Posets of twisted involutions in Coxeter groups

Christian Hoffmeister

July 25, 2012

Contents

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 a 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 [Humphreys, 1992, Theorem 1.9]. Furthermore it turns out, that the finite reflection groups in the Euclidean space are precisely the finite Coxeter groups [Humphreys, 1992, Theorem 6.4].

In this chapter we will compile some basic facts on Coxeter groups, based on Humphreys [1992].

1.1 Introduction to Coxeter groups

Definition 1.1. Let $S = \{s_1, \ldots, s_n\}$ 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. This graph we call the **Coxeter graph** of our Coxeter system (W, S).

Definition 1.3. For an arbitrary element $w \in W$, (W, S) a Coxeter system, we call a product $s_{i_1} \cdots s_{i_n} = w$ of generators $s_{i_1} \ldots s_{i_n} \in S$ 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 infinitely 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 neseccarily distince) 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 ob 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**.

1.2 Exchange and Deletion Condition

We now obtain a way to get a reduced expression of an arbitrary element $s_1 \cdots s_r = w \in W$. But first we define what a reflection is. 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 1.5. *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 neseccarily 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 started from a reduced expression, then i is unique.*

Proof. See [Humphreys, 1992, Theorem 5.8]. □

This theorem is called the **Strong Exchange Condition**. The same theorem can be stated for $t \in S$ (since $S \subset T$). This weaker theorem is called **Exchange Condition**. The Exchange Condition immediatly yields another corollary for Coxeter groups:

Corollary 1.6. *Let (W, S) be a Coxeter system, $w \in W$ and $w = s_1 \cdots s_r$ with $s_i \in S$ a 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 a even number of generators (we just have to apply the Deletion Condition inductively).

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

Definition 1.7. 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 1.8. *Let (W, S) be a Coxeter system and $I, J \subset S$ two subsets of generators. Then $W_I \cap W_J = W_{I \cap J}$.*

Proof. Let $w \in W_{I \cap J}$. Then $w \in W_I$ and $w \in W_J$. To show the other inclusion we induce over the length r. For $r = 0$ we have $w = e$ and so $w \in W_{S'}$ for any $S' \subset S$. So suppose we have proven the assumption for all lengths up to $r - 1$. Let $w \in W_I \cap W_J$ with $l(w) = r$. Then we have two reduced expressions $w = s_1 \cdots s_r = t_1 \cdots t_r$ with $s_i \in I$ and $t_i \in J$. By applying s_r from the right we get $w s_r = s_1 \cdots s_{r-1} = t_1 \cdots t_r s_r$. The expression $t_1 \cdots t_r s_r$ is of length $r - 1$, so Exchange Condition yields $w s_r = s_1 \cdots s_{r-1} = t_1 \cdots \hat{t}_i \cdots t_r$, hence $w s_r \in W_I \cap W_J$. Due to induction we know that $w s_r \in W_{I \cap J}$. TODO □

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 also infinite Coxeter groups without an ∞-relation between two generators. 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 it be seen that this W is infinite?

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

Definition 1.9. A Coxeter system is called **irreducible**, if the corrosponding Coxeter graph is connected. Else it is called **reducible**.

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

Proposition 1.10. *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.*

Proof. See [Humphreys, 1992, Proposition 6.1]. □

This proposition tells us, that an arbitray Coxeter system is finite iff its irreducible parabolic subgroups are finite. Therefor 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 1.11. *The irreducible finite Coxeter systems are exactly the ones in Figure **??**.*

$A_n (n \geq 1)$ E_8

$B_n (n \geq 2)$ F_4

$D_n (n \geq 4)$ H_3

E_6 H_4

E_7 $I_2(m)$
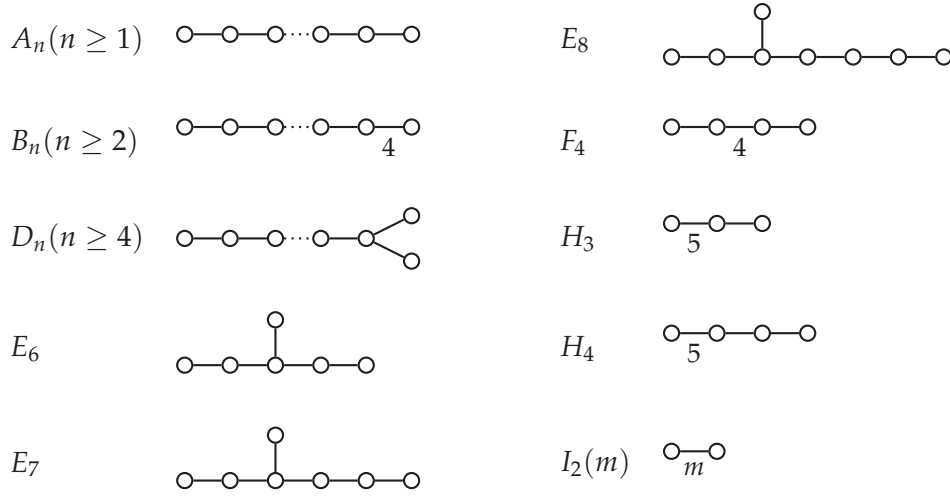
Figure 1: All types of irreducible finite Coxeter systems

Proof. [Humphreys, 1992, Theorem 6.4]   □

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

1.4 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. The most useful way to achieve this is the Bruhat ordering [Humphreys, 1992, Section 5.9].

Definition 1.12. Let M be a set. A binary relation, in this case often denoted as "$\leq$", is called a **partial order** over M, if fullfills the following conditions for all $a, b, c \in M$:

1. $a \leq a$, called **reflexivity**

2. if $a \leq b$ and $b \leq a$ then $a = b$, called **antisymmetry**

3. if $a \leq b$ and $b \leq c$ then $a \leq c$, called **transitivity**

In this case $(M, \leq)$ is called a **poset**.

Definition 1.13. 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 as $\mathrm{Br}(W)$.

Lemma 1.14. *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 $e \to w_1 \to w_2 \to \ldots$ are strictly ascending in length, it must be antisymmetric. By concatenation of sequences we get the transitivity.   □

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. Before lets just mention two other partial orderings, where this property is obvious by definition:

Definition 1.15. 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**.

So lets ensure that the Bruhat ordering is graded as well. For this we need another characterization of the Bruhat ordering with subexpressions. As we will show it is $u \leq w$ iff there is a reduced expression of u that is a subexpression of a reduced expression of w.

Proposition 1.16. *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. $\square$

2 The twisted weak ordering in Coxeter groups

In this section we will introduce the twisted weak ordering $Wk(\theta)$ on Coxeter groups.

3 Residuums of rank 2

Definition 3.1 (One- and twosided operation). Let (W, S) be a Coxeter system and $w \in W, s \in S$. If $w\underline{s} = \theta(s)ws$, then we say s to operate twosided on w. Else we say s to operate onesided on w.

Definition 3.2 (Ein- und beidseitige endende Gesamtwirkung). Seien (W, S) ein Coxetersystem, $w \in W$ und $s_1, \ldots, s_n \in S$. Falls $w\underline{s_1 \cdots s_n} = \theta(s_n)(w\underline{s_1 \cdots s_{n-1}})s_n$ ist, so sagen wir, dass $s_1 \cdots s_n$ eine beidseitig endende Gesamtwirkung auf w hat. Andernfalls sagen wir $s_1 \cdots s_n$ hat eine einseitig endende Gesamtwirkung auf w.

Definition 3.3. Let (W, S) be a Coxeter system and $s, t \in S$ two distinct generators. We define:

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

Assumption 3.4. Seien (W, S) ein Coxetersystem und $s, t \in S$ zwei verschiedene Erzeuger von W. Dann gilt:

1. Sei $m = \mathrm{ord}(st) < \infty$. Falls $w\underline{[st]^n} \neq w$ ist für alle $n \in \mathbb{N}, n < 2m$, dann gilt $w\underline{(st)^{2m}} = w$.

2. In $wC_{\{s,t\}}$ existieren keine drei Elemente derselben getwisteten Länge.

3. Falls s einseitig auf w wirkt, dann gilt $w\underline{st} < w\underline{s}$ oder $w\underline{t} > w$.

4. Sei $w\underline{[st]^n} = w$ für ein $n \in \mathbb{N}$. Dann ist n gerade und es gilt eine der beiden folgenden Eigenschaften:

   a) Für jedes $m \in \mathbb{N}$ hat das Element $[st]^m$ genau dann eine beidseitig endende Gesamtwirkung auf w, wenn $[st]^{n/2+m}$ eine beidseitig endende Gesamtwirkung auf w hat.

   b) Für jedes $m \in \mathbb{N}$ hat das Element $[st]^m$ genau dann eine beidseitig endende Gesamtwirkung auf w, wenn $[st]^{n-m+1}$ eine beidseitig endende Gesamtwirkung auf w hat.

Lemma 3.5. *Let (W, S) be a Coxeter system, $w \in W$ and $s, t \in S$ two distinct generators. Then $wC_{\{s,t\}}$ does not contain three elements of same twisted length.*

Proof. Let (W, S) be a Coxeter system, $w \in W$ with $\mathrm{rank}\, w = k$, $s, t \in S$ with $s \neq t$. Without loss of generality we can choose w such that $w < w\underline{s}$ and $w < w\underline{t}$. Assume the existence of an element $u \in wC_{\{s,t\}}$ with $u\underline{s} < u$ and $u\underline{t} < u$. Then [Hultman, 2007, Lemma 3.8] yields $s, t \in D_R(u)$. By using [Hultman, 2007, Lemma 3.9] we conclude that $w\underline{s} \leq u$ and $w\underline{t} \leq u$. Hence there cannot exist more than two Elements of same twisted length.

If no such u exists, then $wC_{\{s,t\}} = w \,\dot{\cup}\, \{w\underline{[st]^n} : n \in \mathbb{N}\} \,\dot{\cup}\, \{w\underline{[ts]^n} : n \in \mathbb{N}\}$ and the assumption still holds. $\qquad \square$

Lemma 3.6. *Let (W, S) be a Coxeter system, $w \in S$ and $s, t \in S$ two distinct generators. If s operates onesided on w and $w\underline{s} < w$, then either $w\underline{st} < w\underline{s}$ or $w\underline{t} > w$.*

Proof. We have $\theta(s)ws = w$ and $s \in D_R(w)$. If $t \notin D_R(w)$, then we are done. So suppose $t \in D_R(w)$. This means $w\underline{s} \leq w$ and $w\underline{t} \leq w$ and [Hultman, 2007, Lemma 3.9] yields $w\underline{st} < w$ and $w\underline{ts} < w$. If $t \in D_R(w\underline{s})$, then we are done. So suppose $t \notin D_R(w\underline{s})$. Then $t \in D_R(w\underline{st})$. Together with $w\underline{st} \leq w$ [Hultman, 2007, Lemma 3.9(2)] says $(w\underline{st})\underline{t} \leq w\underline{t}$. Finally we get

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

Since $w\underline{s}$ and $w\underline{t}$ are of same twisted length they have to be equal and therefore $s = t$ which contradicts to our assumption of two distinct generators s and t. $\qquad\square$

4 Twisted weak ordering

Wir wollen nun einen Algorithmus zur Berechnung der getwisteten schwachen Ordnung $Wk(\theta)$ einer beliebigen Coxetergruppe W erarbeiten. Also Ausgangspunkt werden wir den Algorithmus aus [Haas and Helmnick, 2012, Algorithm 3.1.1] verwenden, der im wesentlichen benutzt, dass für jede getwistete Involution $w \in \mathcal{I}_\theta$ entweder $w\underline{s} < w$ oder aber $w\underline{s} > w$ gilt.

Algorithm 4.1 (Algorithmus 1)**.**

```
 1: procedure TwistedWeakOrderingAlgorithm1(W)        ▷ W sei die Coxetergruppe
 2:     V ← {(e, 0)}
 3:     E ← {}
 4:     for k ← 0 to k_max do
 5:         for all (w, k_w) ∈ V with k_w = k do
 6:             for all s ∈ S with ∄(·, w, s) ∈ E do    ▷ Nur die s, die nicht schon nach w
                                                           führen
 7:                 y ← ws
 8:                 z ← θ(s)y
 9:                 if z = w then
10:                     x ← y                           ▷ s operiert ungetwistet auf w
11:                     t ← s
12:                 else
13:                     x ← z                           ▷ s operiert getwistet auf w
14:                     t ← s̲
15:                 end if
16:                 isNew ← true
17:                 for all (w', k_{w'}) ∈ V with k_{w'} = k + 1 do   ▷ Prüfen, ob x nicht schon in
                                                                       V liegt
18:                     if x = w' then
19:                         isNew ← false
20:                     end if
21:                 end for
22:                 if isNew = true then
23:                     V ← V ∪ {(x, k + 1)}
24:                     E ← E ∪ {(w, x, t)}
25:                 else
26:                     E ← E ∪ {(w, x, t)}
27:                 end if
28:             end for
29:         end for
30:         k ← k + 1
31:     end for
```

W	$|Wk(\mathrm{id}, W)|$	$\rho(w_0)$	Timings		Element compares	
			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 1: Benchmark

32:      **return** (V, E) ▷ The poset graph
33: **end procedure**

Dieser Algorithmus berechnet alle getwisteten Involutionen und deren getwistete Länge (w, k_w) und deren Relationen (w', w, s) bzw. $(w', w, \underline{s})$. Zu bemerken ist, dass zur Berechnung der getwisteten Involutionen der Länge k nur die Knoten aus V benötigt werden, mit der getwisteten Länge $k - 1$ und k sowie die Kanten aus E, die Knoten der Länge $k - 2$ und $k - 1$ bzw. $k - 1$ und k verbinden. Alle vorherigen Ergebnisse können schon persistiert werden, so dass nie das komplette Ergebnis im Speicher gehalten werden muss.

Eine Operation, die hier als elementar angenommen wurde ist der Vergleich von Elementen in W. Für bestimmte Gruppen wie z.B. die A_n, welche je isomorph zu $Sym(n + 1)$ sind, lässt sich der Vergleich von Element effizient implementieren. Will man jedoch mit Coxetergruppen im Allgemeinen arbeiten, so liegt W als frei präsentierte Gruppe vor und der Vergleich von Element is eine sehr aufwendige Operation. Bei Algorithm 4.1 muss jedes potentiell neue Element x mit allen schon bekannten w' von gleicher getwisteter Länge verglichen werden um zu bestimmen, ob x wirklich ein noch nicht bekanntes Element aus $\mathcal{I}_\theta$ ist.

Algorithm 4.2 (Algorithmus 2)**.**

  1: **procedure** TwistedWeakOrderingAlgorithm2(W)   ▷ W sei die Coxetergruppe
  2:      $V \leftarrow \{(e, 0)\}$
  3:      $E \leftarrow \{\}$
  4:      **for** $k \leftarrow 0$ **to** $k_{\max}$ **do**
  5:          TODO
  6:      **end for**
  7:      **return** (V, E) ▷ The poset graph
  8: **end procedure**

Im Anhang findet sich eine Implementierung der Algorithms 4.1 und 4.2 in GAP 4.5.4. Table **??** zeigt ein Benchmark anhand von fünf ausgewählten Coxetergruppen.
Dabei sind die A_n als symmetrische Gruppen implementiert und die E_n als frei präsentierte Gruppen. Ausgeführt wurden die Messungen auf einem Intel Core i5-3570k mit

vier Kernen zu je 3,40 GHz. Der Algorithmus ist dabei aber nur single threaded und kann
so nur auf einem Kern laufen. Um die Messergebnisse nicht durch Limitierungen des
Datenspeichers zu beeinflussen, wurden die Daten in diesem Benchmark nicht stück-
weise persistiert sondern ausschließlich berechnet.

Wie zu erwarten ist der Geschwindigkeitsgewinn bei den Coxetergruppen vom Typ E_n
deutlich größer, da in diesem Fall die Elementvergleiche deutlich aufwendiger sind als
bei Gruppen vom Typ A_n.

5 Miscellaneous

Definition 5.1 (Geodesic). Let (W, S) be a Coxeter system and $w, u \in W$ with $\rho(u) - \rho(w) = n$. Each sequence $w = w_0 < w_1 < \ldots < w_n = u$ is called a geodesic from w to u.

Question 5.2. *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 $\theta(K) = K$. Futhermore let $T, S_1, S_2, S_3 \subset S$ be four pairwise disjoint sets of generators. For which Coxeter groups W does the implication*

$$w \in w_K C_{T \cup S_i}, i = 1, 2, 3 \Rightarrow w \in w_K C_T \tag{5.2.1}$$

hold for any possible $K, \theta, T, S_1, S_2, S_3$ and w?

Proposition 5.3. *Let (W, S) be a Coxeter system and K, T, S_1, S_2, S_3 be like in Question 5.2. Suppose we have $w \in W$ and $a_1, \ldots, a_n \in T \cup S_1$, $b_1, \ldots, b_n \in T \cup S_2$, $c_1, \ldots, c_n \in T \cup S_3$ with*

$$
\begin{aligned}
w &= w_K \underline{a_1 \cdots a_n} \\
&= w_K \underline{b_1 \cdots b_n} \\
&= w_K \underline{c_1 \cdots c_n}
\end{aligned}
$$

and (5.2.1) does not hold for these three expressions, i.e. $w \notin w_K C_T$. Then there exist $t_1, \ldots, t_m \in T$ and $a'_1, \ldots, a'_{n-m} \in T \cup S_1$, $b'_1, \ldots, b'_{n-m} \in T \cup S_2$, $c'_1, \ldots, c'_{n-m} \in T \cup S_3$ such that

$$
\begin{aligned}
w \underline{t_1 \ldots t_m} &= w_K \underline{a'_1 \cdots a'_{n-m}} \\
&= w_K \underline{b'_1 \cdots b'_{n-m}} \\
&= w_K \underline{c'_1 \cdots c'_{n-m}}
\end{aligned}
$$

with $a'_{n-m}, b'_{n-m}, c'_{n-m} \notin T$.

Proof. Suppose at least one element of a_n, b_n, c_n to be in T, for example $a_n \in T$. Then we can apply $\underline{a_n}$ to all three expressions. Since $\rho(w\underline{a_n}) < \rho(w)$ the exchange condition for $\mathcal{I}_\theta$ [Hultman, 2007, Proposition 3.10] yields

$$
\begin{aligned}
w\underline{a_n} = w_K \underline{a_1 \cdots a_n} a_n &= w_K \underline{a_1 \cdots a_{n-1}} \\
= w_K \underline{b_1 \cdots b_n} a_n &= w_K \underline{b_1 \cdots \hat{b}_i \cdots b_n} \\
= w_K \underline{c_1 \cdots c_n} a_n &= w_K \underline{c_1 \cdots \hat{c}_j \cdots c_n}
\end{aligned}
$$

where $\hat{}$ means omission. The omission cannot occur within w_K since all three expressions are still of same twisted length and in the first expression we can see, that $w_K \leq w\underline{a_n}$ still holds. This step can be repeated until $w = w_K$ or $a_n, b_n, c_n \notin T$. $\square$

Lemma 5.4. *A counterexample to Question 5.2 can only exist, if there is an element $u \in wC_T$ and three distinct generators $s_1, s_2, s_3 \in D_r(u)$ such that $u\underline{s_i} \notin wC_T$ for $i = 1, 2, 3$.*

Proof. According to Proposition 5.3. $\qquad\square$

Lemma 5.5. *A counterexample to Question 5.2 can only exist, if there are three not neseccarily distinct elements $a, b, c \in w_K C_{S \setminus T}$, three distinct generators $s_1 \in A_r(a)$, $s_2 \in A_r(b)$, $s_3 \in A_r(c)$ and an element $u \notin w_K C_{S \setminus T}$ such that*

$$a\underline{s_1} = b\underline{s_2} = c\underline{s_3} = u.$$

Proof. If there is a counterexample, then the two residuums $w_K C_{S \setminus T}$ and $w C_T$ are disjunct. Since we are only interested in w with $w_K \leq w$ it follows, that any geodesic from w_K to w is contained in the union set of both residuums. Hence having one element in $u \in w C_T$ with three distinct generators s_1, s_2, s_3 with $u\underline{s_i} \notin w C_T$ is equivalent to having three elements $a, b, c \notin w C_T$ and the same three generator s_1, s_2, s_3 with $a\underline{s_1} = b\underline{s_2} = c\underline{s_3} = u \in w C_T$. $\qquad\square$

Lemma 5.6. *Let (W, S) be a Coxeter system, $w \in W$ and $s \in S$. Then $s \in D_R(w)$ iff $w\underline{s} < w$.*

Proof. TODO $\qquad\square$

A Source codes

```
1   LoadPackage("io");
2
3   Read("misc.gap");
4   Read("coxeter.gap");
5   Read("twistedinvolutionweakordering-persist.gap");
6
7   TwistedInvolutionDeduceNodeAndEdgeFromGraph := function(matrix, startNode, startLabel,
        labels)
8       local rank, comb, trace, possibleEqualNodes, e, k, n;
9
10      rank := -1/2 + Sqrt(1/4 + 2*Length(matrix)) + 1;
11      possibleEqualNodes := [];
12
13      for comb in List(Filtered(labels, label -> label <> startLabel), label -> rec(
            startNode := startNode, s := [startLabel, label], m := CoxeterMatrixEntry(
            matrix, rank, startLabel, label))) do
14          trace := [];
15          k := 1;
16          n := comb.startNode;
17
18          Add(trace, rec(node := n, edge := rec(label := comb.s[1], type := -1)));
19
20          while k < comb.m do
21              e := FindElement(n.inEdges, e -> e.label = comb.s[k mod 2 + 1]);
22              if e = fail then break; fi;
23              n := e.source;
24
25              Add(trace, rec(node := n, edge := e));
26              k := k + 1;
27          od;
28
29          while k > 0 do
30              e := FindElement(n.outEdges, e -> e.label = comb.s[k mod 2 + 1]);
31              if e = fail then break; fi;
32              n := e.target;
33
34              Add(trace, rec(node := n, edge := e));
35              k := k - 1;
36          od;
37
38          if k <> 0 then continue; fi;
39
40          if Length(trace) = 2*comb.m then
41              return rec(result := 0, node := trace[Length(trace)].node, type := trace[
                    comb.m + 1].edge.type, trace := trace);
42          fi;
43
44          if Length(trace) >= 4 then
45              if trace[Length(trace) / 2 + 1].edge.type <> trace[Length(trace) / 2].edge.
                    type then
46                  # cannot be equal
47              else
48                  if trace[Length(trace)].edge.type = 0 then
49                      return rec(result := 0, node := trace[Length(trace)].node, type :=
                            0, trace := trace);
50                  else
51                      Add(possibleEqualNodes, trace[Length(trace)].node);
52                  fi;
```

```
53                  fi;
54              else
55                  Add(possibleEqualNodes, trace[Length(trace)].node);
56              fi;
57          od;
58
59          return rec(result := -1, possibleEqualNodes := possibleEqualNodes);
60      end;
61
62      # Calculates the poset Wk(theta).
63      TwistedInvolutionWeakOrdering := function (filename, W, matrix, theta)
64          local persistInfo, maxOrder, nodes, edges, absNodeIndex, absEdgeIndex, prevNode,
                  currNode, newEdge,
65              label, type, deduction, startTime, endTime, S, k, i, s, x, y, n;
66
67          persistInfo := TwistedInvolutionWeakOrderingPersistResultsInit(filename);
68
69          S := GeneratorsOfGroup(W);
70          maxOrder := Minimum([Maximum(Concatenation(matrix, [1])), 5]);
71          nodes := [ [], [ rec(element := One(W), twistedLength := 0, inEdges := [], outEdges
                  := [], absIndex := 1) ] ];
72          edges := [ [], [] ];
73          absNodeIndex := 2;
74          absEdgeIndex := 1;
75          k := 0;
76
77          while Length(nodes[2]) > 0 do
78              if not IsFinite(W) then
79                  if k > 200 or absNodeIndex > 10000 then
80                      break;
81                  fi;
82              fi;
83
84              for i in [1..Length(nodes[2])] do
85                  Print(k, " ", i, "          \r");
86
87                  prevNode := nodes[2][i];
88                  for label in Filtered([1..Length(S)], n -> Position(List(prevNode.inEdges,
                          e -> e.label), n) = fail) do
89                      deduction := TwistedInvolutionDeduceNodeAndEdgeFromGraph(matrix,
                              prevNode, label, [1..Length(S)]);
90
91                      if deduction.result = 0 then
92                          type := deduction.type;
93                          currNode := deduction.node;
94                      elif deduction.result = 1 then
95                          type := deduction.type;
96
97                          currNode := rec(element := y, twistedLength := k + 1, inEdges :=
                                  [], outEdges := [], absIndex := absNodeIndex);
98                          Add(nodes[1], currNode);
99
100                         absNodeIndex := absNodeIndex + 1;
101                     else
102                         x := prevNode.element;
103                         s := S[label];
104
105                         type := 1;
106                         y := s^theta*x*s;
107                         if (CoxeterElementsCompare(x, y)) then
108                             y := x * s;
```

```
109                             type := 0;
110                     fi;
111
112                     currNode := FindElement(deduction.possibleEqualNodes, n ->
                            CoxeterElementsCompare(n.element, y));
113
114                     if currNode = fail then
115                         currNode := rec(element := y, twistedLength := k + 1, inEdges
                                := [], outEdges := [], absIndex := absNodeIndex);
116                         Add(nodes[1], currNode);
117
118                         absNodeIndex := absNodeIndex + 1;
119                     fi;
120                 fi;
121
122                 newEdge := rec(source := prevNode, target := currNode, label := label,
                        type := type, absIndex := absEdgeIndex);
123
124                 Add(edges[1], newEdge);
125                 Add(currNode.inEdges, newEdge);
126                 Add(prevNode.outEdges, newEdge);
127
128                 absEdgeIndex := absEdgeIndex + 1;
129             od;
130         od;
131
132         TwistedInvolutionWeakOrderingPersistResults(persistInfo, nodes[2], edges[2]);
133
134         Add(nodes, [], 1);
135         Add(edges, [], 1);
136         if (Length(nodes) > maxOrder + 1) then
137             for n in nodes[maxOrder + 2] do
138                 n.inEdges := [];
139                 n.outEdges := [];
140             od;
141             Remove(nodes, maxOrder + 2);
142             Remove(edges, maxOrder + 2);
143         fi;
144         k := k + 1;
145     od;
146
147     TwistedInvolutionWeakOrderingPersistResultsInfo(persistInfo, W, matrix, theta,
            absNodeIndex - 1, k - 1);
148     TwistedInvolutionWeakOrderingPersistResultsClose(persistInfo);
149
150     return rec(numNodes := absNodeIndex - 1, numEdges := absEdgeIndex - 1,
            maxTwistedLength := k - 1);
151 end;
152
153 # Calculates the poset Wk(theta).
154 TwistedInvolutionWeakOrdering1 := function (filename, W, matrix, theta)
155     local persistInfo, maxOrder, nodes, edges, absNodeIndex, absEdgeIndex, prevNode,
            currNode, newEdge,
156         label, type, deduction, startTime, endTime, S, k, i, s, x, y, n;
157
158     persistInfo := TwistedInvolutionWeakOrderingPersistResultsInit(filename);
159
160     S := GeneratorsOfGroup(W);
161     maxOrder := Minimum([Maximum(Concatenation(matrix, [1])), 5]);
162     nodes := [ [], [ rec(element := One(W), twistedLength := 0, inEdges := [], outEdges
            := [], absIndex := 1) ] ];
```

```
163        edges := [ [], [] ];
164        absNodeIndex := 2;
165        absEdgeIndex := 1;
166        k := 0;
167
168        while Length(nodes[2]) > 0 do
169            if not IsFinite(W) then
170                if k > 200 or absNodeIndex > 10000 then
171                    break;
172                fi;
173            fi;
174
175            for i in [1..Length(nodes[2])] do
176                Print(k, " ", i, "          \r");
177
178                prevNode := nodes[2][i];
179                for label in Filtered([1..Length(S)], n -> Position(List(prevNode.inEdges,
                       e -> e.label), n) = fail) do
180                    x := prevNode.element;
181                    s := S[label];
182
183                    type := 1;
184                    y := s^theta*x*s;
185                    if (CoxeterElementsCompare(x, y)) then
186                        y := x * s;
187                        type := 0;
188                    fi;
189
190                    currNode := FindElement(nodes[1], n -> CoxeterElementsCompare(n.element
                           , y));
191
192                    if currNode = fail then
193                        currNode := rec(element := y, twistedLength := k + 1, inEdges :=
                               [], outEdges := [], absIndex := absNodeIndex);
194                        Add(nodes[1], currNode);
195
196                        absNodeIndex := absNodeIndex + 1;
197                    fi;
198
199                    newEdge := rec(source := prevNode, target := currNode, label := label,
                           type := type, absIndex := absEdgeIndex);
200
201                    Add(edges[1], newEdge);
202                    Add(currNode.inEdges, newEdge);
203                    Add(prevNode.outEdges, newEdge);
204
205                    absEdgeIndex := absEdgeIndex + 1;
206                od;
207            od;
208
209        TwistedInvolutionWeakOrderingPersistResults(persistInfo, nodes[2], edges[2]);
210
211        Add(nodes, [], 1);
212        Add(edges, [], 1);
213        if (Length(nodes) > maxOrder + 1) then
214            for n in nodes[maxOrder + 2] do
215                n.inEdges := [];
216                n.outEdges := [];
217            od;
218            Remove(nodes, maxOrder + 2);
219            Remove(edges, maxOrder + 2);
```

```
220            fi;
221         k := k + 1;
222     od;
223
224     TwistedInvolutionWeakOrderingPersistResultsInfo(persistInfo, W, matrix, theta,
            absNodeIndex - 1, k - 1);
225     TwistedInvolutionWeakOrderingPersistResultsClose(persistInfo);
226
227     return rec(numNodes := absNodeIndex - 1, numEdges := absEdgeIndex - 1,
            maxTwistedLength := k - 1);
228 end;
229
230 TwistedInvolutionWeakOrderungResiduum := function (vertex, labels)
231     local visited, queue, residuum, current, edge;
232
233     visited := [ vertex ];
234     queue := [ vertex ];
235     residuum := [];
236
237     while Length(queue) > 0 do
238         current := queue[1];
239         Remove(queue, 1);
240         Add(residuum, current);
241
242         for edge in current.outEdges do
243             if edge.label in labels and not edge.target in visited then
244                 Add(visited, edge.target);
245                 Add(queue, edge.target);
246             fi;
247         od;
248     od;
249
250     return residuum;
251 end;
252
253 TwistedInvolutionWeakOrderungLongestWord := function (vertex, labels)
254     local current;
255
256     current := vertex;
257
258     while Length(Filtered(current.outEdges, e -> e.label in labels)) > 0 do
259         current := Filtered(current.outEdges, e -> e.label in labels)[1].target;
260     od;
261
262     return current;
263 end;
```

B Bibliography

R. Haas and A. G. Helmnick. Algorithms for twisted involutions in weyl groups. *Algebra Colloquium 19*, 2012.

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

J. E. Humphreys. *Reflection groups and Coxeter groups.* Cambridge University Press, 1992.