwistedInvolutionWeakOrderingPersistResultsClose(persistInfo);

    return rec(numNodes := absNodeIndex - 1, numEdges := absEdgeIndex - 1,
        maxTwistedLength := k - 1);
end;

# Calculates the poset Wk(theta).
TwistedInvolutionWeakOrdering1 := function (filename, W, matrix, theta)
    local persistInfo, maxOrder, nodes, edges, absNodeIndex, absEdgeIndex, prevNode,
        currNode, newEdge,
        label, type, deduction, startTime, endTime, S, k, i, s, x, y, n;

    persistInfo := TwistedInvolutionWeakOrderingPersistResultsInit(filename);

```

```
S := GeneratorsOfGroup(W);
maxOrder := Minimum([Maximum(Concatenation(matrix, [1])), 5]);
nodes := [ [], [ rec(element := One(W), twistedLength := 0, inEdges := [], outEdges := [], absIndex := 1) ] ];
edges := [ [], [] ];
absNodeIndex := 2;
absEdgeIndex := 1;
k := 0;

while Length(nodes[2]) > 0 do
    if not IsFinite(W) then
        if k > 200 or absNodeIndex > 10000 then
            break;
        fi;
    fi;

    for i in [1..Length(nodes[2])] do
        Print(k, " ", i, "         \r");

        prevNode := nodes[2][i];
        for label in Filtered([1..Length(S)], n -> Position(List(prevNode.inEdges, e -> e.label), n) = fail) do
            x := prevNode.element;
            s := S[label];

            type := 1;
            y := s^theta*x*s;
            if (CoxeterElementsCompare(x, y)) then
                y := x * s;
                type := 0;
            fi;

            currNode := FindElement(nodes[1], n -> CoxeterElementsCompare(n.element, y));

            if currNode = fail then
                currNode := rec(element := y, twistedLength := k + 1, inEdges := [], outEdges := [], absIndex := absNodeIndex);
                Add(nodes[1], currNode);

                absNodeIndex := absNodeIndex + 1;
            fi;

            newEdge := rec(source := prevNode, target := currNode, label := label, type := type, absIndex := absEdgeIndex);

            Add(edges[1], newEdge);
            Add(currNode.inEdges, newEdge);
            Add(prevNode.outEdges, newEdge);

            absEdgeIndex := absEdgeIndex + 1;
        od;
    od;

    TwistedInvolutionWeakOrderingPersistResults(persistInfo, nodes[2], edges[2]);

    Add(nodes, [], 1);
    Add(edges, [], 1);
    if (Length(nodes) > maxOrder + 1) then
        for n in nodes[maxOrder + 2] do
            n.inEdges := [];
```

```
                n.outEdges := [];
            od;
            Remove(nodes, maxOrder + 2);
            Remove(edges, maxOrder + 2);
        fi;
        k := k + 1;
    od;

    TwistedInvolutionWeakOrderingPersistResultsInfo(persistInfo, W, matrix, theta,
        absNodeIndex - 1, k - 1);
    TwistedInvolutionWeakOrderingPersistResultsClose(persistInfo);

    return rec(numNodes := absNodeIndex - 1, numEdges := absEdgeIndex - 1,
        maxTwistedLength := k - 1);
end;

TwistedInvolutionWeakOrderungResiduum := function (vertex, labels)
    local visited, queue, residuum, current, edge;

    visited := [ vertex ];
    queue := [ vertex ];
    residuum := [];

    while Length(queue) > 0 do
        current := queue[1];
        Remove(queue, 1);
        Add(residuum, current);

        for edge in current.outEdges do
            if edge.label in labels and not edge.target in visited then
                Add(visited, edge.target);
                Add(queue, edge.target);
            fi;
        od;
    od;

    return residuum;
end;

TwistedInvolutionWeakOrderungLongestWord := function (vertex, labels)
    local current;

    current := vertex;

    while Length(Filtered(current.outEdges, e -> e.label in labels)) > 0 do
        current := Filtered(current.outEdges, e -> e.label in labels)[1].target;
    od;

    return current;
end;
```

```
GroupAutomorphismByPermutation := function (G, generatorPermutation)
    local automorphism, generators;

    generators := GeneratorsOfGroup(G);

    if generatorPermutation = "id" or generatorPermutation = [1..Length(generators)]
        then
        automorphism := IdentityMapping(G);
        SetName(automorphism, "id");

```

```
        return automorphism;
    elif generatorPermutation = "-id" then
        generatorPermutation := Reversed([1..Length(GeneratorsOfGroup(G))]);
    fi;

    automorphism := GroupHomomorphismByImages(G, G, generators, generators{
        generatorPermutation});
    SetName(automorphism, Concatenation("(", JoinStringsWithSeparator(
        generatorPermutation, ","), ")"));

    return automorphism;
end;

GroupAutomorphismIdNeg := function (G)
    return GroupAutomorphismByPermutation(G, Reversed([1..Length(GeneratorsOfGroup(G))
        ]));
end;

GroupAutomorphismId := function (G)
    return GroupAutomorphismByPermutation(G, [1..Length(GeneratorsOfGroup(G))]);
end;

FindElement := function (list, selector)
    local i;

    for i in [1..Length(list)] do
        if (selector(list[i])) then
            return list[i];
        fi;
    od;

    return fail;
end;

StringToFilename := function(str)
    local result, c;

    result := "";

    for c in str do
        if IsDigitChar(c) or IsAlphaChar(c) or c = '-' or c = '_' then
            Add(result, c);
        else
            Add(result, '_');
        fi;
    od;

    return result;
end;

IO_ReadLinesIterator := function (file)
    local IsDone, Next, ShallowCopy;

    IsDone := function (iter)
        return iter!.nextLine = "" or iter!.nextLine = fail;
    end;

    Next := function (iter)
        local line;

        line := iter!.nextLine;
```

```

        if line = fail then
            Error(LastSystemError());
            return fail;
        fi;

        iter!.nextLine := IO_ReadLine(iter!.file);

        return Chomp(line);
    end;

    ShallowCopy := function (iter)
        return fail;
    end;

    return IteratorByFunctions(rec(IsDoneIterator := IsDone, NextIterator := Next,
        ShallowCopy := ShallowCopy, file := file, nextLine := IO_ReadLine(file)));
end;

IO_ReadLinesIteratorCSV := function (file, seperator)
    local IsDone, Next, ShallowCopy;

    IsDone := function (iter)
        return iter!.nextLine = "" or iter!.nextLine = fail;
    end;

    Next := function (iter)
        local line, lineSplitted, result, i;

        line := iter!.nextLine;
        if line = fail then
            Error(LastSystemError());
            return fail;
        fi;
        iter!.nextLine := IO_ReadLine(iter!.file);

        lineSplitted := SplitString(Chomp(line), iter!.seperator);
        result := rec();

        for i in [1..Minimum(Length(iter!.headers), Length(lineSplitted))] do
            result.(iter!.headers[i]) := EvalString(lineSplitted[i]);
        od;

        return result;
    end;

    ShallowCopy := function (iter)
        return fail;
    end;

    return IteratorByFunctions(rec(IsDoneIterator := IsDone, NextIterator := Next,
        ShallowCopy := ShallowCopy, file := file, seperator := seperator,
        headers := SplitString(Chomp(IO_ReadLine(file)), seperator),
        nextLine := IO_ReadLine(file)));
end;
```

```
Read("coxeter-generators.gap");

CoxeterElementsCompare := function (w1, w2)
    return w1 = w2;
end;
```

```

CoxeterMatrixEntry := function(matrix, rank, i, j)
    local temp;

    if (i = j) then
        return 1;
    fi;

    if (i > j) then
        temp := i;
        i := j;
        j := temp;
    fi;

    return matrix[(rank-1)*(rank)/2 - (rank-i)*(rank-i+1)/2 + (j-i-1) + 1];
end;
```

```
# Generates a coxeter group with given rank and relations. The relations have to
# be given in a linear list of the upper right entries (above diagonal) of the
# coxeter matrix.
#
# Example:
# To generate the coxeter group A_4 with the following coxeter matrix:
#
# | 1 3 2 2 |
# | 3 1 3 2 |
# | 2 3 1 3 |
# | 2 2 3 1 |
#
# A4 := CoxeterGroup(4, [3,2,2, 3,2, 3]);
CoxeterGroup := function (rank, upperTriangleOfCoxeterMatrix)
    local generatorNames, relations, F, S, W, i, j, k;

    generatorNames := List([1..rank], n -> Concatenation("s", String(n)));

    F := FreeGroup(generatorNames);
    S := GeneratorsOfGroup(F);

    relations := [];

    Append(relations, List([1..rank], n -> S[n]^2));

    k := 1;
    for i in [1..rank] do
        for j in [i+1..rank] do
            Add(relations, (S[i]*S[j])^(upperTriangleOfCoxeterMatrix[k]));
            k := k + 1;
        od;
    od;

    W := F / relations;

    return W;
end;

CoxeterGroup_An := function (n)
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := Flat(List(Reversed([1..n-1]), m -> Concatenation
        ([3], List([1..m-1], o -> 2))));

```

```
    #W := CoxeterGroup(n, upperTriangleOfCoxeterMatrix);
    W := GroupWithGenerators(List([1..n], s -> (s,s+1)));

    SetName(W, Concatenation("A_{", String(n), "}"));
    SetSize(W, Factorial(n + 1));

    return rec(group := W, rank := n, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_BCn := function (n)
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := Flat(List(Reversed([1..n-1]), m -> Concatenation
        ([3], List([1..m-1], o -> 2))));
    upperTriangleOfCoxeterMatrix[Length(upperTriangleOfCoxeterMatrix)] := 4;

    W := CoxeterGroup(n, upperTriangleOfCoxeterMatrix);

    SetName(W, Concatenation("BC_{", String(n), "}"));
    SetSize(W, 2^n * Factorial(n));

    return rec(group := W, rank := n, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_Dn := function (n)
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := Flat(List(Reversed([1..n-1]), m -> Concatenation
        ([3], List([1..m-1], o -> 2))));
    upperTriangleOfCoxeterMatrix[Length(upperTriangleOfCoxeterMatrix)] := 2;
    upperTriangleOfCoxeterMatrix[Length(upperTriangleOfCoxeterMatrix) - 1] := 3;
    upperTriangleOfCoxeterMatrix[Length(upperTriangleOfCoxeterMatrix) - 2] := 3;

    W := CoxeterGroup(n, upperTriangleOfCoxeterMatrix);

    SetName(W, Concatenation("D_{", String(n), "}"));
    SetSize(W, 2^(n-1) * Factorial(n));

    return rec(group := W, rank := n, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_E6 := function ()
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := [3, 2, 2, 2, 2,  3, 2, 2, 2,  3, 3, 2,  2, 2,  3];

    W := CoxeterGroup(6, upperTriangleOfCoxeterMatrix);

    SetName(W, "E_6");
    SetSize(W, 2^7 * 3^4 * 5);

    return rec(group := W, rank := 6, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_E7 := function ()
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := [3, 2, 2, 2, 2, 2,  3, 2, 2, 2, 2,  3, 3, 2, 2,  2,
         2, 2,  3, 2,  3];

    W := CoxeterGroup(7, upperTriangleOfCoxeterMatrix);
```

```

    SetName(W, "E_7");
    SetSize(W, 2^10 * 3^4 * 5 * 7);

    return rec(group := W, rank := 7, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_E8 := function ()
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := [3, 2, 2, 2, 2, 2, 2,  3, 2, 2, 2, 2, 2,  3, 3, 2,
        2, 2,  2, 2, 2, 2,  3, 2, 2,  3, 2,  3];

    W := CoxeterGroup(8, upperTriangleOfCoxeterMatrix);

    SetName(W, "E_8");
    SetSize(W, 2^14 * 3^5 * 5^2 * 7);

    return rec(group := W, rank := 8, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_F4 := function ()
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := [3, 2, 2, 4, 2, 3];

    W := CoxeterGroup(4, upperTriangleOfCoxeterMatrix);

    SetName(W, "F_4");
    SetSize(W, 2^7 * 3^2);

    return rec(group := W, rank := 4, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_H3 := function ()
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := [5, 2, 3];

    W := CoxeterGroup(3, upperTriangleOfCoxeterMatrix);

    SetName(W, "H_3");
    SetSize(W, 120);

    return rec(group := W, rank := 3, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_H4 := function ()
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := [5, 2, 2, 3, 2, 3];

    W := CoxeterGroup(4, upperTriangleOfCoxeterMatrix);

    SetName(W, "H_4");
    SetSize(W, 14400);

    return rec(group := W, rank := 4, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_I2m := function (m)
```

```
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := [m];

    W := CoxeterGroup(2, upperTriangleOfCoxeterMatrix);

    SetName(W, Concatenation("I_2(", String(m), ")"));
    SetSize(W, 2*m);

    return rec(group := W, rank := 2, matrix := upperTriangleOfCoxeterMatrix);
end;

CoxeterGroup_TildeAn := function (n)
    local upperTriangleOfCoxeterMatrix, W;

    upperTriangleOfCoxeterMatrix := Flat(List(Reversed([1..n]), m -> Concatenation([3],
         List([1..m-1], o -> 2))));

    if n = 1 then
        upperTriangleOfCoxeterMatrix[1] := 0;
    else
        upperTriangleOfCoxeterMatrix[n] := 3;
    fi;

    W := CoxeterGroup(n + 1, upperTriangleOfCoxeterMatrix);

    SetName(W, Concatenation("\\tilde A_{", String(n), "}"));
    SetSize(W, infinity);

    return rec(group := W, rank := n + 1, matrix := upperTriangleOfCoxeterMatrix);
end;
```

```
TwistedInvolutionWeakOrderingPersistReadResults := function(filename)
    local fileD, fileV, fileE, csvLine, data, vertices, edges, newEdge, source, target,
         i;

    fileD := IO_File(Concatenation("results/", filename, "-data"), "r");
    fileV := IO_File(Concatenation("results/", filename, "-vertices"), "r", 1024*1024);
    fileE := IO_File(Concatenation("results/", filename, "-edges"), "r", 1024*1024);

    data := NextIterator(IO_ReadLinesIteratorCSV(fileD, ";"));
    vertices := [];
    edges := [];

    i := 1;
    for csvLine in IO_ReadLinesIteratorCSV(fileV, ";") do
        Add(vertices, rec(absIndex := i, twistedLength := csvLine.twistedLength, name
             := csvLine.name, inEdges := [], outEdges := []));
        i := i + 1;
    od;

    i := 1;
    for csvLine in IO_ReadLinesIteratorCSV(fileE, ";") do
        source := vertices[csvLine.sourceIndex + 1];
        target := vertices[csvLine.targetIndex + 1];
        newEdge := rec(absIndex := i, source := source, target := target, label :=
             csvLine.label, type := csvLine.type);

        Add(source.outEdges, newEdge);
        Add(target.inEdges, newEdge);
        Add(edges, newEdge);
```

```
        i := i + 1;
    od;

    IO_Close(fileD);
    IO_Close(fileV);
    IO_Close(fileE);

    return rec(data := data, vertices := vertices, edges := edges);
end;

TwistedInvolutionWeakOrderingPersistResultsInit := function(filename)
    local fileD, fileV, fileE;

    if (filename = fail) then return fail; fi;

    fileD := IO_File(Concatenation("results/", filename, "-data"), "w");
    fileV := IO_File(Concatenation("results/", filename, "-vertices"), "w", 1024*1024);
    fileE := IO_File(Concatenation("results/", filename, "-edges"), "w", 1024*1024);
    IO_Write(fileD, "name;rank;size;generators;matrix;automorphism;wk_size;
        wk_max_length\n");
    IO_Write(fileV, "twistedLength;name\n");
    IO_Write(fileE, "sourceIndex;targetIndex;label;type\n");

    return rec(fileD := fileD, fileV := fileV, fileE := fileE);
end;

TwistedInvolutionWeakOrderingPersistResultsClose := function(persistInfo)
    if (persistInfo = fail) then return; fi;

    IO_Close(persistInfo.fileD);
    IO_Close(persistInfo.fileV);
    IO_Close(persistInfo.fileE);
end;

TwistedInvolutionWeakOrderingPersistResultsInfo := function(persistInfo, W, matrix,
    theta, numNodes, maxTwistedLength)
    if (persistInfo = fail) then return; fi;

    IO_Write(persistInfo.fileD, "\"", ReplacedString(Name(W), "\\", "\\\\"), "\";");
    IO_Write(persistInfo.fileD, Length(GeneratorsOfGroup(W)), ";");
    if (Size(W) = infinity) then
        IO_Write(persistInfo.fileD, "\"infinity\";");
    else
        IO_Write(persistInfo.fileD, Size(W), ";");
    fi;
    IO_Write(persistInfo.fileD, "[", JoinStringsWithSeparator(List(GeneratorsOfGroup(W)
        , n -> Concatenation("\"", String(n), "\"")), ","), "];");
    IO_Write(persistInfo.fileD, "[", JoinStringsWithSeparator(matrix, ","), "];");
    IO_Write(persistInfo.fileD, "\"", Name(theta), "\";");

    if (Size(W) = infinity) then
        IO_Write(persistInfo.fileD, "\"infinity\";");
        IO_Write(persistInfo.fileD, "\"infinity\"");
    else
        IO_Write(persistInfo.fileD, numNodes, ";");
        IO_Write(persistInfo.fileD, maxTwistedLength, "");
    fi;
end;

TwistedInvolutionWeakOrderingPersistResults := function(persistInfo, nodes, edges)
    local n, e, i, tmp, bubbles;
```

```

    if (persistInfo = fail) then return; fi;

    # bubble sort the edges, to make sure, that double edges are neighbours in the list
    bubbles := 1;
    while bubbles > 0 do
        bubbles := 0;
        for i in [1..Length(edges)-1] do
            if edges[i].source.absIndex = edges[i+1].source.absIndex and edges[i].
                target.absIndex > edges[i+1].target.absIndex then
                tmp := edges[i];
                edges[i] := edges[i+1];
                edges[i+1] := tmp;
                bubbles := bubbles + 1;
            fi;
        od;
    od;

    for n in nodes do
        if n.absIndex = 1 then
            IO_Write(persistInfo.fileV, n.twistedLength, ";\"e\"\n");
        else
            IO_Write(persistInfo.fileV, n.twistedLength, ";\"", String(n.element), "\"\
                n");
        fi;
    od;

    for e in edges do
        IO_Write(persistInfo.fileE, e.source.absIndex-1, ";", e.target.absIndex-1, ";",
             e.label, ";", e.type, "\n");
    od;
end;
```

B References

[1] Kathryn Brenneman, Ruth Haas, and Aloysius G. Helminck. Implementing an algorithm for the twisted involution poset for weyl groups. 2006.

[2] Tom Denton. Lifting property and poset structure of finite coxeter groups. 2009.

[3] Vinay V. Deodhar. Some characterizations of bruhat ordering on a coxeter group and determination of the relative möbius function. *Invent. Math. 39*, pages 187–198, 1977. MR0435249.

[4] Ruth Haas and Aloysius G. Helminck. Algorithms for twisted involutions in weyl groups. *Algebra Colloquium 19*, 2012.

[5] Axel Hultman. Fixed points of involutive automorphisms of the bruhat order. *Adv. Math. 195*, pages 283–296, 2005. MR2145798.

[6] Axel Hultman. The combinatorics of twisted involutions in coxeter groups. *Transactions of the American Mathematical Society, Volume 359*, pages 2787–2798, 2007. MR2286056.

[7] James E. Humphreys. *Reflection groups and Coxeter groups.* Cambridge University Press, 1992.