DIPLOMARBEIT

Titel der Diplomarbeit

Highly Arc Transitive Digraphs

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag.rer.nat)

Verfasser	Christoph Marx
Matrikel–Nummer	0103771
Studienrichtung	Mathematik
Betreuer	Bernhard Krön

Wien, am 4. 7. 2010

Contents

Abstract

Infinite, highly arc transitive digraphs are defined and examples are given. The **Reachability–Relation** and **Property-Z** are defined and investigated on infinite, highly arc transitive digraphs using the **valencies**, **spread** and other properties arising from the investigation of the **descendants of lines** or the **automorphism groups**. *Seifters* theorems about highly arc transitive digraphs with more than one **end**, his conjecture on them and the counterexamples that disproved his conjecture, are given. A condition for **C–homogeneous** digraphs to be highly arc transitve is stated and the connection between highly arc transitive digraphs and **totally disconnected**, **topological groups** is mentioned. Some notes on the **Cameron–Praeger–Wormald–Conjecture** are made and a refined conjecture is stated. The properties of the known highly arc transitive digraphs are collected, some but not all of them are **Cayley–graphs**. Finally open questions and conjectures are stated and new ones are added. For the given lemmas, propositions and theorems either proofs or references to proofs are included.

Kurzbeschreibung

Unendliche, hochgradig bogentransitive Digraphen werden definiert und anhand von Beispielen vorgestellt. Die **Erreichbarkeitsrelation** und **Eigenschaft–Z** werden definiert und unter Verwendung von **Knotengraden**, **Wachstum** und anderen Eigenschaften, die von der Untersuchung von **Nachkommen von Doppelstrahlen** oder **Automorphismengruppen** herrühren, auf hochgradig bogentransitiven Digraphen untersucht. *Seifters* Theoreme über hochgradig bogentransitive Digraphen mit mehr als einem **Ende**, seine daherrührende Vermutung und deren sie widerlegende Gegenbeispiele werden vorgestellt. Eine Bedingung, unter der **C–homogene** Digraphen hochgradig bogentransitiv sind, wird angegeben und die Verbindung zwischen hochgradig bogentransitiven Digraphen und **total unzusammenhängenden**, **topologischen Gruppen** wird erwähnt. Einige Bemerkungen über die **Vermutung von Cameron–Praeger–Wormald** werden gemacht und eine verfeinerte Version vermutet. Die Eigenschaften der bekannten hochgradig bogentransitiven Digraphen werden gesammelt. Es wird festgestellt, dass einige, aber nicht alle unter ihnen **Cayley–Graphen** sind. Schließlich werden offen gebliebene Fragestellungen und Vermutungen zusammengefasst und neue hinzugefügt. Für die vorgestellten Lemmata, Propositionen und Theoreme sind entweder Beweise enthalten, oder Referenzen zu Beweisen werden angegeben.

Declaration

I declare that I unassistedly wrote this thesis without additonal help other than supervision and that I didn't use other than quoted sources and knowledge from lectures.

Vienna, July 4, 2010

Christoph Marx

Erklärung

Ich erkläre, dass ich die vorliegende Diplomarbeit selbständig und abgesehen von der vorgesehenen Betreuung ohne fremde Hilfe verfasst und dass ich nur die zitierten Quellen und Wissen aus Lehrveranstaltugen verwendet habe.

Wien am 4. 7. 2010

Christoph Marx

Editorial remark

The setting of the task for the present thesis arose from the project **Vertex transitive infinite graphs and digraphs** in cooperation with the **Department Mathematik und Informationstechnologie Montanuniversität Leoben** and the **Univerza v Ljubljani, Pedagoška fakulteta**. The project was supported by the OeAD - Österreichischer Austauschdienst (`http://www.oead.at`).

Acknowledgement

First and foremost, I want to thank my advisor *Bernhard Krön* for suggesting the topic HIGHLY ARC TRANSITIVE DIGRAPHS. It has not just broadened my horizon mathematically but was also a joy to work upon. It has also brought me back to my roots in informatics since I could use my programming skills for the illustrations. Thus my thanks also go to all my past programming–teachers.
I want to thank everybody who has helped me getting familiar with the topic of highly arc transitive digraphs, especially *Primož Šparl*, *Aleksander Malnič* and *Norbert Seifter*.
Once more I want to thank *Bernhard Krön* for his supervision, all his helpful hints and for never tiring in subject–specific conversation.
My special thanks go to *Éva* for her love, support, patience and the pizza.
Finally, I want to express special thanks to *Aleksander Malnič* for the pullover.

Vienna, July 4, 2010

Christoph Marx

1. Preface

The topic of **infinite highly arc transitive digraphs** started with the paper [1] by *Cameron*, *Praeger* and *Wormald*. The first parts of this paper were already studied in a thesis by *Primož Šparl* written in slovenian language. Thus the present thesis goes the other way round starting at the end of [1] where examples and constructions are given. Afterwards, we work through the properties and theorems, whose number of course increased since *Primož Šparls* thesis. This makes sense as it is neither easy to imagine such graphs nor are there too many known examples. First we look at what infinite highly arc transitive digraphs are to get a picture of the objects we work with.
The object of the present thesis is to collect knowledge about highly arc transitive digraphs rather than collecting proofs. Most of the statements are claimed and have their proofs or references to their proofs in Section 5. The pages where the proofs can be found are indicated with "$\rightarrow$*page*".

1.1. Group Actions

Definition 1.1 (Group action) *Let G be a group and S a set. Let every $g \in G$ define a bijection $g : S \to S$. We say G* **acts** *on S if*

1. $\forall s \in S : \mathbf{1}(s) = s$
2. $\forall s \in S, g, h \in G : g(h(s)) = gh(s)$

If G acts on S we write $G \circlearrowleft S$.

For convenience, we do not denote the brackets unless there is room for misunderstanding.

Definition 1.2 (Pointwise stabilizer) *Let $G \circlearrowleft M$ and $P \subset M$. We define the* **pointwise stabilizer** *of P in G by*

$$\mathrm{Stab}_G(P) := \{g \in G \mid \forall x \in P : gx = x\}$$

Take care to not mix up the pointwise stabilizer with the setwise stabilizer which is defined as $\{g \in G \mid gP = P\}$.

Definition 1.3 (Restriction) *If $G \circlearrowleft S$ and $P \subset S$ such that the action of G on S fixes P setwise, G also defines an action on P. Thus the pointwise stabilizer of P is a normal subgroup of G and we can define the* **restriction of the group action** *as*

$$G \circlearrowleft S|_P := G/\mathrm{Stab}_G(P).$$

Definition 1.4 (Orbit) *Let $G \circlearrowleft S$. For $s \in S$ the set Gs is called* **orbit** *of s under the action of G.*

Remark 1.5 *Being in the same orbit is an equivalence relation. Thus the orbits of the action of G partition the set S.*

Definition 1.6 (transitive, free) *Let $G \circlearrowleft S$.*

1. *The group action is said to be* **transitive**, *if it has only one orbit*

$$\forall s \in S : Gs = S.$$

2. *A group action is said to be* **free**, *if only the identity fixes an element*

$$\forall g \in G, s \in S : g \neq \mathrm{id} \;\Rightarrow\; gs \neq s.$$

Note that free group actions are often called **semiregular** instead.

Theorem 1.7 (orbit–stabilizer) *Let a group G act on a set Ω. For every $x \in \Omega$ the elements of its orbit Gx are in one–to–one correspondence with the cosets of its stabilizer $\mathrm{Stab}_G(x)$. In particular*

$$|Gx| = [G : \mathrm{Stab}_G(x)]$$

→ 77

1.2. Graph Theory

There is an inexhaustible amount of definitions of **graphs**, **digraphs**, **multi–graphs** and so on. Each tries to allow or avoid constructions like **multiple edges**, **loops**, **semi–edges**, **directions** and so on. We just exemplarily give two definitions which should be enough for this thesis. The first allowing only single edges and loops, the second also multiple edges.

Definition 1.8 *A* **digraph** *X is a pair $(V(X), E(X))$ where $V(X)$ is a set (of vertices) and $E(X) \subset V(X) \times V(X)$.*

There are several problems with this definition. To mention just one of them, there cannot be two edges (x, y) but there can be an edge (x, y) and an edge (y, x), thus if one wants to use it to define undirected graphs by forgetting the direction it yields graphs with at most two parallel edges (which can be either a bug or a feature).

Definition 1.9 *A* **digraph** *X is a quadruple $(V(X), E(X), s, t)$ with $V(X)$ and $E(X)$ arbitrary sets (with the exception that $V(X)$ cannot be empty if $E(X)$ is nonempty) and $s, t : E(X) \to V(X)$ are functions, assigning to each edge an* **initial-** *and a* **terminal vertex**.

This sounds like a pretty safe definition of a **multi–digraph**, and does the job for a wide range of applications, but it often leads to very uncomfortable notation. Thus for convenience the notation often pretends that edges are ordered pairs of vertices, even if graphs were defined differently before. As these notational problems are resolvable we will ignore them and jump between the definitions as it suits – unless it could give room for misunderstanding.

Definition 1.10 (bipartite) *A digraph X with*

$$\begin{aligned} V(X) &= V_1 \dot{\cup} V_2 \\ E(X) &\subseteq V_1 \times V_2 \end{aligned}$$

is called **bipartite***.*

Remark 1.11 *The definition above says that all the edges are directed from V_1 to V_2. Thus one can understand V_1 as **source–partition** and V_2 as **sink–partition**. Moreover, note that the underlying undirected graph is bipartite in the undirected sense, but not every **orientation** on a bipartite graph yields a bipartite digraph.*

Definition 1.12 (subgraphs) *Let $D = (V(D), E(D), s, t)$ be a digraph. Let $V(G) \subseteq V(D)$ and $E(G) \subseteq E(D)$ be subsets of vertices and edges of D. If $G = (V(G), E(G), s|_{E(G)}, t|_{E(G)})$ is a digraph, we call it* **subgraph** *of D (that is, if $s(E(G)) \subset V(G)$ and $t(E(G)) \subset V(G)$.)*
We say that a subgraph X is **induced** *if $E(X) = s^{-1}(V(X)) \cup t^{-1}(V(X))$ (that is, if all the possible edges are contained). A set V of vertices* **induces** *the unique induced subgraph X with vertex set $V(X) = V$. A set of edges induces the subgraph that is induced by its covered vertices.*
If S is a set of vertices and/or edges we denote the subgraph it induces by $\langle S \rangle$.

Definition 1.13 (K_n, $K_{n,m}$) *Let N and M be sets, $n = |N|$ and $m = |M|$. We define the* **complete** *digraph K_n as*

$$\begin{aligned} V(K_n) &:= N \\ E(K_n) &:= N \times N \end{aligned}$$

and the **complete bipartite** *digraph $K_{n,m}$ as*

$$\begin{aligned} V(K_{n,m}) &:= N \dot{\cup} M \\ E(K_{n,m}) &:= N \times M \end{aligned}$$

Note that this definition of K_n includes the loops at every vertex.

Definition 1.14 (degree,valency,neighbours) *Let X be a digraph with multiple edges and loops and $x \in V(X)$.*
The **in–valency** *of x is the size of the set of edges terminating in x*

$$\delta^-(x) := |\{e \in E(X) \mid t(e) = x\}|$$

The **out–valency** *of x is the size of the set of edges starting in x*

$$\delta^+(x) := |\{e \in E(X) \mid s(e) = x\}|$$

The **valency** *of x is the sum of its in–valency and the out–valency*

$$\delta(x) := \delta^-(x) + \delta^+(x)$$

The **in–neighbours** *of x are the vertices from which there is an edge to x*

$$N^-(x) := \{y \in V(X) \mid (y, x) \in E(X)\}$$

The **out–neighbours** *of x are the vertices to which there is an edge from x*

$$N^+(x) := \{y \in V(X) \mid (x, y) \in E(X)\}$$

The **neighbours** *of x are the vertices adjacent to x*

$$N(x) := N^+(x) \cup N^-(x)$$

The **in–degree** *is the size of the set of in–neighbours*

$$d^-(x) := \left|N^-(x)\right|$$

The **out–degree** *is the size of the set of out–neighbours*

$$d^+(x) := \left|N^+(x)\right|$$

The **degree** *is the size of the set of neighbours*

$$d(x) := |N(x)|$$

Remark 1.15 *Note that there will always be problems with these definitions. Considering loops, a vertex could be its own in- and out–neighbour. As defined here, a loop would at least add two to the valency of its vertex such that the theorem that the sum of all degrees in a finite graph is even, stays true at least true for the valency. Note also the notational abuse that illustrates the above mentioned use of different definitions at the same time.*

Definition 1.16 (walk, arc, path, ray, line, cycle) *Let X be a digraph.*

1. *A* **walk** *is a sequence of vertices* $(x_0, \ldots, x_n)$ *such that there is an edge between* x_i *and* x_{i+1} *(no matter if it is the edge* (x_i, x_{i+1}) *or* (x_{i+1}, x_i)*). If the length is of importance we write* n–**walk**. *We understand a single vertex as* 0–**walk**.

2. *An* **arc** *is a walk such that all the edges are directed in the same direction* $e_i = (x_i, x_{i+1})$*. We often denote an arc by its corresponding sequence of edges rather then by its sequence of vertices* $((e_0, \ldots, e_{n-1}) = (x_0, \ldots, x_n))$*. We write* n–**arc** *if we want to specify the length and understand a single vertex as* 0–**arc**. *We denote the set of* n*–arcs of* X *by* $\mathrm{Arc}_n(X)$ *or* Arc_n *if there is no room for misunderstanding.*

3. *A* **path** *(or* n*–path) is a walk that does not visit a vertex twice.*

4. *A* **directed path** *is an arc that does not visit a vertex twice.*

5. *A* **closed** *walk or arc is a walk or arc with the property that the initial vertex coincides with the terminal vertex.*

6. *A* **cycle** *is a closed path in the above sense. We have to make the exception that the initial vertex is allowed to be visited a second time when the path finally returns. A cycle is* **balanced** *if it has equally many forward and backward edges (the choice of forward and backward obviously does not matter in that sense).*

7. *A* **ray** *is an infinite sequence of vertices* $(x_0, x_1, \ldots)$ *such that every finite subsequence of neighbouring entries (that is* $(x_{i_j})_j$ *with* $i_{j+1} = i_j + 1$*) is a path.*

8. *A* **double–ray** *is a two way infinite sequence* $(\ldots, x_{-1}, x_0, x_1, \ldots)$ *such that every finite subsequence of neighbouring entries is a path.*

9. *A* **positive half–line** *is an infinite sequence of vertices* $(x_0, x_1, \ldots)$ *such that every finite subsequence of neighbouring entries is a directed path. We speak of a* **negative half–line** *if the reverse such subsequences are directed paths. If it is not necessary to distinguish or the positivity or negativity of the half–line is clear from the context we just speak of* **half–lines**.

10. *A* **line** *is a two way infinite sequence* $(\ldots, x_{-1}, x_0, x_1, \ldots)$ *such that every finite subsequence of neighbouring entries is a directed path. We denote the set of all lines of* X *by* $\mathcal{L}(X)$*.*

Remark 1.17 *The definitions of walks, arcs and paths often differ from author to author. Usually the disagreement is about whether vertices or edges may not be visited twice and how the properties are distributed on the notions.*

Definition/Lemma 1.18 (connected) *A digraph is* **connected** *it there is a walk between any two vertices. Being connected by a walk is obviously an equivalence relation on the vertices. The partitions of this equivalence relation induce connected subgraphs. We call them* **components**. → 77

Definition 1.19 (descendants, predecessors) *Let X be a digraph and $x \in V(X)$. Let $x^{\Rightarrow s}$ be the set of vertices in X in which an s–arc terminates that started in x, that is*

$$x^{\Rightarrow s} := \{y \in V(X) \mid \exists\ s\text{–}arc\ (x, \dots, y)\}.$$

We define the descendants of x as the lot of these vertices:

$$x^{\Rightarrow} := \bigcup_{s \in \mathbb{N}^+} x^{\Rightarrow s}$$

Analogously we define for the predecessors ${}^{s\Rightarrow}x$ and ${}^{\Rightarrow}x$ of x.
Furthermore we define the descendants and predecessors of a set $A \subset V(X)$ of vertices canonically as

$$\begin{aligned} A^{\Rightarrow n} &:= \bigcup_{a \in A} a^{\Rightarrow n} \\ A^{\Rightarrow} &:= \bigcup_{a \in A} a^{\Rightarrow} \\ {}^{n\Rightarrow}A &:= \bigcup_{a \in A} {}^{n\Rightarrow}a \\ {}^{\Rightarrow}A &:= \bigcup_{a \in A} {}^{n\Rightarrow}a. \end{aligned}$$

Definition 1.20 (regular) *A graph X is said to be* **regular**, *if every vertex has the same degree. For a digraph D we ask all the in–degrees to be equal and all the out–degrees to be equal (the in- and out–degree can be different). In that cases we denote the degree with $\mathrm{d}(X)$, the in–degree with $\mathrm{d}^-(D)$ and the out–degree with $\mathrm{d}^+(D)$ or if there is no room for misunderstanding just d, d^- and d^+ respectively. Analogously we denote $\delta(X)$, $\delta^-(D)$ and $\delta^+(D)$ or again δ, δ^+ and δ^- for the valencies.*

Note 1.21 *Most of the graphs we are going to consider agree on degree and valency (i.e. have no loops or multiple edges). Hence we can safely speak of valency but denote $\mathrm{d}^{\cdots}(\dots)$.*

Definition 1.22 (homomorphism) *Let X and Y be digraphs. A* **digraph homomorphisms** *φ sloppily denoted $\varphi : X \to Y$ is a map*

$$\varphi : V(X) \cup E(X) \to V(Y) \cup E(Y)$$

that takes vertices to vertices and edges to edges such that

$$\begin{aligned} (a,b) \in E(X) &\Rightarrow (\varphi(a), \varphi(b)) \in E(Y) \text{ and} \\ \varphi((a,b)) &= (\varphi(a), \varphi(b)). \end{aligned}$$

Definition 1.23 (epimorphism) *A homomorphism is called* **epimorphism** *if it is surjective.*

Definition 1.24 (isomorphism) *A bijective homomorphism is called* **isomorphism** *if its reverse is a homomorphism as well. If there is an isomorphism between two graphs X and Y, we call them* **isomorphic** *and write $X \cong Y$.*

Definition/Lemma 1.25 (automorphism) *An isomorphism $\phi : G \to G$ from a graph G onto itself is called* **automorphism**. *The set of automorphisms together with the composition is a group which we denote by Aut_G.* $\to 77$

Remark 1.26 *Usually one defines graph–homomorphisms on the vertices only. This ends up in problems thinking about what one wants to be an epimorphism and what not. Thus the above three definitions read a bit unfamiliar on first sight.*

We will need **covering projections** in a different way as they are usually defined. First we give the standard definition.

Definition 1.27 (covering projection) *Let X and Y be digraphs and $\phi : X \to Y$ be an epimorphism, such that for all $x \in V(X)$ the restriction $\phi|_{N(x)} : N(x) \to N(\phi(x))$ is an isomorphism. Then X is called* **covering digraph** *of Y and ϕ* **covering projection**.

Remark 1.28 *In [1] the authors define covering projections in a different way. The reason for that is the following: Consider a 2- or 3–cycle, we desire to allow a line to wind around this cycle forever – the standard definition from above allows that only if the cycle–length is greater or equal 4. We thus weaken the requirements by splitting $N(x)$ into $N^-(x)$ and $N^+(x)$. That results in the definition below which we are going to use.*

Definition 1.29 (covering projection) *Let X and Y be digraphs and $\phi : X \to Y$ be an epimorphism, such that for all $x \in V(X)$ the restrictions $\phi|_{N^-(x)} : N^-(x) \to N^-(\phi(x))$ and $\phi|_{N^+(x)} : N^+(x) \to N^+(\phi(x))$ are isomorphisms. Then X is called* **covering digraph** *of Y and ϕ* **covering projection**.

Definition 1.30 (end) *An* **end** *is an equivalence class of rays under the equivalence relation that two rays are equivalent, if there is a third ray that meets both in infinitely many vertices.*

Remark 1.31 (end) *There are different equivalent definitions of ends. We mention two more*

1. *Two rays are considered equivalent if there are infinitely many disjoint walks connecting them.*

2. *Two rays are considered equivalent if there is no finite set of vertices that separates them.*

Definition 1.32 (thin, thick) *If an end contains only finitely many disjoint rays it is called* **thin**. *Otherwise we call it* **thick**.

Cayley–graphs are visualizations of groups respecting the chosen generators. A group can have different (non–isomorphic) Cayley–graphs. Sometimes it is convenient to choose for every used generator also the inverse in order to end up with an undirected graph.

Definition 1.33 (Cayley–graph) *Given a group G generated by the elements $g_1, \dots, g_n \in G$. The Cayley–graph* $\mathrm{Cay}(G, \{g_1, \dots, g_n\})$ *is defined by*

$$\begin{aligned} V(\mathrm{Cay}(G, \{g_1, \dots, g_n\})) &= G \\ E(\mathrm{Cay}(G, \{g_1, \dots, g_n\})) &= \{(g,h) \in G \times G \mid \exists i \in [n] : gg_i = h\} \end{aligned}$$

One can decide whether a given graph is a Cayley–graph using the **closed path property**.

Proposition 1.34 *A graph G is a Cayley–graph if and only if there is an edge-colouring $c : E(X) \to C$ (where C is a set of colours) such that*

1. *Every vertex is incident to exactly one incoming and outgoing edge of each colour.*

2. *(***closed path property***) If a walk $w = (e_1, \dots, e_n)$ starting at a vertex $x \in V(G)$ returns to x after n steps, the walks $w^y = (e_1^y, \dots, e_n^y)$ starting at an arbitrary vertex $y \in V(G)$ and agreeing on colour and direction with w ($c(e_i) = c(e_i^y)$ and e_i^y is a forward edge exactly if e_i is) return to y after n steps.* $\to$ 77

Definition 1.35 (Group action on a graph) *We say that a group G* **acts on a graph** *X if there is a homomorphism $\varphi : g \to \mathrm{Aut}X$. We write $G \circlearrowleft X$.*

Definition 1.36 (free) *A group G acts* **free** *on a graph X if its action $G \circlearrowleft X$ restricted to $V(X)$ is free as a group action on a set.*

Definition 1.37 (transitive) *Let $G \circlearrowleft X$ be a group action on a graph. It is* **transitive** *if the induced group action of a set on $V(X)$ is transitive. The graph X is said to be* **transitive** *if the action $\mathrm{Aut}X \circlearrowleft X$ is transitive.*

A famous theorem about Cayley–graphs was introduced by *Sabidussi*.

Theorem 1.38 (Sabidussi) *A graph X is a Cayley–graph of a group G if and only if there is an action $G \circlearrowleft X$ that is free and transitive.* $\rightarrow 77$

Definition 1.39 (arc transitive) *Let $G \circlearrowleft X$ be a group action on a digraph. It is* **arc transitive** *if the action it induces on $E(X)$ is transitive. The digraph X is said to be* **arc transitive** *if the action $\mathrm{Aut}X \circlearrowleft X$ is arc transitive.*

Definition 1.40 (s–arc transitive) *Let $G \circlearrowleft X$ be a group action on a digraph. It is* s–**arc transitive** *if the action it induces on Arc_s is transitive and Arc_s is not empty. The digraph X is said to be* s–**arc transitive** *if the action $\mathrm{Aut}X \circlearrowleft X$ is s–arc transitive.*

Definition 1.41 (highly arc transitive) *Let $G \circlearrowleft X$ be a group action on a digraph. It is* **highly arc transitive** *if it is s–arc transitive for all $s \in \mathbb{N}^+$. The digraph X is said to be* **highly arc transitive** *if the action $\mathrm{Aut}X \circlearrowleft X$ is highly arc transitive.*

Remark 1.42 *The title of [1] (Infinite highly arc transitive digraphs and ...) suggests that one could think also about finite, highly arc transitive digraphs, but indeed, the only connected, finite digraphs which are highly arc transitive are directed cycles. For suppose G is finite and highly arc transitive. If it does not contain a directed cycle it does not contain an arc of length $|V(G)| + 1$ and thus is not highly arc transitive. Thus it contains a directed cycle C_1. Now suppose it has an edge that is not an edge of the cycle. If the edge connects two vertices of the cycle, G contains a shorter directed cycle, what immediately contradicts the highly arc transitivity. Thus there must be a vertex x in the cycle that has either an additional in- or out–edge that starts/ends outside the cycle (other would contradict the connectedness). Because of highly arc transitivity this edge must lie on a second directed cycle C_2 of the same length that differs from C_1 at least on the vertex following x. Thus a map that takes the arc (C_1, C_2) to the arc (C_1, C_1) is not injective, thus not an automorphism contradicting the highly arc transitivity. Thus G cannot contain an edge other than the edges of C_1. Since it is connected it has also no vertices outside C_1. Thus it is C_1.*

Remark 1.43 *The condition $\mathrm{Arc}_s \neq \emptyset$ in Definition 1.40 actually is important. Otherwise i.e. the $K_{n,n}$ with a source- and a sink–partition would be highly arc transitive because the condition would be empty satisfied for $s > 1$. But actually we do not want $K_{n,n}$ to be highly arc transitive.*

Remember that arcs unlike paths may use the same edge multiple times. Thus any digraph that contains a directed cycle contains arcs of arbitrary length.
Before we now turn to the examples and constructions section we have a short look at some examples of s–arc transitivity.

Example 1.44

1. *The digraph in Figure 1 is* 1*–arc transitive but not* 2*–arc transitive, as the red* 2*–arc cannot be mapped to the green* 2*–arc by an automorphism.*

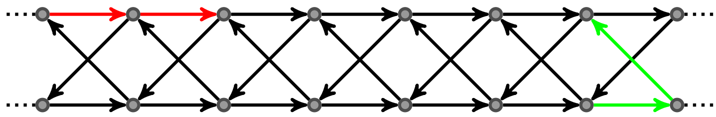

Figure 1: two way infinite, 1–arc transitive digraph

2. *The digraph in Figure 2 is* 2*–arc transitive but not* 1*–arc transitive. Obviously the only two* 2*–arcs can be mapped to each other by the only nontrivial automorphism, but the inner and outer edges cannot be exchanged.*

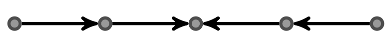

Figure 2: evil finite digraph

3. *The* K_4 *in Figure 3 is* 1*–path transitive. Thinking on the tetrahedron it is obvious, that it is also* 2*–path transitive, as any* 2*–path can be thought of as an angle of one of the triangles. Then already the group of rotational symmetries (the* A_4*) acts transitively on the angles. Thus the full symmetry group (the* S_4*) acts transitively on the* 2*–paths as it can flip the angles. However, there are non–returning* 3*–paths which cannot be mapped to the triangles, thus the* K_4 *is not* 3*–path transitive.*

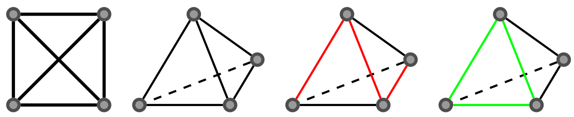

Figure 3: K_4 - undirected

Thus the different transitivities are not just specializations of each other, but really different properties. A lot of work was done on transitivity of finite graphs. We will not look closer on these but turn now to highly arc transitive digraphs. Nevertheless we keep in mind that bad things like in Figure 2 may occur.

Remark 1.45 *Finally we remark that infinite, connected, transitive graphs have either* 1*,* 2 *or infinitely many ends. There exist a couple of different versions of this well known theorem. It holds under much weaker assumptions and can give more information about the set of ends. Here it is just mentioned because it obviously holds for highly arc transitive digraphs as well. Thus we can keep in mind that they always have* 1*,* 2 *or infinitely many ends.*

2. Examples and Constructions

2.1. The line Z

The Cayley–graph of $(\mathbb{Z}, +)$ with the generator 1 is a highly arc transitive digraph. We define it as the integer line and draw it from left to right.

Definition 2.1 (Integer line) *The* **integer line** *is the digraph Z given by*

$$\begin{aligned} V(Z) &:= \mathbb{Z} \\ E(Z) &:= \{(x, x+1) \mid x \in \mathbb{Z}\}. \end{aligned}$$

Proposition 2.2 *Z is highly arc transitive.* → 62

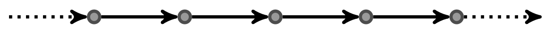

Figure 4: The integer line Z

2.2. Trees

The second obvious example for highly arc transitive digraphs are regular directed trees.

Definition 2.3 (Tree) *A tree is a graph/digraph without cycles.*

Figure 5 shows a tree with in–valency 1 and out–valency 2 on the left side and a tree with in–valency 2 and out–valency 3 on the right side.

Proposition 2.4 *A regular tree T is highly arc transitive.* → 62

2.3. Z–like digraphs

One can understand Z as infinitely many $K_{1,1}$s glued together to a line. Following this idea we construct Z–like digraphs.

2.3.1. $K_{n,n}$–lines

Construction 2.5 *We construct a two way infinite line of $K_{n,n}$s by gluing together infinitely many $K_{n,n}$s in the canonical way. As vertex set of the new digraph $LK_{n,n}$ we use*

$$V(LK_{n,n}) := \mathbb{Z} \times \mathbb{Z}_n.$$

Then we draw $K_{n,n}$ between the layers, thus the edge set is

$$E(LK_{n,n}) := \{((k, x), (k+1, y)) \mid k \in \mathbb{Z}, x, y \in \mathbb{Z}_n\}.$$

Definition 2.6 *We call the digraph $LK_{n,n}$ from Construction 2.5 the $K_{n,n}$–line.*

Proposition 2.7 *For every positive integer $n \in \mathbb{N}^+$ the $K_{n,n}$–line is highly arc transitive.* → 62

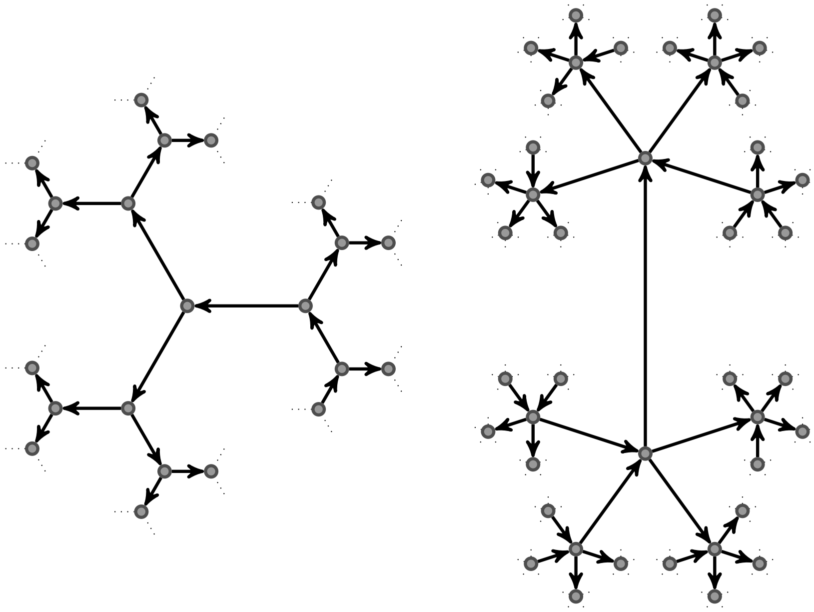

Figure 5: regular trees with $d^- = 1$, $d^+ = 2$ and $d^- = 2$, $d^+ = 3$

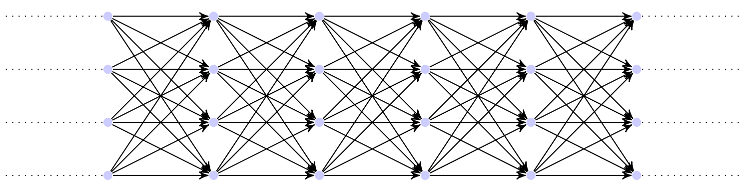

Figure 6: $K_{4,4}$–line

2.3.2. $K_{n,n}$–tubes

We are going to give an infinite class of highly arc transitive digraphs that was first considered by *McKay* and *Praeger* and mentioned in [1] but seems to have been forgotten afterwards. A special case was rediscovered in [9]. The author generalized this idea to regain the original class.
The following construction uses **voltage assignments** and **derived graphs** following the approach from [9]. Therefore we first need to define **voltage graphs**.

Definition 2.8 (voltage assignment) *Let X be a digraph and G be a group. A* **voltage assignment** *is a function $\alpha : E(X) \to G$ (i.e. every edge of X is coloured with a group element $g \in G$).*

Definition 2.9 (derived graph) *Let X be a digraph and $\alpha : E(X) \to \mathbb{Z}_n$ (for $n \in \mathbb{N}^+$) a voltage assignment. We define the* **derived digraph** $\tilde{X}$ *by*

$$\begin{aligned} V(\tilde{X}) &:= V(X) \times \mathbb{Z}_n \\ E(\tilde{X}) &:= \{((v_0, k), (v_1, k + \alpha((v_0, v_1)))) \mid (v_0, v_1) \in E(X)\} \end{aligned}$$

That is we replace every vertex with n new vertices putting labels 0 to $n-1$ on them. Then wherever there is an edge in X, we put n edges between the corresponding sets of new edges. The voltage assignment tells us, to which label we put the edge starting at label 0. We add the other edges in cyclic order.

Construction 2.10 ($K_{n,n}$–tube) *Let $n, m \in \mathbb{N}$ be integers. Consider the group of voltages $\mathbb{Z}_n^m$ and the graph Z. Replace every edge in Z by n edges, all having the same initial and terminal vertex as the replaced edge had. Now we assign voltages to every edge in the following way. The edges between the vertices x and $x+1$ get the voltages 0, the $x \bmod m$–th standard base vector from $\mathbb{Z}_n^m$ and all its multiples (as illustrated in Figure 7). Finally we consider the derived graph from this voltage assignment, denoting it by* $\mathrm{Tube}(n, m)$*.*

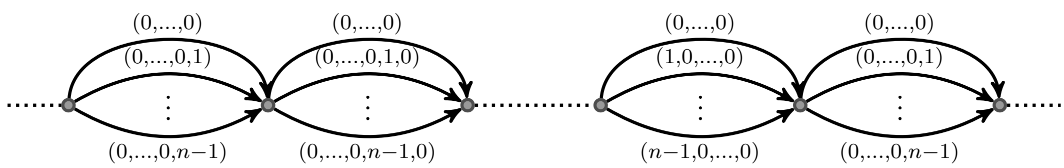

Figure 7: Voltage assignment

Definition 2.11 ($K_{n,n}$–tube) *The digraph* $\mathrm{Tube}(n, m)$ *from Construction 2.10 is called m–***periodic** $K_{n,n}$**–tube***.*

Remark 2.12 *One can understand the $K_{n,n}$–tube in the following way. In every layer there are n^m vertices which can be labeled by the n^m elements of $\mathbb{Z}_n^m$ (i.e. by the at most m–digited numbers of base n). Edges only run between neighbouring layers (from layer x to layer $x+1$). There they run precisely between vertices whose labels differ in at most the $x \bmod m$–th digit. This idea is illustrated in the Figures 8, 9 and 10. It results in n^{m-1} $K_{n,n}$s which lie "parallel" in every layer, that motivated the name $K_{n,n}$–tube.*
McKay and Praeger described the same thing with labels from the m–th power of an arbitrary n–sized set and put edges whenever the label in the next layer was a right shift of the current one.

Proposition 2.13 *For $n, m \in \mathbb{N}^+$ the* Tube(n, m) *is highly arc transitive.* $\rightarrow$ 63

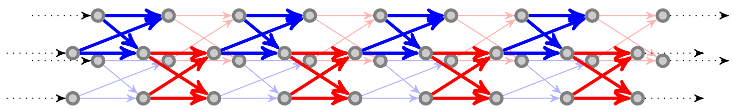

Figure 8: Tube$(2, 2)$

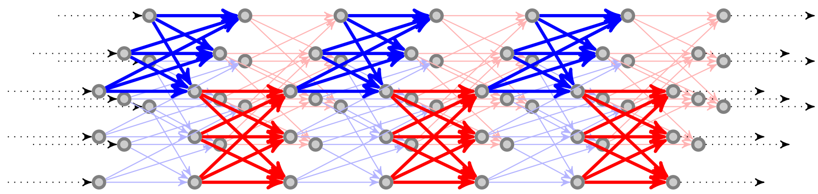

Figure 9: Tube$(3, 2)$

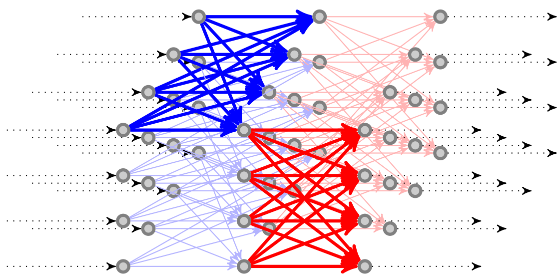

Figure 10: Tube$(4, 2)$

2.4. Line digraphs

First, we recall the definition of the linegraph.

Definition 2.14 (Line digraph) *Given a digraph X, the* **line digraph** *L is defined by*

$$\begin{aligned} V(L) &:= E(X) \\ E(L) &:= \{(e_1, e_2) \mid \exists x : e_1 = (\cdot, x) \wedge e_2 = (x, \cdot)\} \end{aligned}$$

Example 2.15 *The line digraph of a regular tree with in- and out–valency 2 consists of $K_{2,2}$s. Figure 11 shows this line digraph in blue and the underlying tree in gray.*

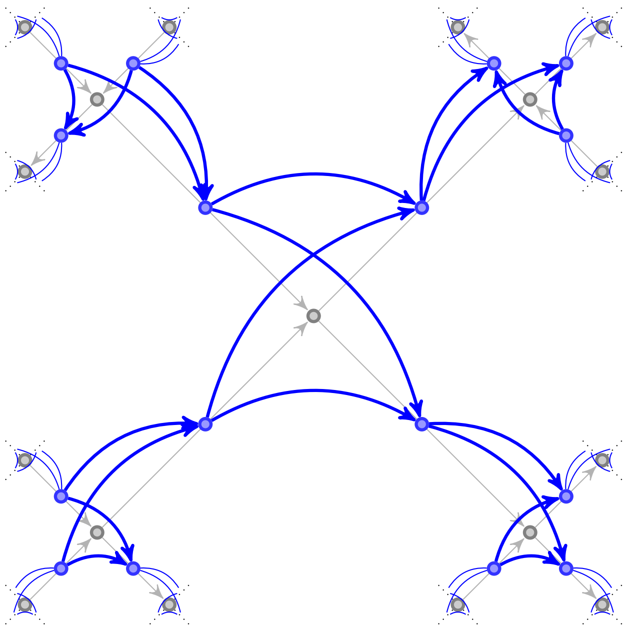

Figure 11: Linegraph of the 2–in–2–out–regular tree

Proposition 2.16 *If a digraph D is highly arc transitive and connected then so is its line digraph.* → 63

We will encounter some more constructions alike. Particularly in the next subsection we provide a very important example of a subgraph of a line digraph.

2.5. Universal covering digraphs

We now come to the most important graph from [1]. We are going to construct a digraph $DL(\Delta)$ from a bipartite digraph Δ and a tree T (which depends on Δ).

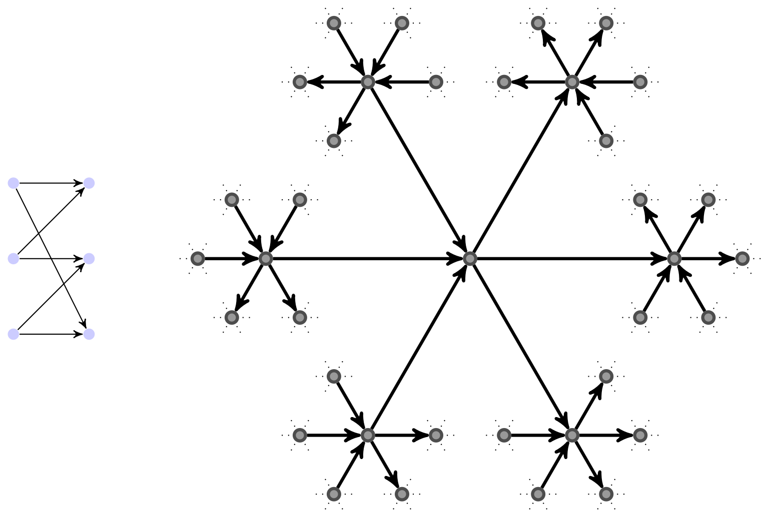

Figure 12: A bipartite digraph Δ and its tree T

Construction 2.17 ($DL(\Delta)$) *We start with an arbitrary bipartite digraph Δ consisting of the source partition Δ^- and the sink partition Δ^+. Let $n = |\Delta^-|$ and $m = |\Delta^+|$. We consider the regular tree T with in–valency $d^-(T) = n$ and out–valency $d^+(T) = m$.*

Now we consider the line digraph of T and define an appropriate subgraph.

Wherever a vertex v of T has been, there appears a $K_{n,m}$ in the line digraph. We replace all these $K_{n,m}$s by copies of Δ. Therefore we need to choose which edges to drop, or more intuitively, which to take:
For $v \in T$ let v^- be the set of in–edges of v and v^+ the set of out–edges of v. We choose bijections $\phi_v^+ : \Delta^+ \to v^+$ and $\phi_v^- : \Delta^- \to v^-$. Now we choose for every edge in $(x, y) \in \Delta$ and every $v \in T$ the edge $(\phi_v(x), \phi_v(y))$ to be in $E(DL(\Delta))$.

Definition 2.18 (Universal covering digraph) *The digraph $DL(\Delta)$ defined in Construction 2.17 with*

$$\begin{aligned} V(DL(\Delta)) &:= E(T) \\ E(DL(\Delta)) &:= \bigcup_{v \in T} \{(\phi_v^+(x), \phi_v^-(y)) | x = (\cdot, v), y = (v, \cdot)\} \end{aligned}$$

is called **universal covering digraph** *of Δ, if Δ is connected and 1–arc transitive.*

Proposition 2.19

(1) The structure of a universal covering digraph does not depend on the choices of ϕ_b^a.

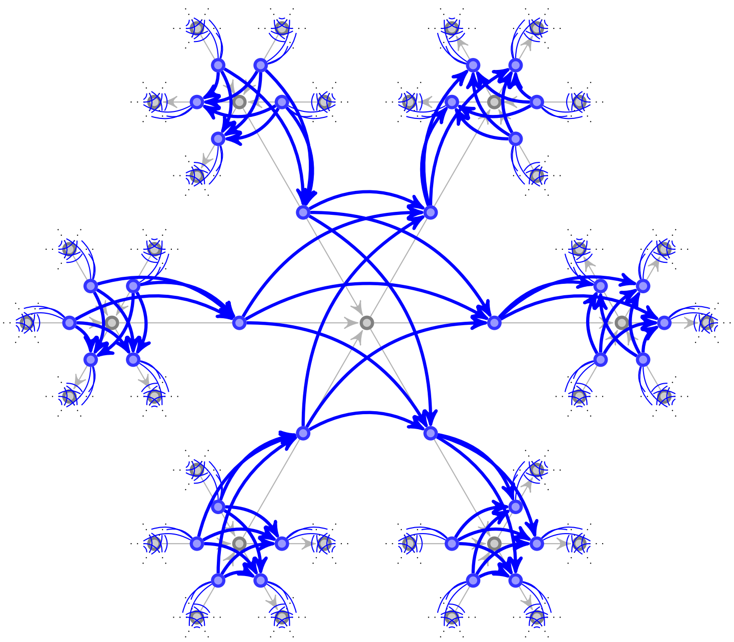

Figure 13: The line digraph of T

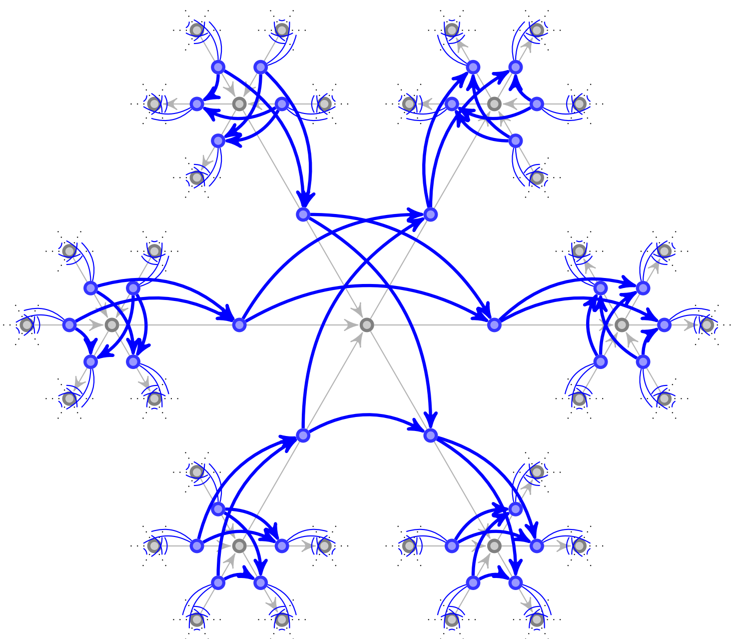

Figure 14: $DL(\Delta)$

(2) Universal covering digraphs are highly arc transitive. → 63

2.6. Ordered field digraphs

A quite boring example that illustrates that highly arc transitive digraphs can be pretty structureless, is an order digraph of an **ordered field**.

Definition 2.20 (ordered field) *An* **ordered field** $(F, +, \cdot, \leq)$ *is gained from a field* $(F, +, \cdot)$ *by adding an order relation* $\leq$ *such that* $(F, \leq)$ *is totally ordered and*

1. $\forall a, b, c \in F : a \leq b \;\Rightarrow\; a + c \leq b + c$

2. $\forall a, b \in F : 0 \leq a \wedge 0 \leq b \;\Rightarrow\; 0 \leq ab$

Proposition 2.21 *The ordered field digraph D of any ordered field* $(F, \leq)$

$$\begin{aligned} V(D) &:= F \\ E(D) &:= \{(x, y) \in F \times F \mid x \leq y\} \end{aligned}$$

is highly arc transitive. → 64

Remark 2.22 *Note that ordered fields have characteristic* 0 *and thus are infinite.*

2.7. The alternating–cycle digraph

There are different ways of defining the alternating–cycle digraph. For the present thesis it will be sufficient to define it as a Cayley–graph of a certain group.

Definition 2.23 (alternating–cycle digraph) *For* $n \geq 3$ *we define the* **alternating–cycle digraph** *as*

$$\mathrm{AC}(n) := \mathrm{Cay}(\langle L, R \mid (RL^{-1})^n, ((RL^{-1})^{\frac{n-1}{2}} R)^2 \rangle, \{L, R\})$$

Proposition 2.24 *Let* $n \geq 3$ *be an odd integer. Then the graph* $\mathrm{AC}(n)$ *is highly arc transitive.* → 64

Figure 15 shows the alternating–cycle digraph for $n = 5$.

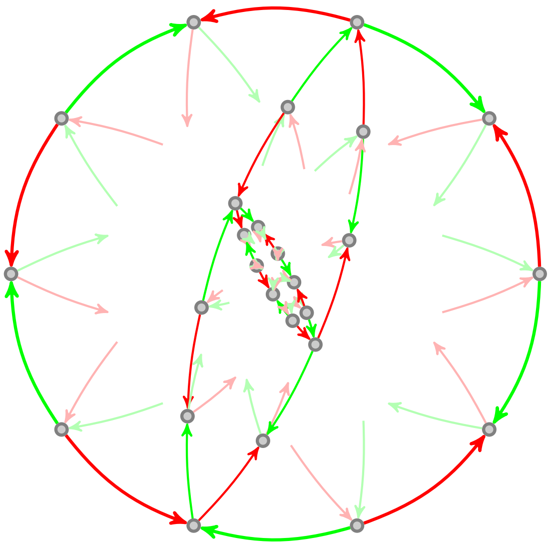

Figure 15: AC(5)

2.8. The Evans–graph

The Evans–graph was introduced in [6]. *Evans* just called it "an infinite highly arc–transitive digraph". It is constructed from countably many trees. In order to do so we will need the definition of independent sets.

Definition 2.25 (Independent set) *Let D be a digraph. A set $\{d_1, \ldots, d_n\} \subset V(D)$ is said to be* **independent** *if no $v \in V(D)$ is a descendant of more than one d_i.*

Construction 2.26 (Evans–graph) *We start with a tree T with constant out–valency n. We are going to construct a chain of digraphs $X_0 \subset X_1 \subset \ldots$ whose limit*

$$X = \bigcup_{i \in \mathbb{N}} X_i$$

will be our desired digraph.
First we set

$$X_0 = T.$$

In the i–th step we construct X_i from X_{i-1} by attaching a copy T_i of T in a certain way. Namely, we will identify an independent set and all its descendants in T_i with an independent set and all its descendants in X_{i-1}. Therefore we need to specify an order of attaching.
Indeed, we will only need finite independent sets. There are only countably many such in T. Thus there are only countably many independent sets in every X_i (by induction). Thus there are only countably many independent sets in X (because

there are only countably many X_is). Let I be this set and $\varphi : I \to \mathbb{N}$ an enumeration which exists because I is countable.
Now there are countably many injective mappings $\phi : \varphi^{-1}(m) \to T$ which take an arbitrary independent set $\varphi^{-1}(m)$ to an independent set in T. Thus the set

$$\Phi{:=}\{\phi : \varphi^{-1}(m) \to T | m \in \mathbb{N}, \mathrm{Im}\ (\phi)\ \textit{independent}\}$$

is countable and we can choose an enumeration

$$\nu : \Phi \to \mathbb{N}$$

in a way that the domain of $\nu^{-1}(j)$ is contained in X_{i-1} with $0 < i \leq j$. This condition ensures, that during the construction the independent set we choose lies in a digraph that already exists. If it was violated, we would try to attach a copy of T to say X_{17} having constructed only say X_{12}. The existence of such a ν can easily be guaranteed because there are already infinitely many independent sets in $X_0 = T$.
Now we finish the construction with the identification of the domain and the image of $\nu^{-1}(i)$ by $\nu^{-1}(i)$ in the i–th step. The descendants of the independent elements in the attached copy can be identified with the descendants in X_{i-1} in the canonical way (or they can just be deleted as they will disappear either way).

Definition 2.27 (Evans–graph) *The digraph X defined in Construction 2.26 is called* **Evans–graph** *of out–valency n.*

Proposition 2.28 *The Evans–graph of any out–valency $n \in \mathbb{N}^+$ is highly arc transitive.* $\to 64$

Remark 2.29 *For $n = 1$ the Evans–graph is the tree with out–valency 1 and countably infinite in–valency, thus it is not very interesting. But it is still interesting to note that its in–valency is not 2. This is what one could assume as there is only one kind of independent sets in Z (namely a single vertex). But still there are infinitely many vertices in Z each of which is an independent set.*

2.9. k–arc–digraphs

Given an highly arc transitive digraph we construct a new one by choosing k–arcs as edges.

Construction 2.30 *Let D be a highly arc transitive digraph. We construct D_k with the same vertex set as D. We choose the edge set*

$$E(D_k){:=}\{(x, y) \in V(D_k)^2 \mid \ \textit{There is a } k\textit{–arc in } D \textit{ from } x \textit{ to } y\}$$

Proposition 2.31 *D_k from Construction 2.30 is highly arc transitive.* $\to 64$

Remark 2.32 *D_k may not be connected, even if D was connected. I.e. the k–arc–digraph of Z consists of k copies of Z. Thus this construction does not necessarily end up with a new digraph.*

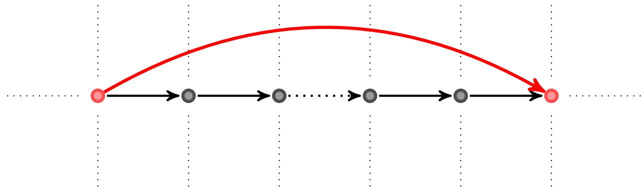

Figure 16: An edge of D_k

2.10. s–arc–k–arc–digraphs

Again we construct a highly arc transitive digraph from a given one.

Construction 2.33 *Let D be a highly arc transitive digraph. For $k \geq 2s$ and $s \geq 1$ we construct a digraph $D_{s,k}$ which has the set of s–arcs of D as vertex set.*

$$V(D_{s,k}):=\{(d_0, \ldots, d_s) \mid (d_i, d_{i+1}) \in E(D)\}$$

We put an edge for every k–arc from the initial s–arc to the terminal s–arc.

$$\begin{aligned} E(D_{s,k}) \;\; := \;\; & \{((a_0, \ldots, a_s), (b_0, \ldots, b_s)) \subset V(D_{s,k})^2 \mid \\ & \exists (c_0, \ldots, c_k) : (c_i, c_{i+1}) \in E(D), a_j = c_j, b_l = c_{k-s+l}\} \end{aligned}$$

wherever i, j and l make sense.

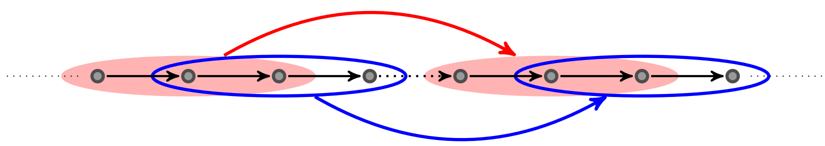

Figure 17: Edges of $D_{s,k}$

Proposition 2.34 *$D_{s,k}$ from Construction 2.33 is highly arc transitive.* $\rightarrow$ 64

Remark 2.35 *$D_{s,k}$ may not be connected, even if D was connected.*

2.11. The DeVos–Mohar–Šámal–digraph

Matt DeVos, *Bojan Mohar* and *Robert Šámal* constructed in [5] an interesting family of highly arc transitive digraphs which together with [2] answers a question from [1] best possible. For every integer product d it gives a digraph with $d^+ = d^- = d$ with a certain property.

Construction 2.36 (DeVos–Mohar–Šámal–digraph) *Let $n, m \geq 3$ be integers. We first construct an undirected tree T. Trees are bipartite. We call one partition A and the other B and choose the $a \in A$ to be n–valent and the $b \in B$ to be m–valent. Now we construct the digraph $\mathrm{DMS}(n, m)$ with vertex set*

$$V(\mathrm{DMS}(n, m)) := E(T)$$

and use a set of paths as edge set. Namely, we use the set of 3–paths of T with initial vertex in A. We understand the 3–path (e_1, e_2, e_3) in T as edge (e_1, e_3) in $\mathrm{DMS}(n, m)$ (as Figure 18 illustrates).

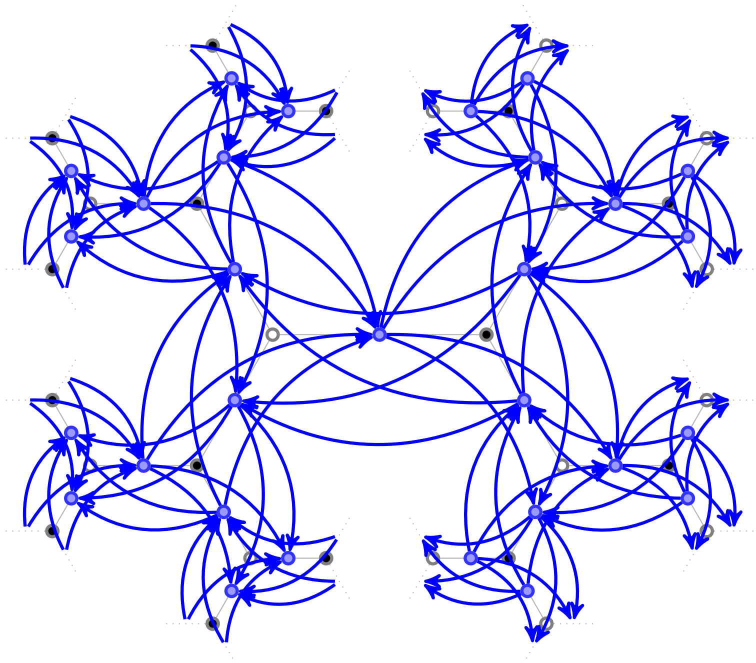

Figure 18: $\mathrm{DMS}(3, 3)$

Definition 2.37 *The digraph $\mathrm{DMS}(n, m)$ defined in Construction 2.36 is called* **DeVos–Mohar–Šámal–digraph***.*

Proposition 2.38 *Let $n, m \geq 3$ be integers, then the digraph $\mathrm{DMS}(n, m)$ is highly arc transitive.* → 64

2.12. The Hamann–Hundertmark–digraph

While characterizing C–homogeneous digraphs with more then one end *Matthias Hamann* and *Fabian Hundertmark* came up with a class of graphs which turned out to be highly arc transitive.

Construction 2.39 (Hamann–Hundertmark) *Let $n \geq 2$ be an integer and $\kappa \geq 3$ a cardinal. We start with a bipartite, undirected tree $T_{\kappa,n}$ with natural bipartition $V(T_{\kappa,n}) = A \cup B$. Let the vertices in A have valency n and the vertices in B valency κ. We construct a graph* HH *that has the edge set of $T_{\kappa,n}$ as vertex set*

$$V(\mathrm{HH}) := E(T_{\kappa,n}).$$

For every $a \in A$ we bijectively assign values from $\mathbb{Z}_n$ to its incident edges. This defines an edge–colouring $c : E(T_{\kappa,n}) \to \mathbb{Z}_n$ and is well defined since every edge e in $E(T_{\kappa,n})$ is incident to a unique vertex $a_e \in A$ (i.e. these **edge–neighbourhoods** *partition the edge set $E(T_{\kappa,n})$). As edge set for* HH *we use a set of paths. Namely, we use the set of 3–paths (e_1, e_2, e_3) with initial vertex in A and $c(e_2) + 1 = c(e_3)$. We understand the arc (e_1, e_2, e_3) in $T_{\kappa,n}$ as edge (e_1, e_3) in* HH *(as Figure 19 illustrates).*

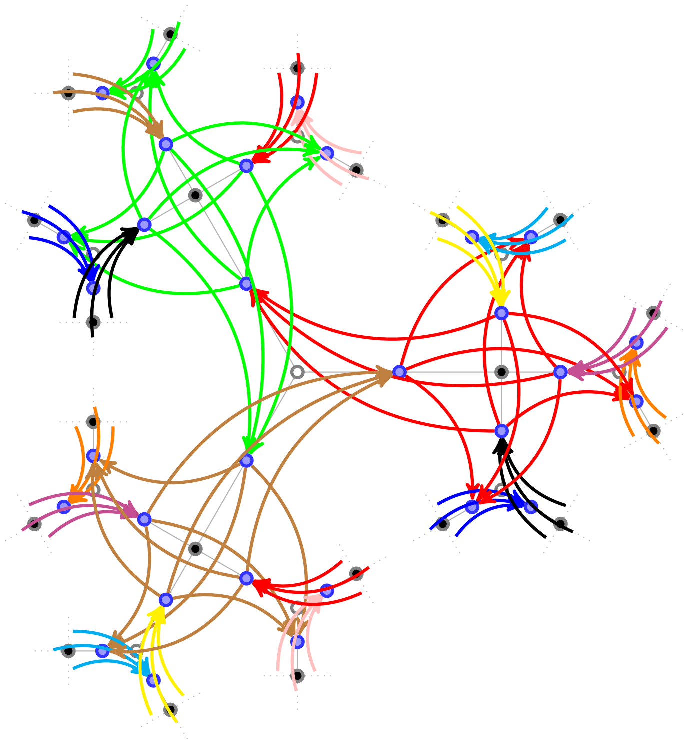

Figure 19: $\mathrm{HH}(4, 3)$

Definition 2.40 (Hamann–Hundertmark–digraph) *Let $n \geq 2$ be an integer and $\kappa \geq 3$ a cardinal. The digraph* $\mathrm{HH}(\kappa, n)$ *from Construction 2.39 is called Hamann–Hundertmark–digraph.*

Proposition 2.41 *Let $n \geq 2$ be an integer and $\kappa \geq 3$ a cardinal. The digraph* $\mathrm{HH}(\kappa, n)$ *from Construction 2.39 is highly arc transitive.* $\to$ 64

2.13. Tensor–products

We will define the tensor–product (also called conjunction) of two digraphs and see, that the tensor–product of two highly arc transitive digraphs is again highly arc transitive.

Definition 2.42 (Tensor–product) *Given to digraphs C and D, we define the* **tensor product** $C \otimes D$ *by*

$$\begin{aligned} V(C \otimes D) &:= V(C) \times V(D) \\ E(C \otimes D) &:= \{((c_1, d_1), (c_2, d_2)) \mid (c_1, c_2) \in E(C), (d_1, d_2) \in E(D)\} \end{aligned}$$

Proposition 2.43 *If C and D are highly arc transitive, so is $C \otimes D$. Indeed if C and D are s–arc transitive, so is $C \otimes D$.* → 64

It is possible to weaken the requirements on D in the above proposition if C has an additional property, see Proposition 4.23.

Proposition 2.44 *Let C be highly arc transitive and n be the cardinality of any nonempty set. Then $C \otimes K_n$ is highly arc transitive.* → 64

Remark 2.45

1. *The name tensor product comes from the adjacency matrix of $C \otimes D$ which is the tensor or Kronecker product of the adjacency matrices of C and D.*
2. *Obviously $C \otimes D$ and $D \otimes C$ are isomorphic (that is by the symmetry of the definition).*
3. *[1] claims that $C \otimes D$ is connected if and only if both C and D are. This is wrong as example 2.46 illustrates.*

Example 2.46 *Figure 20 shows the local view of the tensor–product $LK_{2,2} \otimes LK_{2,2}$. Indeed, the tensor–product is a set of infinitely many copies of $LK_{4,4}$.*

2.13.1. Sequences digraphs

Now we come to a more involved construction that was introduced in [1]. We use two way infinite sequences as vertices and put edges in between them if a certain kind of shift transforms the one into the other.

Construction 2.47 *We start with a connected bipartite digraph Δ with source partition Δ^- and sink partition Δ^+. We choose fixed elements $\delta_1 \in \Delta^-$ and $\delta_2 \in \Delta^+$. Now we define the digraph $S(\Delta, \delta_1, \delta_2)$. First we define the vertex set to be the set of two way infinite sequences $x = (\ldots, x_{-1}, x_0, x_1, \ldots)$ with*

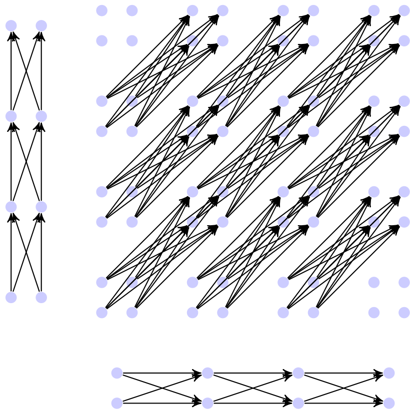

Figure 20: $LK_{2,2} \otimes LK_{2,2}$

1. *$n < 0$: $x_n \in \Delta_1$ but almost all $x_n = \delta_1$*
2. *$n \geq 0$: $x_n \in \Delta_2$ but almost all $x_n = \delta_2$*

The edge set of $S(\Delta, \delta_1, \delta_2)$ is defined by a right–shift, that respects Δ at index 0

$$E(S(\Delta, \delta_1, \delta_2)) := \{(x, y) \mid \forall i \neq 0 : x_{i-1} = y_i \text{ and } (x_{-1}, y_0) \in E(\Delta)\}$$

Definition 2.48 (Sequences digraph) *The digraph $S(\Delta, \delta_1, \delta_2)$ from construction 2.47 is called* **sequences digraph** *of the bipartite digraph Δ with respect to δ_1 and δ_2.*

Proposition 2.49 *Let $\Delta = \Delta^- \cup \Delta^+$ be a bipartite, connected, 1–arc transitive digraph with $\delta_1 \in \Delta^-$ and $\delta_2 \in \Delta^+$ then $Z \otimes S(\Delta, \delta_1, \delta_2)$ is highly arc transitive and connected.* $\to$ 64

Remark 2.50

1. *If Δ contains the edge (δ_1, δ_2) the sequences digraph $S(\Delta, \delta_1, \delta_2)$ with the "zero–sequence" $\mathbf{0} := (\ldots, \delta_1, \delta_2, \ldots)$ contains the loop $(\mathbf{0}, \mathbf{0})$. Unfortunately this loop is not guaranteed what makes the proof of the above proposition a little more involved.*
2. *The sequences digraph itself is far away from being highly arc transitive (it is not even transitive).*
3. *In the above proposition the result remains true if instead of Z a highly arc transitive digraph with an epimorphism onto Z is used. See Theorem 4.20.*

Example 2.51 *As it is actually not all that easy to understand what is going on in this sequences digraph construction, we will have a look at the local view of four small examples starting with the trivial ones*

1. *If we start with* $\Delta = K_{1,1}$*, there is only one possible sequence. Thus the sequences digraph is just a single loop* L*. But* $Z \otimes L = Z$*.*

2. *The first nontrivial example with* $\Delta = K_{1,2}$ *yields a result which is surprising on first sight. The sequences digraph* $S(K_{1,2}, 0, 0)$ *shown in Figure 21 does not look too promising to get a highly arc transitive digraph out of it. But as Figure 22 demonstrates, the digraph* $Z \otimes S(K_{1,2}, 0, 0)$ *is the infinite tree with in–valency* 1 *and out–valency* 2*.*

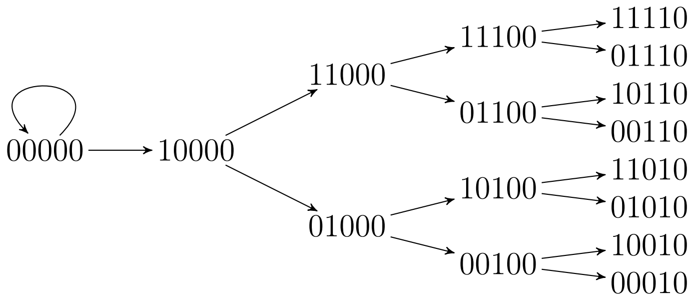

Figure 21: $S(K_{1,2}, 0, 0)$

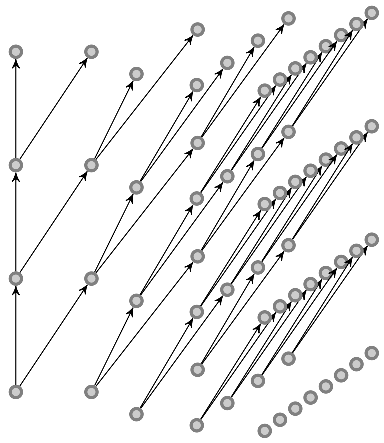

Figure 22: $Z \otimes S(K_{1,2}, 0, 0)$

3. *Once we have more than one vertex in both partitions of* Δ*, the resulting graphs get much more involved. First we have a look at the situation in the case* $\Delta = K_{2,2}$*. If we do not move more than three steps away from the zero–sequence this example has a very beautiful and symmetric embedding that on first sight might suggest that such graphs are easy to draw, but that hope vanishes immediately when considering any larger views. Again Figure 23 shows the local view of the sequences digraph* $S(K_{2,2}, 0, 0)$ *and Figure 24 shows the arising highly arc transitive digraph. It consists of* $K_{2,2}$*s which for easier detection are coloured.*

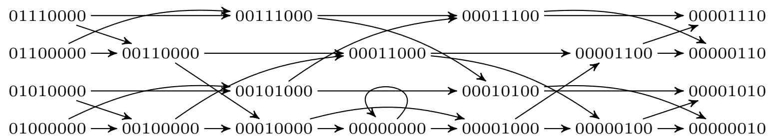

Figure 23: $S(K_{2,2},0,0)$

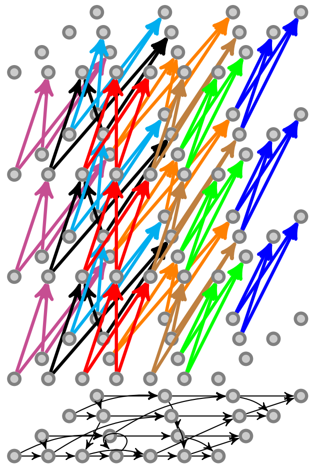

Figure 24: $Z \otimes S(K_{2,2},0,0)$

4. *Finally, we draw* $Z \otimes S(K_{2,3},0,0)$ *which is the smallest nontrivial, asymmetric example. Here, it is already nontrivial to arrange the vertices of the sequences digraph in a way to not immediately lose the overview. Thus Figure 25 only looks two shifts away from the zero–sequence – any larger image would be very involved and confusing. As above in the following figure the local view of the arising highly arc transitive digraph is shown. Not surprisingly it consists of* $K_{2,3}$*s which again are coloured.*

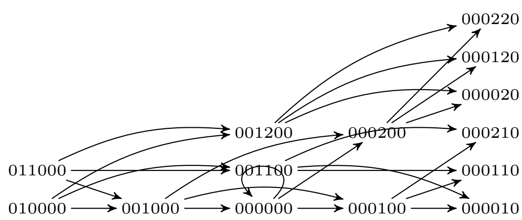

Figure 25: $S(K_{2,3},0,0)$

2.13.2. Rational–circles digraphs

The graph $Z \otimes S(K_{n,m},\delta_1,\delta_2)$ can also be gained from a very different approach.

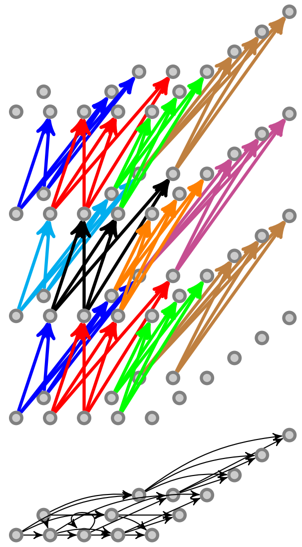

Figure 26: $Z \otimes S(K_{2,3}, 0, 0)$

Construction 2.52 *Let* $u, v \in \mathbb{N}$ *with* $\gcd(u, v) = 1$. *Then*

$$A_{u,v} = \{\frac{w}{u^m v^n} \mid w, m, n \in \mathbb{Z}\}$$

is a subgroup of $(\mathbb{Q}, +)$*. Obviously,* $\mathbb{Z}$ *is a normal subgroup of* $A_{u,v}$ *and so one can think of* $A_{u,v}/\mathbb{Z}$ *as "rational circles". Now we define the digraph* $D(u, v)$ *with the vertex set*

$$V(D(u, v)) := \mathbb{Z} \times A_{u,v}/\mathbb{Z}.$$

We define a group $G = \langle g, h \rangle$ *generated by*

$$\begin{aligned} g : (n, r) &\mapsto (n + 1, r) \\ h : (n, r) &\mapsto (n, r + \left(\frac{u}{v}\right)^n) \end{aligned}$$

and consider its action on the set of possible edges $V(D(u, v))^2$*. We choose the edge set to be the orbit containing* $((0, 0), (1, 0))$*:*

$$E(D(u, v)) := G(((0, 0), (1, 0)))$$

It turns out, that $G \subset \mathrm{Aut}D(u, v)$ acts transitively on the s–arcs. But we are not going into detail on that because we can kill the topic by recognizing the following theorem.

Theorem 2.53 *If* $u, v \in \mathbb{N}$ *are relatively prim then*

$$D(u, v) \cong Z \otimes S(K_{u,v}, \delta_1, \delta_2)$$

$\rightarrow$ 66

2.14. Diestel–Leader graphs

The **Diestel–Leader graphs** were introduced in [15]. They are constructed from two regular trees. If both are binary, the resulting graph is a Cayley–graph of the lamplighter group on $\mathbb{Z}$. But we will not focus on that in the present thesis.

Construction 2.54 (Diestel–Leader) *We start with a tree A with in–valency 1 and out–valency n and a second tree B with in–valency m and out–valency 1. Then we choose epimorphisms $\phi_A : A \to Z$ and $\phi_B : B \to Z$. The sets $\phi_A^{-1}(n) \cup \phi_B^{-1}(n)$ are called* **horocycles**. *Every pair of vertices $a \in V(A)$ and $b \in V(B)$ which are in the same horocycle is a vertex of the digraph $DL_{n,m}$. Set*

$$V(DL_{n,m}){:=}\{(a,b) \mid a \in V(A), b \in V(B), \phi_A(a) = \phi_B(b)\}.$$

Finally, we put edges between vertices if both their coordinates are adjacent in the underlying trees

$$E(DL_{n,m}){:=}\{((a,b),(c,d)) \mid (a,c) \in E(A), (b,d) \in E(B)\}.$$

Definition 2.55 (Diestel–Leader) *The digraph $DL_{n,m}$ from construction 2.54 is called Diestel–Leader graph with valencies n and m.*

Remark 2.56

1. *Be aware of the slightly dangerous notation as the Diestel–Leader graph $DL_{n,m}$ and the universal covering digraph $DL(\Delta)$ almost produce a notational conflict.*

2. *The Diestel–Leader graph $DL_{2,2}$ is a Cayley–graph of the lamplighter group on $\mathbb{Z}$. But for $n \neq m$ they are not even quasi–isometric to any Cayley–graph. Thus they answer a question by Woess whether there are transitive graphs which are not quasi–isometric to any Cayley–graph.*

Proposition 2.57 *The Diestel–Leader graph $DL_{n,m}$ is highly arc transitive.*→ 65

2.14.1. Broom–graphs

Diestel–Leader graphs are also known as broom–graphs with a different approach and construction. It is quite similar to the construction of the Evans–graph since we are again going to glue trees together.

Construction 2.58 (Broom–graph) *We start with a regular tree T with in–valency $d^- = 1$ and out–valency $d^+ = n \geq 2$ and choose an epimorphism $\phi : T \to Z$. We will glue "upper halfs" of T onto it. Therefore let an upper half be the subgraph G induced by $\{x \in V(T) \mid \phi(x) \leq 0\}$ (Any other integer would yield an isomorphic graph.). G can be thought of as T truncated at a horocycle.*

At every horocycle H_i:=$\{x \in V(T) \mid \phi(x) = i\}$ of T we attach a copy G_i of G by identifying H_i with the bottom horocycle of G_i in the canonical way. We call the resulting graph D_1.
In the next step we attach copies $G_{i,j}$ of G at every non–bottom horocycle of every G_i in the same way to get D_2. Similarly we get D_m from D_{m-1} by attaching copies $G_{i_1,\dots,i_m}$ at every non–bottom horocycle of every $G_{i_1,\dots,i_{m-1}}$. (By referring to horocycles here we have to take care that we really mean horocycles of $G_{\dots}$ and not horocycles of D_k.)
Finally we define B_n as the limit

$$B_n := \bigcup_{k \in \mathbb{N}^+} D_k.$$

Remark 2.59

1. *The trivial case $n = 1$ would yield an upward binary tree (with in–valency 2 and out–valency 1).*
2. *From the view of any vertex x, the broom–graphs look like trees in both directions. $x^{\Rightarrow}$ induces a subtree with out–valency n whereas ${}^{\Rightarrow}x$ induces a subtree with in–valency 2.*
3. *Obviously, one is not bound to attach a single tree at the horocycles. Indeed we can choose any $m \in \mathbb{N}$ and attach m copies of G at each horocycle and call the resulting digraph $B_{n,m+1}$. The above subgraph ${}^{\Rightarrow}x$ will then be a tree with in–valency $m + 1$.*

Definition 2.60 *The digraph B_n from Construction 2.58 and the digraph $B_{n,m}$ from Remark 2.59 are called* **broom–graph** *with out–valency n (and in–valency m, respectively).*

As mentioned above we finally remark Theorem 2.61.

Theorem 2.61 *The Diesel–Leader graph is isomorphic to the broom–graph.*

$$DL_{n,m} \cong B_{n,m}$$

→ 66

2.15. Pancake–tree

2.15.1. Quadratic pancake–tree

The quadratic pancake–tree was invented in [2] as example of an highly arc transitive digraph with thick and thin ends. It could be easily described as $D(\Delta)$ for Δ the bipartite directed grid. Nevertheless, we are going to present the intuitive construction from [2].

Construction 2.62 (Quadratic pancake–tree) *We start with defining the bipartite directed grid (the pancake). The undirected grid has vertices from* $\mathbb{Z}^2$ *and edges between vertices which are equal in the one and neighbouring in the other component. Every vertex has a well defined distance to* $(0,0)$ *and with that arising a parity. Thus every edge connects a vertex of odd parity with a vertex of even parity. We orientate every edge from even to odd to get the pancake as shown in Figure 27.*

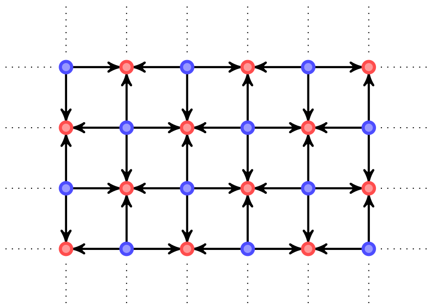

Figure 27: quadratic pancake

We are now going to glue infinitely many copies of the pancake together. We start by attaching a pancake to every vertex of an initial pancake by identifying the vertex with a vertex of the other parity of the attached pancake.
Then we iteratively glue pancakes to all the vertices of the new graph which have no attached pancake yet. As a limit we get the pancake–tree.

Definition 2.63 *The graph from the above Construction 2.62 is called* **quadratic pancake–tree***.*

Proposition 2.64 *The quadratic pancake–tree is highly arc transitive.* → 66

2.15.2. Hexagon pancake–tree

Obviously every other infinite, bipartite, 1–arc transitive, connected, one–ended digraph could be used as Δ and would yield the same properties we are looking for (see later), but there is just one more plain one. Thus we mention it here separately.

Construction 2.65 (Hexagon pancake–tree) *We define the hex–pancake on* $\mathbb{Z}^2$*. We start again with an undirected graph on the vertex set* $\mathbb{Z}^2$*. We put vertical edges between vertices which agree on the first component and which are neighbouring in the second. Then we add edges* $((i,2j),(i+1,2j+1))$*. The distance from* $(0,0)$ *gives again a well defined parity and the rest of the construction runs analogously to Construction 2.62.*

Figure 28 shows on the left the hex–pancake as defined above and on the right the intuitive hex–pancake.

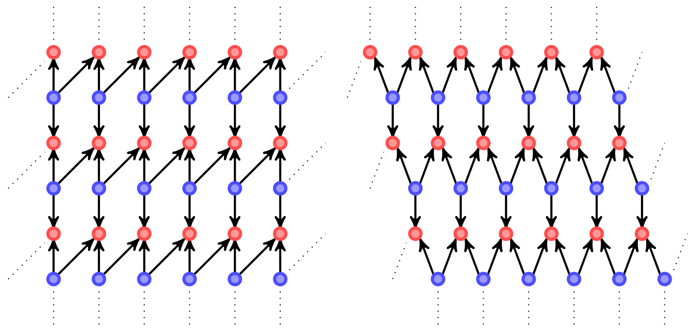

Figure 28: hex–pancake

Definition 2.66 *The digraph from the above Construction 2.65 is called* **hexagon pancake–tree**.

Proposition 2.67 *The hexagon pancake–tree is highly arc transitive.* → 66

3. Introduction

3.1. Overview

The research on highly arc transitive digraphs was started with [1] by *Cameron*, *Praeger* and *Wormald* who defined associated digraphs for 1–arc transitive digraphs and the universal covering digraphs. They investigated Property Z and finally presented some constructions for new highly arc transitive digraphs. They stated some questions and conjectures that gave rise to further research over the past 20 years, but still highly arc transitive digraphs are far from being understood. Still we are far away form characterizing them or at least interesting classes of them.
The main theorem about universal covering digraphs presented in [1] understands the class of digraphs with the same associated digraph as a category and finds the universal covering digraph to be projective in that category.
In [11] *Praeger* investigates the connection between the valencies on the one and Property Z and universal reachability relation on the other hand. This work was continued by *Evans* in [6] where he found a highly arc transitive digraph with finite out–valency that has universal reachability relation, and by *Malnič*, *Marušič*, *Seifter* and *Zgrablič* in [7], where they presented a locally finite highly arc transitive digraph that has neither universal reachability relation nor Property Z. Finally *DeVos*, *Mohar* and *Šámal* clarified in [5] for which in- and out–valencies highly arc transitive digraphs with universal reachability relation exist and for which not.
Möller investigated lines and their descendants in highly arc transitive digraphs in [4]. He finds that highly arc transitive digraphs which are covered by the descendants of one of their lines can be epimorphic mapped onto a tree with finite fibres in a unique way. Furthermore he finds the spread of a locally finite highly arc transitive digraph to be an integer.
In the same year (2002), *Möller* published [12] where he found a connection between highly arc transitive digraphs and *Willis*' structure theory about totally disconnected, topological groups. This topic was picked up by *Malnič*, *Marušič*, *Seifter* and *Zgrablič* in [7] where they presented two proofs about topological groups that are lead using highly arc transitive digraphs.
In [2], *Seifter* studied transitive graphs with more than one end. He found that highly arc transitive digraphs can simultaneously have thin and thick ends but not both kinds can contain half–lines. Moreover he discovered that 2–arc transitive digraphs with prime–degrees and a connected D–cut (Dunwoody–structre–cut) must already be highly arc transitive. Moreover he showed for 1–arc transitive digraphs with different in- and out–degree where the larger one is a prime that they also have to be highly arc transitive. These theorems motivated him to conjecture that connected, locally finite digraphs with more than one end that are 2–arc transitive are already highly arc transitive. This

conjecture was disproved by *Mansilla* in [10] by finding a family of strictly 2–arc transitive digraphs. In the present thesis we even conjecture that certain families of digraphs are strictly k–arc transitive digraphs for every positive integer k.
Recently, in [13] *Hamann* and *Hundertmark* characterized C–homogeneous digraphs with more than one end. C–homogeneous digraphs are related to highly arc transitive digraphs in the sense, that they are highly arc transitive if they contain no triangles. Their work does not characterize any classes of highly arc transitive digraphs, but gives rise to an interesting family of highly arc transitive digraphs, which seems to be the second known family that has neither Property Z nor universal reachability relation – the first such was given by *Malnič*, *Marušič*, *Seifter* and *Zgrablič* in [7].
In [8], *Seifter* and *Imrich* recognized that already connected, transitive, two–ended graphs are spanned by a set of lines. A fact that might be helpful in trying to prove the conjecture from [1] that two–ended, highly arc transitive digraphs consist of $K_{n,n}$s. *Krön*, *Seifter* and the author did not succeed in proving that but made some notes which are presented in Section 4.2.8.

In the following we present the needed notions for the topics mentioned above.

3.1.1. Reachability

First we are going to define an equivalence relation on the edges of a digraph. We will use it to define an associated digraph for every 1–arc transitive digraph and form in that way association classes. Therefore we first need the notion of an **alternating walk**.

Definition 3.1 (alternating walk) *An* **alternating walk** *is a sequence of vertices $(x_0, \ldots, x_n)$ such that there is a sequence of edges of alternating directions between the vertices. That is that there are edges $(x_{2k}, x_{2k+1}) \in E(D)$ pointing forward and edges $(x_{2k+2}, x_{2k+1}) \in E(D)$ pointing backward or edges $(x_{2k+1}, x_{2k}) \in E(D)$ pointing backward and edges $(x_{2k+1}, x_{2k+2}) \in E(D)$ pointing forward.*

Remark 3.2 *Note that the two cases in the definition above do not exclude each other.*

Definition 3.3 (Reachability–Relation) *Let D be a digraph. Two edges e_1, e_2 of D are* **reachable** *from each other if there exists an alternating walk starting with e_1 and ending with e_2. We write $e_1 \mathcal{A} e_2$.*

Remark 3.4

1. *The Reachability–Relation obviously is an equivalence relation since*

- *A single edge is always an alternating walk, thus it is reflexive.*
- *The inverse walk must be alternating as well, thus it is symmetric.*
- *Given two alternating walks which agree on one terminating edge, its concatenation is again an alternating walk, thus the reachability relation is also transitive, thus an equivalence relation.*

2. *We denote the corresponding equivalence classes by $\mathcal{A}(e)$ and the induced subgraph by $\langle\mathcal{A}(e)\rangle$.*

Definition/Lemma 3.5 (associated digraph) *If D is 1–arc transitive then the digraphs $\langle\mathcal{A}(e)\rangle$ are isomorphic for all $e \in E(D)$. We call this digraph the* **associated digraph** *and denote it by $\Delta(D)$* $\rightarrow 77$

Example 3.6 *In Section 2 we presented examples whose figures make it easy to see the associated digraphs.*

1. *Figures 8, 9 and 10 show $K_{n,n}$–tubes with associated digraph $K_{n,n}$.*
2. *The associated digraph of the universal covering digraph $DL(\Delta)$ is Δ (see Figure 14).*
3. *The associated digraphs of the tensor–products with sequences digraphs are shown coloured in Figures 22, 24 and 26.*

We will be interested in the class

Definition 3.7 (association class) *Let Δ be a connected, bipartite, 1–arc transitive digraph. We define the* **association class** *of Δ as*

$$\mathcal{D}(\Delta) := \{D \mid D \text{ digraph}, \Delta(D) = \Delta\}$$

Remark 3.8 *Definition 3.7 is due to [1]. It does not exclude Δ from $\mathcal{D}(\Delta)$. But we will need the digraphs in $\mathcal{D}(\Delta)$ to contain arcs of arbitrary length. To guarantee that we will consider $\mathcal{D}(\Delta)$ as the class above without Δ.*

3.1.2. Property Z

Cameron, *Praeger* and *Wormald* noticed, that all highly arc transitive digraphs they considered in [1] could be epimorphicly mapped onto the integer line and formalized this property.

Definition 3.9 (Property Z) *A digraph D is said to have* **Property Z** *if there exists an epimorphism $\phi : D \rightarrow Z$ onto the integer line as defined in 2.1.*

Obviously in a digraph with Property Z every $\mathcal{A}(e)$ is mapped to a single edge of Z. Thus if the reachability relation was universal, Property Z would be excluded and vice verse. The questions arose if there are highly arc transitive digraphs without Property Z or even with universal reachability relation. Indeed there are such digraphs, even digraphs without any of these properties (so kind of in between these extreme cases) were found.

3.1.3. Spread and bounded automorphisms

Cameron, *Praeger* and *Wormald* studied the connection between Property Z and the spread of a highly arc transitive digraph in [1]. Later *Möller* proved in his paper about descendants [4] that the spread of an highly arc transitive digraph is an integer. For technical reasons we also include the definition of bounded automorphism as it is of importance in [4].

Definition 3.10 (Spread) *Let D be a transitive digraph with finite out–valency $d^+(D)$ and finite in–valency $d^-(D)$ and let $x \in V(D)$. We define the* **out–spread** *of D as*

$$s^+(D) := \text{limsup}_{\ k\to\infty} \sqrt[k]{|x^{\Rightarrow k}|}$$

and analogously the **in–spread** *as*

$$s^-(D) := \text{limsup}_{\ k\to\infty} \sqrt[k]{|{}^{k\Rightarrow}x|}.$$

Definition 3.11 (bounded automorphism) *Let D be a connected digraph. An automorphism $g \in \text{Aut}(D)$ is* **bounded** *if there is a constant C such that*

$$\forall x \in V(D) : \text{dist}\ (x, g(x)) \leq C.$$

3.1.4. Categories

The universal covering digraph defined in [1] is a projective object in an association class. We need some terminology of categories to formulate this result.

Definition 3.12 (Category) *A* **category** $C(\text{ob}\ (C), \text{Mor}\ (C), \circ)$ *consists of a class of* **objects** $\text{ob}\ (C)$*, a class of* **morphisms** $\text{Mor}\ (C)$*, which contains morphisms $f : a \to b$ where $a, b \in \text{ob}\ (C)$ (specifying the morphisms between two objects we denote $\text{Mor}\ (a, b)$) and a composition $\circ : \text{Mor}\ (a, b) \times \text{Mor}\ (b, c) \to \text{Mor}\ (a, c)$ such that*

- $\forall f \in \text{Mor}\ (a, b), g \in \text{Mor}\ (b, c), h \in \text{Mor}\ (c, d) : f \circ (g \circ h) = (f \circ g) \circ h$
- $\forall x \in \text{ob}\ (C)\ \exists I_x \in \text{Mor}\ (x, x)\ \forall f \in \text{Mor}\ (x, \cdot)\ \forall g \in \text{Mor}\ (\cdot, x) :$ $I_x \circ f = f \ \wedge\ g \circ I_x = g$

Remark 3.13 *The neutral morphism often is more intuitively introduced with*

$$I_a \circ f = f = f \circ I_b \textit{ for } f \in \text{Mor}\ (a, b),$$

but that leads to an even worse mess of quantifiers in the formalization.

Definition 3.14 (projective object) *An object $p \in \text{ob}\ (C)$ of the category C is called* **projective***, if*

$$\forall h \in \text{Mor}\ (p, c)\ \forall g \in \text{Mor}\ (b, c)\ \exists h' \in \text{Mor}\ (p, b) : h' \circ g = h.$$

That is, if for all g and h there is a h' such that the diagram in Figure 29 commutes.

$$\begin{tikzcd}
p \arrow[d, "h'"'] \arrow[dr, "h"] & \\
b \arrow[r, "g"'] \arrow[ur, phantom, "\circlearrowleft"] & c
\end{tikzcd}$$

Figure 29: morphisms of an projective object

3.1.5. Cuts

Seifter deals in [2] with graphs with more than one end, cuts play an important role there. We are using edge cuts here. *Dunwoody* and *Krön* recently came up with vertex cuts which already found application on C–homogeneous digraphs, which are also mentioned in the present thesis. But we will not use vertex cuts here, for more information on these see the references in [13].

Definition 3.15 (cut, crossing cuts, tight cut, connected cut) *Let X be a graph, $A \subset V(X)$ and $B = V(X) \setminus A$.*

1. *The set F of edges that connect A and B is called a* **cut**. *A and B are called* **sides** *of F.*

2. *If the induced graphs $\langle A \rangle$ and $\langle B \rangle$ are connected then the cut F is called a* **tight** *cut.*

3. *Let F_1 and F_2 be two cuts of X with sides A_1, B_1 and A_2, B_2 respectively. If the sets $A_1 \cap A_2$, $A_1 \cap B_2$, $B_1 \cap A_2$ and $B_1 \cap B_2$ are all nonempty we say that the cuts F_1 and F_2* **cross**.

4. *If a cut F induces a connected subgraph $\langle F \rangle$ the cut is said to be* **connected**.

Definition 3.16 (D–cut) *Let X be an infinite, connected graph and F a finite, tight cut with two infinite sides. F is called* **D–cut** *if it does not cross any $g(F)$ for $g \in \mathrm{Aut}X$.*

The following result by *Dunwoody* gave rise to the notion of a D–cut.

Theorem 3.17 (Dunwoody) *Every infinite, connected graph X which has a finite cut with infinite sides also has a D-cut.* $\rightarrow$ 77

3.1.6. C–homogeneous digraphs

In a very recent work [14] *Hamann* and *Hundertmark* classified C–homogeneous digraphs with more than one end. This is interesting for the present thesis because C–homogeneous digraphs without triangles are highly arc transitive. Nevertheless, it is not really a step towards a classification of highly arc transitive digraphs, but it provides us with another example. *Hamann* and *Hundertmark* give an interesting class of graphs (actually subgraphs of the DeVos–Mohar–Šámal–digraphs) that has neither Property Z nor universal reachability relation.

Definition 3.18 (C–homogeneous) *A graph X or digraph D is called* **C–homogeneous** *if every isomorphism between two finite, induced subgraphs extends to an automorphism.*

Remark 3.19

1. *s–arcs are subgraphs but not necessarily induced. A C–homogeneous digraph in which all s–arcs are induced is obviously highly arc transitive.*

2. *Note that C–homogeneity is a very different notion for graphs and digraphs because the isomorphisms look very different.*

Definition 3.20 (C–homogeneous types) *Let X be a C–homogeneous digraph.*

1. *If the underlying undirected graph is C–homogeneous, X is of* **Type I**.

2. *If the underlying undirected graph is not C–homogeneous, X is of* **Type II**.

3.1.7. Topological Groups

In [12], *Möller* discovered a connection between highly arc transitive digraphs and *Willis*' structure theory of totally disconnected topological groups. This idea is picked up again in [3], where two more proofs in that area are presented that use highly arc transitive digraphs.

Definition 3.21 (topological group) *Let $(G, \circ)$ be a group and τ a topology on G. Then G is called a* **topological group** *with topology τ if $\circ : G \times G \to G$ is continuous with respect to τ.*

Definition 3.22 (tidy) *Let G be a locally compact, totally disconnected group, $g \in G$ an element and $U < G$ compact and open. Set*

$$\begin{aligned} U_+ &:= \bigcap_{i=0}^{\infty} g^i U g^{-i} \\ U_- &:= \bigcap_{i=0}^{\infty} g^{-i} U g^i. \end{aligned}$$

U is **tidy** *for g if*

1. $U = U_+U_- = U_-U_+$ *and*

2. $\bigcup_{i=0}^{\infty} g^i U_+ g^{-i}$ *and* $\bigcup_{i=0}^{\infty} g^{-i} U_- g^i$ *are both closed in* G.

Definition 3.23 (scale function) *Let* G *be a locally compact, totally disconnected group. The* **scale function** *of* G *is given by*

$$s(g) := \min\{\left[U : U \cap g^{-1}Ug\right] \mid U < G \text{ compact, open}\}$$

Definition 3.24 (FC^- element) *Let* G *be a totally disconnected, locally compact group. An element* $g \in G$ *is called a* FC^- **element** *if the conjugacy class of* g *has compact closure in* G.

Definition 3.25 *Let* G *be a topological group. An element* $g \in G$ *is called* **periodic** *if the cyclic subgroup* $\langle g \rangle < G$ *has compact closure in* G. *Set* $P(G)$ *the set of all periodic elements of* G.

3.2. Questions

In [1], *Cameron*, *Praeger* and *Wormald* stated some questions each of which subsequently led to further investigation. In this subsection we will quote these questions.

3.2.1. Universality?

Question 3.26 (1.2) *Are there any locally finite highly arc transitive digraphs for which the reachability relation* $\mathcal{A}$ *is universal?*

Answer 3.27 *Yes. See Proposition 4.24.*

3.2.2. Property Z?

Question 3.28 (1.3) *Are there any highly arc transitive digraphs (apart from directed cycles) in* $\mathcal{D}(\Delta)$, *which do not have a digraph homomorphism onto* Z*?*

Answer 3.29 *Yes. See Theorem 4.31.*

3.2.3. Are covering projections isomorphisms?

Question 3.30 (2.10) *Must a covering projection* $\phi : D \to D$ *of a connected locally finite digraph* D *be an isomorphism?*

Answer 3.31 *Open.*

3.2.4. Spread > 1 and Property Z

Question 3.32 (3.8) *Given a real number $c > 1$, are there any highly arc transitive digraphs with out–spread or in–spread c which do not have Property Z?*

Answer 3.33 *There are only highly arc transitive digraphs with integer spread. See Theorem 4.32.*
For every $n \in \mathbb{N}^+$ there is an Evans–graph with out–spread n. One can consider an inverse Evans–graph for the in–spread.

3.2.5. Finite fibres – the Cameron–Praeger–Wormald–Conjecture

Question 3.34 (3.9) *Let D be a connected highly arc transitive digraph with finite out–valency (respectively, in–valency) such that the out–spread (respectively, in–spread) of D is 1. Let D have Property Z with $\phi : D \to Z$.*
Is it true, that the inverse Image $\phi^{-1}(0)$ of 0 is finite?

Answer 3.35 *No.*
The regular tree with arbitrary in–valency d^- and out–valency $d^+ = 1$ is a counterexample.

Conjecture 3.36 (3.3) *If D is a connected highly arc transitive digraph in $\mathcal{D}(\Delta)$ with Property Z, and $\phi : D \to Z$ is a digraph epimorphism such that the inverse image $\phi^{-1}(0)$ of 0 is finite, then Δ is a complete bipartite digraph.*

Answer 3.37 *Open.*

4. Statements

This section splits into four parts. First we collect some lemmas in Section 4.1 before coming to the main results in Section 4.2. We continue by collecting the properties of known highly arc transitive digraphs in Section 4.3. Section 4.2.8 is the only part of the thesis whose proofs are not shifted to Section 5. This is because the tiny results there are new, whereas the object of the rest of the thesis was to collect examples and facts with minor concern on proofs. We conclude with a short view on open questions in Section 4.4.

4.1. Useful lemmas

4.1.1. Associated digraphs and reachability

We start off with a basic result about association digraphs.

Proposition 4.1 *Let D be a connected 1–arc transitive digraph.*

(1) *$\Delta(D)$ is connected and 1–arc transitive.*

(2) *Exactly one of the following is true:*

- (a) *The reachability-relation is universal.*
- (b) *$\Delta(D)$ is bipartite.* → 71

4.1.2. Covering projections

The upcoming lemma gave rise to Question 3.30. Is transitivity really a necessary assumption?

Lemma 4.2 *Let X be a connected, locally finite digraph. If X is transitive or 1–arc transitive then every covering projection $\phi : X \to X$ is an isomorphism.* → 72

From the above lemma we get a corollary that we are going to formulate as a lemma about how covering projections behave on associated digraphs Δ. This will later be a key step to the main theorem about universal covering digraphs and covering projections.

Lemma 4.3 *Let Δ be a connected, locally finite, 1–arc transitive, bipartite digraph, $X, Y \in \mathcal{D}(\Delta)$ and $\phi : X \to Y$ a covering projection. Then for every edge $e \in E(C)$ the restriction $\phi|_{\mathcal{A}(e)} : \langle \mathcal{A}(e) \rangle \to \langle \mathcal{A}(\phi(e)) \rangle$ is an isomorphism.* → 72

4.1.3. Property Z

Lemma 4.4 is formulated as a step of a proof in [1] in a more technical way. It is given as a lemma here because it provides us with a very important condition in order to prove or disprove that a digraph has Property Z.

Lemma 4.4 *A digraph D has Property Z if and only if all its cycles are balanced.* → 72

The universal covering digraph is the "least connected" digraph in its association class. It has Property Z because all its cycles must stay inside the same reachability class.

Lemma 4.5 *If Δ is a connected, 1–arc transitive, bipartite digraph, then $DL(\Delta)$ has Property Z.* → 72

We also formulate as lemma that Property Z and universal reachability relation are opposing properties.

Lemma 4.6 *If D is a connected, 1–arc transitive digraph then it cannot have universal reachability relation and Property Z simultaneously.* → 72

4.1.4. Paths and alternating walks

On first sight it might seem typical for highly arc transitive digraphs that all arcs between the two fixed vertices have the same length. But indeed there are counterexamples that are not locally finite e.g. the ordered field digraph that even has arcs of every length between any two vertices.

Lemma 4.7 *Let D be a connected, highly arc transitive digraph with finite out–valency and let $x, y \in V(D)$ such that $y \in x^{\Rightarrow}$. Then either all directed paths from x to y have the same length d or D is a directed cycle.* → 72

We find a technical condition using a setwise stabilizer of a vertex set that does not fix any vertices in it and which has only subgroups with a big enough index. This condition excludes the existence of certain alternating walks and thus can be used to disprove the universality of the reachability relation.

Lemma 4.8 *Let D be a digraph with $d^{-}(D) = d^{+}(D) = d$. Let $e = (x, y)$ be an edge of D and $\Omega \subseteq V(D) \setminus \{x, y\}$ and let $H \leq \mathrm{Aut}D$ be a group of automorphisms with $H\Omega = \Omega$ but $\forall v \in \Omega\ \exists h \in H : hv \neq v$. Let finally all $A < H|_{\Omega}$ have index $[A : H|_{\Omega}] \geq d$. Then there is no alternating walk with initial vertex in e and terminal vertex in Ω.* → 72

4.1.5. Lines and descendants

The automorphism group of an highly arc transitive digraph acts transitively not just on the s–arcs but also on the lines. Moreover every line can be shifted by an automorphism. Remember that we defined $\mathcal{L}(D)$ as the set of lines in a digraph D.

Lemma 4.9 *Let D be an infinite, locally finite, highly arc transitive digraph and $L = (\dots, x_{-1}, x_0, x_1 \dots)$ a line. Then*

1. $\mathrm{Aut}(D) \circlearrowleft \mathcal{L}(D)$ *transitively.*
2. $\forall k \in \mathbb{Z}\ \exists g \in \mathrm{Aut}(D)\ \forall i \in \mathbb{Z} : g(x_i) = x_{i+k}$ → 73

We collect some properties of the descendants of a line in a highly arc transitive digraph.

Lemma 4.10 *Let D be an infinite, locally finite, highly arc transitive digraph and $L \in \mathcal{L}(D)$. Then*

1. $d^+(\langle L^{\Rightarrow} \rangle) = d^+(D)$
2. $\langle L^{\Rightarrow} \rangle$ *is highly arc transitive.*
3. $\langle L^{\Rightarrow} \rangle$ *has more than one end.*
4. $\langle L^{\Rightarrow} \rangle$ *has Property Z.* → 73

We can use bounded automorphisms to gain information about the ends of a highly arc transitive digraph.

Lemma 4.11 *Let D be a locally finite, infinite, highly arc transitive digraph and let L be a line in D. Let $g \in \mathrm{Aut}(D)$ be a bounded automorphism. Then*

1. *Let $L_1, L_2 \subset L^{\Rightarrow}$ be two positive half–lines with $L_1 = g(L_2)$, then L_1 and L_2 lie in the same end.*
2. *If there is a vertex $v \in V(D)$ such that $g(v) \in v^{\Rightarrow}$ then D is two–ended.* → 73

Given a line in a highly arc transitive digraph the following lemma says basically that if we cannot "see away" from the end of the line in the one direction, that we see the entire graph in the other direction.

Lemma 4.12 *Let D be a connected, locally finite, highly arc transitive digraph and $L \in \mathcal{L}(D)$.*

1. *If $L^{\Rightarrow}$ is two–ended then $D = {}^{\Rightarrow}L$.*
2. *If ${}^{\Rightarrow}L$ is two–ended then $D = L^{\Rightarrow}$.* → 73

4.1.6. Spread

If the in- and out–valency do not agree then the digraph must spread fast.

Lemma 4.13 *Let D be a transitive, locally finite digraph. If $d^{-}(D) \neq d^{+}(D)$ then in–spread or out–spread must be greater than* 1. → 73

4.1.7. Cuts

In [2] *Seifter* comes up with an interesting observation about D–cuts

Lemma 4.14 *Let D be a connected, locally finite, 2–arc transitive digraph. Then no D–cut contains a 2–arc.* → 73

4.2. Theorems

4.2.1. Universal covering digraphs and covering projections

First we legitimate the name "universal covering digraph" by explaining the covering projections.

Theorem 4.15 *Let Δ be a connected, bipartite, 1–arc transitive digraph.*

1. *The universal covering digraph $DL(\Delta)$ lies in the class $\mathcal{D}(\Delta)$.*
2. *$DL(\Delta)$ is a covering digraph for each digraph $X \in \mathcal{D}(\Delta)$.*
3. *Let $X \in \mathcal{D}(\Delta)$, then for any pair of s–arcs $a_1 \subset DL(\Delta)$, $a_2 \subset X$ there exists a covering projection $\phi_{a_1,a_2} : DL(\Delta) \to X$ that takes a_1 to a_2.* → 73

Let A and B be classes of graphs and denote with $\mathrm{Cov}\,(A, B)$ the class of covering projections from graphs in A to graphs in B, then we can formulate the following result

Theorem 4.16 *Consider the category $(\mathcal{D}(\Delta), \mathrm{Cov}\,(\mathcal{D}(\Delta), \mathcal{D}(\Delta)), \circ)$ where Δ is connected, locally finite, bipartite and 1–arc transitive. Then $DL(\Delta)$ is a projective object.* → 74

4.2.2. Property Z versus universal reachability relation

Property Z and having universal reachability relation exclude each other, as we have seen in Lemma 4.6. So for highly arc transitive digraphs they are opposing properties. In this section we collect some facts and conditions about these two notions. We start with a sufficient condition for having Property Z.

Theorem 4.17 *Let D be a connected, highly arc transitive digraph with finite out–valency, such that the out–spread of D is 1, then D has Property Z. (The same holds for the in–spread.)* $\to 74$

If we control both spreads we get the following condition. Compare these two results (4.17, 4.18) with Question 3.34

Theorem 4.18 *Let D be a connected, highly arc transitive digraph with in–spread and out–spread both 1. Then every epimorphism $\phi : D \to Z$ has finite fibres.* $\to 74$

In [11], *Praeger* gives a strong result if the in- and out–valencies differ.

Theorem 4.19 *Let D be an infinite, connected, transitive and 1–arc transitive digraph with finite in- and out–valencies. If $d^-(D) \neq d^+(D)$ then D has Property Z with infinite fibres.* $\to 74$

In Section 2, we already saw some highly arc transitive digraphs with Property Z. One family of such graphs was only remarked there (Remark 2.50) because Property Z was defined in Section 3. Here is the corresponding theorem.

Theorem 4.20 *Let $\Delta = \Delta^- \cup \Delta^+$ be a connected, 1–arc transitive, bipartite digraph and let $\delta_1 \in \Delta^-, \delta_2 \in \Delta^+$ and let D be a highly arc transitive, connected, digraph with Property Z. Then $D \otimes S(\Delta, \delta_1, \delta_2)$ is connected and highly arc transitive with Property Z.* $\to 74$

More generally (not considering highly arc transitivity), the following result is true.

Proposition 4.21 *Let D be a digraph with Property Z and G be any nonempty digraph. Then $D \otimes G$ has Property Z.* $\to 74$

Remark 4.22 *Note that if both D and G have Property Z then $D \otimes G$ cannot be connected. If there is an edge $((d_1, g_1), (d_2, g_2)) \in E(D \otimes G)$ then $\phi(d_1) + 1 = \phi(d_2)$ and $\phi(g_1) + 1 = \phi(g_2)$ and thus $\phi(d_1) + \phi(g_1) + 2 = \phi(d_2) + \phi(g_2)$. That means that edges can only run between vertices of $D \otimes G$ if they have the same parity and thus there are at least two components.*
Also note that it follows that the sequences digraph never has Property Z. We earn a method to disprove Property Z by proving the connectedness of a tensor–product.

Another proposition that would also fit in Section 2.13 is stated here for the same reason.

Proposition 4.23 *If D is a digraph, C is highly arc transitive with Property Z and $Z \otimes D$ is highly arc transitive, then $C \otimes D$ is highly arc transitive. The same holds for s–arc transitive.* $\to 64$

Let us see what we can on the other hand say about highly arc transitive digraphs with universal reachability relation. In particular, if there are any such digraphs.

Proposition 4.24

1. *There are highly arc transitive digraphs with universal reachability relation.*
2. *There are highly arc transitive digraphs with finite out–spread and universal reachability relation.*
3. *There are locally finite, highly arc transitive digraphs with universal reachability relation.* $\rightarrow 74$

Initially, it was not clear if locally finite, highly arc transitively digraphs with universal reachability relation existed. Thus it was attempted to disprove that. That way some conditions were made up that excluded the universal reachability relation.

Theorem 4.25 *Let D be a connected, locally finite, highly arc transitive digraph. Then*

1. *If $d^-(D) \neq d^+(D)$ then D does not have universal reachability relation.*
2. *If $d^-(D) = d^+(D)$ is prime then D does not have universal reachability relation.*
3. *If $d^-(D) = d^+(D) = 1$ then either D is a directed cycle or $D \cong Z$ and thus does not have universal reachability relation.* $\rightarrow 74$

Theorem 4.26 *Let p be a prime and D a 2–arc transitive digraph with in- and out–valency p. Then D does not have universal reachability relation.* $\rightarrow 75$

Theorem 4.27 *Let D be a 1–arc transitive digraph with $d^-(D) = d^+(D) = d > 1$. Let $e = (x, y) \in E(D)$ be an edge of D such that $\mathrm{Stab}_{\mathrm{Aut}D}(e)|_{y \Rightarrow 1}$ contains a nontrivial subgroup K which has no nontrivial permutation representation of degree less than d (that is every permutation π of $V(D)$ induced by the action of K on $V(D)$ has the property that if $\pi^n = \mathrm{id}$ for some $n < d$ then already $\pi = \mathrm{id}$). Then the reachability relation of D is not universal.* $\rightarrow 75$

After the first examples for Proposition 4.24 were found the question arose if one could do better than Theorem 4.25. This was answered by *DeVos*, *Mohar* and *Šámal* in [5] by constructing a family of graphs with universal reachability relation.

Theorem 4.28 *For every pair of integers $n, m > 1$ there is a connected, highly arc transitive digraph D with $d^+(D) = d^-(D) = nm$ that has universal reachability relation.* → 75

Möller proved the following theorem that implies Property Z. Alike Property Z it uses an epimorphism onto an underlying structure – this time a tree T rather than the integer line.

Theorem 4.29 *Let D be a locally finite, highly arc transitive digraph such that there is an $L \in \mathcal{L}$ with $L^{\Rightarrow} = V(D)$. Let T be the tree with in–valency $d^-(T) = 1$ and finite out–valency $d^+(T) = t$. Then there exists an epimorphism $\phi : D \to T$. Moreover, ϕ induces a group action of $\mathrm{Aut}(D)$ on T in a natural way, i.e. for every $g \in \mathrm{Aut}(D)$ there is an automorphism $g_T{:=}\phi \circ g$ of T. In that sense $\mathrm{Aut}(D)$ acts transitive on the s–arcs of T for all $s \in \mathbb{N}$. Moreover the fibres $\phi^{-1}(x)$ are finite and of equal size for all $x \in V(T)$.* → 75

Corollary 4.30

1. *Let in the above situation $L^{\Rightarrow}$ have in–valency d^- and out–valency d^+. Then*
$$t = \frac{d^+}{d^-}$$
2. *Either $L^{\Rightarrow}$ has exactly two ends or its in- and out–valency differ.* → 75

We have already seen, that Property Z and universal reachability relation exclude each other. For some time only highly arc transitive digraphs with either the one or the other property were known. Thus it was interesting to find something in between.

Theorem 4.31 *There are highly arc transitive digraphs without Property Z and without universal reachability relation.* → 75

4.2.3. Spread

Möller observed that the spread of highly arc transitive digraphs is an integer.

Theorem 4.32 *Let D be a locally finite, highly arc transitive digraph with out–spread $s^+(D)$. Then $s^+(D) \in \mathbb{N}$. (The same holds for the in–spread.)* → 75

He also came up with another observation concerning the connection between the spread and lines (compare Lemma 4.12).

Theorem 4.33 *Let D be a connected, locally finite, highly arc transitive digraph with in–spread s^- and out–spread s^+.*

1. $s^- = 1 \iff \exists L \in \mathcal{L}(D) : L^{\Rightarrow} = V(D) \iff \forall L \in \mathcal{L}(D) : L^{\Rightarrow} = V(D)$
2. $s^+ = 1 \iff \exists L \in \mathcal{L}(D) : {}^{\Rightarrow}L = V(D) \iff \forall L \in \mathcal{L}(D) : {}^{\Rightarrow}L = V(D)$
→ 75

4.2.4. Automorphism groups

It does not necessarily take the full automorphism group to act highly arc transitive on a highly arc transitive digraph. Considering the stabilizer of an edge we can give a pretty small group that already acts highly arc transitively.

Theorem 4.34 *Let D be a connected, infinite, highly arc transitive digraph and $H \leq \mathrm{Aut}(D)$ be a group of automorphisms such that H acts highly arc transitively on D. Let $x \in V(D)$ and $e = (x, y)$ be an edge of D and $g \in \mathrm{Aut}(D)$ with $g(x) = y$. Then $\langle \mathrm{Stab}_H(e) \cup \{g\} \rangle$ acts highly arc transitively on D.* → 75

4.2.5. Ends, cuts, prime–degree and the Seifter–Conjecture

Seifter found a highly arc transitive digraph that simultaneously has thick and thin ends. Actually it is a universal covering digraph with an infinite Δ that contains a thick end. The point is that thin and thick ends cannot contain half–lines simultainously. As we know from *Möller* the automorphism group of a highly arc transitive digraphs acts transitive on the lines and thus transitively on the forward directed and backward directed ends. But we have no idea what happens with the ends that do not contain a half–line.

Theorem 4.35 *There are infinite, locally finite, connected, highly arc transitive digraphs with both thin and thick ends.* → 75

Theorem 4.36 *There are no infinite, locally finite, connected, highly arc transitive digraphs with both thin, directed and thick, directed ends.* → 76

In [2], *Seifter* investigated graphs with more than one end. He found the following condition taking advantage of the prime–degree. Note that a graph that has a D–cut has at least two ends.

Theorem 4.37 *Let D be a connected, 2–arc transitive digraph with $d^+(D) = d^-(D)$ prime that has a connected D–cut F. Then D is highly arc transitive.* → 76

He also found the following related condition. Note that there are one–ended graphs fulfilling the requirements of Theorem 4.38 e.g. the Diestel–Leader graph DL_{d^-,d^+}.

Theorem 4.38 *Let D be a connected 1–arc transitive digraph with prime out–valency $d^+ \in \mathbb{P}$ and in–valency $1 \leq d^- < d^+$. Then D is highly arc transitive.* → 76

Theorem 4.37 motivated *Norbert Seifter* to conjecture:

Conjecture 4.39 (Seifter) *A connected, locally finite 2–arc transitive digraph with more than one end is highly arc transitive.*

This conjecture was disproved by *Sònia Mansilla* in [11] by finding a family of sharply 2–arc transitive digraphs. There she stated without proof.

Theorem 4.40 *Let Γ_n be the digraph defined by*

$$\begin{aligned} V(\Gamma_n) &= \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z} \\ E(\Gamma_n) &= \{((i,j,k),(j,i,k+1)),((i,j,k),(j,i+1,k+1)) \mid (i,j,k) \in V(\Gamma_n)\} \end{aligned}$$

for $n \geq 3$. Then Γ_n is a connected 2–regular sharply 2–arc transitive digraph with Property Z. → 76

It was up to the author to recognize that Γ_n can be gained from $\mathrm{Tube}(n, 2)$ by replacing the $K_{n,n}$s with alternating cycles. From there it is straight forward to conjecture that the same construction with a $\mathrm{Tube}(n, k)$ yields a sharply k–arc transitive digraph. In that sense Theorem 4.37 is best possible.

4.2.6. C–homogeneous digraphs

Hamann and *Hundertmark* recently classified the C–homogeneous digraphs with more than one end. In doing so they recognized that some of them are highly arc transitive.

Theorem 4.41 *A connected C–homogeneous digraph with more than one end that does not contain a triangle is highly arc transitive.* → 76

Theorem 4.42 *A connected C–homogeneous digraph of Type II with more than one end does not contain a triangle. Thus it is highly arc transitive.* → 76

4.2.7. Highly arc transitive digraphs and topological groups

In this section we will state four results which describe the connection between topological groups and highly arc transitive digraphs or which can be proven using highly arc transitive digraphs. Indeed *Willis*' Theorem can be understood in terms of automorphism groups of graphs. We will state it first.

Theorem 4.43 (Willis) *Let G be a locally compact, totally disconnected group and $g \in G$. Then $U < G$ is tidy if and only if*

$$s(g) = \left[U : U \cap g^{-1}Ug\right]$$

where $s(g)$ is the scale function of g. → 76

We summarize the connection between topological groups and highly arc transitive digraphs in the following theorem.

Theorem 4.44 *Let G be a locally compact, totally disconnected group, $g \in G$ an element and $U < G$ compact and open. Let $\Omega = G/U$. Let $v_0 \in \Omega$ be a point and define $v_i = g^i(v_0)$. Set $e = (v_0, v_1) \in \Omega^2$. Consider the digraph D with*

$$\begin{aligned} V(D) &:= \Omega \\ E(D) &:= Ge \subset \Omega^2. \end{aligned}$$

1. *If $U = U_+U_- = U_-U_+$ then D is highly arc transitive.*
2. *If U is tidy for g then $v_0^{\Rightarrow}$ is a tree.* → 76

Finally we state two theorems which can be proven using highly arc transitive digraphs.

Theorem 4.45 *Let G be a totally disconnected, locally compact group with scale function $s : G \to \mathbb{R}$. If g is an FC^- element in G then*

$$s(g) = 1 = s(g^{-1}).$$

→ 76

Theorem 4.46 *Let G be a totally disconnected, locally compact group and $P(G)$ the set of periodic elements in G. Then $P(G)$ is closed in G.* → 76

4.2.8. Notes on the Cameron–Praeger–Wormald–Conjecture

We will have a look at Conjecture 3.36 in this section. Therefore we will start with quoting a useful result from [8].

Definition 4.47 (boundary) *Let D be a digraph and $C \subset V(D)$. The* **boundary** *∂C of C is the set vertices in $V(D) \setminus C$ which are adjacent to a vertex in C*

$$\partial C := \Big(\bigcup_{x \in C} N(x)\Big) \setminus C.$$

Definition 4.48 (strip) *A strip is a graph X which contains a connected set $C \subset V(X)$ such that there exists an automorphism $\alpha \in \mathrm{Aut}(X)$ such that $0 < |\partial C| < \infty$, $\alpha(C \cup \partial C) \subseteq C$ and $|C \setminus \alpha(C)| < \infty$.*

Remark 4.49 *We are interested in locally finite, connected, highly arc transitive digraphs D with Property Z and finite fibres. Let $\phi : D \to Z$ satisfy Property Z. Take $C = \phi^{-1}(\{n \mid n > 0\})$ then $\partial C = \phi^{-1}(0)$ is finite. By highly arc transitivity there is an automorphism that takes an vertex from $\phi^{-1}(0)$ into $\phi^{-1}(1)$ and by Property Z it takes $C \cup \partial C$ into C. Hence the graphs we are interested in are strips. Moreover, they are transitive and thus we are going to apply the upcoming proposition.*

Proposition 4.50 (Imrich, Seifter) *A transitive, locally finite strip X is spanned by finitely many disjoint lines. To these paths there exists an $\alpha \in \mathrm{Aut}(X)$ of infinite order leaving these lines invariant.*

Additionally, we will need a tiny result about the degree of such a graph.

Lemma 4.51 *If D is a connected, locally finite, highly arc transitive digraph with Property Z, and $\phi : D \to Z$ is a digraph epimorphism such that the inverse image $\phi^{-1}(0)$ of 0 is finite, then $d^+(D) = d^-(D)$.*

Proof
All the fibres have the same size r (this is obvious by Property Z and transitivity). By transitivity we have that all the in–degrees are equal (d^- say) and vice versa all the out–degrees are d^+. Then there are $r \cdot d^+$ edges starting in $\phi^{-1}(n)$ and $r \cdot d^-$ edges terminating in $\phi^{-1}(n+1)$. But since these sets are equal we have $r \cdot d^- = r \cdot d^+$. □

Remark 4.52 *We could prove this lemma alternatively as corollary of Theorem 4.19*

First, we derive some technical lemmas.

Lemma 4.53 (join and meet) *If D is a connected, highly arc transitive digraph with Property Z, and $\phi : D \to Z$ is a digraph epimorphism such that the inverse image $\phi^{-1}(0)$ of 0 is finite and $x, y \in \phi^{-1}(0)$ then x and y have a common descendant (i.e. $x^{\Rightarrow} \cap y^{\Rightarrow} \neq \emptyset$) and predecessor (i.e. ${}^{\Rightarrow}x \cap {}^{\Rightarrow}y \neq \emptyset$).*

Proof
By Proposition 4.50 D is spanned by finitely many disjoint lines. Every $x \in \phi^{-1}(0)$ is contained in exactly one of these lines. We show that there is a directed path from any $x \in \phi^{-1}(0)$ to every line.
We assume otherwise, that there is a set M of lines which cannot be reached from x with a directed path. Then there are no edges from $V(D) \setminus M$ to M since otherwise by Proposition 4.50 there would be such edges arbitrarily far in positive direction and thus there would be a path from x that extends to M what we chose not to be the case. Thus there are edges from M to $V(D) \setminus M$ since otherwise the graph would not be connected. We consider such an edge from $M \cap \phi^{-1}(a)$ to $(V(D) \setminus M) \cap \phi^{-1}(a+1)$. By Lemma 4.51 there is an in- and out–degree d thus there are $|M| \cdot d$ edges leaving $M \cap \phi^{-1}(a)$. Because of Property Z and the fact that there are no edges from $V(D) \setminus M$ to M, these are the only edges that could end in $M \cap \phi^{-1}(a+1)$. But one of them does not do so. Thus there are at most $|M| \cdot d - 1$ edges terminating in $M \cap \phi^{-1}(a+1)$ contradicting that all the vertices there have in–degree d. Thus M is empty.
We can prove the result for the predecessors analogously. □

Lemma 4.54 (exhaustion) *If D is a connected, highly arc transitive digraph with Property Z, and $\phi : D \to Z$ is a digraph epimorphism such that the inverse image $\phi^{-1}(0)$ of 0 is finite and $x \in \phi^{-1}(0)$ then there is an $n \in V(Z)$ such that $\phi^{-1}(n) \subset x^{\Rightarrow}$.*

Proof
We recognize in the proof above, that once we reache a line we can keep on running on it. Thus once we reache the last line we reache a neighbourhood of the sink end that is completely contained in $x^{\Rightarrow}$. □

The same results are obviously true for $\phi^{-1}(-n)$ and ${}^{\Rightarrow}x$.
A simple counting argument yields Conjecture 3.36 for prime fibre–sizes:

Proposition 4.55 (prime case) *If D is a connected, highly arc transitive digraph in $\mathcal{D}(\Delta)$ with Property Z, and $\phi : D \to Z$ is a digraph epimorphism such that $|\phi^{-1}(0)| = p$ is a prime, then $D = Z \otimes K_p$ and thus $\Delta = K_{p,p}$.*

Proof
Choose $x \in \phi^{-1}(0)$. By Lemma 4.54 there is an $n \in N$ such that $\phi^{-1}(n) \subset x^{\Rightarrow}$. Let n be minimal with that property. Considering the out–degree d there are exactly d^n n–arcs starting in x all of which terminate in $\phi^{-1}(n)$. We chose n in a way, that in every vertex in $\phi^{-1}(n)$ at least one of these arcs terminates. Because D is highly arc transitive, there must terminate equally many of these n–arcs in every vertex. But then d must equal p because otherwise $|\phi^{-1}(n)| = p$ is not a prime factor of d and thus no divisor of d^n. □

Corollary 4.56 (valency) *If D is a connected, highly arc transitive digraph in $\mathcal{D}(\Delta)$ with Property Z, and $\phi : D \to Z$ is a digraph epimorphism such that $|\phi^{-1}(0)| = r$, then the out–valency (respectively in–valency) d must contain all the prime factors of r.*

Proof
Let otherwise be p a prime factor of r that does not divide d. In the argument form the proof above we find that p does not divide d^n and so r cannot divide d^n. □

Let us summarize the situation. If D is a connected, highly arc transitive digraph with Property Z and in- and out–degree d, fibre–size $|\phi^{-1}(0)| = r$ and $\Delta = \Delta^+ \cup \Delta^-$ with partitionsize $|\Delta^+| = |\Delta^-| = n$.
We know

$$\begin{aligned} d \le n &\le r \\ \forall p \in \mathbb{P} : p \mid r \;\Rightarrow\; p &\mid d \\ n &\mid r \end{aligned}$$

The graphs from Construction 2.10 show that $n < r$ is possible. Conjecture 3.36 says that $d = n$. However, we do not even have $d \mid n$.
If r is a prime, we have $d = n = r$. But if only $r = p^2$, then the above does not exclude $n = p^2$, $d = p$ as $\mathrm{Tube}(p, 2)$ illustrates.

4.3. Properties

In this subsection we investigate the properties of the highly arc transitive digraphs we defined in Section 2.

4.3.1. The integer line Z

The integer line is the most trivial case of an highly arc transitive digraph.

Proposition 4.57 *Let Z be the line as in Definition 2.1.*

1. *Z has Property Z.*
2. *$Z \in \mathcal{D}(\bullet\!\!\rightarrow\!\!\bullet)$ and thus the reachability relation is not universal on Z.*
3. *Z is locally finite with $d^- = d^+ = 1$.*
4. *Z has out–spread $s^+ = 1$ and in–spread $s^- = 1$.*
5. *Z has two thin ends.*
6. *$Z = \mathrm{Cay}((\mathbb{Z}, +), 1)$ is a Cayley–graph.* → 66

4.3.2. Trees

The automorphism groups of trees can act on them with maximal freedom because they contain no cycles. Therefore trees are the ideal candidates not just for being highly arc transitive but also for the substructure of highly arc transitive digraphs. The universal covering digraph, the Evans-graph, the Diestel–Leader graph, the DeVos–Mohar–Šámal–digraph and the Hamann–Hundertmark–digraph are all constructed using trees. The substructure of the alternating–cycle digraph is a tree as well and since Z is a tree also the $K_{n,n}$–tubes come from a tree. Just the sequences digraphs and the ordered field digraph are far away from being trees.
The properties of regular trees are almost as easy as the ones of Z. Before we look at them we need to define **alternating trees**:

Definition 4.58 (Alternating Tree) *The* **alternating tree** *$AT(n, m)$ is the bipartite tree with in–valency n for every vertex in the sink–partition and out–valency m for every vertex in the source–partition.*

Proposition 4.59 *Let T be a regular directed tree with in–valency $d^-(T) > 0$ and out–valency $d^+(T) > 0$.*

1. *T has Property Z.*
2. *T has as associated digraph the alternating tree $AT(d^-(T), d^+(T))$ and thus the reachability relation on T is not universal.*
3. *The in- and out–spread of T correspond to its valencies.*
4. *If $T \not\cong Z$ then T has at least (depending on the valencies) continuum many thin ends.*
5. *T is a Cayley–graph if and only if $d^-(T) = d^+(T)$.* → 66

4.3.3. $K_{n,n}$–tubes

$K_{n,n}$–tubes were first considered by *McKay* and *Praeger* to show that there are indeed graphs satisfying the Cameron–Praeger–Wormald–Conjecture i.e. have two thin ends, Property Z (with ϕ) and have associated digraph $K_{n,n}$ with $n \neq |\phi^{-1}(0)|$. Later, they were rediscovered as an example that showed that two vertices $x, y \in \phi^{-1}(a)$ can meet arbitrary late i.e. that the distance $d(x, x^{\rightrightarrows} \cap y^{\rightrightarrows})$ (respectively $d(y, x^{\rightrightarrows} \cap y^{\rightrightarrows})$) can be arbitrary large. E.g. for $n \geq 2$ in $\mathrm{Tube}(n, m)$ these distances can be up to

$$\begin{aligned} m &= d((0,(0\dots 0)),(0,(0\dots 0))^{\rightrightarrows} \cap (0,(1\dots 1))^{\rightrightarrows}) \\ &= d((0,(1\dots 1)),(0,(0\dots 0))^{\rightrightarrows} \cap (0,(1\dots 1))^{\rightrightarrows}). \end{aligned}$$

Proposition 4.60 *Let* $\mathrm{Tube}(n, m)$ *be as in Definition 2.11.*

1. $\mathrm{Tube}(n, m)$ *has Property Z with finite fibres.*
2. $\mathrm{Tube}(n, m) \in \mathcal{D}(K_{n,n})$ *and thus the reachability relation on* $\mathrm{Tube}(n, m)$ *is not universal.*
3. $\mathrm{Tube}(n, m)$ *is locally finite with in- and out–valency n and has in- and out–spread* 1.
4. $\mathrm{Tube}(n, m)$ *has two thin ends.* → 67

4.3.4. $\mathrm{Tube}(n, m) \otimes K_k$

In the $\mathrm{Tube}(n, m)$ the different $\mathcal{A}(e)$ have at most one vertex in common. We can blow up these intersections arbitrarily by tensoring. The result is again a highly arc transitive digraph with two ends. The author conjectures that all highly arc transitive, two–ended digraphs with Property Z are of the form $\mathrm{Tube}(n, m) \otimes K_k$.

Proposition 4.61 *Let* $\mathrm{Tube}(n, m)$ *be as in Definition 2.11.*

1. $\mathrm{Tube}(n, m) \otimes K_k$ *has Property Z with finite fibres.*
2. $\mathrm{Tube}(n, m) \otimes K_k$ *has associated digraph* $K_{nk,nk}$ *and thus the reachability relation on* $\mathrm{Tube}(n, m) \otimes K_k$ *is not universal.*
3. $\mathrm{Tube}(n, m) \otimes K_k$ *is locally finite with in- and out–valency* nk *and has in- and out–spread* 1.
4. $\mathrm{Tube}(n, m) \otimes K_k$ *has two thin ends.* → 67

4.3.5. $DL(\Delta)$

The universal covering digraph was invented in [1]. It is a projective object in the category of 1–arc transitive digraphs which contain arcs of arbitrary length.

Proposition 4.62 *Let* $DL(\Delta)$ *be the universal covering digraph as in Definition 2.18. Let* δ^- *be in the source–partition and* δ^+ *in the sink–partition of* Δ.

1. $DL(\Delta)$ *has Property* Z.
2. $DL(\Delta) \in \mathcal{D}(\Delta)$ *and thus the reachability relation on* $DL(\Delta)$ *is not universal.*
3. $DL(\Delta)$ *takes the valencies* $d^+(DL(\Delta)) = d^+(\delta^-)$ *and* $d^-(DL(\Delta)) = d^-(\delta^+)$ *from* Δ *and thus is locally finite if and only if* Δ *is.*
4. $DL(\Delta)$ *has in–spread* $d^-(\Delta)$ *and out–spread* $d^+(\Delta)$.
5. *If* $\Delta \neq \bullet\!\!\rightarrow\!\!\bullet$ *then* $DL(\Delta)$ *has at least continuum many ends. If* Δ *is finite these are all thin.*
6. *Depending on* Δ *there are* $DL(\Delta)$*s which are Cayley–graphs and such that are not.* → 68

4.3.6. Ordered field digraph

The ordered field digraph is one of the few known highly arc transitive digraphs which is constructed without the use of trees.

Proposition 4.63 *Let* D *be the ordered field digraph as in Proposition 2.21.*

1. D *has universal reachability relation. Thus it does not have Property Z and it is its own associated digraph.*
2. D *is not locally finite.*
3. D *has one thick end.*
4. D *is a Cayley–graph.* → 68

4.3.7. Alternating–cycle digraph

The alternating–cycle digraph was the first known locally finite, highly arc transitive digraph without Property Z. Its reachability relation is not universal, thus it is one of the few known highly arc transitive digraphs in between these properties.

Proposition 4.64 *Let* $\mathrm{AltCyc}(n)$ *be as in Definition 2.23.*

1. $\mathrm{AltCyc}(n)$ *has neither Property* Z *nor universal reachability relation.*
2. $\mathrm{AltCyc}(n)$ *is locally finite with in- and out–valency* 2 *and in- and out–spread* 2*. Indeed,* $\langle x^{\Rightarrow} \cup \{x\}\rangle$ *is a rooted binary tree for every* $x \in V(\mathrm{AltCyc}(n))$*.*
3. $\mathrm{AltCyc}(n) \in \mathcal{D}(\mathrm{AC}(n))$*.*
4. $\mathrm{AltCyc}(n)$ *is a Cayley–graph.* → 68

4.3.8. Evans–graph

The Evans–graph was constructed as an example of a highly arc transitive digraph with finite out–spread and without Property Z. Its reachability relation is even universal. Only the in–valency is infinite.

Proposition 4.65 *Let* X *be the Evans–graph as in Definition 2.27 with* $n > 1$*.*

1. X *has universal reachability relation and thus it is its own associated digraph and does not have Property* Z*.*
2. X *is not locally finite, but has finite out–valency* n *and out–spread* n*. Indeed* $x^{\Rightarrow}$ *is a rooted* n*–out–valent tree for every vertex* $x \in V(X)$*.*
3. X *has one thick end.*
4. X *is not a Cayley–graph.* → 69

4.3.9. DeVos–Mohar–Šámal–digraph

The DeVos–Mohar–Šámal–digraph is a locally finite digraph with universal reachability relation. Its construction yields digraphs with universal reachability relation for all valencies which do not exclude it ($d^+ = d^-$ not prime).

Proposition 4.66 *Let* $\mathrm{DMS}(n,m)$ *be the DeVos–Mohar–Šámal–digraph as defined in 2.37.*

1. $\mathrm{DMS}(n,m)$ *has universal reachability relation and thus is its own associated digraph and does not have Property* Z*.*
2. $\mathrm{DMS}(n,m)$ *is locally finite with in- and out–valency* $(n-1)(m-1)$*.*
3. $\mathrm{DMS}(n,m)$ *has infinitely many thin ends.* → 69

4.3.10. Hamann–Hundertmark–digraph

The Hamann–Hundertmark–digraphs appear as a class of digraphs in the characterization of C–homogeneous digraphs that happens to be highly arc transitive. After the alternating–cycle digraph it is only the second known class of digraphs between Property Z and universal reachability relation. Its associated digraph is the complement of a perfect matching (a set of disjoint edges which cover the entire vertex set).

Definition 4.67 *Let PM be a perfect matching in $K_{c,c}$ where c is a cardinal. We define CP_c as the subgraph of $K_{c,c}$ which misses the edges of PM.*

Proposition 4.68 *Let $\mathrm{HH}(\kappa, n)$ be the Hamann–Hundertmark–digraph as in Definition 2.40.*

1. $\mathrm{HH}(\kappa, n)$ *does not have Property Z.*
2. $\mathrm{HH}(\kappa, n)$ *has associated digraph CP_κ and thus its reachability relation is not universal.*
3. $\mathrm{HH}(\kappa, n)$ *has in- and out–valency $\kappa - 1$, thus it is locally finite if κ is finite.*
4. $\mathrm{HH}(\kappa, n)$ *has infinitely many thin ends.* $\mathrm{HH}(\kappa, n)$ *has also thick ends, if κ is infinite.* → 69

4.3.11. $Z \otimes S(\Delta, \delta_1, \delta_2)$

The tensor–product of Z and a sequences digraph was the first highly arc transitive digraph constructed by the authors of [1]. Its construction is one of the rare ones that does not use trees.

Proposition 4.69 *Let $S(\Delta, \delta_1, \delta_2)$ be the sequences digraph as in Definition 2.48.*

1. *$Z \otimes S(\Delta, \delta_1, \delta_2)$ has Property Z and thus the reachability relation is not universal on it.*
2. *$Z \otimes S(\Delta, \delta_1, \delta_2)$ has associated digraph Δ.*
3. *$Z \otimes S(\Delta, \delta_1, \delta_2)$ is locally finite. It has in–valency $d^-(\Delta)$ and out–valency $d^+(\Delta)$.* → 70

In Example 2.51 we saw that $Z \otimes S(\Delta, \delta_1, \delta_2)$ can be a tree (in which case the answer is trivial) or not.

4.3.12. Diestel–Leader graph

The Diestel–Leader graphs are well known from research in different areas. It is locally finite but has only one end.

Proposition 4.70 *Let $DL_{n,m}$ be the Diestel–Leader graph as in Definition 2.55.*

1. *$DL_{n,m}$ has Property Z and thus the reachability relation is not universal on it.*
2. *$DL_{n,m}$ is locally finite with in–valency and in–spread m and out–valency and out–spread n.*
3. *$DL_{n,m}$ has $K_{m,n}$ as associated digraph.*
4. *$DL_{n,m}$ has one thick end.* → 70

4.3.13. Pancake–tree

The pancake–trees are special universal covering digraphs which were considered in [2] because they have thin and thick ends.

Proposition 4.71 *For the hexagon pancake–tree and the quadratic pancake–tree we have*

1. *The pancake–tree has Property Z.*
2. *The pancake–tree has the pancake as associated digraph and its reachability relation is not universal.*
3. *The pancake–tree is locally finite with in- and out–valency and in- and out–spread either all 3 or 4. Indeed $x^{\Rightarrow}$ is either a 3- or 4–out–valent tree.*
4. *The pancake–tree has both infinitely many thin and infinitely many thick ends.*
5. *The pancake–tree is a Cayley–graph.* → 71

4.3.14. Summary of Properties

Table 1 shows a summary of which graphs have which properties. With ∞ we denote an appropriate infinite cardinal, n, m, k, d^- and d^+ denote positive integers the latter the in- and out–valencies. With κ we mean an arbitrary cardinal greater or equal 3 and $\Delta = \Delta^- \cup \Delta^+$ is as usual a 1–arc transitive, bipartite, connected digraph. Finally $\delta^- \in \Delta^-$ and $\delta^+ \in \Delta^+$.
Empty cells mean that the property is not determined (e.g. not all trees are locally finite). Dots mean that the cell has not yet been dealt.

	Property Z	Universal reachability relation	locally finite	out–valency	in–valency	out–spread	in–spread	Ends	has thin ends	has thick ends	Associated Digraph	is a Cayley–Graph
Z	✓	✗	✓	1	1	1	1	2	✓	✗	$\bullet \to \bullet$	✓
Tree (not Z)	✓	✗		d^+	d^-	d^+	d^-	∞	✓	✗	$\mathrm{AT}(d^-, d^+)$	
$\mathrm{Tube}(n, m)$	✓	✗	✓	n	n	1	1	2	✓	✗	$K_{n,n}$	$\cdots$
$\mathrm{Tube}(n, m) \otimes K_k$	✓	✗	✓	nk	nk	1	1	2	✓	✗	$K_{nk,nk}$	$\cdots$
$DL(\Delta)$ (not Z)	✓	✗		$d^+(\delta^-)$	$d^-(\delta 1+)$	$d^+(\delta^-)$	$d^-(\delta^+)$	∞	✓		Δ	
Ordered field digraph	✗	✓	✗	∞	∞	∞	∞	1	✗	✓	itself	✓
$\mathrm{AltCyc}(n)$	✗	✗	✓	2	2	2	2	∞	✓	✗	$\mathrm{AC}(n)$	✓
Evans–graph	✗	✓	✗	n	∞	n	∞	1	✗	✓	itself	✗
$\mathrm{DMS}(n+1, m+1)$	✗	✓	✓	nm	nm	$\cdots$	$\cdots$	∞	✓	✗	itself	$\cdots$
$\mathrm{HH}(\kappa, n)$	✗	✗		$\kappa - 1$	$\kappa - 1$	$\cdots$	$\cdots$	∞	✓		CP_κ	$\cdots$
$Z \otimes S(\Delta, \delta^-, \delta^+)$	✓	✗		$d^+(\delta^-)$	$d^-(\delta^+)$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	Δ	$\cdots$
$DL_{n,m}$	✓	✗	✓	n	m	n	m	1	✗	✓	$K_{m,n}$	
q. Pancake–tree	✓	✗	✓	4	4	4	4	∞	✓	✓	q. pancake	✓
h. Pancake–tree	✓	✗	✓	3	3	3	3	∞	✓	✓	h. pancake	✓

Table 1: Highly arc transitive digraphs and their properties

4.4. Conjectures and open Questions

First we mention, that the Cameron–Praeger–Wormald–Conjecture 3.36 is still open.

Conjecture 4.72 (Cameron–Praeger–Wormald) *If D is a connected highly arc transitive digraph in $\mathcal{D}(\Delta)$ with Property Z, and $\phi : D \to Z$ is a digraph epimorphism such that the inverse image $\phi^{-1}(0)$ of 0 is finite, then Δ is a complete bipartite digraph.*

From Section 2 we know $\mathrm{Tube}(n, m)$ as an example for digraphs with complete bipartite associated digraph. Also from Section 2 we know about tensor–products. We recognize that $\mathrm{Tube}(n, m) \otimes K_k$ has the same property.

Conjecture 4.73 *Let $D \in \mathcal{D}(\Delta)$ be a connected, highly arc transitive digraph with Property Z, and let $\phi : D \to Z$ be a digraph epimorphism such that the inverse image $\phi^{-1}(0)$ of 0 is finite. Then D is of the form $D \cong \mathrm{Tube}(n, m) \otimes K_k$.*

The Questions 3.30 is still open. We also restate it here.

Question 4.74 (2.10) *Does a covering projection of a connected locally finite digraph have to be an isomorphism?*

We state two sets of new canonical questions.

Question 4.75 *Clarify the properties of the known highly arc transitive digraphs. That is fill in the unknown cells in Table 1. E.g. it would be interesting under which circumstances $Z \otimes S(\Delta, \delta_1, \delta_2)$ is a Cayley–graph.*

Question 4.76 *Section 2 presents some constructions to gain new highly arc transitive digraphs from given ones: Line digraph, Tensor–products, k–arc–digraphs and s–arc–k–arc–digraphs (all these were already known to Cameron, Praeger and Wormald).*

1. *Under which circumstances do these constructions yield new highly arc transitive digraphs? E.g. the tensor–product of two $K_{n,n}$–lines is a graph that consists of infinitely many copies of a different $K_{n,n}$–line; that should not be considered new.*

2. *Which of the properties listed in Table 1 are preserved by which of these constructions under which circumstances?*

Furthermore the author recognized an interesting sequence of proper subgraphs and asks:

Question 4.77 *If we choose the κ in the definition of the Hamann–Hundertmark–digraph* $\mathrm{HH}(\kappa, n)$ *finite, we can assign a second set of labels to the edges which represent cyclic orders with respect to the κ–valent partition. If we put the same condition for the first two edges of the* 3*–path with these labels as we did for the last two edges, we end up with a disconnected digraph that consists of infinitely many copies of Z which cover the vertex set of* HH*. We denote this digraph by ∞Z. Consider the sequence of proper subgraphs*

$$\infty Z \subset \mathrm{HH}(n, m) \subset \mathrm{DMS}(n, m).$$

All these digraphs are highly arc transitive and have the same vertex set. The sequence starts with ∞Z having Property Z, continues with $\mathrm{HH}(n, n)$ *having neither Property Z nor universal reachability relation, ending with* $\mathrm{DMS}(n, n)$ *having universal reachability relation. Thus it is a sequence running from the one extreme to the other.*
Are there more such or similar sequences? Where do these sequences come from and is there a pattern to generate them?

5. Proofs

In this section we present details on some statements claimed so far. Since the main goal of this thesis is to collect as many facts as possible about highly arc transitive digraphs, the amount of statements presented is quite huge. Thus proofs are not provided for all of them. But for completeness for all of them at least a reference is given where a proof can be found.

5.1. Proofs for highly arc transitivity

Proof of Proposition 2.2
This follows directly from the definition. □

Proof of Proposition 2.4
The automorphism group of T is generated by two kinds of operations:

- The shift along an arbitrary but fixed line L. This is one generator.
- The permutations of in–subtrees and out–subtrees at an arbitrary but fixed vertex $x \in L$. These are as many generators as you need to generate the S_n and the S_m.

Start with a s–arc a that contains x. We first consider the part of a which is contained in $x^{\Rightarrow}$ and permute it into the subtree containing a part of L. Then we shift downwards and permute again. By iteration we get the whole "upper" part of the arc a into L. We than shift back to x and perform the same procedure in the other direction. We end up with that s–arc a lying on L starting in x. Now assume that a doses not contain x. Then it is contained in a subtree which can be permuted into the subtree containing a part of L. By a shift the distance between x and the a can be reduced. Since the distance must be finite, by iteration at one point the arc a will contain x and we can perform as above. Thus any s–arc can be mapped by an automorphism to the s–arc contained in L starting in x. Thus the automorphism group acts transitively on Arc_s. □

Figure 30 shows the generators of the automorphism group of a regular tree with $d^+ = d^- = 2$. On the left the shift along a line is shown. The green edges are taken to the green edges and take their subtrees with them. The blue edges do just the same. On the right the "backward"–permutations will flip the green edges and the "forward"–permutations will flip the blue edges.

Proof of Proposition 2.7
This follows from Proposition 2.13. □

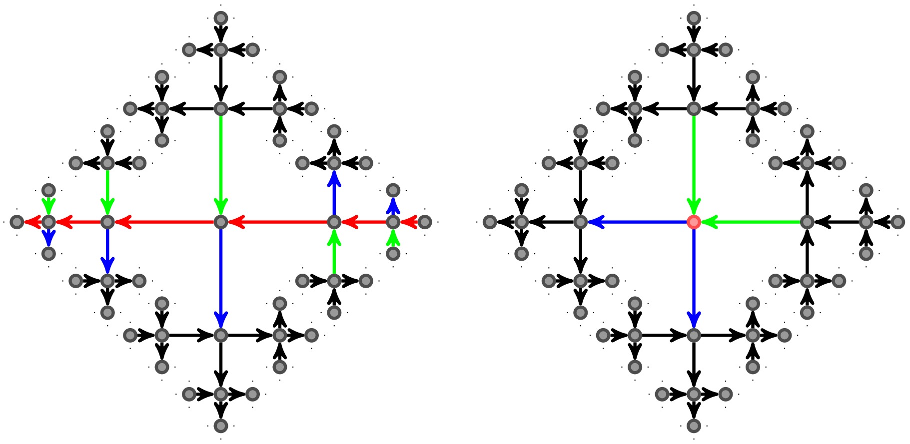

Figure 30: generators of the automorphism group of a tree

Proof of Proposition 2.13
First we notice, that $\mathrm{Tube}(n, m)$ is transitive by circularly permuting the digits and digit values of the labels. Then we observe that the stabilizer $\mathrm{Stab}_{\mathrm{Aut}(\mathrm{Tube}(n,m))}{}^{\Rightarrow}x$ acts transitively on the out–neighbours $x^{\Rightarrow 1}$ of x for every $x \in V(\mathrm{Tube}(n, m))$ (this is just by label–permutation in the entire "right–half" of $\mathrm{Tube}(n, m)$).
We can shift every initial vertex of any arc to the initial vertex of any other arc by transitivity. And then we can inductively flip the edges of the one arc onto the other and simultaneously keeping the previous edges on it by the above property. □

Proof of Proposition 2.16
A proof can be found in [1] Lemma 4.1 (a) on page 389. □

Proof of Proposition 2.19

1. This is mentioned but not proved in [1] between Definition 2.1. and Theorem 2.2 on page 380. But it follows pretty straight forward from the 1–arc transitivity of Δ and the tree–structure earned from the underlying tree.

2. The universal covering digraph $DL(\Delta)$ lies in the class $\mathcal{D}(\Delta)$. Thus we apply Theorem 4.15 (2) to get $\phi : DL(\Delta) \to DL(\Delta)$ that takes any s–arc to an arbitrary s-arc. Since every vertex separates $DL(\Delta)$ its cycles must stay in one $\mathcal{A}(e)$. Hence they are all alternating. Using Lemma 4.3 it follows that ϕ is an isomorphism.
   Details can be found in [1] Theorem 2.3. □

Proof of Proposition 2.21
Every arc in the ordered field digraph consists of strictly increasing vertices. Thus it can be thought of as a finite sequence of strictly increasing elements of F. But any order preserving map from a finite sequence of strictly increasing elements of F onto another sequence of the same size can be extended to an order preserving bijection from F onto itself. But that immediately indicates an automorphism of the ordered field digraph. Hence its automorphism group acts highly arc transitive on it. $\square$

Proof of Proposition 2.24
The proof is provided in [7]. $\square$

Proof of Proposition 2.28
A proof is given in [6] Theorem 2.6 on page 238. $\square$

Proof of Proposition 2.31
A proof can be found in [1] Lemma 4.1 (b) on page 389. $\square$

Proof of Proposition 2.34
A proof can be found in [1] Lemma 4.1 (c) on page 389. $\square$

Proof of Proposition 2.38
The proof is provided in [5]. $\square$

Proof of Proposition 2.41
The Hamann–Hundertmark–digraph is connected and C–homogeneous of Type II. A proof for that can be found in [13] Theorem 7.6 on pages 19 to 21. Thus by Theorem 4.42 it is highly arc transitive. $\square$

Proof of Proposition 2.43
A proof can be found in [1] Lemma 4.3 (a) on page 390. $\square$

Proof of Proposition 4.23
A proof can be found in [1] Lemma 4.3 (b) on page 390. $\square$

Proof of Proposition 2.44
A proof can be found in [1] Theorem 4.5 on page 391. $\square$

Proof of Proposition 2.49
A proof can be found in [1] Theorem 4.8 on page 393. We include a proof for the connectedness, because the corresponding argument in [1] is wrong.
If $S(\Delta, \delta_1, \delta_2)$ has a loop at the "zero–sequence" $\mathbf{0}$:=$(\dots, \delta_1, \delta_1, \delta_2, \delta_2, \dots)$ then

$Z \otimes S(\Delta, \delta_1, \delta_2)$ will contain a line

$$L{:=}(l_i)_i = (\dots, (-1, \mathbf{0}), (0, \mathbf{0}), (1, \mathbf{0}), \dots).$$

Otherwise we have to construct a double–ray R which contains the vertices of L as subsequence. For it is sufficient to construct walks from l_i to l_{i+1}. We denote general entries on the negative half of the sequences with $\bullet_j$ and on the non–negative half with $*_k$. Every edge $(\bullet_j, *_k)$ of Δ induces amongst others an edge

$$((i, (\dots, \delta_1, \bullet_j, \delta_2, \delta_2, \dots)), (i+1, (\dots, \delta_1, \delta_1, *_k, \delta_2, \dots)))$$

in $Z \otimes S(\Delta, \delta_1, \delta_2)$. For that reason it is also true that $Z \otimes S(\Delta, \delta_1, \delta_2) \in \mathcal{D}(\Delta)$. Since Δ is connected we find an alternating walk from δ_1 to δ_2 starting and ending with an forward edge. This alternating walk indicates an alternating walk

$$((i, (\dots, \delta_1, \bullet_j, \delta_2, \delta_2, \dots)), \dots, (i+1, (\dots, \delta_1, \delta_1, *_k, \delta_2, \dots)))$$

in $Z \otimes S(\Delta, \delta_1, \delta_2)$. The concatenation of all these walks yields our desired R. We prove that from every vertex in $Z \otimes S(\Delta, \delta_1, \delta_2)$ there is a walk with terminal vertex l_i for some i. For we consider an arbitrary vertex v which has the form

$$v = (i, (\dots, \delta_1, \bullet_{-n}, \dots, \bullet_{-1}, *_0, \dots, *_m, \delta_2, \dots)).$$

Using the right–shift as defined we find an arc to a vertex w coming from a sequence that is constant δ_1 on its negative half and thus has the form

$$w = (i+n, (\dots, \delta_1, *_{-n}, \dots, *_m, \delta_2, \dots)).$$

With the above argument we find a walk

$$((i{+}n, (\dots, \delta_1, *_{-n}, \dots, *_m, \delta_2, \dots)), \dots, (i{+}n{-}1, (\dots, \delta_1, *_{1-n}, \dots, *_m, \delta_2, \dots))).$$

Iteration results in a walk

$$(v, \dots, (i-m, (\dots, \delta_1, \delta_2, \dots)) = l_{i-m})$$

which terminates on R. □

Proof of Proposition 2.57
Constructions of the broom–graph can be found in [4] Example 1 on page 152 or [9] Example 4.10 on page 1579. However, the proof is not explicitly given in either of the papers.
Considering the defining trees of the Diestel–Leader graph, actions on them induce actions on $DL_{n,m}$. These actions generate a subgroup $G \subset \mathrm{Aut}(DL_{n,m})$ which acts highly arc transitively on $DL_{n,m}$. □

Proof of Proposition 2.64
The quadratic pancake–tree is the universal covering digraph of the quadratic pancake and thus it is highly arc transitive. □

Proof of Proposition 2.67
The hexagon pancake–tree is the universal covering digraph of the hex–pancake and thus it is highly arc transitive. □

5.2. Isomorphy–proofs

Proof of Theorem 2.53
A proof can be found in [1] Theorem 4.12. on pages 394 and 395. □

Proof of Theorem 2.61
We just give a sketch of the proof. Given a line L in the $B_{n,m}$ we gain trees $^{\Rightarrow}L$ and $L^{\Rightarrow}$ which will correspond to the trees used in the construction of the Diestel–Leader graph. We can map the broom–graph canonically to this trees (i.e. using the canonical embedding in $\mathbb{R}^3$) to gain coordinates. These coordinates correspond to the pairs in the horocycle construction of the Diestel–Leader graph. It immediately turns out that the edges of the broom graph are in the same positions as asked for the Diestel–Leader graph. □

5.3. Property–proofs

Proof of Proposition 4.57

1. Obvious with $\phi = \mathrm{id}$.
2. As the in- and out–valencies are both 1, no edge can be equivalent to any edge but itself.

3. – 6. are immediate from the definitions. □

Proof of Proposition 4.59

1. T has no cycles. Hence all cycles are balanced. Thus T has Property Z by Lemma 4.4.
2. Since T has Property Z with ϕ, ϕ maps Δ to a single edge of Z

   $$\phi(\Delta) = \bullet\!\!\rightarrow\!\!\bullet = (\{i, i+1\}, \{(i, i+1)\})$$

   where $\phi^{-1}(i)$ is the source partition of Δ and $\phi^{-1}(i+1)$ the sink partition. Obviously the vertices in the source partition have out–valency d^+ and the vertices in the sink partition in–valency d^-. Since T is cycleless so is Δ. But that already forces Δ to be the alternating tree $\mathrm{AT}(d^-, d^+)$.

3. This again follows from the fact that T has no cycles.

4. Any two non–identical rays in T can be separated by a finite set of vertices, either a single vertex on the shortest path connecting the rays or the intersection of the rays. If T is locally finite, rays in T can be understood as infinite sequences of digits in $\mathbb{Z}_{d^+ + d^-}$ and thus there is obviously a bijection between the rays of T and the real interval $[0, 1]$.

5. Every Cayley–graph has equal in- and out–valency. On the other hand the tree with $d^- = d^+$ is the Cayley–graph of the free group on d^+ letters. □

Proof of Proposition 4.60

1. The $K_{n,n}$–tube $\mathrm{Tube}(n, m)$ is derived from a Property Z graph in a Property Z respecting way.

2. The $K_{n,n}$s arise directly from the construction.

3. Since the associated digraph is $K_{n,n}$, the valencies must be n. The spreads must be 1 since $\mathrm{Tube}(n, m)$ has Property Z with finite fibres.

4. Any fibre is a finitely separating set. Both components are infinite, thus there are at least two ends. Obviously the rays in either component cannot be separated as the fibres are finite and the components connected. Hence there are exactly two ends. These are thin because again the fibres are finite. □

Proof of Proposition 4.61

1. By Proposition 4.21 the tensor–product respects Property Z. The fibres stay the same (in the one component), therefore their sizes are just multiplied by the size of the second factor.

2. The vertices of the associated digraph must have vertices of the associated digraph of $\mathrm{Tube}(n, m)$ in the first entry. Since we tensor with K_k all the possible edges appear. Hence $K_{nk,nk}$s are generated.

3. The valencies are nk because the associated digraph is $K_{nk,nk}$. The fibres stay finite by tensoring, thus the spread must stay 1.

4. The argument is the same as in the proof of Proposition 4.60 (4). □

Proof of Proposition 4.62

1. That is Lemma 4.5.
2. By construction.
3. This follows immediately from (2).
4. All cycles in $DL(\Delta)$ are inside an $\mathcal{A}(e)$ for some edge e (that is by the structure inherited from the underlying tree). By the construction, all out–neighbours x_i of a vertex x have all their outgoing edges in a different $\mathcal{A}(e_i)$. Thus $x^{\Rightarrow}$ is a tree and out–spread and out–valency are equal. The same argument holds for the in–spread.
5. These are the ends of the underlying tree. As $DL(\Delta)$ is sharply 1–vertex–connected, they are thin. If Δ is finite no ray can stay in a Δ forever and $DL(\Delta)$ cannot contain any further ends.
6. $DL(\bullet\!\!\to\!\!\bullet) = Z = Cay((Z,+),\{1\})$ is a Cayley–graph. On the other hand in- and out–valency of the universal covering digraph are not necessarily equal (e.g. $DL(K_{1,2})$ is not a Cayley–graph). □

Proof of Proposition 4.63

1. All elements of F are comparable. Hence for any two different edges the initial vertex of the one is comparable with the terminal vertex of the other. It is possible, that the terminal vertex of one of the edges is the initial vertex of the other, but then the other terminal and initial vertex have to be different. Therefore there is an edge from an initial vertex of one to the terminal vertex of the other edge. Hence they lie on an alternating 3–path.
2. Any vertex is comparable with all the other infinitely many vertices and thus has an edge to all of them.
3. Any two vertices are adjacent. Hence rays cannot be separated.
4. The underlying field as additive group acts transitively and freely on the ordered field digraph. By Sabidussi's Theorem 1.38 the ordered field digraph is its Cayley–graph, namely the one generated by all non–zero elements. □

Proof of Proposition 4.64

1. The second relation indicates an unbalanced cycle, hence it does not have Property Z. Since it has in and out–valency 2 its reachability relation is not universal by Theorem 4.26.

2. The valencies follow directly from the definition of $\mathrm{AltCyc}(n)$ as Cayley–graph. For every $x \in V(\mathrm{AltCyc}(n))$ the descendants form a tree since otherwise there would be two arcs that meet twice. This would induce another relation in the presentation which is not there.

3. Every edge lies on an alternating cycle by construction. Every vertex has in- and out–valency 2. Hence at no vertex of the alternating cycle an alternating walk can leave the cycle. Thus Δ is the alternating cycle.

4. This follows directly from the definition. □

Proof of Proposition 4.65

1. For any two vertices x and y in the Evans–graph there is a vertex z that is independent of both. Note that x and y are not necessarily independent, thus the set $\{x, y, z\}$ is not guaranteed to be independent, but the sets $\{x, z\}$ and $\{y, z\}$ are. By construction a tree was attached at x and y in a way that the incoming edges have the same initial vertex. Actually infinitely many such trees were attached. The same holds for y and z. We get an alternating 4–walk from x to y. Hence, every pair of edges with terminal vertices x and y lies on an alternating 6–walk.

2. This follows immediately from the construction.

3. There are even infinitely many vertices independent from two arbitrary vertices. Thus they are connected by infinitely many disjoint 4–walks. So there is no way to separate rays finitely.

4. $d^+ \neq d^-$. □

Proof of Proposition 4.66

1. A proof can be found in [5].

2. This follows directly from the definition.

3. Since all the edges come from 3–arcs, every vertex in the underlying tree yields a finite separator of $\mathrm{DMS}(n, m)$ (namely the set of vertices which come from the incident edges). Hence there is a correspondence between the ends of the underlying tree and the ends of $\mathrm{DMS}(n, m)$. □

Proof of Proposition 4.68

1. For every $a \in A$ there is a surrounding cycle C of length $3n$ with 3 consecutive edges in an $\mathcal{A}(e)$. The first and last of these 3 edges must be forward say. The same holds for the first and last edge of the neighbouring $\mathcal{A}(e_i)$s, otherwise they would be within the same $\mathcal{A}(e)$. Thus C has $2n$ forward and n backward edges.

2. For $b \in B$ we have κ possible initial vertices and κ possible terminal vertices. Every initial vertex produces all but one edges, namely the one terminal vertex, which is its A–neighbour, is missing.

3. This follows from 2.

4. Since all the edges come form 3–arcs, every vertex with finite degree in the underlying tree yields a finite sepatator of $\mathrm{HH}(\kappa, n)$ (as in the above proof). Hence every end in the underlying tree yields an end of $\mathrm{HH}(\kappa, n)$. If κ is infinite there is a countable subset in the source–partition of $\mathrm{CP}(\kappa)$ which we can enumerate with $(0_-, 1_-, \dots)$. We enumerate the corresponding elements in the sink–partition with $(0_+, 1_+, \dots)$ (corresponding means adjacent in the underlying perfect matching). Then there are disjoint rays $(p_-, p_+^2, p_-^3, \dots)$ for all primes p. These rays are all in the same end (by infinitely many disjoint paths between any pair of them) and thus every $\mathrm{CP}(\kappa)$ contains a thick end.

Proof of Proposition 4.69

1. This follows from Proposition 4.21

2. A proof is claimed in [1] Theorem 4.8 (b). The argument is given in the proof of Proposition 2.49

3. This follows directly from 2. □

Proof of Proposition 4.70

1. This follows directly from the fact that the defining trees have Property Z.

2. In forward direction the digraph looks like the forward defining tree and thus takes its out–valency and out–spread. The same holds for the backward direction.

3. Considering the construction of the broom–graph and asking what is attached to $x^{\Rightarrow 1}$ this is immediate.

4. First we notice that a ray that leaves every bounded set in the Diestel–Leader graph has a ϕ–image that leaves every bounded set in Z. That implies that a ray either hits infinitely many horocycles that are ϕ–mapped

to positive integers or infinitely many to negative integers, or both. Any two vertices on the same horocycle can be connected with a path that does not hit more horocycles than the coordinates of the vertices are apart – thinking with the vertices of the defining trees as coordinates. That means that any two positive rays can be connected with another path, that stays in a bounded set, outside any bounded set. Thus there are infinitely many disjoint paths connecting the rays and thus they are in the same end. The same holds for negative rays. Thus we are left with finding a forward and a backward ray that cannot be separated finitely. But taking a line and considering the smallest cycles that run a certain distance on it, the other part of the cycles yield infinitely many disjoint paths connecting the positive and negative half of the line. □

Proof of Proposition 4.71

1. It is a universal covering digraph, thus it has Property Z by Lemma 4.5.
2. This follows directly from the construction.
3. This again is immediate.
4. The thin ends are the ones of the underlying tree of the universal covering digraph. The thick ends are the pancakes.
5. The quadratic pancake–tree is a Cayley–graph of the group
$$\langle \{a, b, c, d\}, ad^{-1}bc^{-1} = ac^{-1}bd^{-1} = 1\rangle.$$
For the hexagon pancake–tree we have the group
$$\langle \{a, b, c\}, ac^{-1}ba^{-1}cb^{-1} = ab^{-1}ca^{-1}bc^{-1} = 1\}\rangle.$$
Easier than checking that these groups indeed induce the pancake–tree, one could alternatively use the closed path property from 1.34. □

5.4. Statement–proofs

Due to the large amount of statements in Sections 4.1 and 4.2 we just include some of their proofs and only references for the others.

Proof of Proposition 4.1

1. Any automorphism of D that takes an edge e into $\mathcal{A}(e)$ must stabilize $\mathcal{A}(e)$ setwise. Since D is 1–arc transitive its automorphism group induces a $G \subset \mathrm{Aut}(\mathcal{A}(e))$ which acts 1–arc transitive on $\mathcal{A}(e)$.

2. If D contains a loop then because of 1–arc transitivity every edge has to be a loop. Since D it connected it has then only one vertex and thus its reachability relation is universal. Thus we assume that D has no loops. Assume that $\Delta(D)$ is not bipartite. Then it contains a 2–arc (e_1, e_2). Consider the vertex v in the middle of this 2–arc. All its in–edges $e_{1,i}$ must be in $\mathcal{A}(e_1)$ and all its out–edges $e_{2,i}$ in $\mathcal{A}(e_2)$. But $\mathcal{A}(e_1) = \mathcal{A}(e_2)$. Because of 1–arc transitivity there are automorphisms $e_1 \mapsto e_2$, $e_1 \mapsto e_{2,i}$, $e_2 \mapsto e_1$ and $e_2 \mapsto e_{1,i}$ with which we can extend the associated digraph beyond the in- and out–neighbours of v. Inductively the reachability relation is universal since D is connected.

A similar proof can be found in [1] Proposition 1.1 on page 379. □

Proof of Lemma 4.2
A proof can be found in [1] Lemma 2.8 on page 382. □

Proof of Lemma 4.3
A proof can be found in [1] Corollary 2.9 on page 282. □

Proof of Lemma 4.4
A proof can be found in [1] proof of Theorem 3.6 on pages 386 and 387. □

Proof of Lemma 4.5
A proof can be found in [1] Lemma 3.2 (a) on page 385. □

Proof of Lemma 4.6
Suppose $\phi : D \to Z$ is a homomorphism. It takes some $e_D \mapsto e_Z$. Every alternating walk must be entirely mapped to a single edge. If D has universal reachability relation then every edge in D must map to e_Z. Thus ϕ cannot be epimorphic.
A different proof can be found in [1] Lemma 3.2 (b) on page 385. □

Proof of Lemma 4.7
A proof can be found in [1] Proposition 3.10 on page 388. □

Proof of Lemma 4.8
Let $(e, f) = (x, y, z)$ be an alternating walk. The orbit Hz must be contained in $N^+(y)\backslash\{x\}$ (respectively $N^-(y)\backslash\{x\}$) and hence it has less than d elements. Thus by Theorem 1.7 $[H : \mathrm{Stab}_H(x)] < d$. Thus we also have $\left[H|_\Omega : \mathrm{Stab}_{H|_\Omega(x)}\right] < d$. From our prerequisite we therefore have $H|_\Omega = \mathrm{Stab}_{H|_\Omega(x)}$. But that means that $\mathrm{Stab}_H(x)$ fixes no element of Ω because we assumed that H does so. Inductively we obtain that for all $x \in \mathcal{A}(e)$ the stabilizer $\mathrm{Stab}_H(x)$ fixes no element of Ω.
Finally we indirectly assume that there exist an alternating walk starting with

e and terminating with an $x \in \Omega$. We get that $H|_\Omega = \text{Stab}_{H|_\Omega(x)}$ fixes x contradicting the assumption that the elements of Ω are moved.
A similar proof with different notation can be found in [3] Proposition 3.2 on page 25. □

Proof of Lemma 4.9
A proof can be found in [4] Lemma 1 on page 148. □

Proof of Lemma 4.10
A proof can be found in [4] Lemmas 3 and 4 on pages 149 and 150. □

Proof of Lemma 4.11
A proof can be found in [3] Proposition 2.4. □

Proof of Lemma 4.12
A proof can be found in [3] Lemma 2.5. □

Proof of Lemma 4.13
Without loss of generality we assume that $d^+(D) > d^-(D)$. Consider a vertex $x \in V(D)$. There are $d^+(D)\left|x^{\Rightarrow k}\right|$ edges with initial vertex in $x^{\Rightarrow k}$ and $d^-(D)\left|x^{\Rightarrow k+1}\right|$ edges with terminal vertex in $x^{\Rightarrow k+1}$. Obviously we have

$$d^-(D)\left|x^{\Rightarrow k+1}\right| \geq d^+(D)\left|x^{\Rightarrow k}\right|$$

and thus

$$\left|x^{\Rightarrow k+1}\right| \geq \left|x^{\Rightarrow k}\right| \frac{d^+(D)}{d^-(D)}.$$

Inductively we get

$$\left|x^{\Rightarrow k+1}\right| \geq \left|x^{\Rightarrow 1}\right| \left(\frac{d^+(D)}{d^-(D)}\right)^k = d^+(D)\left(\frac{d^+(D)}{d^-(D)}\right)^k.$$

Thus the out–spread is greater or equal $\frac{d^+(D)}{d^-(D)} > 1$. A similar proof can be found in [4] Lemma 6 on pages 155 and 156. □

Proof of Lemma 4.14
A proof can be found in [2] Lemma 2.4 on page 1534. □

Proof of Theorem 4.15

1. This follows from the construction.

2. This will follow by 3.

3. The idea of the proof is to simply construct the covering projection inductively. Starting at the initial edge of the s–arc one uses the isomorphisms between the $\mathcal{A}(e)$s to built the covering projection along the target s–arc and finally on the whole digraph. Then one checks that the result indeed is a covering projection.

For details see [1] Theorems 2.2 and 2.3 on pages 380 and 381. □

Proof of Theorem 4.16
The idea of the proof is again to inductively define the required covering projection from the given ones and checking that the result is indeed a covering projection. The construction runs along the underlying tree of $DL(\Delta)$ using Lemma 4.3 in every step on the next reached $\mathcal{A}(e)$. For details see [1] Theorem 2.6 on pages 382 and 383. □

Proof of Theorem 4.17
A proof can be found in [1] Theorem 3.6 on pages 386 to 388. □

Proof of Theorem 4.18
A proof can be found in [4] Theorem 3 on pages 155 and 156. □

Proof of Theorem 4.19
A proof can be found in [11]. □

Proof of Theorem 4.20
A proof can be found in [1] Corollary 4.9 on page 393. □

Proof of Proposition 4.21
This is immediate from the definition of the tensor product. □

Proof of Proposition 4.24

1. The ordered field digraph has universal reachability relation.
2. The Evans–graph has universal reachability relation and finite out–spread.
3. The DeVos–Mohar–Šámal–digraph has universal reachability relation and finite spread. □

Proof of Theorem 4.25

1. This is by Theorem 4.19 and Lemma 4.6
2. This is by Theorem 4.26.

3. This is evident. □

Proof of Theorem 4.26
Since D is 2–arc transitive the stabilizer of an edge $e = (x, y)$ acts transitively on the out–neighbours of y. Since $d^+(D) = |N^+(y)|$ is a prime, there is a subgroup $A \subseteq \mathrm{Stab}_{\mathrm{Aut}(D)}(e)$ such that $A|_{N^+(D)} \cong \mathbb{Z}_p$. Thus we can apply Lemma 4.8 with $\Omega = N^+(y)$ and the reachability relation cannot be universal.
A similar proof can be found in [3] Theorem 3.3 on page 25. □

Proof of Theorem 4.27
Since K has degree greater or equal d, there is a subgroup $\bar{K} < \mathrm{Stab}_{\mathrm{Aut}(D)}(e)$ with degree greater or equal d. We can apply Lemma 4.8 with $\Omega = N^+(y)$ and $H = \bar{K}$. A similar proof can be found in [3] Theorem 3.4 on pages 25 and 26. □

Proof of Theorem 4.28
The DMS$(n + 1, m + 1)$ is such a graph. A proof can be found in [5]. □

Proof of Theorem 4.29
A proof can be found in [4] Theorem 1 on page 151. □

Proof of Corollary 4.30

1. A proof can be found in [4] Lemma 5 pages 153 and 154.
2. A proof can be found in [4] in the remark after Lemma 5 on page 154. □

Proof of Theorem 4.31
The alternating–cycle digraph is such a graph. See Theorem 4.26 and Lemma 4.4 □

Proof of Theorem 4.32
A proof can be found in [4] Theorem 2 on page 154. □

Proof of Theorem 4.33
A proof can be found in [3] Theorem 2.6 on page 24. □

Proof of Theorem 4.34
A proof can be found in [3] Proposition 2.7 on page 24. □

Proof of Theorem 4.35
The pancake trees are such graphs. □

Proof of Theorem 4.36
A proof can be found in [2] Proposition 2.2 on page 1534. □

Proof of Theorem 4.37
A proof can be found in [2] Theorem 3.1 on page 1536. □

Proof of Theorem 4.38
The digraph D is 1–arc transitive. We will prove by induction that it is $(k+1)$–arc transitive and thus highly arc transitive. Therefore we consider a k–arc $(x_0, \dots, x_k)$. Because again D is 1–arc transitive the stabilizer $\mathrm{Stab}_{\mathrm{Aut}(D)}(x_k)$ acts transitively on $N^+(x_k)$. Since $|N^+(x_k)| = p$ is a prime there is a cyclic subgroup $\langle g_0 \rangle \subseteq \mathrm{Stab}_{\mathrm{Aut}(D)}(x_k)$ that stabilizes $N^+(x_k)$ setwise and has restriction $\langle g \rangle|_{N^+(x_k)} \cong \mathbb{Z}_p$. Also the in–neighbours $N^-(x_k)$ are stabilized setwise by $\langle g \rangle$. Since $|N^-(x_k)| < p$ (by assumption) the restriction $\langle g \rangle|_{N^-(x_k)}$ has a degree m_0 that does not contain p as prime factor. Thus $\langle g^{m_0} \rangle$ stabilizes $N^-(x_k)$ pointwise and acts transitively on $N^+(x_k)$.
Now we do the same with the in–neighbours of x_{k-1} and inductively get a group $\langle g^{m_0 \dots m_k} \rangle \leq \mathrm{Aut}(D)$ that stabilizes $(x_0, \dots, x_k)$ and acts transitively on $N^+(x_k)$. Thus D is $(k+1)$–arc transitive.
A similar proof can be found in [2] Proposition 3.2 on pages 1536 and 1537. □

Proof of Theorem 4.40
The graph is constructed in [10]. □

Proof of Theorem 4.41
A proof can be found in [13] Theorem 5.1 on page 10. □

Proof of Theorem 4.42
A proof can be found in [13] Lemma 7.1 on page 14. □

Proof of Theorem 4.43
A proof can be found in [12] Theorem 6.1 on pages 817 and 818. □

Proof of Theorem 4.44
Proofs can be found in [12] Sections 2, 3 and 4. □

Proof of Theorem 4.45
A proof can be found in [3] Theorem 4.5 on page 27. □

Proof of Theorem 4.46
A proof can be found in [3] Theorem 4.7 on page 28. □

5.5. Other proofs

Proof of Lemma 1.18
Every vertex is a 0–walk to itself, thus the relation is reflexive. It is symmetric, because walks do not care about the orientation of edges and transitive because we can concatenate walks. Moreover the components are well–defined because no edges can leave them, since otherwise there would be more vertices in the component. □

Proof of Lemma 1.25
The identity is the neutral element. The reverse maps are the inverse elements. Finally

$$g \circ (h \circ k)(x) = g(h(k(x))) = (g \circ h) \circ k(x)$$

□

Proof of Theorem 1.7
Consider the map $o : g\mathrm{Stab}_G(x) \to gx$. This is onto because $G \to Gx : g \mapsto gx$ is onto. It is one to one because if $gx = hx$ then $x = h^{-1}(gx) = (h^{-1}g)x$, thus $h^{-1}g \in \mathrm{Stab}_G(x)$ and thus $g\mathrm{Stab}_G(x) = h\mathrm{Stab}_G(x)$. □

Proof of Proposition 1.34
That a Cayley–graph has the claimed two properties is evident. If on the other hand a digraph has these properties we can define a group that has it as Cayley–graph. We can present this group with all the colours as generators and all closed paths as relations. If this presentation defines a group, it must have the original graph as Cayley–graph (by construction). The relations cannot contradict each other because we got them from the digraph which would be impossible if they would contradict. □

Proof of Theorem 1.38
A proof can be found in [14]. □

Proof of Lemma 3.5
That follows directly from the fact, that D is 1–arc transitive. Under an automorphism every edge e must take its $\mathcal{A}(e)$ with it (else would contradict it to be isomorphic). □

Proof of Theorem 3.17
The result can be derived from [16] Theorem 1.1. Alternatively a reference can be found in [2] Theorem 2.3 on page 1534. □

A. Sources

This appendix is for the reader who is interested in how to create such nice pictures. They were drawn using the TikZ package

```
\usepackage{tikz}
\usetikzlibrary{arrows,decorations.pathmorphing,
    backgrounds,positioning ,fit ,calc ,through,shapes}
```

and C++. Most of the pictures probably could have been created using the LaTeX and PGF inherent programming functions, but it turned out that it is way easier to produce simple TikZ code using C++. The first figures drawn were Figures 1, 2 and 3 which where just coded in TikZ. This was boring and relatively time consuming. Thus the author decided to use C++ to get on with the pictures faster.

In the following the used code is described. In case that the reader likes to use the code the author provides electronic copies (so do not start typing it).

The TikZ code for the some more pictures which were not drawn using C++ but using only TikZ, PGF and LaTeX is of course also available electronically.

A.1. Adjacency matrices

The chronologically first figure from Section 2 was Figure 20. The C++ code drawing it was actually not intended to be a graphic application but to visualize some tensor–products for the authors better understanding (that was motivated by [1] and resulted in Remark 2.45 (3)).

The code is not more than a very basic matrix class `Matrix` that beside the necessary members consists mainly of a drawing method `void Matrix::tikz` and a Kronecker product `friend Matrix& operator *`.

The parts of Figure 20 were both created by the function `void TensorLKZwei()`.

Note the parameters of `void Matrix::tikz` after the output file name are just alignments. They specify the shape of the output matrix, the size of the gaps between the vertices, where additional gaps between groups of vertices shall be placed and if there should be dots that indicates that the pattern extends. To understand the gaps just look at the parameters in `matrix.h` in line 121 and 122 and keep in mind that the standard integer division in C++ truncates rather than rounds.

The other pictures created with `void Matrix::tikz` and there codes are:

- Figure 6 by `void LK(char[],int,int)`
- the left part of Figure 12 by `void BigRegDreiDreiZwei()`
- Figure 4 by `void ZLine()`

A.2. Trees

Obviously its not satisfying to draw trees as matrices. Thus the vertices of the trees were placed recursively. The function `void tree` draws the root in the center and calls the recursive function `void treenode` with coordinates arranged at the roots of unity around as `void treenode` itself does after drawing a vertex and an edge. `void tree` takes also a parameter that specifies an initial angle, so one can turn the tree as desired. The pictures in Figure 5 left, 30 and 12 right were created with these functions. In the latter one the colored edges were later adjusted by hand. For the picture in Figure 5 right a slightly different root function `void edgetree` that calls one of the subsequent `void treenode`s with a different scaling such that the resulting tree is centered on an edge.

A.3. Line digraphs

The Figures 13, 14 and 11 show a line–graphs of a tree and a subgraph of the same line–graph. Since TikZ offers the feature to define a vertex at an edge it was easy to modify the functions `void tree` and `void treenode` in a way to put an additional vertex in the center of every edge. Thanks to the recursive structure it was also easy to put a given graph on the edge vertices surrounding every vertex. Thus we cheaply get a linegraph of a tree if we choose this given graph (here defined by a `Matrix`) complete bipartite (or easier complete since the algorithm ignores the additional edges) or any desired universal covering digraph by choosing the desired Δ.
The adapted functions are `void DL` and `void DLnode`.

A.4. $K_{n,n}$–tubes

The Figures 8, 9 and 10 required some more layouting. Thus the class `CGraph` was invented. It provides a list of vertices and a list of edges. The vertices can be arranged in $\mathbb{R}^3$, this is simulated by a vector that gives the projection of the direction z into the x–y–plain. The style and colour can be assigned to every single edge as string. That is not very elegant programming but it does the job. For drawing the figures mentioned above the function `void KNNCon` was used.

A.5. Sequences digraphs

The sequences are generated and checked by some non–member functions located at the end of the file `graph.h`. The bipartite Δ that should be respected at the middle of the sequence is just assumed to be a $K_{n,m}$. The rest is done by the methods `void CGraph::sequences(int n,int m,int depth)` and `void CGraph::edges_simpleseq()`.

A.6. DMS and HH

These graphs use the edge set of an underlying tree as vertex set, like the line-graphs. But unlike them the edge set is not easily drawn with the same recursive function that draws the vertices. Thus the functions for generating trees were adapted as members of `Class CGraph` to run the algorithms as members `void CGraph::DMS` and `void CGraph::HH`.

A.7. Alternating–cycle digraph

Since the AC has a tree as substructure it was canonically drawn with a recursive approach by the member `void CGraph::AC`.

A.8. Functions and Main

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "graph.h"
#include "matrix.h"


void TensorLKZwei();
void ZLine();
void LK(char Datei[],int n,int l);
void tree(char Datei[],int inval,int outval,int depth,
    float shrinkexp=1,float shrinklin=0,float scale = 1.,
    float startarc = 0.);
void edgetree(char Datei[],int inval,int outval,int depth,
    float shrinkexp=1,float shrinklin=0,float scale = 1.,
    float startarc = 0.);
void treenode(int inval,int outval,int depth,bool edge,
    float x,float y,float prex,float prey,char prename[],
    int name_i,float shrinkexp,float shrinklin,float koeff,
    FILE *fp);
void BipRegDreiDreiZwei();
void DL(char Datei[],char Befehl[],Matrix *pDelta,int
    inval,int outval,int depth,float shrinkexp=1,float
    shrinklin=0,float scale = 1.,float startarc = 0.);
```

```
void DLnode(Matrix *pDelta,int inval,int outval,int depth,
    bool edge,float x,float y,float prex,float prey,char
    prename[],int name_i,float shrinkexp,float shrinklin,
    float koeff,FILE *fp);
void KDreiDreiCon();
void KNNCon(int n,int length,double arc,char Datei[],char
    Befehl[]);
void TensorSeq1_2();
void TensorSeq(int n,int m,int depth);
void HH(char Datei[],char Befehl[],int inval1,int inval2,
    int outval1,int outval2,int depth,int part=1,double
    shrinkexp=1.,double shrinklin=0.,double startarc=0.,
    double scale=1.,bool bend=true,bool color=true);


int main ( void )
{
    TensorLKZwei();
    ZLine();
    LK("LKVier.tex",4,6);
    tree("BaumEinsZweiVier.tex",1,2,4,(float)sqrt(3.)
        ,-0.2,0.3);
    tree("BaumZweiZweiVier.tex",2,2,4,2,-0.2,0.3);
    tree("BaumZweiDreiVier.tex",2,3,4,2.75,0.,0.08);
    edgetree("EBaumEinsZweiVier.tex",1,2,4,(float)sqrt(3.)
        ,-0.2,0.3);
    edgetree("EBaumZweiDreiDrei.tex",2,3,3,2.5,0.,0.1,3*
        M_PI/2);
    BipRegDreiDreiZwei();
    tree("BaumDreiDreiDrei.tex",3,3,3,3,0,0.1,2*M_PI/3);

    Matrix A(6,6);
    A.set(false);
    A(0,3)=A(0,4)=A(1,4)=A(1,5)=A(2,5)=A(2,3)=true;
    DL("DL.tex","DL",&A,3,3,3,3,0,0.2,2*M_PI/3);

    A.set(true);
    DL("LineBaumDreiDreiDrei.txt","LineBaumDreiDreiDrei",&A
        ,3,3,3,3,0,0.15,2*M_PI/3);

    Matrix B(4,4);
    B.set(true);
```

```
    DL("LineBaum.tex","LineBaum",&A,2,2,3,3,0,0.15,3*M_PI/4);

    KDreiDreiCon();
    KNNCon(2,9,5*M_PI/16,"KZweiZweiCon.txt","KZweiZweiCon");
    KNNCon(3,7,5*M_PI/16,"KDreiDreiCon.txt","KDreiDreiCon");
    KNNCon(4,3,5*M_PI/16,"KVierVierCon.txt","KVierVierCon");

    TensorSeq1_2();
    TensorSeq(2,3,2);
    TensorSeq(2,2,3);

    HH("HHDreiDreiVier.tex","HHDreiDreiVier",1,1,2,2,4,1,sqrt(2.),0.,0.,0.4,true,true);
    HH("HHDreiDreiFuenf.tex","HHDreiDreiFuenf",1,1,2,2,5,2,sqrt(2.),0.,0.,0.2,true,true);
    HH("HHVierDreiVier.tex","HHVierDreiVier",2,1,2,2,4,1,sqrt(3.),0.,0.,0.4,true,true);
    HH("HHVierDreiFuenf.tex","HHVierDreiFuenf",2,1,2,2,5,2,sqrt(3.),0.,0.,0.2,true,true);
    HH("HHDreiVierVier.tex","HHDreiVierVier",1,2,2,2,4,1,sqrt(3.),0.,0.,0.4,true,true);
    HH("HHDreiVierFuenf.tex","HHDreiVierFuenf",1,2,2,2,5,2,sqrt(3.),0.,0.,0.2,true,true);

    CGraph G;
    G.biedgetree(1,1,2,2,4,sqrt(2.),0.,0.,false);
    G.color_edge("black!30");
    G.DMS(1);
    G.style_edge("very thick, bend left=30",1);
    G.paint("DMSDreiDreiVier.tex","DMSDreiDreiVier",0.6);

    CGraph D;
    D.AC(5);
    D.paint("ACFuenf.tex","ACFuenf",3.5);

    getch();

    return 0;
}
```

```
void HH(char Datei[],char Befehl[],int inval1,int inval2,
    int outval1,int outval2,int depth,int part,double
    shrinkexp,double shrinklin,double startarc,double scale
    ,bool bend,bool color)
{
    CGraph G;
    G.bitree(inval1,inval2,outval1,outval2,depth,shrinkexp,
        shrinklin,startarc,false);
    G.color_edge("black!30");
    G.HH(part);
    if(color)
        G.edge_color_delta_part(1);
    if(bend)
        G.style_edge("very thick, bend left=30",1);
    else
        G.style_edge("very thick");
    G.paint(Datei,Befehl,scale);
}

void TensorSeq(int n,int m,int depth)
{
    double arc=M_PI/5;
    CGraph G;
    CGraph Z;
    CGraph T;
    G.setzvek(cos(arc),sin(arc));
    Z.setzvek(cos(arc),sin(arc));
    T.setzvek(cos(arc),sin(arc));
    char vertname[4];
    vertname[1]=vertname[3]='\0';

    int i;
    for(i=0;i<4;i++)
    {
        vertname[0]='0'+i;
        Z.vertex(0,3*i,0,vertname,true);
    }
    for(i=0;i<3;i++)
        Z.edge(i,i+1);

    G.sequences(n,m,depth);

```

```
    G.rotxy(M_PI/2);
    G.rotyz(M_PI/2);

    G.paint("SeqNM.tex","SeqNM",0.5);

    T.tensor(&Z,&G);
    T.edge_color_delta();
    T.style_edge("very thick");
    T.paint("TensorZSeqNM.tex","TensorZSeqNM",0.5);

    G.strech(3,1,1);
    G.setzvek(0,1);
    G.paint("SeqNMnames.tex","SeqNMnames",0.5,true);
}

void TensorSeq1_2()
{
    double arc=M_PI/5;
    CGraph G;
    CGraph Z;
    CGraph T;
    G.setzvek(cos(arc),sin(arc));
    Z.setzvek(cos(arc),sin(arc));
    T.setzvek(cos(arc),sin(arc));
    char vertname[4];
    vertname[1]=vertname[3]='\0';

    int i;
    for(i=0;i<4;i++)
    {
        vertname[0]='0'+i;
        Z.vertex(0,3*i,0,vertname,true);
    }
    for(i=0;i<3;i++)
        Z.edge(i,i+1);

    G.vertex(0,0,0,"00000",true);
    G.vertex(2,0,0,"10000",true);
    G.vertex(4,0,1,"11000",true);
    G.vertex(4,0,-1,"01000",true);
    G.vertex(6,0,1.5,"11100",true);
    G.vertex(6,0,0.5,"01100",true);
    G.vertex(6,0,-0.5,"10100",true);
```

```
    G.vertex(6,0,-1.5,"00100",true);
    G.vertex(8,0,1.75,"11110",true);
    G.vertex(8,0,1.25,"01110",true);
    G.vertex(8,0,0.75,"10110",true);
    G.vertex(8,0,0.25,"00110",true);
    G.vertex(8,0,-0.25,"11010",true);
    G.vertex(8,0,-0.75,"01010",true);
    G.vertex(8,0,-1.25,"10010",true);
    G.vertex(8,0,-1.75,"00010",true);

    G.edges_simpleseq();
    G.paint("SeqEinsZwei.tex","SeqEinsZwei",1,true);

    T.tensor(&Z,&G);
    T.paint("TensorZSeqEinsZwei.tex","TensorZSeqEinsZwei");
}

void KNNCon(int n,int length,double arc,char Datei[],char
    Befehl[])
{
    CGraph G;
    int i,j,k,l;
    G.setzvek(cos(arc),sin(arc));

    for(i=0;i<length+2;i++)
        for(j=0;j<n;j++)
            for(k=0;k<n;k++)
                G.vertex((cos(arc)*(n-1)+1)*i,j,k,"",i&&i<
                    length+1);
    for(i=1;i<length;i++)
        for(j=0;j<n;j++)
            for(k=0;k<n;k++)
                for(l=0;l<n;l++)
                    if(i%2)
                        G.edge(n*n*i+n*j+k,n*n*(i+1)+n*j+l,"",j
                            ==n-1?"blue":"blue!30",j==n-1?"very␣
                            thick":"thin");
                    else
                        G.edge(n*n*i+n*j+k,n*n*(i+1)+n*l+k,"",!k
                            ?"red":"red!30",!k?"very␣thick":"thin
                            ");
    for(i=0;i<n*n;i++)
    {
```

```
        G.edge(i,i+n*n,"","black","dotted");
        G.edge(n*n*length+i,n*n*length+i+n*n,"","black","dotted");
    }
    G.paint(Datei,Befehl,0.6);
}

void KDreiDreiCon()
{
    CGraph G;
    int i,j,k,l;
    G.setzvek(cos(5*M_PI/16),sin(5*M_PI/16));

    for(i=0;i<9;i++)
        for(j=0;j<3;j++)
            for(k=0;k<3;k++)
                G.vertex(2.3*i,j,k,"",i&&i<8);
    for(i=1;i<7;i++)
        for(j=0;j<3;j++)
            for(k=0;k<3;k++)
                for(l=0;l<3;l++)
                    if(i%2)
                        G.edge(9*i+3*j+k,9*(i+1)+3*j+l,"",j==2?"blue":"blue!30",j==2?"very thick":"thin");
                    else
                        G.edge(9*i+3*j+k,9*(i+1)+3*l+k,"",!k?"red":"red!30",!k?"very thick":"thin");
    for(i=0;i<9;i++)
    {
        G.edge(i,i+9,"","black","dotted");
        G.edge(63+i,63+i+9,"","black","dotted");
    }
    G.paint("KDreiDreiCon.txt","KDreiDreiCon",0.6);
}

void DL(char Datei[],char Befehl[],Matrix *pDelta,int inval,int outval,int depth,float shrinkexp,float shrinklin,float scale,float startarc)
{
    int i,j;
    int k=inval+outval;
```

```
    float x=0.,y=0.;
    float koeff=1.;
    char inedgename[4]="e__";
    char outedgename[4]="e__";

    for(i=0;i<depth;i++)
    {
        koeff+=shrinklin;
        koeff*=shrinkexp;
    }

    FILE *fp;
    fp=fopen(Datei,"w");

    fprintf(fp,"\\def\\%s{\n",Befehl);
    fprintf(fp,"\\begin{tikzpicture}\n");
    fprintf(fp,"[scale=%0.2f,\n",scale);
    fprintf(fp,"inner␣sep=1.5,\n");
    fprintf(fp,"vertex/.style={circle,draw=black!50,fill=black!20,␣very␣thick},\n");
    fprintf(fp,"edgevertex/.style={circle,draw=blue!80,fill=blue!40,␣very␣thick},\n");
    fprintf(fp,"post/.style={->,␣>=stealth'},\n");
    fprintf(fp,"pre/.style={<-,␣>=stealth'}]\n");
    fprintf(fp,"\\node␣[vertex]␣(a)␣at␣(%f,%f)␣{};\n",x,y);

    for(i=0;i<k;i++)
        DLnode(pDelta,inval,outval,depth-1,i<inval,
        koeff*cos(2*i*M_PI/k+startarc),koeff*sin(2*i*M_PI/k+startarc),
            x,y,"a",i,shrinkexp,shrinklin,koeff/shrinkexp-shrinklin,fp);

    for(i=0;i<inval;i++)
    {
        inedgename[2]='0'+i;
        for(j=inval;j<k;j++)
            if((*pDelta)(i,j))
            {
                outedgename[2]='0'+j;
                fprintf(fp,"\\draw␣[post,blue,very␣thick]␣(%s)␣to␣[bend␣left=30]␣(%s);\n",inedgename,outedgename);
```

```
            }
    }

    fprintf(fp,"\\end{tikzpicture}\n}\n");

    fclose(fp);
}

void DLnode(Matrix *pDelta,int inval,int outval,int depth,
    bool edge,float x,float y,float prex,float prey,char
    prename[],int name_i,float shrinkexp,float shrinklin,
    float koeff,FILE *fp)
{
    int i,n,j;
    for(n=0;prename[n]!='\0';n++);
    char *newname=new char[n+3];
    char *inedgename=new char[n+5];
    char *outedgename=new char[n+5];
    char *edgename=new char[n+3];
    for(i=0;i<n;i++)
        edgename[i]=outedgename[i]=inedgename[i]=newname[i]=prename[i];
    inedgename[n]=outedgename[n]=newname[n]=edgename[n]='_';
    inedgename[n+1]=outedgename[n+1]=newname[n+1]=edgename[n+1]='0'+name_i;
    inedgename[n+2]=outedgename[n+2]='_';
    newname[n+2]=edgename[n+2]='\0';
    inedgename[n+4]=outedgename[n+4]='\0';
    inedgename[0]=outedgename[0]=edgename[0]='e';

    if(depth==0)
    {
        fprintf(fp,"\\node (%s) at (%f,%f) {}\n",newname,x,y);
        fprintf(fp,"   edge [dotted] node (%s) {} (%s);\n",
            edgename,prename);
    }
    else
    {
        fprintf(fp,"\\node [vertex] (%s) at (%f,%f) {}\n",
            newname,x,y);
```

```
        fprintf(fp,"   edge [%s,black!30] node (%s) [edgevertex] {} (%s);\n",edge?"post":"pre",edgename,prename);
    }

    if(depth<=0)
        return;

    int k=inval+outval;
    float arc=0;
    if(y==prey)
        if(x<prex) arc=0;
        else arc=M_PI;
    else
        if(x==prex)
            if(y<prey) arc=M_PI/2;
            else arc=3*M_PI/2;
        else
        {
            arc=atan((y-prey)/(x-prex));
            if(x>prex)
                arc+=M_PI;
        }

    for(i=1;i<k;i++)
        DLnode(pDelta,inval,outval,depth-1,i<inval+edge?1:0,
            x+koeff*cos(2*i*M_PI/k+arc),y+koeff*sin(2*i*M_PI/k+arc),
            x,y,newname,i,shrinkexp,shrinklin,koeff/shrinkexp-shrinklin,fp);

    if(edge)
    {
        for(i=0;i<inval;i++)
        {
            inedgename[n+3]='0'+i+1;
            for(j=inval+1;j<k+1;j++)
                if((*pDelta)(i,j-1))
                {
                    outedgename[n+3]='0'+j;
                    if(depth==1)
                        fprintf(fp,"\\draw [blue,thin] (%s) to [bend left=30] (%s);\n",inedgename,j==
```

```
                        k?edgename:outedgename);
                else
                    fprintf(fp,"\\draw [post,blue,very thick] (%s) to [bend left=30] (%s);\n",inedgename,j==k?edgename:outedgename);
            }
        }
    }
    else
    {
        for(i=0;i<inval;i++)
        {
            inedgename[n+3]='0'+i;
            for(j=inval;j<k;j++)
                if((*pDelta)(i,j))
                {
                    outedgename[n+3]='0'+j;
                    if(depth==1)
                        fprintf(fp,"\\draw [blue,thin] (%s) to [bend left=30] (%s);\n",i?inedgename:edgename,outedgename);
                    else
                        fprintf(fp,"\\draw [post,blue,very thick] (%s) to [bend left=30] (%s);\n",i?inedgename:edgename,outedgename);
                }
        }
    }

}

void tree(char Datei[],int inval,int outval,int depth,float shrinkexp,float shrinklin,float scale,float startarc)
{
    int i;
    int k=inval+outval;
    float x=0.,y=0.;
    float koeff=1.;

    for(i=0;i<depth;i++)
    {
```

```
        koeff+=shrinklin;
        koeff*=shrinkexp;
    }

    FILE *fp;
    fp=fopen(Datei,"w");

    fprintf(fp,"\\begin{tikzpicture}\n");
    fprintf(fp,"[scale=%0.2f,\n",scale);
    fprintf(fp,"inner sep=1.5,\n");
    fprintf(fp,"vertex/.style={circle,draw=black!70,fill=black!40,very thick},\n");
    fprintf(fp,"post/.style={->, >=stealth',very thick},\n");
    fprintf(fp,"pre/.style={<-, >=stealth',very thick}]\n");

    fprintf(fp,"\\node [vertex] (a) at (%f,%f) {};\n",x,y);

    for(i=0;i<k;i++)
        treenode(inval,outval,depth-1,i<inval,
        koeff*cos(2*i*M_PI/k+startarc),koeff*sin(2*i*M_PI/k+startarc),
            x,y,"a",i,shrinkexp,shrinklin,koeff/shrinkexp-shrinklin,fp);

    fprintf(fp,"\\end{tikzpicture}\n");

    fclose(fp);
}

void edgetree(char Datei[],int inval,int outval,int depth,float shrinkexp,float shrinklin,float scale,float startarc)
{
    int i;
    int k=inval+outval;
    float x=0.,y=0.;
    float koeff=1.;

    for(i=0;i<depth;i++)
    {
        koeff+=shrinklin;
```

```
        koeff*=shrinkexp;
    }

    FILE *fp;
    fp=fopen(Datei,"w");

    fprintf(fp,"\\begin{tikzpicture}\n");
    fprintf(fp,"[scale=%0.2f,\n",scale);
    fprintf(fp,"inner sep=1.5,\n");
    fprintf(fp,"vertex/.style={circle,draw=black!70,fill=black!40,very thick},\n");
    fprintf(fp,"post/.style={->, >=stealth',very thick},\n");
    fprintf(fp,"pre/.style={<-, >=stealth',very thick}]\n");

    fprintf(fp,"\\node [vertex] (a) at (%f,%f) {};\n",x,y);

    treenode(inval,outval,depth,true,
            (koeff+shrinklin)*shrinkexp*cos(startarc),(koeff+shrinklin)*shrinkexp*sin(startarc),
            x,y,"a",0,shrinkexp,shrinklin,koeff,fp);
    for(i=1;i<k;i++)
        treenode(inval,outval,depth-1,i<inval,
            koeff*cos(2*i*M_PI/k+startarc),koeff*sin(2*i*M_PI/k+startarc),
            x,y,"a",i,shrinkexp,shrinklin,koeff/shrinkexp-shrinklin,fp);

    fprintf(fp,"\\end{tikzpicture}\n");

    fclose(fp);
}

void treenode(int inval,int outval,int depth,bool edge,float x,float y,float prex,float prey,char prename[],int name_i,float shrinkexp,float shrinklin,float koeff,FILE *fp)
{
    int i,n;
    for(n=0;prename[n]!='\0';n++);
    char *newname=new char[n+3];
    for(i=0;i<n;i++)
```

```
        newname[i]=prename[i];
    newname[n]='_';
    newname[n+1]='0'+name_i;
    newname[n+2]='\0';

    if(depth==0)
    {
        fprintf(fp,"\\node (%s) at (%f,%f) {}\n",newname,x,y);
        fprintf(fp,"   edge [dotted] (%s);\n",prename);
    }
    else
    {
        fprintf(fp,"\\node [vertex] (%s) at (%f,%f) {}\n",newname,x,y);
        fprintf(fp,"   edge [%s] (%s);\n",edge?"post":"pre",prename);
    }

    if(depth<=0)
        return;

    int k=inval+outval;
    float arc=0;
    if(y==prey)
        if(x<prex) arc=0;
        else arc=M_PI;
    else
        if(x==prex)
            if(y<prey) arc=M_PI/2;
            else arc=3*M_PI/2;
        else
        {
            arc=atan((y-prey)/(x-prex));
            if(x>prex)
                arc+=M_PI;
        }

    for(i=1;i<k;i++)
        treenode(inval,outval,depth-1,i<inval+edge?1:0,
            x+koeff*cos(2*i*M_PI/k+arc),y+koeff*sin(2*i*M_PI/k+arc),
```

```
            x,y,newname,i,shrinkexp,shrinklin,koeff/shrinkexp
               -shrinklin,fp);

}

void LK(char Datei[],int n,int l)
{
    Matrix A;
    A.init(n*l,n*l);
    int i,j,k;

    for(i=0;i<l-1;i++)
        for(j=0;j<n;j++)
            for(k=0;k<n;k++)
                A(n*i+j,n*(i+1)+k)=true;

    A.tikz(Datei,l,n,2,0,1,1,0,1,true);
}

void BipRegDreiDreiZwei()
{
    Matrix A(6,6);
    A.set(false);

    A(0,3)=A(0,4)=A(1,4)=A(1,5)=A(2,5)=A(2,3)=true;
    A.tikz("BipRegDreiDreiZwei.txt",2,3);
}

void ZLine()
{
    Matrix A(7,7);
    A.set(false);
    int i;
    for(i=0;i<6;i++)
        A(i,i+1)=true;

    A.out();
    A.tikz("ZLine.txt",7,1);

}

void TensorLKZwei()
{
```

```
    Matrix A(8,8);
    Matrix B(8,8);
    Matrix C;
    int i;

    A.set(false);
    B.set(false);

    for(i=0;i<6;i+=2)
        A(i,i+2)=A(i,i+3)=A(i+1,i+2)=A(i+1,i+3)=B(i,i+2)=B(i,i+3)=B(i+1,i+2)=B(i+1,i+3)=true;
    C=A*B;
    A.tikz("LKZwei.txt",4,2,3);
    C.tikz("TensorLKZwei.txt",8,8,1,1,2,1,1,2);
}
```

A.9. matrix.h

```
#include <stdio.h>
#include <assert.h>

class Matrix {
    friend const Matrix operator *(const Matrix& X,const Matrix& Y);
public:

    Matrix ();
    Matrix(int nR, int nC = 1);
    Matrix(const Matrix& mat);
    ~Matrix();
    Matrix& operator=(const Matrix& mat);
    int nRow(){return nRow_;}
    int nCol(){return nCol_;}
    bool& operator() (int i, int j = 1) const;
    void init(int nR, int nC = 1);
    void set(bool value);
    void out ();
    int tikz (char Datei[],int n,int m,int xkoeff = 1,int xzaehler = 0,int xnenner = 1,int ykoeff = 1,int yzaehler =0,int ynenner = 1,bool pattern=false);

private:
```

```

int nRow_, nCol_;
bool* data_;
};

Matrix::Matrix () {
    nRow_ = 0; nCol_ = 0;
    data_ = NULL;
}

Matrix::Matrix(int nR, int nC) {
    init(nR,nC);
}

Matrix::Matrix(const Matrix& mat) {
    int i,n;
    nRow_=mat.nRow_;
    nCol_=mat.nCol_;
    n=nRow_*nCol_;
    data_=new bool[n];
    for(i=0;i<n;i++)
        data_[i]=mat.data_[i];
}

Matrix::~Matrix() {
    if(data_!=NULL)
        delete [] data_;
}

Matrix& Matrix::operator=(const Matrix& mat) {
    if( this == &mat ) return *this;
    delete [] data_;
    data_=NULL;
    if(mat.data_ == NULL) return *this;
    nCol_=mat.nCol_;
    nRow_=mat.nRow_;
    int n=nCol_*nRow_;
    data_ = new bool [n];
    assert(data_ != NULL);
    for(n--;n>=0;n--)
        data_[n]=mat.data_[n];
    return *this;
}
```

```

bool& Matrix::operator() (int i, int j ) const {
    assert(i >= 0 && i < nRow_);
    assert(j >= 0 && j < nCol_);
    return data_[ nCol_*i + j ];
}

void Matrix::init(int nR,int nC)
{
    assert(nR > 0 && nC > 0);
    nRow_ = nR; nCol_ = nC;
    data_ = new bool [nR*nC];
    assert(data_ != NULL);
    set(false);
}

void Matrix::set(bool value) {
    int i, n = nRow_*nCol_;
    for( i=0; i<n; i++ )
        data_[i] = value;
}

void Matrix::out(){
    int i,j;
    for (i=0;i<nRow_;i++){
        for (j=0;j<nCol_;j++)
            printf("%d ",data_[ nCol_*i + j ]);
        printf("\n");
    }
}

int Matrix::tikz(char Datei[],int n, int m,int xkoeff,int xzaehler,int xnenner,int ykoeff,int yzaehler,int ynenner,bool pattern){
    if(n*m!=nCol_){
        printf("Ungültige Dimension");
        return -1;
    }
    int i,j;
    bool k=true;

    FILE *fp;
    fp=fopen(Datei,"w");
```

```

    fprintf(fp,"\\begin{tikzpicture}\n");
    fprintf(fp,"[scale=0.4,\n");
    fprintf(fp,"auto=left,\n");
    fprintf(fp,"inner␣sep=1.5,\n");
    fprintf(fp,"vertex/.style={circle,fill=blue!20},\n");
    fprintf(fp,"post/.style={->,␣>=stealth'}]\n");

    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
            fprintf(fp,"\\node␣[vertex]␣(a%d)␣at␣(%d,%d)␣{};\n", i*m+j,
                    -xkoeff*(n/2)+xkoeff*i+xzaehler*i/xnenner,
                    -ykoeff*(m/2)+ykoeff*j+yzaehler*j/ynenner);

    if(pattern)
        for(j=0;j<m;j++)
        {
            fprintf(fp,"\\node␣(l%d)␣at␣(%d,%d)␣{}\n", j,
                    -xkoeff*(n/2)-xkoeff,
                    -ykoeff*(m/2)+ykoeff*j+yzaehler*j/ynenner);
            fprintf(fp,"␣␣␣edge␣[dotted]␣(a%d);\n",j);
            fprintf(fp,"\\node␣(r%d)␣at␣(%d,%d)␣{}\n", j,
                    -xkoeff*(n/2)+xkoeff*n+xzaehler*(n-1)/xnenner,
                    -ykoeff*(m/2)+ykoeff*j+yzaehler*j/ynenner);
            fprintf(fp,"␣␣␣edge␣[dotted]␣(a%d);\n",(n-1)*m+j);
        }
    fprintf(fp,"\\foreach␣\\from/\\to␣in␣");

    for(i=0;i<nCol_;i++)
        for(j=0;j<nCol_;j++){
            if((*this)(i,j)){
                if(k){
                    fprintf(fp,"{");
                    k=false;
                } else {
                    fprintf(fp,",");
                }
                fprintf(fp,"a%d/a%d",i,j);
            }
        }
```

```

    fprintf(fp,"}\n");
    fprintf(fp,"\\draw [post] (\\from) -- (\\to);\n");
    fprintf(fp,"\\end{tikzpicture}\n");

    fclose(fp);
    return 0;
}

const Matrix operator *(const Matrix& X,const Matrix& Y){
    Matrix Z;
    Z.init(X.nCol_*Y.nCol_,X.nCol_*Y.nCol_);

    if(X.nCol_!=X.nRow_ || Y.nCol_!=Y.nRow_ )
        return Z;

    int i,j;
    bool test;
    for (i=0;i<Z.nCol_;i++)
        for (j=0;j<Z.nCol_;j++){
            test=(X(i/Y.nCol_,j/Y.nCol_)&&Y(i%Y.nCol_,j%Y.nCol_));
            Z(i,j)=test;
        }
    return Z;
}
```

A.10. graph.h

```
#ifndef _graph_h
#define _graph_h

#define _USE_MATH_DEFINES
#include <math.h>

int doublecomp(const void *a,const void *b){return (int)(bool)(*(double*)a<*(double*)b);}
bool streq(const char a[],const char b[]){for(int i=0;a[i]!='\0';i++)if(a[i]!=b[i])return false;return true;}
bool check_simpleseq(const char a[],const char b[]);
int seqx(int n,int m,int depth,char vertexname[]);
int seqy(int depth,char vertexname[]);
```

```
bool check_seq(const char a[],const char b[],int depth);
int makeseq(int i,int n,int m,int depth,char vertexname[]);

struct SVertex{
    int id;
    char *name;
    double x,y,z;
    bool draw;
    int partition;
    SVertex *l;
    SVertex *r;

    SVertex(){name=NULL;x=y=z=0;draw=true;}
    ~SVertex(){if(name!=NULL) delete name;}
};

struct SEdge{
    int id;
    int preid,postid;
    int label;
    int partition;
    char *name;
    char *color;
    char *style;
    bool directed;

    SEdge *l;
    SEdge *r;
    SEdge(){name=NULL;color=NULL;style=NULL;}
    ~SEdge(){if(name!=NULL) delete name;if(color!=NULL) delete color;if(style!=NULL) delete style;}
};

class CGraph {
public:
    CGraph();
    ~CGraph();
    SVertex* getpVert(int id);
    SEdge* getpEdge(int id);
    void setzvek(double x,double y){z_x=x;z_y=y;}
    void getstraightmean(int id,double &x,double &y,double &z);
```

```
    bool visibleverts(int id);

    int vertex(double x=0.,double y=0.,double z=0.,char
       name[]="",bool draw=true,int part=0);
    int edge(int preid,int postid,char name[]="",char color
       []="",char style[]="",bool dir=true,int label=0,int
       part=0);
    int edge(char prename[],char postname[],char name[]="",
       char color[]="",char style[]="",bool dir=true,int
       label=0,int part=0);

    void edges_simpleseq();
    void sequences(int n,int m,int depth);

    void tree(int inval,int outval,int depth,double
       shrinkexp=1,double shrinklin=0,double startarc = 0.,
       bool dir=true);
    void edgetree(int inval,int outval,int depth,double
       shrinkexp=1,double shrinklin=0,double startarc = 0.,
       bool dir=true);
    void treenode(int inval,int outval,int depth,bool
       edgedir,double x,double y,double prex,double prey,
       char prename[],int name_i,double shrinkexp,double
       shrinklin,double koeff,bool dir=true);

    void bitree(int inval1,int inval2,int outval1,int
       outval2,int depth,double shrinkexp=1,double
       shrinklin=0,double startarc = 0.,bool dir=true);
    void biedgetree(int inval1,int inval2,int outval1,int
       outval2,int depth,double shrinkexp=1,double
       shrinklin=0,double startarc = 0.,bool dir=true);
    void bitreenode(int inval1,int inval2,int outval1,int
       outval2,int part,int depth,bool edgedir,double x,
       double y,double prex,double prey,char prename[],int
       name_i,double shrinkexp,double shrinklin,double
       koeff,bool dir=true);

    void DMS(int part=1);
    void DMSpath(SEdge *pStart,SEdge *pCurrent,int n,bool
       direction);

    void HH(int part=1);
```

```
    void HHpath(SEdge *pStart,SEdge *pCurrent,int n,bool
       direction,int Zn);

    void AC(int n);
    void ACrek(int n,SVertex *pVa,SVertex *pVb,int depth);

    void strech(double x=1.,double y=1.,double z=1.);
    void rotxy(double arc);
    void rotxz(double arc);
    void rotyz(double arc);

    void edge_color_delta();
    void edge_color_delta_rek(SEdge *pE,const char color[],
       int i);

    void edge_color_delta_part(int part);
    void edge_color_delta_part_rek(SEdge *pE,const char
       color[],int i,int part);

    void tensor(CGraph *a,CGraph *b);

    void style_edge(char edgestyle[],int part=-1);
    void color_edge(char edgecolor[],int part=-1);
    void label_edge(int id,int label);

    void paint(char Datei[],char Befehl[],double scale=1.,
       bool name=false);

private:
    SVertex *pVert;
    SVertex *plastVert;
    SEdge *pEdge;
    SEdge *plastEdge;

    double z_x,z_y;
};

CGraph::CGraph()
{
    pVert=NULL;
    plastVert=NULL;
    pEdge=NULL;
    plastEdge=NULL;
```

```
    z_x=0.;
    z_y=0.;
}

CGraph::~CGraph()
{
    SVertex *pv=pVert;
    SVertex *pvnext=pVert;
    SEdge *pe=pEdge;
    SEdge *penext=pEdge;

    while(pv!=NULL)
    {
        pvnext=pv->r;
        delete pv;
        pv=pvnext;
    }
    while(pe!=NULL)
    {
        penext=pe->r;
        delete pe;
        pe=penext;
    }
}

SVertex* CGraph::getpVert(int id)
{
    SVertex *pV=pVert;
    while(pV!=NULL){
        if(pV->id==id)
            return pV;
        pV=pV->r;
    }
}

SEdge* CGraph::getpEdge(int id)
{
    SEdge *pE=pEdge;
    while(pE!=NULL){
        if(pE->id==id)
            return pE;
        pE=pE->r;
    }
```

```
}

int CGraph::vertex(double x,double y,double z,char name[],
    bool draw,int part)
{
    int i;
    for(i=0;name[i]!='\0';i++);

    if(pVert==NULL)
    {
        plastVert=pVert=new SVertex();
        pVert->x=x;
        pVert->y=y;
        pVert->z=z;
        pVert->draw=draw;
        pVert->partition=part;
        pVert->name=NULL;

        pVert->name=new char[i+1];
        for(;i>=0;i--) pVert->name[i]=name[i];

        pVert->l=pVert->r=NULL;
        pVert->id=0;
    }
    else
    {
        plastVert->r=new SVertex();
        plastVert->r->x=x;
        plastVert->r->y=y;
        plastVert->r->z=z;
        plastVert->r->draw=draw;
        plastVert->r->partition=part;
        plastVert->r->name=NULL;

        plastVert->r->name=new char[i+1];
        for(;i>=0;i--) plastVert->r->name[i]=name[i];

        plastVert->r->l=plastVert;
        plastVert->r->r=NULL;
        plastVert->r->id=plastVert->id+1;
        plastVert=plastVert->r;
    }
    return plastVert->id;
```

```
}

void CGraph::edges_simpleseq()
{
    SVertex *pInVert;
    SVertex *pOutVert;

    pInVert=pVert;
    while(pInVert!=NULL){
        pOutVert=pVert;
        while(pOutVert!=NULL){
            if(check_simpleseq(pInVert->name,pOutVert->name))
                edge(pInVert->id,pOutVert->id);
            pOutVert=pOutVert->r;
        }
        pInVert=pInVert->r;
    }
}

int CGraph::edge(int preid,int postid,char name[],char
    color[],char style[],bool dir,int label,int part)
{
    int i,j,k;
    for (i=0;name[i]!='\0';i++);
    for (j=0;color[j]!='\0';j++);
    for (k=0;style[k]!='\0';k++);

    if(pEdge==NULL)
    {
        plastEdge=pEdge=new SEdge();

        pEdge->name=new char[i+1];
        for (;i>=0;i--) pEdge->name[i]=name[i];

        pEdge->color=new char[j+1];
        for (;j>=0;j--) pEdge->color[j]=color[j];

        pEdge->style=new char[k+1];
        for (;k>=0;k--) pEdge->style[k]=style[k];

        pEdge->directed=dir;
        pEdge->label=label;
        pEdge->partition=part;
```

```

        pEdge->preid=preid;
        pEdge->postid=postid;
        pEdge->l=pEdge->r=NULL;
        pEdge->id=0;
    }
    else
    {
        plastEdge->r=new SEdge();

        plastEdge->r->name=new char[i+1];
        for(;i>=0;i--) plastEdge->r->name[i]=name[i];

        plastEdge->r->color=new char[j+1];
        for(;j>=0;j--) plastEdge->r->color[j]=color[j];

        plastEdge->r->style=new char[k+1];
        for(;k>=0;k--) plastEdge->r->style[k]=style[k];

        plastEdge->r->directed=dir;
        plastEdge->r->label=label;
        plastEdge->r->partition=part;

        plastEdge->r->preid=preid;
        plastEdge->r->postid=postid;
        plastEdge->r->l=plastEdge;
        plastEdge->r->r=NULL;
        plastEdge->r->id=plastEdge->id+1;
        plastEdge=plastEdge->r;
    }
    return plastEdge->id;
}

int CGraph::edge(char prename[],char postname[],char name
    [],char color[],char style[],bool dir,int label,int
    part)
{
    int pre=-1,post=-1;
    SVertex *pV=this->pVert;

    while(pV!=NULL){
        if(streq(pV->name,prename))pre=pV->id;
        if(streq(pV->name,postname))post=pV->id;
```

```
        pV=pV->r;
    }
    if(pre==-1||post==-1)
        return 0;
    return edge(pre,post,name,color,style,dir,label,part);
}

void CGraph::tree(int inval,int outval,int depth,double shrinkexp,double shrinklin,double startarc,bool dir)
{
    int i;
    int k=inval+outval;
    double x=0.,y=0.;
    double koeff=1.;

    vertex(x,y,0.,"a");

    for(i=0;i<depth;i++)
    {
        koeff+=shrinklin;
        koeff*=shrinkexp;
    }

    for(i=0;i<k;i++)
        treenode(inval,outval,depth-1,i<inval,
        koeff*cos(2*i*M_PI/k+startarc),koeff*sin(2*i*M_PI/k+startarc),
            x,y,"a",i,shrinkexp,shrinklin,koeff/shrinkexp-shrinklin,dir);
}

void CGraph::edgetree(int inval,int outval,int depth,double shrinkexp,double shrinklin,double startarc,bool dir)
{
    int i;
    int k=inval+outval;
    double x=0.,y=0.;
    double koeff=1.;

    for(i=0;i<depth;i++)
    {
        koeff+=shrinklin;
```

```
        koeff*=shrinkexp;
    }

    vertex(x,y,0.,"a");

    treenode(inval,outval,depth,true,
            (koeff+shrinklin)*shrinkexp*cos(startarc),(koeff+
                shrinklin)*shrinkexp*sin(startarc),
            x,y,"a",0,shrinkexp,shrinklin,koeff,dir);
    for(i=1;i<k;i++)
        treenode(inval,outval,depth-1,i<inval,
            koeff*cos(2*i*M_PI/k+startarc),koeff*sin(2*i*M_PI
                /k+startarc),
            x,y,"a",i,shrinkexp,shrinklin,koeff/shrinkexp-
                shrinklin,dir);
}

void CGraph::treenode(int inval,int outval,int depth,bool
    edgedir,double x,double y,double prex,double prey,char
    prename[],int name_i,double shrinkexp,double shrinklin,
    double koeff,bool dir)
{
    int i,n;
    for(n=0;prename[n]!='\0';n++);
    char *newname=new char[n+3];
    for(i=0;i<n;i++)
        newname[i]=prename[i];
    newname[n]='_';
    newname[n+1]='0'+name_i;
    newname[n+2]='\0';

    if(depth==0)
    {
        vertex(x,y,0.,newname,false);
        edge(prename,newname,"","","dotted",false);
    }
    else
    {
        vertex(x,y,0.,newname);
        if(edgedir)
            edge(newname,prename,"","","",dir);
        else
            edge(prename,newname,"","","",dir);
```

```
    }

    if(depth<=0)
        return;

    int k=inval+outval;
    float arc=0;
    if(y==prey)
        if(x<prex) arc=0;
        else arc=M_PI;
    else
        if(x==prex)
            if(y<prey) arc=M_PI/2;
            else arc=3*M_PI/2;
        else
        {
            arc=atan((y-prey)/(x-prex));
            if(x>prex)
                arc+=M_PI;
        }

    for(i=1;i<k;i++)
        treenode(inval,outval,depth-1,i<inval+edgedir?1:0,
            x+koeff*cos(2*i*M_PI/k+arc),y+koeff*sin(2*i*M_PI/
                k+arc),
            x,y,newname,i,shrinkexp,shrinklin,koeff/shrinkexp
                -shrinklin,dir);

}

void CGraph::bitree(int inval1,int inval2,int outval1,int
    outval2,int depth,double shrinkexp,double shrinklin,
    double startarc,bool dir)
{
    int i;
    int k=inval1+outval1;
    double x=0.,y=0.;
    double koeff=1.;

    vertex(x,y,0.,"a",true,1);

    for(i=0;i<depth;i++)
    {
```

```
        koeff+=shrinklin;
        koeff*=shrinkexp;
    }

    for(i=0;i<k;i++)
        bitreenode(inval1,inval2,outval1,outval2,2,depth-1,i<inval1,
        koeff*cos(2*i*M_PI/k+startarc),koeff*sin(2*i*M_PI/k+startarc),
            x,y,"a",i,shrinkexp,shrinklin,koeff/shrinkexp-shrinklin,dir);
}

void CGraph::biedgetree(int inval1,int inval2,int outval1,int outval2,int depth,double shrinkexp,double shrinklin,double startarc,bool dir)
{
    int i;
    int k=inval1+outval1;
    double x=0.,y=0.;
    double koeff=1.;

    for(i=0;i<depth;i++)
    {
        koeff+=shrinklin;
        koeff*=shrinkexp;
    }

    vertex(x,y,0.,"a",true,1);

    bitreenode(inval1,inval2,outval1,outval2,2,depth,true,
            (koeff+shrinklin)*shrinkexp*cos(startarc),(koeff+shrinklin)*shrinkexp*sin(startarc),
            x,y,"a",0,shrinkexp,shrinklin,koeff,dir);
    for(i=1;i<k;i++)
        bitreenode(inval1,inval2,outval1,outval2,2,depth-1,i<inval1,
            koeff*cos(2*i*M_PI/k+startarc),koeff*sin(2*i*M_PI/k+startarc),
            x,y,"a",i,shrinkexp,shrinklin,koeff/shrinkexp-shrinklin,dir);
}

```

```
void CGraph::bitreenode(int inval1,int inval2,int outval1,int outval2,int part,int depth,bool edgedir,double x,double y,double prex,double prey,char prename[],int name_i,double shrinkexp,double shrinklin,double koeff,bool dir)
{
    int i,n;
    for(n=0;prename[n]!='\0';n++);
    char *newname=new char[n+3];
    for(i=0;i<n;i++)
        newname[i]=prename[i];
    newname[n]='_';
    newname[n+1]='0'+name_i;
    newname[n+2]='\0';

    if(depth==0)
    {
        vertex(x,y,0.,newname,false,part);
        edge(prename,newname,"","","dotted",false);
    }
    else
    {
        vertex(x,y,0.,newname,true,part);
        if(edgedir)
            edge(newname,prename,"","","",dir);
        else
            edge(prename,newname,"","","",dir);
    }

    if(depth<=0)
        return;

    float arc=0;
    if(y==prey)
        if(x<prex) arc=0;
        else arc=M_PI;
    else
        if(x==prex)
            if(y<prey) arc=M_PI/2;
            else arc=3*M_PI/2;
        else
        {
            arc=atan((y-prey)/(x-prex));
```

```
                if(x>prex)
                    arc+=M_PI;
            }
        int k;
        if(part==1)
        {
            k=inval1+outval1;
            for(i=1;i<k;i++)
                bitreenode(inval1,inval2,outval1,outval2,2,depth
                    -1,i<inval1+edgedir?1:0,
                    x+koeff*cos(2*i*M_PI/k+arc),y+koeff*sin(2*i*
                        M_PI/k+arc),
                    x,y,newname,i,shrinkexp,shrinklin,koeff/
                        shrinkexp-shrinklin,dir);
        }
        else
        {
            k=inval2+outval2;
            for(i=1;i<k;i++)
                bitreenode(inval1,inval2,outval1,outval2,1,depth
                    -1,i<inval2+edgedir?1:0,
                    x+koeff*cos(2*i*M_PI/k+arc),y+koeff*sin(2*i*
                        M_PI/k+arc),
                    x,y,newname,i,shrinkexp,shrinklin,koeff/
                        shrinkexp-shrinklin,dir);
        }


}

void CGraph::DMS(int part)
{
    SVertex *pV=pVert;
    SEdge *pE=pEdge;
    double x,y,z;
    char name[10];

    while(pE!=NULL){
        x=0.;y=0.;z=0.;
        getstraightmean(pE->id,x,y,z);
        sprintf(name,"dms%d\0",pE->id);
        vertex(x,y,-1.,name,visibleverts(pE->id),3);
        pE=pE->r;
```

```
    }
    while(pV!=NULL){
        if(pV->partition==part){
            pE=pEdge;
            while(pE!=NULL){
                if(pE->preid==pV->id)
                    DMSpath(pE,pE,2,true);
                if(pE->postid==pV->id)
                    DMSpath(pE,pE,2,false);
                pE=pE->r;
            }
        }
        pV=pV->r;
    }
}

void CGraph::DMSpath(SEdge *pStart,SEdge *pCurrent,int n,
    bool direction)
{
    if(n<=0){
        char prename[10],postname[10];
        sprintf(prename,"dms%d\0",pStart->id);
        sprintf(postname,"dms%d\0",pCurrent->id);
        edge(prename,postname,"","blue","bend left=30",true
            ,0,1);
        return;
    }
    else
    {
        SEdge *pE=pEdge;
        if(direction)
        {
            while(pE!=NULL){
                if(pE!=pCurrent)
                {
                    if(pE->preid==pCurrent->postid)
                        DMSpath(pStart,pE,n-1,true);
                    if(pE->postid==pCurrent->postid)
                        DMSpath(pStart,pE,n-1,false);
                }
                pE=pE->r;
            }
        }
```

```
        else
        {
            while(pE!=NULL){
                if(pE!=pCurrent)
                {
                    if(pE->preid==pCurrent->preid)
                        DMSpath(pStart,pE,n-1,true);
                    if(pE->postid==pCurrent->preid)
                        DMSpath(pStart,pE,n-1,false);
                }
                pE=pE->r;
            }
        }
    }
}

void CGraph::HH(int part)
{
    SVertex *pV=pVert;
    SEdge *pE=pEdge;
    double x,y,z;
    int Zn,i;
    char name[10];

    while(pE!=NULL){
        x=0.;y=0.;z=0.;
        getstraightmean(pE->id,x,y,z);
        sprintf(name,"hh%d\0",pE->id);
        vertex(x,y,-1.,name,visibleverts(pE->id),3);
        pE=pE->r;
    }
    Zn=0;
    while(pV!=NULL){
        if(pV->partition==part)
        {
            pE=pEdge;
            while(pE!=NULL){
                if(pE->preid==pV->id || pE->postid==pV->id)
                    Zn++;
                pE=pE->r;
            }
            pV=NULL;
            break;
```

```
            }
            pV=pV->r;
        }
        pV=pVert;
        while(pV!=NULL){
            if(pV->partition==part)
            {
                pE=pEdge;
                i=0;
                while(pE!=NULL){
                    if(pE->preid==pV->id || pE->postid==pV->id)
                    {
                        label_edge(pE->id,i);
                        i++;
                    }
                    pE=pE->r;
                }
            }
            pV=pV->r;
        }
        pV=pVert;
        while(pV!=NULL){
            if(pV->partition==part){
                pE=pEdge;
                while(pE!=NULL){
                    if(pE->preid==pV->id)
                        HHpath(pE,pE,2,true,Zn);
                    if(pE->postid==pV->id)
                        HHpath(pE,pE,2,false,Zn);
                    pE=pE->r;
                }
            }
            pV=pV->r;
        }
}

void CGraph::HHpath(SEdge *pStart,SEdge *pCurrent,int n,
    bool direction,int Zn)
{
    SEdge *pE=pEdge;
    if(n<=0){
        char prename[10],postname[10];
        sprintf(prename,"hh%d\0",pStart->id);
```

```
            sprintf(postname,"hh%d\0",pCurrent->id);
            edge(prename,postname,"","blue","bend left=30",true,0,1);
            return;
        }
        if(n==1)
        {
            if(direction)
            {
                while(pE!=NULL){
                    if(pE!=pCurrent)
                    {
                        if(pE->preid==pCurrent->postid && (pCurrent->label+1)%Zn==pE->label)
                            HHpath(pStart,pE,n-1,true,Zn);
                        if(pE->postid==pCurrent->postid && (pCurrent->label+1)%Zn==pE->label)
                            HHpath(pStart,pE,n-1,false,Zn);
                    }
                    pE=pE->r;
                }
            }
            else
            {
                while(pE!=NULL){
                    if(pE!=pCurrent)
                    {
                        if(pE->preid==pCurrent->preid && (pCurrent->label+1)%Zn==pE->label)
                            HHpath(pStart,pE,n-1,true,Zn);
                        if(pE->postid==pCurrent->preid && (pCurrent->label+1)%Zn==pE->label)
                            HHpath(pStart,pE,n-1,false,Zn);
                    }
                    pE=pE->r;
                }
            }
        }
        if(n>1)
        {
            if(direction)
            {
                while(pE!=NULL){
```

```
                if(pE!=pCurrent)
                {
                    if(pE->preid==pCurrent->postid)
                        HHpath(pStart,pE,n-1,true,Zn);
                    if(pE->postid==pCurrent->postid)
                        HHpath(pStart,pE,n-1,false,Zn);
                }
                pE=pE->r;
            }
        }
        else
        {
            while(pE!=NULL){
                if(pE!=pCurrent)
                {
                    if(pE->preid==pCurrent->preid)
                        HHpath(pStart,pE,n-1,true,Zn);
                    if(pE->postid==pCurrent->preid)
                        HHpath(pStart,pE,n-1,false,Zn);
                }
                pE=pE->r;
            }
        }
    }
}

void CGraph::AC(int n)
{
    if(!(n&1)||n<3)return;
    int i;
    char styleleft[27];
    sprintf(styleleft,"very thick, bend left=%d",90/n-2);
    char styleright[28];
    sprintf(styleright,"very thick, bend right=%d",90/n-2);
    SVertex *pVa;
    SVertex *pVb;
    SVertex *pVA=getpVert(vertex(1,0));;
    SVertex *pVB=getpVert(vertex(-1,0));;
    SVertex *pVa_=pVA;
    SVertex *pVb_=pVB;

    ACrek(n,pVB,pVA,0);
    for(i=1;i<n;i++)
```

```
    {
        pVa=getpVert(vertex(cos(i*M_PI/n),sin(i*M_PI/n)));
        pVb=getpVert(vertex(-cos(i*M_PI/n),-sin(i*M_PI/n)));
        if(i&1)
        {
            edge(pVa_->id,pVa->id,"","red",styleright);
            edge(pVb->id,pVb_->id,"","green",styleleft);
            ACrek(n,pVa,pVb,i==n/2?2:0);
        }
        else
        {
            edge(pVa->id,pVa_->id,"","green",styleleft);
            edge(pVb_->id,pVb->id,"","red",styleright);
            ACrek(n,pVb,pVa,i==n/2?2:0);
        }
        pVa_=pVa;
        pVb_=pVb;
    }
    edge(pVA->id,pVb->id,"","green",styleleft);
    edge(pVa->id,pVB->id,"","red",styleright);
}

void CGraph::ACrek(int n,SVertex *pVA,SVertex *pVB,int depth)
{
    double Hx=(pVB->x+pVA->x)/2.;
    double Hy=(pVB->y+pVA->y)/2.;
    double HBx=pVB->x-Hx;
    double HBy=pVB->y-Hy;
    double HB=sqrt(HBx*HBx+HBy*HBy);
    const double r=0.5+1.5*HB;
    double Nx=r*(-HBy)/HB;
    double Ny=r*HBx/HB;
    double X1x=Hx+Nx;
    double X1y=Hy+Ny;
    double X2x=Hx-Nx;
    double X2y=Hy-Ny;
    double alpha=atan(HB/r);
    double phi=M_PI/2.-alpha;
    double s=sqrt(r*r+HB*HB);

    int i;
    char styleleft[22];
```

```
    sprintf(styleleft,"thick, bend left=%d",(int)((float)(180*alpha/(n*M_PI))));
    char styleright[23];
    sprintf(styleright,"thick, bend right=%d",(int)((float)(180*alpha/(n*M_PI))));
    SVertex *pVa;
    SVertex *pVb;
    SVertex *pVa_=pVA;
    SVertex *pVb_=pVB;
    for(i=1;i<n;i++)
    {
        if(depth>0)
        {
            pVa=getpVert(vertex(X1x-cos(phi+2*i*alpha/n)*HBx*s/HB+sin(phi+2*i*alpha/n)*(-Nx)*s/r,
                                X1y-cos(phi+2*i*alpha/n)*HBy*s/HB+sin(phi+2*i*alpha/n)*(-Ny)*s/r));
            pVb=getpVert(vertex(X2x+cos(phi+2*i*alpha/n)*HBx*s/HB-sin(phi+2*i*alpha/n)*(-Nx)*s/r,
                                X2y+cos(phi+2*i*alpha/n)*HBy*s/HB-sin(phi+2*i*alpha/n)*(-Ny)*s/r));
        }
        else if(i==1 || i==n-1)
        {
            pVa=getpVert(vertex(X1x-cos(phi+2*i*alpha/n)*HBx*s/HB+sin(phi+2*i*alpha/n)*(-Nx)*s/r,
                                X1y-cos(phi+2*i*alpha/n)*HBy*s/HB+sin(phi+2*i*alpha/n)*(-Ny)*s/r,
                                1,"",false));
            pVb=getpVert(vertex(X2x+cos(phi+2*i*alpha/n)*HBx*s/HB-sin(phi+2*i*alpha/n)*(-Nx)*s/r,
                                X2y+cos(phi+2*i*alpha/n)*HBy*s/HB-sin(phi+2*i*alpha/n)*(-Ny)*s/r,
                                1,"",false));
        }
        if(i&1)
        {
            if(depth>0)
            {
                edge(pVa_->id,pVa->id,"","red",styleright);
                edge(pVb->id,pVb_->id,"","green",styleleft);
```

```
                ACrek(n,pVa,pVb,i==n/2?depth-1:0);
            }
            else if(i==1) /* (1) */
            {
                edge(pVa_->id,pVa->id,"","red!30",styleright);
                edge(pVb->id,pVb_->id,"","green!30",styleleft);
            }
        }
        else
        {
            if(depth>0)
            {
                edge(pVa->id,pVa_->id,"","green",styleleft);
                edge(pVb_->id,pVb->id,"","red",styleright);
                ACrek(n,pVb,pVa,i==n/2?depth-1:0);
            }
/*          else //This code together with removing the condition i==1 at (1) would draw the full alternating cycle rather than just the initial edges.
            {
                edge(pVa->id,pVa_->id,"","green!30",styleleft);
                edge(pVb_->id,pVb->id,"","red!30",styleright);
            }
*/      }
        pVa_=pVa;
        pVb_=pVb;
    }
    if(depth>0)
    {
        edge(pVA->id,pVb->id,"","green",styleleft);
        edge(pVa->id,pVB->id,"","red",styleright);
    }
    else
    {
        edge(pVA->id,pVb->id,"","green!30",styleleft);
        edge(pVa->id,pVB->id,"","red!30",styleright);
    }
}

void CGraph::sequences(int n,int m,int depth)
{
```

```
    if(pVert!=NULL)
        return;
    if(n<1||m<1||n>9||m>9)
        return;
    char *vertexname;
    vertexname=new char[2*depth+3];

    int i,r=1;
    for(i=0;i<depth;i++)
        r*=(n*m);

    for(i=0;i<r;i++)
        if(makeseq(i,n,m,depth,vertexname)<=depth)
            vertex(seqx(n,m,depth,vertexname),seqy(depth,vertexname),0,vertexname);

    SVertex *pInVert;
    SVertex *pOutVert;

    pInVert=pVert;
    while(pInVert!=NULL){
        pOutVert=pVert;
        while(pOutVert!=NULL){
            if(check_seq(pInVert->name,pOutVert->name,depth))
                edge(pInVert->id,pOutVert->id,"","",
                (pInVert->y-pOutVert->y)*(pInVert->y-pOutVert->y)>1.1&&
                (pInVert->x==0.&&pOutVert->x==0. || pInVert->x!=pOutVert->x)?
                ((pInVert->x-pOutVert->x)*(pInVert->x-pOutVert->x)*(pInVert->y-pOutVert->y)*(pInVert->y-pOutVert->y)>=0?
                "bend left=15":"bend right=15"):"");
            pOutVert=pOutVert->r;
        }
        pInVert=pInVert->r;
    }
}

void CGraph::tensor(CGraph *a,CGraph *b)
{
    if(a==this || b==this)
        return;
```

```
    if(a->pVert==NULL || b->pVert==NULL)
        return;

    int navert=a->plastVert->id+1;
    int nbvert=b->plastVert->id+1;

    SVertex *pAVert=NULL;
    SVertex *pBVert=NULL;
    SEdge *pAEdge=NULL;
    SEdge *pBEdge=NULL;

    pAVert=a->pVert;
    while(pAVert!=NULL){
        pBVert=b->pVert;
        while(pBVert!=NULL){
            vertex(pAVert->x+pBVert->x,pAVert->y+pBVert->y,
                pAVert->z+pBVert->z);
            pBVert=pBVert->r;
        }
        pAVert=pAVert->r;
    }

    pAEdge=a->pEdge;
    while(pAEdge!=NULL){
        pBEdge=b->pEdge;
        while(pBEdge!=NULL){
            edge(pBEdge->preid+pAEdge->preid*nbvert,pBEdge->
                postid+pAEdge->postid*nbvert);
            pBEdge=pBEdge->r;
        }
        pAEdge=pAEdge->r;
    }
}

void CGraph::strech(double x,double y,double z)
{
    SVertex *pv=pVert;
    while(pv!=NULL)
    {
        pv->x*=x;
        pv->y*=y;
        pv->z*=z;
        pv=pv->r;
```

```
    }
}

void CGraph::rotxy(double arc)
{
    SVertex *pv=pVert;
    double x,y;
    while(pv!=NULL)
    {
        x=pv->x;
        y=pv->y;
        pv->x=x*cos(arc)-y*sin(arc);
        pv->y=x*sin(arc)+y*cos(arc);
        pv=pv->r;
    }
}

void CGraph::rotxz(double arc)
{
    SVertex *pv=pVert;
    double x,z;
    while(pv!=NULL)
    {
        x=pv->x;
        z=pv->z;
        pv->x=x*cos(arc)-z*sin(arc);
        pv->z=x*sin(arc)+z*cos(arc);
        pv=pv->r;
    }
}

void CGraph::rotyz(double arc)
{
    SVertex *pv=pVert;
    double y,z;
    while(pv!=NULL)
    {
        y=pv->y;
        z=pv->z;
        pv->y=y*cos(arc)-z*sin(arc);
        pv->z=y*sin(arc)+z*cos(arc);
        pv=pv->r;
    }
```

```
}

void CGraph::style_edge(char edgestyle[], int part)
{
    int i=0,n;
    while(edgestyle[i]!='\0')i++;
    n=i+1;
    SEdge *pE=pEdge;
    while(pE!=NULL){
        if(pE->partition==part || part==-1)
        {
            if(pE->style!=NULL)
                delete pE->style;
            pE->style=new char[n];
            for(i=0;i<n;i++) pE->style[i]=edgestyle[i];
        }
        pE=pE->r;
    }
}

void CGraph::color_edge(char edgecolor[], int part)
{
    int i=0,n;
    while(edgecolor[i]!='\0')i++;
    n=i+1;
    SEdge *pE=pEdge;
    while(pE!=NULL){
        if(pE->partition==part || part==-1)
        {
            if(pE->color!=NULL)
                delete pE->color;
            pE->color=new char[n];
            for(i=0;i<n;i++) pE->color[i]=edgecolor[i];
        }
        pE=pE->r;
    }
}

void CGraph::label_edge(int id, int label)
{
    SEdge *pE=pEdge;
    while(pE!=NULL){
        if(pE->id==id)
```

```
        {
            pE->label=label;
            return;
        }
        pE=pE->r;
    }
}

void CGraph::edge_color_delta()
{
    const char colors[10][15]={"red","green","brown","blue","black","orange","MyLightMagenta","cyan","yellow","pink"};
    int i=0;
    SEdge *pE=pEdge;
    while(pE!=NULL){
        if(pE->color!=NULL)
            delete pE->color;
        pE->color=NULL;
        pE=pE->r;
    }
    pE=pEdge;
    while(pE!=NULL){
        if(pE->color==NULL)
        {
            edge_color_delta_rek(pE,colors[i%10],i/10);
            i++;
        }
        pE=pE->r;
    }
}

void CGraph::edge_color_delta_rek(SEdge *pE,const char color[],int i)
{
    SEdge *pE_=pEdge;

    if(pE->color!=NULL) return;

    int n=0;
    while(color[n]!='\0') n++;
    n++;
    pE->color=new char[n];
```

```
    for(n--;n>=0;n--)pE->color[n]=color[n];

    while(pE_!=NULL){
        if( (pE->postid==pE_->postid || pE->preid==pE_->
            preid) && pE_->color==NULL )
            edge_color_delta_rek(pE_,color,i);
        pE_=pE_->r;
    }
}

void CGraph::edge_color_delta_part(int part)
{
    const char colors[10][15]={"red","green","brown","blue"
        ,"black","orange","MyLightMagenta","cyan","yellow","
        pink"};
    int i=0;
    SEdge *pE=pEdge;
    while(pE!=NULL){
        if(pE->partition==part)
        {
            if(pE->color!=NULL)
                delete pE->color;
            pE->color=NULL;
        }
        pE=pE->r;
    }
    pE=pEdge;
    while(pE!=NULL){
        if(pE->color==NULL && pE->partition==part)
        {
            edge_color_delta_part_rek(pE,colors[i%10],i/10,
                part);
            i++;
        }
        pE=pE->r;
    }
}

void CGraph::edge_color_delta_part_rek(SEdge *pE,const
    char color[],int i,int part)
{
    SEdge *pE_=pEdge;

```

```
    if(pE->color!=NULL || pE->partition!=part) return;

    int n=0;
    while(color[n]!='\0') n++;
    n++;
    pE->color=new char[n];
    for(n--;n>=0;n--)pE->color[n]=color[n];

    while(pE_!=NULL){
        if( (pE->postid==pE_->postid || pE->preid==pE_->
            preid) && pE_->color==NULL && pE_->partition==
            part)
            edge_color_delta_part_rek(pE_,color,i,part);
        pE_=pE_->r;
    }
}

void CGraph::getstraightmean(int id,double &x,double &y,
    double &z)
{
    SEdge *pE=pEdge;
    SVertex *pV=pVert;
    x=y=z=0;
    while(pE!=NULL){
        if(pE->id==id){
            while(pV!=NULL){
                if(pE->preid==pV->id){
                    x+=pV->x;
                    y+=pV->y;
                    z+=pV->z;
                }
                if(pE->postid==pV->id){
                    x+=pV->x;
                    y+=pV->y;
                    z+=pV->z;
                }
                pV=pV->r;
            }
            x/=2;
            y/=2;
            z/=2;
            return;
        }
```

```
        pE=pE->r;
    }
}

bool CGraph::visibleverts(int id)
{
    SEdge *pE=pEdge;
    SVertex *pV=pVert;
    while(pE!=NULL){
        if(pE->id==id){
            while(pV!=NULL){
                if((pE->preid==pV->id || pE->postid==pV->id)&&!pV->draw) return false;
                pV=pV->r;
            }
            return true;
        }
        pE=pE->r;
    }
    return false;
}

void CGraph::paint(char Datei[],char Befehl[],double scale,bool name)
{
    FILE *fp;
    fp=fopen(Datei,"w");

    fprintf(fp,"\\def\\%s{\n",Befehl);
    fprintf(fp,"\\begin{tikzpicture}\n");
    fprintf(fp,"[scale=%0.2f,\n",scale);
    fprintf(fp,"inner␣sep=1.5,\n");
    fprintf(fp,"vertex/.style={circle,draw=black!50,fill=black!20,␣very␣thick},\n");
    fprintf(fp,"parta/.style={circle,draw=black!50,fill=black!0,␣very␣thick},\n");
    fprintf(fp,"partb/.style={circle,draw=black!50,fill=black!100,␣very␣thick},\n");
    fprintf(fp,"edgevertex/.style={circle,draw=blue!80,fill=blue!40,␣very␣thick},\n");
    fprintf(fp,"normal/.style=,\n");
    fprintf(fp,"post/.style={->,␣>=stealth'},\n");
    fprintf(fp,"pre/.style={<-,␣>=stealth'}]\n");
```

```

    SVertex *pv=pVert;
    SEdge *pe=pEdge;

    int nedge=plastEdge==NULL?0:plastEdge->id+1;
    int nvert=plastVert==NULL?0:plastVert->id+1;
    bool *vertdrawn;
    bool *edgedrawn;
    double *z;
    if(nvert>0)
    {
        vertdrawn=new bool[nvert];
        z=new double[nvert];
    }
    if(nedge>0)
        edgedrawn=new bool[nedge];

    int zcount=0,i;
    bool zexists;

    for(i=0;i<nvert;i++)
        vertdrawn[i]=false;
    for(i=0;i<nedge;i++)
        edgedrawn[i]=false;

    while(pv!=NULL)
    {
        zexists=false;
        for(i=0;i<zcount;i++)
            if(pv->z==z[i])
            {
                zexists=true;
                break;
            }
        if(!zexists)
        {
            z[zcount]=pv->z;
            zcount++;
        }
        pv=pv->r;
    }
    if(nvert>0)
        qsort(z,zcount,sizeof(double),doublecomp);
```

```

    for(i=0;i<zcount;i++)
    {
        pv=pVert;
        while(pv!=NULL)
        {
            if(pv->z==z[i])
            {
                fprintf(fp,"\\node");
                if(name){
                    fprintf(fp,"(a%d) at (%f,%f) {\\tiny %s};\n",pv->id,pv->x+pv->z*z_x,pv->y+pv->z*z_y,pv->name);
                }
                else
                {
                    if(pv->draw)
                    {
                        switch(pv->partition)
                        {
                        case 1: fprintf(fp,"[parta]");break;
                        case 2: fprintf(fp,"[partb]");break;
                        case 3: fprintf(fp,"[edgevertex]");break;
                        default: fprintf(fp,"[vertex]");break;
                        }

                    }
                    fprintf(fp,"(a%d) at (%f,%f) {};\n",pv->id,pv->x+pv->z*z_x,pv->y+pv->z*z_y);
                }

                vertdrawn[pv->id]=true;
            }
            pv=pv->r;
        }
        pe=pEdge;
        while(pe!=NULL)
        {
            if(vertdrawn[pe->preid]&&vertdrawn[pe->postid]&&!edgedrawn[pe->id])
            {
                fprintf(fp,"\\draw [");
```

```
                if(pe->directed)
                    fprintf(fp,"post");
                else
                    fprintf(fp,"normal");
                if(pe->color!=NULL && pe->color[0]!='\0') fprintf(fp,",%s",pe->color);
                if(pe->style!=NULL && pe->style[0]!='\0') fprintf(fp,",%s",pe->style);
                if(pe->preid==pe->postid) fprintf(fp,", loop");
                fprintf(fp,"] (a%d) to (a%d);\n",pe->preid,pe->postid);
                edgedrawn[pe->id]=true;
            }
            pe=pe->r;
        }
    }
    fprintf(fp,"\\end{tikzpicture}\n}\n");
    fclose(fp);
    delete z;
}

bool check_simpleseq(const char a[],const char b[])
{
    int i;
    for(i=0;b[i+1]!='\0'&&a[i]!='\0';i++)
        if(a[i]!=b[i+1])
            return false;

    return true;
}

int seqx(int n,int m,int depth,char vertexname[])
{
    int x=0,i,t;
    bool isnull=true;
    if(seqy(depth,vertexname)>=0)
    {
        for(i=1;i<2*depth+1;i++)
            if(vertexname[i]!='0')
                isnull=false;
        if(isnull)
            return 0;
```

```
        i=0;
        while(vertexname[i]=='0') i++;
        x+=vertexname[i]-'0'-1;
        for(t=i+depth,i++;i<t;i++)
        {
            x*=(i<depth+1?n:m);
            x+=vertexname[i]-'0';
        }
    }
    else
    {
        i=2*depth+1;
        while(vertexname[i]=='0') i--;
        x+=vertexname[i]-'0'-1;
        for(t=i-depth,i--;i>t;i--)
        {
            x*=(i>=depth+1?m:n);
            x+=vertexname[i]-'0';
        }
    }
    return x;
}

int seqy(int depth,char vertexname[])
{
    int y=0,i;
    for(i=0;vertexname[i]=='0'&&i<depth+1;i++) y--;
    for(i=2*depth+1;vertexname[i]=='0'&&i>=depth+1;i--) y
        ++;
    return y;
}

bool check_seq(const char a[],const char b[],int depth)
{
    int i;
    for(i=0;i<depth;i++)
    {
        if(a[i]!=b[i+1]) return false;
        if(a[depth+1+i]!=b[depth+1+i+1]) return false;
    }
    return true;
}

```

```
int makeseq(int i,int n,int m,int depth,char vertexname[])
{
    int l=2*depth+2,j;
    for(j=2*depth;j>=depth+1;j--)
    {
        vertexname[j]='0'+i%m;
        i/=m;
    }
    for(;j>=0;j--)
    {
        vertexname[j]='0'+i%n;
        i/=n;
    }
    vertexname[0]=vertexname[2*depth+1]='0';
    vertexname[2*depth+2]='\0';
    for(j=0;vertexname[j]=='0'&&j<2*depth+2;j++) l--;
    for(j=2*depth+1;vertexname[j]=='0'&&j>0;j--) l--;
    if(l<0)l=0;
    return l;
}

#endif
```

List of Figures

List of Tables

Curriculum Vitae

Personal Data

Name:	Christoph Marx
Date of Birth:	May 30, 1981
Place of Birth:	Klosterneuburg, Lower Austria, Austria
Nationality:	Austria
Current Residence:	Vienna, Austria
Current Email:	`christophnikolaus1981@gmail.com`

Education

Diploma Studies:	Mathematics, University of Vienna, October 2001 – July 2010
Exchange Semester:	University of Leeds, September 2005 – January 2006
High school diploma:	BORG Vienna XX Unterbergergasse, June 2000
High school:	BORG Vienna XX Unterbergergasse, September 1998 – June 2000
Technical high school:	Federal technical institution for studies and experiments for informatics and organization Vienna V Spengergasse, September 1995 – June 1998
Secondary school:	Hauptschule Hermannstraße, Klosterneuburg, September 1991 – June 1995
Elementary school:	Volksschule Weidling, Weidling, September 1987 – June 1991

Military Service

Military Service:	Magdeburg–Barracks, Klosterneuburg, January 2001 – September 2001

Language Skills

German:	native
English:	business fluent

Work Experience

Internship, Informatics:	AI-Informatics, July 1997
Tutor, Mathematics:	University of Vienna, Department for Mathematics, September 2002 – June 2004
Internship, Mathematics:	Allianz Investment Bank, May 2008 – August 2008

References

[1] Peter J. Cameron, Cheryl E. Praeger and Nicholas C. Wormald:
Infinite Highly Arc Transitive Digraphs and Universal Covering Digraphs
Combinatorica 13 (4) (1993) 377-396

[2] Norbert Seifter:
Transitive digraphs with more than one end
Discrete Mathematics 308 (2008) 1531-1537

[3] Aleksander Malnič, Dragan Marušič, Rögnvaldur G. Möller, Norbert Seifter, Vladimir Trofimov, Boris Zgrablič:
Highly arc transitive digraphs: reachability, topological groups
European Journal of Combinatorics 26 (2005) 19-28

[4] Rögnvaldur G. Möller:
Descendants in highly arc transitive digraphs
Discrete Mathematics 247 (2002) 147-157

[5] Matt DeVos, Bojan Mohar and Robert Šámal:
Reachability in highly arc-transitive digraphs
Preprint

[6] David M. Evans:
An Infinite Highly Arc-transitive Digraph
European Journal of Combinatorics 18 (1997) 281-286

[7] Aleksander Malnič, Dragan Marušič, Norbert Seifter and Boris Zgrablić:
Highly arc–transitive digraphs with no homomorphism onto $\mathbb{Z}$
Combinatorica 22 (3) (2002) 435-443

[8] Norbert Seifter and Wilfried Imrich:
A note on the growth of transitive graphs
Descrete Mathematics 73 (1988/89) 111-117 North-Holland

[9] Aleksander Malnič, Dragan Marušič, Norbert Seifter, Primož Šparl and Boris Zgrablič:
Reachability relations in digraphs
European Journal of Combinatorics 29 (2008) 1566-1581

[10] Sònia P. Mansilla:
An infinite family of sharply two–arc transitive digraphs
Electronic Notes in Discrete Mathematics 29 (2007) 243-247

[11] Cheryl E. Praeger:
On homomorphic images of edge transitive directed graphs
Australasian Journal of Combinatorics 3 (1991) 207-210

[12] Rögnvaldur G. Möller:
Structure Theory of Totally Disconnected Locally Compact Groups via Graphs and Permutations
Canadian Journal of Mathematics Vol. 54 (4), 2002 pp. 795-827

[13] Matthias Hamann and Fabian Hundertmark:
A classification of connected–homogeneous digraphs
arXiv:1004.5273v1 [math.CO] 29 Apr 2010

[14] Gert Sabidussi:
On a class of fixed–point–free graphs
Proceedings of the American Mathematical Society 9 1958 800-804

[15] Reinhard Diestel and Imre Leader:
A conjecture concerning a limit of non-Cayley graphs
Journal of Algebraic Combinatorics 14 (2001), no. 1, 17–25

[16] Martin John Dunwoody:
Cutting up Graphs
Combinatoica 2 (1) (1982) 15–23

Index