The Construction of Cubic Maps on Orientable Surfaces

Dissertation zur Erlangung des Grades eines Dr. math.
der Fakultät für Mathematik der Universität Bielefeld

eingereicht von

Dipl.–Math. Thomas Harmuth

Universität Bielefeld
Fakultät für Mathematik
FSP Mathematisierung

im Februar 2000

Universität Bielefeld
Graduiertenkolleg Strukturbildungs–prozesse

Erster Gutachter: Dr. habil. Gunnar Brinkmann

Zweiter Gutachter: Prof. Dr. Andreas Dress

Datum der mündlichen Prüfung: 12. Mai 2000

to Rita

Gedruckt auf alterungsbeständigem Papier ∞ ISO 9706.

Preface

In this text, constructive definitions of several classes of cubic maps, as well as methods to avoid constructing pairwise isomorphic maps are given. In the first part, inductive definitions of all simple cubic toroidal maps, as well as of those simple cubic toroidal maps containing only triangles, squares, pentagons, hexagons, and heptagons, are given. In the second part, an algorithm for the construction of restricted classes of cubic maps is presented. The restrictions are to be given by the user. The user can determine the genus of the maps in the class, the vertex number and the permitted faces. Moreover, the user can give further restrictions concerning connectivity and simpleness or incidence relationships between faces of certain degrees.

To every definition and algorithm, there exists a corresponding computer program which can be used to actually construct the class of maps. The computer programs are described in the appendix.

In order to make reading this text as easy as possible, the text is designed as follows:

- **Bold letters** are used for mathematical terms that are defined in this text and are of importance in the whole text. These defintions should be easy to find.
- *Italic letters* are used for mathematical terms that are not defined in this text or are of local importance, only. Moreover, italic letters are used for names, in particular for names of computer programs, and for emphasized words. Sometimes, italic letters are used for mathematical terms that were defined earlier in this text. This is done whenever the term is also a term in colloquial speech and thus could be mixed up.
- Sequences of numbers are embraced in French braces "$\langle$" and "$\rangle$". An example is the following sequence: $\langle a_0, a_1, a_2, a_3, a_4\rangle$.
- Sometimes dots are used in order to abbreviate sequences, e.g. $\langle a_s, a_{s+1}, \ldots, a_t\rangle$. In this case, we often omit also the second entry and write $\langle a_s, \overset{\nearrow}{\ldots}, a_t\rangle$ instead of $\langle a_s, a_{s+1}, \ldots, a_t\rangle$ and $\langle a_s, \overset{\searrow}{\ldots}, a_t\rangle$ instead of $\langle a_s, a_{s-1}, \ldots, a_t\rangle$. In particular, sequences like $\langle a_2, \overset{\nearrow}{\ldots}, a_1\rangle$ and $\langle b_1, \overset{\searrow}{\ldots}, b_2\rangle$ are empty. Moreover, we will frequently use sequences where the index is increased or decreased only after every second or third entry. For example, $\langle a_0, b_0, \overset{\nearrow}{\ldots}, a_3, b_3\rangle$ denotes the sequence $\langle a_0, b_0, a_1, b_1, a_2, b_2, a_3, b_3\rangle$, and $\langle a_3, b_3, c_3, \overset{\searrow}{\ldots}, a_0, b_0, c_0\rangle$ denotes the sequence $\langle a_3, b_3, c_3, a_2, b_2, c_2, a_1, b_1, c_1, a_0, b_0, c_0\rangle$. Finally, it should be mentioned that the expansion of the dots must be done with respect to those equally indexed entries of the sequence that are closest to the dots. For example, $\langle a_5, b_7, c_{23}, a_4, b_4, \overset{\searrow}{\ldots}, a_1, b_1, c_3, a_0, b_0\rangle$ denotes the sequence $\langle a_5, b_7, c_{23}, a_4, b_4, a_3, b_3, a_2, b_2, a_1, b_1, c_3, a_0, b_0\rangle$ because the entries that are closest to the dots have the indices 4 and 1, and the neighbours of these entries have the same indexes. Hence in the expansed sequence, each index must occur twice, too.
- In this text, many indexed variables occur. In the term a_i, the i is an index with variable value. On the other hand, sometimes suffixes that do not provide any value are provided

with the variables, e.g. in the term f_M. Here M denotes a fixed object, e.g. a map. If it is necessary to have an index and a suffix for one variable, then the suffix is placed above the variable. An example is h_i^u. Here, i is an index and u is a suffix.

- The set $\mathbb{N}$ does not contain the element 0. The set $\mathbb{N}_0$ contains the element 0.

This text was not written by a native English speaker. Moreover, no native English speaker has ever been in my debt so much that I could have asked him to proof read this text. However, I am convinced that most readers prefer sSCE (= standard Scientific Community English) compared with proper German. Concerning the remaining mistakes in this text, I would like to give the kind readers' spirits a lift: enjoy the mistakes, the mathematics in this text is rough enough...

The work in hand was supported by:

- Graduiertenkolleg "Strukturbildungsprozesse"/Deutsche Forschungsgemeinschaft (DFG)
- Studienstifung des deutschen Volkes

I would like to thank my supervisors Dr. Gunnar Brinkmann and Prof. Dr. Andreas Dress for the extraordinary support throughout the past five years. Moreover, since I used the computers in our department exhaustively, I want to thank the staff of the computer laboratory for "keeping the engines running". In order to finish the computations in time so that the results could be included in this text, I also used the computers in other departments exhaustively. I would like to thank Prof. Brendan D. McKay of the Australian National University (Canberra) very much for offering me processing time on the computers in his department and for conscientiously supervising and maintaining the processes that ran on these computers. Moreover, I would like to thank Prof. Tomaž Pisanski of the University of Ljubljana (Slovenia) very much for offering me processing time on the computers in his department. A special thanks goes to Drago Bokal of the University of Ljubljana for adjusting *CGF* to the *Windows NT* requirements and for supervising and maintaining the processes that ran on the *Windows NT* cluster in his department. Without this world–wide assistance, I would not have been able to include the results of the computations in this text. Finally, I would like to thank Dr. Olaf Delgado Friedrichs for computing and drawing the tilings that correspond to the Delaney graphs presented in Section 10.2.

The computer programs that are described in this text as well as the text itself are available at the following URLs:

- `http://www.mathematik.uni-bielefeld.de/~harmuth/Welcome.html`
- `http://people.freenet.de/thomas.harmuth`
- `http://www.harmuths.de`

Bielefeld, February 2000

Thomas Harmuth

Contents

Chapter 1

Introduction

1.1 Basic definitions

The following definitions are mostly standard definitions in graph theory. For convenience, they are briefly recalled, and extended or modified when necessary. For an extensive introduction into graph theory, see e.g. [1].

A **graph** $G = (V, H, E, T)$ is a quadruple consisting of two disjoint sets V and H, a function $T : H \to V \times V$, and a partition E of H (considered as a subset of $\mathcal{P}(H)$), satisfying the following conditions: V is finite and non–empty. H is finite (and may be empty). Every subset $e \subseteq H$ in E has cardinality two, and $u, v \in V$, $h, h' \in H$, $\{h, h'\} \in E$, and $T(h) = (u, v)$ implies $T(h') = (v, u)$. The elements of V are called **vertices**, the elements of H are called **semi–edges**, and the elements of E are called **edges**. The semi–edges h and h' of an edge $e = \{h, h'\} \in E$ are called **elements** of e. If $\{h, h'\} \in E$, then the semi–edge h is called the **inverse** of h' (denoted by $\overline{h} = h'$) and vice versa. The cardinality of V (denoted by $|V|$) is called the **order** of G. In a semi–edge h with $T(h) = (u, v)$, the vertex u is called the **source** and v is called the **target** of h. For convenience, we will often say "*G contains x*" if $x \in V$, $x \in H$, or $x \in E$.

An edge $e = \{h, h'\} \in E$ with $T(h) = T(h')$ ($\Rightarrow T(h) = (v, v)$ for some $v \in V$) is called a **loop**. If there exist $n > 1$ pairwise distinct edges $e_1, \ldots, e_n$ with $T(h_1) = T(h_2) = \ldots = T(h_n)$ for some $h_i \in e_i$, $i \in \{1, \ldots, n\}$ (and hence $T(h'_1) = \ldots = T(h'_n)$ for $h'_i := \overline{h_i}$, $i \in \{1, \ldots, n\}$), then the union of these edges is a **multiple edge**. In particular, if $n = 2$, then $\{e_1, e_2\}$ is a **double edge**, and if $n = 3$, then $\{e_1, e_2, e_3\}$ is a **triple edge**. If G contains neither multiple edges nor loops, then G is called **simple**.

If $G = (V, H, E, T)$ and $G' = (V', H', E', T')$ are graphs, then G' is called a **subgraph** of G if and only if $V' \subseteq V$, $H' \subseteq H$, $E' \subseteq E$, and $T'(h) = T(h)$ for every $h \in H'$. G is called a **supergraph** of G' if and only if G' is a subgraph of G. The **induced subgraph** of $G = (V, H, E, T)$ **on** $V' \subseteq V$ is the subgraph $G' = (V', H', E', T')$ whose semi–edge set H' contains every semi–edge $h \in H$ for which both, source and target, are elements of V'.

Two vertices $u, v \in V$ of a graph $G = (V, H, E, T)$ are **adjacent** (to each other) if and only if there exists a semi–edge $h \in H$ with $T(h) = (u, v)$. Note that, due to this definition, a vertex

v is adjacent to itself exactly if there is a loop $e = \{h, h'\}$ with $T(h) = T(h') = (v, v)$. A vertex that is adjacent to a vertex v is called a **neighbour** of v. A vertex v is **incident** with a semi–edge h (and vice versa) if and only if v is the source of h. A vertex v is **incident** with an edge $e = \{h, h'\}$ (and vice versa) if and only if v is incident with h or h'. Two semi–edges are **adjacent** (to each other) if and only if they have the same source. Two edges $e = \{h, h'\}$ and $f = \{k, k'\}$ with $e \neq f$ are **adjacent** (to each other) if and only if they are incident with a common vertex v, i.e. v is the source of h or h' and of k or k'. Thus, at least one of the semi–edges h, h' is adjacent to (at least) one of the semi–edges k, k'. The **valency** $\mathrm{val}(v)$ of a vertex v is given by the number of semi–edges having the source v. Thus, the valency of v is the number of edges that are incident with v with loops being counted twice. A vertex of valency i is called i**–valent**. A graph G is i**–regular** if and only if each vertex of G is i–valent. A 3–regular graph is called **cubic**.

A **path of length** n in G is a sequence $p = \langle h_0, h'_0, h_1, h'_1, \ldots, h_{n-1}, h'_{n-1}\rangle$ of semi–edges with the following properties:

- $e_i := \{h_i, h'_i\} \in E$ for every $i \in \{0, \ldots, n-1\}$
- h'_i is adjacent to h_{i+1} for every $i \in \{0, \ldots, n-2\}$
- the sources of h_i, $i \in \{1, \ldots, n-1\}$, are pairwise distinct (while there is no restriction to the source of h_0 and to the target of h_{n-1})
- $e_0, \ldots, e_{n-1}$ are pairwise distinct

The edges $e_0, \ldots, e_{n-1}$ are called **elements** of the path. This definition of a path differs from the classical definition. However, we will need it exactly this way. The definition implies that a loop l can be an element of a path only if $l = e_0$ or $l = e_{n-1}$. Let v_i, $i \in \{0, \ldots, n-1\}$, be the source of h_i. Let v_n be the target of h_{n-1}. The vertex v_0 is called the **start vertex** and v_n is called the **end vertex** of p. The edge e_0 is called the **start edge** and e_{n-1} is called the **end edge** of p. The semi–edge h_0 is called the **start semi–edge** and h'_{n-1} is called the **end semi–edge** of p. These definitions emphasize that a path is implicitly directed. The start vertex is the source of the start semi–edge and the end vertex is the source of the end semi–edge.

If $v_0 = v_n$ and $v_0 \neq v_i$ for every $i \in \{1, \ldots, n-1\}$, then the path $p = \langle h_0, h'_0, \ldots, h_{n-1}, h'_{n-1}\rangle$ defined as above is called a **cycle of length** n or just an n**–cycle**. Note that, in a cycle, the start edge and the end edge are defined as before. Although it appears to be contraintuitive, it is sometimes important to have a start edge and an end edge in a cycle (see Sections 6.3 and 7.1).

The **inverse** of a path $p = \langle h_0, h'_0, \ldots, h_{n-1}, h'_{n-1}\rangle$ is given by $\overline{p} = \langle h'_{n-1}, h_{n-1}, \ldots, h'_0, h_0\rangle$. Two paths are **incident** (with each other) if and only if they have at least one common element. An edge is **incident** with a path (and vice versa) if and only if it is an element of the path. A semi–edge is **incident** with a path (and vice versa) if and only if it is an element of an edge which is incident with the path. A vertex v is **incident** with a path (and vice versa) if and only if an edge that is incident with v is an element of the path. Clearly, two cycles are **incident** (with each other) if and only if they have at least one common element, a vertex v is **incident** with a cycle if and only if an edge that is incident with v is an element of the cycle etc. For convenience, we will sometimes say "*p contains x*" if x is a vertex, a semi–edge, or an edge that is incident with p.

If the graph G is simple, then — as an abbreviation — we may denote the path $p = \langle h_0, h'_0, \ldots, h_{n-1}, h'_{n-1}\rangle$ also by $p = \langle e_0, \ldots, e_{n-1}\rangle$ because the orders of the elements of the e_i in the sequence are unique. This is trivial because three consecutive vertices v_i, v_{i+1}, and v_{i+2} (denoted as above) are pairwise distinct.

If there exists a path p with start vertex u and end vertex v (or vice versa) for two vertices u and v, then the vertices u and v are called **connected** (by the path p). A graph G is called **connected** if and only if its vertices are pairwise connected. Otherwise G is called **disconnected**. A **connected component** $G' = (V', H', E', T')$ of a graph $G = (V, H, E, T)$ is a connected induced subgraph of G on V' where the induced subgraph of G on $V' \cup \{v\}$ is disconnected for every $v \in V - V'$.

There are some standard operations that transform a graph into another graph. One of these operations is the **removal** of an edge $e = \{h, h'\} \in E$ from a graph $G = (V, H, E, T)$. This operation transforms G into a graph $G' = (V, H', E', T')$ with $H' = H - \{h, h'\}$, $E' = E - \{e\}$, and $T' = T|_{H'}$.

Another operation is the **deletion** of a vertex v. If a vertex $v \in V$ is deleted in a graph $G = (V, H, E, T)$, then all edges that are incident with v are removed, too. Hence for the resulting graph $G' = (V', H', E', T')$, it holds that G' is the induced subgraph of G with $V' = V - \{v\}$.

A third operation is the **contraction** of an edge $e = \{h, h'\}$ where e must not be a loop. Contracting e in G means that the two adjacent vertices u and v of e are identified. Let $x \notin V \cup H$ be this new vertex. Hence for the resulting graph $G' = (V', H', E', T')$, it holds that $V' = (V \cup \{x\}) - \{u, v\}$, $H' = H - \{h, h'\}$, $E' = E - \{e\}$, and $T' : H' \to V' \times V'$ with

$$T'(h) = \begin{cases} (x, x) & \text{if } T(h) \in \{(v, v), (u, u), (u, v), (v, u)\}, \\ (x, a) & \text{if } a \notin \{u, v\} \text{ and } T(h) \in \{(v, a), (u, a)\}, \\ (a, x) & \text{if } a \notin \{u, v\} \text{ and } T(h) \in \{(a, v), (a, u)\}, \\ T(h) & \text{else.} \end{cases}$$

A fourth operation is the **removal** of a 2–valent vertex v. This operation is not allowed if v is incident with a loop. Otherwise the two incident edges $a = \{l, l'\}$ and $b = \{k, k'\}$ are substituted in the following way: v is incident with two vertices u, w. Let $j, j' \notin V \cup H$ and $c := \{j, j'\}$. The resulting graph $G' = (V', H', E', T')$ is given by $V' = V - \{v\}$, $H' = (H - \{l, l', k, k'\}) \cup \{j, j'\}$, $E' = (E - \{a, b\}) \cup \{c\}$, and $T' : H' \to V' \times V'$ with

$$T'(h) = \begin{cases} (u, w) & \text{if } h = j, \\ (w, u) & \text{if } h = j', \\ T(h) & \text{else.} \end{cases}$$

The removal of a 2–valent vertex v can also be realized by contracting an incident edge of v.

A graph G of order at least $i + 1$ is called i–**connected** if and only if it is not possible to transform G into a disconnected graph by deleting less than i vertices. The definition implies that an i–connected graph is j–connected for every $j \in \mathbb{N}$, $j \leq i$. A graph G is called **exactly** i–**connected** if it is i–connected but not $(i + 1)$–connected.

Two graphs $G_1 = (V_1, H_1, E_1, T_1)$ and $G_2 = (V_2, H_2, E_2, T_2)$ are called **isomorphic** if and only if there exists an **incidence preserving isomorphism** $f : V_1 \to V_2$, i.e. there exists a bijection $g : H_1 \to H_2$ such that $T_1(h) = (u, v)$ implies $T_2(g(h)) = (f(u), f(v))$ for every $h \in H_1$ (u, v not necessarily distinct). An isomorphism $f : V_1 \to V_1$ is called an **automorphism**.

A **rotation system** of a graph $G = (V, H, E, T)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ is a collection $\pi = \{\pi_{v_1}, \pi_{v_2}, \ldots, \pi_{v_n}\}$ such that π_{v_i} is a transitive cyclic permutation of the semi–edges incident with v_i. The permutation π_{v_i} is called the π–**clockwise orientation around** v_i. Let h be a semi–edge incident with a vertex v. Then the **rotation sequence** of v (with respect to π) is given by $r_\pi(v) = \langle h, \pi_v(h), \pi_v^2(h), \ldots, \pi_v^{\mathrm{val}(v)-1}(h)\rangle$ where π_v^j denotes the j–th iterated application of the function π_v. Depending on the choice of h, the sequence is cyclically shifted. The corresponding sequences are not distinguished.

Since every semi–edge is incident with exactly one vertex, the vertex v in the expression $\pi_v(h)$ is always unique: v is always the source of h. Hence π induces a function $\Pi : H \to H$ with $\Pi(h) = \pi_v(h)$ for all $h \in H$ where v is the source of h. Hence for all $h \in H$, it holds that h and $\Pi(h)$ have the same source, and if v is the source of h, then $\Pi^j(h) \neq h$ for all $j \in \mathbb{N}$ with $j < \mathrm{val}(v)$. Π is called a **semi–edge rotation** of G. In the following text, we will use both π and Π in order to give the orientation around a vertex. We will use π or Π depending on whether it is useful or cumbersome to denote the source of a semi–edge explicitly.

A **map** M is a pair $M = (G, \Pi)$ where $G = (V, H, E, T)$ is a *connected* graph and Π is a semi–edge rotation of G. G is called the **underlying** graph of M. M is called **cubic**, **(exactly)** i–**connected**, **simple** if and only if G has these properties. The **order** of M is defined as the order of G. A vertex, edge, or semi–edge of G is called a vertex, edge, or semi–edge of M, too. For convenience, we will often say "*M contains x*" if $x \in V$, $x \in H$, $x \in E$, or x is a path in M or a submap of M. The four operations introduced above — deletion of a vertex, contraction of an edge, removal of an edge, removal of a vertex — are extended with respect to Π in the obvious manner. Since the result of these operations is not necessarily connected, the result is not necessarily a map but a *bunch of maps*: A **bunch of maps** B is a pair $B = (G, \Pi)$ where G is a graph and Π is a semi–edge rotation of G. For every connected component $G' = (V', H', E', T')$ of G, the pair $(G', \Pi|_{H'})$ is a map. Thus a bunch of maps can easily be transformed into a set of maps. $(G', \Pi|_{H'})$ is called a **connected component** of B. In a bunch of maps (and hence also in a map), π and E imply T: Every semi–edge h has exactly one source and hence it is incident with exactly one vertex. Thus h occurs in the rotation sequence of exactly one vertex v. Analogously, the inverse $\overline{h}$ occurs in the rotation sequence of exactly one vertex u which is the target of h. Hence $T(h) = (v, u)$ and $T(h') = (u, v)$. So if the incidence structure of a map must be described and π is provided, then T does not need to be listed. It can be evaluated using π and E.

The **mirror image** $\overline{M}$ of a map $M = (G, \Pi)$ is given by $\overline{M} = (G, \overline{\Pi})$ where $\overline{\Pi}$ is the inversion of Π. Analogously, $\overline{\pi_v}$ denotes the inversion of π_v for every $v \in V$. Two maps $M = (G, \Pi)$ and $M' = (G', \Pi')$ are **isomorphic** if and only if $G = (V, H, E, T)$ and $G' = (V', H', E', T')$ are isomorphic and there exists an incidence preserving isomorphism $f : V \to V'$ and a bijection $g : H \to H'$ as above with one of the two following properties:

- For every $h_1, h_2 \in H$ with $\pi_v(h_1) = h_2$, it holds $\pi'_{f(v)}(g(h_1)) = g(h_2)$.
- For every $h_1, h_2 \in H$ with $\pi_v(h_1) = h_2$, it holds $\overline{\pi'_{f(v)}}(g(h_1)) = g(h_2)$.

Such a pair (f, g) is called a **map isomorphism**. If $V = V'$, then (f, g) is called a **map automorphism**. In the first case, (f, g) is called **orientation preserving**, and in the second case (f, g) is called **orientation reversing**. A map M is called **specular** if and only if at least one of the map automorphisms mapping M onto $\overline{M}$ is orientation preserving.

We do not normally distinguish between isomorphic maps.

A map automorphism *acts* on vertices, semi–edges, and edges, i.e. it maps vertices on vertices, semi–edges on semi–edges, and edges on edges. Similarly, paths are mapped on paths. Two vertices (semi–edges, edges, paths) are **isomorphic in** M if and only if both vertices (semi–edges, edges, paths) are in the same *orbit* of the automorphism group. If we do not want to name M, then we may also say that the two considered objects are **map isomorphic**.

If $M = (G, \Pi)$ with $G = (V, H, E, T)$ and $M' = (G', \Pi')$ with $G' = (V', H', E', T')$ are maps, then M' is called a **submap** of M if and only if G' is a subgraph of G and the semi–edge rotation Π' corresponds to Π in the following way: Let $v \in V'$ be n–valent in G and m–valent in G', and let $\langle h_1, \ldots, h_n\rangle$ be the rotation sequence of v with respect to Π. Let $j_1, \ldots, j_m$ be the indices for which $h_{j_k} \in H'$ ($k \in \{1, \ldots, m\}$), in increasing order. Then $\langle h_{j_1}, \ldots, h_{j_m}\rangle$ is the rotation sequence of v with respect to Π'. If this holds for every vertex $v \in V'$, then M' is a submap of M. M is called a **supermap** of M' if and only if M' is a submap of M. M' is called an **induced submap** of M if and only if M' is a submap of M and G' is an induced subgraph of G.

A **face** F in a map $M = (G, \Pi)$ with $G = (V, H, E, T)$ is a sequence $F = \langle v_0, h_0, h'_0, \ldots, v_{r-1}, h_{r-1}, h'_{r-1}\rangle$ of vertices $v_i \in V$ and semi–edges $h_i, h'_i \in H$ ($i \in \{0, \ldots, r-1\}$) where the indices are expressed modulo r, $e_i := \{h_i, h'_i\} \in E$, $T(h_i) = (v_i, v_{i+1})$, $T(h'_i) = (v_{i+1}, v_i)$, and $\pi_{v_{i+1}}(h'_i) = h_{i+1}$ for every $i \in \{0, \ldots, r-1\}$. We do not distinguish between cyclically shifted sequences, i.e. all sequences in $\{\langle v_k, h_k, h'_k, \ldots, v_{k+r-1}, h_{k+r-1}, h'_{k+r-1}\rangle | k \in \{0, \ldots, r-1\}\}$ (all indices modulo r) are considered to be identical. Moreover, for convenience a sequence may begin with a semi–edge instead of a vertex if it is clear which entries of the sequence correspond to vertices and which entries correspond to semi–edges. In either case, every third entry is a vertex and the remaining entries are semi–edges. The edges $e_0, \ldots, e_{r-1}$ are called the **elements** of F. The number r is the **degree** of F. The degree of F is denoted by $\deg(F)$. A face of degree 1,2,3,4,5,6,... is called a **monogon**, **digon**, **triangle**, **square**, **pentagon**, **hexagon**,..., respectively. In general, a face of degree i is also called an i–**gon**. The set D_M of all face degrees that occur in M is called the **degree set** of M.

If M is simple, then — as an abbreviation — we may denote the face F also by $F = \langle e_0, e_1, \ldots, e_{r-1}\rangle$ because the vertices and the orders of the elements of the e_i in the original sequence are unique. This is trivial because three consecutive vertices v_i, v_{i+1}, and v_{i+2} are always pairwise distinct. In isomorphic maps M_1 and M_2, an orientation *preserving* map isomorphism (f, g) maps face sequences onto each other, or — roughly spoken — maps faces onto each other, while an orientation *reversing* map isomorphism maps face sequences onto inverses of face sequences.

A face F is called **incident** with an edge e (and vice versa) if and only if e is an element of F. In this case, we also say "*F contains e*". Two faces are called **incident** (with each other) if and only if they contain at least one common element (edge). Analogously, a face is called **incident** with a cycle (and vice versa) if and only if they contain at least one common element. Moreover, a face F_2 is called a **neighbour** of a face F_1 if and only if F_1 is incident with F_2. A vertex v is **incident** with a face F (and vice versa) if and only if one of the edges incident with v is an element of F. In this case, we also say "*F contains v*". A semi–edge h is **incident** with a face F (and vice versa) if and only if h is an element of an edge which is incident with F. In this case, we also say "*F contains h*". Finally, we say "*M contains F*" if F is a face in a map M.

Each edge $e = \{h, h'\}$ appears exactly twice in the union of all face sequences, once in the order $\langle \ldots, h, h', \ldots\rangle$ and once in the order $\langle \ldots, h', h, \ldots\rangle$. Hence each edge in a map is an element of at most 2 different faces. e is an element of only one face F if and only if e appears twice in the face sequence of F (i.e. $F = \langle \ldots, h, h', \ldots, h', h, \ldots \rangle$). Such an edge e is called a **face–bridge**.

A **zig–zag pseudo–path** of length n in a map $M = (G, \Pi)$ is a sequence $p = \langle h_0, h'_0, \ldots, h_{n-1}, h'_{n-1} \rangle$ of semi–edges with the two properties

- $e_i := \{h_i, h'_i\} \in E$ for every $i \in \{0, \ldots, n-1\}$,
- h'_i is adjacent to h_{i+1} for every $i \in \{0, \ldots, n-2\}$,

and p satisfies the following condition: Let v_i, $i \in \{1, \ldots, n-1\}$, be the source of h_i. Then for every $i \in \{1, \ldots, n-1\}$, it holds $\mathrm{val}(v_i) = 3$, and one of the following conditions is true:

- $\overline{\pi_{v_i}}(h'_{i-1}) = h_i$ for every odd i and $\pi_{v_i}(h'_{i-1}) = h_i$ for every even i
- $\pi_{v_i}(h'_{i-1}) = h_i$ for every odd i and $\overline{\pi_{v_i}}(h'_{i-1}) = h_i$ for every even i

In the first case, p is called **continued to the right**, and in the second case p is called **continued to the left**. For a path of length 1, it can be chosen if the path is continued to the left or to the right. In general, the i**–th continuation** is `right` if $\overline{\pi_{v_i}}(h'_{i-1}) = h_i$ and `left` if $\pi_{v_i}(h'_{i-1}) = h_i$ ($i \in \{1, \ldots, n-1\}$). The semi–edge h_0 is called the **start semi–edge** and h'_{n-1} is called the **end semi–edge** of p. If p is a path, i.e. $v_1, \ldots, v_{n-1}$ are pairwise distinct and $e_0, \ldots, e_{n-1}$ are pairwise distinct, then p is called a **zig–zag path**.

Let B be a bunch of maps. Then n_B, e_B, and f_B denote the number of vertices, edges, and faces of B, respectively. The (orientable) **genus** g_M of a map $M = (G, \Pi)$ is determined by **Euler's formula**

$$n_M - e_M + f_M = 2 - 2g_M \tag{1.1}$$

(see [1]). It always holds $g_M \geq 0$ as we will see later. A map of genus 0 is called **planar** and a map of genus 1 is called **toroidal**. If B is a bunch of maps, then g_B denotes the **sum** of the genera of the connected components of B (each defined by Euler's formula). If B is a map, then the definition is of course consistent with the previous definition of g_M.

1.2 Drawing a map

In order to give the reader some intuition about some arguments, we will frequently provide figures with this text. Most of these figures show maps. In this section, a brief overview how a map is *drawn* (i.e. represented topologically) is given. The topological background is considered extensively e.g. in [2] and [3].

A map M of genus g can be drawn on an orientable surface S of genus g where a **surface** is a connected compact 2–manifold without boundary. When $M = (G, \Pi)$ with $G = (V, H, E, T)$ is drawn on the surface S, then the vertices are visualized by pairwise distinct points on the surface. Let $p : V \to S$ be the (injective) function which maps the vertices on the surface. Every semi–edge h with $T(h) = (u, v)$ (u and v not necessarily distinct) is mapped on the surface by a homeomorphism $c(h) : [0, 1] \to R_h \subset S$ where $(c(h))(0) = p(u)$ and $(c(h))(1) = (c(\overline{h}))(1)$. The interiors of the resulting arcs are mutually disjoint and do not meet the points representing vertices, i.e. $\{(c(h_1))(x) \mid x \in (0,1)\} \cap \{(c(h_2))(x) \mid x \in [0,1]\} = \emptyset$ for every pair $h_1, h_2 \in H$ with $h_1 \neq h_2$. Moreover, $(c(h_1))(1) \neq (c(h_2))(1)$ for every $h_1, h_2 \in H$ with $h_2 \notin \{h_1, \overline{h_1}\}$. The arcs must be embedded in a way that by surrounding an embedded vertex $p(v)$ sufficiently close in clockwise direction, the embedded semi–edges incident with v are traversed in the order indicated by the rotation π_v. See Figure 1.1. These conditions cannot be satisfied if the genus of S is smaller than g, but they can always be satisfied if the genus of S is g.

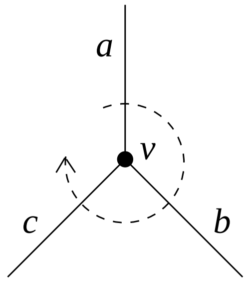

Figure 1.1: This drawing represents $r_\pi(v) = \langle a, b, c\rangle$

In such an embedding, every connected component C of $S - R_H$ (with $R_H = \bigcup_{h\in H} R_h$) is homeomorphic to the interior of a disk (this would not hold if the genus of M was smaller than the genus of S), and the boundary of C is the topological representation of the corresponding face in M. Thus there is a one–to–one correspondence between the connected components of $S - R_H$ and the faces of M. By tracing the sequence of a face, the corresponding connected component of $S - R_H$ in the drawing is surrounded. For a face–bridge $e = \{h, h'\}$, the two connected components on both sides of the arc $R_h \cup R_{h'}$ are identical.

For examples, see e.g. Figure 1.3 in Section 1.3.

In the figures, we may label edges either by the names of the edges themselves or by the names of their elements, i.e. by the names of the semi–edges. In the latter case, every edge $e = \{h, h'\}$ is labelled twice, and the labels h and h' are placed in a way that the label of a semi–edge h is placed closer to its corresponding arc than to the arc that corresponds to its inverse $\overline{h}$.

When drawing a map on an orientable surface, the resulting figure should be as transparent as possible. As shown in [2], Section 1.7, each orientable surface of genus g is homeomorphic to a connected sum of g tori if $g > 0$ and to a sphere if $g = 0$. The upper row of Figure 1.2 shows orientable surfaces of genus 0, 1, and 2, respectively, embedded into $\mathbb{R}^3$. Maps drawn on these surfaces tend not to be transparent at all. Hence we will now introduce *canonical forms* of the considered surfaces.

Let M be a planar map which is embedded into the sphere via (p, c). $S - R_H$ is non–empty and one can obtain a topologically equivalent embedding into the Euclidian plane by applying the well known *stereographic projection*. This works in both directions, hence in a book it is convenient to draw planar maps into the plane in order to get transparent figures even if an embedding into the sphere might be more relevant for some purposes, as we will see in Section 1.8.

We use a similar method also for surfaces of higher genus: The canonical form of an orientable surface of genus g is a $4g$–gon (compare [2]). As some examples, the lower row in Figure 1.2 shows the canonical forms of the surfaces in the upper row. These canonical forms are embedded into $\mathbb{R}^2$. Identically labelled edges are identified.

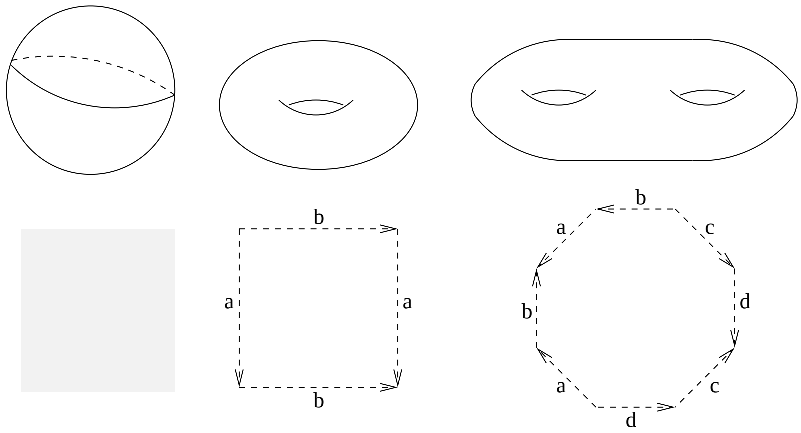

Figure 1.2: Orientable surfaces of genus 0, 1 and 2

1.3 Properties of maps

Non–isomorphic maps can have isomorphic underlying graphs. Thus the concept of maps is more general than the concept of graphs. As an example, take the three maps shown in Figure 1.3. The underlying graph of all three maps is the so–called *complete graph* on 4 vertices $K_4 = (\{1,2,3,4\}, \{a,a',b,b',c,c',d,d',e,e',f,f'\}, \{\{a,a'\},\{b,b'\},\{c,c'\},\{d,d'\},\{e,e'\},\{f,f'\}\}, T)$ with $T(a) = (1,2)$, $T(b) = (1,3)$, $T(c) = (1,4)$, $T(d) = (2,3)$, $T(e) = (2,4)$, and $T(f) = (3,4)$ (the other values of T follow from the given ones). The rotation sequences are as follows:

- Left map: $r_\pi(1) = \langle a,b,c\rangle$, $r_\pi(2) = \langle a',e,d\rangle$, $r_\pi(3) = \langle b',d',f\rangle$, $r_\pi(4) = \langle c',f',e'\rangle$
- Center map: $r_\pi(1) = \langle a,c,b\rangle$, $r_\pi(2) = \langle a',e,d\rangle$, $r_\pi(3) = \langle b',f,d'\rangle$, $r_\pi(4) = \langle c',f',e'\rangle$
- Right map: $r_\pi(1) = \langle a,c,b\rangle$, $r_\pi(2) = \langle a',d,e\rangle$, $r_\pi(3) = \langle b',d',f\rangle$, $r_\pi(4) = \langle c',e',f'\rangle$

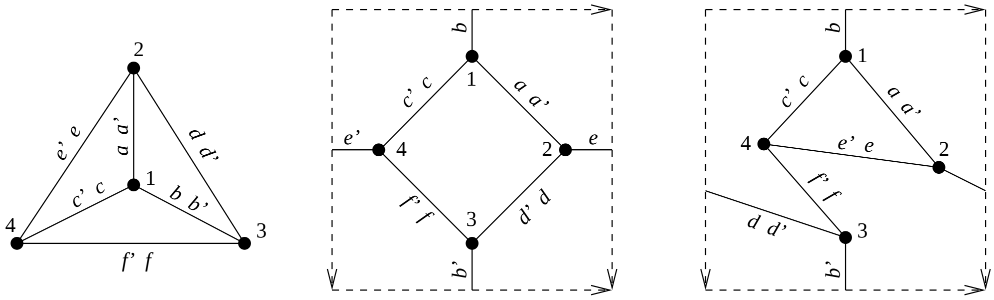

Figure 1.3: Three maps whose underlying graph is K_4.

The left map has genus 0 and the other two maps each have genus 1. Hence for one graph G, there might exist maps of different genera where the underlying graph is always G. The **genus** of a graph is defined as the smallest genus among all maps whose underlying graph is G. If the genus of a map is equal to the genus of its underlying graph, then the map is called **minimal**. In general, it is NP–complete to determine the genus of a graph [4]. However, given a map M of genus $g > 0$, it can be determined in linear time whether the underlying graph has genus at most $g - 1$, i.e. whether M is minimal [5]. Moreover, for a graph or a map, it can be verified in linear time if the (underlying) graph is planar [6]. In [7], a sufficient condition for a map to be minimal is given.

A 3–connected minimal map whose underlying graph is *planar* is unique (up to isomorphism) [8]. In general, this does not hold for maps of lower connectivity and/or of higher genus. However, in [7] a sufficient condition for a 3–connected minimal map to be unique is given. General results for surfaces of higher genera are much weaker, see e.g. [9].

1.4 Properties of cubic maps

For a cubic map M of given genus g_M, Euler's formula (1.1) yields a simple relation between the number of faces in M and the number of vertices in M: In cubic maps, every vertex is incident with 3 semi–edges and hence it always holds $e_M = \frac{3}{2}n_M$. So we have

$$f_M = \frac{1}{2}n_M + 2 - 2g_M \tag{1.2}$$

for cubic maps. Since f_M is always an integer, the order of cubic maps is always even. Since $f_M \geq 1$, we have $n_M \geq 4g_M - 2$ as a lower bound for the number of vertices in a map of genus g_M. This lower bound obviously is not sharp for $g_M = 0$.

By easy counting arguments (see [10]) applied to Euler's formula, we get the following formula: Let $p_d(M)$ be the number of faces of degree d in a cubic map M. Then it holds:

$$\sum_{d\geq 1} p_d(M) \cdot (6-d) = 12 \cdot (1 - g_M) \tag{1.3}$$

An easy consequence of this formula is that every planar map contains a face of degree at most 5 (otherwise the sum on the left side could not add up to 12), every toroidal map contains a face of degree at most 6 (otherwise the sum would be negative instead of 0) and a face of degree at least 6 (otherwise the sum would be greater than 0), and every map M with $g_M \geq 2$ contains a face of degree at least 7 (otherwise the sum would be non–negative, but $12 \cdot (1 - g_M) \leq -12$).

Finally, there are some simple results concerning the connectivity of M: If M is simple and $n_M > 4$, then M is at most 3–connected because the deletion of all 3 neighbours of an arbitrary vertex v disconnects v from the remaining vertices. If M contains a loop and $n_M > 2$, then M is exactly 1–connected: Let v be a vertex which is incident with a loop in M. Then v is adjacent to exactly one more vertex $u \neq v$, and the deletion of u transforms M into a disconnected bunch of maps. If M does not contain a loop but a multiple edge and if $n_M > 2$ (as a consequence, every multiple edge in M is a double edge because otherwise M would be disconnected), then M is at most 2–connected: Let u and v be the vertices that are incident with a double edge in M. Then u is adjacent to v and exactly one more vertex u', and v is adjacent to u and exactly one more vertex v'. The deletion of u and v' disconnects u' from v.

1.5 Two operations on maps

We will now introduce two additional standard operations on maps and see how they affect the genus of a map. Let $M = (G, \Pi)$ with $G = (V, H, E, T)$ be a map. The first of the two operations is the **subdivision** of an edge $e = \{h, h'\} \in E$ by a new 2–valent vertex $v' \notin V \cup H$. This operation transforms M into a map $M' = (G', \Pi')$, $G' = (V', H', E', T')$ with

$V' = V \cup \{v'\}$, $H' = H \cup \{k, k'\}$ (where $k, k' \notin V' \cup H$), $E' = (E - \{e\}) \cup \{\{h, k\}, \{h', k'\}\}$, and π' (and implicitly Π') defined as follows:

$$r_{\pi'}(v) = \begin{cases} r_\pi(v) & \text{if } v \in V, \\ \langle k, k' \rangle & \text{if } v = v'. \end{cases}$$

Remarks:

- As always, E' and π' imply T'.
- The *removal* of v' in M' transforms M' back into M.
- It holds $n_{M'} = n_M + 1$, $e_{M'} = e_M + 1$, and $f_{M'} = f_M$. Hence due to Euler's formula, it holds $g_{M'} = g_M$.

The second operation is the **insertion** of an edge $e = \{h^u, h^w\} \notin E$ (where $h^u, h^w \notin V \cup H$). The insertion of e transforms M into a map $M' = (G', \Pi')$ with $G' = (V', H', E', T')$ where $V' = V$, $H' = H \cup \{h^u, h^w\}$, and $E' = E \cup \{e\}$. If $H = \emptyset$ (and thus $V = \{v\}$), then we define $r_{\pi'}(v) = \langle h^u, h^w \rangle$. Let now $H \neq \emptyset$. Let $h \in H$ and $T(h) = (u, w)$. In a map, the insertion is not uniquely determined by $T(h)$. Let h_0^u be a semi–edge incident with u and h_0^w be a semi–edge incident with w (h_0^u and h_0^w exist because $H \neq \emptyset$ and M is connected). Then π' is uniquely defined with respect to h_0^u and h_0^w: For every vertex $v \notin \{u, w\}$, we define $r_{\pi'}(v) = r_\pi(v)$. Let $r_\pi(u) = \langle h_0^u, \dots, h_l^u \rangle$, $l \in \mathbb{N}_0$.

- If $u \neq w$, then let $r_\pi(w) = \langle h_0^w, \dots, h_m^w \rangle$, $m \in \mathbb{N}_0$. We define $r_{\pi'}(u) = \langle h^u, h_0^u, \dots, h_l^u \rangle$ and $r_{\pi'}(w) = \langle h^w, h_0^w, \dots, h_m^w \rangle$.
- If $u = w$ and $h_0^w \neq h_0^u$, then $h_0^w = h_i^u$ for some $i \in \{1, \dots, l\}$. We define $r_{\pi'}(u) = \langle h^u, h_0^u, \dots, h_{i-1}^u, h^w, h_i^u, \dots, h_l^u \rangle$.
- If $u = w$ and $h_0^w = h_0^u$, then we define $r_{\pi'}(u) = \langle h^u, h^w, h_0^u, \dots, h_l^u \rangle$.

Remarks:

- As always, E' and π' imply T'.
- The *removal* of e in M' transforms M' back into M.
- It holds $n_{M'} = n_M$ and $e_{M'} = e_M + 1$. The face numbers and the genera are related as follows: If $h_0^u = h_0^w$, then one face is modified and one new face $\langle u, h^w, h^u \rangle$ is obtained in M'. Hence $f_{M'} = f_M + 1$ and due to Euler's formula, it holds $g_{M'} = g_M$. If $h_0^u \neq h_0^w$, then let $F = \langle v_0, h_0^F, k_0^F, \dots, v_t, h_t^F, k_t^F \rangle$ with $h_0^u = h_0^F$ and $u = v_0$ be the face in M which is incident with h_0^u such that h_0^u follows u in the sequence of the face. It holds $h_l^u = k_t^F$.
  - If $h_0^w = h_i^F$ for some $i \in \{1, \dots, t\}$ ($h_0^w = h_0^F = h_0^u$ has already been considered), then $w = v_i$ and $k_{i-1}^F = h_m^w$. In this case, F in M is replaced by two new faces $F_1 = \langle v_0, h_0^F, k_0^F, \dots, v_{i-1}, h_{i-1}^F, k_{i-1}^F, v_i, h^w, h^u \rangle$ and $F_2 = \langle v_i, h_i^F, k_i^F, \dots, v_t, h_t^F, k_t^F, v_0, h^u, h^w \rangle$ in M'. See Figure 1.4 for an example with $t = 7$ and $i = 4$. Thus $f_{M'} = f_M + 1$ and hence $g_{M'} = g_M$ due to Euler's formula.
  - If $h_0^w \neq h_i^F$ for every $i \in \{0, \dots, t\}$, then let $D = \langle u_0, h_0^D, k_0^D, \dots, u_r, h_r^D, k_r^D \rangle$ be the face with $u_0 = w$ and $h_0^D = h_0^w$ so that h_0^w follows w in the sequence of D (moreover, $k_r^D = k_m^w$). In this case, F and D are replaced by one new face $F_1 = \langle v_0, h_0^F, k_0^F, \dots, v_t, h_t^F, k_t^F, v_0, h^u, h^w, u_0, h_0^D, k_0^D, \dots, u_r, h_r^D, k_r^D, u_0, h^w, h^u \rangle$ in M'. See Figure 1.5 for an example with $r = 3$ and $t = 3$. Thus $f_{M'} = f_M - 1$ and hence $g_{M'} = g_M + 1$ due to Euler's formula.

  In the second case, e is a face–bridge. So it can be stated that $g_{M'} = g_M + 1$ if and only if a face–bridge is inserted. On the other hand, by removing a face–bridge from a map M, the resulting bunch of maps B is either disconnected or B is a map with $g_B = g_M - 1$.

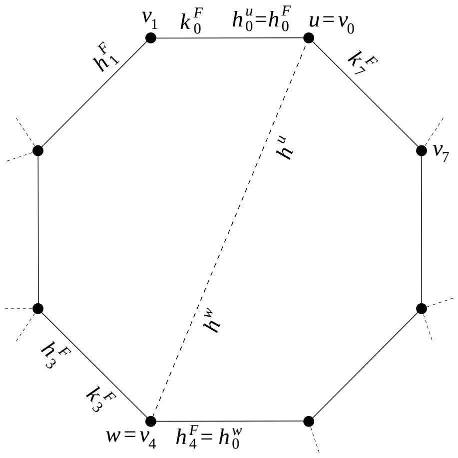

Figure 1.4

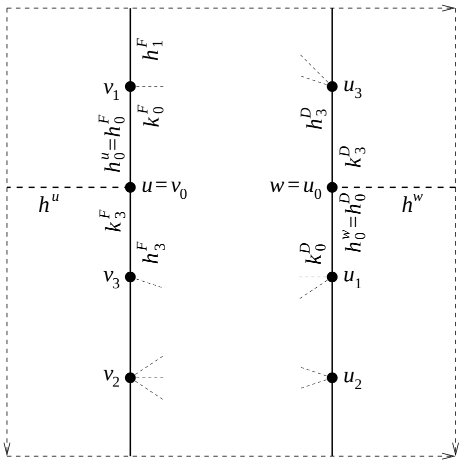

Figure 1.5

Since every map can be constructed out of the map $\hat{M} = ((\{v\}, \emptyset, \emptyset, \cdot), \cdot)$ by a sequence of insertions, subdivisions, and inverses of contractions, $g_{\hat{M}} = 0$ implies $g_M \geq 0$ for every map M. Moreover, it can be shown that if M' is a submap of M, then $g_{M'} \leq g_M$. For that, it suffices to show that the inverse of the contraction of an edge does not modify the genus of a map. This suffices because M' can be obtained out of M by a sequence of contractions (of edges where one incident vertex is 1–valent), edge removals, and vertex removals. After a contraction, for the resulting map $M' = (G', \Pi')$, $G' = (V', H', E', T')$, it holds $n_{M'} = n_M - 1$, $e_{M'} = e_M - 1$, and $f_{M'} = f_M$ and hence $g_{M'} = g_M$.

1.6 The "Jordan curve theorem"

In this section, we will provide and prove a combinatorial analogon to the well known Jordan curve theorem. For reasons of intuition, we will also call it Jordan curve theorem although it is much weaker in its importance than the topological Jordan curve theorem. We will provide it only for cubic maps because we will only need it for cubic maps. In cubic maps, the semi–edge rotation is quite simple so that the notation will be easier to follow than in the general case. However, the general case and the proof for it can easily be derived from the cubic case.

Theorem 1.1 (Jordan curve theorem) *Let $M = (G, \Pi)$ with $G = (V, H, E, T)$ be a cubic planar map. Let $c = \langle h_0, h'_0, \ldots, h_n, h'_n \rangle$ be a cycle in M. Let $k, k' \in H$ be semi–edges not incident with c but $\Pi(h_i) = k$ and $\overline{\Pi}(h_j) = k'$ for some $i, j \in \{0, \ldots, n\}$. Then for every path p with start semi–edge k and end edge k', it holds that p is incident with c.*

This theorem corresponds to the topological Jordan curve theorem in the following sense: The cycle c represents a Jordan curve, the semi–edges k and k' represent the two sides of the cycle, and the theorem states that it is not possible to get from a semi–edge on one side to a semi–edge on the other side without crossing the cycle. Figure 1.6 illustrates the situation. Here $i = 0$ and $j = 3$.

Proof: Let $p = \langle k_0, k'_0, \ldots, k_l, k'_l \rangle$ with $k = k_0$ and $k' = k'_l$ be a path which is not incident with c, i.e. $\{k_0, k_0, \ldots, k_l, k'_l\} \cap \{h_0, h'_0, \ldots, h_n, h'_n\} = \emptyset$. By removing all edges and vertices that are in-

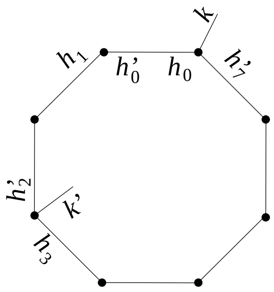

Figure 1.6

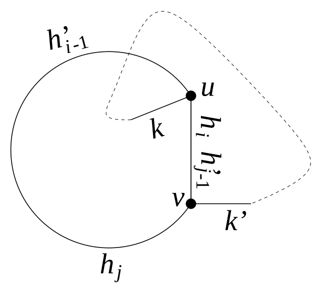

Figure 1.7

cident neither with p nor with c we obtain a submap M_1 of M with $g_{M_1} \leq g_M$ due to Section 1.5 (here vertices can be removed whenever they become 2–valent; this happens whenever an incident edge is removed). Now we remove all 2–valent vertices in M'. These are all vertices except the vertices u and v with $r_\pi(u) = \langle h_i, k, h'_{i-1}\rangle$ and $r_\pi(v) = \langle h_j, h'_{j-1}, k'\rangle$ (indices modulo $n+1$). We obtain a map $M_2 = (G_2, \Pi_2)$, $G_2 = (V_2, H_2, E_2, T_2)$, with $g_{M_2} = g_{M_1}$ due to Section 1.5. It holds $V_2 = \{u, v\}$, $H_2 = \{h_i, h'_{i-1}, k, k', h_j, h'_{j-1}\}$, and $E_2 = \{\{k, k'\}, \{h_i, h'_{j-1}\}, \{h'_{i-1}, h_j\}\}$, and M_2 contains only one face $\langle u, h_i, h'_{j-1}, v, k', k, u, h'_{i-1}, h_j, v, h'_{j-1}, h_i, u, k, k', v, h_j, h'_{i-1}\rangle$ (M_2 is shown in Figure 1.7). Hence due to Euler's theorem, it holds $g_{M_2} = 1$ and hence $g_M \geq 1$. This is a contradiction to the assumption $g_M = 0$. ■

1.7 The goal: The construction of classes of cubic maps

In the following parts of this text, some methods to construct several classes of cubic maps will be described. In mathematics, the two ways to define an infinite class of objects are given by

(1) listing the properties that every object belonging to the class has to satisfy, and by
(2) describing how the objects belonging to the class can be built from a finite class of so–called **basic objects** by a sequence of **generating rules**.

The second case is called an **inductive definition** of a class. An inductive definition can be used to develop an algorithm that constructs the class of objects or — in practice, e.g. if the algorithm is implemented as a computer program — at least a well–defined finite subclass.

There exist inductive definitions for several classes of graphs. Inductive definitions for some classes of cubic graphs are given in [11] and [12]. The objects that we are interested in are cubic *maps*. In part I of this text, the maps that are to be constructed are simple and toroidal. We will give an inductive definition of the class of all (pairwise non–isomorphic) simple cubic toroidal maps. This class is infinite, but if we limit the order of the maps, then we can construct all simple cubic toroidal maps up to that order with the help of a computer program. This computer program has been developed due to the definition in part I of this text. It is called *togen*. Moreover, we will give a constructive definition of the class of those simple cubic toroidal maps whose faces all have a degree at most 7. This definition can be used to construct all corresponding maps with a limited order as well. The corresponding computer program is called *togen7*. To be precise, we should say that we construct one representative of each isomorphism class of maps.

In part II, we describe an algorithm which is able to construct all (pairwise non–isomorphic) cubic maps having a given vertex number, a given genus and a given set of permitted faces, i.e. for every possible face degree it can be prescribed if this face degree is permitted, and if it is permitted, then it can be prescribed how many faces of that degree must be contained at least and/or at most in every constructed map. The vertex number, the genus, and the set of permitted faces are to be chosen by the user. The corresponding computer program is called *CGF*. In principle, the algorithm can be used to construct all cubic maps (i.e. one representative of each isomorphism class) with a given vertex number and a given genus. However, the algorithm is most useful if the set of permitted faces is small because for the construction of all cubic maps, there exist other powerful algorithms, e.g. *togen* for toroidal and planar maps. For *CGF*, the underlying definition of the desired class of maps describes how the maps can be built from a finite class of basic objects, but it is no *inductive* definition in a strict sense because the basic objects themselves do not belong to the desired class of maps, i.e. they are no cubic maps.

1.8 Applications

Algorithms that construct[1] a certain class of maps are always helpful to test new conjectures on this class of maps or on a superclass before publishing the conjectures. But apart from that general application, there are also some more specific applications for the above algorithms.

Cubic maps are of interest in chemistry. Some molecular structures are modelled using cubic maps. The most famous molecular structures that can be represented by cubic maps are *fullerenes*. A fullerene is a spherically shaped molecule built of carbon atoms such that each carbon atom has bonds to exactly 3 other carbon atoms. The atoms form 5- and 6–(membered)–rings. If every atom is represented by a vertex and every bond is represented by a (single) edge, then one obtains a cubic planar map where the degree of each face is either 5 or 6. Such a map is called a *combinatorial fullerene*.

There already exists an algorithm that has been designed to construct combinatorial fullerenes [13]. However, not only fullerenes with 5- and 6–rings are of interest in chemistry, but also *generalized fullerenes* where the ring sizes differ. Moreover, the molecules need not have a spherical shape, but can e.g. have a toroidal shape. In 1997, several chemists succeeded in synthesizing a fullerene–like structure with a toroidal shape [14]. However, the exact structure, i.e. the exact arrangement of carbon rings in the molecule, was unknown. An algorithm that can construct all "candidates" (structural isomers) is helpful to determine the exact structure of such a molecule. Energy computations on the constructed theoretical structures can be compared with the properties of the synthesized structure. To this end, it is necessary to have a chemically "plausible" topological embedding of a structure, i.e. bond lengths and angles between bonds within a structure should vary as few as possible. In particular, the torus must be embedded in $\mathbb{R}^3$. Examples for calculations on sets of structural isomers of fullerenes and generalized fullerenes can be found in [15, 16, 17, 18, 19, 20].

In order to construct all structural isomers, *CGF* as well as *togen* are applicable. For energetical reasons, it is likely for the toroidal molecule to have only 5-, 6- and 7–rings. While it seems not to be possible to give a strict inductive definition of those cubic toroidal maps where each face is a pentagon, a hexagon, or a heptagon, the definition used by *togen7* gives the best known

[1] or "generate"

approximation to this task. Both *CGF* and *togen7* provide only the combinatorial structures, i.e. they provide $M = (G, \Pi)$. Providing chemically plausible coordinates for the vertices, i.e. providing a topological embedding (p, c) on a surface S, is a task which is done neither by *CGF* nor by *togen7*. This task is not trivial at all. For the sphere, a so–called *spring embedder* designed by O. Delgado Friedrichs can be found in the program package *cage* [21].

CGF has been used to answer a concrete question posed by the chemist Prof. P.W. Fowler (Exeter/Great Britain): Do there exist cubic maps of genus 2 such that each face is a heptagon? The answer is: Yes, there exist exactly 56 of them. If it must hold for every two faces F_1 and F_2 that F_1 and F_2 share at most one edge,[2] then there remain 6 maps which satisfy this condition. Figure 1.8 shows one of them. These 6 maps can be obtained by verifying the desired property for all 56 maps. In this case, the filtering is easy. However, the condition can also be included directly into the algorithm (see Section 8.5) which makes the algorithm faster. It took 5.9 seconds on an Intel Pentium II computer with 350 MHz to construct the 56 maps.

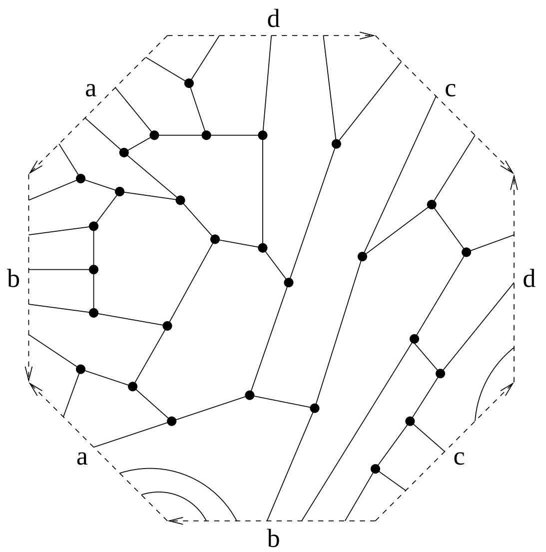

Figure 1.8: A cubic map of genus 2

Another application of *CGF* is given by the famous 5–flow conjecture which has been posed by W.T. Tutte in 1954. This conjecture states that every 2–connected graph has a *5–flow*. For a detailed introduction into the concept of flows, see e.g. [1] (Chapter 6) and [22]. Flows have been considered by various mathematicians. In [23], it is proved that every 2–connected graph has an 8–flow, and in [24] it has been proved that every 2–connected graph has a 6–flow. In [25], it is proved that the 5–flow conjecture is true for all 2–connected graphs that are embeddable in the real projective plane, in [26] it is proved that the 5–flow conjecture is true for 2–connected graphs of orientable genus ≤ 2 or nonorientable genus ≤ 4, and in [27] it is proved that the 5–flow conjecture is true for 2–connected graphs of nonorientable genus 5. Moreover, in [26] it is proved that, in order to check the 5–flow conjecture for graphs of orientable genus g, it suffices to check all maps of genus g with at most $28 \cdot (g-1)$ vertices where the degree of every face is at least 7. This class of maps is of course finite for every g, and for $g = 3$ the problem of constructing all corresponding maps seems to be solvable using *CGF*. In a strict sense, only *minimal* maps of genus 3 should be constructed and tested. However, if a non–minimal map is constructed, then the orientable genus of the underlying graph is smaller than 3 and the test

[2] Later in this text, a map with this property will be called *polyhedral*.

for a 5–flow will always succeed. Since we are interested in counterexamples for the conjecture, non–minimal maps can be tolerated during the tests.

A further application is given by a conjecture of D. Barnette [28] which states that every simple cubic 3–connected bipartite planar graph (C3CBP–graph) is **Hamiltonian**, i.e. it contains a Hamiltonian cycle. A **Hamiltonian cycle** in a graph $G = (V, H, E, T)$ is a $|V|$–cycle (so that each vertex in G is incident with an edge of the cycle, or in other words: each vertex is passed exactly once). G is **bipartite** if V can be partitioned into two disjoint sets V_1 and V_2 such that no two vertices in V_1 are adjacent to each other and no two vertices in V_2 are adjacent to each other. Barnette's conjecture is true for some restricted subclasses of all C3CBP–graphs. See e.g. [29] and [30]. In [31], it is shown that Barnette's conjecture is true for all C3CBP–graphs of order at most 64. On the other hand, for the class of all simple cubic 3–connected bipartite graphs not being planar, a counterexample to the conjecture is constructed in [32]. For the class of simple cubic bipartite planar graphs being exactly 2–connected, the smallest counterexample to the conjecture is given in [33]. Moreover, in [34] it is shown that the conjecture is not true for some simple cubic 3–connected planar graphs that are not bipartite. Hence the properties "bipartite", "planar", and "3–connected" are necessary for the conjecture.

In addition to the results given in [29] and [30], further investigation has been done on simple cubic 3–connected planar maps where the degree of each face is either m or n for two integers m and n. Let $G(m, n)$ denote the class of all those maps. In [35], a non–Hamiltonian member of $G(5, 8)$ is given. In [36], it is shown that $G(5, k)$ contains non–Hamiltonian members for every $k \geq 11$. In [37], it is shown that $G(4, k)$ contains non–Hamiltonian members for all odd $k \geq 9$.

Using *CGF*, for given n and m, all maps in $G(m, n)$ with a limited order can be constructed, and then these maps can be tested for Hamiltonicity in order to find smallest non–Hamiltonian members. For example, $G(4, 7)$ contains a non–Hamiltonian map of order 98 ([38]). This is the smallest known non–Hamiltonian map in $G(4, 7)$. In order to find out if this is the smallest non–Hamiltonian map in $G(4, 7)$ at all, all maps in $G(4, 7)$ with up to 92 vertices must be constructed and tested (due to formula 1.3 on page 9, the set $G(4, 7)$ does not contain maps of order 94 or 96). This can be done using *CGF*. Similar results for $G(3, 7)$ and $G(5, 7)$ can be found in [39] and [40].

A novel concept in graph theory which is investigated by M. Deza (Paris) is the concept of **face–regularity** [41]. A map M is *face–regular* if and only if every face in M of a given degree i has the same number of neighbours of degree i' for every i'. This condition is quite restrictive and already implies finiteness in case of **polyhedra** (3–connected planar maps) where the maximal degree of a face is bounded [41]. The concept can be applied also to toroidal maps. Here two cases are of special interest:

(1) cubic toroidal maps where only triangles and heptagons are permitted
(2) cubic toroidal maps where only pentagons and heptagons are permitted

In the first case, if every heptagon is a neighbour of exactly one triangle, then the map corresponds to a **hexagonal lattice**, i.e. a simple cubic toroidal map where every face is a hexagon. One gets the corresponding hexagonal lattice by contracting all edges that are incident with a triangle. *CGF* as well as *togen7* can be used to construct cubic toroidal maps with the above permitted face degrees.

A further application is given by *tilings* of the sphere. *CGF* can be used to construct *flag graphs* or *Delaney graphs* which are used to represent tilings of the sphere. See [42] and [43] for a detailed introduction into tilings, flag graphs, and Delaney graphs. Loopless Delaney graphs are cubic. The corresponding Delaney graphs of some tilings can be represented by cubic planar bipartite 2–connected maps where also digons are permitted. In particular, a fixed set of faces is permitted, and there are further restrictions to the incidences of the faces. These restrictions can be included into the algorithm presented in part II of this text. In Section 8.5 and Section 10.2, the details will be discussed.

1.9 Trees

Trees are graph theoretical objects. However, throughout this text, they will be used basically only for visualization. We will not construct complete sets of trees with given properties, we will not search for spanning trees inside the constructed maps etc. In order to emphasize this fact, we will use a different terminology for trees than for arbitrary graphs: A **tree** $F = (G, D)$ is a pair where $G = (V, H, E, T)$ is a connected graph without cycles and $D : E \to H$ is a function with $D(e) \in e$ for every $e \in E$ where D satisfies the condition listed below. The vertices of F are called **nodes**. The edges of F are called **branches**. $D(e)$ is called the **direction** of e. The source of $D(e)$ is called the **source** of e, and the target of $D(e)$ is called the **target** of e. D must be defined such that each node except one node r is the target of exactly one branch, and the node r is the target of no branch. r is called the **tree–source**[3] of F.

In the figures, we will always draw a tree in the following way: The source of every branch is always drawn below its target. Thus, the root of the tree is always at the bottom of the figure and we do not need to indicate the directions of the branches by arrows.

A node that is the source of no branch is called a **leaf**. A **forest** $F = (G, D)$ is a pair where $G = (V, H, E, T)$ is a graph and for every connected component G' of G it holds that $(G', D|_{G'})$ is a tree. For an example of a drawn forest, see Figure 4.12 on page 53.

A **tree–path** p of length l in a tree F is a sequence $p = \langle b_1, \dots, b_l\rangle$ where $b_1, \dots, b_l$ are branches, and for every $i \in \{2, \dots, l\}$ it holds that the source of b_i is the target of b_{i-1}. The source of b_1 is called the **start node** of p, the target of b_l is called the **end node**, and the branches $b_1, \dots, b_l$ are called the **elements** of p. The i**–th tree–level** of a tree is the set of all nodes v for which there exists a tree–path p of length i such that the tree–source r is the start node of p and v is the end node of p. Since a tree does not contain cycles, for every $v \in V - \{r\}$ it holds that v and r are connected by exactly one tree–path. The **depth** of F is the length of the longest tree–path in F.

Let v be a node in F. Then $(G', D|_{G'})$ with $G' = (V', H', E', T|_{H'})$ is the **subtree of** F **with tree–source** v if and only if

- V' contains exactly those nodes that are incident (in F) with a tree–path whose start vertex is v,
- E' contains exactly those branches that are elements (in F) of a tree–path whose start vertex is v, and
- H' contains exactly those semi–edges that are elements of a branch $e \in E'$.

[3] A more customary term is "root", but later in this text, we will use the term "root" for something else.

Part I

An inductive definition and the construction of cubic toroidal maps

Chapter 2

Introduction

In this part, we will give an inductive definition of *simple* cubic toroidal maps. Furthermore, we will describe how this inductive definition was used to develop the computer program *togen* which constructs all cubic toroidal maps up to a given vertex number. We denote the class of all[1] simple cubic toroidal maps by $\mathcal{T}$ and the class of all simple cubic toroidal maps with up to n vertices by $\mathcal{T}_n$.

2.1 Directions and sides

In Section 1.1, four basic operations which transform maps into different maps have been introduced. In order to give an inductive definition of $\mathcal{T}$, we will use more complex operations.

2.1.1 Directions

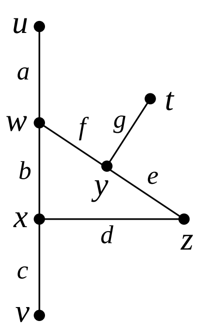

Figure 2.1: A map M'.

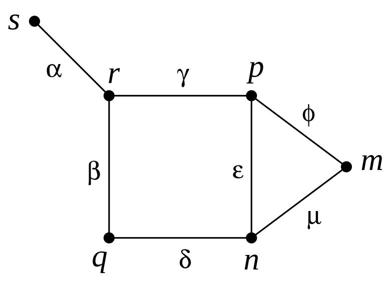

Figure 2.2: A map M.

Suppose we want to substitute an edge $e = \{h_1, h_2\}$ in a cubic map $M = (G, \Pi)$ with $G = (V, H, E, T)$ by a submap $M' = (G', \Pi')$ with $G' = (V', H', E', T')$. Such a substitution can

[1]As always, we consider only pairwise non–isomorphic maps so that $\mathcal{T}$ and $\mathcal{T}_n$ contain exactly one element of each isomorphism class of maps having the desired properties.

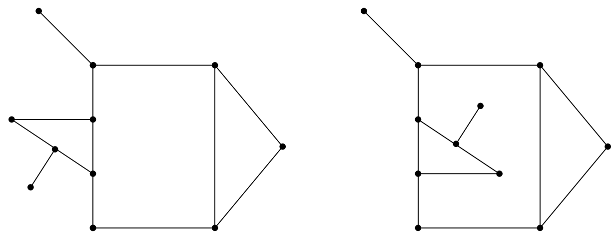

Figure 2.3

be performed as follows: In M', two semi–edges k_1 and k_2 are determined that substitute the semi–edges h_1 and h_2, respectively. So in the resulting map $\hat{M} = (\hat{G}, \hat{\Pi})$ with $\hat{G} = (\hat{V}, \hat{H}, \hat{E}, \hat{T})$, we have $\hat{V} = (V \cup V') - \{v_1, v_2\}$ where v_1 and v_2 are the sources of k_1 and k_2, respectively, $\hat{H} = (H \cup H') - \{h_1, h_2\}$, $\hat{E} = (E \cup E') - \{e\}$, and $\hat{\Pi} : \hat{H} \to \hat{H}$ as follows (note that since M is cubic, we have $\Pi(h) \neq h$ for all $h \in H$):

$$\hat{\Pi}(h) = \begin{cases} \Pi(h_1) & \text{if } h = k_1, \\ \Pi(h_2) & \text{if } h = k_2, \\ \Pi'(k_1) & \text{if } h = \overline{\Pi}(h_1), \\ \Pi'(k_2) & \text{if } h = \overline{\Pi}(h_2), \\ \Pi(h) & \text{if } h \in H - \{h_1, h_2, \overline{\Pi}(h_1), \overline{\Pi}(h_2)\}, \\ \Pi'(h) & \text{else (i.e. } h \in H' - \{k_1, k_2\}). \end{cases}$$

The sources of k_1 and k_2 are substituted by the sources of h_1 and h_2. Thus T is implied by E and $\hat{\pi}$.

As an example, consider $M' = (G', \Pi')$ with $G' = (V', H', E', T')$ where $V' = \{t, u, v, w, x, y, z\}$, $H' = \{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2, e_1, e_2, f_1, f_2\}$, $E' = \{a, b, c, d, e, f\}$ with $a = \{a_1, a_2\}$, $b = \{b_1, b_2\}$, $\ldots$, $f = \{f_1, f_2\}$, and Π' is given by $r_{\pi'}(t) = \langle g_1 \rangle$, $r_{\pi'}(u) = \langle a_1 \rangle$, $r_{\pi'}(v) = \langle c_1 \rangle$, $r_{\pi'}(w) = \langle a_2, f_1, b_1 \rangle$, $r_{\pi'}(x) = \langle b_2, d_1, c_2 \rangle$, $r_{\pi'}(y) = \langle g_2, e_1, f_2 \rangle$, $r_{\pi'}(z) = \langle d_2, e_2 \rangle$. Figure 2.1 shows M' where the semi–edges are not given explicitly in order not to overload the figure. We determine that a_1 and c_1 play the roles of k_1 and k_2 in the above description.

Let $M = (G, \Pi)$ with $G = (V, H, E, T)$ be the map shown in Figure 2.2. M is given by $V = \{m, n, p, q, r, s\}$, $H = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2, \delta_1, \delta_2, \varepsilon_1, \varepsilon_2, \phi_1, \phi_2, \mu_1, \mu_2\}$, $E = \{\alpha, \beta, \gamma, \delta, \varepsilon, \phi, \mu\}$ with $\alpha = \{\alpha_1, \alpha_2\}$, $\ldots$, $\mu = \{\mu_1, \mu_2\}$, and Π as follows: $r_\pi(s) = \langle \alpha_1 \rangle$, $r_\pi(r) = \langle \alpha_2, \gamma_1, \beta_1 \rangle$, $r_\pi(q) = \langle \beta_2, \delta_1 \rangle$, $r_\pi(p) = \langle \gamma_2, \phi_1, \varepsilon_1 \rangle$, $r_\pi(n) = \langle \delta_2, \varepsilon_2, \mu_2 \rangle$, $r_\pi(m) = \langle \phi_2, \mu_2 \rangle$.

Suppose we want to substitute edge $\beta = \{\beta_1, \beta_2\}$ by M'. For this, there are two possibilities depending on whether a_1 replaces β_1 or β_2. The resulting maps are shown in Figure 2.3. We will omit the precise descriptions of the resulting maps because they are rather tedious and can easily be derived from the figures or from the above description of the substitution. In order to have a unique substitution, we must fix whether a_1 is to replace β_1 or β_2. This can be done by marking (emphasizing) one of the semi–edges β_1 or β_2. The emphasis of a semi–edge of an edge e is called a **direction** of e. In a figure, the direction of an edge is indicated by drawing an arrow from the source to the target of the marked semi–edge.[2] In M', we mark a semi–edge, too. This is done in order to help formulating a shortcut of the substitution rule.

[2]This description is sufficient only for *simple* maps. If the source and the target are identical, i.e. the directed edge is a loop, then we must be more precise: By tracing the edge in the drawing in the direction indicated by the arrow, one first traces the arc that represents the marked semi–edge and then the arc that represents the other semi–edge.

For example, if we mark a_1, then a shortcut of the substitution rule could be formulated like this: "Replace the marked semi–edge in M by the marked semi–edge in M'." Of course, we must keep the complete substitution rule in mind, i.e. if a_1 is marked, then not only the marked semi–edge in M is replaced by the marked semi–edge (a_1) in M', but also the inverse of the marked semi–edge in M is replaced by c_1 (and not by any other semi–edge).

In Chapter 3, the substitution rules are not given explicitly. Only figures are provided. However, the figures are self–explaining so that the corresponding substitution rules can easily be derived.

2.1.2 Sides

Suppose we want to substitute an edge in a map M by the *mirror image* $\overline{M'}$ of M'. We could consider this operation to be a completely different operation from the one before, but then the algorithm that detects isomorphic maps (see Chapter 4) would not work. Instead, we comprehend both possibilities in one operation. For example, both maps in Figure 2.4 can be obtained by the above operation if β_1 is the marked semi–edge in M. In order to distinguish the two possibilities, a **side** is assigned to the directed edge in M (more precisely: to the marked semi–edge in the directed edge). A side s is given by $s \in \{\texttt{left}, \texttt{right}\}$ (in a cubic map, it could also be given by $s \in \{\overline{\Pi}(h), \Pi(h)\}$ where h is the marked semi–edge). Now we must add a condition to the substitution rule that tells us how to handle the side. For example: "If the side is `left`, then M' is inserted, and if the side is `right`, then $\overline{M'}$ is inserted". In this case, the right map in Figure 2.4 is constructed if the side is `left` and the left map is constructed if the side if `right`.

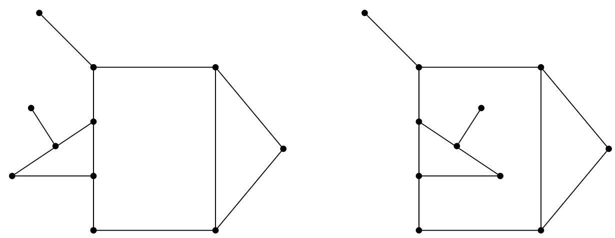

Figure 2.4

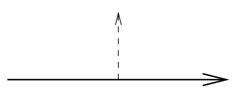

Figure 2.5

An edge e to which a direction and a side is assigned is called a **rooted edge** (or just a **root**). A root is denoted by $r(e) = (e, h, s_e)$ where e is the underlying edge, h is the marked semi–edge (which must be an element of e) indicating the direction, and $s_e \in \{\texttt{left}, \texttt{right}\}$ is the side. In a figure, the side of a rooted edge is indicated by a dashed arrow perpendicular to the edge. Figure 2.5 shows a rooted edge whose side is `left`. A **rooted map** is a pair $(M, r(e))$ where M is a map and e is an edge of M (i.e. exactly one edge is rooted).

In our example, by rooting edge β and applying the same operation with respect to the root, we obtain up to four pairwise non–isomorphic maps. The maps that are related to the above example are shown in Figure 2.6.

In principle, a side can be assigned to every semi–edge in a map. Let s be the source and t be the target of a semi–edge h. Then the **left face** of h is the face containing the sequence $\langle \ldots, h, \overline{h}, \ldots \rangle$ (for distinctness more extensively: $\langle \ldots, \overline{\pi_s}(h), s, h, \overline{h}, t, \pi_t(\overline{h}), \ldots \rangle$) and the **right face** of h is the face containing the sequence $\langle \ldots, \overline{h}, h, \ldots \rangle$ (more extensively:

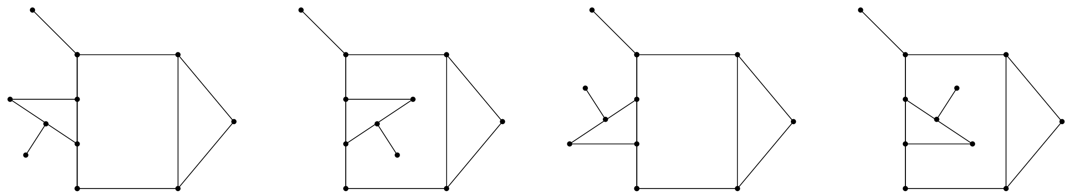

Figure 2.6

$\langle \ldots, \overline{\pi_t}(\overline{h}), t, \overline{h}, h, s, \pi_s(h), \ldots \rangle$). Both faces might of course be identical. In this case, $e = \{\overline{h}, h\}$ is a face–bridge. Every face $F = \langle v_0, h_0, h'_0, \ldots, v_{r-1}, h_{r-1}, h'_{r-1} \rangle$ is the left face of $h_0, h_1, \ldots, h_{r-1}$ and the right face of $h'_0, h'_1, \ldots, h'_{r-1}$.

In a rooted map, the face on the indicated side of the root is called the **root face**. The face on the other side is called the **opposite face**. If the rooted edge is a face–bridge, then both faces are identical.

A map isomorphism can be extended to act on roots as follows: An orientation *preserving* map isomorphism maps roots whose sides are `right` onto roots whose sides are `right` and roots whose sides are `left` onto roots whose sides are `left`. An orientation *reversing* map isomorphism maps roots whose sides are `right` onto roots whose sides are `left` and vice versa. Two roots are **isomorphic in** M if and only if both roots are in the same *orbit* of the automorphism group of a corresponding map automorphism. This extension is consistent with the way the above operation acts depending on the direction and the side of a root, i.e. if a submap is inserted into two copies of a map with respect to two different roots, then isomorphic rooted maps are obtained if and only if the roots are isomorphic (in the original map).

From now on, in order to omit the notation of semi–edges completely, we will introduce a substitute ρ for the rotation system π: Instead of writing $\pi_v(h_1) = h_2$, we will write $\rho_v(e_1) = e_2$ where e_1 and e_2 are those edges with $h_1 \in e_1$ and $h_2 \in e_2$. Since the considered maps are simple, the source of a semi–edge is never equal to its target, and hence the original rotation system π can uniquely be derived from ρ (if $e = \{h, h'\}$ and v is incident with e, then v is incident either with h or with h'). Thus we do not obtain any loss of information. So let $V = \{v_1, v_2, \ldots, v_n\}$ be the vertex set of a simple map M. Then the **edge rotation system** ρ of M is a collection $\rho = \{\rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_n}\}$ where $\rho_v(e_1) = e_2$ if and only if $\pi_v(h_1) = h_2$ for some $h_1 \in e_1$ and $h_2 \in e_2$. In particular, π implies ρ and vice versa. As usual, $\overline{\rho_v}$ denotes the inversion of ρ_v. Let e be an edge incident with v. Then the **edge rotation sequence** of v (with respect to ρ) is given by $r_\rho(v) = \langle e, \rho_v(e), \rho_v^2(e), \ldots, \rho_v^{\mathrm{val}(v)-1}(e) \rangle$ where ρ_v^j denotes the j–th iterated application of ρ_v. Depending on the choice of e, the sequence is cyclically shifted. The resulting sequences are not distinguished.

With ρ, the Jordan curve theorem of Section 1.6 can be formulated as follows:

Theorem 2.1 (Jordan curve theorem) *Let $M = (G, \Pi)$ with $G = (V, H, E, T)$ be a simple cubic planar map. Let $c = \langle e_0, \ldots, e_n \rangle$ be a cycle in M. Let v_i, $i \in \{0, \ldots, n\}$, be the vertex which is incident with e_i and e_{i-1} (indices modulo $n+1$). Let $e, e' \in E$ be edges not incident with c but $\rho_{v_i}(e_i) = e$ and $\overline{\rho_{v_j}}(e_j) = e'$ for some $i, j \in \{0, \ldots, n\}$. Then for every path p with start edge e and end edge e', it holds that p is incident with c.*

The vertices $v_0, \ldots, v_n$ are necessary in order to secure that e_i and e_j in Theorem 2.1 correspond to h_i and h_j (and not to h'_i and h'_j) in Theorem 1.1.

2.2 About faces in toroidal maps

In order to make the arguments in the following chapters easy to follow, we will provide some figures. However, all arguments can be verified purely combinatorially. This will also be done. As already mentioned in Section 1.2, the most transparent way to draw a toroidal map is to draw the map on a rectangle whose opposite sides are identified. If only a part of a map is drawn, then it might not be clear how an edge can be drawn even if both incident vertices are known and lie inside the drawn part. In particular, this is the case if we do not know if this edge must or must not cross two sides of the rectangle. Figure 2.7 shows an example. If both possibilities shall be kept, then this edge is not drawn completely, but two parts which are labelled identically are drawn (see Figure 2.7). If two partly drawn edges are not labelled identically or not labelled at all, then the corresponding edges might be identical or not. On the other hand, drawn *vertices* are always pairwise distinct. For brevity, we will refer to partly drawn edges as **dangling edges** from now on.

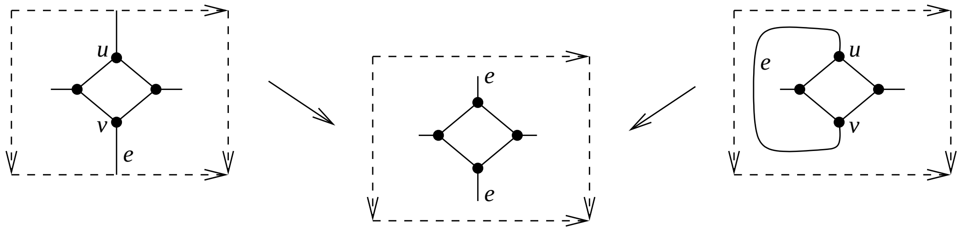

Figure 2.7: Two possibilities how u and v are connected via edge e summarized in one picture

2.3 Faces of degree at most 7

The following lemma states that an edge which is incident with a face of degree at most 7 is no face–bridge.

Lemma 2.2 *If an edge e in a simple cubic map M is incident with only one face F, then deg$(F) \geq 8$.*

Proof: In Figure 2.8, the situation is illustrated. The adjacent vertices u and v of e are each incident with two more edges a, b and c, d, respectively. a, b, c, and d are pairwise distinct, otherwise we would have loops or double edges. For the same reason, u is the only common incident vertex of a and b, and v is the only common incident vertex of c and d. Since e is only incident with F, e must be counted twice in order to compute deg(F). a, b, c, and d are counted at least once. So deg$(F) \geq 6$ is clear. Now there are 3 cases:

- If a, b, c, and d have pairwise no common incident vertices except u and v, then there must be at least two more edges that are incident with F and hence deg$(F) \geq 8$.
- If a and c are adjacent, then the 3–cycle $\langle a, e, c\rangle$ does not correspond to a triangle because deg$(F) \geq 6$. Let w be the vertex which is incident with both a and c. It holds $\rho_v(e) = c$, $\rho_u(a) = e$ but $\rho_w(c) \neq a$, and hence the situation is as shown in Figure 2.9. Here $h \neq b$ and $h \neq d$ because otherwise M would contain a double edge. Moreover, h is incident only with F and must therefore be counted twice: F contains the sequence $\langle \ldots, h, a, e, c, h, \ldots, d, e, b, \ldots, \rangle$. So deg$(F) \geq 8$. The same argumentation can be applied if b and d instead of a and c are adjacent.

- If a and d are adjacent, then the situation is as shown in Figure 2.10. Here $g \neq c$ and $g \neq b$ as above because otherwise M would contain a double edge. Either a and b or c and d must be counted twice depending on whether $\rho_w(d) = g$ or $\rho_w(g) = d$: If $\rho_w(d) = g$ (as in Figure 2.10), then F contains the sequence $\langle \ldots, b, a, d, e, b, \ldots, g, a, e, c \ldots \rangle$, and if $\rho_w(g) = d$, then F contains the sequence $\langle \ldots, g, d, e, b, \ldots, c, d, a, e, c, \ldots \rangle$. Thus even $\deg(F) \geq 9$. The same argumentation can be applied if b and c are adjacent. ■

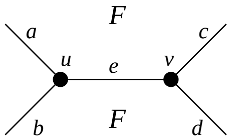

Figure 2.8

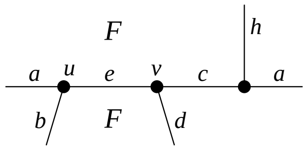

Figure 2.9

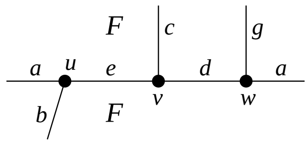

Figure 2.10

As a consequence of this lemma, a face F of degree $d \leq 7$ corresponds to a d–cycle being incident with exactly those edges that are incident with F (none of these edges appears twice in the face sequence), and hence a face of degree $d \leq 7$ is incident with exactly d vertices. Figure 2.11 shows a map where the degree of the outermost face F is 8 but F is incident with only 6 vertices. Moreover, there is no cycle being incident with exactly those edges that are elements of F.

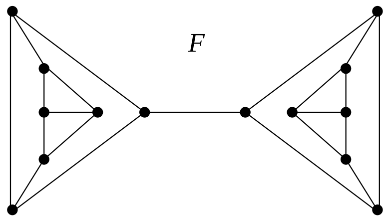

Figure 2.11: The outermost face F does not correspond to a cycle

Chapter 3

The inductive definition of $\mathcal{T}$

In order to give the inductive definition of $\mathcal{T}$, we need to provide a set $\mathcal{B} \subseteq \mathcal{T}$ of so–called **basic maps** and a set $\mathcal{R}$ of **generating rules**. We will show that every map $M \in \mathcal{T}$ can be constructed by applying a finite number of generating rules to a basic map[1] where every intermediate map will be in $\mathcal{T}$.

3.1 Basic maps

We need the following basic maps:

- the two toroidal embeddings K_4' and K_4'' of K_4 shown in Figure 3.1
- 6 more maps with up to 14 vertices shown in Figure 3.2

The reason why we need the maps in Figure 3.2 as basic maps is given by the generating rules. This will be shown in Section 3.3.1.

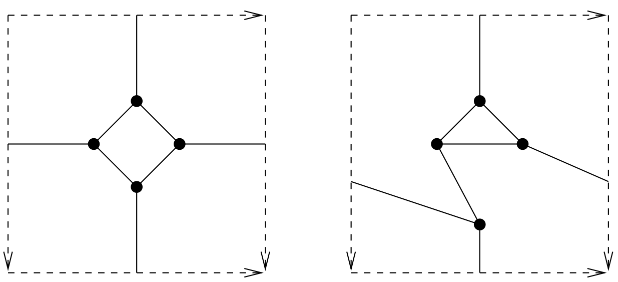

Figure 3.1: the two basic maps with 4 vertices K_4' (left map) and K_4'' (right map)

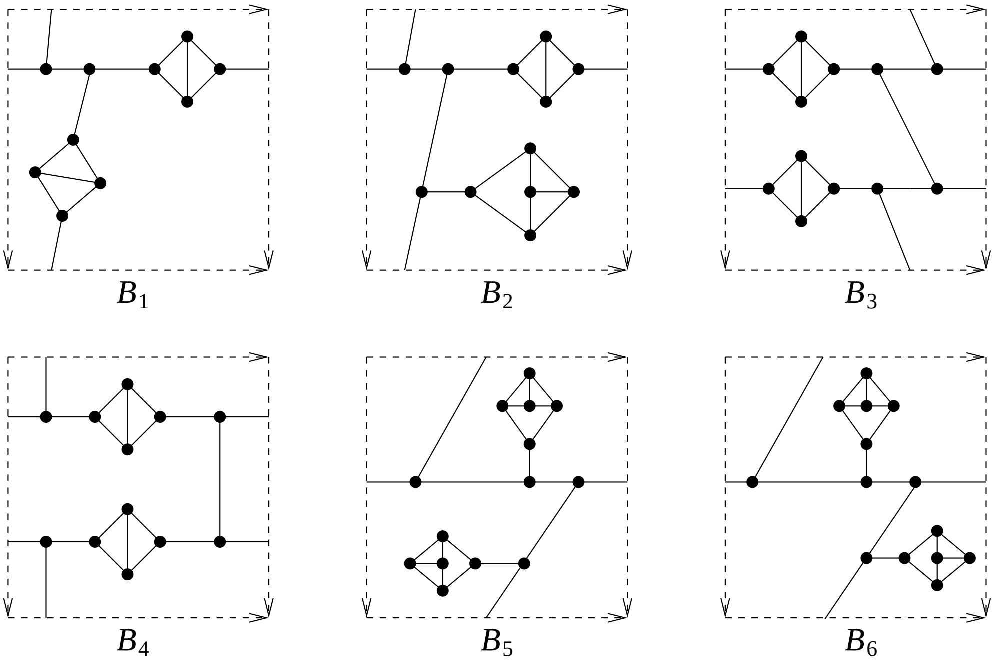

Figure 3.2: six more basic maps $B_1 - B_6$

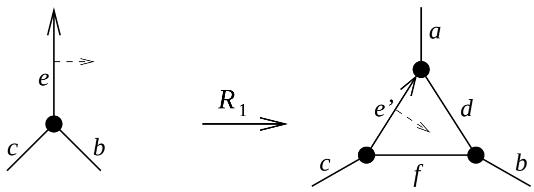

Figure 3.3

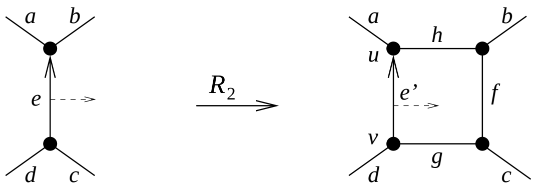

Figure 3.4

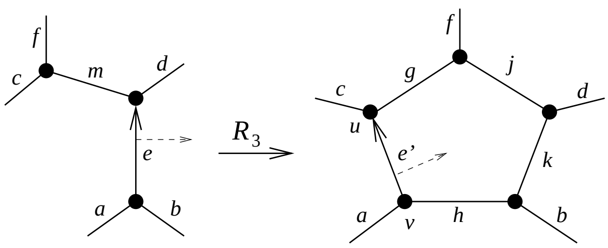

Figure 3.5

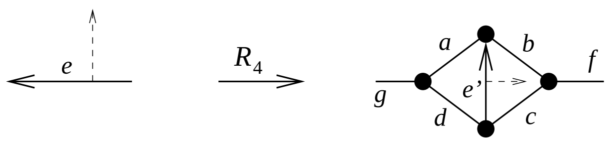

Figure 3.6

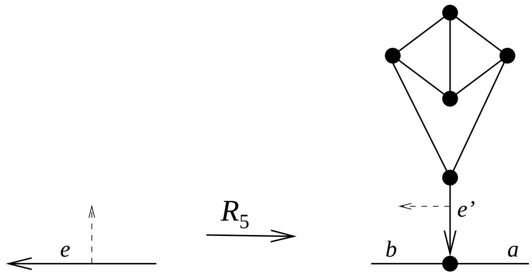

Figure 3.7

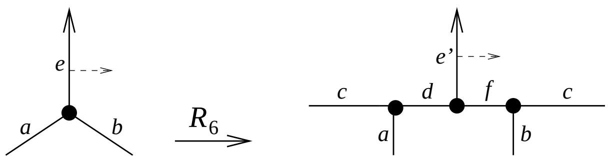

Figure 3.8

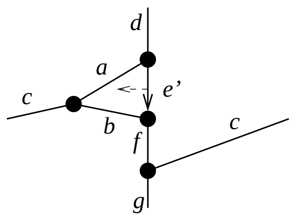

Figure 3.9

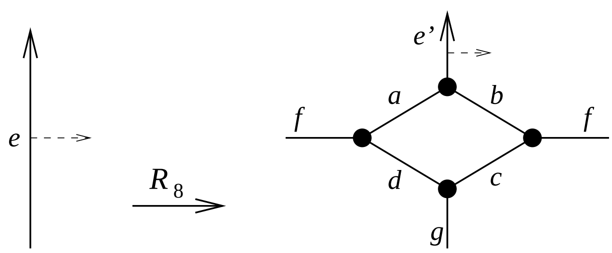

Figure 3.10

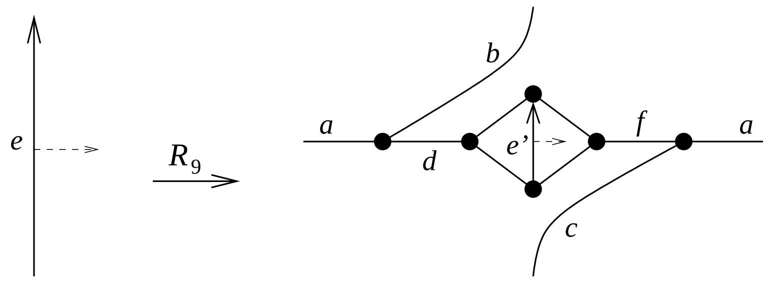

Figure 3.11

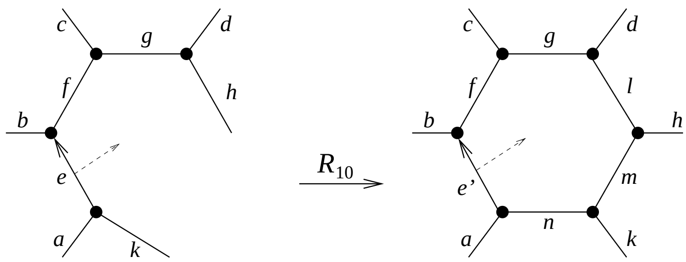

Figure 3.12

3.2 Generating rules

We use the generating rules shown in Figures 3.3-3.12. Each rule inserts a new submap into an existing map. For convenience, we will number the submaps in the same way the generating rules are numbered. Thus the application of rule R_i inserts submap S_i into a map. Here, all drawn vertices and completely drawn edges (on the right side of the figure) are contained in the submap, while a dangling edge is contained in the submap exactly if there exists another dangling edge which is labelled identically.

The figures illustrate what could also be denoted purely combinatorially. For example, the incidence structure of S_2 could be denoted as follows: $r_\rho(v) = \langle d, e', g\rangle$, $r_\rho(u) = \langle a, h, e'\rangle$ etc.

Some of the generating rules are related to each other in the sense that if we would consider graphs instead of maps, then these rules could not be distinguished. This holds for the rules R_1 and R_6 as well as for R_4, R_7, and R_8: The edges a, b, c, d, e', f in S_6 (Figure 3.8) correspond to the edges c, b, f, e', a, d in S_1 (Figure 3.3), and the edges a, b, c, d, e', f, g in S_4 (Figure 3.6) correspond to the edges c, a, e', f, b, d, g in S_7 (Figure 3.9) as well as to the edges a, d, c, b, f, g, e' in S_8 (Figure 3.10).

For every generating rule, there is one **rooted reference edge** before and one rooted reference edge after the application of the rule. In Figures 3.3-3.12, the rooted reference edge before the application of the rule is always $r(e)$, and after the application of the rule, the rooted reference edge is always $r(e')$. The underlying edge of a rooted reference edge is called the **reference edge**. The knowledge about the reference edge is not sufficient for a unique application of a generating rule. For that, the *rooted* reference edge must be given.

[1]To be precise: The first generating rule is applied to the basic map, and every other generating rule is applied to the map that has been obtained in the previous step.

There are some restrictions to the application of the generating rules:

- It is obvious that by applying R_3, we obtain a double edge if $c = a$ in Figure 3.5. Hence R_3 may only be applied if $c \neq a$ holds.
- For the same reason, R_{10} must not be applied if $k = h$.
- If we want to apply one of the generating rules $R_6 - R_9$, then the corresponding edge e must be a face–bridge. Otherwise the resulting map would have genus 2. This can be shown in a similar way as in Section 1.5 using Euler's formula. As an example, we will do this for R_8: Let an edge e in a map $M \in \mathcal{T}$ be incident with two different faces $D = \langle d_1, \ldots, d_r \rangle$ on the left side and $E = \langle e_1, \ldots, e_s \rangle$ on the right side. Here $r = \deg(D)$ and $s = \deg(E)$. Let w.l.o.g. $e = d_1$ and $e = e_1$ (see Figure 3.13). By inserting S_8 using R_8 applied to $r(e)$, we obtain a map M' with one new face F (the square $F = \langle d, c, b, a \rangle$ in Figure 3.10). Furthermore, the faces D and E in M are replaced by one face $C = \langle e', b, f, a, e', d_2, \ldots, d_r, g, d, f, c, g, e_2, \ldots, e_s \rangle$ in M' (see Figure 3.13). No further faces are modified. Thus $f_{M'} = f_M$, $e_{M'} = e_M + 6$, and $n_{M'} = n_M + 4$. Hence with Euler's formula on page 6, we obtain $2g_{M'} = 2 + e_{M'} - n_{M'} - f_{M'} = 2 + e_M + 6 - n_M - 4 - f_M = 4 + e_M - n_M - f_M = 2g_M + 2$, and thus $g_{M'} = g_M + 1 = 2$.

  On the other hand, if e is incident with only one face $E = \langle e_1, \ldots, e_n \rangle$ (with $e = e_1$ and $e_1 = e_i$ for some $i \in \{4, \ldots, n-2\}$, see Figure 3.14), then by inserting S_8 into M, we obtain a map M' where F as before is a new face and E is replaced by the faces $C = \langle g, e_2, \ldots, e_{i-1}, g, d, f, c \rangle$ and $D = \langle e', e_{i+1}, \ldots, e_n, e', b, f, a \rangle$ (see Figure 3.14) so that $f_{M'} = f_M + 2$, $e_{M'} = e_M + 6$, and $n_{M'} = n_M + 4$. Hence with Euler's formula, we obtain $2g_{M'} = 2 + e_{M'} - n_{M'} - f_{M'} = 2 + e_M + 6 - n_M - 4 - f_M - 2 = 2 + e_M - n_M - f_M = 2g_M$, and thus $g_{M'} = g_M$.

  In the same way, it can be verified for R_6, R_7, and R_9 that the genus is preserved if and only if the corresponding edge e is a face–bridge, and it can be verified that every other operation preserves the genus of a map.

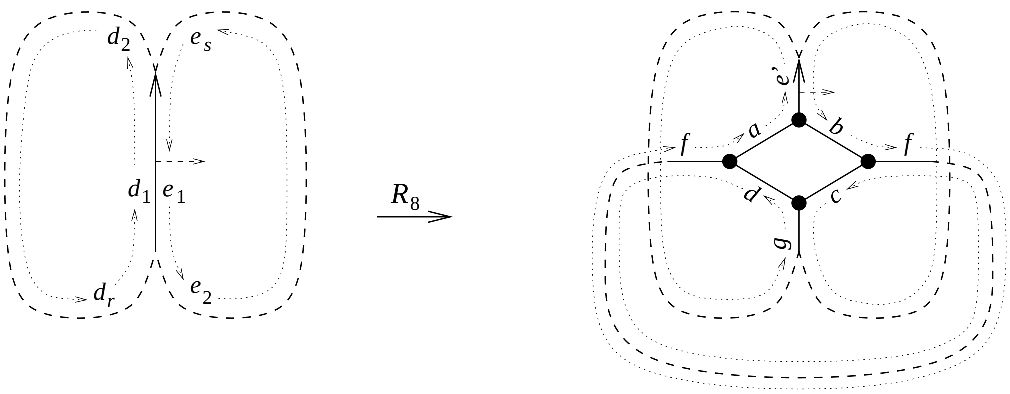

Figure 3.13

3.3 Main theorem

Theorem 3.1 *Let $\mathcal{B} = \{K_4', K_4'', B_1, B_2, B_3, B_4, B_5, B_6\}$. Then the set $\mathcal{M}$ of maps that can be constructed by applying a finite number of generating rules $R_1 - R_{10}$ to an element of $\mathcal{B}$ considering the above restrictions is exactly $\mathcal{T}$.*

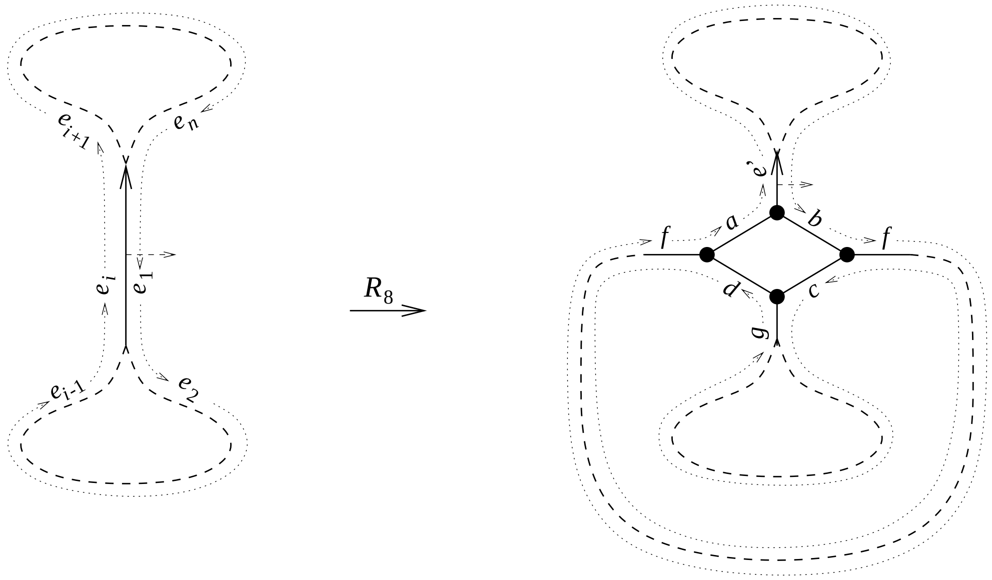

Figure 3.14

3.3.1 Proof: Preparation

It is clear that $\mathcal{M} \subseteq \mathcal{T}$ holds because if the above restrictions are considered, then applying a generating rule to a map $M \in \mathcal{T}$ preserves simplicity, connectivity (the resulting bunch of maps is connected), cubicity, and genus so that the resulting map is in $\mathcal{T}$, too.

Let R_i^- denote the inverse of the generating rule R_i. In order to prove that $\mathcal{T} \subseteq \mathcal{M}$, it suffices to show that every map $M \in \mathcal{T} - \mathcal{B}$ contains a submap which can be **reduced** by applying the inverse of a generating rule. Thus, a map $M' \in \mathcal{T}$ is obtained having a smaller order than M. If M' is not a basic map, then again we apply the inverse of a generating rule to M'. We repeat this procedure until the resulting map $\hat{M}$ is a basic map. By starting with $\hat{M}$ and applying the sequence of original generating rules (not their inverses), we obtain M.

The generating rules $R_1 - R_{10}$ are not redundant, i.e. there exists no R_i which can be substituted by a sequence of other generating rules. For example, if we apply the generating rule R_1 to an arbitrary vertex, then the degree of each of the (at most 3) incident faces will increase (by at least 1) as well as the lengths of all incident cycles. Hence the new triangle will not be incident with any 3–cycle (we call that **isolated triangle**). Thus, the triangle will not be contained in a submap S_4, S_5, S_7, S_8, or S_9. Hence none of the generating rules R_4, R_5, R_7, R_8, and R_9 can be substituted by a sequence of generating rules including R_1.

Contrary, to every submap at most one inverse of a generating rule $R_1 - R_{10}$ can be applied. For example, R_1^- cannot be applied to a triangle which is contained in a submap S_5 because then a double edge would be obtained. The same holds for every other submap. So for convenience, in search of submaps $S_1 - S_{10}$ inside a map M, we will only consider those submaps $S_1 - S_{10}$ that are not contained in bigger submaps $S_1 - S_{10}$. Such submaps will be called **maximal submaps in** M. So maximal submaps might contain other submaps $S_1 - S_{10}$. Explicitly, *an*[2] S_1 might be contained in an S_4, S_5, S_7, S_8, or S_9, an S_2 might be contained in an S_5 or S_8,

[2] For convenience, we omit the more precise formulation "a submap in M isomorphic to S_1".

an S_4 might be contained in an S_5 or S_9, and an S_6 might be a contained in an S_7 or S_8. In these cases, the smaller submaps are not considered. So for example, a "twin triangle" (two triangles incident with a common edge) in M will not be considered as a submap S_4 if both triangles are incident with a square. Instead, the twin triangle and the square are contained in a submap S_5 (which is maximal in M), and thus only the S_5 will be considered. This is done because the intention of searching a submap inside a map is given by reducing this submap, and — as mentioned above — one would obtain loops or double edges anyway if the inverse of a generating rule would be applied to a submap which is a submap of another submap $S_1 - S_{10}$.

But even though we consider only maximal submaps, it is possible that the reduction of a (maximal) submap applying the inverse of the appropriate generating rule would result in a loop, a double edge, or even in a planar map. In this case, the resulting map is not in $\mathcal{T}$ and the submap is called **irreducible**. Otherwise the submap is called **reducible**. Examples for irreducible submaps can be found in Figure 3.2: In $B_1 - B_6$, all (maximal) submaps are irreducible (this is the reason why $B_1 - B_6$ need to be basic maps at all). This can be verified using Table 3.1: Table 3.1 shows all conditions for a submap to be irreducible and thus all restrictions for the application of the inverse of the corresponding generating rule. The left column ("case 1") contains constellations that would result in a loop or a double edge because some edges in a short cycle would be contracted or replaced by a fewer number of edges. The right column ("case 2") contains constellations that would result in a planar map. See Figures 3.3-3.12.

Submap	Case 1	Case 2
S_1	—	—
S_2	a and b are adjacent or c and d are adjacent	—
S_3	a and b are adjacent or c and f are adjacent or $a = d$ or $c = d$ or both $a = f$ and $b = c$	—
S_4	f and g are incident with a pentagon	—
S_5	a and b are elements of a 3–cycle	—
S_6	e' is adjacent to a or to b or a and b are adjcent to each other	c is a face–bridge
S_7	d and g are adjacent to each other or to a common edge	c is a face–bridge
S_8	e' and g are adjacent to each other or to a common edge	f is a face–bridge
S_9	b and c are adjacent to each other or to a common edge except a	a is a face–bridge
S_{10}	k and a are adjacent or h and d are adjacent or $h = c$ or $k = b$ or both $h = a$ and $d = k$	—

Table 3.1: How to detect irreducible submaps

Remarks:

- We assume that in S_4, it holds $f \neq g$ because otherwise we would have a planar embedding of K_4. Moreover, we assume that in S_8, it holds $e' \neq g$, and in S_7, it holds $g \neq d$ because otherwise we would have a K_4' or a K_4'', respectively, and these maps do not contain reducible submaps because they are the smallest possible maps in $\mathcal{T}$.

- If f and g in S_4 are adjacent to each other, then one of the 4–cycles $\langle a, b, f, g\rangle$ or $\langle f, c, d, g\rangle$ *is*[3] a square and the S_4 is not maximal but contained in an S_5. If f and g in S_4 are not adjacent to each other but to a common edge x, then either one of the 5–cycles $\langle a, b, f, x, g\rangle$ or $\langle f, c, d, g, x\rangle$ is a pentagon (which would make the S_4 irreducible as indicated in Table 3.1), or the S_4 would be contained in an S_9. Hence we presume that these cases do not hold.
- In S_7, c is a face–bridge exactly if f is a face–bridge. In S_9, a is a face–bridge exactly if d is a face–bridge and exactly if f is a face–bridge.
- If it holds $a = f$ and $b = c$ in S_3, then we have a 4–cycle $\langle a, g, c, h\rangle$ whose elements g and h would both be contracted by applying ${R_3}^-$ so that we would obtain the double edge $\{a, c\}$. The same effect of contracting 2 edges in a 4–cycle would be given if $a = c$ and $b = f$, but $a = c$ is not possible because this implies a double edge $\{a, e'\}$. For the same reasons, in Figure 3.12, $h = a$ and $d = k$ make the S_{10} irreducible, and $h = k$ (and $d = a$) is not possible. In S_2, a similar constellation is not possible at all: If g and h in Figure 3.4 are elements of a 4–cycle, then $a = c$ and $b = d$ so that we have a K_4'.
- The constellations that would result in a planar map ("case 2") can be verified as usual by using Euler's formula. For example, if f is a face–bridge in S_8 (see Figure 3.10), then by applying ${R_8}^-$, the face F in M being incident with f is replaced by two faces D and E in the resulting map M'. Hence $f_{M'} = f_M$, and with $n_{M'} = n_M - 4$ and $e_{M'} = e_M - 6$, Euler's formula implies $g_{M'} = g_M - 1$.

Now we show that every map $M \in \mathcal{T} - \mathcal{B}$ contains at least one reducible submap $S_1 - S_{10}$.

3.3.2 Step 0

First, we show that every *hexagonal lattice* contains a reducible submap S_{10}. A hexagonal lattice H does not contain 3–cycles: Triangles are not permitted per definition, and 3–cycles that are no triangles (as in S_6, S_7, and S_8) are not possible because otherwise H would contain a face–bridge, but then due to Lemma 2.2, H must contain a face of degree at least 8 which is impossible. Hence the conditions given for S_{10} in Table 3.1 cannot hold. In particular, $d = k$ cannot hold for any S_{10} in H because otherwise $\langle d, l, m\rangle$ would be a 3–cycle and h would be the corresponding face–bridge. Hence ${R_{10}}^-$ may be applied to every hexagonal lattice where an arbitrary rooted edge may be used as the rooted reference edge. So every hexagonal lattice contains a reducible submap S_{10}.

Let $\mathcal{H}$ be the set of all hexagonal lattices. It remains to be shown that every map $M \in \mathcal{T}-(\mathcal{H}\cup\mathcal{B})$ contains a reducible submap $S_1 - S_{10}$. We will show that M even contains a reducible submap $S_1 - S_9$. Formula 1.3 on page 9 implies that each map $M \in \mathcal{T} - (\mathcal{H} \cup \mathcal{B}) \subset \mathcal{T} - \mathcal{H}$ contains at least one face whose degree is at most 5.

3.3.3 Step 1

First, we prove that a map $M \in \mathcal{T}$ containing a pentagon also contains a reducible submap $S_1 - S_9$. In most cases, this submap will be the S_3 containing the pentagon. We will consider those cases first. Figure 3.15 shows the submap S_3 in M. Assume that none of the following 5 cases holds:

[3] To be precise: The sequence $\langle a, b, f, g\rangle$ or $\langle f, c, d, g\rangle$ is also the sequence of a square.

- $\alpha = \gamma$ and $\beta = \varepsilon$
- $\beta = \delta$ and $\gamma = \alpha$
- $\gamma = \varepsilon$ and $\delta = \beta$
- $\delta = \alpha$ and $\varepsilon = \gamma$
- $\varepsilon = \beta$ and $\alpha = \delta$

We will assign the edges of the pentagon in Figure 3.15 to the edges of Figure 3.5 such that the conditions for S_3 in Table 3.1 are not satisfied and ${R_3}^-$ may be applied. The above 5 restrictions imply that there is no assignment such that it holds both $a = f$ and $b = c$ in Figure 3.5. Hence we need to care only about the 4 remaining conditions in Table 3.1. Let $a \mid b$ denote that the edges a and b are adjacent, and let $a \nmid b$ denote that a and b are not adjacent.

a) Let $\beta \neq \delta$, $\alpha \neq \delta$, $\alpha \nmid \varepsilon$, $\beta \nmid \gamma$. Then we assign $\phi \to e'$ and $\kappa \to g$ so that $\alpha \to a$, $\beta \to c$, $\gamma \to f$, $\delta \to d$, and $\varepsilon \to b$, and thus no condition for S_3 in Table 3.1 is satisfied.

b) Let $\beta = \delta$. Then $\alpha \neq \gamma$ (see above). Moreover, $\alpha \neq \delta$, $\gamma \nmid \beta$, and $\delta \nmid \varepsilon$. We assign $\lambda \to e'$ and $\mu \to g$ so that $\alpha \to d$, $\beta \to b$, $\gamma \to a$, $\delta \to c$, and $\varepsilon \to f$.

c) Let $\alpha = \delta$. Then $\varepsilon \neq \gamma$ (see above). Moreover, $\alpha \neq \gamma$, $\delta \nmid \varepsilon$, and $\alpha \nmid \beta$. We assign $\nu \to e'$ and $\phi \to g$ so that $\alpha \to c$, $\beta \to f$, $\gamma \to d$, $\delta \to b$, and $\varepsilon \to a$.

d) Let $\alpha \mid \varepsilon$.

   d$_1$) Let $\beta \mid \alpha$. Then $\beta \mid \varepsilon$ and hence $\beta \nmid \gamma$, $\delta \nmid \varepsilon$, $\alpha \neq \gamma$, and $\alpha \neq \delta$. We assign $\lambda \to e'$ and $\mu \to g$ as in case b$_2$.

   d$_2$) Let $\delta \mid \varepsilon$. Then $\delta \mid \alpha$ and hence $\delta \nmid \gamma$, $\beta \nmid \alpha$, $\varepsilon \neq \gamma$, and $\varepsilon \neq \beta$. We assign $\kappa \to e'$ and $\lambda \to g$ as in case b$_1$.

   d$_3$) Let $\beta \nmid \alpha$ and $\delta \nmid \varepsilon$. $\alpha \mid \varepsilon$ implies that $\alpha \neq \gamma$ and $\varepsilon \neq \gamma$. Then we assign $\nu \to e'$ and $\phi \to g$ as in case c$_2$.

e) Let $\beta \mid \gamma$.

   e$_1$) Let $\delta \mid \gamma$. Then $\delta \mid \beta$ and hence $\alpha \nmid \beta$, $\delta \nmid \varepsilon$, $\gamma \neq \alpha$, and $\gamma \neq \varepsilon$. We assign $\nu \to e'$ and $\phi \to g$ as in case c$_2$.

   e$_2$) Let $\alpha \mid \beta$. Then $\gamma \mid \alpha$ and hence $\alpha \nmid \varepsilon$, $\delta \nmid \gamma$, $\beta \neq \delta$, and $\beta \neq \varepsilon$. We assign $\mu \to e'$ and $\nu \to g$ as in case c$_1$.

   e$_3$) Let $\alpha \nmid \beta$ and $\gamma \nmid \delta$. $\beta \mid \gamma$ implies that $\varepsilon \neq \beta$ and $\varepsilon \neq \gamma$. Then we assign $\kappa \to e'$ and $\lambda \to g$ as in case b$_1$.

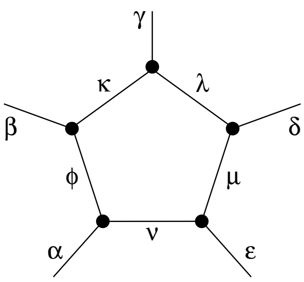

Figure 3.15

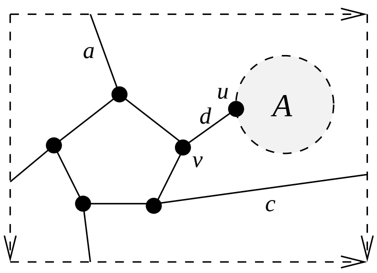

Figure 3.16

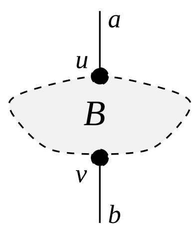

Figure 3.17

Now we assume that one of the remaining 5 cases holds:

- $\alpha = \gamma$ and $\beta = \varepsilon$
- $\beta = \delta$ and $\gamma = \alpha$
- $\gamma = \varepsilon$ and $\delta = \beta$

- $\delta = \alpha$ and $\varepsilon = \gamma$
- $\varepsilon = \beta$ and $\alpha = \delta$

In these cases, there exists an assignment of the edges in Figure 3.15 to the edges in Figure 3.5 such that in Figure 3.5 it holds both $a = f$ and $b = c$. For example, if $\alpha = \gamma$ and $\beta = \varepsilon$, then we assign $\phi \to e'$ and $\kappa \to g$. Thus, the situation is as shown in Figure 3.16. The S_3 is not reducible. But we will prove now that a planar submap A as in Figure 3.16 (the grey area may be replaced by an arbitrary planar map) either contains at least one reducible submap S_1–S_9 or is itself an S_5 (where A contains the two vertices u and v and the edge d).

First, we show that A is indeed planar: It holds $f_M = f_A + 1$, $e_M = e_A + 7$, and $n_M = n_A + 4$ (see Figure 3.16). Euler's formula yields $0 = 2 - 2g_M = n_M - e_M + f_M = n_A + 4 - e_A - 7 + f_A + 1 = n_A - e_A + f_A - 2 = -2g_A$, and thus we have $g_A = 0$. In the same way, it can be shown that a submap is planar whenever we claim that a submap is planar throughout the rest of this chapter.

Now we go back to the assertion that A either contains at least one reducible submap S_1–S_9 or is itself an S_5. If A is an S_5, then we are done. In Figure 3.16, this S_5 is reducible (compare Table 3.1). Let now A not be an S_5. If we remove d and u, then the resulting connected component A' corresponding to A is cubic and planar (not necessarily simple, namely if u was incident with a 3–cycle before the removal). Hence formula 1.3 holds, and this formula implies that A' contains at least three faces of degree smaller than 6 (because $\sum_{i=1}^{5}(6-i)p_i(A')$ must be at least 12 and hence $12 \leq \sum_{i=1}^{5}(6-i)p_i(A') \leq \sum_{i=1}^{5} 5p_i(A') = 5\sum_{i=1}^{5} p_i(A')$). At most one of these faces is the outermost face, at most one of these faces is the second face that has been modified by removing u. Hence at least one face that has not been modified has a degree smaller than 6, and thus also A contains at least one face F whose degree is smaller than 6 where F is not the outermost face, i.e. F is not incident with d (this is impossible anyway because d is a face–bridge).

Case 1: A contains a pentagon. Then this pentagon is reducible because $a = f$ and $b = c$ (Figure 3.5) cannot hold simultaneously in a planar map: If $a = f$, then $\langle a, e', g\rangle$ is a cycle. Moreover, it holds $\rho_v(e') = h$ and $\overline{\rho_u}(g) = c$, so that due to the Jordan curve theorem on page 22, there is no path p with start edge h and end edge c such that p is not incident with a, e', or g. In particular, this does not hold for the path $p = \langle h, c\rangle$. So $b = c$ is impossible (because $\langle h, b\rangle$ is a path in A that contains neither a, e', nor c). Hence either $a \neq f$ or $b \neq c$, and one of the above cases (a–e) is applicable.

Case 2: A does not contain a pentagon, but a square. See Figure 3.4.

- If $a = c$, then $b \neq d$, and b and d cannot be adjacent neither to each other nor to a common edge due to the Jordan curve theorem (the argumentation is analogous to the one in case 1; the cycle is again $\langle a, e', h\rangle$). Hence the square may be reduced using ${R_8}^-$.
- If $b = d$, then the same argument as in the previous subcase can be applied.
- Let $a \neq c$ and $b \neq d$. Let a and b be adjacent. Let w be the vertex which is incident with a and b, and let k be the third edge that is incident with w. If the 3–cycle $\langle a, h, b\rangle$ is not a triangle, then it holds $\rho_u(h) = e'$ and $\overline{\rho_w}(a) = k$. Hence due to the Jordan curve theorem, k and e' are not adjacent. For the same reason, k and f are not adjacent. Moreover, e' and f are not adjacent and h is no face–bridge. Thus, the 3–cycle $\langle a, h, b\rangle$ may be reduced using ${R_6}^-$. So let now $\langle a, h, b\rangle$ be a triangle.
  - If a and d are adjacent, then for the vertex w being incident with a, d, and b, it holds $r_\rho(w) = \langle b, a, d\rangle$. Hence with $\rho_u(e') = a$ and $\rho_v(d) = e'$, we obtain that $\langle a, d, e'\rangle$ is also a

triangle. Since all incident edges of u, v, and w are fixed, c is adjacent neither to b nor to d, and thus we have an S_5 where c corresponds to e' in Figure 3.7. Now look at Figure 3.7. The S_5 is either reducible (then we are done), or a and b are elements of a 3–cycle $\langle a, b, c\rangle$.

- If $\langle a, b, c\rangle$ is a triangle, then the situation is as shown in Figure 3.18: It holds $\rho_u(b) = c$ and $\rho_v(c) = a$. a and b cannot be incident with another 3–cycle except $\langle a, b, c\rangle$ because otherwise this 3–cycle would contain both a and b, and then for the third element c' of the cycle it would hold that $\{c, c'\}$ is a double edge. If c is incident with another 3–cycle $\langle c, f, g\rangle$, then let w be the vertex which is incident with both f and g. Moreover, let w.l.o.g. f be incident with u in Figure 3.18. If $\rho_w(f) = g$, then $\langle c, f, g\rangle$ is a triangle so that we have a reducible S_4 (see Figure 3.19). If $\rho_w(g) = f$, then the 3–cycle $\langle c, f, g\rangle$ does not correspond to a triangle (see Figure 3.20). Here we have an S_7 which is reducible because e' is not adjacent to k, k and e' are not adjacent to a common edge, and neither f nor g is a face–bridge.
- If it does not hold $\rho_u(b) = c$ and $\rho_v(c) = a$, then $\langle a, b, c\rangle$ is no triangle. Thus the situation is as shown in Figure 3.21 where one edge incident with u and one edge incident with v are not yet drawn.
  - If $\rho_u(b) = c$ and $\rho_v(a) = c$ (see Figure 3.22), then due to the Jordan curve theorem, g is not adjacent to f. Moreover, e' is adjacent neither to g nor to f. Finally, b is no face–bridge: Let F_1 be the face that contains f, b, e', a, and c. Let F_2 be the face that contains g, a, b, and c. If b is a face–bridge, then $F_1 = F_2$, and hence f and g are incident with a common face $F_1 = \langle b, e', \ldots, e', a, c, f, \ldots, g, a, b, c, g, \ldots, f\rangle$. Thus there would exist a path $p = \langle f, \ldots, g\rangle$ containing none of the edges a, b, or c. This is impossible due to the Jordan curve theorem. Hence the 3–cycle $\langle a, b, c\rangle$ may be reduced using ${R_6}^-$.
  - If $\rho_u(c) = b$ and $\rho_v(c) = a$, then the same arguments can be applied.
  - Let finally $\rho_u(c) = b$ and $\rho_v(a) = c$. c is no face–bridge (see Figure 3.23). If f and g are not adjacent, then the 3–cycle $\langle a, b, c\rangle$ may be reduced using ${R_6}^-$ because e' is adjacent neither to f nor to g. If f and g are adjacent, then let w be the common incident vertex. Depending on whether $\rho_w(f) = g$ or $\rho_w(g) = f$, either $\langle a, b, f, g\rangle$ is a square (see Figure 3.24) or $\langle g, f, c\rangle$ is a triangle (see Figure 3.25). If $\rho_w(f) = g$, then ${R_8}^-$ may be applied because e' and k are adjacent neither to each other nor to a common edge, and c is no face–bridge. If $\rho_w(g) = f$, then ${R_7}^-$ may be applied because e' and k are adjacent neither to each other nor to a common edge, and neither a nor b is a face–bridge.

  - If c and b are adjacent (in Figure 3.4), then the same argumentation as in the previous subcase can be applied.
  - If neither a and d nor c and b are adjacent, then the S_2 may be reduced using g or h as the reference edge (rooted appropriately).

- Let $a \neq c$ and $b \neq d$. Let c and d be adjacent. The same argumentation as in the previous subcase can be applied.
- Let $a \neq c$ and $b \neq d$. Let neither a and b nor c and d be adjacent. Then the square may be reduced using ${R_2}^-$.

Case 3: A contains neither a pentagon nor a square. Hence A contains a triangle $\langle e', f, d\rangle$ as in Figure 3.3 (see there).

- Let $\langle a, c, e'\rangle$ be a 3–cycle but not a triangle. Thus $\rho_v(c) = a$ where v is the vertex that is incident with both a and c. Let x be the third edge which is incident with v. (in Figure 3.9,

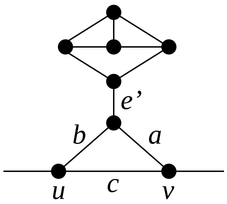

Figure 3.18

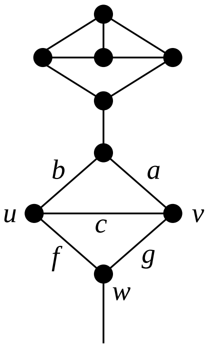

Figure 3.19

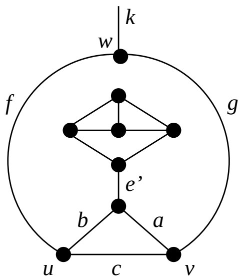

Figure 3.20

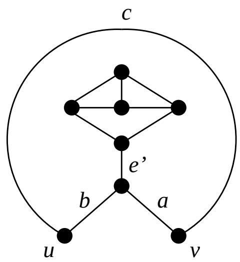

Figure 3.21

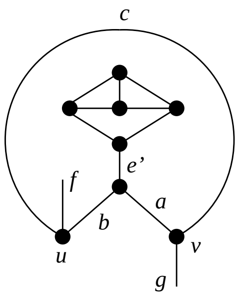

Figure 3.22

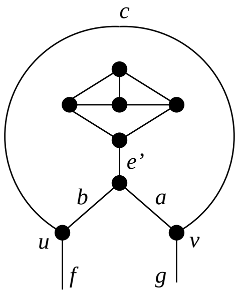

Figure 3.23

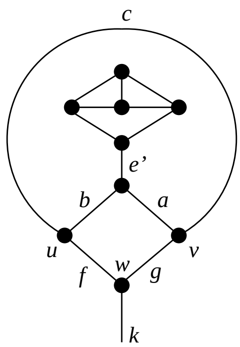

Figure 3.24

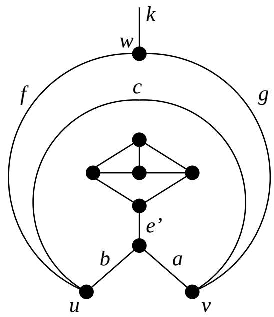

Figure 3.25

$x = g$). Due to the Jordan curve theorem, c and a (Figure 3.3) are no face–bridges because otherwise there would be a path p with x being the start edge, b being the end edge, and none of the edges a, c, and e' being incident with p. Moreover, it holds $x \neq b$ (otherwise we would have a connected component isomorphic to K_4'' so that A would be disconnected), and x and b cannot be adjacent neither to each other nor to a common edge (due to the Jordan curve theorem). Hence ${R_7}^-$ may be applied.

- Let $\langle a, c, e' \rangle$ be a triangle. See Figure 3.6. It holds $g \neq f$ because otherwise we would have a connected component isomorphic to a planar embedding of K_4 so that A would be disconnected. Furthermore, f and g are not adjacent because otherwise A would contain a square.
  - If f and g are adjacent to a common edge x, then due to the assumption, neither the 5–cycle $\langle a, b, f, x, g \rangle$ nor the 5–cycle $\langle f, c, d, g, x \rangle$ is a pentagon. Hence we have a submap S_9. See Figure 3.11. As usual, due to the Jordan curve theorem, a is no face–bridge, and b and c are adjacent neither to each other nor to a common edge except a. Hence ${R_9}^-$ may be applied.
  - If f and g in Figure 3.6 are not adjacent to a common edge x, then ${R_4}^-$ may be applied.
- The same argumentation holds if $\langle a, d, b \rangle$ or $\langle b, f, c \rangle$ instead of $\langle a, c, e' \rangle$ is a 3–cycle in Figure 3.3.
- If neither $\langle a, d, b \rangle$ nor $\langle b, f, c \rangle$ nor $\langle a, c, e' \rangle$ is a 3–cycle, then the triangle $\langle e', f, d \rangle$ may be reduced by applying ${R_1}^-$.

This completes the proof that A — and hence also every map $M \in \mathcal{T}$ containing a pentagon — contains at least one reducible submap $S_1 - S_9$.

3.3.4 More preparation

Next, we will prove that a planar submap like B in Figure 3.17 (containing the vertices u and v) either contains at least one reducible submap $S_1 - S_9$ or is an S_4.

If B is an S_4, then we are done. Let B not be an S_4. If we remove u and v (in order to do this, we must also remove a and b first), then the connected component B' that corresponds to B is cubic and planar. Hence formula 1.3 holds, and as above, it follows that B' contains at least three faces of degree smaller than 6. At most one of these faces is the outermost face.

- If u and v are adjacent in B, then B and B' can be sketched as in Figure 3.26. Here C and E are the only faces that are modified by the removal of u and v, and hence B contains at least one face of degree smaller than 6.
- If u and v are not adjacent in B, then B and B' can be sketched as in Figure 3.27. Here C, D, and E are modified by the removal of u and v.
  - If $\deg(C) \geq 6$ or $\deg(D) \geq 6$, then at least one of the three faces in B' of degree smaller than 6 is not contained in $\{C, D, E\}$. Let H be this face. Since H has not been modified by the removal of u and v, B also contains a face of degree smaller than 6.
  - Let $\deg(C) < 6$ and $\deg(D) < 6$. It holds $\deg(E) \geq 2$ (otherwise u and v would be adjacent in B) and also $\deg(C) \geq 2$ and $\deg(D) \geq 2$. Hence B' does not contain monogons because all other faces have not been modified and must therefore each be of degree at least 3. Moreover, it must hold $\deg(C) > 2$ or $\deg(D) > 2$ because otherwise B would be an S_4. So C or D is a triangle, a square, or a pentagon. Thus $x := p_3(B') + p_4(B') + p_5(B') \geq 1$, but

$$\begin{aligned} 12 &\leq \sum_{i=2}^{5} p_i(B') \cdot (6-i) \;=\; 4p_2(B') + \sum_{i=3}^{5} p_i(B') \cdot (6-i) \;\leq\; 4p_2(B') + \sum_{i=3}^{5} 3p_i(B') \\ &= 4p_2(B') + 3x \;=\; 4 \cdot (p_2(B') + x) - x \;=\; -x + 4\sum_{i=2}^{5} p_i(B') \end{aligned}$$

  so that $13 \leq 12 + x \leq 4\sum_{i=2}^{5} p_i(B')$ and hence $\sum_{i=2}^{5} p_i(B') \geq 4$. Hence B' contains at least 4 faces of degree smaller than 6. Since only three faces have been modified, B also contains a face of degree smaller than 6.

Now in the same way as for A, it can be proved that B contains at least one reducible submap $S_1 - S_9$.

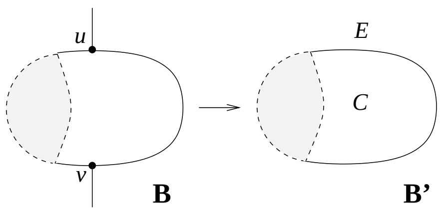

Figure 3.26

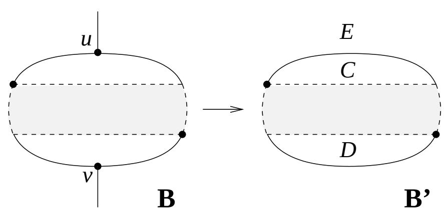

Figure 3.27

Next, we will show that every map $M \in \mathcal{T} - \mathcal{H}$ that does not contain a pentagon is either a basic map or contains at least one reducible submap $S \in \{S_1, S_2, S_4, \ldots, S_9\}$.

3.3.5 Step 2

Let $M \in \mathcal{T} - \mathcal{H}$ not contain a pentagon, but a square. Look at Figure 3.4.

- If $a = c$ or $d = b$, then the square is contained in an S_8. Let w.l.o.g. $a = c$. If $d = b$, then M is a $K_4' \in \mathcal{B}$. Hence we presume that $d \neq b$.
  - If b and d are adjacent to each other, then the situation is as shown in Figure 3.28`i`. A is planar. As shown above, A is either an S_5 (with reference edge k) which is reducible in this case, or A contains another reducible submap.
  - If b and d are not adjacent to each other but to a common edge m, then let u be the vertex incident with d and m, and let v be the vertex incident with b and m. There are 4 cases: Either $\rho_u(m) = d$ and $\rho_v(m) = b$ (case `ii`, see Figure 3.28), $\rho_u(d) = m$ and $\rho_v(m) = b$ (case `iii`, see Figure 3.28), $\rho_u(d) = m$ and $\rho_v(b) = m$ (similar to case `ii`), or $\rho_u(m) = d$ and $\rho_v(b) = m$ (similar to case `iii`).
    - Let k and l be adjacent, and let w be the vertex that is incident with both k and l.
      - If $r_\rho(w) = \langle k, l, x\rangle$ (for some edge x) in case `ii`, then M contains the face $F = \langle k, l, m, d, e', a, g, d\rangle$. Thus l is no face–bridge (it appears only once in the sequence of F), and d is adjacent neither to x nor to b. Moreover, x and b are not adjacent to each other. So the conditions for S_6 in Table 3.1 are not satisfied and the 3–cycle $\langle k, l, m\rangle$ may be reduced applying $R_6{}^-$.
      - If $r_\rho(w) = \langle l, k, x\rangle$ in case `ii`, then M contains the face $\langle l, k, m, b, f, a, h, b\rangle$. Thus k is no face–bridge, and b is adjacent neither to x nor to d. Moreover, x and d are not adjacent to each other. So the conditions for S_6 in Table 3.1 are not satisfied and the 3–cycle $\langle k, l, m\rangle$ may be reduced applying $R_6{}^-$.
      - If $r_\rho(w) = \langle k, l, x\rangle$ in case `iii`, then the 3–cycle $\langle k, l, m\rangle$ is an isolated triangle that may be reduced applying $R_1{}^-$.
      - If $r_\rho(w) = \langle l, k, x\rangle$ in case `iii`, then M contains the face $\langle l, k, d, e', a, g, d, m, b, f, a, h, b\rangle$. Thus m is no face–bridge, and x is adjacent neither to b nor to d. Moreover, b and d are not adjacent to each other. So the 3–cycle $\langle k, l, m\rangle$ may be reduced applying $R_6{}^-$.
    - Let k and l not be adjacent. Let both k and l be face–bridges as the edge k in Figure 3.28`i`. Then as in case `i`, there are two planar submaps in M containing a reducible submap each.
    - Let k and l not be adjacent. Let k and l correspond to a and b in Figure 3.17.
      - If B is an S_4, then case `iii` is impossible because the edges k, l, and m would be

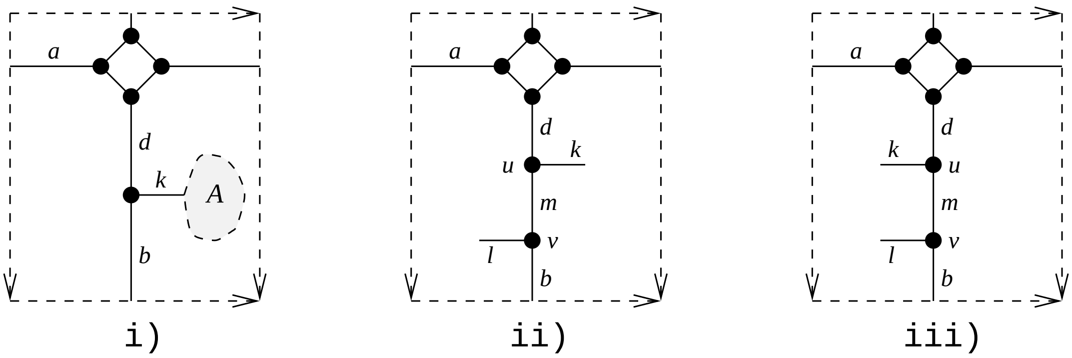

Figure 3.28: Three situations for an irreducible S_8 (mirror images excluded)

incident with a pentagon. In case **ii**, the S_4 is not maximal, but together with k, l, and m (and the incident vertices), the S_4 is contained in a reducible S_9.

- If B is not an S_4, then it contains a reducible submap as shown above.

– Let b and d be adjacent neither to each other nor to a common edge m. If a (in Figure 3.4) is a face–bridge, then b and d correspond to b and a in Figure 3.17. B is either a reducible S_4 or contains a reducible submap (B contains all edges except a, e', f, g, and h in Figure 3.4). If a is no face–bridge, then ${R_8}^-$ may be applied using b or d as the reference edge (rooted appropriately).

- Let $a \neq c$ and $d \neq b$ in Figure 3.4. If neither a and b nor c and d are adjacent, then ${R_2}^-$ may be applied. The same holds if neither a and d nor b and c are adjacent. In this case, we use g or h as the reference edge for the application of ${R_2}^-$. In the remaining cases, either a is adjacent to both b and d, b is adjacent to both a and c, c is adjacent to both b and d, or d is adjacent to both a and c. Let w.l.o.g. a be adjacent to both b and d. Let w be the vertex incident with a, b, and d.

  – If $\langle a, h, b\rangle$ is a triangle, then we can proceed as in Case 2 of Step 1 of this proof (Case 2 of Step 1 begins on page 34, the relevant item starts right after the sentence "So let now $\langle a, h, b\rangle$ be a triangle" where it is assumed that a and d are adjacent). The only difference is that we cannot apply the Jordan curve theorem. So we must reconsider the case $\rho_u(b) = c$ and $\rho_v(a) = c$ (see Figure 3.22). We argue as follows: If the S_6 containing a, b, and c is reducible, then we have found a reducible submap in M. And if the S_6 is irreducible, then due to Table 3.1, b is a face–bridge or f and g are adjacent to each other (e' is adjacent neither to f nor to g). In both cases, the situation is as shown in Figure 3.29. The situation corresponds to the one in Figure 3.28 where f and g in Figure 3.29 correspond to d and b in Figure 3.28.

    - If f and g are adjacent, then the situation corresponds to the one shown in Figure 3.28**i**. See there. A is planar and either A contains a reducible submap S_1, S_2, or S_4–S_9, or A is an S_5. In this case, let w be the vertex that is incident with f and g. Depending on whether $\rho_w(g) = f$ or $\rho_w(f) = g$, M is one of the basic maps B_5 or B_6.
    - If f and g are not adjacent, then b is a face–bridge. Thus M contains a face $F = \langle b, e', \ldots, e', a, c, f, \ldots, g, a, b, c, g, \ldots, f\rangle$. As a consequence, f and g correspond to a and b in Figure 3.17. So M contains a submap B as in Figure 3.17 where B is planar. If B does not contain a reducible submap S_1, S_2, or S_4–S_9, then B is an S_4 so that M is the basic map B_2.

  – Let $\langle a, h, b\rangle$ in Figure 3.4 not be a triangle. Then $\langle a, d, e'\rangle$ is also not a triangle (because $\rho_v(d) = e'$, $\rho_u(e') = a$, but $\rho_w(a) = b$ and hence $\rho_w(a) \neq d$) and the situation is as shown in Figure 3.30. As usual, A (containing c and w) is planar and either A is a reducible S_5 or A contains another reducible submap.

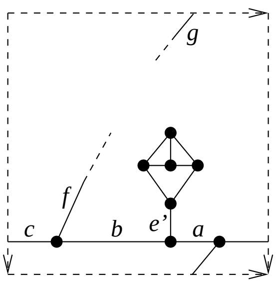

Figure 3.29

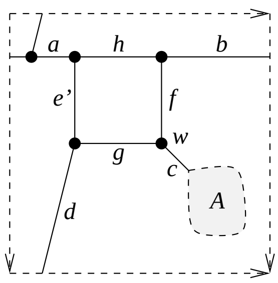

Figure 3.30

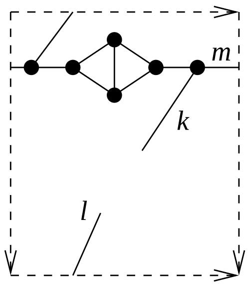

Figure 3.31

3.3.6 Step 3

Let $M \in \mathcal{T} - \mathcal{H}$ contain neither a pentagon nor a square.

Thus M contains a triangle. Look at Figure 3.3. If neither a and b nor a and c nor b and c are adjacent, then the triangle is isolated and may be reduced applying ${R_1}^-$. Let w.l.o.g. a and b be adjacent.

- If $\langle a, d, b\rangle$ is a triangle, then we have an S_4. c is adjacent neither to a nor to b because otherwise we would have a planar embedding of K_4. Look at Figure 3.6. The edges f and g are not adjacent because otherwise M would contain a square.
  - If f and g are not adjacent to a common edge m, then the S_4 is reducible.
  - If f and g are adjacent to a common edge m, then due to the assumption, the 5–cycles containing f, g, and m are no pentagons. Hence we have an S_9. If this S_9 is not reducible, then due to Table 3.1, the edge m in Figure 3.31 is a face–bridge or k and l are adjacent to each other or to a common edge except m. Thus the situation in Figure 3.31 is similar to the one in Figure 3.29. Hence the argumentation for the existence of a reducible submap is similar to that given above.
    - If k and l are adjacent to each other, then as in Step 2 of this proof, M contains a planar submap A containing a reducible submap (S_1, S_4, S_6, S_7, or S_9). A cannot be an S_5 because M does not contain squares.
    - If k and l are not adjacent to each other but to a common edge x ($x \neq m$), then the situation corresponds to the one in Figure 3.28`ii` or `iii`. Thus the arguments of Step 2 of this proof apply except that if x is contained in another S_9, then this S_9 is not reducible. Instead, M is one of the basic maps B_3 or B_4.
    - If m is a face–bridge, then as usual, k and l correspond to a and b in Figure 3.17 so that M contains a planar submap B which either contains a reducible submap or is an S_4 so that M is the basic map B_1.
- Let the 3–cycle $\langle a, d, b\rangle$ in Figure 3.3 be no triangle. In this case, we have an S_7. If c is adjacent to a or b, then we have $K_4'' \in \mathcal{B}$, else with the same arguments as for the S_8 (in step 2 of this proof) one can show that M contains a reducible submap. For that, replace the S_8 in Figure 3.28 by an S_7.

This completes the proof of Theorem 3.1. ■

Chapter 4

Fast construction of $\mathcal{T}_n$

Since $\mathcal{T}$ is infinite, the above inductive definition can of course not be used to actually construct $\mathcal{T}$, but it can be used to construct $\mathcal{T}_n$ or $\mathcal{T}_n - \mathcal{T}_{n-2}$ for a given n.

We will now present an algorithm that constructs $\mathcal{T}_n$. Starting with all basic maps of order $\leq n$, the algorithm applies sequences of the generating rules $R_1 - R_{10}$ in every possible way. This means that every edge of a map will be rooted in every possible way in order to serve as a reference edge, and then every generating rule that *can* be applied to this reference edge (due to the restrictions listed in Section 3.2) *will* be applied. This procedure will be repeated as long as the resulting maps do not have more than n vertices. Since every application of a generating rule adds vertices to a map, it is then secured that no map $M \in \mathcal{T}_n$ will be missed, or to be precise: Of each isomorphism class of maps in $\mathcal{T}_n$, at least one representative will be constructed.

The remaining task is to secure that *exactly* one representative of each isomorphism class is accepted, i.e. no two isomorphic maps will be accepted. To construct $\mathcal{T}_n$, we apply the *McKay–type orderly generation* principle which is discussed in [44] and [45]. Let $D(M, R, r(e))$ denote the map that is obtained by applying the generating rule $R \in \{R_1, \ldots, R_{10}\}$ to a map $M \in \mathcal{T}$ where $r(e) = (e, h, s_e)$ is the rooted reference edge (as in Section 3.2). We presume that the application is permitted due to the restrictions in Section 3.2, i.e. $D(M, R, r(e)) \in \mathcal{T}$. Moreover, let $A(M, R, r(e'))$ denote the map which is obtained by applying the inverse $R \in \{R_1{}^-, \ldots, R_{10}{}^-\}$ of a generating rule to a map $M \in \mathcal{T}$ with respect to the rooted reference edge $r(e')$. Here we also presume that the application is permitted (in particular, if $R = R_i{}^-$, then $r(e')$ must be the rooted reference edge of a submap S_i). $D(M, R, r(e))$ is called the **descendant** of M with respect to R and $r(e)$, and $A(M, R, r(e'))$ is called the **ancestor**. In principle, the parameter R is redundant in $A(M, R, r(e'))$ because the rooted reference edges in $S_1 - S_{10}$ are fixed in a way that a rooted edge can be the rooted reference edge of at most one reducible (maximal) submap. Thus $r(e')$ implies R.

In order to secure that exactly one representative of each isomorphism class is accepted, we need to consider the following problems:

- Never do the same thing twice.

Since every application of a generating rule adds vertices to a map, no sequence of generating

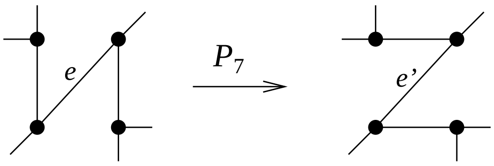

Figure 4.1

rules will transform one map into itself. Hence we will never do exactly the same thing twice. This is the reason why we do not use generating rules like rule P_7 in [11] (see Figure 4.1). Here, with e and e' rooted arbitrarily, it holds

$$D(D(M, P_7, r(e)), P_7, r(e')) = M$$

so that it is difficult to avoid copies of one map M during the construction process. However, avoiding doing exactly the same thing twice is not the only problem. We must also avoid applying one generating rule R to two isomorphic rooted edges $r(e_1)$ and $r(e_2)$ inside a map M because otherwise the resulting maps $D(M, R, r(e_1))$ and $D(M, R, r(e_2))$ would obviously be isomorphic, too. Hence before applying generating rules to M, we compute the isomorphism classes of all rooted edges in M, and then we take one representative out of each class. This representative serves as a rooted reference edge for generating rules. In Section 4.2, it will be explained in detail how the isomorphism classes of the rooted edges are determined.

- If a map can be constructed in more than one way (applying different sequences of generating rules), then accept only one of them.

This is obviously a problem for every map M that contains more than one reducible submap. Each of these submaps can be the one that has been inserted last. Since we apply generating rules in every possible (non–isomorphic) way, M will be constructed several times. But even for a single reducible submap, it is often not unique how this submap can be reduced. For example, in an S_3, there are up to 10 rooted edges that can serve as the rooted reference edge $r(e')$ (each of the five elements of the pentagon can be directed in two possible ways; the side is then given by the side on which the pentagon is located). So if the rooted reference edge $r(e')$ is not unique for a reducible submap S in M, then it is also not unique how M can be constructed inserting S into a smaller map.

We solve this problem in the following way: To every map $M \in \mathcal{T} - \mathcal{B}$, we assign a unique ancestor from which it must be constructed and a way how it must be constructed from it. This is done by assigning a **canonical generating rule** R and a **canonical rooted edge** $r(e')$ to M so that up to isomorphism, the map $A(M, R^-, r(e')) \in \mathcal{T}$ is the unique **canonical ancestor** of M. We accept M only if M has been obtained by applying the canonical generating rule R to the canonical ancestor of M in a way that in M, the obtained rooted reference edge is the canonical rooted edge. In all other cases, M will be rejected and no further generating rules will be applied to M. Thus M will be accepted exactly once even though it might be constructed several times.

The construction can be visualized using a graph as in Figure 4.2: Every map is represented by a vertex. Here $A_1 - A_4$ are four ancestors of M, but only one of them is canoncial. Every edge represents the application of a generating rule. A dotted edge between M and its ancestor A indicates that the application of the generating rule is not canoncial, i.e. either A is not the canonical ancestor of M or the obbtained rooted reference edge in M is not the canonical rooted edge. The solid edge indicates that A_2 is the canoncial ancestor of M and the application of the corresponding generating rule is canonical, too. If there is more than one way to construct M out of one ancestor A, then we have a multiple edge between A and M in the visualization (where at most one edge is solid and all other ones are dotted).

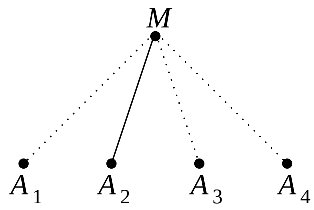

Figure 4.2

If we consider more than one map M, then the visualization becomes more complex: In Figure 4.3, it is visualized how six maps $M_1 - M_6$ can be constructed out of three basic maps $B_1 - B_3$. Since every map (except the basic maps) has exactly one canoncial ancestor, there is exactly one solid edge connecting a map with another map on a lower level (here the levels represent the orders of the maps). If we remove all dotted edges, then we obtain a forest where every basic map corresponds to the tree–source of a tree in that forest, and the source of every branch b represents the canonical ancestor of the map that corresponds to the target of b. Thus, defining canonical ancestors results in a tree–like construction of the maps. We will refer to the forest as the **construction forest**.

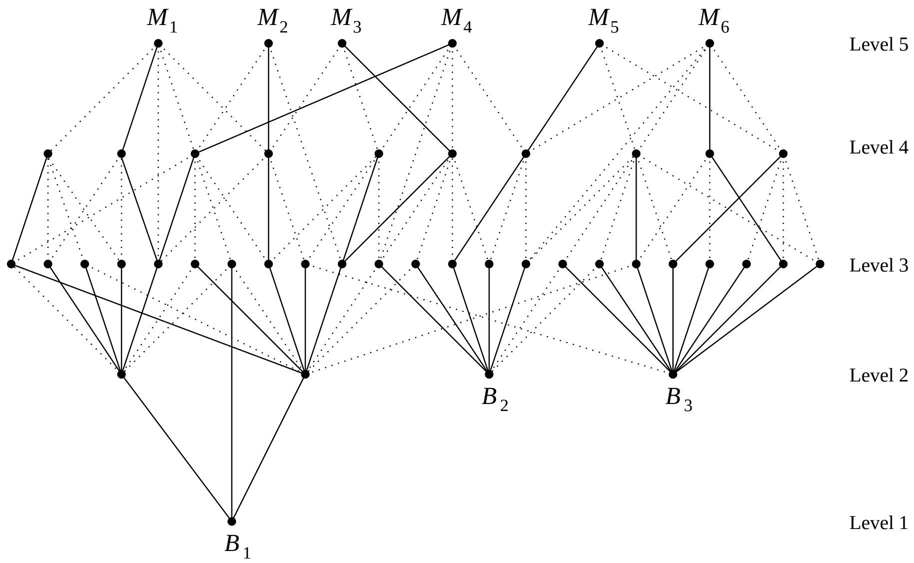

Figure 4.3

In order to assign a canonical generating rule to M, **qualities** are assigned to the reducible submaps in M. To S_i, we assign the quality $10 - i$. Let E_i, $i \in \{1, \ldots, 10\}$, be the set of all rooted edges in M that can be used as rooted reference edges $r(e')$ for the application of ${R_i}^{-}$. In particular, E_i is non–empty if and only if M contains a reducible submap S_i. Let $j \in \{1, \ldots, 10\}$ be the smallest number for which E_j is non–empty. Then M does not contain a submap with a higher quality than S_j, and hence R_j is the canonical generating rule and one element of E_j is the canonical rooted edge. We distinguish the rooted edges in E_j by encoding the map M with respect to the rooted edges and comparing the codes. This will be explained in detail in the next two sections.

4.1 Uniquely encoding a simple rooted map

Suppose $M = (G, \Pi)$ with $G = (V, H, E, T)$ contains n vertices. Suppose $(V \cup H) \cap \{1, 2, \ldots, n\} = \emptyset$. Then the vertices will be labelled with the numbers $1, \ldots, n$ in increasing order. In the following description, we will use also the new labels in order to refer to the vertices. This works well because we will only refer to vertices by their labels if they have been labelled before.

The source of the semi–edge h indicating the direction of the root $r(e) = (e, h, s_e)$ is labelled 1 and the target is labelled 2. Next, we label the neighbours of vertex 1 that have not yet been labelled, and that is done starting with vertex 2 using the semi–edge rotation Π if $s_e = \texttt{right}$, and its inversion $\overline{\Pi}$ if $s_e = \texttt{left}$, respectively. Thus, we label the unlabelled targets of $\pi_1(h)$, $\pi_1^2(h)$, … or $\overline{\pi_1}(h)$, $\overline{\pi_1}^2(h)$, …, respectively. For the labelling, it is always taken the smallest remaining (unplaced) label. Thus in the cubic case, the two remaining unlabelled neighbours get the labels 3 and 4.

Next, we repeat the following steps for the vertices $v = 2, \ldots, n$: At least one of the neighbours of v is labelled (v is labelled and there must be a path p connecting vertex 1 with v such that p is incident only with labelled vertices). Starting with the neighbour u that has the smallest label among all labelled neighbours, the neighbours of v are glanced over in the order given by π_v or $\overline{\pi_v}$ depending on s_e as above, and whenever an unlabelled neighbour w occurs, the smallest remaining label will be assigned to w. In detail: Let h be the semi–edge with $T(h) = (v, u)$. Then we label the unlabelled targets of $\pi_v(h)$, $\pi_v^2(h)$, … or $\overline{\pi_v}(h)$, $\overline{\pi_v}^2(h)$, …, respectively, in exactly this order. Since M is connected, every vertex is reached and labelled.

From this labelling, we obtain the unique **vertex code** of $(M, r(e))$ by recording the labels of the targets of the semi–edges in the order they appear whenever a rotation π_v or $\overline{\pi_v}$ is applied for some vertex v. Here the labels are recorded no matter if the targets have already been labelled before or not. Thus for every vertex $v = 1, \ldots, n$, the labels of all neighbours are recorded in a well–defined order. After all neighbours of one vertex have been recorded, a zero is inserted into the code. Thus in the cubic case, we obtain a sequence of length $4n$ (each of the n vertices has 3 neighbours; additionally, we have n zero entries). For practical reasons, the order n of M precedes the code, but in principle this number is redundant. Figure 4.4 shows an example of a rooted map. Its vertex code with respect to $r(e)$ is $\langle 10,\ 2,3,4,0,\ 1,5,6,0,\ 1,7,4,0,\ 1,3,7,0,\ 2,8,7,0,\ 2,9,10,0,\ 3,5,4,0,\ 5,10,9,0,\ 6,8,10,0,\ 6,9,8,0 \rangle$.

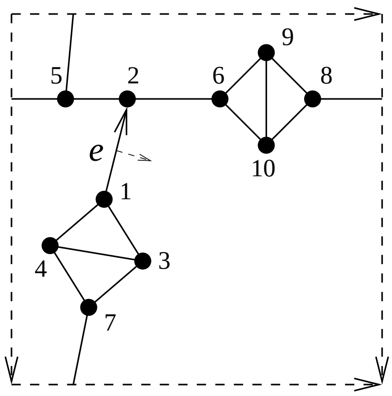

Figure 4.4

4.2 Comparing rooted edges

If the above unique encoding is applied in order to compare two rooted edges $r(e_1)$ and $r(e_2)$ in M, then identical vertex codes are assigned exactly to isomorphic rooted edges (where the labelling induces the map automorphism). Otherwise the codes differ. Hence the vertex codes can always be used to assign qualities to the rooted edges. In *togen*, the quality of a rooted edge increases with a lexicographically decreasing vertex code (since the quality is used only to compare rooted edges, this relation suffices; the precise values of the qualities are not important).

However, the encoding and comparison of two rooted edges in a cubic map of order n takes $O(n)$ time. In most cases, simpler criteria help to assign different qualities to the rooted edges in E_j and hence to distinguish non–isomorphic rooted edges in E_j. In *togen*, the following criteria are applied in the order they are listed (i.e. the $(i+1)$–th criterion is applied only if the preceding i criteria did not yield different results):

(1) *Zig–zag pseudo–paths* of length 11 are considered in the sense that the degrees of the faces that are incident with the paths are compared in the following way: Let $r(e_1) = (e_1, h_1, s_{e_1})$ and $r(e_2) = (e_2, h_2, s_{e_2})$ be the rooted edges in M that are to be compared. Then starting with h_1 and h_2, respectively, zig–zag pseudo–paths $p_1 = \langle a_1, a'_1, \ldots, a_{11}, a'_{11}\rangle$ and $p_2 = \langle b_1, b'_1, \ldots, b_{11}, b'_{11}\rangle$ are traced (here the a_i, a'_i, b_i, and b'_i are semi–edges). Here $a_1 = h_1$ and $b_1 = h_2$, and the *first continuation* is indicated by s_{e_1} and s_{e_2}, respectively. Let k_i and l_i be the i–th continuation of p_1 and p_2, respectively. It holds $k_i, l_i \in \{\texttt{left}, \texttt{right}\}$. For every $i \in \{1, \ldots, 10\}$, let $c_i = \{a_i, a'_i\}$, $d_i = \{b_i, b'_i\}$, let C_i be the root face of $(M, r(c_i))$ where $r(c_i) := (c_i, a_i, k_i)$, and let D_i be the root face of $(M, r(d_i))$ where $r(d_i) := (d_i, b_i, l_i)$. Then the sequences $\langle \deg(C_1), \ldots, \deg(C_{10})\rangle$ and $\langle \deg(D_1), \ldots, \deg(D_{10})\rangle$ are compared with each other. The quality of a rooted edge increases with a lexicographically decreasing sequence. Note that the sequences have length 10 (not 11). The number 10 was chosen using heuristics.

(2) As a second step, again zig–zag pseudo–paths of length 11 are considered, but this time the first continuations of the paths are flipped: If s_{e_1} or s_{e_2} is `right`, then the first continuation of p_1 or p_2, respectively, is `left` and vice versa.

(3) The degrees of those faces that are incident with the root faces in $(M, r(e_1))$ and $(M, r(e_2))$ are compared pair by pair. For that, if $s_{e_i} = \texttt{right}$, then starting with e_i, the sequence of the corresponding root face is traced, and if $s_{e_i} = \texttt{left}$, then starting with e_i, the inverse of the sequence of the corresponding root face is traced. The details are similar to those in step 1.

(4) As a fourth step, step 3 is repeated where the sides of $r(e_1)$ and $r(e_2)$ are flipped so that the opposite faces are traced.

In this context, the determination of j such that R_j is the canonical generating rule (and one rooted edge in E_j is the canonical rooted edge) can be considered as "criterion 0" because it also helps to reduce the number of vertex codes that have to be generated. However, the determination of a canonical generating rule has another advantage that will be discussed in Section 4.3.

Only if none of these 5 criteria is suitable to distinguish two non–isomorphic rooted edges $r(e_1)$ and $r(e_2)$ in M, then two vertex codes of M are generated with respect to the roots $r(e_1)$ and $r(e_2)$. In *togen*, the vertex codes of the two rooted maps $(M, r(e_1))$ and $(M, r(e_2))$ are determined simultaneously so that $r(e_1)$ and $r(e_2)$ can be distinguished as soon as the first difference in the codes occurs because once a difference occurs, the lexicographical order of the two vertex codes is fixed regardless of the rest of the codes. In the same way, the sequences in step 1 are generated simultaneously so that as soon as a difference occurs, the higher quality is assigned to the rooted map with the smaller face degree.

If the rooted reference edges $r(e'_1)$ and $r(e'_2)$ are isomorphic, then they have the same quality and $A(M, R, r(e'_1))$ and $A(M, R, r(e'_2))$ as well as their rooted reference edges are isomorphic, too. Since we construct only one representative M' of the isomorphism class of $A(M, R, r(e'_1))$ and $A(M, R, r(e'_2))$, and since we take only one representative of each isomorphism class of rooted edges in M', any of the rooted edges in M having the highest possible quality is canonical.

4.3 Look–aheads

When a generating rule R shall be applied to a map M using the reference edge $r(e)$, then of course, in general it cannot be seen in advance whether R is the canonical generating rule for $D(M, R, r(e))$ because in order to answer this question, it is necessary to determine all reducible submaps in $D(M, R, r(e))$. However, the insertion of a new submap in M modifies only a few of the existing submaps. Hence it is sometimes possible to conclude in advance that R cannot be the canonical generating rule in $D(M, R, r(e))$ because there will remain at least one *reducible* submap having a higher quality than the new one. We call tests on M that tell us in advance if $D(M, R, r(e))$ will be rejected **look–aheads**.

There exist two different types of look–aheads: A **local look–ahead** finds out whether $D(M, R, r(e))$ will be rejected for a given rooted reference edge $r(e)$. A **global look–ahead** finds out whether $D(M, R, r(e))$ will be rejected for *any* suitable rooted reference edge $r(e)$ in M. Thus, a global look–ahead finds out if R needs to be applied to M at all. Look–aheads help to increase the so–called **accept–reject efficiency** given by the number of accepted maps divided by the number of constructed maps.

The conditions for a generating rule R to be applied to M (using the rooted reference edge $r(e)$) are listed in Table 4.1 where $\mathcal{S}_i$ is the set of all reducible (maximal) submaps S_i in M and s_i is the cardinality of $\mathcal{S}_i$. The edge labels listed in the column "local look–ahead" refer to Figures 3.3-3.12. In order to apply the global look–aheads to a map M, first all reducible submaps in M are counted. A local look–ahead is applied whenever M passes the corresponding global look–ahead.

In detail: The look–aheads permit the application of a generating rule R_i if it is possible that all (maximal) submaps in M having a higher quality than S_i are either destroyed or become irreducible in $D(M, R_i, r(e))$, and no new reducible submap having a higher quality than S_i occurs in $D(M, R_i, r(e))$. If this is not possible, then after the application of R_i, $D(M, R_i, r(e))$ will contain a reducible submap S having a higher quality than S_i, and therefore S_i will not be the canonical submap in $D(M, R_i, r(e))$. In this case, the look–aheads prevent the application of R_i. In the following description, let $M' = D(M, R_i, r(e))$.

R_1: S_1 is the submap with the highest quality and therefore R_1 must always be applied to every possible rooted reference edge.

R_2:
- Since the application of R_2 does not decrease the length of any cycle, no submap S_1 in M will become non–maximal in M'. The application of R_2 modifies at most 2 faces (see Figure 3.4): The face $F_1 = \langle \ldots, a, b, \ldots \rangle$ and the face $F_2 = \langle \ldots, c, d, \ldots \rangle$ (these

R	global look–aheads	local look–aheads
R_1	—	—
R_2	$s_1 \leq 2$	1) $\forall S \in \mathcal{S}_1$: S contains both a and b or both c and d 2) e is not an element of a triangle 3) $\forall S \in \mathcal{S}_4 \cup \mathcal{S}_5 \cup \mathcal{S}_9$: no triangle contained in S contains both a and b or both c and d
R_3	1) $s_1 = 0$ 2) $s_2 \leq 2$	1) $\forall S \in \mathcal{S}_2$: S contains both a and b or both c and f 2) the face $\langle a, e, m, c \ldots \rangle$ is neither a square nor a pentagon 3) $\forall S \in \mathcal{S}_4 \cup \mathcal{S}_5 \cup \mathcal{S}_9$: no triangle contained in S contains both a and b or both c and f
R_4	1) $s_1 \leq 1$ 2) $s_1 + s_2 + s_3 \leq 2$	$\forall S \in \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$: S contains e
R_5	$(\sum_{i=1}^{4} s_i \leq 2$ and $s_3 > 0)$ or $\sum_{i=1}^{4} s_i \leq 1$	$\forall S \in \bigcup_{i=1}^{4} \mathcal{S}_i$: S contains e
R_6	1) $s_1 = 0$ 2) $s_4 = 0$ 3) $s_2 + s_3 \leq 1$ 4) $s_2 + s_3 + s_5 \leq 2$	1) if $s_2 + s_3 = 1$, then the reducible S_2 or S_3 contains a and b 2) $\forall S \in \mathcal{S}_5$: e, a, or b (see Figure 3.8) is the face–bridge in S
R_7	1) $\sum_{i=1}^{4} s_i = 0$ 2) $s_5 + s_6 \leq 1$	1) if $s_5 = 1$, then e is the face–bridge in that reducible S_5 2) if $s_6 = 1$, then e is contained in that reducible S_6
R_8	1) $\sum_{i=1}^{4} s_i = 0$ 2) $s_5 + s_6 \leq 1$ 3) $s_7 = 0$	1) if $s_5 = 1$, then e is the face–bridge in that reducible S_5 2) if $s_6 = 1$, then e is contained in that reducible S_6
R_9	1) $\sum_{i=1}^{4} s_i = 0$ 2) $s_5 + s_6 \leq 1$ 3) $s_7 + s_8 = 0$	1) if $s_5 = 1$, then e is the face–bridge in that reducible S_5 2) if $s_6 = 1$, then e is contained in that reducible S_6

Table 4.1: Look–aheads for M before applying $R_1 - R_9$

faces might be identical). Thus, a triangle F in M will be destroyed by the application of R_2 if and only if $F \in \{F_1, F_2\}$. Hence it must hold $s_1 \leq 2$. If M contains an isolated triangle $F \notin \{F_1, F_2\}$ (i.e. F contains neither both a and b nor both c and d), then after the application of R_2, this isolated triangle will remain isolated in M' and the corresponding submap S_1 will have a higher quality than the inserted submap S_2.

- Moreover, if e (in Figure 3.4) is incident with a triangle in M, then after the application of R_2, this triangle will be isolated in M' so that M' will contain an S_1 (having a higher quality than the inserted S_2): Let w.l.o.g. $\langle a, d, e\rangle$ be a triangle in M. See Figure 4.5. If $\langle a, d, e\rangle$ is already isolated in M, then we are done. So let $\langle a, d, e\rangle$ be incident with another 3–cycle p.
  - If d is an element of p, then b cannot be an element of p because then $p = \langle d, e, b\rangle$ or $p = \langle d, b, e\rangle$ so that we would have a double edge $\{a, b\}$. Hence c is also an element of p. In this case, p in M is replaced by a 4–cycle in M'.
  - If a instead of d is an element of p, then the same argument can be applied.
  - If neither a nor d is an element of p, then $p = \langle b, e, c\rangle$ or $p = \langle b, c, e\rangle$. p will be separated from $\langle a, d, e\rangle$ by the application of R_2 so that the triangle $\langle a, d, e\rangle$ will become isolated in M'.
- Furthermore, if M contains a twin triangle like in S_4, S_5, or S_9 with one triangle being F_1 or F_2 (defined as above), then the other triangle becomes isolated in M' so that an S_1 is obtained. See Figure 4.6. This holds also if the S_4, S_5, or S_9 is irreducible.

R_3:
- First, we prove that by applying R_3, no S_2 becomes irreducible and no isolated triangle becomes incident with another 3–cycle. By the application of R_3, one new 3–cycle might occur in M', namely if a and c (Figure 3.5) are adjacent in M. In this case, $\langle a, e', c\rangle$ is the new 3–cycle in M'.
  - If $\langle a, e', c\rangle$ is not a triangle, then the 3–cycle is incident neither with a triangle nor with a square: See Figure 4.7. l is a face–bridge, and hence the degree of the face $\langle l, a, e', c, l, \ldots\rangle$ is at least 8. The degree of the face $\langle b, h, a, c, g, \ldots\rangle$ is at least 5.
  - If $\langle a, e', c\rangle$ is a triangle, then $\langle a, e, m, c\rangle$ (Figure 3.5) is a square in M. Here the local look–ahead 2 can be applied (see below), and hence it does not matter whether an S_1 becomes non–maximal or an S_2 becomes irreducible in M'.
- Similar to R_2, a triangle F in M will be destroyed by the application of R_3 if $F \in \{F_1, F_2\}$ where $F_1 = \langle b, a, \ldots\rangle$ and $F_2 = \langle c, f, \ldots\rangle$ in Figure 3.5. But if F_1 or F_2 is an isolated triangle, then it will be replaced by a reducible submap S_2 in M'. Hence the existence of a submap S_1 in M causes the application of R_3 always to be non–canonical, and thus we demand $s_1 = 0$.
- In the same way as for triangles, a square F will be destroyed if $F \in \{F_1, F_2\}$. If $\langle a, e, m, c\rangle$ is a square in M, then it will also be destroyed, but it will be replaced in M' by the triangle $\langle a, e', c\rangle$ which is isolated in this case: It cannot be incident with another 3–cycle p because then $p \in \{\langle a, b, h\rangle, \langle a, h, b\rangle, \langle c, f, g\rangle, \langle c, g, f\rangle\}$, but this is impossible because then $\{a, b\}$ or $\{c, f\}$ would be a double edge in M. So if $\langle a, e, m, c\rangle$ is a square in M, then we obtain an S_1 in M' so that the application of R_3 is not canonical. Hence we demand $s_2 \leq 2$, and the (at most) two corresponding submaps S_2 in M must contain both a and b or both c and f, respectively.
- If $\langle a, e, m, c, \ldots\rangle$ is a pentagon in M, then it will be replaced in M' by an S_2 which is reducible using e' as the reference edge (rooted such that the square is the root face).
- Furthermore, if M contains a twin triangle like in S_4, S_5, or S_9 with one triangle being F_1 or F_2 (defined as above), then the other triangle becomes isolated in M' so that an S_1 is obtained. See Figure 4.6. This holds also if the S_4, S_5, or S_9 is irreducible.

R_4: If e (Figure 3.6) is contained in an S_1 (isolated triangle), S_2 (square), or S_3 (pentagon), then the corresponding face will be destroyed and replaced in M' by a face of degree at least 6. Faces that are not incident with e will not be modified. Thus, the local look–ahead follows, and since e is incident with at most 2 faces, we demand $s_1 + s_2 + s_3 \leq 2$. If e is incident with 2 triangles, then we have a twin triangle that is a part of an S_4, S_5, or S_9. Thus e cannot be incident with two isolated triangles and hence we demand $s_1 \leq 1$.

R_5: By applying R_5 to $r(e)$ (Figure 3.7), the degrees of the faces that are incident with e are increased by 7 (root face) and 1 (opposite face). Both faces might be identical. No other faces are modified. Hence we demand $\sum_{i=1}^{4} s_i \leq 2$, and e must be contained in every reducible $S_1 - S_4$. But there are further restrictions:

- If the opposite face in $(M, r(e))$ is an isolated triangle, then in M', it is replaced by a square contained in a reducible S_2.
- If the opposite face in $(M, r(e))$ is a square contained in a reducible S_2, then in M', it is replaced by a pentagon contained in a reducible S_3: Look at Figure 4.8. We know by step 1 of the proof of Theorem 3.1 that the S_3 in M' is reducible unless it holds $a = f$ and $b = c$. But in this case, the S_2 in M is not maximal (and hence not reducible) but contained in an irreducible S_8.
- If the opposite face in $(M, r(e))$ is a triangle which is contained in a reducible S_4, then the root face cannot be contained in an S_1, S_2, S_3, or in another S_4:
  - If e in Figure 3.7 corresponds to e' in Figure 3.6, then both triangles incident with e are contained in the same S_4.
  - If e in Figure 3.7 corresponds to a, b, c, or d in Figure 3.6, then the degree of the root face is at least 6 because the considered S_4 is reducible.

Hence by applying R_5, the only possibility to destroy two reducible submaps $S_1 - S_4$ in M is given by having a pentagon (contained in an S_3) as the opposite face. Hence if $\sum_{i=1}^{4} s_i = 2$, then we demand $s_3 > 0$.

R_6: Edge e (Figure 3.8) is a face–bridge. Hence the degree of the face F incident with e is at least 8. The only further face that will be modified by the application of R_6 is the face $G = \langle b, a, \ldots \rangle$ (where possibly $F = G$). Hence we demand $\sum_{i=1}^{4} s_i \leq 1$, and if M contains a reducible S_2 or S_3, then this S_2 or S_3 must contain G. In this case, it holds $\deg(G) \leq 5$ and thus $F \neq G$. So the local look–ahead 1 is unique (because F is neither a square nor a pentagon).

- If G is a triangle in M, then in M', this triangle is replaced by a pentagon contained in a reducible S_3: See Figure 4.9. It holds $e' \neq g$ and $e' \neq h$ because otherwise there would be a double edge $\{e, a\}$ or $\{e, b\}$, respectively, in M. Moreover, c is adjacent neither to g nor to h. Hence the S_3 is reducible using k as the reference edge (rooted such that the pentagon is the root face). Hence we demand $s_1 = 0$ and $s_4 = 0$. Together with $\sum_{i=1}^{4} s_i \leq 1$, we have $s_2 + s_3 \leq 1$.
- A submap S_5 contains exactly one face–bridge, namely the edge e' in Figure 3.7. If one of the edges e, a, or b in M (Figure 3.8) is the face–bridge of an S_5 and the remaining two edges correspond to a and b in Figure 3.7, then the S_5 becomes irreducible in M'. However, at most two of the three edges e, a, and b in M can each be the face–bridge in an S_5 because otherwise M would be a planar map as shown in Figure 4.10. Hence we demand $s_5 \leq 2$. If $s_5 = 2$, then G cannot be a square of a pentagon. Thus we demand that $s_5 = 2$ implies $s_2 + s_3 = 0$. On the other hand, if G is a square or a pentagon in M, then a and b in M (Figure 3.8) are no face–bridges, in particular they are no face–bridges in an S_5. Hence we demand that $s_2 + s_3 = 1$ implies $s_5 \leq 1$, and thus altogether we demand $s_2 + s_3 + s_5 \leq 2$.

R_7: Edge e (Figure 3.9) is a face–bridge. Hence the degree of the face F incident with e is at least 8. Moreover, F is the only face that will be modified. Hence we demand $\sum_{i=1}^{4} s_i = 0$. If e is the face–bridge in an S_5, then this S_5 will become irreducible in M'. If e was the face–bridge in two different S_5, then M would be a planar map of order 10 as shown in Figure 4.11. Hence we demand $s_5 \leq 1$. If e is one of the edges c, d, or f in Figure 3.8, then the corresponding S_6 will be destroyed in M', but then e cannot be the face–bridge in an S_5. Hence we demand $s_5 + s_6 \leq 1$. The local look–aheads follow immediately from the arguments.

R_8: The arguments are the same as for R_7. An S_7 in M will be destroyed only if e in Figure 3.10 corresponds to c in Figure 3.9, but then c is a face–bridge and the S_7 is irreducible in M. Hence we demand $s_7 = 0$.

R_9: The arguments are the same as for R_7 and R_8.

R_{10}: No look–aheads are considered for R_{10}. In Section 4.4, it will be explained why.

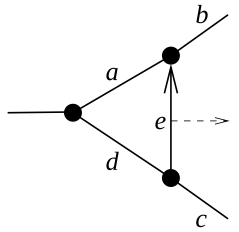

Figure 4.5: before the application of R_2

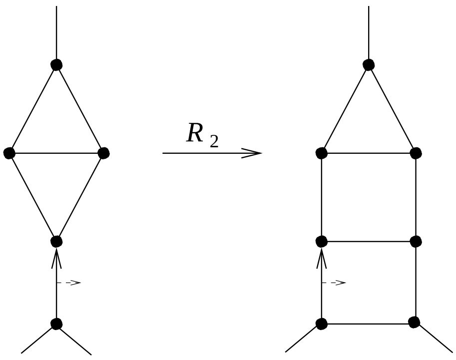

Figure 4.6

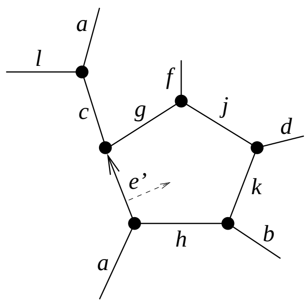

Figure 4.7: after the application of R_3

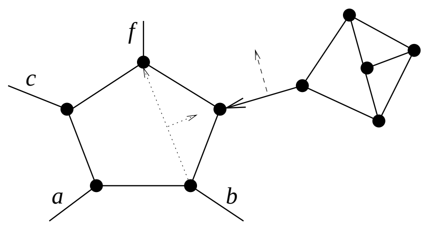

Figure 4.8: after the application of R_5

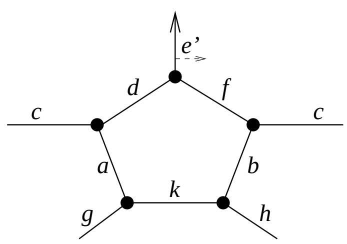

Figure 4.9: after the application of R_6

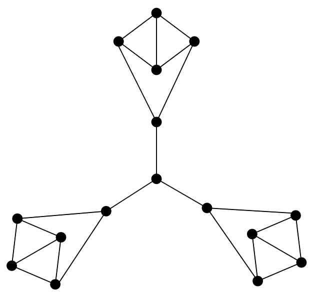

Figure 4.10

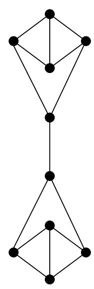

Figure 4.11

4.4 Remarks on the algorithm

The order of $R_1 - R_9$ is not chosen arbitrarily: Some statistics on the maps in $\mathcal{T}$ show that the average number of submaps S_i in a map $M \in \mathcal{T}$ increases with decreasing i. By assigning high qualities to those generating rules whose corresponding submaps are most likely to occur in a map, the look–aheads become most effective: Generating rules with low qualities often need not be applied due to look–aheads that find reducible submaps of higher qualities.

Let $\mathcal{H}_n$ be the set of all hexagonal lattices with up to n vertices. S_{10} has the lowest quality for the following reason: By assigning the quality 0 to S_{10}, the application of R_{10} is canonical only if the resulting map is a hexagonal lattice because every other map $M \in \mathcal{T} - (\mathcal{H} \cup \mathcal{B})$ contains a reducible submap $S_1 - S_9$. Since the cardinality of $\mathcal{H}_n$ is very small in comparison to the cardinality of $\mathcal{T}_n$ (see Table 4.3), it is much faster to omit R_{10} and construct the hexagonal lattices directly. Hexagonal lattices are very easy to classify and hence to construct. In part II of this text, an algorithm is described that can be used to construct hexagonal lattices. Thus *togen* invokes *CGF* in order to obtain $\mathcal{H}_n$. So in the actual implementation of the algorithm, we use only $R_1 - R_9$ as generating rules, and as basic maps we use $\mathcal{B} = \{K_4', K_4'', B_1, B_2, B_3, B_4, B_5, B_6\} \cup \mathcal{H}_n$ for the construction of $\mathcal{T}_n$.

The algorithm can also be used to construct simple cubic *planar* maps. In this case, we have the following changes:

- The only basic map is a planar embedding of K_4.
- In Table 3.1, case 2 can be rejected due to the Jordan curve theorem. For the same reason, some of the conditions in case 1 need not be verified because they cannot hold in planar maps. Thus we obtain Table 4.2.
- The look–aheads for $R_6 - R_9$ must be modified as indicated in the explanation of Table 4.1, i.e. the upper limit for s_5 must be increased by 1.

Submap	Conditions for irreducibility
S_1	—
S_2	a and b are adjacent or c and d are adjacent
S_3	a and b are adjacent or c and f are adjacent or $a = d$ or $c = d$
S_4	f and g are incident with a pentagon
S_5	a and b are elements of a 3–cycle
S_6	—
S_7	—
S_8	—
S_9	—
S_{10}	k and a are adjacent or h and d are adjacent or $h = c$ or $k = b$

Table 4.2: How to detect irreducible submaps in simple cubic planar maps

4.5 Parallelization and recovering

The construction of $\mathcal{T}_n$ can be parallelized. The technique is described in detail in [45], Section 8. Parallelization means that a process is substituted by a set of subprocesses. The idea is as follows: Since the construction can be divided into a clean forest structure (compare Figure 4.3), several subprocesses $p_0, \ldots, p_{k-1}$, $k > 0$, can be invoked where every subprocess constructs $\mathcal{T}_m$ for a given vertex number $m < n$. The number m is a threshold in the sense that as soon as a map M having more than m vertices is constructed in a canonical way out of a map having at most m vertices, only every k–th map will be accepted (so only every k–th map that would be accepted in the original process is accepted in the subprocess). Let all maps that match the description of M (order $> m$, order of the canonical ancestor $\leq m$) be numbered beginning with 0. Then the subprocess p_i accepts every map having the number i modulo k. Thus disjoint subtrees in the construction forest will be traced by the subprocesses, and each subtree will be traced exactly once, with one exception: If there are basic maps containing more than m vertices, then these basic maps and all descendants would occur in every subprocess. Hence we arrange that basic maps containing more than m vertices are accepted only in p_0. Altogether, the construction is parallelized in a way that the subprocesses do not even need to interact.

A further property of the construction algorithm is the fact that the construction of $\mathcal{T}_n$ can be recovered, i.e. if *togen* is interrupted (e.g. when the computer is turned off by mistake), then not the whole construction needs to be redone from the start. Since the construction can be divided into a clean forest structure, the targets of the branches can uniquely be labelled. Targets of branches having the same source are labelled beginning with 1. Figure 4.12 shows the labelling for the example in Figure 4.3. Now the position of any node v in the construction forest can uniquely be described by the appropriate tree–source r in the search forest and the sequence of labels that occur while tracing the tree–path with start node r and end node v. Hence in order to be able to recover an interrupted process, it suffices to store the number of the current tree–source and the sequence of labels representing the tree–path to the current position from time to time. Then after a restart, *togen* can continue at this position. This is done in the following way: Let t be the number of the stored tree–source. Then the tree–sources with the numbers $1, \ldots, t-1$ are skipped. Now starting with the lowest level, the levels are traced. Let t be the stored number in level l. Then the maps represented by the targets $1, \ldots, t-1$ are constructed, but they are not accepted. Thus target number t will be approached very quickly.

The amount of information that has to be stored is quite small. In *togen*, every 30 minutes a so–called *dump file* containing about 600 bytes is stored. Thus, the loss of time after a restart is at most 30 minutes, namely the elapsed time between the last update of the dump file and the interruption of *togen*. If maps have been constructed and stored during this period of time, then these maps will be constructed and stored again. If this is not desired, then the dump file must be updated each time maps are stored. To this end, in order to decrease the number of updates, some accepted maps should be collected before they are stored.

The recovering can be combined with the parallelization very easily. Whenever a dump file is written, the number of maps that have already been constructed and that match the above definition of M must be stored. After the recovering has finished, this number is again valid. During the recovering, the subprocess number does not need to be considered at all because clearly, the recovering guides the subprocess to a node in the construction forest that has also been approached by the subprocess *before* the interruption.

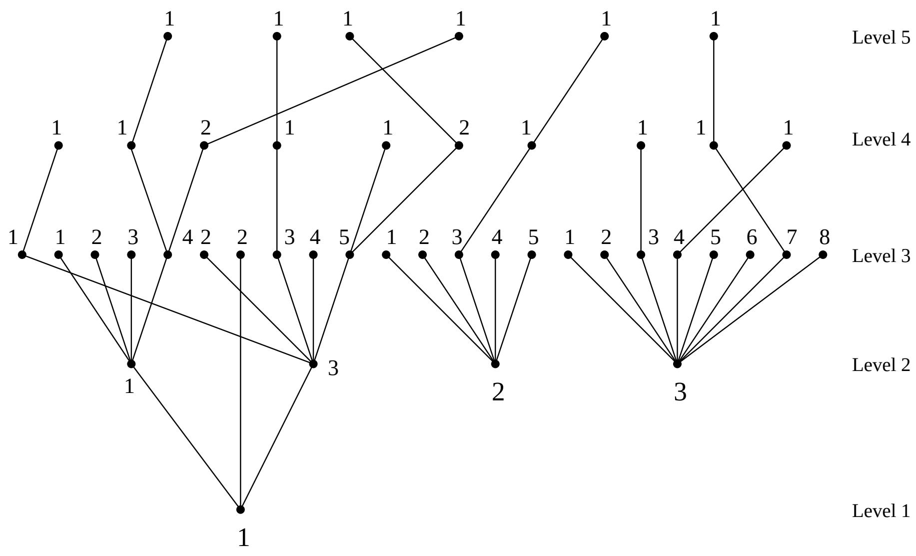

Figure 4.12

4.6 Results

The results in Table 4.3 have been computed on a large cluster of computers using the parallelizing technique described above and in [45]. The **generation rate** of *togen*, i.e. the number of constructed maps within a given amount of time, is approximately 14000 structures per second for the toroidal case and approximately 9400 structures per second for the planar case (computed on an Intel Pentium II computer with 350 MHz). In Table 4.3, $\mathcal{T}'_n := \mathcal{T}_n - \mathcal{T}_{n-2}$ is the set of all simple cubic toroidal maps of order n, $\mathcal{P}_n$ is the set of all simple cubic planar maps of order at most n, $\mathcal{P}'_n := \mathcal{P}_n - \mathcal{P}_{n-2}$ is the set of all simple cubic planar maps of order (exactly) n, and $A_{\mathcal{M}}$ is the accept–reject efficiency for the construction of $\mathcal{M}$. Owing to the look–aheads, the accept–reject efficiency locally becomes constant or even increases with increasing vertex number (for $\mathcal{T}_n$, the local minimum is given for $n = 14$, for $\mathcal{P}_n$, the expected local minimum is given for $n \approx 32$) although the average number of reducible submaps in a map increases with increasing vertex number.

n	$\lvert\mathcal{H}_n\rvert$	$\lvert\mathcal{T}'_n\rvert$	$A_{\mathcal{T}_n}$	$\lvert\mathcal{P}'_n\rvert$	$A_{\mathcal{P}_n}$
4	0	2	—	1	—
6	1	7	0.375	1	1.000
8	2	37	0.406	3	1.000
10	1	232	0.321	9	0.765
12	2	1 742	0.283	37	0.521
14	2	14 379	0.269	172	0.418
16	4	127 282	0.270	993	0.368
18	3	1 170 626	0.276	6 308	0.337
20	3	11 021 519	0.283	44 145	0.321
22	2	105 245 468		327 051	0.308
24	7	1 014 547 266		2 530 761	0.299
26	3	9 846 232 922		20 179 785	0.290
28	4	96 055 936 574		164 672 106	

Table 4.3

Chapter 5

Cubic toroidal maps with faces of degree ≤ 7

In this chapter, we will give a constructive definition of simple cubic toroidal maps where only triangles, squares, pentagons, hexagons, and heptagons are permitted as faces. The proceeding is quite similar to the proceeding in the last chapters. Hence we use the last chapters as background and add or modify arguments wherever it is necessary. We denote the class of all simple cubic toroidal maps where all faces are of degree ≤ 7 by $\mathcal{U}$ and the class of all maps $M \in \mathcal{U}$ with up to n vertices by $\mathcal{U}_n$.

5.1 The constructive definition of $\mathcal{U}$

5.1.1 Basic maps

As basic maps, we need all hexagonal lattices. This cannot be avoided because by applying ${R_{10}}^-$ to a hexagonal lattice, the resulting map would contain an octagon (in Figure 3.12, this octagon is $\langle h, g, f, e, k, \ldots \rangle$). Since the set of all hexagonal lattices $\mathcal{H}$ is not finite, the following definition is not inductive in a strict sense.

5.1.2 Generating rules

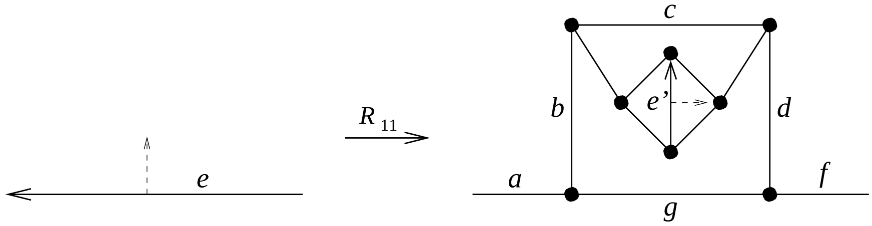

Figure 5.1

We use the generating rules R_1, R_2, R_3, R_4, and R_{11} illustrated in Figures 3.3, 3.4, 3.5, 3.6, and 5.1. It is obvious that the rules $R_5 - R_9$ are obsolete because due to Lemma 2.2, *every map $M \in \mathcal{U}$ contains neither face–bridges nor 3–cycles that do not correspond to triangles*: A 3–cycle that does not correspond to a triangle implies the existence of a face–bridge (see Figure 3.8). So in particular, *no $S_5 - S_9$ can occur in a map $M \in \mathcal{U}$*. This fact will make the proof in Section 5.1.3 much easier. Hence it should be kept in mind throughout the proof.

The restrictions to the application of the generating rules to a map $M \in \mathcal{U}$ are as follows:

- R_3 must not be applied if $c = a$ in M (Figure 3.5). Since 3–cycles always correspond to triangles in M, the restriction can be formulated as follows: R_3 must not be applied if the opposite face is a triangle.
- R_1 must not be applied if b, c, or e is incident with a heptagon because otherwise this heptagon would be replaced by an octagon.
- For the same reason, R_2 must not be applied if the face $\langle a, b, \ldots \rangle$ or the face $\langle c, d, \ldots \rangle$ (see Figure 3.4) is a heptagon. Moreover, R_2 must not be applied if there exists a hexagon $\langle a, b, \ldots, c, d, \ldots \rangle$ because otherwise this hexagon would also be replaced by an octagon.
- For the same reasons, R_3 must not be applied if the face $\langle b, a, \ldots \rangle$ or the face $\langle c, f, \ldots \rangle$ (see Figure 3.5) is a heptagon, or if there exists a hexagon $\langle b, a, \ldots, c, f, \ldots \rangle$.
- If we want to apply R_4, then the degrees of both faces containing e (Figure 3.6) must each be at most 4 because by the application of R_4, their degrees will each be increased by 3.
- If we want to apply R_{11}, then the root face in $(M, r(e))$ must be a triangle and the degree of the opposite face must be at most 5 because otherwise the degrees of the faces would be greater than 7 after the application of R_7.

5.1.3 Main theorem

Theorem 5.1 *Let $\mathcal{B} = \mathcal{H}$. Then the set $\mathcal{M}$ of maps that can be constructed by applying a finite number of generating rules $R_1 - R_4$ and R_{11} to an element of $\mathcal{B}$ considering the above restrictions is exactly $\mathcal{U}$.*

Proof: $\mathcal{M} \subseteq \mathcal{U}$ holds because if the above restrictions are considered, then applying a generating rule to a map $M \in \mathcal{U}$ preserves simplicity, connectivity, cubicity, and genus, and after the application of a generating rule the resulting map contains only triangles, squares, pentagons, hexagons, and heptagons.

- The generating rules $R_1 - R_4$ and R_{11} are again not redundant.
- S_1 might be contained in S_4 and S_{11}, S_3 and S_4 might be contained in S_{11}.
- Table 5.1 shows all conditions for a *maximal* submap to be irreducible and thus all restrictions to the application of the inverse of the corresponding generating rule. All restrictions are given in order to avoid loops, double edges, and octagons (if an S_3 shall be reduced). There is no reduction such that the resulting map is planar. See Figures 3.3-3.6 and 5.1.
- Unlike for $\mathcal{T}$, we will not only reduce *maximal* submaps. There is one exception where a submap S_3 is reduced although it is not maximal. See step 2 of the proof. In all other cases, we consider only maximal submaps.

Remarks to table 5.1:

- The condition for S_2 in could be written as "a and b are adjacent or c and d are adjacent".

Submap	Constellations for an irreducible submap
S_1	—
S_2	$\langle a, h, b\rangle$ or $\langle c, g, d\rangle$ is a triangle
S_3	$\langle a, b, h\rangle$ or $\langle c, g, f\rangle$ is a triangle or $\langle a, e', c, \ldots\rangle$ is a heptagon
S_4	—
S_{11}	$\langle a, b, c, d, f, \ldots\rangle$ is a hexagon or $\langle f, g, a, \ldots\rangle$ is a square

Table 5.1: How to detect irreducible submaps

However, if a and b are adjacent in S_2, then we automatically have a triangle $\langle a, h, b\rangle$ because — as stated above — in every map $M \in \mathcal{U}$, every 3–cycle corresponds to a triangle. The same holds for c and d in S_2 as well as for a and b or for c and f in S_3.

- In S_3, unlike in Section 3.3, it holds neither $a = d$ nor $c = d$ nor both $a = f$ and $b = c$: If $a = d$, then b is a face–bridge, if $c = d$, then f is a face–bridge, and if $a = f$, then c is a face–bridge.
- In S_4, it holds $f \neq g$ (see Figure 3.6) because otherwise we would have a planar embedding of K_4. Moreover, f and g are not adjacent because otherwise we would have an S_5 and thus a face–bridge which is impossible. If f and g are incident with a pentagon (say the face $F_1 = \langle a, b, f, \ldots, g\rangle$ is a pentagon), then the S_4 is contained in an S_{11}: Since $f \neq g$ and f and g are not adjacent, the face $F_2 = \langle f, c, d, g, \ldots\rangle$ is neither a triangle nor a square. Moreover, F_2 is no pentagon because otherwise we would have a double edge (whose elements are adjacent to f and g), and F_2 is no hexagon because otherwise we would have a face–bridge (see Figure 5.2). Hence F_2 is a heptagon, and thus the S_4 in contained in an S_{11}. So if the S_4 is maximal, then $\deg(F_1) \geq 6$ and $\deg(F_2) \geq 6$. Thus, there are no restrictions to the application of ${S_4}^-$.
- In S_{11}, it holds $a \neq f$ (see Figure 5.1) because otherwise we would have a double edge $\{a, g\}$. Hence $\langle a, b, c, d\rangle$ is no square. Moreover, a and f are not adjacent because otherwise we would have a face–bridge similar to the face–bridge in Figure 5.2. Hence $\langle f, g, a\rangle$ is no triangle, and $\langle a, b, c, d, f\rangle$ is no pentagon. Thus the list of restrictions for S_{11} in Table 5.1 is complete.

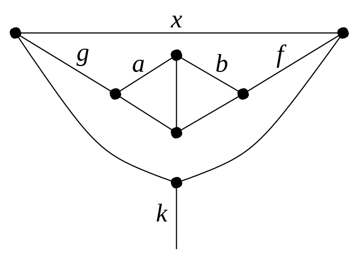

Figure 5.2: k is a face–bridge

Now we show that every map $M \in \mathcal{U} - \mathcal{H}$ contains at least one reducible submap $S_1 - S_4$ or S_{11}. Formula 1.3 on page 9 implies that each map $M \in \mathcal{U} - \mathcal{H}$ contains at least one face of degree at most 5.

Step 1: First, we prove that a square in a map $M \in \mathcal{U}$ is always reducible. This is quite easy: Look at Figure 3.4. If a and b are adjacent in S_2 so that $\langle a, h, b\rangle$ is a triangle, then neither

$\langle a, d, e' \rangle$ nor $\langle c, b, f \rangle$ can be a triangle because otherwise M would contain an S_5. Hence $R_2{}^-$ can be applied using g or h as the reference edge (rooted such that $\langle e', g, f, h \rangle$ is the root face).

Step 2: Let $M \in \mathcal{U} - \mathcal{H}$ be a map that contains no squares but at least one triangle. We prove that M contains also a reducible submap S_1, S_3, or S_{11}.

- If the triangle is not incident with another triangle, then it is isolated and can be reduced applying $R_1{}^-$.
- Let the triangle be incident with another triangle. Thus M contains a twin triangle. As stated above, the twin triangle cannot be incident with another triangle because otherwise M would be a planar embedding of K_4.
  - If f and g (see Figure 3.6) are not incident with a pentagon, then the twin triangle can be reduced applying $R_4{}^-$.
  - If f and g are incident with a pentagon, then — as stated above — the other face which is incident with f and g must be a heptagon and hence M contains an S_{11}. Look at Figure 5.1. The face $\langle f, g, a, \ldots \rangle$ is no square because M does not contain squares. Moreover, as already stated, $\langle a, b, c, d, f \rangle$ cannot be a pentagon. If the face $F = \langle a, b, c, d, f, \ldots \rangle$ is a heptagon, then $R_{11}{}^-$ can be applied. If F is a hexagon, then the pentagon contained in S_{11} is reducible using c as the reference edge for the application of $R_3{}^-$ (where c is rooted such that the pentagon is the root face). In this case, the hexagon is replaced by a heptagon.

Step 3: Let $M \in \mathcal{U} - \mathcal{H}$ be a map that contains neither squares nor triangles. Hence M contains at least one pentagon. Look at Figure 3.5. Since M does not contain triangles, neither $\langle a, b, h \rangle$ nor $\langle c, g, f \rangle$ is a triangle. This holds for every pentagon in M. If we can find a pentagon P in M having a common element k with a face that is *not* a heptagon, then due to Table 5.1, P can be reduced applying $R_3{}^-$ where k is the reference edge (rooted such that P is the root face).

Thus, it remains to be shown that M contains a pentagon which is incident not only with heptagons. Formula 1.3 on page 9 yields $1p_5(M) + 0p_6(M) + (-1)p_7(M) = 0$, and hence $p_5(M) = p_7(M)$. Let x be the number of edges in M that are each incident with one pentagon and one heptagon. If every pentagon is incident only with heptagons, then $x = 5 \cdot p_5(M)$ (each pentagon contributes 5 edges), but on the other hand, every heptagon can be incident with at most 3 different pentagons because otherwise two pentagons would be incident with each other. So $x \leq 3 \cdot p_7(M)$. With $p_5(M) = p_7(M)$, we obtain $3 \cdot p_7(M) \geq x = 5 \cdot p_5(M) = 5 \cdot p_7(M)$ which is false for $p_7(M) \geq 1$. Hence the assumption that every pentagon is incident only with heptagons is false.

This completes the proof of Theorem 5.1. ■

5.2 Fast construction of $\mathcal{U}_n$

The principles for the fast construction of $\mathcal{U}_n$ for a given n are equal to those described in Chapter 4. To S_{11}, the lowest quality is assigned among all submaps.

5.2.1 Look–aheads

The conditions for a generating rule R_i to be applied to M (with respect to the rooted reference edge $r(e)$) are listed in Table 5.2 where $\mathcal{S}_i$ is the set of all reducible submaps S_i in M and s_i is the cardinality of $\mathcal{S}_i$. The edge labels listed in column "local look–ahead" refer to Figures 3.3-3.6 and 5.1. In the following description, let $M' = D(M, R_i, r(e))$.

R	global look–aheads	local look–aheads
R_1	—	—
R_2	$s_1 \leq 2$	1) $\forall S \in \mathcal{S}_1$: S contains both a and b or both c and d 2) e is not an element of a triangle 3) $\forall S \in \mathcal{S}_4 \cup \mathcal{S}_{11}$: no triangle contained in S contains both a and b or both c and d
R_3	1) $s_1 = 0$ 2) $s_2 \leq 2$	1) $\forall S \in \mathcal{S}_2$: S contains both a and b or both c and f 2) the face $\langle a, e, m, c, \ldots \rangle$ is neither a square nor a pentagon 3) $\forall S \in \mathcal{S}_4 \cup \mathcal{S}_{11}$: no triangle contained in S contains both a and b or both c and f
R_4	1) $s_1 \leq 1$ 2) $s_1 + s_2 \leq 2$ 3) $s_3 \leq s_2$	1) $\forall S \in \mathcal{S}_1 \cup \mathcal{S}_2$: S contains e 2) $\forall S \in \mathcal{S}_3$: S is incident with a square containing e
R_{11}	1) $s_4 = 1$ and 2) $s_1 + s_2 + s_3 = 0$ or 1) $s_4 = 0$, 2) $s_1 = 1$, and either 3a) $s_2 = 1$, 4a) $s_3 = 0$, or 3b) $s_2 = 0$, 4b) $s_3 \leq 3$	1) if $s_4 = 1$, then the S_4 contains the root face 2) if $s_2 = 1$, then the S_2 contains the opposite face 3) if $s_3 > 0$, then the opposite face is contained in a reducible S_3 4) $\forall S \in \mathcal{S}_3$: S contains the opposite face or a face that is incident with the opposite face 5) e is not an element of a triangle contained in an S_{11}

Table 5.2: Look–aheads for $(M, r(e))$ before applying $R_1 - R_4$ or R_{11}

In detail:

R_1: R_1 must always be applied to every possible rooted reference edge.

R_2: See Section 4.3.

R_3: See Section 4.3.

R_4:
- As usual, the application of R_4 destroys only the two incident faces of e, and hence we demand $s_1 + s_2 \leq 2$ and the local look–ahead 1. Moreover, we demand $s_1 \leq 1$ as explained in Section 4.3.
- Since e must not be incident with a pentagon (see Section 5.1.2), no pentagon in M will be modified. However, it is possible that a reducible S_3 in M becomes irreducible in M'. If an S_3 is reducible with respect to a reference edge x, then it might become irreducible if the opposite face in $(M, r(x))$ is replaced by a heptagon (to be precise:

the S_3 becomes irreducible if firstly, the opposite face is replaced by a heptagon and secondly, due to the restrictions in Table 5.1, the S_3 contains no other edge than x that may be used as the reference edge). Hence if e is incident with a square and the square is incident with a pentagon contained in an S_3, then the S_3 might become irreducible because the square in M will be replaced by a heptagon in M'. Since this is the only possibility for a reducible S_3 in M to become irreducible in M', the local look–ahead 2 follows. Finally, we demand $s_3 \leq s_2$:

- Assume that e is incident with two triangles. Then we demand $s_2 = 0$ because an S_2 that is not incident with e will remain unmodified and thus reducible in M'. The two triangles in M incident with e will be replaced by hexagons in M'. No S_3 becomes irreducible by this replacement because no restriction to the application of ${R_3}^-$ (see Table 5.1) refers to hexagons. Thus we demand that $s_2 = 0$ implies $s_3 = 0$.
- Assume that e is incident with one square and one triangle as shown in Figure 5.3. Then we demand $s_2 \leq 1$ because an S_2 that is not incident with e will remain unmodified and thus reducible in M'. If a in Figure 5.3 is incident with a reducible S_3, then this S_3 will remain reducible in M': d will be incident in M' with a hexagon (and not with a heptagon), and since it holds $h \neq j$ (otherwise g would be a face–bridge), neither the face $\langle h, f, b, j, \ldots\rangle$ nor the square $\langle a, c, b, e\rangle$ is a triangle. Hence the conditions in Table 5.1 do not hold if d is the reference edge for the application of ${R_3}^-$. The same argumentation can be applied if b instead of a is incident with a reducible S_3. Hence only an S_3 whose pentagon is incident with c might become irreducible. Hence we demand that $s_2 \leq 1$ implies $s_3 \leq 1$.
- Assume that $s_2 = 2$, i.e. e is incident with two squares. Let two adjacent edges of one square be incident with one pentagon each. In this case, the pentagons are incident with each other, and after the application of R_4, the situation is as shown in Figure 5.4.
  - If $a = h$ (so that $\langle a, d, k\rangle$ is a triangle), then the pentagon $\langle b, j, d, g, f\rangle$ can be reduced applying ${R_3}^-$ using d as the reference edge because d is not incident with a heptagon and neither g nor j are incident with a triangle.
  - If $a \neq h$, then the pentagon $\langle b, j, d, g, f\rangle$ can be reduced applying ${R_3}^-$ using j as the reference edge because j is not incident with a heptagon and neither b nor d is incident with a triangle.

  Hence if there are two pentagons incident with each other and incident with one common square in M, then one of the corresponding S_3 is reducible in M'. Since at most 4 pentagons can be incident with the two squares in M, at most two S_3 might become irreducible in M'. Hence we demand that $s_2 \leq 2$ implies $s_3 \leq 2$.

  Remark: In the above situation (illustrated in Figure 5.4), it can be shown that even both S_3 are reducible. But we cannot conclude that $s_2 = 2$ implies $s_3 = 0$ because if the face containing f and b in Figure 5.4 is a heptagon instead of a pentagon, then the S_3 containing the pentagon $\langle k, j, c, \ldots\rangle$ might be irreducible.

R_{11}: As stated in Section 5.1.2, the root face of $(M, r(e))$ must be a triangle and the degree of the opposite face must be at most 5. No further faces in M are modified by the application of R_{11}.

- Let e be an element of a triangle contained in an S_{11}. We show that after the application of R_{11}, M' contains a reducible submap having a higher quality than the (new) S_{11} (so that the local look–ahead 5 follows). For the following argumentation, we consider the right side in Figure 3.6 to show the twin triangle of the S_{11} in M. Thus g and f in Figure 3.6 are incident with a pentagon, and either a or d — say a — is incident with that pentagon, too. So the pentagon is $\langle g, a, b, f, \ldots\rangle$.

- If e in Figure 5.1 corresponds to e' in Figure 3.6, then after the application of R_{11}, either the triangle $\langle a, d, e' \rangle$ or the triangle $\langle e', c, b \rangle$ (depending on the direction and the side of $r(e)$) in M is replaced by a pentagon in M'. See Figure 5.5 for the first case ($\langle a, d, e' \rangle$). Here the new pentagon is $\langle a, d, k, j, h \rangle$. This pentagon is contained in a reducible S_3 using a as the reference edge: a is not incident with a heptagon (but with two pentagons $\langle a, d, k, j, h \rangle$ and $\langle g, a, b, f, \ldots \rangle$), and neither d nor h are incident with a triangle. The second case ($\langle e', c, b \rangle$) is treated analogously.
- If e in Figure 5.1 corresponds to a, b, c, or d in Figure 3.6 (rooted such that the root face is a triangle), then after the application of R_{11}, the situation is as shown in Figure 5.6 (here e in Figure 5.1 corresponds to c in Figure 3.6). The triangle $\langle a, d, h \rangle$ is isolated and thus contained in a reducible S_1.

- Let M contain an S_4. Then we demand that this S_4 contains e (Figure 5.1) because otherwise the S_4 will not be destroyed so that after the application of R_{11}, M' will contain a submap having a higher quality than the new S_{11}. So e in Figure 5.1 corresponds to a, b, c, d, or e' in Figure 3.6. Since the triangles of the S_4 are not incident with a third triangle and since the root face must be a triangle, the root face must be a triangle contained in the S_4. Thus the local look–ahead 1 follows. Moreover, since the triangles of the S_4 are incident neither with a third triangle nor with a square nor with a pentagon, the opposite face is either a hexagon, a heptagon, or the other triangle contained in the S_4. So since the root face and the opposite face are the only faces that are destroyed by the application of R_{11}, at most one S_4 and no S_1, S_2, or S_3 in M will be destroyed. Hence we demand $s_4 \leq 1$, and we demand that $s_4 = 1$ implies $s_1 + s_2 + s_3 = 0$.
- Let now $s_4 = 0$. Then we may presume that e (Figure 5.1) is not an element of a triangle contained in an S_{11} because otherwise, the local look–ahead 5 can be applied. So the root face in $(M, r(e))$ is an isolated triangle, and the opposite face is not a triangle but a square or a pentagon. Hence we demand $s_1 = 1$.
  - Let the opposite face be a square (contained in a reducible S_2). Since no further face in M is modified by the application of R_{11}, we demand $s_2 = 1$ in this case. By applying R_{11}, the square in M is replaced by a hexagon in M'. No S_3 in M becomes irreducible in M': Look at Figure 5.3.
    - If M contains a reducible S_3 that contains neither d nor f, then the pentagon P contained in the S_3 is not incident with the triangle $\langle d, e, f \rangle$. So the square $\langle a, c, b, e \rangle$ is the only remaining face in M that will be modified by the application of R_{11} and that might be incident with P. The square is replaced by a hexagon in M'. So maybe an incident square of P might be replaced by a hexagon. Since the restrictions to the application of $R_3{}^-$ (see Table 5.1) refer neither to hexagons nor to squares, the S_3 remains reducible in M' if it is reducible in M.
    - If M contains a reducible S_3 containing d, then the face $\langle g, a, d, h, x \rangle$ in Figure 5.3 (x is not shown) is the pentagon contained in this S_3. The face $\langle j, c, g, \ldots \rangle$ is no triangle because otherwise the edge which is incident with h and x would be a face–bridge. So in M', the S_3 can be reduced appplying $R_3{}^-$ using a as the reference edge (rooted such that the pentagon is the root face).
    - If M contains a reducible S_3 containing f, then the same argumentation as in the previous subcase can be applied.

    Hence we demand that $s_2 = 1$ implies $s_3 = 0$.
  - Let the opposite face be a pentagon contained in an S_3. Since the root face is an *isolated* triangle, the S_3 is reducible applying $R_3{}^-$ using e as the reference edge (rooted such that the pentagon is the root face). So $s_3 \geq 1$. Since we demand

$s_3 = 0$ in all other cases, the local look–ahead 3 follows. Since no other face except the root face (triangle) and the opposite face (pentagon) is modified by the application of R_{11} and since a square is always contained in a reducible S_2, we demand $s_2 = 0$ in this case. The local look–ahead 2 follows because we demand $s_2 = 0$ in all cases except the case where the opposite face is a square. In the current case, the opposite face (pentagon) in $(M, r(e))$ will be replaced by a heptagon in M'. Moreover, at most two more reducible S_3 whose pentagons are incident with the opposite face might become irreducible. The argumentation is similar to the one given for R_4: Up to four pentagons can be incident with the opposite face in $(M, r(e))$. But if two of them are incident with each other, then at least one of them is reducible in M'. Look at Figure 5.7. After the application of R_{11}, both faces incident with e are replaced by heptagons. The edge j is incident with two pentagons. The edges are labelled in the same way as in Figure 5.4, and hence the argumentation referring to Figure 5.4 can be applied directly. Hence we demand $s_3 \leq 3$ and the local look–ahead 4.

Note that the case where j is incident with two pentagons is the only case that needs to be considered:

- If m or g instead of j (or l) in Figure 5.7 is incident with two pentagons, then it can also be shown that one of the corresponding S_3 is reducible in M', but this is not necessary because two more S_3 (in addition to the destroyed S_3 containing the opposite face) may become irreducible due to the restriction $s_3 \leq 3$.
- If the opposite face in $(M, r(e))$ is incident with 3 pentagons, then two of them have j or l as common elements and the above argumentation can be applied in order to state that one of the 3 corresponding S_3 is reducible in M'.
- If the opposite face in $(M, r(e))$ is incident with 4 pentagons, then both j and l are incident with two pentagons each, and thus the above argumentation can be applied twice in order to exclude 2 corresponding S_3 from being irreducible in M'.

5.2.2 Remarks on the algorithm

The algorithm can also be used to construct simple cubic *planar* maps whose faces are of degree ≤ 7. In this case, we have the following changes:

- The only basic map is a planar embedding of K_4.
- In step 3 of the proof of Theorem 5.1, formula 1.3 yields $p_5(M) = p_7(M) + 12$ instead of $p_5(M) = p_7(M)$. However, for the planar case the proof holds more than ever because $3 \cdot p_7(M) \geq x = 5 \cdot p_5(M) = 5 \cdot (p_7(M) + 12) = 5 \cdot p_7(M) + 60$ is false for $p_7(M) \geq 1$.
- The planar embedding of K_4 contains an *irreducible* (maximal) S_4 because the triangles of the S_4 are incident with a third triangle. This must be considered in the look–aheads. Precisely, for R_{11}, if $M = (G, \Pi)$ contains an S_4, then we cannot conclude $s_4 = 1$ if $G = K_4$. So the global look–ahead for R_{11} must not be applied to $M = (G, \Pi)$ if $G = K_4$.

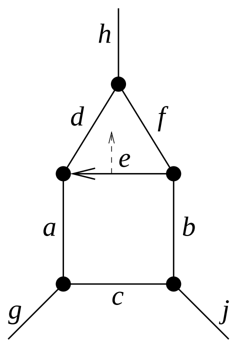

Figure 5.3: before the application of R_4 or R_{11}

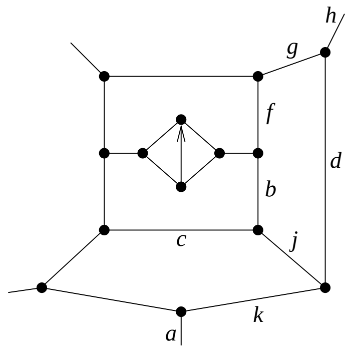

Figure 5.4: after the application of R_4

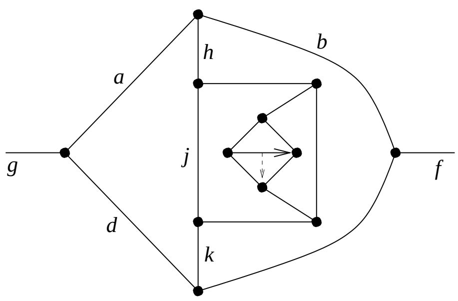

Figure 5.5: after the application of R_{11}

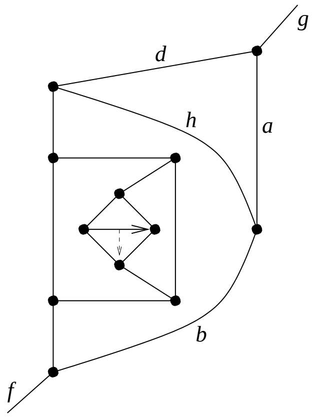

Figure 5.6: after the application of R_{11}

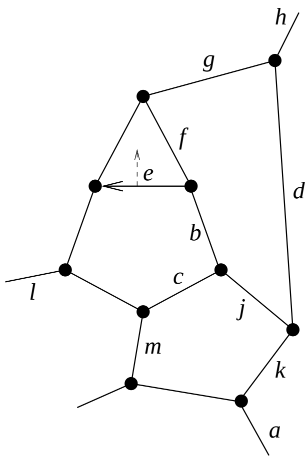

Figure 5.7: before the application of R_{11}

5.3 Results

The generation rate of *togen7* on an Intel Pentium II computer with 350 MHz is approximately 6000 structures per second for the toroidal case and 8000 structures per second for the planar case. In Table 5.3, $\mathcal{U}'_n := \mathcal{U}_n - \mathcal{U}_{n-2}$ is the set of all simple cubic toroidal maps of order n where all faces have a degree ≤ 7, $\mathcal{Q}_n$ is the set of all simple cubic planar maps of order at most n where all faces have a degree ≤ 7, $\mathcal{Q}'_n := \mathcal{Q}_n - \mathcal{Q}_{n-2}$ is the set of all simple cubic planar maps of order (exactly) n where all faces have a degree ≤ 7, and $A_{\mathcal{M}}$ is the accept–reject efficiency for the construction of $\mathcal{M}$.

n	$\lvert\mathcal{H}_n\rvert$	$\lvert\mathcal{U}'_n\rvert$	$A_{\mathcal{U}_n}$	$\lvert\mathcal{Q}'_n\rvert$	$A_{\mathcal{Q}_n}$
4	0	0	—	1	—
6	1	1	—	1	1.000
8	2	4	1.000	3	1.000
10	1	6	0.500	7	0.917
12	2	15	0.313	20	0.660
14	2	33	0.260	50	0.536
16	4	115	0.221	160	0.430
18	3	297	0.187	480	0.356
20	3	997	0.170	1 646	0.321
22	2	3 118	0.154	5 485	0.297
24	7	10 853	0.146	19 191	0.283
26	3	37 618	0.141	67 003	0.273
28	4	133 601	0.139	237 261	0.267
30	5	479 795	0.139	841 043	0.263
32	8	1 742 584		2 995 904	
34	3	6 352 313		10 684 404	
36	7	23 339 438		38 197 129	
38	4	85 938 937		136 701 570	
40	9	317 871 047		490 049 415	
42	7	1 178 635 174			
44	6	4 381 732 192			
46	4	16 327 051 499			

Table 5.3

Part II

The construction of cubic maps with a given set of faces

Chapter 6

Introduction

In this part, we will give a constructive definition of all cubic maps of a given genus, a given vertex number, and a given set of permitted faces. For a given genus, formula 1.2 on page 9 yields a fixed relation between the number of vertices and the number of faces in a cubic map. So instead of the number of vertices, we could as well prescribe the number of faces that must be contained in every desired map. The set of permitted faces may contain more faces than the desired maps will actually contain. In most cases, we prescribe only the set of permitted face *degrees*, and then for every permitted degree, an arbitrary number of faces is permitted in the desired maps.

As in part I, we will describe how the definition was used to develop the computer program *CGF* that constructs all cubic maps satisfying the given prescriptions.

Let $g \in \mathbb{N}_0$, $n \in \mathbb{N}$, $D \subset \mathbb{N}$, and $b_l, b_u : D \to \mathbb{N}_0$. If $D \cap \{1,2\} \neq \emptyset$, then $\mathcal{M}(g, n, D, b_l, b_u)$ denotes the set of all[1] cubic maps of genus g and order n where D is the set of permitted face degrees (so that $D_M \subseteq D$ for every $M \in \mathcal{M}(g, n, D, b_l, b_u)$) and for every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ and every degree $d \in D$ it holds that M contains at least $b_l(d)$ and at most $b_u(d)$ faces of degree d (so $b_l(d) \leq p_d(M) \leq b_u(d)$). If $D \cap \{1,2\} = \emptyset$, then $\mathcal{M}(g, n, D, b_l, b_u)$ denotes the set of all *simple* cubic maps M with $g_M = g$, $n_M = n$, $D_M \subseteq D$, and $b_l(d) \leq p_d(M) \leq b_u(d)$ for every $d \in D$. If no explicit upper limit shall be given for a face degree $d \in D$, then we denote $b_u(d) = \infty$. Of course, in this case there is an implicit upper limit for the number of d–gons in M, namely $f_M = \frac{n}{2} + 2 - 2g$ (see formula 1.2 on page 9). In a cubic map $M = (G, \Pi)$ with $G = (V, H, E, T)$, a loop $e = \{h, h'\}$ with $T(h) = T(h') = (v, v)$ for some $v \in V$ always corresponds to a monogon: v is incident with exactly 3 semi–edges where two of them are h and h'. Let k be the third incident semi–edge. Then it holds either $r_\pi(v) = \langle h, h', k\rangle$ or $r_\pi(v) = \langle h, k, h'\rangle$. Hence M contains a monogon $\langle v, h', h\rangle$ or $\langle v, h, h'\rangle$, respectively. On the other hand, a double edge does not necessarily correspond to a digon.

In part I, we used a bottom–up approach for the construction, i.e. bigger maps were constructed out of smaller ones. However, in order to avoid isomorphic maps, we used a top–down canonicity criterion, i.e. when a map M is constructed out of a smaller map A, then in general we must look at M and not at A in order to find out if A is the canonical ancestor of M. This top–down canonicity criterion based on a top–down inductive definition of $\mathcal{T}$, i.e. we showed how every

[1] As always, we consider only pairwise non–isomorphic maps so that $\mathcal{M}(g, n, D, b_l, b_u)$ contains exactly one element of each isomorphism class of maps having the desired properties.

map $M \in \mathcal{T} - \mathcal{B}$ could be reduced to a smaller map.

Now, the goal is to construct $\mathcal{M}(g, n, D, b_l, b_u)$ for given parameters g, n, D, b_l, and b_u. For the constructive definition of $\mathcal{M}(g, n, D, b_l, b_u)$, we will again provide a top–down approach: We will show how a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ can be transformed into a set of maps representing the single faces of M by applying a sequence of operations. The canoncity criterion will be a top–down canonicity criterion, too. The top–down definition can uniquely be applied to every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ and to every intermediate map. The top–down definition can be used to construct M out of its faces. This is again a bottom–up approach for the construction. The bottom–up approach is an advancement of the approach described in [46]. Moreover, we will also provide a top–down approach for the construction.

6.1 When $\mathcal{M}(g, n, D, b_l, b_u)$ is empty

Let $M \in \mathcal{M}(g, n, D, b_l, b_u)$ so that $b_l(d) \leq p_d(M) \leq b_u(d)$ for every $d \in D$ and $p_d(M) = 0$ for every $d \notin D$. Then formula 1.3 on page 9 always holds. Since $g_M = g$ and $p_d(M) = 0$ for every $d \notin D$ in every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, formula 1.3 can be rewritten as follows:

$$\sum_{d \in D} p_d(M) \cdot (6 - d) = 12 \cdot (1 - g) \tag{6.1}$$

Let $n' := |D|$ and $D = \{d_1, \dots, d_{n'}\}$ where the elements are sorted such that $d_i < d_j$ if and only if $i < j$. Let $f_i := p_{d_i}(M)$, $i \in \{1, \dots, n'\}$, be the number of d_i–gons in M. Then the sequence $s_D(M) = \langle f_1, \dots, f_{n'} \rangle$ is called the **face vector** of M (with respect to D).

Given D, g, n, and a sequence $s = \langle f_1, \dots, f_{n'} \rangle$ with $f_i \in \mathbb{N}_0$, $b_l(d_i) \leq f_i \leq b_u(d_i)$ for every $i \in \{1, \dots, n'\}$, and $\sum_{i=1}^{n'} f_i = f$ (where $f = \frac{n}{2} + 2 - 2g$ is the number of faces in a cubic map of order n and genus g), it can be verified applying formula 6.1 whether there possibly exists a cubic map M with $g_M = g$, $n_M = n$, and $s_D(M) = s$. For the test, formula 6.1 is rewritten as follows:

$$\sum_{i=1}^{n'} f_i \cdot (6 - d_i) = 12 \cdot (1 - g) \tag{6.2}$$

If the formula is not satisfied for any sequence $\langle f_1, \dots, f_{n'} \rangle$ with $f_i \in \mathbb{N}_0$, $b_l(d_i) \leq f_i \leq b_u(d_i)$ for every $i \in \{1, \dots, n'\}$, and $\sum_{i=1}^{n'} f_i = f$, then any construction can be omitted. For example, $\mathcal{M}(0, 50, \{5, 7\}, b_l, b_u) = \emptyset$ for any b_l and b_u because $f = 27$ and no sequence $\langle f_1, f_2 \rangle$ with $f_1, f_2 \in \mathbb{N}_0$ and $f_1 + f_2 = 27$ satisfies formula 6.2.

Lemma 6.1 *If $n' \leq 2$, then for a fixed number f of faces, there is at most one sequence $\langle f_1, \dots, f_{n'} \rangle$ with $f_i \in \mathbb{N}_0$ for every $i \in \{1, \dots, n'\}$ and $\sum_{i=1}^{n'} f_i = f$ satisfying formula 6.2.*

Proof: If $n' = 1$, then the statement is clear: It must hold $f_1 = f$. The formula $f_1 \cdot (6 - d_1) = 12 \cdot (1 - g)$ might be satisfied or not. If $n' = 2$, then formula 6.2 yields $f_1 \cdot (6 - d_1) + f_2 \cdot (6 - d_2) = 12 \cdot (1 - g)$, and together with the condition $f_1 + f_2 = f$, the equation can be transformed as follows:

$$\begin{aligned} & & f_1 \cdot (6 - d_1) + f_2 \cdot (6 - d_2) &= 12 \cdot (1 - g) \\ \Leftrightarrow & & f_1 \cdot (6 - d_1) + (f - f_1) \cdot (6 - d_2) &= 12 \cdot (1 - g) \\ \Leftrightarrow & & f_1 \cdot (d_2 - d_1) &= 12 \cdot (1 - g) - f \cdot (6 - d_2) \\ \Leftrightarrow & & f_1 &= \frac{12 \cdot (1 - g) - f \cdot (6 - d_2)}{d_2 - d_1} \end{aligned}$$

Thus the value of f_1 is unique. ■

If f_1 in Lemma 6.1 is an integer, then with $f_2 = f - f_1$, formula 6.2 is satisfied, otherwise formula 6.2 cannot be satisfied with integers. It can easily be verified if the conditions $b_l(d_1) \leq f_1 \leq b_u(d_1)$ and $b_l(d_2) \leq f_2 \leq b_u(d_2)$ are satisfied, too.

On the other hand, if $n' \geq 3$, then there might exist several sequences of integers f_i that satisfy formula 6.2 as well as the limits b_l and b_u as well as the equation $\sum_{i=1}^{n'} f_i = f$. In both cases ($n' \leq 2$ and $n' \geq 3$), it is useful to modify b_l and b_u such that the limits become as sharp as possible, i.e. for every $i \in \{1, \ldots, n'\}$, there exists a sequence $\langle f_1, \ldots, f_{n'} \rangle$ (with the above properties) such that $f_i = b_l(d_i)$ and a sequence $\langle f'_1, \ldots, f'_{n'} \rangle$ such that $f'_i = b_u(d_i)$. None of the $2n'$ considered sequences need to be identical with each other. We will see in Section 8.7 why sharp limits are useful. So in order to modify the limits and for further considerations throughout the rest of this text, we compute all *valid* sequences, i.e. all sequences $s = \langle f_1, \ldots, f_{n'} \rangle$ with $f_i \in \mathbb{N}_0$ and $b_l(d_i) \leq f_i \leq b_u(d_i)$ for every $i \in \{1, \ldots, n'\}$, $\sum_{i=1}^{n'} f_i = f$, and $\sum_{i=1}^{n'} f_i \cdot (6 - d_i) = 12 \cdot (1 - g)$ (so that possibly $s = s_D(M)$ for some map $M \in \mathcal{M}(g, n, D, b_l, b_u)$), before performing any construction.

Due to Section 1.4, in some cases it can be found out very easily that any construction can be omitted: It holds $\mathcal{M}(g, n, D, b_l, b_u) = \emptyset$ if one of the following conditions is satisfied:

- $g = 0$ and $D \cap \{1, \ldots, 5\} = \emptyset$
- $g = 1$ and $D \cap \{1, \ldots, 6\} = \emptyset$
- $g = 1$ and $D \subseteq \{1, \ldots, 5\}$
- $g \geq 2$ and $D \subseteq \{1, \ldots, 6\}$

There exist some more sophisticated theorems that state for given parameters g, n, and D and for given face vectors that there exists no appriopriate cubic map although the parameters match formula 1.3. In order to prevent useless processes, it is verified in *CGF* whether the parameters match the conditions in one of the theorems before performing any construction.

Theorem 6.2 ([47]) *Let p_i be the number of i–gons in a simple 3–connected cubic map of genus g. Let $f = \sum_{i \geq 3} p_i \geq 7$. Then it holds*

$$p_6 \geq 4(1 - g) - \frac{1}{3}(2p_4 + 3p_5) + \frac{1}{3} \sum_{i \geq 7} (\lfloor \frac{k+1}{2} \rfloor - 6) \cdot p_i$$

Theorem 6.3 ([47, 48]) *Let p_i be the number of i–gons in a simple 3–connected cubic map of genus g. Let $\sum_{i \geq 7} p_i \geq 3$. Then it holds*

$$p_6 \geq 2(1 - g) - \frac{1}{2}(p_3 - p_5) - \sum_{i \geq 7} p_i$$

These theorems can be used to modify $b_l(6)$ in the case that only *3–connected* maps shall be accepted. In this case, $b_l(6)$ is the minimum over all lower bounds for p_6 that have been computed by applying the above theorems to every valid sequence. If $b_l(6) > b_u(6)$ after the modification, then any construction can be omitted. Although Theorem 6.3 is formulated only for 3–connected maps in [48], it is valid also for maps that are not 3–connected.

The following theorem is a summarization of Theorems $1(\beta)$ and $1(\gamma)$ in [49]:

Theorem 6.4 *If* $\sum_{i \not\equiv 0 \pmod 3} p_i \leq 2$ *and* $p_6 \not\equiv \sum_{3 \leq i \neq 6} p_i \pmod 2$, *then there does not exist a simple 3–connected planar cubic map* M *where* p_i, $i \geq 3$, *is the number of* i*–gons in* M.

Theorem 6.5 ([49]) *If* $p_5 = 1$, $p_7 = 1$, $p_i = 0$ *for* $i \notin \{5, 6, 7\}$, *and* $p_6 \in \mathbb{N}_0$, *then there does not exist a simple 3–connected cubic toroidal map* M *where* p_i, $i \geq 3$, *is the number of* i*–gons in* M.

Theorems about planar cubic maps that are not necessarily 3–connected are found in [50] (Section 13.4, Theorems $3(k)$, $4(3)$, $4(4)$, and $4(5)$).

For some parameters D and b_l, Section 1.4 yields simple results about the connectivity of the maps in $\mathcal{M}(g, n, D, b_l, b_u)$: If $n > 2$, $1 \in D$, and $b_l(1) > 0$, then every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is exactly 1–connected. Hence if only 2- or 3–connected maps are desired, then any construction can be omitted. If $n > 2$, $2 \in D$, and $b_l(2) > 0$, then every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is at most 2–connected. Hence if only 3–connected maps are desired, then any construction can be omitted.

6.2 Uniquely encoding a rooted map

The vertex code described in Section 4.1 is applicable only to *simple* rooted maps because the elements of multiple edges cannot be distinguished. However, for maps of genus 0 it is possible to introduce an additional encoding rule that yields unique vertex codes also in case of maps that are not simple. On the other hand, this is not possible for maps of higher genus. Hence we introduce a different encoding that uniquely encodes arbitrary rooted maps.

Suppose $M = (G, \Pi)$ with $G = (V, H, E, T)$ contains n vertices and m edges. Suppose $(V \cup H) \cap (\{1_\text{v}, 2_\text{v}, \ldots, n_\text{v}\} \cup \{1_\text{e}, 2_\text{e}, \ldots, m_\text{e}\}) = \emptyset$. Then the *edges* and semi–edges will be labelled with the numbers $1_\text{e}, \ldots, m_\text{e}$ in increasing order. As soon as a semi–edge h is labelled, its inverse h' and the edge $e = \{h, h'\}$ are labelled with the same label, too. The same holds vice versa. This fact should be kept in mind throughout the following description of the encoding. While labelling the edges and semi–edges, simultaneously the vertices will be labelled with the numbers $1_\text{v}, \ldots, n_\text{v}$ in increasing order as described in Section 4.1 (the suffixes "e" and "v" are used in order to prevent mixing up the labels). In the following description, we will use also the new labels in order to refer to the vertices. This works well because we will only refer to vertices by their labels if they have been labelled before.

Let $r(e) = (e, h, s_e)$ be the root. Then e is labelled 1_e. The source of h is labelled 1_v and the target is labelled 2_v. Next, we label the semi–edges that are incident with the vertex 1_v, and that is done starting with h (which is already labelled) using the semi–edge rotation Π if $s_e =$ `right` and $\overline{\Pi}$ if $s_e =$ `left`, respectively. Thus, we label the semi–edges (and the corresponding inverses and edges) $\pi_{1_\text{v}}(h)$, $\pi_{1_\text{v}}^2(h)$, ... or $\overline{\pi_{1_\text{v}}}(h)$, $\overline{\pi_{1_\text{v}}}^2(h)$, ..., respectively, as long as they are not yet labelled (in this first step, this is always the case unless we have a loop). For the labelling, it is always taken the smallest remaining label. Simultaneously, we label those targets of the considered semi–edges that have not yet been labelled. Here, for the labelling, it is also always taken the smallest remaining label.

Next, we repeat the following steps for the vertices $v = 2_{\mathrm{v}}, \ldots, n_{\mathrm{v}}$: At least one of the semi–edges incident with v is labelled (v is labelled and there must be a path connecting vertex 1_{v} with v such that p contains only labelled edges). Starting with the semi–edge h having the smallest label among all labelled semi–edges incident with v (h cannot be an element of a loop except for $v = 1_{\mathrm{v}}$ because h must have been labelled by applying π_u or $\overline{\pi_u}$ to a vertex $u \neq v$), the semi–edges incident with v are glanced over in the order given by π_v or $\overline{\pi_v}$ depending on s_e as above, and whenever an unlabelled semi–edge occurs, the smallest remaining label will be assigned to it as well as to its inverse and the corresponding edge. Simultaneously, the neighbours of v are labelled. In detail: We label the unlabelled semi–edges of $\pi_v(h)$, $\pi_v^2(h)$, ... or $\overline{\pi_v}(h)$, $\overline{\pi_v}^2(h)$, ..., respectively, in exactly this order. Simultaneously, we label the unlabelled targets of these semi–edges in the same order. Since M is connected, every edge and vertex is reached and labelled.

From this labelling, we obtain the unique **edge code** of $(M, r(e))$ by recording the labels of the semi–edges in the order they appear whenever a rotation π or $\overline{\pi}$ is applied. Here the labels are recorded no matter if the semi–edges had already been labelled before or not. Thus for every vertex $v = 1_{\mathrm{v}}, \ldots, n_{\mathrm{v}}$, the labels of all incident semi–edges (and thus of all incident edges) are recorded in a well–defined order (corresponding to r_{π_v} or $r_{\overline{\pi_v}}$). After all incident semi–edges of one vertex have been recorded, a zero is inserted into the code. Thus in the cubic case, we obtain a sequence of length $4n$ (each vertex is incident with 3 semi–edges; additionally, we have n zero entries). For practical reasons, the number n precedes the code, but in priciple this number is redundant. Figure 6.1 shows an example of a rooted map. Here the labels of the vertices and edges can be distinguished by their sizes instead of their suffixes. The edge code of the example map with respect to $r(e)$ is $\langle 10,\ 1,2,3,0,\ 1,4,5,0,\ 2,6,7,0,\ 3,7,8,0,\ 4,9,10,0,\ 5,11,12,0,\ 6,10,8,0,\ 9,13,14,0,$ $11,14,15,0,\ 12,15,13,0\rangle$.

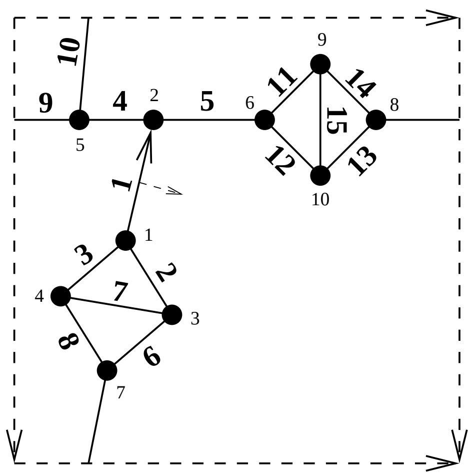

Figure 6.1

6.3 Petrie paths

For the constructive definition of $\mathcal{M}(g, n, D, b_l, b_u)$, we use *Petrie paths*. A **Petrie path** in a cubic map $M = (G, \Pi)$ is a zig–zag path $p = \langle h_1, h'_1, \ldots, h_l, h'_l \rangle$ that is maximally extended in the following sense: Let $e_0 = \{h_0, h'_0\}$ be the unique edge such that $p_1 = \langle h_0, h'_0, h_1, h'_1, \ldots, h_l, h'_l \rangle$ is a zig–zag pseudo–path. Let $e_{l+1} = \{h_{l+1}, h'_{l+1}\}$ be the unique edge such that $p_2 = \langle h_1, h'_1, \ldots, h_l, h'_l, h_{l+1}, h'_{l+1} \rangle$ is a zig–zag pseudo–path. Then neither p_1 nor p_2 is a path, i.e. $e_0 = \{h_i, h'_i\}$ and $e_{l+1} = \{h_j, h'_j\}$ for some $i, j \in \{1, \ldots, l\}$.

To every rooted edge $r(e) = (e, h, s_e)$, a unique Petrie path can be assigned as follows: Using h as the start semi–edge, we trace a zig–zag pseudo–path in M as long as possible. If $s_e = \texttt{right}$, then the path must be continued to the right, too, and if $s_e = \texttt{left}$, then the path must be continued to the left, too. Let $e_0 = \{h_0, h'_0\}, e_1 = \{h_1, h'_1\}, \ldots, e_{k-1} = \{h_{k-1}, h'_{k-1}\}$ be the first k edges of the zig–zag pseudo–path with start semi–edge $h_0 = h$. Then there is some upper limit for k such that the zig–zag pseudo–path $\langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1} \rangle$ is still a path but $\langle h_0, h'_0, \ldots, h_k, h'_k \rangle$ is not: Since G is finite, there is some smallest k such that $e_k = e_r$ for some index $r \in \{0, \ldots, k-1\}$.

Let k be this upper limit. Now we use h' as the start semi–edge in order to trace a zig–zag pseudo–path into the other direction. As before, this zig–zag pseudo–path is continued to the right if and only if $s_e = \texttt{right}$. We follow the zig–zag path and label the semi–edges $h'_0, h_0, h'_{-1}, h_{-1}, \ldots$ where $h'_0 = h'$ and $h_0 = h$ as before. Again, there is some biggest index m such that the zig–zag pseudo–path $p = \langle h_m, h'_m, h_{m+1}, h'_{m+1}, \ldots, h_0, h'_0, \ldots, h_{k-1}, h'_{k-1} \rangle$ is still a path but $\langle h_{m-1}, h'_{m-1}, h_m, h'_m, \ldots, h_0, h'_0, \ldots, h_{k-1}, h'_{k-1} \rangle$ is not. The resulting Petrie path p is the Petrie path that is assigned to $r(e)$ (we say "$r(e)$ *initiates* p"). $r(e) = (e, h, s_e)$ is called the **initial rooted edge** of p, e is called the **initial edge** of p, and h is called the **initial semi–edge** of p. In general, h is *not* the start semi-edge of p (only if $m = 0$).

A Petrie path has minimum length 2. If M is simple, then a Petrie path in M has minimum length 4. There exist three **types** of Petrie paths: **Jordan curve paths**, **sandwich paths**, and **dumbbell paths**. Examples for all three types of paths can be found in Figure 6.2. Let $p = \langle h_0, h'_0, \ldots, h_l, h'_l \rangle$ be a Petrie path in M. The three types have the following properties:

- If p is a Jordan curve path, then p is a cycle of even length $l + 1$.
- If p is a sandwich path, then the start vertex u and the end vertex v of p are connected by three disjoint paths that are all parts of p (i.e. there exist $r, s \in \{0, \ldots, l\}$ such that u and v are connected by $\langle h_0, h'_0, \ldots, h_r, h'_r \rangle$ as well as by $\langle h_{r+1}, h'_{r+1}, \ldots, h_{s-1}, h'_{s-1} \rangle$ as well as by $\langle h_s, h'_s, \ldots, h_l, h'_l \rangle$).
- If p is a dumbbell path, then the start vertex u and the end vertex v of p are connected by only one part of p (i.e. there exist $r, s \in \{0, \ldots, l\}$ such that u and v are connected by $\langle h_{r+1}, h'_{r+1}, \ldots, h_{s-1}, h'_{s-1} \rangle$, but $\langle h_0, h'_0, \ldots, h_r, h'_r \rangle$ as well as $\langle h_s, h'_s, \ldots, h_l, h'_l \rangle$ are cycles).

In the list below, there will be given a simpler combinatorial distinction between sandwich paths and dumbbell paths.

In general, not every edge in a map is the start edge or the end edge of some Petrie path. This is why the initial rooted edge is so important in order to refer to an appointed Petrie path. A Petrie path is uniquely defined by its initial rooted edge. However, the inverse does not hold: Different initial rooted edges might initiate identical Petrie paths.

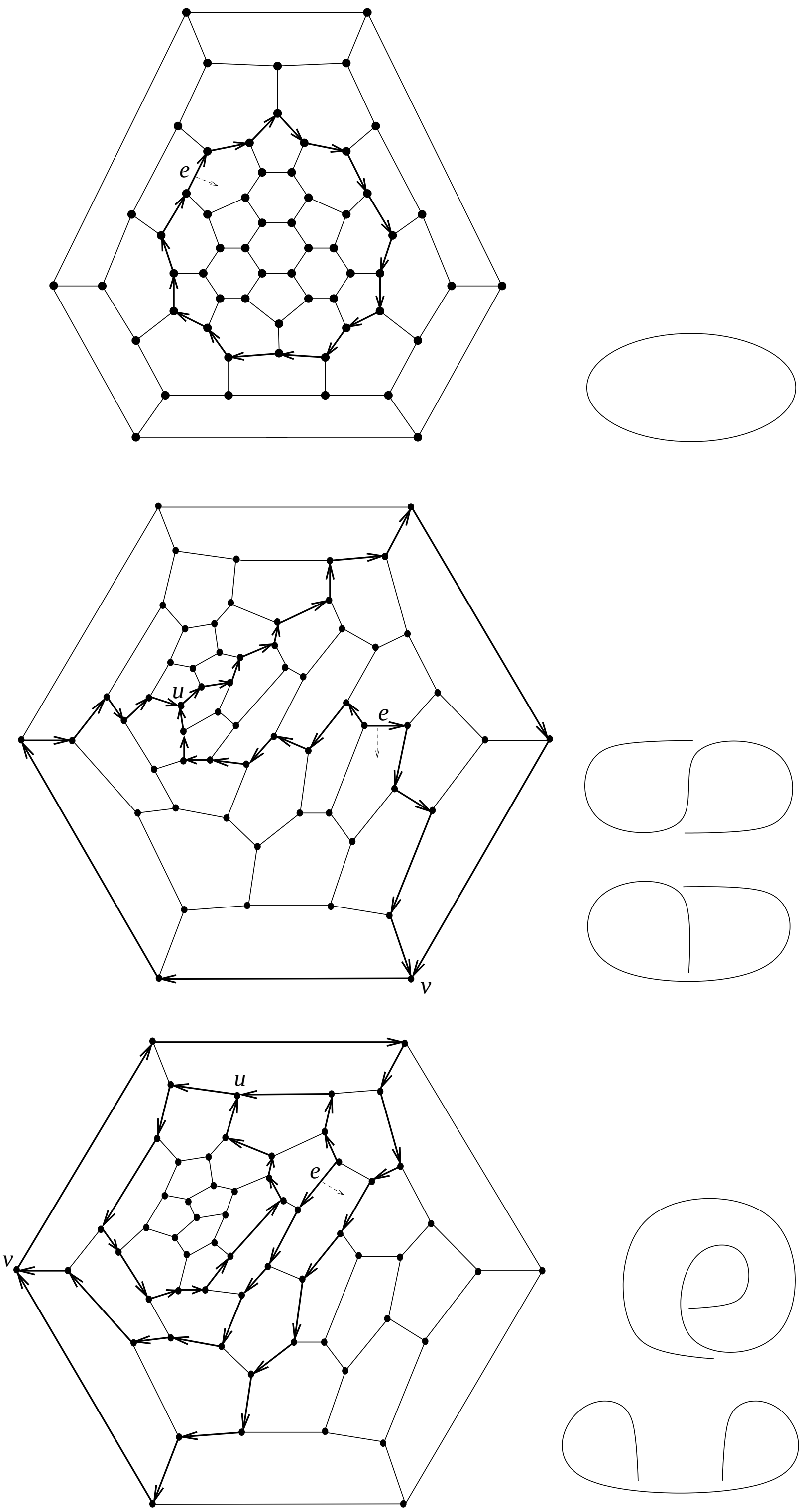

Figure 6.2: A Jordan curve path, a Sandwich path, and a dumbbell path in a concrete map and in schematic form. The arrows indicate in which direction the zig–zag path was traced while finding the Petrie path.

We will now give a detailed answer to the question when different initial rooted edges initiate identical Petrie paths. Let $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ with $k \geq 2$ be a Petrie path in M. Moreover (also throughout the rest of this section), let $e_i = \{h_i, h'_i\}$ and let v_i be the source of h_i for every $i \in \{0, \ldots, k-1\}$. Let v_k be the target of h_{k-1}.

- Let p be a Jordan curve path. Then k is even and $v_0 = v_k$. The initial semi–edge is always the start semi–edge. Since the first continuation is uniquely given by the side of the initial rooted edge, the initial edge that initiates p is unique.
- Let p be a sandwich path. Then $v_0 = v_t$ for some $t \in \{2, \ldots, k-1\}$ and $v_k = v_s$ for some $s \in \{1, \ldots, t-1\}$. Let p be continued to the right. Then every initial rooted edge $(e_i, h_i, \texttt{right})$ for even i and $(e_i, h_i, \texttt{left})$ for odd i initiates p provided that $i \in \{0, \ldots, s-1\}$ if $k-s$ is even and $i \in \{0, \ldots, s\}$ if $k-s$ is odd (see below). If p is continued to the left, then every side in the initial rooted edges must be reversed, i.e. every initial edge $(e_i, h_i, \texttt{left})$ for even i and $(e_i, h_i, \texttt{right})$ for odd i initiates p provided that $i \in \{0, \ldots, s-1\}$ if $k-s$ is even and $i \in \{0, \ldots, s\}$ if $k-s$ is odd.
- Let p be a dumbbell path. Then $v_0 = v_t$ for some $t \in \{1, \ldots, k-2\}$ and $v_k = v_s$ for some $s \in \{t+1, \ldots, k-1\}$. Let w.l.o.g. p be continued to the right (otherwise as before, every side in the set of initial rooted edges must be reversed). Then every initial edge $(e_i, h_i, \texttt{right})$ for even i and $(e_i, h_i, \texttt{left})$ for odd i initiates p provided that $i \in \{0, \ldots, s-1\}$ if $k-s$ is even and $i \in \{0, \ldots, s\}$ if $k-s$ is odd.

Here is some explanation for the sandwich and the dumbbell case: Since $v_k = v_s$, the semi–edge h'_{k-1} is adjacent to h'_{s-1} and h_s. If (e_i, h_i, x), $i \in \{0, \ldots, s-1\}$, is the initial rooted edge (with the appropriate side $x \in \{\texttt{left}, \texttt{right}\}$ as indicated in the above list), then both h'_{s-1} and h_s and hence both e_{s-1} and e_s are already in the path as soon as h'_{k-1} is traced. Hence the path cannot be extended and thus the path indeed ends with h'_{k-1}. Also, both e_{t-1} and e_t are already in the path as soon as h_0 is traced. Hence the path cannot be extended into the other direction and thus it indeed starts with h_0. Hence $\langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ is the resulting Petrie path. Moreover, if and only if $k-s$ is odd, then the zig–zag path $\langle h_s, h'_s, \ldots, h_{k-1}, h'_{k-1}\rangle$ cannot be extended after h'_{k-1} because it would have to be extended with h_s. Hence in this case, it may hold $i \in \{0, \ldots, s\}$ for the initial edge in order to initiate p.

For future use, we introduce the following definitions and notation:

- Let $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ be a sandwich path with $v_0 = v_t$ for some $t \in \{2, \ldots, k-1\}$ and $v_k = v_s$ for some $s \in \{1, \ldots, t-1\}$. Then the path $\langle h_0, h'_0, \ldots, h_{s-1}, h'_{s-1}\rangle$ is called the **first part**, $\langle h_s, h'_s, \ldots, h_{t-1}, h'_{t-1}\rangle$ is called the **second part**, and $\langle h_t, h'_t, \ldots, h_{k-1}, h'_{k-1}\rangle$ is called the **third part** of p. By $l_i(p)$, $i \in \{1,2,3\}$, we denote the length of the i–th part of p.
- Let $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ be a dumbbell path with $v_0 = v_t$ for some $t \in \{1, \ldots, k-2\}$ and $v_k = v_s$ for some $s \in \{t+1, \ldots, k-1\}$. Then $\langle h_0, h'_0, \ldots, h_{t-1}, h'_{t-1}\rangle$ is called the **first part**, $\langle h_t, h'_t, \ldots, h_{s-1}, h'_{s-1}\rangle$ is called the **second part**, and $\langle h_s, h'_s, \ldots, h_{k-1}, h'_{k-1}\rangle$ is called the **third part** of p. By $l_i(p)$, $i \in \{1,2,3\}$, we denote the length of the i–th part of p.

With the terminology just introduced, one can say that if $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ is a sandwich path, then p is initiated by (e_i, h_i, x_i) if h_i is incident with the first part of p (where x_i is the $(i+1)$–th continuation of p). If $l_2(p) + l_3(p)$ is even, then p is also initiated by (e_i, h_i, x_i) if h_i is the start semi–edge of the second part of p. Similarly, if p is a dumbbell path, then p is initiated by (e_i, h_i, x_i) if h_i is incident with the first or second part of p, and if $l_3(p)$ is even, then p is also initiated by (e_i, h_i, x_i) if h_i is the start semi–edge of the third part of p.

In M, two Petrie paths $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ and $q = \langle d_0, d'_0, \ldots, d_{m-1}, d'_{m-1}\rangle$ are called **equivalent** if and only if the following holds:

- Both p and q are Jordan curve paths and $\{h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\} = \{d_0, d'_0, \ldots, d_{m-1}, d'_{m-1}\}$.
- Both p and q are sandwich or dumbbell paths and $p \in \{q, \overline{q}\}$.

The condition for sandwich and dumbbell paths is stronger than the condition for Jordan curve paths. See Figure 6.3 for an example of two Petrie paths $p = \langle a, b, c, d, e, f, g, h, j, k, l\rangle$ and $q = \langle b, c, d, e, f, g, h, j, k, l, a\rangle$ that contain the same set of edges but that are not equivalent. Note that "equivalent" in this context does *not* mean "isomorphic": Neither of the two terms implies the other. An example for two equivalent Petrie paths that are not isomorphic is given by the paths p and $\overline{p}$ in Figure 6.3. An example for two isomorphic paths that are not equivalent is given by any two Petrie paths in the *dodecahedron* (the map in $\mathcal{M}(0, 20, \{5\}, b_l, b_u)$ with $b_l(5) = b_u(5) = 12$) that do not contain the same set of edges.

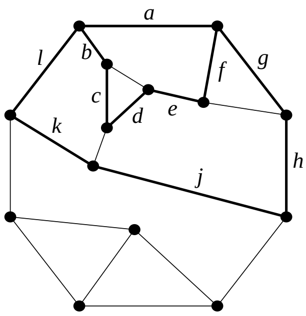

Figure 6.3

Petrie paths that are equivalent but not identical have different start (semi–)edges, end (semi–) edges, start vertices, end vertices, and parts:

- In the sandwich or dumbbell case, to each Petrie path $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ with start semi–edge h_0 and end semi–edge h'_{k-1} ($\Rightarrow$ start edge e_0, end edge e_{k-1}, start vertex v_0, and end vertex v_k), there exists exactly one equivalent Petrie path $q \neq p$, namely $q = \overline{p} = \langle h'_{k-1}, h_{k-1}, \ldots, h'_0, h_0\rangle$. This follows directly from the definition. In $\overline{p}$, the start edge is e_{k-1}, the end edge is e_0, the start semi–edge is h'_{k-1}, the end semi–edge is h_0, the start vertex is v_k, and the end vertex is v_0. The inverse of the first part of p is the third part of $\overline{p}$, the inverse of the second part of p is the second part of $\overline{p}$, and the inverse of the third part of p is the first part of $\overline{p}$.
- In the Jordan curve case, to each Petrie path $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$, there exist exactly $2k$ equivalent but pairwise non–identical Petrie paths (one of them being p): Every cyclic shift of $\langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ by a multiple of 2 semi–edges and every cyclic shift of its inverse $\langle h'_{k-1}, h_{k-1}, \ldots, h'_0, h_0\rangle$ by a multiple of 2 semi–edges yields a Petrie path, and all the resulting Petrie paths are equivalent. Hence given a set of semi–edges $h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}$ incident with a Jordan curve path p, every semi–edge is the start semi–edge of a path that is equivalent to p.

Similar to the question when different initial rooted edges initiate *identical* Petrie paths, we can now answer the question when different initial rooted edges initiate *equivalent* Petrie paths. Let $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ be a Petrie path.

- Let p be a Jordan curve path.

- If p is continued to the right, then the following initial rooted edges (e, h, s_e) initiate a Petrie path that is equivalent to p:
  - $(e_i, h_i, \texttt{right})$ for even $i \in \{0, \ldots, k-1\}$
  - $(e_i, h_i, \texttt{left})$ for odd $i \in \{0, \ldots, k-1\}$
  - $(e_i, h'_i, \texttt{right})$ for even $i \in \{0, \ldots, k-1\}$
  - $(e_i, h'_i, \texttt{left})$ for odd $i \in \{0, \ldots, k-1\}$
- If p is continued to the left, then every side must be reversed. In detail:
  - $(e_i, h_i, \texttt{left})$ for even $i \in \{0, \ldots, k-1\}$
  - $(e_i, h_i, \texttt{right})$ for odd $i \in \{0, \ldots, k-1\}$
  - $(e_i, h'_i, \texttt{left})$ for even $i \in \{0, \ldots, k-1\}$
  - $(e_i, h'_i, \texttt{right})$ for odd $i \in \{0, \ldots, k-1\}$

- Let p be a sandwich or a dumbbell path with s and t defined as above. Let w.l.o.g. p be continued to the right (otherwise every side in the following list must be reversed). The following initial rooted edges initiate a Petrie path that is equivalent to p:
  - $(e_i, h_i, \texttt{right})$ for even i and $(e_i, h_i, \texttt{left})$ for odd i where $i \in \{0, \ldots, s-1\}$ if $k-s$ is even and $i \in \{0, \ldots, s\}$ if $k-s$ is odd.
  - If k is odd (so that $\overline{p}$ is also continued to the right): $(e_i, h'_i, \texttt{right})$ for even i and $(e_i, h'_i, \texttt{left})$ for odd i where $i \in \{t, \ldots, k-1\}$ if t is even and $i \in \{t-1, \ldots, k-1\}$ if t is odd.
  - If k is even (so that $\overline{p}$ is continued to the left): $(e_i, h'_i, \texttt{left})$ for even i and $(e_i, h'_i, \texttt{right})$ for odd i where $i \in \{t, \ldots, k-1\}$ if t is even and $i \in \{t-1, \ldots, k-1\}$ if t is odd.

6.4 Cutting a cubic map along a Petrie path

The constructive definition of $\mathcal{M}(g, n, D, b_l, b_u)$ bases on an operation that will be called *cutting*. Here is how to **cut** a cubic map: A cubic map is cut along a Petrie path such that each cut edge is replaced by a pair of edges. The start vertex and the end vertex of a sandwich or a dumbbell path are each replaced by three 2–valent vertices, and every other vertex incident with the path is replaced by a 2–valent and a 3–valent vertex. Figures 6.4, 6.5, and 6.6 show examples of cuts along a Jordan curve path, a sandwich path, and a dumbbell path. Note that after the cut, of course the outermost face occurs in one of the resulting connected components, too.

The details are given in the following sections.

6.4.1 Jordan curve paths

In the following description, all indices are expressed modulo k. Let $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1} \rangle$ be a Jordan curve path in a cubic map $M = (G, \Pi)$ with $G = (V, H, E, T)$. Let v_i, $i \in \{0, \ldots, k-1\}$, be the source of h_i. v_i is also the source of h'_{i-1}. Let t_i be the third semi–edge whose source is v_i and let $e_i = \{h_i, h'_i\}$ for every $i \in \{0, \ldots, k-1\}$. Let $(V \cup H) \cap (\{u_0, \ldots, u_{k-1}\} \cup \{w_0, \ldots, w_{k-1}\} \cup \{a_0, a'_0, \ldots, a_{k-1}, a'_{k-1}\} \cup \{b_0, b'_0, \ldots, b_{k-1}, b'_{k-1}\}) = \emptyset$. By cutting the map M along p, we obtain a bunch of maps $B = (G', \Pi')$, $G' = (V', H', E', T')$, with the following properties (see Figure 6.7):

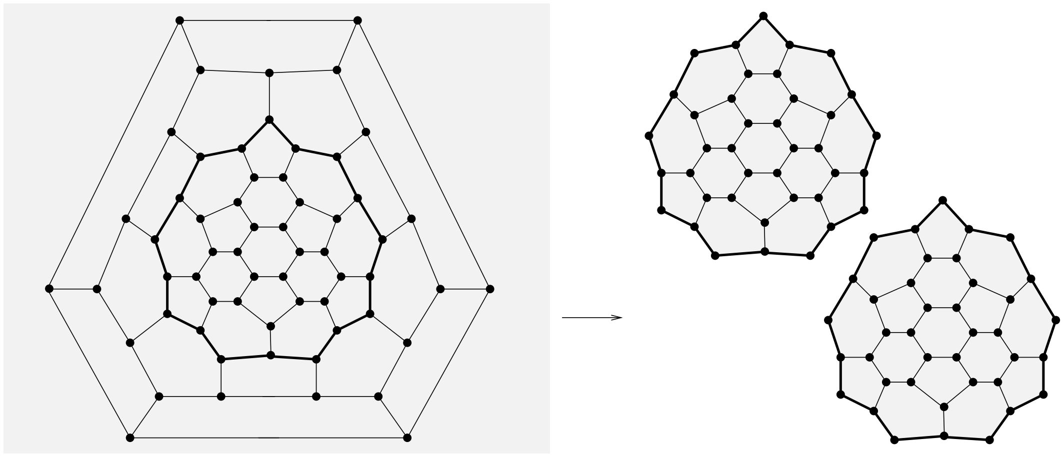

Figure 6.4

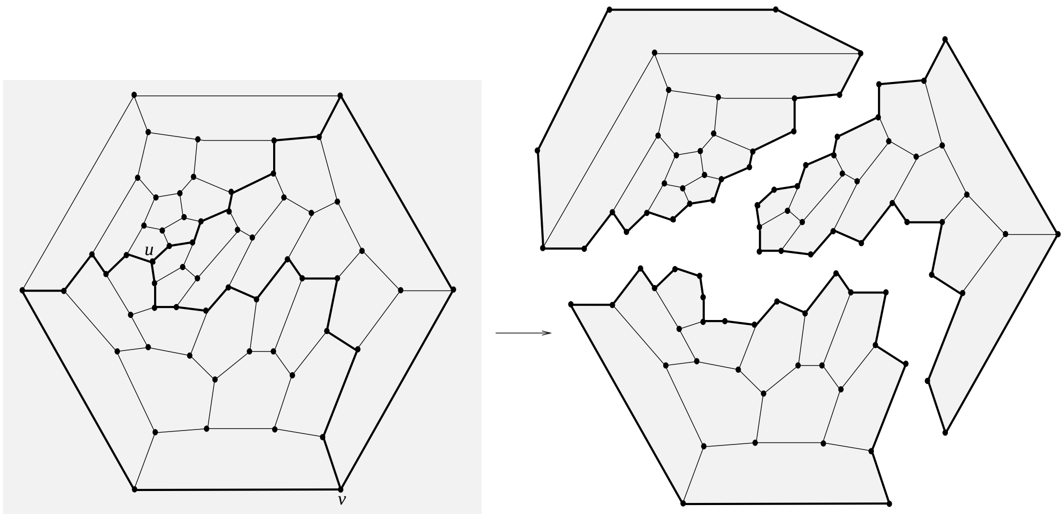

Figure 6.5

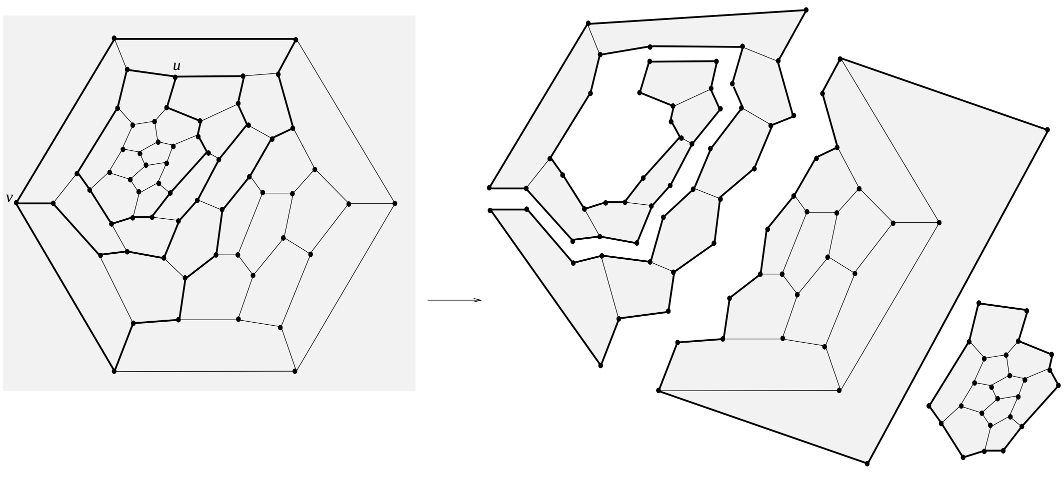

Figure 6.6

- $V' = (V - \{v_0, \ldots, v_{k-1}\}) \cup \{u_0, \ldots, u_{k-1}\} \cup \{w_0, \ldots, w_{k-1}\}$
- $H' = (H - \{h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\}) \cup \{a_0, a'_0, \ldots, a_{k-1}, a'_{k-1}\} \cup \{b_0, b'_0, \ldots, b_{k-1}, b'_{k-1}\}$
- $E' = (E - \{e_0, \ldots, e_{k-1}\}) \cup \{c_0, \ldots, c_{k-1}\} \cup \{d_0, \ldots, d_{k-1}\}$ where $c_i := \{a_i, a'_i\}$ and $d_i := \{b_i, b'_i\}$ for every $i \in \{0, \ldots, k-1\}$
- Π' is induced by the following definitions:
  - For all vertices $v \in V - \{v_0, \ldots, v_{k-1}\}$, we define $\pi'_v = \pi_v$.
  - Let $i \in \{0, \ldots, k-1\}$ and $\pi_{v_i}(h'_{i-1}) = h_i$ (if p is continued to the right, then this holds for all even i; if p is continued to the left, then this holds for all odd i). Then we define $r_{\pi'}(u_i) = \langle a'_{i-1}, a_i \rangle$, i.e. we define $\pi'_{u_i}(a'_{i-1}) = a_i$ and $\pi'_{u_i}(a_i) = a'_{i-1}$. Moreover, we define $r_{\pi'}(w_i) = \langle b_i, t_i, b'_{i-1} \rangle$.
  - Let $i \in \{0, \ldots, k-1\}$ and $\pi_{v_i}(h_i) = h'_{i-1}$. Then we define $r_{\pi'}(w_i) = \langle b_i, b'_{i-1} \rangle$ and $r_{\pi'}(u_i) = \langle a_i, a'_{i-1}, t_i \rangle$.

As always, T' is implied by E' and π'. Nevertheless, as a kind of exercise, we will give it explicitly:

$$T'(h) = \begin{cases} (u_i, u_{i+1}) & \text{if } h = a_i \text{ for some } i \in \{0, \ldots, k-1\}, \\ (u_{i+1}, u_i) & \text{if } h = a'_i \text{ for some } i \in \{0, \ldots, k-1\}, \\ (w_i, w_{i+1}) & \text{if } h = b_i \text{ for some } i \in \{0, \ldots, k-1\}, \\ (w_{i+1}, w_i) & \text{if } h = b'_i \text{ for some } i \in \{0, \ldots, k-1\}, \\ (u_i, u_j) & \text{if } h = t_i,\ \overline{t_i} = t_j,\ \Pi(h_i) = h'_{i-1}, \text{ and } \Pi(h_j) = h'_{j-1} \\ & \text{for some } i, j \in \{0, \ldots, k-1\}, \\ (u_i, w_j) & \text{if } h = t_i,\ \overline{t_i} = t_j,\ \Pi(h_i) = h'_{i-1}, \text{ and } \Pi(h'_{j-1}) = h_j \\ & \text{for some } i, j \in \{0, \ldots, k-1\}, \\ (w_i, u_j) & \text{if } h = t_i,\ \overline{t_i} = t_j,\ \Pi(h'_{i-1}) = h_i, \text{ and } \Pi(h_j) = h'_{j-1} \\ & \text{for some } i, j \in \{0, \ldots, k-1\}, \\ (w_i, w_j) & \text{if } h = t_i,\ \overline{t_i} = t_j,\ \Pi(h'_{i-1}) = h_i, \text{ and } \Pi(h'_{j-1}) = h_j \\ & \text{for some } i, j \in \{0, \ldots, k-1\}, \\ (u_i, v) & \text{if } h = t_i \text{ for some } i \in \{0, \ldots, k-1\},\ \overline{t_i} \neq t_j \text{ for every } j \in \{0, \ldots, k-1\}, \\ & T(h) = (v_i, v), \text{ and } \Pi(h_i) = h'_{i-1}, \\ (w_i, v) & \text{if } h = t_i \text{ for some } i \in \{0, \ldots, k-1\},\ \overline{t_i} \neq t_j \text{ for every } j \in \{0, \ldots, k-1\}, \\ & T(h) = (v_i, v), \text{ and } \Pi(h'_{i-1}) = h_i, \\ T(h) & \text{else.} \end{cases}$$

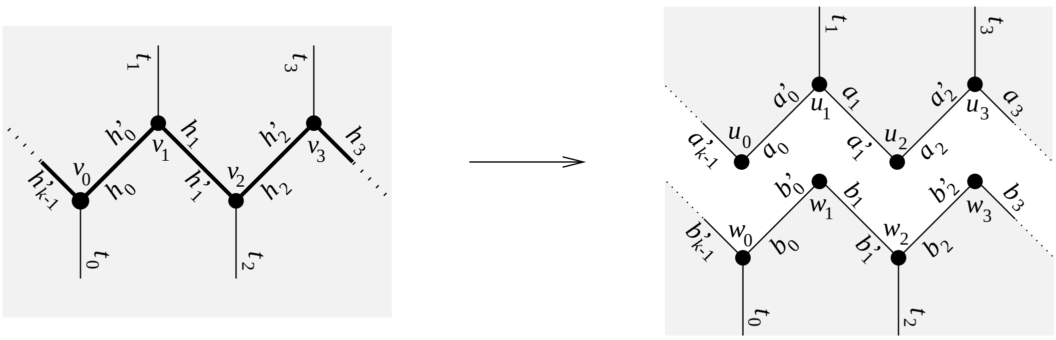

Figure 6.7

Since $\pi'_{u_i}(a_i) = a'_{i-1}$ and $\pi'_{w_i}(b'_{i-1}) = b_i$ for all $i \in \{0, \ldots, k-1\}$, we obtain two new faces $F_1 = \langle a'_k, a_k, u_k, \ldots, a'_0, a_0, u_0\rangle$ and $F_2 = \langle w_0, b_0, b'_0, \ldots, w_{k-1}, b_{k-1}, b'_{k-1}\rangle$. These faces are called *boundary faces*. All other faces in B correspond to faces in M in the following way: Whenever a face in M contains the sequence $\langle v_i, h_i, h'_i\rangle$ for any $i \in \{0, \ldots, k-1\}$, this sequence is replaced by $\langle u_i, a_i, a'_i\rangle$ in the corresponding face in B. Whenever a face in M contains the sequence $\langle h'_i, h_i, v_i\rangle$, this sequence is replaced by $\langle b'_i, b_i, w_i\rangle$ in the corresponding face in B. This can be derived from the definition of π'. Roughly spoken, these faces remain "unmodified". Thus $f_B = f_M + 2$. In Figure 6.7, the "unmodified" faces are drawn in light grey.

B contains one or two connected components: Clearly, the vertices $u_0, \ldots, u_{k-1}$ are in one common component and the the vertices $w_0, \ldots, w_{k-1}$ are in one common component (because they are incident with one common face). These components might be identical. Every other vertex v is connected with v_i for every $i \in \{0, \ldots, k-1\}$ in M, hence v is connected with u_i or w_i for some $i \in \{0, \ldots, k-1\}$ in B: Consider a path q in M with start vertex v and end vertex v_i for some $i \in \{0, \ldots, k-1\}$ where q is not incident with any v_j, $j \in \{0, \ldots, k-1\} - \{i\}$ (if it is, take j instead of i iteratively until the condition is true). Since v_i in M is substituted by u_i and w_i in B, v is connected by q with either u_i or w_i in B, and hence v is connected with all vertices in the same component. Thus there are at most 2 connected components in B.

For g_B, the following lemma holds:

Lemma 6.6 *If B is connected, then $g_B = g_M - 1$. If B contains two connected components, then $g_B = g_M$.*

Proof: From Euler's formula on page 6, we have $n_M + f_M - e_M = 2 - 2g_M$. Moreover, it holds $f_B = f_M + 2$ (see above), $n_B = n_M + k$ (see the defintion of V'), and $e_B = e_M + k$ (see the definition of E').

- Let B be connected, i.e. B is a map. Euler's formula yields

$$\begin{aligned} & n_B + f_B - e_B = 2 - 2g_B \\ \Leftrightarrow\quad & (n_M + k) + (f_M + 2) - (e_M + k) = 2 - 2g_B \\ \Leftrightarrow\quad & n_M + f_M - e_M = 2 - 2(g_B + 1) \\ \Leftrightarrow\quad & 2 - 2g_M = 2 - 2(g_B + 1) \\ \Leftrightarrow\quad & g_M = g_B + 1 \end{aligned}$$

- Let B contain two connected components (which are maps) M_1 and M_2. It holds $n_B = n_{M_1} + n_{M_2}$, $e_B = e_{M_1} + e_{M_2}$, $f_B = f_{M_1} + f_{M_2}$, $g_B = g_{M_1} + g_{M_2}$, $n_{M_1} + f_{M_1} - e_{M_1} = 2 - 2g_{M_1}$, and $n_{M_2} + f_{M_2} - e_{M_2} = 2 - 2g_{M_2}$. Hence

$$\begin{aligned} & n_{M_1} + f_{M_1} - e_{M_1} + n_{M_2} + f_{M_2} - e_{M_2} = 2 - 2g_{M_1} + 2 - 2g_{M_2} \\ \Leftrightarrow\quad & n_B + f_B - e_B = 4 - 2g_B \\ \Leftrightarrow\quad & (n_M + k) + (f_M + 2) - (e_M + k) = 4 - 2g_B \\ \Leftrightarrow\quad & n_M + f_M - e_M = 2 - 2g_B \\ \Leftrightarrow\quad & 2 - 2g_M = 2 - 2g_B \\ \Leftrightarrow\quad & g_M = g_B \end{aligned}$$

$\blacksquare$

6.4.2 Dumbbell paths

Let $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1} \rangle$ be a dumbbell path in a cubic map $M = (G, \Pi)$ with $G = (V, H, E, T)$. Let v_i, $i \in \{0, \ldots, k-1\}$, be the source of h_i, let v_k be the target of h_{k-1}, and let $e_i := \{h_i, h'_i\}$ for every $i \in \{0, \ldots, k-1\}$. Let $t \in \{1, \ldots, k-2\}$ such that $v_0 = v_t$ and let $s \in \{t+1, \ldots, k-1\}$ such that $v_s = v_k$. The vertex v_t is incident with h_0, h'_{t-1}, and h_t. The vertex v_s is incident with h'_{s-1}, h_s, and h'_{k-1}. For $i \in \{1, \ldots, k-1\} - \{s, t\}$, let j_i be the third semi–edge besides h_i and h'_{i-1} that is incident with v_i. By cutting the map M along p, we obtain a bunch of maps $B = (G', \Pi')$, $G' = (V', H', E', T')$, with the properties listed below. We must distinguish 4 different cases:

(1) $r_\pi(v_t) = \langle h_0, h_t, h'_{t-1} \rangle$ and $r_\pi(v_s) = \langle h_s, h'_{s-1}, h'_{k-1} \rangle$
(2) $r_\pi(v_t) = \langle h_0, h'_{t-1}, h_t \rangle$ and $r_\pi(v_s) = \langle h_s, h'_{s-1}, h'_{k-1} \rangle$
(3) $r_\pi(v_t) = \langle h_0, h_t, h'_{t-1} \rangle$ and $r_\pi(v_s) = \langle h_s, h'_{k-1}, h'_{s-1} \rangle$
(4) $r_\pi(v_t) = \langle h_0, h'_{t-1}, h_t \rangle$ and $r_\pi(v_s) = \langle h_s, h'_{k-1}, h'_{s-1} \rangle$

These cases depend on $l_1(p)$, $l_2(p)$, $l_3(p)$, and on the first continuation of p. Figure 6.8 shows sketches of the 4 cases.

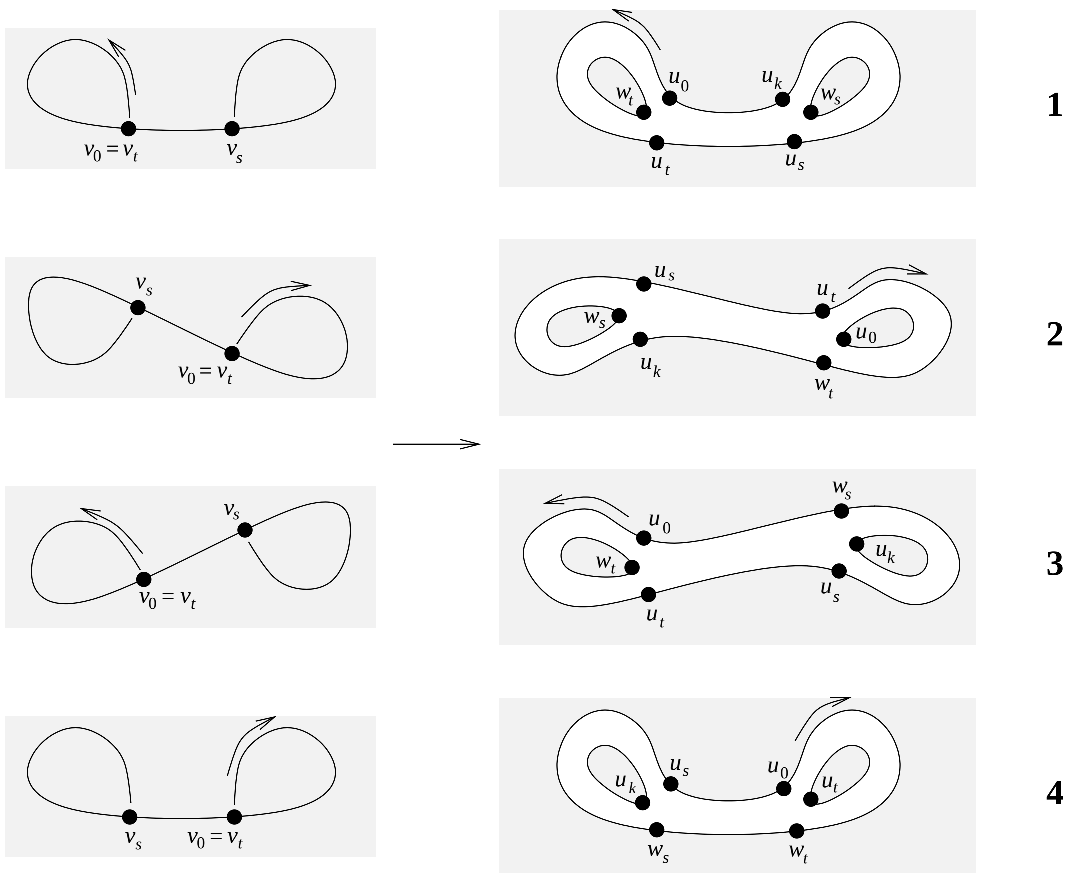

Figure 6.8: Sketches of the 4 cases. The directions of the paths are indicated by the arrows. In order to make the run of a path evident, there is a small gap between the start vertex and the start edge as well as between the end vertex and the end edge of the path.

Let $(V \cup H) \cap (\{u_0, \ldots, u_k\} \cup \{w_1, \ldots, w_{k-1}\} \cup \{a_0, a'_0, \ldots, a_{k-1}, a'_{k-1}\} \cup \{b_0, b'_0, \ldots, b_{k-1}, b'_{k-1}\}) = \emptyset$. V', H', and E' are defined as follows:

- $V' = (V - \{v_1, \ldots, v_{k-1}\}) \cup \{u_0, \ldots, u_k\} \cup \{w_1, \ldots, w_{k-1}\}$ (note that $v_0 = v_t$ and $v_s = v_k$)
- $H' = (H - \{h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\}) \cup \{a_0, a'_0, \ldots, a_{k-1}, a'_{k-1}\} \cup \{b_0, b'_0, \ldots, b_{k-1}, b'_{k-1}\}$
- $E' = (E - \{e_0, \ldots, e_{k-1}\}) \cup \{c_0, \ldots, c_{k-1}\} \cup \{d_0, \ldots, d_{k-1}\}$ where $c_i := \{a_i, a'_i\}$ and $d_i := \{b_i, b'_i\}$ for every $i \in \{0, \ldots, k-1\}$

In each of the 4 cases, the vertex v_i is substituted by u_i and w_i, $i \in \{1, \ldots, k-1\} - \{s, t\}$. The semi–edge h_i is substituted by a_i and b_i, and the semi–edge h'_i is substituted by a'_i and b'_i, $i \in \{0, \ldots, k-1\}$. The vertex v_t is substituted by u_0, u_t, and w_t. The vertex v_s is substituted by u_s, w_s, and u_k. Since the cases 3 and 4 are obviously symmetric to the cases 2 and 1, respectively, we will give details only for the first 2 cases. We will perform the substitutions in a way that by tracing the cut dumbbell path in the sketches, the u_i, a_i, and a'_i are always on the right side and the w_i, b_i, and b'_i are always on the left side. Figure 6.9 illustrates the following details for case 1.

- For all vertices $v \in V - \{v_0, \ldots, v_k\}$, we define $\pi'_v = \pi_v$.
- In both cases, we define $r_{\pi'}(u_s) = \langle a'_{s-1}, a_s\rangle$, $r_{\pi'}(w_s) = \langle b_s, b'_{k-1}\rangle$, $r_{\pi'}(u_k) = \langle a'_{k-1}, b'_{s-1}\rangle$. Moreover, in case 1 we define $r_{\pi'}(u_0) = \langle a_0, b_t\rangle$, $r_{\pi'}(u_t) = \langle a'_{t-1}, a_t\rangle$, and $r_{\pi'}(w_t) = \langle b_0, b'_{t-1}\rangle$. In case 2, we define $r_{\pi'}(u_0) = \langle a_0, a'_{t-1}\rangle$, $r_{\pi'}(u_t) = \langle b_0, a_t\rangle$, and $r_{\pi'}(w_t) = \langle b'_{t-1}, b_t\rangle$.
- Let $i \in \{1, \ldots, k-1\} - \{s, t\}$. It holds either $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$ or $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$. The semi–edges a_i and b_i correspond to h_i as well as a'_{i-1} and b'_{i-1} to h'_{i-1} in the following way: If $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, j_i, a'_{i-1}\rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}\rangle$, and if $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, a'_{i-1}\rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}, j_i\rangle$.

As always, T' is implied by E' and π'. In either case, we obtain three new *boundary faces*:

- In case 1, it holds:
  - $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{1, \ldots, t-1\}$ and $\pi'_{w_t}(b_0) = b'_{t-1}$
  - $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{s+1, \ldots, k-1\}$ and $\pi'_{w_s}(b_s) = b'_{k-1}$
  - $\pi'_{u_i}(a'_{i-1}) = a_i$ for all $i \in \{1, \ldots, k-1\}$, $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{t+1, \ldots, s-1\}$, $\pi'_{u_0}(b_t) = a_0$, and $\pi'_{u_k}(a'_{k-1}) = b'_{s-1}$

  Thus, we have the new faces
  - $F_1 = \langle w_t, b'_{t-1}, b_{t-1}, w_{t-1}, \ldots, b'_1, b_1, w_1, b'_0, b_0\rangle$,
  - $F_2 = \langle b'_{k-1}, b_{k-1}, w_{k-1}, \ldots, b'_s, b_s, w_s\rangle$, and
  - $F_3 = \langle u_0, a_0, a'_0, \ldots, u_{k-1}, a_{k-1}, a'_{k-1}, u_k, b'_{s-1}, b_{s-1}, w_{s-1}, \ldots, b'_{t+1}, b_{t+1}, w_{t+1}, b'_t, b_t\rangle$.
- In case 2, it holds:
  - $\pi'_{u_i}(a'_{i-1}) = a_i$ for all $i \in \{1, \ldots, t-1\}$ and $\pi'_{u_0}(a'_{t-1}) = a_0$
  - $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{s+1, \ldots, k-1\}$ and $\pi'_{w_s}(b_s) = b'_{k-1}$
  - $\pi'_{u_i}(a'_{i-1}) = a_i$ for all $i \in \{t+1, \ldots, k-1\}$, $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{1, \ldots, s-1\}$, $\pi'_{u_k}(a'_{k-1}) = b'_{s-1}$, and $\pi'_{u_t}(b_0) = a_t$

  Thus, we have the new faces
  - $F_1 = \langle u_0, a_0, a'_0, \ldots, u_{t-1}, a_{t-1}, a'_{t-1}\rangle$,
  - $F_2 = \langle b'_{k-1}, b_{k-1}, w_{k-1}, \ldots, b'_s, b_s, w_s\rangle$, and
  - $F_3 = \langle u_t, a_t, a'_t, \ldots, u_{k-1}, a_{k-1}, a'_{k-1}, u_k, b'_{s-1}, b_{s-1}, w_{s-1}, \ldots, b'_1, b_1, w_1, b'_0, b_0\rangle$.

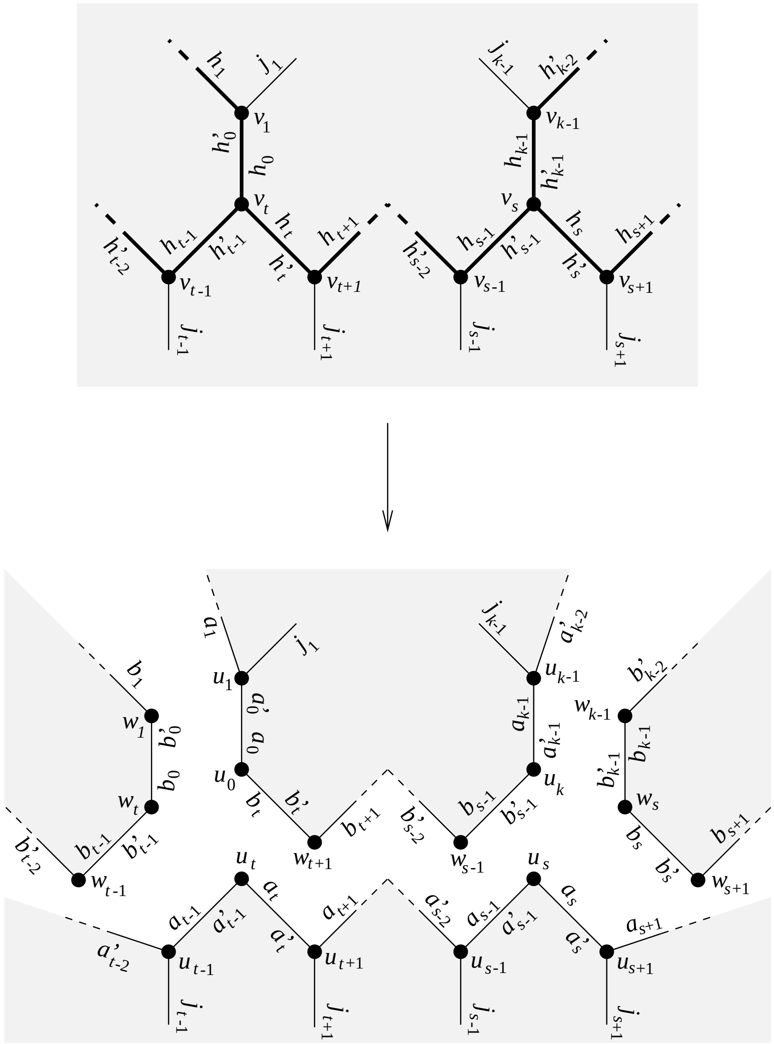

Figure 6.9

In both cases, F_3 is called the **main boundary** and F_1 and F_2 are called **side boundaries**. F_1 corresponds to the first part of p in the sense that the elements of F_1 replace elements of the first part of p. In the same way, F_2 corresponds to the third part of p. Thus, F_1 is called the **start side boundary** and F_2 is called the **end side boundary**. All other faces in B correspond to faces in M in the same way as in the Jordan curve case. The precise replacements are omitted here, but can easily be derived from the definition of π'.

B contains at most three connected components: All vertices incident with F_1, F_2, or F_3, respectively, are in one common component. Some of these components might be identical. Every other vertex v is connected with v_i for every $i \in \{0, \ldots, k-1\}$ in M, hence in B the vertex v is connected with u_i for some $i \in \{0, \ldots, k\}$ or with w_i for some $i \in \{1, \ldots, k-1\}$, and hence v is connected with all vertices in that connected component. The argumentation in detail is analogous to the one in the Jordan curve case.

Lemma 6.7 *If B is connected, then $g_B = g_M - 2$. If B contains exactly two connected components, then $g_B = g_M - 1$. If B contains three connected components, then $g_B = g_M$.*

Proof: From Euler's formula, we have $n_M + f_M - e_M = 2 - 2g_M$. Moreover, it holds $f_B = f_M + 3$, and the definitions of V' and E' imply $n_B = n_M + k + 1$ and $e_B = e_M + k$.

- Let B be connected, i.e. B is a map. Euler's formula yields

$$\begin{aligned}
& & n_B + f_B - e_B &= 2 - 2g_B \\
&\Leftrightarrow & (n_M + k + 1) + (f_M + 3) - (e_M + k) &= 2 - 2g_B \\
&\Leftrightarrow & n_M + f_M - e_M &= 2 - 2(g_B + 2) \\
&\Leftrightarrow & 2 - 2g_M &= 2 - 2(g_B + 2) \\
&\Leftrightarrow & g_M &= g_B + 2
\end{aligned}$$

- Let B contain exactly two connected components (maps) M_1 and M_2. It holds $n_B = n_{M_1} + n_{M_2}$, $e_B = e_{M_1} + e_{M_2}$, $f_B = f_{M_1} + f_{M_2}$, $g_B = g_{M_1} + g_{M_2}$, $n_{M_1} + f_{M_1} - e_{M_1} = 2 - 2g_{M_1}$, and $n_{M_2} + f_{M_2} - e_{M_2} = 2 - 2g_{M_2}$. Hence

$$\begin{aligned}
& & n_{M_1} + f_{M_1} - e_{M_1} + n_{M_2} + f_{M_2} - e_{M_2} &= 2 - 2g_{M_1} + 2 - 2g_{M_2} \\
&\Leftrightarrow & n_B + f_B - e_B &= 4 - 2g_B \\
&\Leftrightarrow & (n_M + k + 1) + (f_M + 3) - (e_M + k) &= 4 - 2g_B \\
&\Leftrightarrow & n_M + f_M - e_M &= 2 - 2(g_B + 1) \\
&\Leftrightarrow & 2 - 2g_M &= 2 - 2(g_B + 1) \\
&\Leftrightarrow & g_M &= g_B + 1
\end{aligned}$$

- Let B contain three connected components M_1, M_2, M_3. It holds $n_B = n_{M_1} + n_{M_2} + n_{M_3}$, $e_B = e_{M_1} + e_{M_2} + e_{M_3}$, $f_B = f_{M_1} + f_{M_2} + f_{M_3}$, $g_B = g_{M_1} + g_{M_2} + g_{M_3}$, $n_{M_1} + f_{M_1} - e_{M_1} = 2 - 2g_{M_1}$, $n_{M_2} + f_{M_2} - e_{M_2} = 2 - 2g_{M_2}$, and $n_{M_3} + f_{M_3} - e_{M_3} = 2 - 2g_{M_3}$. Hence

$$\begin{aligned}
& & n_{M_1} + f_{M_1} - e_{M_1} + n_{M_2} + f_{M_2} - e_{M_2} + n_{M_3} + f_{M_3} - e_{M_3} &= 6 - 2g_{M_1} - 2g_{M_2} - 2g_{M_3} \\
&\Leftrightarrow & n_B + f_B - e_B &= 6 - 2g_B \\
&\Leftrightarrow & (n_M + k + 1) + (f_M + 3) - (e_M + k) &= 6 - 2g_B \\
&\Leftrightarrow & n_M + f_M - e_M &= 2 - 2g_B \\
&\Leftrightarrow & 2 - 2g_M &= 2 - 2g_B \\
&\Leftrightarrow & g_M &= g_B
\end{aligned}$$

■

6.4.3 Sandwich paths

Let $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1} \rangle$ be a sandwich path in a cubic map $M = (G, \Pi)$ with $G = (V, H, E, T)$. Let v_i, $i \in \{0, \ldots, k-1\}$, be the source of h_i, let v_k be the target of h_{k-1}, and let $e_i := \{h_i, h'_i\}$ for every $i \in \{0, \ldots, k-1\}$. Let $t \in \{2, \ldots, k-1\}$ such that $v_0 = v_t$ and let $s \in \{1, \ldots, t-1\}$ such that $v_s = v_k$. The vertex v_t is incident with h_0, h'_{t-1}, and h_t, and the vertex v_s is incident with h'_{s-1}, h_s, and h'_{k-1}. For $i \in \{1, \ldots, k-1\} - \{s, t\}$, let j_i be the third semi–edge besides h_i and h'_{i-1} that is incident with v_i. By cutting the map M along p, we obtain a bunch of maps $B = (G', \Pi')$, $G' = (V', H', E', T')$, with the properties listed below. We must distinguish 4 different cases:

(1) $r_\pi(v_t) = \langle h_0, h_t, h'_{t-1} \rangle$ and $r_\pi(v_s) = \langle h_s, h'_{s-1}, h'_{k-1} \rangle$

(2) $r_\pi(v_t) = \langle h_0, h'_{t-1}, h_t\rangle$ and $r_\pi(v_s) = \langle h_s, h'_{s-1}, h'_{k-1}\rangle$
(3) $r_\pi(v_t) = \langle h_0, h_t, h'_{t-1}\rangle$ and $r_\pi(v_s) = \langle h_s, h'_{k-1}, h'_{s-1}\rangle$
(4) $r_\pi(v_t) = \langle h_0, h'_{t-1}, h_t\rangle$ and $r_\pi(v_s) = \langle h_s, h'_{k-1}, h'_{s-1}\rangle$

These cases depend on $l_1(p)$, $l_2(p)$, $l_3(p)$, and on the first continuation of the path. Figure 6.10 shows sketches for the 4 cases. Note that the cases 1 and 4 are impossible if M is planar. This will also be shown later.

Let $(V \cup H) \cap (\{u_0, \ldots, u_{k-1}\} \cup \{w_1, \ldots, w_k\} \cup \{a_0, a'_0, \ldots, a_{k-1}, a'_{k-1}\} \cup \{b_0, b'_0, \ldots, b_{k-1}, b'_{k-1}\}) = \emptyset$. V', H', and E' are defined as follows:

- $V' = (V - \{v_1, \ldots, v_{k-1}\}) \cup \{u_0, \ldots, u_{k-1}\} \cup \{w_1, \ldots, w_k\}$ (note that $v_0 = v_t$ and $v_s = v_k$)
- $H' = (H - \{h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\}) \cup \{a_0, a'_0, \ldots, a_{k-1}, a'_{k-1}\} \cup \{b_0, b'_0, \ldots, b_{k-1}, b'_{k-1}\}$
- $E' = (E - \{e_0, \ldots, e_{k-1}\}) \cup \{c_0, \ldots, c_{k-1}\} \cup \{d_0, \ldots, d_{k-1}\}$ where $c_i := \{a_i, a'_i\}$ and $d_i := \{b_i, b'_i\}$ for every $i \in \{0, \ldots, k-1\}$.

In each of the 4 cases, the vertex v_i is substituted by u_i and w_i, $i \in \{1, \ldots, k-1\} - \{s, t\}$. The semi–edge h_i is substituted by a_i and b_i, and the semi–edge h'_i is substituted by a'_i and b'_i, $i \in \{0, \ldots, k-1\}$. The vertex v_t is substituted by u_0, u_t, and w_t. The vertex v_s is substituted by u_s, w_s, and w_k. Since the cases 3 and 4 are obviously symmetric to the cases 2 and 1, respectively, we will give details only for the first 2 cases. We will perform the substitutions in a way that by tracing the cut sandwich path in the sketches, the u_i, a_i, and a'_i are always on the right side and the w_i, b_i, and b'_i are always on the left side. Figure 6.11 illustrates the following details for case 2.

- For all vertices $v \in V - \{v_0, \ldots, v_k\}$, we define $\pi'_v = \pi_v$.
- In both cases, we define $r_{\pi'}(u_s) = \langle a'_{s-1}, a_s\rangle$, $r_{\pi'}(w_s) = \langle b'_{s-1}, a'_{k-1}\rangle$, $r_{\pi'}(w_k) = \langle b_s, b'_{k-1}\rangle$. Moreover, in case 1 we define $r_{\pi'}(u_0) = \langle a_0, b_t\rangle$, $r_{\pi'}(u_t) = \langle a'_{t-1}, a_t\rangle$, and $r_{\pi'}(w_t) = \langle b'_{t-1}, b_0\rangle$. In case 2, we define $r_{\pi'}(u_0) = \langle a_0, a'_{t-1}\rangle$, $r_{\pi'}(u_t) = \langle b_0, a_t\rangle$, and $r_{\pi'}(w_t) = \langle b'_{t-1}, b_t\rangle$.
- Let $i \in \{1, \ldots, k-1\} - \{s, t\}$. It holds either $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$ or $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$. The semi–edges a_i and b_i correspond to h_i as well as a'_{i-1} and b'_{i-1} to h'_{i-1} in the following way: If $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, j_i, a'_{i-1}\rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}\rangle$, and if $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, a'_{i-1}\rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}, j_i\rangle$.

As always, T' is implied by E' and π'. Considering the new *boundary faces*, it holds:

- In case 2, it holds:
  - $\pi'_{u_i}(a'_{i-1}) = a_i$ for all $i \in \{1, \ldots, t-1\}$ and $\pi'_{u_0}(a'_{t-1}) = a_0$
  - $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{s+1, \ldots, k-1\}$ and $\pi'_{w_k}(b_s) = b'_{k-1}$
  - $\pi'_{u_i}(a'_{i-1}) = a_i$ for all $i \in \{t+1, \ldots, k-1\}$, $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{1, \ldots, s-1\}$, $\pi'_{u_t}(b_0) = a_t$, and $\pi'_{w_s}(a'_{k-1}) = b'_{s-1}$

  Thus, we have three new faces
  - $F_1 = \langle u_0, a_0, a'_0, \ldots, u_{t-1}, a_{t-1}, a'_{t-1}\rangle$,
  - $F_2 = \langle w_k, b'_{k-1}, b_{k-1}, \ldots, w_{s+1}, b'_s, b_s\rangle$, and
  - $F_3 = \langle w_s, b'_{s-1}, b_{s-1}, w_{s-1}, \ldots, b'_1, b_1, w_1, b'_0, b_0, u_t, a_t, a'_t, \ldots, u_{k-1}, a_{k-1}, a'_{k-1}\rangle$.

  F_3 is called the **main boundary** and F_1 and F_2 are called the **side boundaries**. F_1 corresponds to the first part of p in the sense that the elements of F_1 replace elements of the first and the second part but not of the third part of p. In the same way, F_2 corresponds to

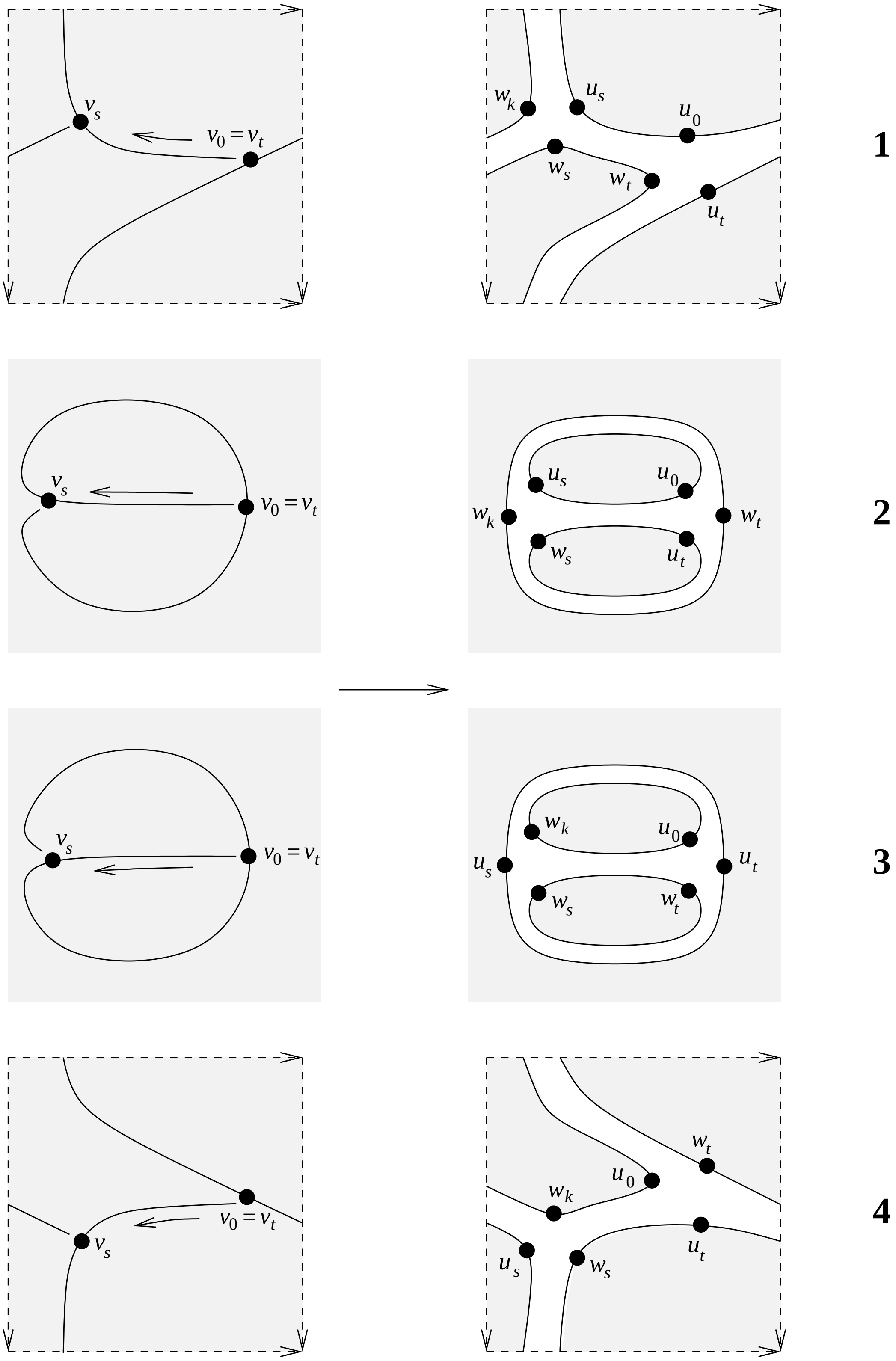

Figure 6.10: sketches of the 4 cases

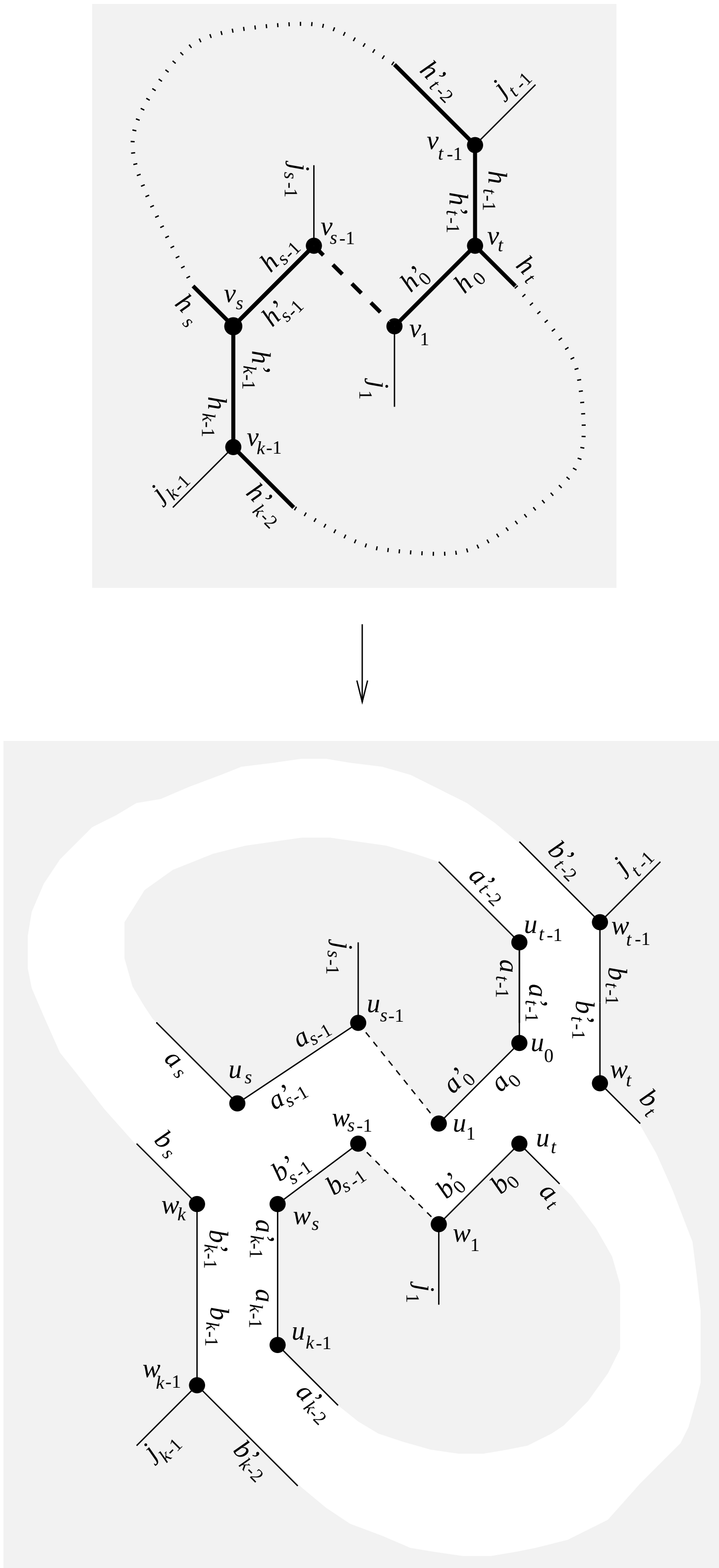

Figure 6.11

the third part of p. Thus F_1 is called the **start side boundary** and F_2 is called the **end side boundary**. The elements of F_3 replace elements of the first and the third part of p.

- In case 1, it holds $\pi'_{u_i}(a'_{i-1}) = a_i$ for all $i \in \{1, \ldots, k-1\}$, $\pi'_{w_s}(a'_{k-1}) = b'_{s-1}$, $\pi'_{u_0}(b_t) = a_0$, $\pi'_{w_i}(b_i) = b'_{i-1}$ for all $i \in \{1, \ldots, k-1\} - \{s, t\}$, $\pi'_{w_t}(b_0) = b'_{t-1}$, and $\pi'_{w_k}(b_s) = b'_{k-1}$. Thus, we have one new face $F_1 = \langle u_0, a_0, a'_0, \ldots, u_{k-1}, a_{k-1}, a'_{k-1}, w_s, b'_{s-1}, b_{s-1}, w_{s-1}, \ldots, b'_1, b_1, w_1, b'_0, b_0, w_t, b'_{t-1}, b_{t-1}, w_{t-1}, \ldots, b'_{s+1}, b_{s+1}, w_{s+1}, b'_s, b_s, w_k, b'_{k-1}, b_{k-1}, w_{k-1}, \ldots, b'_{t+1}, b_{t+1}, w_{t+1}, b'_t, b_t\rangle$.

All other faces in B correspond to faces in M in the same way as in the Jordan curve case. The precise replacements can easily be derived from the definition of π'.

In case 1, B is connected: All vertices $u_0, \ldots, u_{k-1}$ and $w_1, \ldots, w_k$ are in one common component, and every other vertex v is connected with one of these vertices in B because it is connected with v_i for every $i \in \{0, \ldots, k\}$ in M. Moreover, in case 1 it holds $f_B = f_M + 1$. In case 2, B contains at most 3 connected components: The vertices $u_0, \ldots, u_{t-1}$ are in one common component, the vertices $w_{s+1}, \ldots, w_k$ are in one common component, and the vertices $w_1, \ldots, w_s$ and $u_t, \ldots, u_{k-1}$ are in one common component. Some of these components might be identical. Every other vertex v is connected with v_i for every $i \in \{0, \ldots, k-1\}$ in M, hence in B the vertex v is connected with u_i for some $i \in \{0, \ldots, k-1\}$ or with w_i for some $i \in \{1, \ldots, k\}$, and hence v is connected with all vertices in that connected component. The argumentation in detail is analogous to the one in the Jordan curve case. Moreover, in case 2 it holds $f_B = f_M + 3$.

Lemma 6.8 *In case* 2, *if* B *is connected, then* $g_B = g_M - 2$. *If* B *contains exactly two connected components, then* $g_B = g_M - 1$. *If* B *contains three connected components, then* $g_B = g_M$.

Proof: See the proof of Lemma 6.7. ■

Lemma 6.9 *In case* 1, *it holds* $g_B = g_M - 1$.

Proof: The definitions of V' and E' imply $n_B = n_M + k + 1$ and $e_B = e_M + k$. Euler's formula yields $n_M + f_M - e_M = 2 - 2g_M$ and — since B is a map — also $n_B + f_B - e_B = 2 - 2g_B$.

$$\begin{array}{lrcl} & n_B + f_B - e_B &=& 2 - 2g_B \\ \Leftrightarrow & (n_M + k + 1) + (f_M + 1) - (e_M + k) &=& 2 - 2g_B \\ \Leftrightarrow & n_M + f_M - e_M &=& 2 - 2(g_B + 1) \\ \Leftrightarrow & 2 - 2g_M &=& 2 - 2(g_B + 1) \\ \Leftrightarrow & g_M &=& g_B + 1 \end{array}$$

■

As a consequence, case 1 (and 4) cannot occur if $g_M = 0$, i.e. if M is planar. There exists a very simple criterion to determine whether we have case 1 (or 4) or case 2 (or 3):

Lemma 6.10 *Let* p *be a sandwich path. Then* $l_2(p)$ *is even in the cases* 1 *and* 4, *and* $l_2(p)$ *is odd in the cases* 2 *and* 3.

Proof: Let w.l.o.g. $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1}\rangle$ be continued to the right. Let, as usual, v_i be the source of h_i for every $i \in \{0, \ldots, k-1\}$. Since p is a zig–zag path, it holds $\pi_{v_i}(h'_{i-1}) = h_i$

for every even $i \in \{1, \ldots, k-1\}$ and $\overline{\pi_{v_i}}(h'_{i-1}) = h_i$, i.e. $\pi_{v_i}(h_i) = h'_{i-1}$, for every odd $i \in \{1, \ldots, k-1\}$. Moreover, $\pi_{v_i}(h'_{i-1}) \neq h_i$ for every odd $i \in \{1, \ldots, k-1\}$ and $\pi_{v_i}(h_i) \neq h'_{i-1}$ for every even $i \in \{1, \ldots, k-1\}$. Hence if $\pi_{v_t}(h_t) = h'_{t-1}$ and $\pi_{v_s}(h_s) = h'_{s-1}$ (as in case 1), then $t-s$ must be even, if $\pi_{v_t}(h'_{t-1}) = h_t$ and $\pi_{v_s}(h_s) = h'_{s-1}$ (as in case 2), then $t-s$ must be odd, if $\pi_{v_t}(h_t) = h'_{t-1}$ and $\pi_{v_s}(h'_{s-1}) = h_s$ (as in case 3), then $t-s$ must be odd, and if $\pi_{v_t}(h'_{t-1}) = h_t$ and $\pi_{v_s}(h'_{s-1}) = h_s$ (as in case 4), then $t-s$ must be even. ■

6.5 Patches

A **g–patch** (generalized patch) $P = (M, \mathcal{F}, W)$ is a triple where $M = (G, \Pi)$ with $G = (V, H, E, T)$ is a map, $\mathcal{F}$ is a set of faces in M, and $W \subseteq V$ where P satisfies some conditions listed below. If $W \neq \emptyset$, then P is called **marked**. Otherwise P is called **unmarked**. The elements of W are called **marked vertices**. The elements of $\mathcal{F}$ are called **boundary faces**, the other faces in M are called **inner faces**. Vertices, edges, and semi–edges that are incident with a boundary face are called **boundary vertices**, **boundary edges**, and **boundary semi–edges**, respectively. The other vertices, edges, and semi–edges in M are called **inner vertices**, **inner edges**, and **inner semi–edges**, respectively. A g–patch $P = (M, \mathcal{F}, W)$ satisfies the following conditions:

(1) $\mathcal{F} \neq \emptyset$
(2) Every boundary vertex is 2- or 3–valent.
(3) Every inner vertex is 3–valent.
(4) Every vertex is incident with an inner face.
(5) Every edge (and thus every semi–edge) is incident with an inner face.
(6) If $W \neq \emptyset$, then every boundary face is incident with exactly one vertex $v \in W$.

As a consequence, no boundary face is incident with another boundary face because otherwise the common element would be incident only with (two) boundary faces. For the same reason, a face–bridge cannot be an element of a boundary face. Thus every boundary face corresponds to a cycle having the same elements so that for every boundary face, the number of incident vertices is equal to the number of incident edges.

Let v be a 2–valent boundary vertex, and let h and k be the two semi–edges incident with v. Both h and k are *boundary* semi–edges: It holds $\pi_v(h) = k$ and $\pi_v(k) = h$ so that h and k are both incident with the two incident faces $\langle \ldots, h, v, k, \ldots \rangle$ and $\langle \ldots, k, v, h, \ldots \rangle$ of v. Since v is a boundary vertex, one of these faces is a boundary face. So h and k are boundary semi–edges. It follows that every 2–valent boundary vertex is incident with *exactly* one boundary face because due to condition 4, the other face must be an inner face.

Let v be a 3–valent boundary vertex, and let h, j, and k be the three semi–edges incident with v. It holds that two of these semi–edges are boundary semi–edges and the third semi–edge is an inner semi–edge: Let w.l.o.g. $r_\pi(v) = \langle h, j, k \rangle$. Due to the definition of a "boundary vertex", at least one of the three semi–edges is incident with a boundary face. Let w.l.o.g. h be this boundary semi–edge. Then j is also a boundary edge because due to $\pi_v(h) = j$, the semi–edges h and j are elements of a common face $\langle \ldots, h, v, j, \ldots \rangle$. If k was also a boundary semi–edge, then one of the faces containing the sequences $\langle \ldots, j, v, k, \ldots \rangle$ or $\langle \ldots, k, v, h, \ldots \rangle$ would also be a boundary face, and then j or h would not be incident with an inner face. Hence k is an inner edge. The argumentation implies that every 3–valent boundary vertex is incident with *exactly* one boundary face.

Let $\mathcal{F}$ be the set of boundary faces that are obtained by cutting a cubic map according to Section 6.4, i.e. $\mathcal{F} = \bigcup_{i=1}^{x} F_i$ with F_i defined as in Section 6.4 where x is the number of boundary faces that are obtained by the corresponding cut. Then the connected components of B (see Section 6.4) correspond to unmarked g–patches. This can easily be verified: The first three conditions trivially hold. Every semi–edge (and hence every edge and every vertex) is incident with an inner face because every boundary semi–edge appears in the sequence of an "unmodified" face. The unmodified faces are exactly the inner faces.

From the topological point of view, a g–patch of genus g corresponds to a map on an orientable *bordered surface* of genus g (see e.g. [2]) where every boundary face corresponds to a boundary component of the surface.

In the figures, from now on (areas that correspond to) inner faces will be drawn in light grey while (areas that correspond to) boundary faces will be drawn in white. In Section 6.4, it has alrady been done like this. Moreover, boundary faces will in general be drawn as outermost (unbounded) faces.

A boundary edge is called a **convex edge** if and only if all incident vertices are 2–valent. A boundary edge is called a **concave edge** if and only if all incident vertices are 3–valent. A g–patch that does not contain concave edges is called **pseudo–convex**. A pseudo–convex g–patch is called a **patch**. Consequently, the underlying graph of a patch is *not* cubic. A **bunch of g–patches** is a triple $Q = (B, \mathcal{F}, W)$ where B is a bunch of maps, $\mathcal{F}$ is a set of boundary faces such that each connected component of B contains at least one face $F \in \mathcal{F}$, and W is defined as above. A **bunch of patches** $Q = (B, \mathcal{F}, W)$ is a bunch of g–patches containing no concave edges. By assigning the faces in $\mathcal{F}$ and the vertices in W to the appropriate connected components, a bunch of g–patches can easily be transformed into a set of g–patches. Until Section 7.4, we will not use marked vertices.

A g–patch $P = (M, \mathcal{F}, W)$ with $M = (G, \Pi)$, $G = (V, H, E, T)$, is called **planar** or *i*–**connected**, respectively, if and only if M has one of these properties. The **genus** of P is defined as g_M and denoted as g_P. A vertex, edge, semi–edge, or face of M is called a vertex, edge, semi–edge, or face of P, too. For convenience, we will often say "P contains x" if $x \in V$, $x \in H$, $x \in E$, or x is a path or a face in M. Moreover, by n_P, f_P, and e_P, we denote the number of vertices, faces, and edges of P, respectively. Thus $n_P = n_M$, $f_P = f_M$, and $e_P = e_M$. If $Q = (B, \mathcal{F}, W)$ is a bunch of patches, then we define $g_Q = g_B$ (the sum of the genera of the connected components of B), $n_Q = n_B$, $f_Q = f_B$, and $e_Q = e_B$.

Lemma 6.11 *The g–patches that are obtained by cutting a cubic map $M = (G, \Pi)$ along a Petrie path $p = \langle h_0, h'_0, \ldots, h_{k-1}, h'_{k-1} \rangle$ (and transforming the resulting bunch of g–patches into a set of g–patches) are always pseudo–convex.*

Proof: Let as usual v_i, $i \in \{0, \ldots, k-1\}$, be the source of h_i, and let v_k be the target of h_{k-1}. Suppose we have obtained a g–patch $P = ((G', \Pi'), \mathcal{F}, W)$ containing a concave edge $\{b, c\}$ as shown in Figure 6.12. As stated before, since u and v (Figure 6.12) are 3–valent and a, b, c, and d are boundary semi–edges, e and f must be inner semi–edges.

- If p is a Jordan curve path, then the statement of the lemma follows directly from the definition: Let v_i and v_{i+1} (indices modulo k) be the vertices in M that correspond to u and v in P (let w.l.o.g. v_i correspond to u). Since $\pi'_u(a) = b$ and $\pi'_v(c) = d$ in P, it holds $\pi_{v_i}(h'_{i-1}) = h_i$ and $\pi_{v_{i+1}}(h'_i) = h_{i+1}$ in M. But this is a contradition because

$\langle h_{i-1}, h'_{i-1}, h_i, h'_i, h_{i+1}, h'_{i+1}\rangle$ (indices modulo k) is a zig–zag path.

- Let p be a sandwich path or a dumbbell path. Since u and v are both 3–valent, neither u nor v corresponds to any of the vertices u_0, u_s, u_t, u_k, w_s, w_t, and w_k in Section 6.4. Now the argumentation is analogous to the one for the Jordan curve case: Let v_i and v_{i+1} be the vertices in M that correspond to u and v in P (let w.l.o.g. v_i correspond to u). Since neither u nor v corresponds to any of the vertices u_0, u_s, u_t, u_k, w_s, w_t, and w_k in Section 6.4, it holds $i \neq 0$ and $i+1 \neq k$. Since $\pi'_u(a) = b$ and $\pi'_v(c) = d$ in P, it holds $\pi_{v_i}(h'_{i-1}) = h_i$ and $\pi_{v_{i+1}}(h'_i) = h_{i+1}$ in M. This is a contradition because $\langle h_{i-1}, h'_{i-1}, h_i, h'_i, h_{i+1}, h'_{i+1}\rangle$ is a zig–zag path. ■

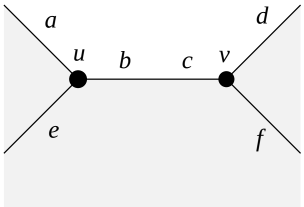

Figure 6.12

A loop l can never be a boundary edge in a (pseudo–convex) patch P except l is the only edge in P: If l is not the only edge in P, then the vertex v which is incident with l is incident with another edge (because P is connected). Thus v is 3–valent and l is a concave edge.

Lemma 6.12 *The patches that are obtained by cutting a cubic map M along a Jordan curve path p do not contain convex edges. The patches that are obtained by cutting a cubic map M along a sandwich or dumbbell path p contain altogether 6 convex edges.*

Proof: Due to Lemma 6.11, the obtained patches do not contain concave edges. Hence every boundary edge is incident with at least one 2–valent boundary vertex.

If p is a Jordan curve path of length k, then every vertex incident with p in M is substituted by one 2–valent and one 3–valent boundary vertex in the patches. Hence the number of 2–valent boundary vertices as well as the number of 3–valent boundary vertices in the patches is k. The number of boundary semi–edges is $4k$ and the number of boundary edges is $2k$. Since the k 2–valent boundary vertices are altogether incident with exactly $2k$ boundary semi–edges and since every boundary edge is incident with at least one 2–valent boundary vertex (so that every boundary edge contains at least one semi–edge which is incident with a 2–valent boundary vertex), every boundary edge contains *exactly* one semi–edge being incident with a 2–valent boundary vertex. Hence the other semi–edge of each edge is incident with a 3–valent vertex. Thus the obtained patches do not contain convex edges.

If p is a sandwich path or a dumbbell path of length k, then $k-3$ vertices incident with p in M are each substituted by one 2–valent and one 3–valent boundary vertex in the patches, while the start vertex and the end vertex of p are substituted by altogether six 2–valent boundary vertices. Hence the number of 2–valent boundary vertices in the patches is $(k-3)+6 = k+3$ and the number of 3–valent boundary vertices is $k-3$. The number of boundary semi–edges is $4k$ and the number of boundary edges is $2k$. Since each 3–valent boundary vertex is incident with exactly 2 boundary semi–edges and one inner semi–edge (see above), the 3–valent boundary vertices are altogether incident with exactly $2k-6$ boundary semi–edges so that $4k-(2k-6) = 2k+6$ boundary semi–edges are incident with 2–valent boundary vertices. Let A be the set of all boundary semi–edges whose source is 3–valent and B be the set of all boundary semi–edges whose source is 2–valent. Thus $|A| + 12 = |B|$. Every edge e containing a semi–edge of A contains also a semi–edge of B because otherwise e would be a concave edge. Hence there are 12 semi–edges in B whose inverse is not in A. These 12 semi–edges are elements of 6 edges that are consequently convex edges. ∎

After cutting a cubic map M into a bunch of patches $Q = (B, \mathcal{F}, W)$ along a Petrie path p, the bunch of patches can easily be transformed into a set $A(M, p)$ of patches by assigning the faces in $\mathcal{F}$ and the vertices in W to the appropriate connected components of B. The patches in $A(M, p)$ are called the **ancestors** of M with respect to p. The cubic map M is called a **descendant** of a patch Q if and only if Q is an element of $A(M, p)$ for some Petrie path p in M.

6.6 Cutting a patch along a zig–zag path

Patches are cut in a similar way as cubic maps. A patch P is cut only if it contains a 3–valent boundary vertex. A patch that does not contain 3–valent boundary vertices is called a **tile**. In a tile P, the 2–valent boundary vertices are incident with one common cycle and P does not contain inner vertices (because P is connected). Thus P contains exactly two faces where one face is an inner face (because every edge must be incident with an inner face) and one face is a boundary face (due to the definition). Every vertex is a boundary vertex and every edge is a boundary edge. The **degree** of a tile is defined as the degree of the inner face (which is equal to the length of the cycle). Figure 6.13 shows tiles of degree 1, 2, 3, 4, and 5. A tile of degree d is called a **d–tile**.

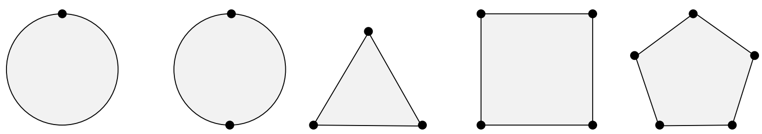

Figure 6.13: tiles

Here is how to **cut** a patch P: Using a 3–valent boundary vertex as the start vertex and the incident inner edge as the start edge, the patch P is cut along a zig–zag path which is continued to the right. The path is maximally extended within the set of inner edges, that is: The zig–zag path is traced until either a boundary vertex is approached, or a vertex that has already been incident with the path before is approached. In either case, this vertex is the end vertex of the path. The first case is called an **intersection cut** if the start vertex and the end vertex of the corresponding path are incident with the same boundary face, and it is called a **linkage cut** if the start vertex and the end vertex are incident with different boundary faces. The second case is called an **enclosure cut**. Figures 6.14, 6.15, and 6.16 show concrete examples and schemes for all cases. Depending on the case, the zig–zag path is called an **intersection path**, a **linkage path**, or an **enclosure path**, respectively. In either case, a set $\{F_1, \ldots, F_r\}$ ($r \in \{1, 2\}$) of boundary faces in P is replaced by a different set $\{F_3, \ldots, F_s\}$ ($s \in \{3, 4\}$) of boundary faces in the resulting bunch of g–patches.

The details are provided in the following sections.

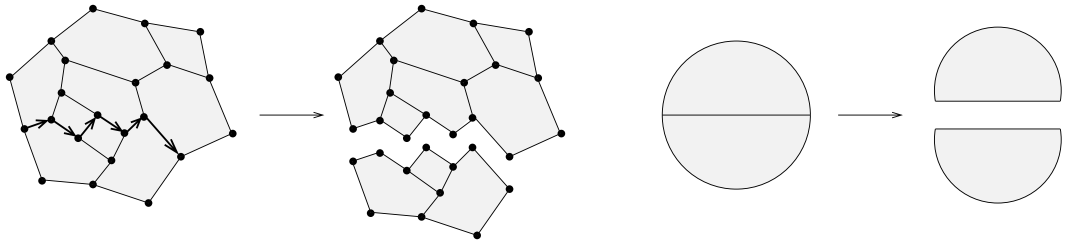

Figure 6.14: an intersection cut

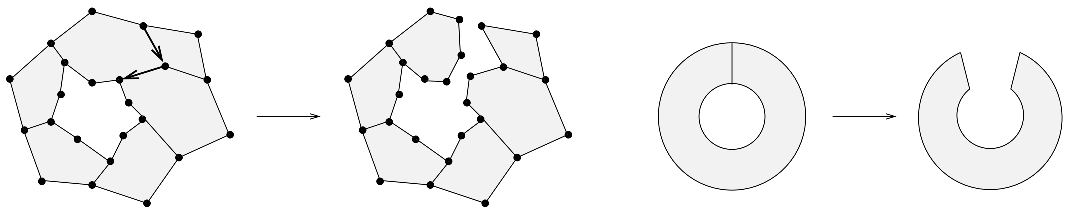

Figure 6.15: a linkage cut

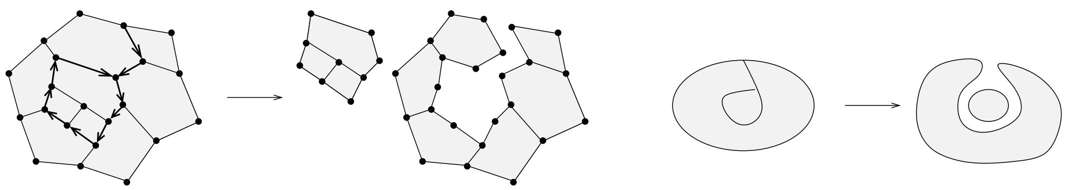

Figure 6.16: an enclosure cut

6.6.1 Intersection cut

Let $F_1 = \langle v_0, h_0, h'_0, \dots, v_{k-1}, h_{k-1}, h'_{k-1}\rangle$ be a boundary face of a patch $P = (M, \mathcal{F}, W)$ with $M = (G, \Pi)$ and $G = (V, H, E, T)$ where v_0 is 3–valent. Let h_k be the inner semi–edge which is incident with v_0, and let $p = \langle h_k, h'_k, \dots, h_{t-1}, h'_{t-1}\rangle$ be a zig–zag path which is continued to the right such that all elements $e_i := \{h_i, h'_i\}$ $(i \in \{k, \dots, t-1\})$ are inner edges and h'_{t-1} is incident with the boundary vertex v_s for some $s \in \{2, \dots, k-2\}$ ($s \in \{1, k-1\}$ is impossible because since P is pseudo–convex, both v_1 and v_{k-1} must be 2–valent). Let v_i be the source of h_i (and of h'_{i-1}) for every $i \in \{k, \dots, t-1\}$. It holds $v_k = v_0$. Let v_t be the target of h_{t-1}. So $v_t = v_s$. Let j_i, $i \in \{k+1, \dots, t-1\}$, be the third semi–edge besides h_i and h'_{i-1} having the source v_i. Then it holds $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$ if $i - k$ is odd and $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$ if $i - k$ is even. Moreover, it holds $r_\pi(v_k) = \langle h'_{k-1}, h_0, h_k\rangle$ and $r_\pi(v_t) = \langle h'_{t-1}, h'_{s-1}, h_s\rangle$.

Let $(V \cup H) \cap (\{u_k, \dots, u_t\} \cup \{w_k, \dots, w_t\} \cup \{a_k, a'_k, \dots, a_{t-1}, a'_{t-1}\} \cup \{b_k, b'_k, \dots, b_{t-1}, b'_{t-1}\}) = \emptyset$. By cutting the patch P along p, we obtain a bunch of g–patches $Q = (B, \mathcal{F}', W')$ with $B = (G', \Pi')$ and $G' = (V', H', E', T')$ having the following properties:

- $V' = (V - \{v_k, \dots, v_t\}) \cup \{u_k, \dots, u_t\} \cup \{w_k, \dots, w_t\}$
- $H' = (H - \{h_k, h'_k, \dots, h_{t-1}, h'_{t-1}\}) \cup \{a_k, a'_k, \dots, a_{t-1}, a'_{t-1}\} \cup \{b_k, b'_k, \dots, b_{t-1}, b'_{t-1}\}$
- $E' = (E - \{e_k, \dots, e_{t-1}\}) \cup \{c_k, \dots, c_{t-1}\} \cup \{d_k, \dots, d_{t-1}\}$ where $c_i := \{a_i, a'_i\}$ and $d_i := \{b_i, b'_i\}$ for every $i \in \{k, \dots, t-1\}$
- For $i \in \{k, \dots, t\}$, the vertex v_i is substituted by u_i and w_i. For $i \in \{k, \dots, t-1\}$, the

semi–edge h_i is substituted by a_i and b_i, and the semi–edge h'_{i-1} is substituted by a'_{i-1} and b'_{i-1}. We will perform the substitutions in a way that by tracing the cut zig–zag path, the u_i, a_i, and a'_i are always on the right side and the w_i, b_i, and b'_i are always on the left side (see Figure 6.17 for an example with $k = 12$, $s = 8$, and $t = 18$).

– For all vertices $v \in V - \{v_k, \ldots, v_t\}$, we define $\pi'_v = \pi_v$.
– We define $r_{\pi'}(u_k) = \langle h'_{k-1}, a_k\rangle$, $r_{\pi'}(w_k) = \langle h_0, b_k\rangle$, $r_{\pi'}(u_t) = \langle a'_{t-1}, h_s\rangle$, and $r_{\pi'}(w_t) = \langle b'_{t-1}, h'_{s-1}\rangle$.
– Let $i \in \{k+1, \ldots, t-1\}$. As stated above, it holds either $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$ (if $i-k$ is even) or $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$ (if $i-k$ is odd). If $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, j_i, a'_{i-1}\rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}\rangle$, and if $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, a'_{i-1}\rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}, j_i\rangle$.

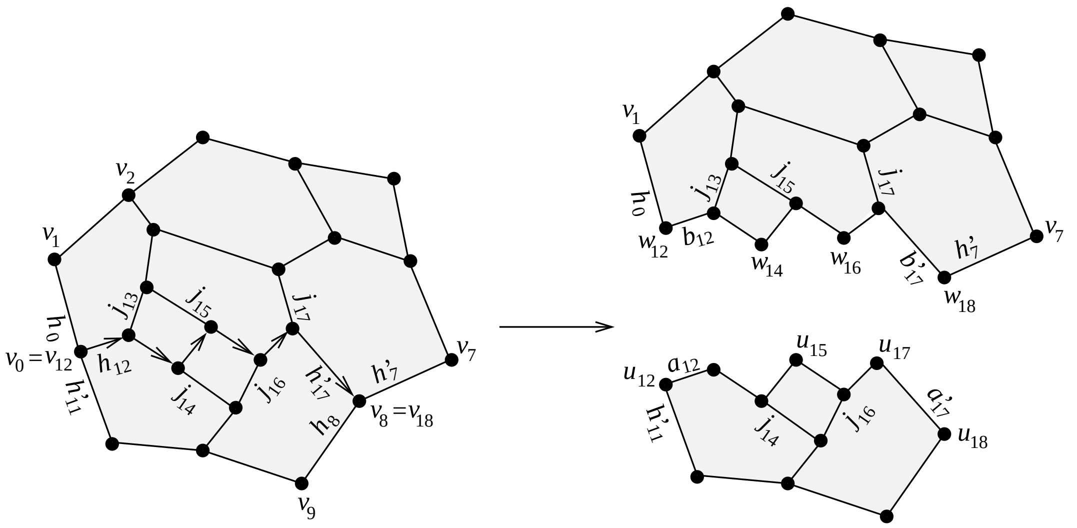

Figure 6.17

As always, T' is implied by E' and π'. The boundary face F_1 in P is substituted by two new boundary faces F_3 and F_4 in Q:

- $F_3 = \langle u_k, a_k, a'_k, \ldots, u_{t-1}, a_{t-1}, a'_{t-1}, u_t, h_s, h'_s, v_{s+1}, h_{s+1}, h'_{s+1}, \ldots, v_{k-1}, h_{k-1}, h'_{k-1}\rangle$
- $F_4 = \langle w_k, h_0, h'_0, v_1, h_1, h'_1, \ldots, v_{s-1}, h_{s-1}, h'_{s-1}, w_t, b'_{t-1}, b_{t-1}, \ldots, w_{k+1}, b'_k, b_k\rangle$

Hence we define $\mathcal{F}' = (\mathcal{F} - \{F_1\}) \cup \{F_3, F_4\}$. All other faces in Q correspond to faces in P in the usual way. If P is marked, then W' must contain all vertices of $W - \{v_0, \ldots, v_{k-1}\}$. Moreover, W' must contain exactly one vertex incident with F_3 and exactly one vertex incident with F_4. Further vertices are not allowed in W'. We will consider the marks in Section 7.4.

B contains one or two connected components: The vertices $u_k, \ldots, u_t$ are in one common component and the the vertices $w_k, \ldots, w_t$ are in one common component. These components might be identical. For every other vertex, the argumentation is analogous to the one given in Section 6.4.1. Thus there are at most 2 connected components in B.

Let $l = t - k$ be the length of the zig–zag path p. It holds $f_Q = f_P + 1$ (see above). Moreover, the definitions of V' and E' imply that $n_Q = n_P - (l+1) + 2 \cdot (l+1) = n_P + l + 1$ and $e_Q = e_P - l + 2l = e_P + l$. For g_Q, the following lemma holds:

Lemma 6.13 *If B is connected, then $g_Q = g_P - 1$. If B contains two connected components, then $g_Q = g_P$.*

Proof: Euler's formula yields $n_P + f_P - e_P = 2 - 2g_P$.

- Let B be connected, i.e. B is a map. Euler's formula implies

$$\begin{array}{lr} & n_Q + f_Q - e_Q = 2 - 2g_Q \\ \Leftrightarrow & (n_P + l + 1) + (f_P + 1) - (e_P + l) = 2 - 2g_Q \\ \Leftrightarrow & n_P + f_P - e_P = 2 - 2(g_Q + 1) \\ \Leftrightarrow & 2 - 2g_P = 2 - 2(g_Q + 1) \\ \Leftrightarrow & g_P = g_Q + 1 \end{array}$$

- Let B contain two connected components M_1 and M_2 (which are maps). It holds $n_Q = n_{M_1} + n_{M_2}$, $e_Q = e_{M_1} + e_{M_2}$, $f_Q = f_{M_1} + f_{M_2}$, $g_Q = g_{M_1} + g_{M_2}$, $n_{M_1} + f_{M_1} - e_{M_1} = 2 - 2g_{M_1}$, and $n_{M_2} + f_{M_2} - e_{M_2} = 2 - 2g_{M_2}$. Hence

$$\begin{array}{lr} & n_{M_1} + f_{M_1} - e_{M_1} + n_{M_2} + f_{M_2} - e_{M_2} = 2 - 2g_{M_1} + 2 - 2g_{M_2} \\ \Leftrightarrow & n_Q + f_Q - e_Q = 4 - 2g_Q \\ \Leftrightarrow & (n_P + l + 1) + (f_P + 1) - (e_P + l) = 4 - 2g_Q \\ \Leftrightarrow & n_P + f_P - e_P = 2 - 2g_Q \\ \Leftrightarrow & 2 - 2g_P = 2 - 2g_Q \\ \Leftrightarrow & g_P = g_Q \end{array}$$

■

6.6.2 Linkage cut

Let $F_1 = \langle v_0, h_0, h'_0, \ldots, v_{k-1}, h_{k-1}, h'_{k-1}\rangle$ be a boundary face of a patch $P = (M, \mathcal{F}, W)$ with $M = (G, \Pi)$ and $G = (V, H, E, T)$ where v_0 is 3–valent. Let h_k be the inner semi–edge which is incident with v_0, and let $p = \langle h_k, h'_k, \ldots, h_{t-1}, h'_{t-1}\rangle$ be a zig–zag path which is continued to the right such that all elements $e_i := \{h_i, h'_i\}$ ($i \in \{k, \ldots, t-1\}$) are inner edges and the end vertex v_t of p is a boundary vertex incident with a boundary face $F_2 \neq F_1$ where $F_2 = \langle v_t, h_t, h'_t, \ldots, v_{s-1}, h_{s-1}, h'_{s-1}\rangle$ for some $s > t$. For every $i \in \{k, \ldots, t-1\}$, let v_i be the source of h_i (and of h'_{i-1}). Let j_i, $i \in \{k+1, \ldots, t-1\}$, be the third semi–edge besides h_i and h'_{i-1} having the source v_i. Then it holds $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$ if $i - k$ is odd and $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$ if $i - k$ is even. Moreover, it holds $v_k = v_0$, $r_\pi(v_k) = \langle h'_{k-1}, h_0, h_k\rangle$, and $r_\pi(v_t) = \langle h'_{t-1}, h'_{s-1}, h_s\rangle$.

Let $(V \cup H) \cap (\{u_k, \ldots, u_t\} \cup \{w_k, \ldots, w_t\} \cup \{a_k, a'_k, \ldots, a_{t-1}, a'_{t-1}\} \cup \{b_k, b'_k, \ldots, b_{t-1}, b'_{t-1}\}) = \emptyset$. By cutting the patch P along p, we obtain a bunch of g–patches $Q = (B, \mathcal{F}', W')$ with $B = (G', \Pi')$ and $G' = (V', H', E', T')$ having the following properties:

- $V' = (V - \{v_k, \ldots, v_t\}) \cup \{u_k, \ldots, u_t\} \cup \{w_k, \ldots, w_t\}$
- $H' = (H - \{h_k, h'_k, \ldots, h_{t-1}, h'_{t-1}\}) \cup \{a_k, a'_k, \ldots, a_{t-1}, a'_{t-1}\} \cup \{b_k, b'_k, \ldots, b_{t-1}, b'_{t-1}\}$
- $E' = (E - \{e_k, \ldots, e_{t-1}\}) \cup \{c_k, \ldots, c_{t-1}\} \cup \{d_k, \ldots, d_{t-1}\}$ where $c_i := \{a_i, a'_i\}$ and $d_i := \{b_i, b'_i\}$ for every $i \in \{k, \ldots, t-1\}$

- The vertices v_i ($i \in \{k, \ldots, t\}$) and the semi–edges h_i and h'_{i-1} ($i \in \{k, \ldots, t-1\}$) are substituted as described in the intersection case (see Figure 6.18 for an example with $k = 12$, $t = 14$, and $s = 24$).
  - For all vertices $v \in V - \{v_k, \ldots, v_t\}$, we define $\pi'_v = \pi_v$.
  - We define $r_{\pi'}(u_k) = \langle h'_{k-1}, a_k \rangle$, $r_{\pi'}(w_k) = \langle h_0, b_k \rangle$, $r_{\pi'}(u_t) = \langle a'_{t-1}, h_t \rangle$, and $r_{\pi'}(w_t) = \langle b'_{t-1}, h'_{s-1} \rangle$.
  - Let $i \in \{k+1, \ldots, t-1\}$. As usual, if $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1} \rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, j_i, a'_{i-1} \rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1} \rangle$, and if $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i \rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, a'_{i-1} \rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}, j_i \rangle$.

As always, T' is implied by E' and π'.

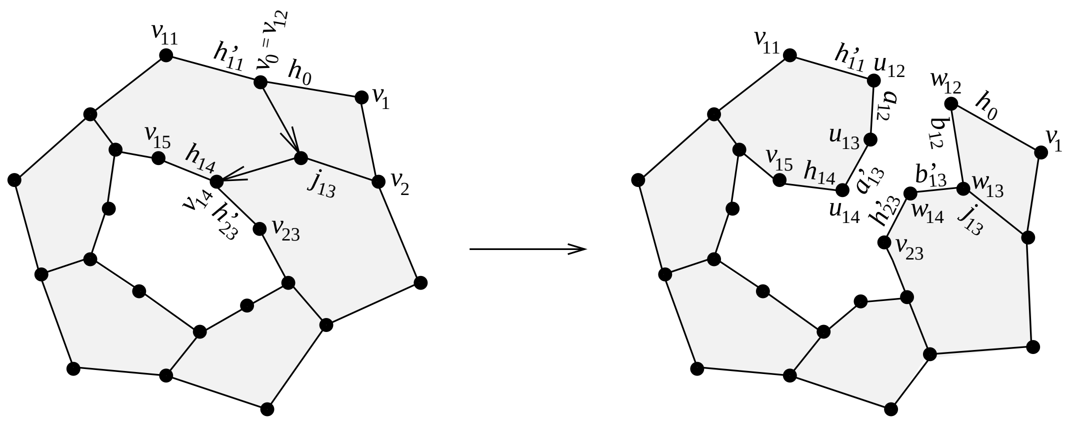

Figure 6.18

The boundary faces F_1 and F_2 are replaced by one new boundary face $F_3 = \langle w_k, h_0, h'_0, v_1, h_1, h'_1, \ldots, v_{k-1}, h_{k-1}, h'_{k-1}, u_k, a_k, a'_k, u_{k+1}, a_{k+1}, a'_{k+1}, \ldots, u_{t-1}, a_{t-1}, a'_{t-1}, u_t, h_t, h'_t, \ldots, u_{s-1}, h_{s-1}, h'_{s-1}, w_t, b'_{t-1}, b_{t-1}, w_{t-1}, \ldots, b'_{k+1}, b_{k+1}, w_{k+1}, b'_k, b_k \rangle$. Hence $\mathcal{F}' = (\mathcal{F} - \{F_1, F_2\}) \cup \{F_3\}$. All other faces in Q correspond to faces in P in the usual way. If P is marked, then W' must contain all vertices of $W - \{v_0, \ldots, v_{k-1}\} - \{v_t, \ldots, v_{s-1}\}$. Moreover, W' must contain exactly one vertex incident with F_3. Further vertices are not allowed in W'. We will consider the marks in Section 7.4.

B is connected: The vertices $u_k, \ldots, u_t$ and $w_k, \ldots, w_t$ are in one common component. For every other vertex, the argumentation is analogous to the one given in Section 6.4.1.

Let $l = t - k$ be the length of the zig–zag path p. It holds $f_Q = f_P - 1$. Moreover, the definitions of V' and E' imply $n_Q = n_P + l + 1$ and $e_Q = e_P + l$. For g_Q, it holds:

Lemma 6.14 $g_Q = g_P$.

Proof: Euler's formula yields $n_P + f_P - e_P = 2 - 2g_P$ and $n_Q + f_Q - e_Q = 2 - 2g_Q$.

$$\begin{aligned}
& & n_Q + f_Q - e_Q &= 2 - 2g_Q \\
&\Leftrightarrow & (n_P + l + 1) + (f - 1) - (e_P + l) &= 2 - 2g_Q \\
&\Leftrightarrow & n_P + f_P - e_P &= 2 - 2g_Q \\
&\Leftrightarrow & 2 - 2g_P &= 2 - 2g_Q \\
&\Leftrightarrow & g_P &= g_Q
\end{aligned}$$

■

6.6.3 Enclosure cut

Let $F_1 = \langle v_0, h_0, h'_0, \ldots, v_{k-1}, h_{k-1}, h'_{k-1}\rangle$ be a boundary face of a patch $P = (M, \mathcal{F}, W)$ with $M = (G, \Pi)$ and $G = (V, H, E, T)$ where v_0 is 3–valent. Let h_k be the inner semi–edge which is incident with v_0, and let $p = \langle h_k, h'_k, \ldots, h_{t-1}, h'_{t-1}\rangle$ be a zig–zag path which is continued to the right such that all elements $e_i := \{h_i, h'_i\}$ ($i \in \{k, \ldots, t-1\}$) are inner edges and h'_{t-1} is incident with h_s for some $s \in \{k+1, \ldots, t-1\}$. Let v_i, $i \in \{k, \ldots, t-1\}$, be the source of h_i (and of h'_{i-1}), It holds $v_k = v_0$. Let j_i, $i \in \{k+1, \ldots, t-1\} - \{s\}$, be the third semi–edge besides h_i and h'_{i-1} having the source v_i. The path $\langle h_k, h'_k, \ldots, h_{s-1}, h'_{s-1}\rangle$ is called the **enclosure entry** and the path $\langle h_s, h'_s, \ldots, h_{t-1}, h'_{t-1}\rangle$ is called the **enclosure cycle**. The length of the enclosure entry is called the **enclosure entry length** and the length of the enclosure cycle is called the **enclosure cycle length**.

Let $(V \cup H) \cap (\{u_k, \ldots, u_t\} \cup \{w_k, \ldots, w_{t-1}\} \cup \{a_k, a'_k, \ldots, a_{t-1}, a'_{t-1}\} \cup \{b_k, b'_k, \ldots, b_{t-1}, b'_{t-1}\}) = \emptyset$. By cutting the patch P along p, we obtain a bunch of g–patches $Q = (B, \mathcal{F}', W')$ with $B = (G', \Pi')$ and $G' = (V', H', E', T')$ having the following properties:

- $V' = (V - \{v_k, \ldots, v_{t-1}\}) \cup \{u_k, \ldots, u_t\} \cup \{w_k, \ldots, w_{t-1}\}$
- $H' = (H - \{h_k, h'_k, \ldots, h_{t-1}, h'_{t-1}\}) \cup \{a_k, a'_k, \ldots, a_{t-1}, a'_{t-1}\} \cup \{b_k, b'_k, \ldots, b_{t-1}, b'_{t-1}\}$
- $E' = (E - \{e_k, \ldots, e_{t-1}\}) \cup \{c_k, \ldots, c_{t-1}\} \cup \{d_k, \ldots, d_{t-1}\}$ where $c_i := \{a_i, a'_i\}$ and $d_i := \{b_i, b'_i\}$ for every $i \in \{k, \ldots, t-1\}$
- For $i \neq s$, it holds $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i\rangle$ if $i - k$ is odd and $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1}\rangle$ if $i - k$ is even. Moreover, it holds $r_\pi(v_k) = \langle h'_{k-1}, h_0, h_k\rangle$. We must distinguish two different cases:
  (1) $r_\pi(v_s) = \langle h'_{t-1}, h'_{s-1}, h_s\rangle$
  (2) $r_\pi(v_s) = \langle h'_{t-1}, h_s, h'_{s-1}\rangle$

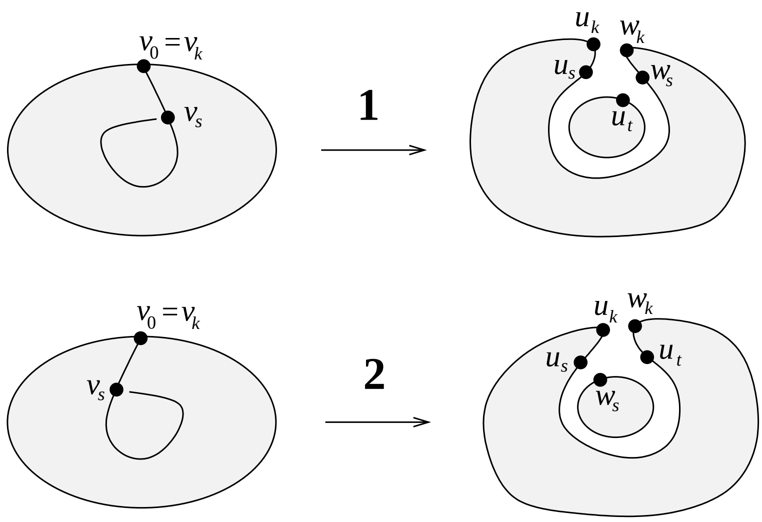

Figure 6.19

The vertices v_i ($i \in \{k, \ldots, t-1\} - \{s\}$) and the semi–edges h_i and h'_{i-1} ($i \in \{k, \ldots, t-1\}$) are substituted as described in the intersection case. v_s is substituted by u_s, w_s, and u_t. Figure 6.19 shows sketches for both cases. Figure 6.20 shows an example for case 1 with $k = 12$, $t = 22$, and $s = 14$.

– For all vertices $v \in V - \{v_k, \ldots, v_{t-1}\}$, we define $\pi'_v = \pi_v$.

- We define $r_{\pi'}(u_k) = \langle h'_{k-1}, a_k \rangle$ and $r_{\pi'}(w_k) = \langle h_0, b_k \rangle$. Moreover, in case 1 we define $r_{\pi'}(u_s) = \langle a'_{s-1}, b'_{t-1} \rangle$, $r_{\pi'}(w_s) = \langle b'_{s-1}, b_s \rangle$, and $r_{\pi'}(u_t) = \langle a'_{t-1}, a_s \rangle$. In case 2, we define $r_{\pi'}(u_s) = \langle a'_{s-1}, a_s \rangle$, $r_{\pi'}(w_s) = \langle b_s, b'_{t-1} \rangle$, and $r_{\pi'}(u_t) = \langle a'_{t-1}, b'_{s-1} \rangle$.
- Let $i \in \{k+1, \nearrow, t-1\} - \{s\}$. As stated above, it holds either $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1} \rangle$ (if $i-k$ is even) or $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i \rangle$ (if $i-k$ is odd). If $r_\pi(v_i) = \langle h_i, j_i, h'_{i-1} \rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, j_i, a'_{i-1} \rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1} \rangle$, and if $r_\pi(v_i) = \langle h_i, h'_{i-1}, j_i \rangle$, then we define $r_{\pi'}(u_i) = \langle a_i, a'_{i-1} \rangle$ and $r_{\pi'}(w_i) = \langle b_i, b'_{i-1}, j_i \rangle$.

As always, T' is implied by E' and π'.

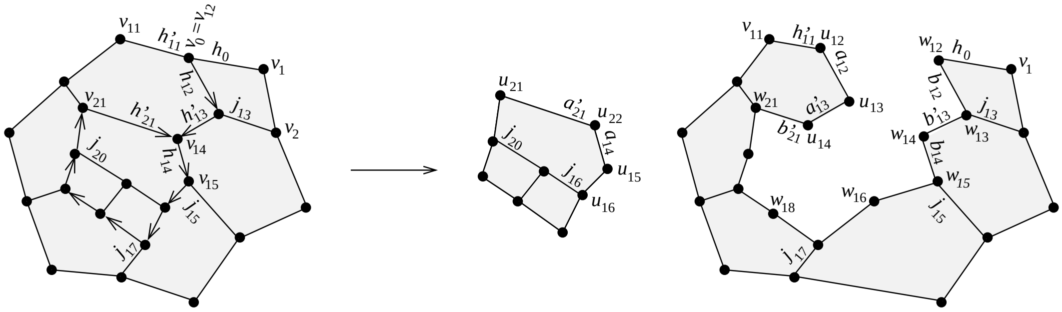

Figure 6.20

The boundary face F_1 in P is substituted by two new boundary faces F_3 and F_4 in Q. In case 1, it holds $F_3 = \langle w_k, h_0, h'_0, v_1, h_1, h'_1, \nearrow, v_{k-1}, h_{k-1}, h'_{k-1}, u_k, a_k, a'_k, \nearrow, u_{s-1}, a_{s-1}, a'_{s-1}, u_s, b'_{t-1}, b_{t-1}, w_{t-1}, \searrow, b'_k, b_k \rangle$ and $F_4 = \langle a_s, a'_s, u_{s+1}, a_{s+1}, a'_{s+1}, \nearrow, u_{t-1}, a_{t-1}, a'_{t-1}, u_t \rangle$. In case 2, it holds $F_3 = \langle w_k, h_0, h'_0, v_1, h_1, h'_1, \nearrow, v_{k-1}, h_{k-1}, h'_{k-1}, u_k, a_k, a'_k, \nearrow, u_{t-1}, a_{t-1}, a'_{t-1}, u_t, b'_{s-1}, b_{s-1}, w_{s-1}, \searrow, b'_k, b_k \rangle$ and $F_4 = \langle b'_{t-1}, b_{t-1}, w_{t-1}, \searrow, b'_s, b_s, w_s \rangle$. Hence $\mathcal{F}' = (\mathcal{F} - \{F_1\}) \cup \{F_3, F_4\}$. All other faces in Q correspond to faces in P in the usual way. If P is marked, then W' must contain all vertices of $W - \{v_0, \nearrow, v_{k-1}\}$. Moreover, W' must contain exactly one vertex incident with F_3 and exactly one vertex incident with F_4. Further vertices are not allowed in W'. We will consider the marks in Section 7.4. B contains one or two connected components where the argumentation is the same as usual.

Let $l = t-k$ be the length of the zig–zag path p. It holds $f_Q = f_P + 1$. Moreover, the definitions of V' and E' imply that $n_Q = n_P - l + (l+1) + l = n_P + l + 1$ and $e_Q = e_P - l + 2l = e_P + l$. For g_Q, it holds:

Lemma 6.15 *If B is connected, then $g_Q = g_P - 1$. If B contains two connected components, then it holds $g_Q = g_P$.*

Proof: Identical to the proof of Lemma 6.13. ∎

Figure 6.21 shows an example how a patch is completely cut into a set of tiles.

6.7 Properties

As mentioned before, after cutting a patch P into a bunch of g–patches $Q = (B, \mathcal{F}', W')$ along a zig–zag path p, Q can easily be transformed into a set $A(P, p)$ of g–patches by assigning

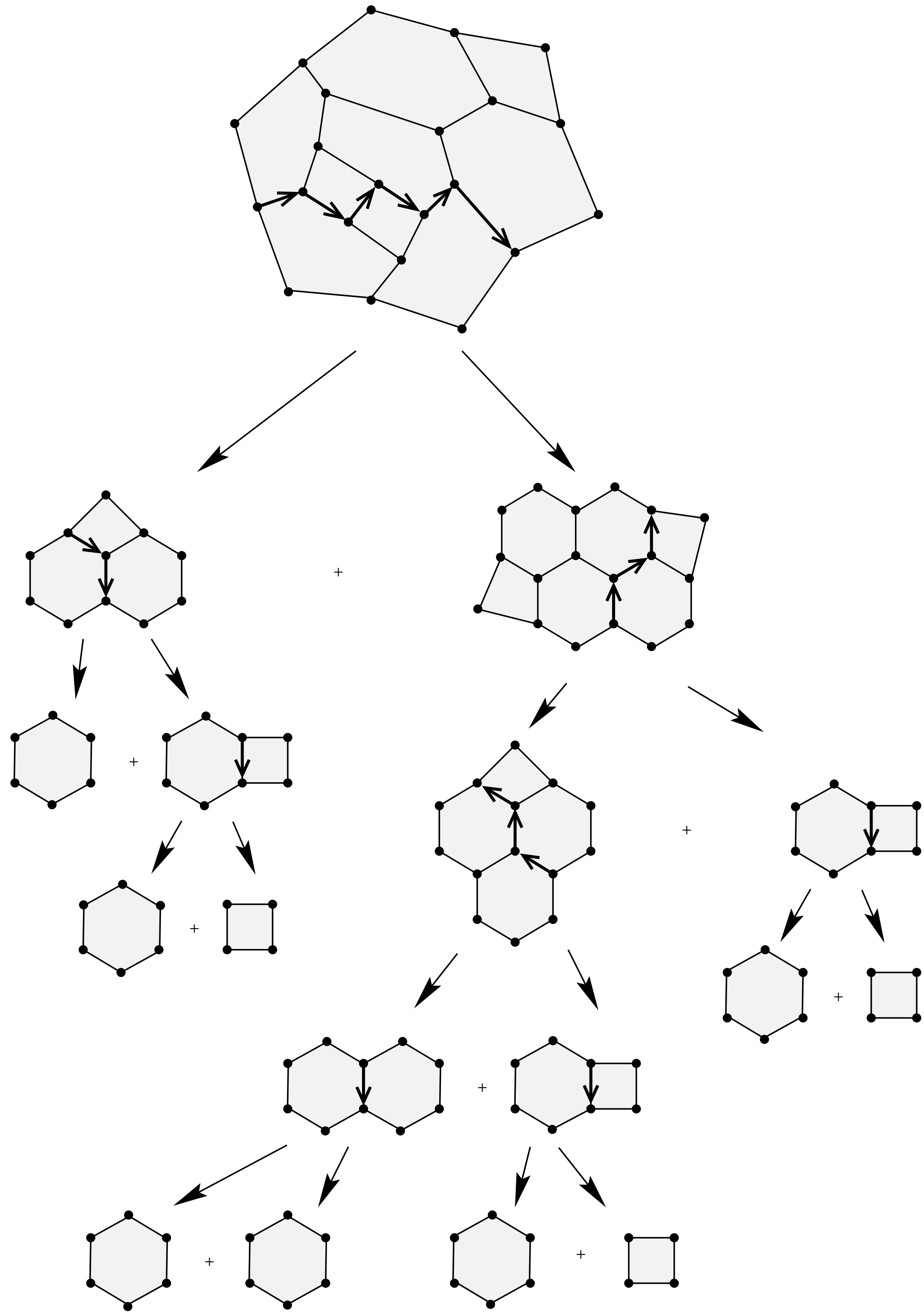

Figure 6.21

the faces in $\mathcal{F}'$ and the vertices in W' to the appropriate connected components of B. The g–patches in $A(P,p)$ are called **ancestors** of P with respect to p. The patch P is called a **descendant** of a g–patch Q if and only if $Q \in A(P,p)$ for some zig–zag path p.

Lemma 6.16 *The ancestors of a patch $P = ((G,\Pi),\mathcal{F},W)$ are pseudo–convex.*

Proof: The proof is similar to the proof of Lemma 6.11. Let p be the zig–zag path which is used to cut P. Suppose we have a concave edge $\{b,c\}$ in an ancestor $Q = ((G,\Pi'),\mathcal{F}',W') \in A(P,p)$ as shown in Figure 6.12. Since u and v are both 3–valent, neither u nor v corresponds to any of the vertices u_k, u_s, u_t, w_k, w_s, w_t in Section 6.6. Let v_i and v_{i+1} be the vertices in P that correspond to u and v (let w.l.o.g. v_i correspond to u) in Q. Since neither u nor v corresponds to any of the vertices u_k, u_s, u_t, w_k, w_s, w_t in Section 6.6, it holds $i \neq k$ and $i+1 \notin \{s,t\}$. Since $\pi'_u(a) = b$ and $\pi'_v(c) = d$, it holds $\pi_{v_i}(h'_{i-1}) = h_i$ and $\pi_{v_{i+1}}(h'_i) = h_{i+1}$ in P. This is a contradition because $\langle h_{i-1}, h'_{i-1}, h_i, h'_i, h_{i+1}, h'_{i+1}\rangle$ is a zig–zag path. ■

As a consequence, the ancestors of a patch are themselves patches. Moreover, the ancestors of a patch P have the following properties:

- The sum of inner faces contained in the ancestors of P equals the number of inner faces contained in P because all inner faces remain “unmodified” as indicated in Section 6.6.
- For the same reason, for every $d \in \mathbb{N}$ it holds that if an ancestor of P contains a d–gon, then P contains a d–gon, too, and if P contains exactly m inner d–gons, then for every *valid* zig–zag path p (i.e. p matches one of the descriptions in Section 6.6), the patches in $A(P,p)$ contain altogether m inner d–gons, too.
- As a consequence of Lemma 6.6, 6.7, 6.8, 6.13, 6.14, and 6.15, by cutting a *planar* map along a Petrie path, one always obtains the maximum number of (planar) patches (i.e. one obtains 2 patches in the Jordan curve case and 3 patches in the sandwich or dumbbell case) each containing one boundary face, and by cutting a planar patch having one boundary face, one always obtains 2 planar patches having one boundary face each. In this case, a linkage cut is impossible.

Lemma 6.17 *If a patch P contains k convex edges, then the patches in $A(P,p)$ contain altogether $k' = k+6$ convex edges for every valid zig–zag path p.*

Proof: If p is an intersection path or a linkage path of length l, then every inner vertex incident with p in P is substituted by one 2–valent and one 3–valent boundary vertex in $A(P,p)$. The start vertex and the end vertex of p (which are both 3–valent boundary vertices in P) are each substituted by a pair of 2–valent boundary vertices in $A(P,p)$. Let s be the number of 2–valent boundary vertices and let t be the number of 3–valent boundary vertices in P. Let s' and t' be the corresponding sums for $A(P,p)$. Thus, it holds $t' = t-2+(l-1)$ and $s' = s+4+(l-1)$. Let C be the set of all boundary semi–edges in P whose source is 3–valent and D be the set of all boundary semi–edges in P whose source is 2–valent. Since every boundary vertex is incident with exactly 2 boundary semi–edges, it holds $|C| = 2t$ and $|D| = 2s$. Every edge e containing a semi–edge in C contains a semi–edge in D, too, because otherwise e would be a concave edge. Hence there are $2(s-t)$ semi–edges in D whose inverse is not in C. These semi–edges are elements of $s-t$ edges that are consequently convex edges. Hence $k = s-t$. With the same arguments, we obtain $k' = s'-t'$. Hence $k' = s'-t' = (s+4+(l-1)) - (t-2+(l-1)) = s-t+6 = k+6$.

If p is an enclosure path of length l, then each inner vertex incident with p in P except the end vertex (i.e. $l-2$ inner vertices) is substituted by one 2–valent and one 3–valent boundary vertex in $A(P,p)$. The start vertex of p (which is a 3–valent boundary vertex in P) is substituted by two 2–valent boundary vertices in $A(P,p)$, and the end vertex of p (which is an inner vertex in P) is substituted by three 2–valent boundary vertices in $A(P,p)$. Hence with s, t, s', t' defined as above, it holds $t' = t - 1 + (l-2)$ and $s' = s + 2 + 3 + (l-2)$. With $k = s - t$ and $k' = s' - t'$ as before, we obtain $k' = s' - t' = s + 5 + (l-2) - (t - 1 + (l-2)) = s - t + 6 = k + 6$. ■

The **size** $s(P)$ of a patch P is defined by the triple $s(P) = (f'_P, g_P, b_P)$ where f'_P is the number of *inner* faces in P and b_P is the number of *boundary* faces in P. A patch P is called **smaller** than a patch Q (indicated by $s(P) < s(Q)$) if and only if the sequence $\langle f'_P, g_P, b_P\rangle$ is lexicographically smaller than the sequence $\langle f'_Q, g_Q, b_P\rangle$, i.e. if one of the following conditions is true:

- $f'_P < f'_Q$
- $f'_P = f'_Q$ and $g_P < g_Q$
- $f'_P = f'_Q$ and $g_P = g_Q$ and $b_P < b_Q$

A patch Q is called **bigger** than a patch P if and only if P is smaller than Q.

Lemma 6.18 *Every ancestor Q of a patch P is smaller than P.*

Proof: Let $s(P) = (f'_P, g_P, b_P)$ and $s(Q) = (f'_Q, g_Q, b_Q)$. Let p be a zig–zag path such that $Q \in A(P,p)$. It holds $|A(P,p)| \leq 2$.

- If $|A(P,p)| = 1$, i.e. $A(P,p) = \{Q\}$, then $f'_P = f'_Q$ because every inner face of Q corresponds to an inner face of P. In the intersection and enclosure cases, it holds $g_Q = g_P - 1 < g_P$. In the linkage case, it holds $g_Q = g_P$ and $b_Q = b_P - 1 < b_P$. Hence Q is smaller than P.
- Let $|A(P,p)| = 2$, i.e. $A(P,p) = \{Q, R\}$ for some patch R. Since every vertex in R must be incident with an inner face, it holds $f'_R > 0$. Since every inner face of R and Q corresponds to an inner face of P, it holds $f'_P = f'_Q + f'_R > f'_Q$. Hence Q is smaller than P. ■

Lemma 6.19 *Every patch which is not a tile can be cut along a zig–zag path.*

Proof: This is trivial because due to the definition, a patch which is not a tile contains a 3–valent boundary vertex, and hence the rules for cutting a patch can be applied. For every 3–valent boundary vertex v in a patch, it holds that among all zig–zag paths whose start vertex is v, exactly one path matches one of the descriptions given in Sections 6.6.1, 6.6.2, and 6.6.3. ■

Lemma 6.20 *The smallest possible patches are exactly the tiles.*

Proof: In every patch P with $s(P) = (f'_P, g_P, b_P)$, it holds $b_P \geq 1$ per definition and $f'_P \geq 1$ because every vertex of P must be incident with an inner face. Since $g_P \geq 0$, the sequence $\langle f'_P, g_P, b_P\rangle = \langle 1, 0, 1\rangle$ is lexicographically smallest possible. For every tile T, it holds $s(T) = (1, 0, 1)$. Hence tiles are smallest possible. Let P be a patch which is not a tile. Then due to Lemma 6.19, this patch can be cut, and due to Lemma 6.18, every ancestor is smaller than P. Hence P is not smallest possible. ■

Chapter 7

The constructive definition of $\mathcal{M}(g, n, D, b_l, b_u)$

We will now give the constructive definition of $\mathcal{M}(g, n, D, b_l, b_u)$. This definition bases on cuts, i.e. for every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, we will define a unique **canonical sequence of cuts** that cuts M into a set S of tiles where (the inner face of) each tile corresponds to a face of M. Thus, the sequence of cuts is the inverse of a "jigsaw puzzle". An example for such a sequence is illustrated in Figure 6.21 on page 98 where the figure shows a tree–like structure instead of a sequence. The sequence is obtained by applying a depth–first search to the tree, i.e. whenever a patch is cut into more than one ancestor, then the first ancestor is cut completely before the second ancestor is started to be cut. Every cut in a canonical sequence of cuts is called a **canonical cut**.

Let a canonical sequence of cuts be given for a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$. By following the sequence of cuts in the inverse direction, one can construct M out of the tiles in S. This will be explained in detail in Chapter 8. However, this is not the only possibility to construct M. Another possibility will be discussed in Chapter 9. By defining the canonical sequence of cuts, we have even more than a pure constructive definition: We have a constructive definition which ensures that each map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is constructed *exactly* once.

7.1 The first cut

First, a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ will be cut along a Petrie path. Depending on the choice of the initial rooted edge, we will obtain different Petrie paths that are applied for the first cut. In order to define a *canonical* Petrie path for the first canonical cut, it suffices to encode M with respect to every possible initial rooted edge according to Section 6.2, i.e. for every rooted edge $r(e)$, we encode the rooted map $(M, r(e))$. Then qualities depending on the *edge codes* are assigned to the initial rooted edges (in *CGF*, the quality of a rooted edge increases with a lexicographically decreasing edge code where — as in *togen* — the absolute values of the qualities are again not important), and the rooted edge having the highest quality is the initial rooted edge of a unique Petrie path p which is taken as the **canonical Petrie path** in M. If there is more than one rooted edge having the highest quality, then all these rooted edges are isomorphic and the corresponding Petrie paths are isomorphic, too.

In a map of order n, there are $6n$ roots and hence $6n$ possible initial rooted edges that must be considered. However, some initial rooted edges initiate equivalent or even identical Petrie paths. For every Petrie path p, we can determine the set S of all initial rooted edges that initiate p due to Section 6.3, page 74. Hence, in order to classify p, the map M needs to be encoded only once with respect to a *canonical* rooted edge $r(e) \in S$. As the canonical initial rooted edge of p, we choose $r(e) = (e, h, s)$ where e is the start edge of p, h is the start semi–edge of p, and s is the first continuation of p. For Jordan curve paths, this is the only possible choice anyway (compare Section 6.3), and for sandwich and dumbbell paths, it always holds $r(e) \in S$, too.

Although the above criterion is sufficient in order to define a canonical Petrie path in M, in most cases simpler criteria help to distinguish Petrie paths and hence to determine a canonical Petrie path. In *CGF*, the following criteria are applied in the order they are listed (i.e. the $(i + 1)$–th criterion is applied only if the preceding i criteria did not yield different results):

(1) Petrie paths are distinguished by their *types*. By default, a dumbbell path has a higher quality than a sandwich path, and both types of paths have a higher quality than a Jordan curve path. However, sometimes this default should be changed. This will be discussed in Section 9.8.

(2) If two paths are of the same type, then they are distinguished by their *lengths*:
   - Two Jordan curve paths are distinguished by their total lengths. The quality of a Jordan curve path increases with decreasing length.
   - Two sandwich paths p and q are distinguished by the lengths of their parts. For the test, first $l_1(p)$ and $l_1(q)$ are compared with each other. In case of equality, $l_2(p)$ and $l_2(q)$ are compared, and if these values are equal, too, then $l_3(p)$ and $l_3(q)$ are compared with each other. In either case, a smaller value yields a higher quality.
   - Two dumbbell paths are distinguished by the lengths of their parts in the same way as two sandwich paths.

(3) In the next step, we compare the degrees of the faces that are incident with the Petrie paths. This is done as described in Section 4.2. As soon as a difference occurs in the sequences of face degrees, the lower degree yields the higher quality.

(4) If all these tests do not help to assign different qualities to two different Petrie paths p and q, then the map is encoded exactly two times described as above with respect to the canonical initial rooted edges of p and q.

If there is more than one Petrie path having the highest quality, then all these Petrie paths are isomorphic and hence canonical. Thus, it does not matter which one is taken for the actual cut. Note that the above criteria distinguish also non–isomorphic *equivalent* Petrie paths. Thus, the criteria provide a canonical Petrie path including the canonical start edge, end edge, first part etc. We will see in the next chapters that this information is indeed important.

A slight improvement of the algorithm that determines the canonical Petrie path in M is given by considering the equivalence classes of Petrie paths in M first: Out of every equivalence class, one representative is taken. Taking other representatives of the same class is avoided as usual by excluding the corresponding initial rooted edges from consideration due to Section 6.3, page 75. For each equivalence class S, the Petrie path with the highest quality in S is determined whenever it is necessary. The advantage of considering whole equivalence classes first is given by the possibility to *reject* whole equivalence classes in one go. For example, if we find a Jordan curve path p and we have already found a sandwich path q earlier, then we can state that all paths equivalent to p are not canonical (because their type has a lower quality than the type of q). It is a trivial task to determine all Petrie paths that are equivalent to p.

7.2 Further cuts

After cutting a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$ along a Petrie path, we obtain up to 3 *unmarked* patches. $A(M,p)$ denotes the set of patches that is obtained by cutting M along the Petrie path p. If p is the canonical Petrie path in M, then the elements of $A(M,p)$ are called the **canonical ancestors** of M. The next step is to cut these patches along zig–zag paths in order to obtain smaller patches. This will be repeated iteratively until all resulting patches are smallest possible, i.e. all resulting patches are tiles (due to Lemma 6.20). Since it is well–defined how to cut a patch along a valid zig–zag path, the only remaining task is to define for every patch (except the tiles) a *canonical 3–valent boundary vertex* which is used as the start vertex for the zig–zag path.

No matter how the canonical 3–valent boundary vertex is defined, there is one problem that must be considered in advance: As in part I of this text, we will construct cubic maps that are isomorphic to each other. However, we do not want to *accept* maps that are isomorphic to each other. Hence if we have a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$ which is not specular, then we want to accept either M or $\overline{M}$. For that, the canonical sequence of cuts must be defined in a way which is independent from the fact whether we are cutting M or $\overline{M}$ (because we do not know in advance whether we obtain M or $\overline{M}$ during the construction). However, by the way a patch is cut we might obtain two completely different sequences of canonical cuts for M and $\overline{M}$: If the canonical Petrie path in M is continued to the right, then it is continued to the left in $\overline{M}$ and vice versa (so that the start edges are mapped onto each other by an orientation reversing map automorphism). If M and $\overline{M}$ are cut along their canonical Petrie paths, then in general the obtained canonical ancestors are not specular, too, and hence each canonical ancestor of M can be assigned to a canonical ancestor of $\overline{M}$ only by orientation reversing map isomorphisms. Hence if a canonical ancestor of M is cut along a zig–zag path which is continued to the right, then the corresponding canonical ancestor of $\overline{M}$ must be cut along a zig–zag path which is continued to the *left* in order to have a corresponding cut. Otherwise we perform two cuts that do not correspond to each other.

However, for convenience we *always* want to cut a patch along a zig–zag path which is continued to the *right*. For that, we introduce the following rule: We define canonical cut sequences for the canonical ancestors of M only if the canonical Petrie path in M is continued to the right. As a consequence, if all canonical Petrie paths are continued to the left in a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$, then we switch to the mirror image $\overline{M}$ and follow the canonical cut sequence for $\overline{M}$.

By prescribing the orientation of the Petrie path, the orientations of the cut patches are prescribed, too. Hence a patch $P = (M, \mathcal{F}, W)$ must not be considered to be isomorphic to its **mirror image** $\overline{P} := (\overline{M}, \mathcal{F}', W)$ unless M is specular ($\mathcal{F}'$ contains the inverses of the faces in $\mathcal{F}$). Moreover, two patches must not be considered to be isomorphic if their boundary faces cannot be mapped onto each other. Thus we have the following definition: Two patches $P_1 = (M_1, \mathcal{F}_1, W_1)$ and $P_2 = (M_2, \mathcal{F}_2, W_2)$ are called **isomorphic** if and only if

- M_1 and M_2 are isomorphic, and
- at least one of the corresponding map isomorphisms is orientation preserving and maps the faces of $\mathcal{F}_1$ onto the faces of $\mathcal{F}_2$.

The corresponding map isomorphism is called a **patch isomorphism**. Two vertices (edges, semi–edges, paths) in a patch $P = (M, \mathcal{F}, W)$ are called **isomorphic in** P if and only if they are isomorphic in M and at least one of the corresponding map automorphisms is orientation preserving and maps the faces of $\mathcal{F}$ onto each other. If we do not want to name P, then we may also say that the two considered objects are **patch isomorphic**. The corresponding map automorphism is called a **patch automorphism**. For vertices, edges, and semi–edges, the condition that the boundary faces must be mapped onto each other implies that either both considered objects are boundary objects or both objects are inner objects.

In order to define a canonical boundary vertex in a patch $P = (M, \mathcal{F}, W)$, it is useful to assign a code called the *valency sequence* to each boundary vertex of P: Let P contain b_P boundary faces $F_1, \ldots, F_{b_P}$ where for every $i \in \{1, \ldots, b_P\}$, l_i is the degree of F_i and F_i is denoted by $F_i = \langle v_{i,0}, h_{i,0}, h'_{i,0}, \ldots, v_{i,l_i-1}, h_{i,l_i-1}, h'_{i,l_i-1}\rangle$. Beginning with an arbitrary boundary vertex $v_{i,j}$, $i \in \{1, \ldots, b_P\}$, $j \in \{0, \ldots, l_i - 1\}$, we trace the sequence of F_i and note down the valency of every vertex. Thus, we obtain a **valency sequence** $s_{i,j} = \langle \mathrm{val}(v_{i,j}), \ldots, \mathrm{val}(v_{i,j+l_i-1})\rangle$ (all indices modulo l_i) of length l_i where $\mathrm{val}(v_{i,k}) \in \{2, 3\}$ for every $k \in \{0, \ldots, l_i - 1\}$. Now the **quality** of the boundary vertex $v_{i,j}$ is defined as follows:

- The higher the number of convex edges incident with F_i, the higher the quality of $v_{i,j}$.
- For a fixed number of convex edges, the quality of $v_{i,j}$ increases with a lexicographically increasing valency sequence $s_{i,j}$.

We can also assign qualities to the boundary faces: The **quality** of a boundary face F_i is defined as the highest quality among all incident boundary vertices. Let $v_{i,j}$ be a boundary vertex in F_i having the highest quality, i.e. the sequence $s_{i,j}$ is lexicographically maximal among all $j \in \{0, \ldots, l_i - 1\}$. Then $s_{i,j}$ is called the **canonical valency sequence** of F_i. Thus, $v_{i,j}$ corresponds to the first entry of the canonical valency sequence of F_i. Of course, if P is not a tile, then *every* boundary face of P is incident with a 3–valent boundary vertex because otherwise, P would be disconnected. Hence every canonical valency sequence in P begins with a "3".

Figure 7.1 shows some example patches containing one boundary face each. The canonical valency sequences are $\langle 3, 2, 2, 2, 3, 2, 2, 2\rangle$ for the left patch, $\langle 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2\rangle$ for the mid patch, and $\langle 2, 2, 2, 2, 2\rangle$ for the right patch. In particular, the canonical valency sequence is defined also for the boundary faces of tiles.

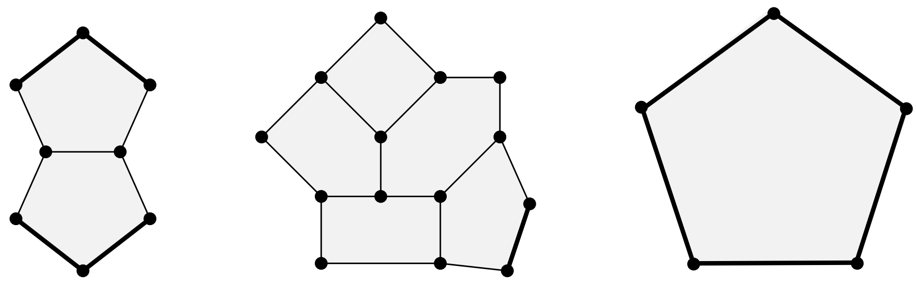

Figure 7.1: Example patches. The convex edges are emphasized.

Now we prescribe that the canonical boundary vertex v must have the highest quality among all boundary vertices in P. Let S be the set of all boundary vertices that satisfy this condition. If $|S| > 1$, i.e. if v is not unique, then we distinguish the non–isomorphic elements of S by assigning so–called **extra qualities** to them in the following way: To every vertex $v_{i,j} \in S$, we assign the *edge code* of the rooted map $(M, (\{h_{i,j}, h'_{i,j}\}, h_{i,j}, \texttt{right}))$. Since the orientation of P is fixed, the side can be chosen arbitrarily (but of course homogeneously). We define that the extra quality of $v_{i,j}$ increases with a lexicographically increasing corresponding edge code.

The vertex with the highest extra quality among the boundary vertices in S is the **canonical boundary vertex** in P. If there is more than one such vertex, then all of them are patch isomorphic and canonical.

This completes the constructive definition of $\mathcal{M}(g, n, D, b_l, b_u)$.

The valid zig–zag path p in P whose start vertex is the canonical 3–valent boundary vertex is called the **canonical zig–zag path** in P. If p is the canonical zig–zag path in P, then the patches in $A(P, p)$ are called the **canonical ancestors** of P. Tiles do not have canonical ancestors. The patches that occur at any point in the canonical sequence of cuts of a patch P or a cubic map M are called **iterated canonical ancestors** of P or M, respectively. A patch Q or a cubic map M is called a **canonical descendant** of P if P is a canonical ancestor of Q or M, respectively. A patch Q or a cubic map M is called an **iterated canonical descendant** of P if and only if P is an iterated canonical ancestor of Q or M, respectively.

For future use, we introduce the following notation and definition: By $\langle x_0, \ldots, x_l\rangle^m$, we denote a sequence which is a concatenation of $m \in \mathbb{N}_0$ copies of the sequence $\langle x_0, \ldots, x_l\rangle$. Thus, the length of the sequence $\langle x_0, \ldots, x_l\rangle^m$ is $m \cdot (l + 1)$. A **period** of a sequence s is a sequence $s' = \langle x_0, \ldots, x_l\rangle$ such that $s = \langle x_0, \ldots, x_l\rangle^m$ for some $m \geq 1$ and $s' \neq \langle y_0, \ldots, y_k\rangle^r$ for every sequence $\langle y_0, \ldots, y_k\rangle$ and every $r > 1$. If $m > 1$, then s is called **periodical**. We use the symbol "$\circ$" as the concatenation operator. For example, we may write $\langle a, b, c, d\rangle = \langle a\rangle \circ \langle b, c, d\rangle$.

7.3 About automorphisms in patches

From the definition of isomorphic maps on page 4, it follows that a map automorphism (f, g) in a map or a patch is uniquely given by the two following informations:

- the pair $(h, g(h))$ for *one* arbitrary semi–edge h of the map or the patch
- the information whether (f, g) is orientation preserving or orientation reversing

In a patch, we only consider orientation preserving map automorphisms. Hence in this case, the first information is sufficient. In both cases (patches or maps), the complete map automorphism (f, g) can be reconstructed out of the given information. Thus in a patch, a patch automorphism that maps a given semi–edge h_1 onto a given semi–edge h_2 is unique. Analogously, in a map there is at most one orientation preserving and at most one orientation reversing map automorphism that maps a given semi–edge h_1 onto a given semi–edge h_2. In a cubic map, there are up to 6 map automorphisms that map a given vertex v_1 onto a given vertex v_2: Let h be an arbitrary semi–edge incident with v_1. This semi–edge must be mapped onto a semi–edge which is incident with v_2. There exist 3 such semi–edges, and the map automorphism may be orientation preserving or orientation reversing. Hence there are 6 possibilities. Similarly, in a patch there are up to 3 patch automorphisms that map a given inner vertex v_1 onto a given inner vertex v_2. However, a patch automorphism that maps a given *boundary* vertex v_1 onto a given boundary vertex v_2 is always unique: If v_1 and v_2 are 3–valent, then both vertices each have one incident inner semi–edge. These semi–edges must be mapped onto each other, hence the mapping is unique. If v_1 and v_2 are 2–valent, then the incident semi–edges having the inner face on their right side must be mapped onto each other (otherwise the map automorphism would be orientation reversing), hence the mapping is unique. As a consequence, a non–trivial patch automorphism (f, g) does not fix boundary vertices of a patch (because the trivial patch automorphism (f, g) whose elements f and g are identity functions is the unique patch automorphism that maps a boundary vertex onto itself).

The above statements hold not only for automorphisms, but also for isomorphisms between two maps or two patches. For example, a patch isomorphism that maps a given boundary vertex v_1 of a patch P_1 onto a given boundary vertex v_2 of a patch P_2 is always unique.

7.4 Marked patches

The determination of the canonical boundary vertex in an unmarked patch P makes it sometimes necessary to compute the edge code of P with respect to all candidates for the canonical boundary vertex (to be precise: with respect to all corresponding incident boundary semi–edges having the inner face on their right side). This is the case whenever $|S| > 1$ (with S defined in Section 7.2), i.e. the pair (i, j) described in Section 7.2 is not unique. Especially for boundary faces not containing convex edges, S can be rather large. The computation of an edge code is much more time consuming than the other steps that are performed in order to assign qualities to boundary faces and boundary vertices. Hence the goal is to decrease the number of occasions where an edge code of P must be computed. For that, we *mark* patches.

Let $P = (M, \mathcal{F}, W)$ be a marked patch. Two marked patches $P_1 = (M_1, \mathcal{F}_1, W_1)$ and $P_2 = (M_2, \mathcal{F}_2, W_2)$ are called **mark–isomorphic** if and only if P_1 and P_2 are isomorphic and one corresponding patch isomorphism maps the vertices of W_1 onto the vertices of W_2. Two *marked vertices* $v_1, v_2 \in W$ in a patch $P = (M, \mathcal{F}, W)$ are called **mark–isomorphic** if and only if they are patch isomorphic and the corresponding patch automorphism maps the elements of W onto each other. We do not distinguish between mark–isomorphic patches.

For the constructive definition of $\mathcal{M}(g, n, D, b_l, b_u)$ using marked patches, we must provide rules how to mark the patches that occur in the canonical cut sequences. The **basic marking rule** is that a marked vertex v in a patch P must correspond to the first entry of the canonical valency sequence of the boundary face F incident with v, i.e. v must have the highest quality among all boundary vertices incident with F. In particular, a marked vertex is always 3–valent except if P is a tile. A boundary vertex is called **markable** if and only if it satisfies the basic marking rule.

For the boundary faces in the patches that are obtained by cutting a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ along a canonical Petrie path p, there are the following marking rules (here we assume as usual that p is continued to the right):

- If p is a Jordan curve path, then w_0 and u_1 are marked (see Section 6.4.1) where h_0 is the start semi–edge of p. Since w_0 and u_1 are 3–valent and the canonical valency sequence of a boundary face without convex edges is $\langle 3, 2\rangle^m$ for some $m \in \mathbb{N}$ (so that we obtain the canonical valency sequence by starting from an arbitrary 3–valent boundary vertex), the basic marking rule is satisfied.
- If p is a sandwich path whose second part has even length so that we obtain one boundary face F, then F contains exactly one markable boundary vertex. This will be shown in Section 8.6.
- If p is a sandwich path whose second part has odd length (so that we obtain three boundary faces), then starting with u_0, u_t, and w_t, respectively (see Section 6.4.3), the three boundary faces are traced, and in each face, the first vertex that satisfies the basic marking rule is marked. h_0 is the start semi–edge of p. We will see in Section 8.6 that in the side

boundaries, the markable vertex is unique, and the main boundary contains at most two markable vertices.

- If p is a dumbbell path, then as in the previous case, the boundary faces are traced, and in each case the first markable vertex is marked. Look at Figure 6.8 on page 80. The start vertices for the traces are:
  (1) u_t, w_t, and w_s
  (2) u_t, u_0, and w_s
  (3) u_t, w_t, and u_k
  (4) u_0, u_t, and w_s
  Since p is always continued to the right, the start vertices for the traces can be chosen arbitrarily: When the first continuation of a dumbbell path p is fixed, then $l_1(p)$, $l_2(p)$, and $l_3(p)$ imply the case 1-4 in Figure 6.8. Hence no inconsistencies can occur that could occur if paths that are continued to the left would be considered, too. We will see in Section 8.6 that in the side boundaries, the markable vertex is unique, and the main boundary contains at most two markable vertices.

For the boundary faces that are obtained by cutting a marked patch P along the canonical zig–zag path p, there are the following marking rules in addition to Section 6.6:

- After an intersection cut, starting with u_{k+1} and w_{t-1} (see Section 6.6.1), the new boundary faces F_3 and F_4 are traced, and in each face the first markable vertex is marked.
- After a linkage cut, starting with u_{k+1} (see Section 6.6.2), the new boundary face F_3 is traced, and the first markable vertex is marked.
- After an enclosure cut, the new boundary face F_3 (see Section 6.6.3) is traced starting with u_{k+1}, and the first markable vertex is marked. In the other new boundary face F_4, the markable vertex is unique which will be shown in Section 8.2.

In a marked patch P, the *canonical boundary vertex* and hence the canonical zig–zag path p for the next cut (if P is not a tile) is chosen as follows:

- Assign qualities to the boundary vertices as before.
- The marked vertex having the highest quality is the canonical boundary vertex.
- If there is more than one marked vertex having the highest quality, then take the vertex with the highest *extra* quality among those vertices as the canonical boundary vertex.
- If there is more than one marked vertex having the highest extra quality among all marked vertices having the highest quality, then let S be the set of all those vertices. Let $u, v \in S$. Then u and v are patch isomorphic but not necessarily mark–isomorphic because in the edge code, marks are not considered (Figure 7.2 shows an example: a and b are patch isomorphic but not mark–isomorphic). But in a *marked* patch, u and v must be *mark*–isomorphic in order not to be distinguishable. Hence to every vertex $v \in S$, a *mark–quality* is assigned in the following way: The labels that are assigned to all marked vertices while computing the extra quality of v are noted down in a lexicographically minimal order. The resulting sequence of labels is assigned to v. The **mark–quality** of v increases with a lexicographically decreasing sequence of labels. Thus, a mark–quality is assigned to every $v \in S$. The vertex with the highest mark–quality is the **canonical boundary vertex** in P. If there is more than one vertex having the highest mark–quality, then the corresponding marked vertices are mark–isomorphic and all of them are canonical.

Throughout the rest of this text, we assume that we *always* compute the extra quality of all marked boundary vertices having the highest quality (even if the marked boundary vertex

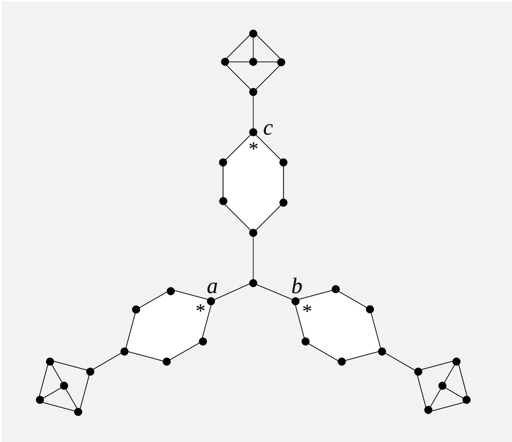

Figure 7.2: a, b, and c are marked

having the highest quality is unique) and we *always* compute the mark–quality of all boundary vertices having the highest extra quality so that we may simply say that the canonical boundary vertex is the boundary vertex having the highest mark–quality (among all boundary vertices to which a mark–quality is assigned). However, in general, not to every marked vertex an extra quality and a mark–quality is assigned. So in order to be able to compare marked vertices without checking explicitly whether an extra quality or a mark–quality has been assigned to them, we define the *total quality* of a marked vertex: The **total quality** of a marked boundary vertex v depends on the sequence $\langle q_1, q_2, q_3, q_4, q_5\rangle$:

- If a mark–quality is assigned to v, then $q_1 = 1$, otherwise $q_1 = 0$.
- If $q_1 = 1$, then q_2 is the mark–quality of v, otherwise $q_2 = 0$.
- If an extra quality is assigned to v, then $q_3 = 1$, otherwise $q_3 = 0$.
- If $q_3 = 1$, then q_4 is the extra quality of v, otherwise $q_4 = 0$.
- q_5 is the quality of v.

The total quality of v increases with a lexicographically increasing sequence $\langle q_1, q_2, q_3, q_4, q_5\rangle$ (so that as before, marked boundary vertices to which a mark–quality is assigned have a higher quality than the other marked boundary vertices). We also define the total quality of boundary faces: The **total quality** of a boundary face F in a marked patch is defined as the total quality of the marked vertex v that is incident with F.

With the additional marking rules and the additional rules for the canonicity of a boundary vertex, it is faster to determine the canonical boundary vertex of a (marked) patch. The only difference to the definition in Section 7.2 is that now, every boundary face F contributes at most one boundary vertex to the set S defined on page 104, namely the marked vertex incident with F.

From now on, we consider every patch to be marked with respect to the above marking rules. In the figures, every marked vertex will be marked with a star "*" whenever it is important to know which vertices are marked. The marked patches that occur in the canonical cut sequence of some map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, i.e. the iterated canonical ancestors of some map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, will be called **canonical patches** from now on. Of course, it depends on g, n, D, b_l, and b_u whether a patch is canonical. Thus, we presuppose that these values are fixed.

Chapter 8

The bottom–up approach

In this chapter, the so–called *bottom–up approach* for the construction of $\mathcal{M}(g, n, D, b_l, b_u)$ will be described. The principle of the bottom–up approach is given by starting with the tiles and following the canonical sequence of cuts in the inverse direction in order to construct a map M out of its corresponding tiles.

8.1 The construction of the patches

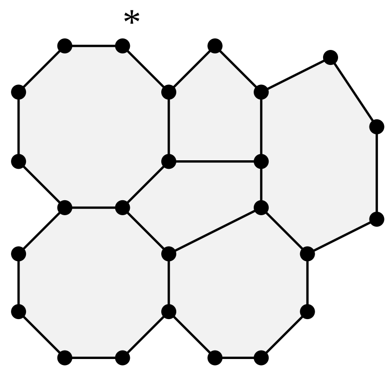

Figure 8.1

The first step in the bottom–up approach is to construct all canonical patches. When the patches are constructed in a bottom–up manner, it is clear that it is not possible to construct only the canonical patches straightforward. Figure 8.1 shows an example patch. By looking at the patch, one e.g. cannot say if this patch occurs in the canonical cut sequence of a map $M \in \mathcal{M}(0, 40, \{5, 7, 8\}, b_l, b_u)$ with $l_u(5) = l_u(7) = l_u(8) = 0$ and $b_u(5) = b_u(7) = b_u(8) = \infty$ because this depends not only on the patch itself but also on the other patches that are cut while tracing the canonical cut sequences. Hence it is not possible to follow the inverses of canonical cut sequences straightforward. Instead, we perform every inverse of a cut that does not obviously contradict a canonicity rule so that the resulting patch is *possibly* canonical. We call the inverse of a cut an **agglutination**. An agglutination is called **canonical** if the following 8 conditions are satisfied for the resulting g–patch P:

(1) Every inner face in P has a degree $d \in D$.
(2) For every $d \in D$, it holds that the number of inner faces of degree d in P is at most $b_u(d)$.
(3) It holds $f'_P \leq f$ where $f = \frac{1}{2}n + 2 - 2g$ is the number of faces in every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ (due to formula 1.2 on page 9).
(4) $g_P \leq g$
(5) $b_P \leq g - g_P + 1$
(6) The inverse of a cut takes place along a zig–zag path p that is continued to the right.
(7) P is pseudo–convex, i.e. P is a patch.
(8) p is the canonical zig–zag path in P.

A patch P that satisfies the conditions 1-5 (and 7) is called **possibly canonical**. Of course, if P has been obtained by an agglutination, then for each of the new boundary faces, one boundary vertex incident with that boundary face must be marked. This will be discussed below when condition 8 is discussed. Note that if P has been obtained by a canonical agglutination, then P is not necessarily canonical. For example, there are some more sophisticated conditions that detect non–canonical patches and that will be discussed in the Sections 8.5 and 8.7.

The conditions 1-5 and 7 refer to properties that every canonical *patch* always has (in particular, every canonical patch is possibly canonical). The conditions 6 and 8 refer to properties that a canonical *cut* always has. The first condition and the second condition is clear because after cutting a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, the inner faces in the ancestors of M correspond to the faces in M. The third and the fourth condition is clear due to Section 6.4. The last three conditions are trivial. The 5th condition follows from the following theorem:

Theorem 8.1 *In a canonical patch P of size $s(P) = (f'_P, g_P, b_P)$, it holds $b_P + g_P \leq g + 1$.*

Proof: We will prove the lemma by induction. Let $M \in \mathcal{M}(g, n, D, b_l, b_u)$. The patches that are obtained by cutting M along an arbitrary Petrie path p satisfy the above inequation. This can easily be verified using Section 6.4. We will do this here for an example: Let p be a dumbbell path such that $A(M, p) = \{P_1, P_2\}$ where w.l.o.g. $b_{P_1} = 1$ and $b_{P_2} = 2$. Due to Section 6.4.2, it holds $g_{P_1} + g_{P_2} + 1 = g$. Hence $b_{P_1} + g_{P_1} = 1 + g_{P_1} = 1 + (g - g_{P_2} - 1) = g - g_{P_2} \leq g < g + 1$ and $b_{P_2} + g_{P_2} = 2 + g_{P_2} = 2 + (g - g_{P_1} - 1) = 1 + g - g_{P_1} \leq g + 1$.

Let now P be a canonical patch that satisfies $b_P + g_P \leq g + 1$ and that is cut along an arbitrary zig–zag path p. It holds $|A(P, p)| \leq 2$.

Let $A(P, p) = \{Q\}$. If p is a linkage path, then $g_Q = g_P$ and $b_Q = b_P - 1$. Hence $g_Q + b_Q = g_P + (b_P - 1) < g_P + b_P \leq g + 1$. If p is an intersection path or an enclosure path, then $g_Q = g_P - 1$ and $b_Q = b_P + 1$. Hence $g_Q + b_Q = (g_P - 1) + (b_P + 1) = g_P + b_P \leq g + 1$.

Let $A(P, p) = \{Q_1, Q_2\}$. It holds $g_{Q_1} + g_{Q_2} = g_P$ and $b_{Q_1} + b_{Q_2} - 1 = b_P$. Since $b_{Q_1} \geq 1$ and $b_{Q_2} \geq 1$, it holds $b_{Q_1} \leq b_P$ and $b_{Q_2} \leq b_P$. Hence $b_{Q_1} + g_{Q_1} \leq b_P + g_{Q_1} = b_P + g_P - g_{Q_2} \leq b_P + g_P \leq g + 1$ and $b_{Q_2} + g_{Q_2} \leq b_P + g_{Q_2} = b_P + g_P - g_{Q_1} \leq b_P + g_P \leq g + 1$. ∎

Let P be a patch that has been obtained by an agglutination. If the agglutination is canonical, then P will be accepted, otherwise P will be rejected because either P is not possibly canonical (if one of the conditions 1-5 or 7 is not satsified) or P has not been agglutinated in a canonical way, i.e. the corresponding cut is not a canonical cut (if condition 6 or 8 is not satisfied). Thus if we assume that every possibly canonical patch P is constructed once in every possible way (with respect to the valid zig–zag paths in P), then every possibly canonical patch will be accepted exactly once. Details will be given in Section 8.3.

In order to verify if the above 8 conditions are satisfied for a patch P that has been agglutinated, the first five conditions are verified in the following way:

- The first condition is satisfied automatically if every tile that is agglutinated has a degree $d \in D$.
- The second and third condition can be verified in advance (before actually performing an agglutination): The number b_P of inner faces of a patch P is the sum of inner faces of its canonical ancestors. The same holds for inner faces of a given degree. In particular, if a patch Q satisfying condition 2 and 3 is agglutinated with itself, then the second and third condition need not be verified at all for the resulting patch P.
- In order to verify the fourth condition, g_P can easily be calculated according to Section 6.6 before performing an agglutination. In particular, if a patch Q satisfying condition 4 is agglutinated with itself along a linkage path, then it holds $g_P = g_Q$ so that the fourth condition dores not need to be verified at all for P.
- The fifth condition does need not be verified at all if one patch Q is agglutinated with itself along an intersection or enclosure path: In these cases, it holds $g_P = g_Q + 1$ and $b_P = b_Q - 1$ for the resulting descendant P of Q. Hence if Q satisfies the fifth condition, then P satisfies it, too. In all remaining cases, g_P and b_P can easily be calculated according to Section 6.6 before performing an agglutination.

In order to verify the last three conditions, some more effort must be done:

Condition 6: In order to secure that the agglutination takes place along a zig–zag path that is continued to the right, the valencies of the vertices that are identified with each other must be known. The following details can easily be derived from Section 6.6, but nevertheless they will be listed explicitly here. We call the inverse of an intersection cut, a linkage cut, or an enclosure cut an **intersection agglutination**, a **linkage agglutination**, or an **enclosure agglutination**, respectively. If it is clear whether we talk about a cut or an agglutination, we may omit the corresponding word and just write "intersection", "linkage", or "enclosure" depending on the type of cut or agglutination.

- In order to perform an *intersection agglutination*, we must have two boundary faces $F_3 = \langle u_0, a_0, a'_0, \ldots, u_{k-1}, a_{k-1}, a'_{k-1}\rangle$ and $F_4 = \langle b'_{j-1}, b_{j-1}, w_{j-1}, \ldots, b'_0, b_0, w_0\rangle$ (where as usual, the u_i and w_i are vertices and the a_i, a'_i, b_i, and b'_i are semi–edges). Suppose we want to perform an intersection agglutination of length l, i.e. the length of the corresponding zig–zag path is l. Then for some $s \in \{0, \ldots, k-1\}$ and $t \in \{0, \ldots, j-1\}$, the vertices $u_s, \ldots, u_{s+l}$ (indices modulo k) are identified with the vertices $w_t, \ldots, w_{t+l}$ (indices modulo j). For convenience, let w.l.o.g. $s = t = 0$ so that $u_0, \ldots, u_l$ are identified with $w_0, \ldots, w_l$. Simultaneously, the semi–edges $a_0, \ldots, a_{l-1}$ are identified with $b_0, \ldots, b_{l-1}$, and $a'_0, \ldots, a'_{l-1}$ are identified with $b'_0, \ldots, b'_{l-1}$. See Figure 8.2 for an example with $l = 6$, $k = 10$, and $j = 14$. Due to Section 6.6, the vertices u_0, w_0, u_l, and w_l must each be 2–valent. Moreover, for every $i \in \{1, \ldots, l-1\}$, it must hold $\mathrm{val}(u_i) = 2$ if i is odd and $\mathrm{val}(u_i) = 3$ if i is even, and it must hold $\mathrm{val}(w_i) = 3$ if i is odd and $\mathrm{val}(w_i) = 2$ if i is even.
- In order to perform a *linkage agglutination* of length l, we must have one boundary face $F_3 = \langle u_0, a_0, a'_0, \ldots, u_{k-1}, a_{k-1}, a'_{k-1}\rangle$. For some $s, t \in \{0, \ldots, k-1\}$, the vertices $u_s, \ldots, u_{s+l}$ are identified with the vertices $u_{t+l}, \ldots, u_t$ (all indices modulo k). Of course, the sets $\{u_s, \ldots, u_{s+l}\}$ and $\{u_t, \ldots, u_{t+l}\}$ must be disjoint. For convenience, let $s = 0$ and let $w_i := u_{t+l-i}$ for every $i \in \{0, \ldots, l\}$ and $b_i := a'_{t+l-i}$ as well as $b'_i := a_{t+l-i}$ for every $i \in \{0, \ldots, l-1\}$. Then the situation and hence the conditions are exactly the same as in the intersection case. See Figure 8.3 for an example with $l = 2$, $k = 26$, and $t = 12$.

- In order to perform an *enclosure agglutination* of length l, we must have two boundary faces $F_3 = \langle u_0, a_0, a_0', \ldots, u_{k-1}, a_{k-1}, a_{k-1}' \rangle$ and $F_4 = \langle w_0, b_0, b_0', \ldots, w_{j-1}', b_{j-1}, b_{j-1}' \rangle$. It must hold $l > j$. For some $s \in \{0, \ldots, k-1\}$ and $t \in \{0, \ldots, j-1\}$, the vertex w_t is identified with the vertices u_s and u_{s+j} ($s+j$ modulo k). For convenience, let $s = t = 0$. See Figure 8.4 for an example with $l = 10$, $j = 8$, and $k = 24$. For $i \in \{1, \ldots, j-1\}$, the vertices u_i and w_{j-i} are identified with each other. Moreover, for $i \in \{1, \ldots, l-j\}$, the vertices u_{-i} and u_{j+i} (indices modulo k) are identified with each other. Of course, the sets $\{u_0, \ldots, u_{-(l-j)}\}$ and $\{u_j, \ldots, u_l\}$ must be disjoint. The vertices u_0, u_j, w_0, $u_{-(l-j)}$, and u_l must be 2–valent.
  - If the enclosure entry length $l - j$ is even, then for every $i \in \{1, \ldots, l-1\}$, it must hold $\mathrm{val}(u_i) = 3$ if $l - i$ is odd and $\mathrm{val}(u_i) = 2$ if $l - i$ is even.
  - If $l - j$ is odd, then for every $i \in \{-(l-j)+1, \ldots, j-1\}$ (modulo k), it must hold $\mathrm{val}(u_i) = 3$ if i is odd and $\mathrm{val}(u_i) = 2$ if i is even.

  In either case, for the vertex v that is identified with u_i, it must hold $\mathrm{val}(v) + \mathrm{val}(u_i) = 5$ except for $i = 0$.

Condition 7: Let v_0 be the vertex in the resulting patch P that is obtained by identifying u_0 (in an intersection or a linkage) or $u_{k-(l-j)}$ (in an enclosure), respectively, with the appropriate vertex w (i.e. $w = w_0$ in an intersection, $w = u_{t+l}$ in a linkage, and $w = u_l$ in an enclosure). v_0 is the start vertex of the zig–zag path p containing the identified edges in P. Let $F_1 = \langle v_0, h_0, h_0', \ldots, v_{m-1}, h_{m-1}, h_{m-1}' \rangle$ be the boundary face that is incident with v_0 in P (see Figure 8.5 for the intersection case). Then, in order to secure that P is pseudo–convex, both v_1 and v_{m-1} must be 2–valent because v_0 is 3–valent. In the intersection and the linkage case, the same condition holds for the end vertex of p. In detail:

- In an intersection agglutination, let v_0 and v_l be the boundary vertices that are obtained by identifying u_0 and w_0 as well as u_l and w_l, respectively. v_0 and v_l are 3–valent. Hence the incident boundary semi–edges must have 2–valent targets. Hence it must hold $l < k - 1$ and $l < j - 1$, and the vertices u_{k-1}, w_{j-1}, u_{l+1}, and w_{l+1} must each be 2–valent.
- For a linkage agglutination, the argumentation is analogous. Hence it must hold $l < t - 1$ and $t + l < k - 1$, and the vertices u_{k-1}, u_{t+l+1}, u_{l+1}, and u_{t-1} must each be 2–valent.
- In an enclosure agglutination, applying the same argumentation, it must hold $l < k - (l - j) - 1$, and the vertices $u_{k-(l-j)-1}$ and u_{l+1} must each be 2–valent.

Condition 8: Let v_0 and F_1 be defined as in the discussion of condition 7. In order to secure that p is the *canonical* zig–zag path in P, the vertex v_0 must be a canonical boundary vertex. Thus v_0 and F_1 must satisfy the following conditions:

- v_0 must be the vertex that is marked in F_1. So first of all, v_0 must be markable at all. In order to be markable, the sequence $s = \langle \mathrm{val}(v_0), \ldots, \mathrm{val}(v_{m-1}) \rangle$ must be lexicographically maximal among all valency sequences of F_1. The (canonical) valency sequence of F_1 can easily be derived from the sequences of the old boundary faces. In Section 8.2, a so–called *boundary sequence* will be introduced that makes this task and some more considerations much easier. A necessary but not sufficient condition for s to be lexicographically maximal (that will also be essential for the considerations in Section 8.2) is that if F_1 contains a convex edge, then it must hold $\mathrm{val}(v_{m-2}) = 2$ (see Figure 8.5) because otherwise the sequence $s' = \langle \mathrm{val}(v_{m-2}), \mathrm{val}(v_{m-1}), \mathrm{val}(v_0), \ldots, \mathrm{val}(v_{m-3}) \rangle$ would be lexicographically larger than s (this follows from the fact that due to the pseudo–convexity of P, every entry “3” in every valency sequence is preceded and followed by an entry “2”; so if s starts with exactly r repetitions of the sequence $\langle 3, 2 \rangle$ followed by one more entry “2”, then s' starts with exactly $r+1$ repetitions of the sequence $\langle 3, 2 \rangle$ followed by one additional entry “2”). So if F_1 contains

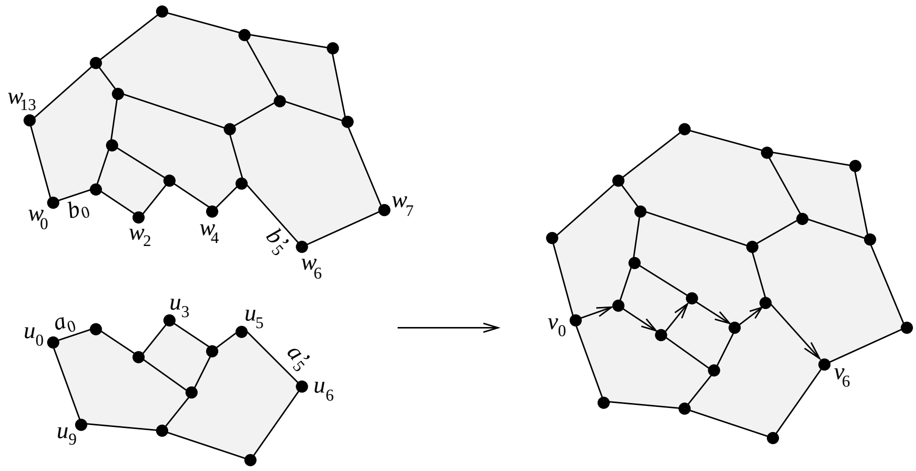

Figure 8.2

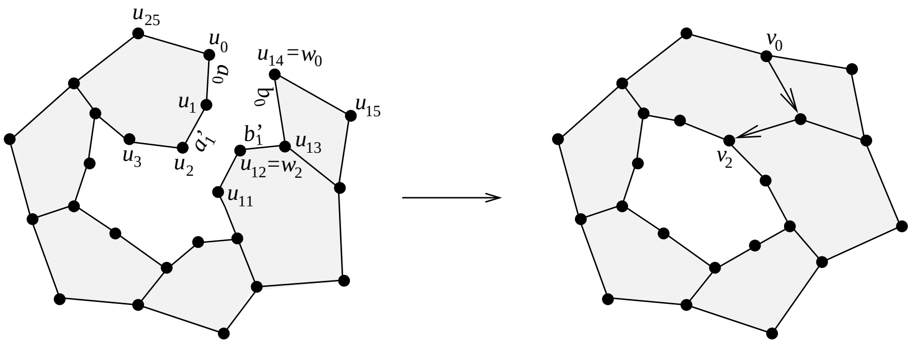

Figure 8.3

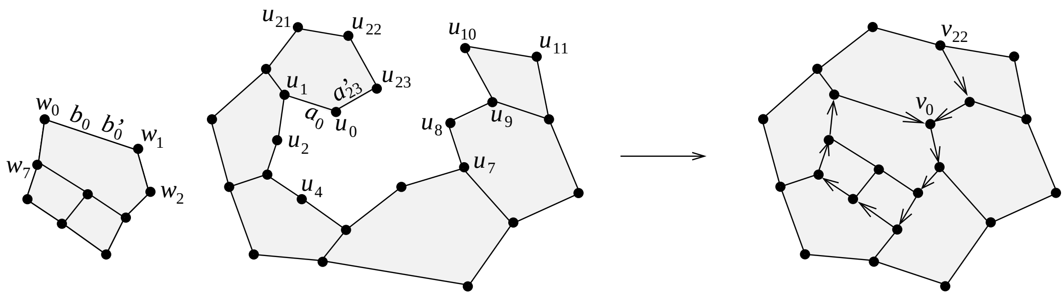

Figure 8.4

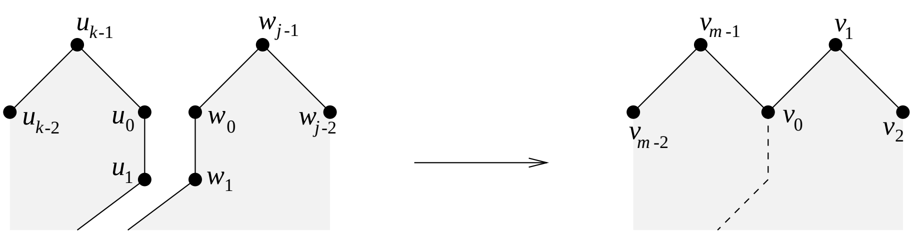

Figure 8.5

a convex edge, then due to the condition $\mathrm{val}(v_{m-2}) = \mathrm{val}(v_{m-1}) = 2$, necessarily but not sufficiently, a convex edge must precede v_{m-1} while tracing F_1.

- If v_0 is markable, then we mark v_0 in F_1. Now in order to be canonical, v_0 must have the highest total quality among all marked boundary vertices in P. In order to check if this is true, the quality of v_0 and eventually the extra quality and the mark–quality of v_0 must be computed and compared with the other marked vertices in P (see Section 7.4).

  Important: If p is a linkage path, then also the new boundary face F_2 being incident with the end vertex of p must be marked. Assume that v_0 is the canonical boundary vertex in P (so that p is the canonical zig–zag path in P). Then the canonical cut is defined independently of the marked vertex in F_2. Contrary, when a linkage agglutination is performed, then every markable vertex contained in F_2 may be marked. In order to secure that we construct all canonical patches, we even *must* consider all markable vertices. Hence we do not construct a single patch P but a set of patches in the following way: Let $v_1, \ldots, v_m$ be the markable boundary vertices that are incident with F_2. Then we construct a set of patches $P_1, \ldots, P_m$ where $P_i = (M, \mathcal{F}, W_i)$, $i \in \{1, \ldots, m\}$, is the patch which is obtained by the linkage agglutination along p where v_0 and v_i are marked (the marked vertices in the unmodified boundary faces are prescribed and identical for every $i \in \{1, \ldots, m\}$). It is possible that not all of the obtained patches may be accepted because some of the marks might not correspond to inverses of a canonical cut: If v_0 is the unique marked vertex having the highest quality among all marked vertices in P_1, then all agglutinations are canonical (because all markable vertices contained in F_2 have the same quality so that v_0 is the unique marked vertex having the highest quality in $P_2, \ldots, P_m$, too). But if there are other marked vertices having the same quality as v_0 (in particular, this might be the case for the marked vertex contained in F_2), then it might happen that depending on the marked vertex in F_2, another marked vertex has a higher total quality than v_0. We reject the non–canonical possibilities. For the remaining canonical possibilities, it might happen that some of the corresponding patches are mark–isomorphic. In this case, we accept one representative out of each class of mark–isomorphic patches.

Altogether, the 8 conditions for an agglutination to be canonical can easily be verified: The first 5 conditions can be verified without even looking at the boundary faces, and the first 7 conditions can be verified without actually constructing the patch P. Only for the verification of condition 8, it is sometimes necessary to actually construct P (namely if an edge code must be computed).

8.2 The boundary code

The valency sequences of the boundary faces are rather dissipated. They can be abbreviated. We will now introduce the so–called *boundary code* for a boundary face $F = \langle v_0, h_0, h'_0, \ldots, v_{m-1}, h_{m-1}, h'_{m-1} \rangle$ of a patch P.

Let v_0 be the marked boundary vertex contained in F. Beginning with v_{m-1}, we trace the sequence of F and note down the number of 3–valent vertices between the current position and the next convex edge. Then, the next current position is the vertex that follows this convex edge in the sequence of F. Let c be the number of convex edges in F.

- If $c = 0$, then we note down the total number of 3–valent vertices by b_0 and obtain a sequence $\langle b_0 \rangle$ of length $k = 1$.
- If $c > 0$, then we obtain a sequence $b = \langle b_0, \ldots, b_{k-1} \rangle$. In principle, it holds $k \in \{c, c+1\}$:

The "current position" is always preceded by a convex edge except maybe for the vertex v_{m-1} where the trace started. So we have either c or $c+1$ "current positions". However, since v_0 is markable, it holds that v_{m-1} is always preceded by a convex edge, too (see Section 8.1), so that it always holds $k = c$.

In either case, the sum $\sum_{i=0}^{k-1} b_i$ of the entries in the sequence is equal to the number of 3–valent vertices incident with F. The pair (c, b) is called the **canonical boundary code** and b is called the **canonical boundary sequence** of F.

Figure 7.1 on page 104 shows some example patches. The canonical boundary codes of the boundary faces are $(4, \langle 1, 0, 1, 0\rangle)$ for the left patch, $(1, \langle 5\rangle)$ for the patch in the middle, and $(5, \langle 0, 0, 0, 0, 0\rangle)$ for the right patch. As an example for a boundary face without convex edges, the canonical boundary sequence of the boundary face of the patch on the left side of Figure 6.20 (page 97) is $(0, \langle 6\rangle)$.

It is easy to reconstruct a canonical valency sequence out of the corresponding canonical boundary code $(c, \langle b_0, \ldots, b_{k-1}\rangle)$ of a boundary face F: If $c = 0$, then substitute b_0 by the sequence $\langle 3, 2\rangle^{b_0}$. If $c > 0$, then substitute every entry b_i ($i \in \{0, \ldots, k-1\}$) by the sequence $\langle 3, 2\rangle^{b_i} \circ \langle 2\rangle$. In particular, this operation yields a bijection between the set of canonical valency sequences and the set of canonical boundary codes. This is obvious because on the one hand, since F does not contain concave edges, every entry "3" in the valency sequence is preceded and followed by an entry "2". On the other hand, as long as there does not occur a convex edge while tracing the boundary face, every entry "2" is preceded and followed by an entry "3". Hence the sequence $\langle 3, 2\rangle$ must occur b_i times. Due to the definition of the boundary sequence, it holds that if $c > 0$, then a convex edge must follow, and hence the next boundary vertex is also 2–valent. Hence the substitute sequence ends with an additional entry "2".

There are some easy consequences of the correspondence between the canonical boundary code (c, b) and the canonical valency sequence of a face F:

- If the sequence $b = \langle b_0, \ldots, b_{k-1}\rangle$ is cyclically shifted, then the resulting sequence is never lexicographically larger than b. A cyclical shift b' of the *canonical* boundary sequence b of F is just called a **boundary sequence** of F, and the corresponding pair (c, b') is just called a **boundary code** of F. Later in this section, we will obtain boundary sequences and boundary codes that are not necessarily canonical. For $c = 0$, every boundary sequence is of course canonical. For $c > 0$, a boundary sequence that is not necessarily canonical is obtained when F is not traced beginning with v_{m-1} (as described above) but with an arbitrary vertex v_i ($i \in \{0, \ldots, m-1\}$) which is preceded by a convex edge. In this case, we say that the boundary sequence *corresponds* to v_{i+1} (index modulo m).
- If F contains a convex edge and the canonical valency sequence s is *periodical* with a periods, then b is periodical with a periods, too. Moreover, F contains exactly a markable boundary vertices.
- Since face–bridges cannot be incident with boundary faces, the degree of a boundary face having the boundary code $(c, \langle b_0, \ldots, b_{k-1}\rangle)$ is $c + \sum_{i=0}^{k-1} 2b_i$.

With the canonical boundary code, the conditions 6-8 in Section 8.1 can be reformulated. We will demonstrate the transformation extensively for one case and then give the results for all cases. These results can be derived analogously.

We recall the situation in Section 8.1 (condition 6): In order to perform an *intersection agglutination* of length l, we must have two boundary faces $F_3 = \langle u_0, a_0, a'_0, \ldots, u_{k-1}, a_{k-1}, a'_{k-1}\rangle$ and

$F_4 = \langle b'_{j-1}, b_{j-1}, w_{j-1}, \searrow, b'_0, b_0, w_0\rangle$ having the canonical boundary codes $(c_3, \langle b_0, \nearrow, b_{k_3-1}\rangle)$ for F_3 and $(c_4, \langle d_0, \nearrow, d_{k_4-1}\rangle)$ for F_4. In the following description, all indices i in b_i are expressed modulo k_3 and all indices i' in $d_{i'}$ are expressed modulo k_4. Suppose the vertices in the boundary faces that will be pairwisely identified with each other are $u_0, \nearrow, u_l$ and $w_0, \nearrow, w_l$ as in Section 8.1. Suppose l is odd. Then $\langle \mathrm{val}(u_{k-1}), \mathrm{val}(u_0), \mathrm{val}(u_1), \nearrow, \mathrm{val}(u_l), \mathrm{val}(u_{l+1})\rangle = \langle 2,2,2\rangle\circ\langle 3,2\rangle^{\frac{1}{2}(l-1)}\circ\langle 2\rangle$ and $\langle \mathrm{val}(w_{l+1}), \mathrm{val}(w_l), \searrow, \mathrm{val}(w_0), \mathrm{val}(w_{j-1})\rangle = \langle 2,2,2\rangle\circ\langle 3,2\rangle^{\frac{1}{2}(l-1)}\circ\langle 2\rangle$. These parts of the complete valency sequences of F_3 and F_4 correspond to the following parts of the boundary sequences: $\langle \dots, 0, \frac{l-1}{2}, \dots\rangle$ and $\langle \dots, 0, \frac{l-1}{2}, \dots\rangle$. Hence $b_i = 0$, $d_{i'} = 0$, and $b_{i+1} = d_{i'+1}$ for some i and i'. Thus it must hold $c_3 \geq 2$ and $c_4 \geq 2$ (and hence $k_3 = c_3$ and $k_4 = c_4$). Moreover, $l < k-1$ and $l < j-1$ imply that even $c_3 \geq 3$ and $c_4 \geq 3$. Due to Lemma 6.17, the number of convex edges after the agglutination is $c_3 + c_4 - 6$. Hence if $c_3 + c_4 \geq 7$, then due to condition 8 in Section 8.1, it must hold $\mathrm{val}(u_{k-2}) = 2$ (because u_{k-2} is the vertex in F_3 that corresponds to v_{m-2} in the new boundary face F_1) and hence $c_3 \geq 4$ and $b_{i-1} = 0$ (in particular, $c_3 = 3$ and $c_4 \geq 4$ is impossible).

In the following description, the results for all cases are given. Figure 8.6 shows examples for all cases. The patch that is obtained by the agglutination is P, and the zig–zag path that contains the identified edges in P is p. The new boundary face in P containing the start vertex v_0 of p is $F_1 = \langle v_0, h_0, h'_0, \nearrow, v_{m-1}, h_{m-1}, h'_{m-1}\rangle$.

(1) In order to perform an *intersection agglutination*, we must have two boundary faces F_3 and F_4 with the canonical boundary codes $(c_3, \langle b_0, \nearrow, b_{c_3-1}\rangle)$ for F_3 and $(c_4, \langle d_0, \nearrow, d_{c_4-1}\rangle)$ for F_4 (where $c_3, c_4 > 0$). In the following description, all indices i in b_i are expressed modulo c_3 and all indices i' in $d_{i'}$ are expressed modulo c_4.

(a) Let both $c_3 \geq 3$ and $c_4 = 3$ or both $c_3 \geq 4$ and $c_4 > 3$. If $b_i = 0$, $d_{i'} = 0$, and $b_{i+1} = d_{i'+1}$ for some i and i', and if $b_{i-1} = 0$ for the case that $c_3 \geq 4$, then an intersection agglutination of odd length $l = 2b_{i+1} + 1$ can be performed. Corresponding to v_0, the resulting new boundary face F_1 has the following boundary code C_1 (sometimes the terms in C_1 are not condensed as much as possible in order to clarify the relationship between the different cases; for example, it is always written "$c_3 + c_4 - 6$" even if the values of c_3 and c_4 are known):

c_3	c_4	C_1	remark
3	3	$(c_3 + c_4 - 6, \langle d_{i'+2} + b_{i+2} + 2\rangle)$	
4	3	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+2} + b_{i+2} + 2\rangle)$	$b_{i-1} = 0$
4	4	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+2} + 1, d_{i'-1} + b_{i+2} + 1\rangle)$	$b_{i-1} = 0$
4	> 4	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+2} + 1, d_{i'+3}, \nearrow, d_{i'+c_4-2},$ $d_{i'-1} + b_{i+2} + 1\rangle)$	$b_{i-1} = 0$
> 4	3	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+2} + b_{i+2} + 2, b_{i+3}, \nearrow, b_{i+c_3-2}\rangle)$	$b_{i-1} = 0$
> 4	4	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+2} + 1, d_{i'-1} + b_{i+2} + 1,$ $b_{i+3}, \nearrow, b_{i+c_3-2}\rangle)$	$b_{i-1} = 0$
> 4	> 4	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+2} + 1, d_{i'+3}, \nearrow, d_{i'+c_4-2},$ $d_{i'-1} + b_{i+2} + 1, b_{i+3}, \nearrow, b_{i+c_3-2}\rangle)$	$b_{i-1} = 0$

(b) Let both $c_3 \geq 4$ and $c_4 = 2$ or both $c_3 \geq 5$ and $c_4 > 2$. If $b_i = 0$, $b_{i+2} = 0$, and $b_{i+1} + 1 = d_{i'}$ for some i and i', and if $b_{i-1} = 0$ for the case that $c_3 \geq 5$, then an intersection agglutination of even length $l = 2d_{i'}$ can be performed. The new boundary face F_1 has the following boundary code C_1 corresponding to v_0:

c_3	c_4	C_1	remark
4	2	$(c_3 + c_4 - 6, \langle d_{i'+1} + b_{i+3} + 2\rangle)$	
5	2	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+1} + b_{i+3} + 2\rangle)$	$b_{i-1} = 0$
5	3	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+1} + 1, d_{i'-1} + b_{i+3} + 1\rangle)$	$b_{i-1} = 0$
5	> 3	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+1} + 1, d_{i'+2}, \nearrow, d_{i'+c_4-2}, d_{i'-1} + b_{i+3} + 1\rangle)$	$b_{i-1} = 0$
> 5	2	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+1} + b_{i+3} + 2, b_{i+4}, \nearrow, b_{i+c_3-2}\rangle)$	$b_{i-1} = 0$
> 5	3	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+1} + 1, d_{i'-1} + b_{i+3} + 1, b_{i+4}, \nearrow, b_{i+c_3-2}\rangle)$	$b_{i-1} = 0$
> 5	> 3	$(c_3 + c_4 - 6, \langle b_{i-1} + d_{i'+1} + 1, d_{i'+2}, \nearrow, d_{i'+c_4-2}, d_{i'-1} + b_{i+3} + 1, b_{i+4}, \nearrow, b_{i+c_3-2}\rangle)$	$b_{i-1} = 0$

(2) In order to perform a *linkage agglutination*, we must have a boundary face F_3 with the canonical boundary code $(c_3, \langle b_0, \nearrow, b_{c_3-1}\rangle)$ where $c_3 \geq 6$. In the following description, all indices i in b_i are expressed modulo c_3.

(a) If $b_i = 0$, $b_{i'} = 0$, and $b_{i+1} = b_{i'+1}$ for some i and i' with $|i' - i| \geq 3$, and if $b_{i-1} = 0$ for the case that $i - (i' + 3) = 0$ (modulo c_3), then a linkage agglutination of odd length $l = 2b_{i+1} + 1$ can be performed. Let $c_1 := i - (i' + 3)$ (modulo c_3) and $c_2 := i' - (i + 3)$ (modulo c_3). After the agglutination, the new boundary face F_1 has the following boundary code C_1 corresponding to v_0, and the new boundary face F_2 containing the end vertex of p has the following boundary code C_2:

c_1	C_1	remark
0	$(c_1, \langle b_{i'+2} + 1\rangle)$	
1	$(c_1, \langle b_{i-1} + b_{i'+2} + 1\rangle)$	$b_{i-1} = 0$
> 1	$(c_1, \langle b_{i-1} + b_{i'+2} + 1, b_{i'+3}, \nearrow, b_{i'+c_1+1}\rangle)$	$b_{i-1} = 0$

c_2	C_2	remark
0	$(c_2, \langle b_{i+2} + 1\rangle)$	
1	$(c_2, \langle b_{i'-1} + b_{i+2} + 1\rangle)$	
> 1	$(c_2, \langle b_{i'-1} + b_{i+2} + 1, b_{i+3}, \nearrow, b_{i+c_2+1}\rangle)$	

(b) If $b_i = 0$, $b_{i+2} = 0$, and $b_{i+1} + 1 = b_{i'}$ for some i and i' with $i' \notin \{i-1, i, i+1, i+2, i+3\}$ (modulo c_3), and if $b_{i-1} = 0$ for the case that $i = i' + 2$ (modulo c_3), then a linkage agglutination of even length $l = 2b_{i'}$ can be performed. Let $c_1 := i - (i' + 2)$ (modulo c_3) and $c_2 := i' - (i + 4)$ (modulo c_3). After the agglutination, the new boundary face F_1 has the following boundary code C_1 corresponding to v_0, and the new boundary face F_2 containing the end vertex of p has the following boundary code C_2:

c_1	C_1	remark
0	$(c_1, \langle b_{i'+1} + 1\rangle)$	
1	$(c_1, \langle b_{i-1} + b_{i'+1} + 1\rangle)$	$b_{i-1} = 0$
> 1	$(c_1, \langle b_{i-1} + b_{i'+1} + 1, b_{i'+2}, \nearrow, b_{i'+c_1}\rangle)$	$b_{i-1} = 0$

c_2	C_2	remark
0	$(c_2, \langle b_{i+3} + 1\rangle)$	
1	$(c_2, \langle b_{i'-1} + b_{i+3} + 1\rangle)$	
> 1	$(c_2, \langle b_{i'-1} + b_{i+3} + 1, b_{i+4}, \nearrow, b_{i+c_2+2}\rangle)$	

For C_2, the entry $b_{i'-1}$ never needs to be 0 because the end vertex w of p never needs to be preceded by a convex edge in F_2. Consequently, in general C_2 does not correspond to w.

(3) In order to perform an *enclosure agglutination*, we must have two boundary faces F_3 and F_4 with the canonical boundary codes $(c_3, \langle b_0, \nearrow, b_{c_3-1}\rangle)$ for F_3 and $(c_4, \langle d_0, \nearrow, d_{c_4-1}\rangle)$ for F_4 (where $c_3, c_4 > 0$). In the following description, all indices i in b_i are expressed modulo c_3.

(a) Let $c_4 = 1$ and $c_3 \geq 5$. If $b_i = 0$, $b_{i+2} = 0$, and $b_{i+3} = b_{i+1} + d_0 + 1$, and if $b_{i-1} = 0$ for the case that $c_3 \geq 6$, then an enclosure agglutination can be performed where the enclosure entry length l_e is even, namely $l_e = 2b_{i+1} + 2$, and the enclosure cycle length l_c is odd, namely $l_c = 2d_0 + 1$.

(b) Let $c_4 = 1$ and $c_3 \geq 5$. If $b_i = 0$, $b_{i+2} = 0$, and $b_{i+1} = b_{i+3} + d_0$, and if $b_{i-1} = 0$ for the case that $c_3 \geq 6$, then an enclosure agglutination can be performed where the enclosure entry length l_e is odd, namely $l_e = 2b_{i+3} + 1$, and the enclosure cycle length l_c is odd, too, namely $l_c = 2d_0 + 1$.

(c) Let $c_4 = 2$ and $c_3 \geq 4$. If $b_i = 0$, $d_1 = 0$, and $b_{i+2} = b_{i+1} + d_0 + 2$, and if $b_{i-1} = 0$ for the case that $c_3 \geq 5$, then an enclosure agglutination can be performed where the enclosure entry length l_e is even, namely $l_e = 2b_{i+1} + 2$, and the enclosure cycle length l_c is even, too, namely $l_c = 2d_0 + 2$.

(d) Let $c_4 = 2$ and $c_3 \geq 4$. If $b_i = 0$, $d_1 = 0$, and $b_{i+1} = b_{i+2} + d_0 + 1$, and if $b_{i-1} = 0$ for the case that $c_3 \geq 5$, then an enclosure agglutination can be performed where the enclosure entry length l_e is odd, namely $l_e = 2b_{i+2} + 1$, and the enclosure cycle length l_c is even, namely $l_c = 2d_0 + 2$.

Let $c := 5$ for the cases 1 and 2, and let $c := 4$ for the cases 3 and 4. Thus c is the minimum for c_3. In all cases, the resulting new boundary face F_1 has the following boundary code C_1 corresponding to v_0:

c_3	C_1	remark
c	$(c_3 - c, \langle b_{i+c-1} + 1\rangle)$	
c+1	$(c_3 - c, \langle b_{i-1} + b_{i+c-1} + 1\rangle)$	$b_{i-1} = 0$
$> c + 1$	$(c_3 - c, \langle b_{i-1} + b_{i+c-1} + 1, b_{i+c}, \nearrow, b_{i+c_3-2}\rangle)$	$b_{i-1} = 0$

Following these rules, the agglutinated zig–zag path p is always continued to the right (condition 6) and the obtained patch P is always pseudo–convex (condition 7). In order to secure that p is the *canonical* zig–zag path in P (condition 8), the two conditions described in Section 8.1 (page 112) can be verified as follows:

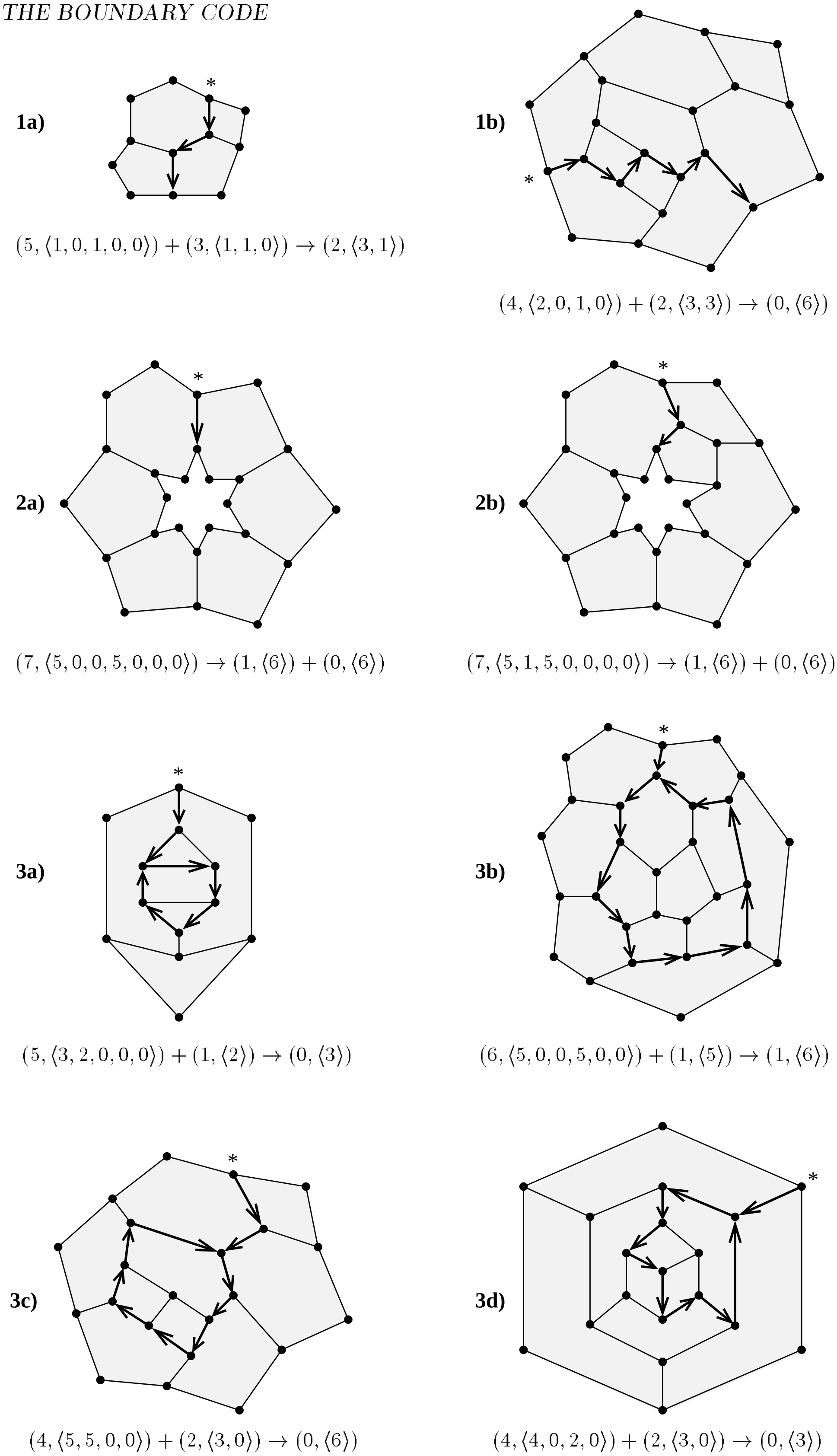
1a)
$(5, \langle 1,0,1,0,0\rangle) + (3, \langle 1,1,0\rangle) \to (2, \langle 3,1\rangle)$
1b)
$(4, \langle 2,0,1,0\rangle) + (2, \langle 3,3\rangle) \to (0, \langle 6\rangle)$
2a)
$(7, \langle 5,0,0,5,0,0,0\rangle) \to (1, \langle 6\rangle) + (0, \langle 6\rangle)$
2b)
$(7, \langle 5,1,5,0,0,0,0\rangle) \to (1, \langle 6\rangle) + (0, \langle 6\rangle)$
3a)
$(5, \langle 3,2,0,0,0\rangle) + (1, \langle 2\rangle) \to (0, \langle 3\rangle)$
3b)
$(6, \langle 5,0,0,5,0,0\rangle) + (1, \langle 5\rangle) \to (1, \langle 6\rangle)$
3c)
$(4, \langle 5,5,0,0\rangle) + (2, \langle 3,0\rangle) \to (0, \langle 6\rangle)$
3d)
$(4, \langle 4,0,2,0\rangle) + (2, \langle 3,0\rangle) \to (0, \langle 3\rangle)$

Figure 8.6

- v_0 must be markable, i.e. the boundary code $C_1 = (c_1, b)$ of F_1 denoted as above must be canonical, i.e. there must not exist a cyclical shift of b that yields a lexicographically larger boundary sequence. If F_1 contains a convex edge, then unlike before, it always holds $\mathrm{val}(v_{m-2}) = 2$, i.e. a convex edge precedes v_{m-1} while tracing F_1. This is secured by the above condition $b_{i-1} = 0$. Thus, the verification if v_0 is markable is faster than before: $\mathrm{val}(v_{m-2})$ does not need to be verified, and the boundary sequence of a boundary face is in general much shorter than the corresponding valency sequence.
- F_1 must have the highest mark–quality among all boundary faces in P. This is checked as before (see page 112).

The *boundary code of patches* is defined as follows: Let $\{F_1, \ldots, F_m\}$ be the set of boundary faces of a patch P sorted by their total qualities (in descending order: F_1 has the highest total quality). Let $s_i = (c_i, b_i)$ with $b_i = \langle b_{i,1}, \ldots, b_{i,k_i}\rangle$ be the canonical boundary code of F_i where $k_i = \max\{c_i, 1\}$. Then the **boundary code** of P is a triple (m, c, b) where $c = \langle c_1, \ldots, c_m\rangle$ and $b = b_1 \circ \ldots \circ b_m$. With the help of c, the canonical boundary sequence of a single boundary face can easily be extracted out of b.

8.3 The complete construction

The goal of the bottom–up approach for patches is to construct and accept all canonical patches. However, no pairwise mark–isomorphic patches must be accepted. Since the canoncial ancestors of a patch P always have a smaller *size* than P, it is secured that P can be constructed as soon as all patches having a size smaller than $s(P)$ are constructed. Hence in order not to miss any canonical patch, the patches are constructed with increasing size. So every triple that might represent the size of a patch is determined in increasing order, and when a triple (f', g', b) is given, then all patches P with $s(P) = (f', g', b)$ are constructed. The number of different possible patch sizes is limited: It must hold $b \in \{1, \ldots, g - g' + 1\}$ due to condition 5 in Section 8.1, $g' \in \{0, \ldots, g\}$ due to condition 4 in Section 8.1, and $f' \in \{1, \ldots, f\}$ due to condition 3 in Section 8.1 where $f = \frac{1}{2}n + 2 - 2g$ (see formula 1.2).

Since we construct patches with increasing size, the tiles are constructed first. Since all vertices in a tile are isomorphic to each other, an arbitrary vertex in every tile can be marked. After constructing the tiles, the patches not having minimal size are constructed. For the complete construction, we must consider 3 questions:

(1) In order to construct a patch P of size $s(P)$, what are the possible sizes of the canonical ancestors of P?

(2) When the patches that are to be agglutinated with each other (in order to obtain a patch of size $s(P)$) are fixed, which boundary faces may be agglutinated with each other?

(3) When the boundary faces that are to be agglutinated with each other are fixed, which parts of them may be agglutinated with each other?

These questions will be discussed below. For this, we need to define *equivalent agglutinations*: Let P_1 and P_2 be two patches that are obtained by performing agglutinations along zig–zag paths p_1 and p_2, respectively. Let v_1 be the start vertex of p_1 and let v_2 be the start vertex of p_2. Then the corresponding agglutinations are called **equivalent** if and only if P_1 and P_2 are mark–isomorphic and a corresponding patch isomorphism maps p_1 onto p_2 (so that v_1 is mapped onto v_2). Moreover, let M_1 and M_2 be two cubic maps that are obtained by performing agglutinations along Petrie paths p_1 and p_2, respectively. Let v_1 be the start vertex of p_1 and let v_2 be the start vertex of p_2. Then the corresponding agglutinations are called **equivalent** if and only if M_1 and M_2 are isomorphic and a corresponding map isomorphism maps p_1 onto p_2 (so that v_1 is mapped onto v_2).

Of course, we must avoid performing equivalent agglutinations because neither in the case of patches nor in the case of cubic maps, the canonicity check is able to distinguish equivalent agglutinations. So if one of the agglutinations is canonical, then the other agglutination is canonical, too, so that we would accept pairwise mark–isomorphic patches or pairwise isomorphic cubic maps, respectively. The trivial way to perform two equivalent agglutinations is given by performing exactly the same agglutination twice. This is of course not done during the construction. However, there are some more situations in which equivalent agglutinations might be performed. They are mentioned below.

Question 1: In order to construct every patch P of size $s(P) = (f', g', b)$, the following steps must be performed (compare Section 6.6):

- If $g' > 0$, then for every patch Q of size $s(Q) = (f', g' - 1, b + 1)$, it must be checked whether an intersection or enclosure agglutination of Q (with itself) can be performed.
- For every patch Q of size $s(Q) = (f', g', b - 1)$, it must be checked whether a linkage agglutination of Q (with itself) can be performed.
- For every pair $\{Q_1, Q_2\}$ of patches with $s(Q_1) = (f_1, g_1, b_1)$ and $s(Q_2) = (f_2, g_2, b_2)$ satisfying $f' = f_1 + f_2$, $g' = g_1 + g_2$, and $b = b_1 + b_2 - 1$, it must be checked whether an intersection or enclosure agglutination of Q_1 and Q_2 can be performed. If Q_1 and Q_2 are not mark–isomorphic, then two agglutination attempts must take place: During the first attempt, Q_1 contains the boundary face F_3 defined in Section 8.1, and during the second attempt Q_2 contains F_3.

Question 2: Assume we have a patch Q or a *sorted* pair (Q_1, Q_2) of patches (so that the roles are fixed: Q_1 contains F_3 and Q_2 contains F_4). Then the next task is to find out which boundary faces may be agglutinated with each other.

- If a pair (Q_1, Q_2) of patches is given, then in principle, every boundary face in Q_1 may be agglutinated with every boundary face in Q_2. However, if Q_1 contains boundary faces having the same total quality (so that their marked vertices are mark–isomorphic), then only one of those faces may contribute to agglutinations because otherwise, we would obviously perform equivalent agglutinations. The same holds if Q_2 contains boundary faces having the same total quality.
- If a patch Q is agglutinated with itself along an intersection or enclosure path, then it holds that if Q contains boundary faces having the same total quality, then in order to avoid equivalent agglutinations, only one of those faces may be taken as F_3 defined in Section 8.1. However, every remaining boundary face in Q may be taken as F_4. This holds because if u, v, and w are three different marked vertices in Q and v is mark–isomorphic to w, then nevertheless the pair (u, v) is *not* mark–isomorphic to the pair (u, w), i.e. there exists no patch automorphism that maps v on w and u on itself because due to Section 7.3, no patch automorphism except the identity function fixes a boundary vertex. If (u, v) is mark–isomorphic to (w, u), then this does not cause any problem because then the face F containing w has the same total quality as the face containing u. Hence F may not be taken as F_3 anyway.
- If a patch Q is agglutinated with itself along a linkage path, then it holds that if there exist boundary faces having the same total quality, then of course only one of those faces may be

taken as F_3 (defined in Section 8.1).

Question 3: Assume we have selected 1 or 2 boundary faces that are to be agglutinated with each other. Finally, we must determine which parts of the selected boundary faces may be agglutinated with each other.

- When two boundary faces F_3 and F_4 shall be agglutinated with each other (along an intersection or enclosure path), then due to Section 8.2, each of the two boundary faces must contain at least one convex edge, and due to Lemma 6.17, altogether they must contain at least 6 convex edges. Let $(c_3, \langle b_0, \ldots, b_{c_3-1}\rangle)$ and $(c_4, \langle d_0, \ldots, d_{c_4-1}\rangle)$ be the canonical boundary codes of F_3 and F_4, respectively. In general, for every pair (i, i') with $i \in \{0, \ldots, c_3 - 1\}$ and $i' \in \{0, \ldots, c_4 - 1\}$, it must be checked whether the conditions given in Section 8.2 are matched. For example, if $b_i = 0$, $d_{i'} = 0$, and $b_{i+1} = d_{i'+1}$ (indices modulo c_3 for i and modulo c_4 for i'), and if $b_{i-1} = 0$ for the case that $c_3 \geq 4$, then F_3 and F_4 can be agglutinated along an intersection path of odd length. However, sometimes not every pair (i, i') may be considered. Only the last period of a boundary sequence may be considered in order to perform an agglutination. For example, if the boundary sequence of F_3 is periodical with a periods, then c_3 is a multiple of a and it must hold $i \in \{c_3 - \frac{c_3}{a}, \ldots, c_3 - 1\}$. Otherwise a marking rule would be violated so that the agglutination would not be the inverse of a canonical cut. Figure 8.7 shows an example patch P: Here the canonical boundary code of the boundary face F_3 is $(6, \langle 1, 0, 1, 0, 1, 0\rangle)$. The vertices u, v, and w are markable, w is actually marked. It must hold $i \in \{4, 5\}$. If $i = 5$, then the corresponding intersection agglutination shown in the middle of Figure 8.7 is canonical because due to the additional marking rule for intersections (see Section 7.4), the boundary face must be traced starting with vertex x. And in this case, the vertex w is the first markable vertex that is approached. Since w is actually marked in P, the agglutination is actually the inverse of a canonical cut (provided that the additional marking rule for intersections is satisfied also for the other boundary face). However, if $i = 1$, then the corresponding intersection agglutination shown at the bottom of Figure 8.7 is not canonical because due to the additional marking rule for intersections, the vertex u would have to be marked if the agglutination would be revoked (because starting with z, vertex u is the first markable vertex that is approached while tracing the face). So this agglutination would be canonical if the vertex u instead of w was marked in P. If we presume that the construction of patches is complete, then there exists a patch Q that differs from P only by the marked vertex u instead of w, and hence the agglutination will be performed when Q is considered instead of P (where in this case, it holds $i = 5$).

  In general, the restriction that only the last period of a boundary sequence is considered for agglutinations secures that each possible agglutination of two fixed boundary faces is performed *exactly* once: For every period that is omitted, it holds that either the period corresponds to a markable boundary vertex which is marked in a different patch, or the period corresponds to a markable vertex which is mark–isomorphic to the marked vertex. For an example of the latter case, replace all inner vertices of P in Figure 8.7 by one inner vertex. The resulting patch contains 3 hexagons and 3 markable vertices that are pairwise mark–isomorphic. Hence considering also the first two periods of the boundary sequence would result in 3 equivalent agglutinations.

  In particular, using marked patches, we perform every possible pairwise non–equivalent agglutination exactly once without computing any edge code. If we would construct *unmarked* patches, then we would have to compute edge codes in order to find out whether the vertices u, v, and w in Figure 8.7 are pairwise mark–isomorphic (so that only one period of the canonical boundary sequence must be considered) or not (so that all periods must be considered). This fact makes the construction of marked patches faster than the construction

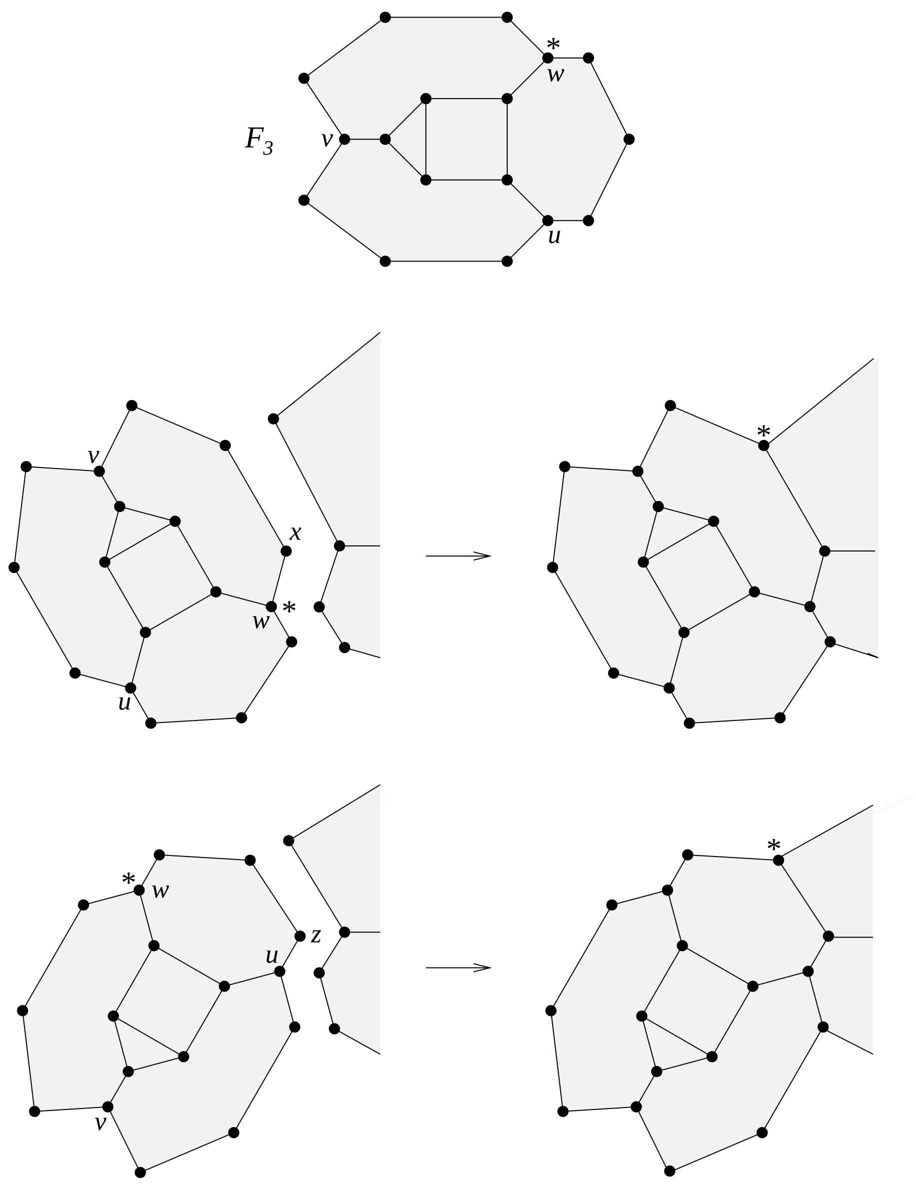

Figure 8.7

of unmarked patches.

- If one boundary face F_3 is agglutinated with itself along a linkage path, then the restrictions to i and i' defined in Section 8.2 are as follows: i must correspond to the last period of the boundary sequence of F_3 as usual. However, for i' there are no further restrictions except those given in Section 8.2 ($|i'-i| \geq 3$ in case (a), $i' \notin \{i-1, i, i+1, i+2, i+3\}$ in case (b)) because for a fixed i, every different value of i' clearly yields different boundary codes for the resulting new boundary faces F_1 and F_2 (because for different values of i', the lengths of the resulting boundary sequences of F_1 — and of F_2, too — are pairwise different). As described in Section 8.1, there might be more than one markable boundary vertex incident with F_2. Thus the linkage case is the only case where edge–codes might have to be computed in order to determine whether an agglutination is equivalent to another agglutination.

8.4 Storing the patches

As we have seen in Section 8.3, it is necessary to find patches of given sizes as quickly as possible. In particular, all accepted patches need to be stored because later, they might be agglutinated with themselves or with other patches. Moreover, as we will see in Section 8.6, it is necessary to find patches of a given size and with a given boundary code as quickly as possible also during the construction of cubic maps (or to find out that an appropriate patch does not exist). For both reasons, the patches are stored in a *tree* which is called the **patch tree**. The patches are stored at the leaves, and the tree–path connecting the tree–source with the leaf where a patch P must be stored is determined by the number of inner faces, the genus, and the boundary code (m, c, b) of P (in this order). To every leaf, a list of all patches having the same number of inner faces, the same genus, and the same boundary code is assigned (the leaf "contains" the patches). Figure 8.8 shows an example patch tree. Here leaf A contains a list of all accepted patches having 8 inner faces, genus 0, and the boundary code $(1, \langle 2\rangle, \langle 2, 1\rangle)$. Leaf B contains a list of all patches having 8 inner faces, genus 0, and the boundary code $(2, \langle 0, 0\rangle, \langle 4, 4\rangle)$ so that the patches each contain two boundary faces having the canonical boundary code $(0, \langle 4\rangle)$ each. Leaf C contains a list of all patches having 8 inner faces, genus 0, and the boundary code $(2, \langle 2, 1\rangle, \langle 2, 2, 2\rangle)$ so that the patches each contain two boundary faces where the canonical boundary code of the first boundary face is $(2, \langle 2, 2\rangle)$ and the canonical boundary code of the second boundary face is $(1, \langle 2\rangle)$.

The patch tree is updated during the construction of the patches, i.e. new branches, nodes, and leaves are inserted into the patch tree as soon as patches are accepted whose corresponding tree–paths require the new leaves (and branches). Thus, the patch tree is kept as "slim" as possible, and every leaf contains at least one patch.

In order to store a patch, it is necessary to encode it. For that, the edge code can be used. However, the length of the edge code of a patch having order n increases linearly with n. We prefer an encoding that has a constant memory consumption. For that, we use the fact that the structure of a patch P can be derived from the canonical ancestor(s) of P. So we do not store the whole patch but only the information how it can be derived out of its canonical ancestor(s). There is an upper limit for the memory consumption of this information:

- Pointers on the canonical ancestor(s) must be stored. In a computer program, a pointer is a number of limited size (depending on the memory that is provided by the computer). Since a patch has at most 2 canonical ancestors, at most 2 pointers must be stored.
- The kind of agglutination must be stored. This can be done by three different numbers,

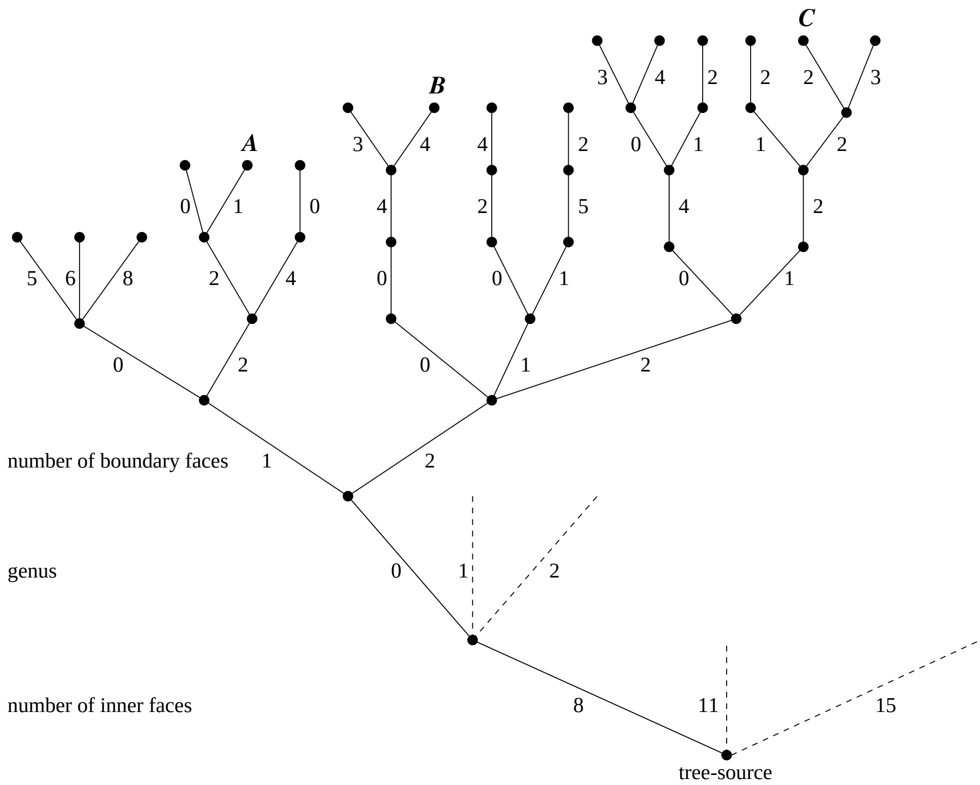

Figure 8.8

e.g. "0" denotes an intersection agglutination, "1" denotes a linkage agglutination, and "2" denotes an enclosure agglutination.

- It must be stored which boundary faces are agglutinated with each other. The boundary faces of a patch can be sorted by their total qualities (e.g. in descending order). So in each canonical ancestor of P, the boundary faces are sorted by their total qualities, and then the boundary faces that are agglutinated with each other can be numbered.
  - If P has one canonical ancestor Q, then the following information must be stored: In the linkage case, we must store one number, namely the number of the boundary face in Q which is agglutinated with itself. In the intersection or enclosure case, we must store a pair (a, b) where a is the number of the boundary face in Q that plays the role of F_3 in Section 8.2 (pages 116 and 118) and b is the number of the boundary face in Q that plays the role of F_4. Thus 2 numbers must be stored.
  - If P has two canonical ancestors Q_1 and Q_2, then one boundary face in Q_1 is agglutinated with one boundary face in Q_2 along an intersection or an enclosure path. In both cases, we must store a pair (a, b) where a is the number of the boundary face in Q_1 and b is the the number of the boundary face in Q_2 that is agglutinated. Moreover, we must store whether the boundary face of Q_1 or Q_2 plays the role of F_1 (Section 8.2). Thus 3 numbers must be stored.
- The parts of the boundary faces that are agglutinated with each other must be specified. In the intersection and linkage case, this is done by storing the numbers i and i' (see Section 8.2). In the enclosure case, only the number i needs to be stored.
- Finally, in the case of a linkage agglutination, the marked vertex v in the boundary face F_2 in P containing the end vertex w of the zig–zag path must be stored. This is done by

storing the **offset** between v and w in F_2, i.e. the number of edges that are passed when the boundary face F_2 is traced from v to w.

When a patch does not have ancestors, then it is a tile and the only information that needs to be stored is the degree of the tile. The information that is stored for a patch P in the patch tree is denoted by $t(P)$. However, wherever the technical details are not important, we will — for convenience — often say that P (itself) is stored.

By the construction of a patch $P = (((V, H, E, T), \Pi), \mathcal{F}, W)$, one obtains the complete incidence structure of P, i.e. V, H, E, T, Π, $\mathcal{F}$, and W are known. By the way the patches are stored in the patch tree, everytime a patch P is constructed, it must be constructed recursively. This takes more time than constructing P iteratively out of a corresponding edge code. This is a disadvantage of the way the patches are stored. On the other hand, during the patch construction process, it is often not necessary to actually construct a patch at all (before the information listed above is stored): In many cases, it suffices to verify the boundary code of a patch in order to decide whether an agglutination is possible and canonical. The only canonicity check where a patch must actually be *constructed* during the patch construction process is when the edge code of the resulting patch must be computed in order to decide whether an agglutination is canonical. Moreover, in Section 8.5, some additional tests are described that require the complete incidence structure of a patch, too. But anyway, as it will be explained in Section 9.6, it is more important to save memory (by using an encoding having a constant memory cunsumption) than to save a little time (by constructing a patch iteratively).

In order to construct a patch P recursively, one does not only need pointers on the canonical ancestors of P but also the boundary codes of the ancestors. They are not stored, but since the construction is recursive, the canonical ancestors of P are constructed out of their canonical ancestors and so on. So starting with the tiles, the boundary code of every patch in the construction can be derived from the boundary codes of its canonical ancestors according to Section 8.2.

So *constructing* a patch P implies that afterwards, the complete incidence structure is given. A patch is **generated** if all the information that is stored in the patch tree is known. Thus, a patch is *generated* if all the information is known which is needed to *construct* the patch. Thus the process of finding and storing all canonical patches is actually a *patch generation process* rather than a *patch construction process*. Throughout the remaining text, we will use the term "construction" only if really the complete incidence structure of the resulting patch is obtained. Otherwise we will use the term "generation".

For further considerations, we introduce the following definitions: An **iterated ancestor** of a patch P or a cubic map M is a patch that can be obtained by applying a sequence of (not necessarily canonical) cuts to P or M, respectively (where, of course, only the first cut is applied to P or M itself, and every further cut is applied to a patch that has been obtained by the previous cut). A patch P or a cubic map M is called an **iterated descendant** of a patch Q if and only if Q is an iterated ancestor of P or M, respectively. Of course, every iterated *canonical* ancestor of a patch P or a cubic map M is also an iterated ancestor of P or M, respectively, and every iterated *canonical* descendant of a patch Q is also an iterated descendant of Q. Moreover, every iterated ancestor of a cubic map M is possibly canonical. Since we are only interested in iterated *canonical* ancestors and descendants, we could also do without these definitions. Nevertheless, we use these definitions in order to emphasize that some properties of patches derived in the following sections do not depend on the canonicity of a cut or an agglutination.

8.5 Tests on patches

Before an accepted patch P is stored, some tests must be run on P. The first test is the test for loops and multiple edges. There are some more tests that are useful if the accepted cubic maps must satisfy some special conditions. The tests help to detect patches that are not canonical although obtained by a canonical agglutination, and hence by the tests, these patches can also be rejected so that the required time and memory of the algorithm can be decreased. The time consumptions of the following tests are all linear in n_P. For every test, the patch P must be constructed.

Loops and multiple edges: If loops and multiple edges are not allowed in $\mathcal{M}(g, n, D, b_l, b_u)$, then for a generated patch P, it must eventually be tested whether P contains a multiple edge. An agglutination along a zig–zag path never yields additional loops, and hence a patch P never needs to be tested whether it contains a loop: If P contained a loop, then due to Chapter 6 (page 67), a 1–tile would have to be an iterated canonical ancestor of P so that loops would be allowed. If P contains a multiple edge, then every iterated descendant of P contains this multiple edge, too, and hence P must neither be stored nor be used to perform further agglutinations. Thus P can just be rejected. On the other hand, by rejecting patches containing multiple edges, it is secured that the ancestors of a patch P which is generated during the bottom–up approach never contain multiple edges. Hence P does not need to be tested for multiple edges if it can be secured that the last agglutination did not yield *additional* multiple edges.

- If P has only one canonical ancestor, then P must always be tested for multiple edges because the boundary edges that are identified with each other by the agglutination might be adjacent to a common inner edge e. Since the vertices that are incident with e are both 3–valent, they are never identified with each other by an agglutination. Hence — as we already know — loops will not occur and the situation is as shown in Figure 8.9.

  Moreover, if the canonical zig–zag path p in P is an enclosure path or a linkage path, then the new boundary face(s) might contain only one 3–valent boundary vertex, namely the start vertex of p or — in the linkage case — the end vertex of p. In these cases, it must be tested whether the degree of one of the new boundary faces is 2. Let F be a new boundary face in P (i.e. $F = F_1$ in the enclosure case and $F \in \{F_1, F_2\}$ in the linkage case where F_1 and F_2 are denoted in Section 8.2). If $\deg(F) = 2$, then P contains a double edge consisting of the two boundary edges incident with F. Since F does not contain concave edges but exactly one 3–valent boundary vertex, the canonical boundary code of F is $(0, \langle 1 \rangle)$.

  If p is an intersection path, then the new boundary face F_1 in P contains at least two 3–valent boundary vertices, namely the start vertex and the end vertex of p. Due to the pseudo–convexity of P, F_1 contains also at least two 2–valent boundary vertices so that $\deg(F_1) \geq 4$.
- If P has two canonical ancestors Q_1 and Q_2 and the canonical zig–zag path p in P is an intersection path, then P does not need to be tested for multiple edges: The degree of the boundary face F_1 containing the start vertex and the end vertex of p is at least 4 (the start vertex and the end vertex of p are both 3–valent; since P does not contain any concave edge, there must be at least two 2–valent boundary vertices contained in F_1). Moreover, no inner edge of Q_1 will be adjacent to any inner edge of Q_2 after the agglutination because by p, no two 3–valent boundary vertices will be identified with each other. Thus P cannot contain a double edge $\{a, b\}$ where a is originally an inner edge of Q_1 and b is originally an inner edge of Q_2.
- If P has two canonical ancestors Q_1 and Q_2 and the canonical zig–zag path p in P is an

enclosure path, then P must be tested for multiple edges provided that $\max D \geq 7$ (where $\max D := \max_{d \in D} d$): As in the intersection case, the enclosure cycle cannot cause any problem, but one ancestor, say Q_1, is agglutinated with itself along the enclosure entry so that as above, two identified boundary edges might be adjacent to a common inner edge e (Figure 8.9). In this case, Q_1 must contain an inner face of degree at least 7: Look at Figure 8.10. p contains the edges f, d, and c. It holds $\deg(A) \geq 7$ because A is incident with a, b, c, d, and e, c is a face–bridge (and therefore contributes 2 to $\deg(A)$), and since $\{a, b\}$ cannot be a double edge (otherwise a canonical ancestor of P would already contain a double edge), A must be incident with another edge x.

Moreover, the new boundary face F_1 might contain only one 3–valent boundary vertex. In this case, it must be tested whether $\deg(F_1) = 2$. As above, this can be done by testing the canonical boundary code of F_1.

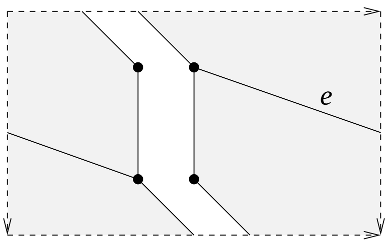

Figure 8.9

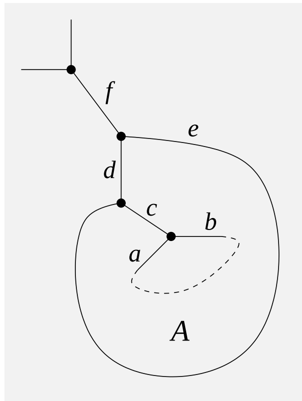

Figure 8.10

Semi–polyhedral maps: A cubic map M is called **semi–polyhedral** if it contains no face–bridge. Since a face–bridge in a patch P will remain in all iterated descendants of P, every patch containing a face–bridge can be rejected if only semi–polyhedral cubic maps are wanted. It is easy to show that if M is a simple planar cubic map, then M is semi–polyhedral if and only if M is 2–connected.

Polyhedral maps: A cubic map M is called **polyhedral** if it is semi–polyhedral and for every pair F_1, F_2 of faces in M, it holds that F_1 and F_2 have at most one common element. If only polyhedral cubic maps are wanted, then P can be rejected if two inner faces of P have more than one common element because the number of common elements of two inner faces in a patch monotonely increases with every agglutination. An example of a process where only polyhedral maps are wanted was given in Section 1.8. It is easy to show that if M is a simple planar cubic map, then M is polyhedral if and only if M is 3–connected.

Delaney graphs: In a Delaney graph G, the edges are partitioned into 3 disjoint sets E_0, E_1, and E_2 (see [42]). We will call the elements of E_0 *red edges* or 0*–edges*, the elements of E_1 *green edges* or 1*–edges*, and the elements of E_2 *blue edges* or 2*–edges*. Every vertex of G is incident with a red edge, a green edge, and a blue edge. If G is loopless, then there exists a bipartite map $M = (G, \Pi)$ such that each face F of M is incident with exactly two types of edges, i.e. either no red edge or no green edge or no blue edge is incident with F. Thus one could obtain a unique colouring of F by assigning the missing colour to F. So if a constructed map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ represents a Delaney graph, then each vertex is incident with a red face, a green face, and a blue face. Moreover, if M represents a Delaney graph, then M

(and hence every canonical patch) is semi–polyhedral because if M contained a face–bridge e, then the vertices u and v incident with e each would be incident with at most 2 faces. Thus u (and v) could not be incident with a red face, a green face, and a blue face.

In order to perform the test whether a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ might represent a Delaney graph, for every face degree $d \in D$, the information whether a d–face may be red, green, or blue must be provided. For one face degree d, several colours may be permitted. Then for every vertex v in M, it is tested by the degrees of the incident faces of v whether it is possible that v is incident with a red face, a green face, and a blue face. This test can be performed not only for the vertices of cubic maps, but also for the inner vertices of patches: If it is impossible that an inner vertex v of a patch P is incident with a red face, a green face, and a blue face, then this holds for every iterated descendant of P, too. A weaker test can be performed for the 3–valent boundary vertices of the patches: Every 3–valent boundary vertex v of a semi–polyhedral patch P is incident with 2 inner faces (because otherwise the inner edge being incident with v would be a face–bridge). If these faces must have the same colour, then this holds for every iterated descendant of P, too, and P can be rejected. Since these tests strongly restrict the incidence relationship of inner faces in the patches, many patches can be filtered out.

8.6 The construction of the cubic maps

Let $\mathcal{M}(g, n, D, b_l, b_u)$ be the set of maps that shall be constructed. The number f of faces in each map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is given by $f = \frac{1}{2}n + 2 - 2g$. According to Section 6.4, up to three patches must be agglutinated with each other along a Petrie path which is continued to the right. The remaining details are given in the following sections. Of course, the limits b_l and b_u must be considered when selecting patches that are to be agglutinated with each other.

8.6.1 Jordan curve paths

When a cubic map is cut along a Jordan curve path, then we obtain two boundary faces without convex edges. Hence we must agglutinate boundary faces having the common canonical boundary code $(0, \langle x \rangle)$ for some $x > 0$. Due to Section 6.4.1, there are two possibilities how to obtain a cubic map M:

(1) One patch P with $s(P) = (f, g-1, 2)$ having the boundary code $(2, \langle 0, 0 \rangle, \langle x, x \rangle)$ for any $x > 0$ is agglutinated with itself.

(2) Two patches P_1 and P_2 with $s(P_1) = (f_1, g_1, 1)$ and $s(P_2) = (f_2, g_2, 1)$ are agglutinated with each other if $f = f_1 + f_2$, $g = g_1 + g_2$, and the boundary code of both patches is $(1, \langle 0 \rangle, \langle x \rangle)$ for some common $x > 0$.

In either case, the length of the Jordan curve path p containing the identified edges in M is $2x$. Let h_0 be the start semi–edge of p (as in Figure 6.7 on page 78). It must be verified whether p is a canonical Petrie path in the constructed map M. This is done due to Section 7.1. If p is a canonical Petrie path in M, then M is accepted, otherwise M is rejected. From now on, the Petrie path p along which the agglutination has been performed (i.e. p contains the identified edges) is called the **candidate** (for the test whether it is the canonical Petrie path in the constructed cubic map).

In the first of the above cases, every patch P satisfying the desired properties must be taken once. If the two marked vertices v and v' in P are not mark–isomorphic, then two agglutinations must be performed where due to the marking rules, v becomes the start vertex of p in the first agglutination, and v' becomes the start vertex of p in the second agglutination (i.e. in the description of the marking rules in Section 7.4, page 106, we assign $w_0 := v$ and $u_1 := v'$ for the first agglutination, and we assign $u_1 := v$ and $w_0 := v'$ for the second agglutination). If v and v' are mark–isomorphic, then the two agglutinations would be equivalent so that only one agglutination is performed.

If two agglutinations are performed for P (i.e. v and v' are not mark–isomorphic), then the two resulting cubic maps M_1 and M_2 are isomorphic: Look at Figure 6.7. Imagine two copies of the sketch on the left side of the figure where to each label, the appropriate suffix M_1 or M_2 is assigned. The map isomorphism that maps M_1 onto M_2 is orientation preserving and maps $h_0^{M_1}$ onto ${h'}_0^{M_2}$. So M_1 and M_2 are isomorphic. However, the semi–edges $h_0^{M_1}$ and ${h'}_0^{M_1}$ cannot be mapped onto each other by an orientation preserving map automorphism because otherwise v and v' would be patch isomorphic and hence mark–isomorphic. Hence in the two maps M_1 and M_2, the candidates are not map isomorphic to each other, and hence at most one of the candidates can be a canonical Petrie path in the corresponding map. So it is not necessary to actually perform two agglutinations. It suffices to perform one agglutination and compare the two Jordan curve paths whose start semi–edges are h_0 and h'_0, respectively. The path with the higher quality is taken as the candidate, i.e. it is compared with the other Petrie paths in the map.

In the second case (where two patches are agglutinated with each other), f and g must each be partitioned into 2 non–negative summands in every possible way. This is done straightforward by count loops. Every pair $\{P_1, P_2\}$ of patches having the desired properties must be taken. If P_1 and P_2 are not mark–isomorphic, then as above, in principle two agglutinations must be performed (one could also say that every *sorted* pair (P_1, P_2) must be considered so that $(P_1, P_2) \neq (P_2, P_1)$), but as above, it suffices to perform one agglutination and compare the two Petrie paths whose start semi–edges are h_0 and h'_0, respectively. In particular, it can be prescribed that $s(P_1) \leq s(P_2)$. It is trivial to determine whether P_1 and P_2 are mark–isomorphic because the stored patches are pairwise non–mark–isomorphic. Thus P_1 and P_2 are mark–isomorphic if and only if they have been taken from the same place in the patch tree.

8.6.2 Dumbbell paths

Let M be a cubic map that is cut along a dumbbell path p which is continued to the right. Let F_1 be the resulting start side boundary, let F_2 be the resulting end side boundary, and let F_3 be the resulting main boundary in the ancestor(s). From Sections 6.4.2 and 8.2, the following data can easily be derived:

- If $l_1(p)$ is odd, then the canonical boundary code c_1 of F_1 is $c_1 = (1, \langle \frac{l_1(p)-1}{2} \rangle)$. If $l_1(p)$ is even, then $c_1 = (2, \langle \frac{l_1(p)-2}{2}, 0 \rangle)$.
- If $l_3(p)$ is odd, then the canonical boundary code c_2 of F_2 is $c_2 = (1, \langle \frac{l_3(p)-1}{2} \rangle)$. If $l_3(p)$ is even, then $c_2 = (2, \langle \frac{l_3(p)-2}{2}, 0 \rangle)$.

In order to note down the canonical boundary code c_3 of F_3, we must distinguish 8 cases depending on $l_1(p)$, $l_2(p)$, and $l_3(p)$. Table 8.1 shows the resulting boundary codes. Here x, y, and z can be derived from $l_1(p)$, $l_2(p)$, and $l_3(p)$: It holds $x = \lfloor \frac{l_1(p)-1}{2} \rfloor$, $y = \lfloor \frac{l_3(p)-1}{2} \rfloor$, and

$z = \lfloor \frac{l_2(p)-1}{2} \rfloor$ (so that $x, y, z \geq 0$).

Case	$l_1(p)$	$l_3(p)$	$l_2(p)$	c_1	c_2	c_3
1	even	even	odd	$(2, \langle x, 0 \rangle)$	$(2, \langle y, 0 \rangle)$	$(2, \langle y+1+z, x+1+z \rangle)$
2	even	odd	odd	$(2, \langle x, 0 \rangle)$	$(1, \langle y \rangle)$	$(3, \langle y+z, 0, x+1+z \rangle)$
3	odd	even	odd	$(1, \langle x \rangle)$	$(2, \langle y, 0 \rangle)$	$(3, \langle y+1+z, x+z, 0 \rangle)$
4	odd	odd	odd	$(1, \langle x \rangle)$	$(1, \langle y \rangle)$	$(4, \langle y+z, 0, x+z, 0 \rangle)$
5	even	even	even	$(2, \langle x, 0 \rangle)$	$(2, \langle y, 0 \rangle)$	$(2, \langle x+y+z+3, z \rangle)$
6	even	odd	even	$(2, \langle x, 0 \rangle)$	$(1, \langle y \rangle)$	$(3, \langle x+y+z+2, z, 0 \rangle)$
7	odd	even	even	$(1, \langle x \rangle)$	$(2, \langle y, 0 \rangle)$	$(3, \langle x+y+z+2, 0, z \rangle)$
8	odd	odd	even	$(1, \langle x \rangle)$	$(1, \langle y \rangle)$	$(4, \langle x+y+z+1, 0, z, 0 \rangle)$

Table 8.1

In the first 4 cases in Table 8.1, it might be necessary to shift the boundary sequence of c_3 depending on the values of x and y in order to obtain a lexicographically maximal (and thus canonical) sequence. However, if p is canonical, then it always holds $l_1(p) \leq l_3(p)$ because otherwise, due to Section 7.1, the inverse $\overline{p}$ would have a higher quality than p. Thus $x \leq y$ and in case 2 even $x+1 \leq y$ so that the boundary sequences denoted in Table 8.1 are always lexicographically maximal among all shifted sequences except maybe in case 2 (if and only if $x+1=y$; in this case, the canonical boundary code is $c_3 = (3, \langle x+1+z, y+z, 0 \rangle)$).

So in order to obtain maps of $\mathcal{M}(g, n, D, b_l, b_u)$, we need to agglutinate patches that contain altogether 3 boundary faces. Each boundary face must contain at least one convex edge. The 3 boundary faces must altogether contain 6 convex edges. So due to Section 6.4.2, there are three possibilities how to obtain the cubic maps:

(1) One patch P with $s(P) = (f, g-2, 3)$ can be agglutinated with itself if the canonical boundary codes of the 3 boundary faces match one of the 8 cases in Table 8.1.

(2) Two patches P_1 and P_2 with $s(P_1) = (f_1, g_1, 1)$ and $s(P_2) = (f_2, g_2, 2)$ can be agglutinated with each other if $f = f_1 + f_2$, $g = g_1 + g_2 + 1$, and the canonical boundary codes of the 3 boundary faces match one of the 8 cases in Table 8.1. Here, in principle, there exist three subcases:

   (a) The boundary face of P_1 is the main boundary and the boundary faces of P_2 are the side boundaries in the agglutination.

   (b) The boundary face of P_2 having the higher total quality is the main boundary and the two remaining boundary faces are the side boundaries in the agglutination.

   (c) The boundary face of P_2 having the lower total quality is the main boundary and the two remaining boundary faces are the side boundaries in the agglutination.

   However, the last subcase cannot occur because the main boundary has always a higher quality than the side boundaries (in the cases 2-4 and 6-8 in Table 8.1, the number of convex edges is higher, and in the cases 1 and 5 the canonical boundary sequence is lexicographically larger). In all subcases, P_1 and P_2 are clearly non–isomorphic because P_1 contains less boundary faces than P_2.

(3) Three patches P_1, P_2, P_3 with $s(P_1) = (f_1, g_1, 1)$, $s(P_2) = (f_2, g_2, 1)$, $s(P_3) = (f_3, g_3, 1)$ can be agglutinated with each other if $f = f_1 + f_2 + f_3$, $g = g_1 + g_2 + g_3$, and the canonical boundary codes of the 3 boundary faces match one of the 8 cases in Table 8.1.

If more than one patch is agglutinated, then f and g must each be partitioned into 2 or 3 non–negative summands in every possible way. This is done straightforward by count loops. It is searched for appropriate patches in the patch tree. Since the side boundaries have the shortest and most restrictive canonical boundary codes, it is most effective to search for those patches first that contain the side boundaries. Then for the canonical boundary code of the main boundary, there are very few alternatives (which are obtained by modifying z).

The required canonical boundary codes imply that a boundary face being a main boundary in one agglutination cannot be a side boundary in a different agglutination and vice versa (provided that the same patches contribute to both agglutinations). Hence as soon as appropriate patches are found, it is clear which boundary face is the main boundary. It holds $c_1 = c_2$ if and only if $l_1(p) = l_3(p)$. Let v_i be the marked vertex contained in F_i, $i \in \{1, 2, 3\}$, and let p be the candidate.

- In the first of the above cases (one patch), every patch P having the desired properties must be considered. If $l_1(p) = l_3(p)$, then $c_1 = c_2$ and it is not unique by c_1 and c_2 which side boundary is the start side boundary. So as well as F_1, F_2 could be the start side boundary, too, and we have two possibilities to agglutinate the 3 boundary faces of P with each other. v_1 and v_2 cannot be mark–isomorphic because v_1 is clearly not mark–isomorphic to v_3, hence the corresponding patch automorphism would map v_1 on v_2 and vice versa, but then the only remaining marked vertex v_3 would have to be fixed under the patch automorphism which is impossible due to Section 7.3. Hence if $c_1 = c_2$, then both possibilities to agglutinate the three boundary faces of P must *always* be performed.

  Due to the canonical boundary codes c_1 and c_2, it is clear that the marked vertices contained in the side boundaries are unique. The marked vertex contained in the main boundary is unique, too, except if $c_1 = c_2$ and $l_2(p)$ is odd. In this case, the canonical boundary sequence of F_3 is periodical with two periods so that F_3 contains two markable vertices. Hence in this case, for every sorted pair (F_1, F_2) of boundary faces, there are theoretically two possibilities to agglutinate F_3 with F_1 and F_2, and the one according to the marking rules presented in Section 7.4 must be performed. In all other cases, there is only one way to agglutinate the three boundaries anyway.
- In the second case (two patches), every pair $\{P_1, P_2\}$ of patches having the desired properties must be considered. If $l_1(p) = l_3(p)$, then it is again not unique which side boundary is the start side boundary. In subcase (2b), v_1 and v_2 cannot be isomorphic because P_1 and P_2 are not isomorphic. Hence as above, both possibilities to choose the start side boundary must always be considered. In subcase (2a), v_1 and v_2 might be mark–isomorphic in P_2, and if this is the case, then only one agglutination is performed because otherwise, the two agglutinations would be equivalent. In both subcases, as above the agglutinations are unique with respect to the marking rules.
- In the third case (three patches), every triple $\{P_1, P_2, P_3\}$ of patches having the desired properties must be considered. Let P_1 and P_2 be the patches containing the side boundaries F_1 and F_2, respectively, and let P_3 be the patch containing the main boundary for the agglutination. If $l_1(p) = l_3(p)$, then as for the second case, in principle two agglutinations must be performed except if P_1 and P_2 are mark–isomorphic. However, as in the Jordan curve case, even if P_1 and P_2 are not mark–isomorphic, it often suffices to perform one agglutination only, and as the candidate we take the path having the higher quality among p and $\overline{p}$. For the following explanation, let $l_1(p) = l_3(p)$.
  - Let $l_2(p)$ be odd. Then — as p — $\overline{p}$ is directed to the right, too. P_3 contains two markable vertices. v_3 is one of them. Let v' be the other one. Let P_3' be the patch that differs from P_3 only by the marked vertex incident with F_3 (i.e. v' is marked instead of v_3). It does not matter whether P_3 and P_3' are mark–isomorphic or not. Let M be the cubic map

that is obtained by agglutinating P_1, P_2, and P_3 using F_1 as the start side boundary, and let M' be the cubic map that is obtained by agglutinating P_1, P_2, and P_3' using F_2 as the start side boundary. Then M and M' are isomorphic (see Figure 8.11; here two reference vertices u and w are given instead of v_3 and v' because the exact positions of v_3 and v' are not known). The only difference is that in M and M', the candidates are inverse to each other. Analogously, agglutinating P_1, P_2, and P_3 using F_2 as the start side boundary and agglutinating P_1, P_2, and P_3' using F_1 as the start side boundary yields isomorphic cubic maps where the candidates are inverse to each other. Hence it suffices to use only F_1 as the start side boundary in order not to miss any map. And in this case, out of p and $\overline{p}$, the path having the higher quality must be chosen as the candidate.

– Let $l_2(p)$ be even. Then $\overline{p}$ is directed to the left. Hence unlike for the case where $l_2(p)$ is odd, one will in general not obtain isomorphic cubic maps by switching P_1 and P_2. Let $\overline{P_1}$, $\overline{P_2}$, and $\overline{P_3}$ be the mirror images of P_1, P_2, and P_3, respectively. Let F_1' and F_2' be the boundary faces of $\overline{P_1}$ and $\overline{P_2}$, respectively. Let M be the cubic map which is obtained by agglutinating P_1, P_2, and P_3 using F_1 as the start side boundary, and let M' be the cubic map which is obtained by agglutinating $\overline{P_1}$, $\overline{P_2}$, and $\overline{P_3}$ using F_2' as the start side boundary. Then it holds $\overline{M'} = M$, and the candidate in M is inverse to the candidate in $\overline{M'}$. (see Figure 8.12). So if we can secure that **exactly** one of the two agglutinations is performed, then as above, we can omit one agglutination and take the path having the higher quality out of p and $\overline{p}$ as the candidate. We can easily secure that exactly one of the two agglutinations is performed whenever P_1 and P_2 are not stored in one common list contained in one leaf in the patch tree. In this case, P_1 and P_2 differ by their sizes (because since $c_1 = c_2$, both P_1 and P_2 have the same boundary code; since both P_1 and P_2 contain one boundary face each, one can even state that $g_{P_1} \neq g_{P_2}$ or $f_{P_1} \neq f_{P_2}$) and we may require $s(P_1) < s(P_2)$. Then $s(\overline{P_1}) > s(\overline{P_2})$ so that the second agglutination will not be performed. If P_1 and P_2 are stored in the same list, then P_1, P_2, $\overline{P_1}$, and $\overline{P_2}$ are all stored in one common list: $\overline{P_1}$ has the same size as P_1 and also the same boundary code (this is trivial because the canonical boundary sequences of F_1 and F_1' each have at most 2 entries, the entries are pairwise identical, and the higher entry is always followed by the lower one). The same holds for $\overline{P_2}$ and P_2. Since the order of the patches in one list (i.e. contained in one leaf in the patch tree) is not defined, one cannot (by a simple rule) partition the pairs of distinct patches of this list into two sets such that (P_1, P_2) is in one set exactly if $(\overline{P_2}, \overline{P_1})$ is in the other. Hence this is the only case where we cannot summarize two agglutinations.

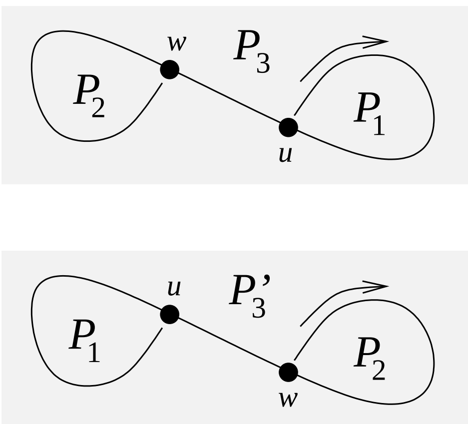

Figure 8.11

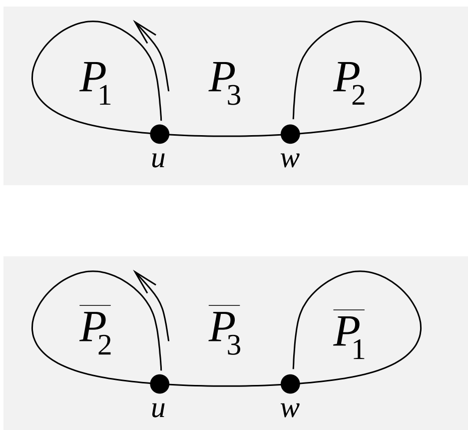

Figure 8.12

8.6.3 Sandwich paths

Let M be a cubic map that is cut along a sandwich path p which is continued to the right where $l_2(p)$ is *odd*. Let F_1 be the resulting start side boundary, let F_2 be the resulting end side boundary, and let F_3 be the resulting main boundary in the ancestor(s). From Sections 6.4.3 and 8.2, the following data can easily be derived:

- If $l_1(p)$ is odd, then the canonical boundary code c_1 of F_1 is $c_1 = (1, \langle \frac{l_1(p)+l_2(p)-1}{2} \rangle)$. If $l_1(p)$ is even, then $c_1 = (2, \langle \frac{l_1(p)+l_2(p)-2}{2}, 0 \rangle)$.
- If $l_3(p)$ is odd, then the canonical boundary code c_2 of F_2 is $c_2 = (1, \langle \frac{l_3(p)+l_2(p)-1}{2} \rangle)$. If $l_3(p)$ is even, then $c_2 = (2, \langle \frac{l_3(p)+l_2(p)-2}{2}, 0 \rangle)$.

In order to note down the canonical boundary code c_3 of F_3, we must distinguish 4 cases depending on $l_1(p)$ and $l_3(p)$. Table 8.2 shows the resulting boundary codes. x, y, and z can be derived from $l_1(p)$, $l_2(p)$, and $l_3(p)$ (where z has an upper limit determined by the fact that all entries in the boundary sequence of c_3 must be non–negative): It holds $x = \lfloor \frac{l_1(p)+l_2(p)-1}{2} \rfloor$, $y = \lfloor \frac{l_2(p)+l_3(p)-1}{2} \rfloor$, and $z = \frac{l_2(p)-1}{2}$ (so that $x, y, z \geq 0$ and the upper limit for z implies an upper limit for $l_2(p)$). It might be necessary to shift the boundary sequence of c_3 depending on the values of x and y in order to obtain a lexicographically maximal (and thus canonical) sequence. However, if p is canonical, then it always holds $l_1(p) \leq l_3(p)$. Thus $x \leq y$ and in case 1 even $x + 1 \leq y$ so that the boundary sequences denoted in Table 8.1 are always lexicographically maximal among all shifted sequences except maybe in case 1 (if and only if $x + 1 = y$; in this case, the canonical boundary code is $c_3 = (3, \langle x - z, y - 1 - z, 0 \rangle))$.

Case	$l_1(p)$	$l_3(p)$	$l_2(p)$	c_1	c_2	c_3
1	odd	even	odd	$(2, \langle x, 0 \rangle)$	$(1, \langle y \rangle)$	$(3, \langle y - 1 - z, 0, x - z \rangle)$
2	odd	odd	odd	$(2, \langle x, 0 \rangle)$	$(2, \langle y, 0 \rangle)$	$(2, \langle y - z, x - z \rangle)$
3	even	even	odd	$(1, \langle x \rangle)$	$(1, \langle y \rangle)$	$(4, \langle y - 1 - z, 0, x - 1 - z, 0 \rangle)$
4	even	odd	odd	$(1, \langle x \rangle)$	$(2, \langle y, 0 \rangle)$	$(3, \langle y - z, 0, x - 1 - z \rangle)$

Table 8.2

So in order to obtain maps of $\mathcal{M}(g, n, D, b_l, b_u)$, we need to agglutinate patches containing altogether 3 boundary faces. The possibilities how to obtain these patches and the details how to decrease the number of agglutinations that must be performed in order not to miss any cubic map are similar to the ones given in Section 8.6.2. However, there are the following differences:

- A boundary face with the canonical boundary code $(2, \langle x, 0 \rangle)$ can be a side boundary and the main boundary (in case 2 of Table 8.2) in an agglutination. Both possibilities must be considered. Once it is fixed which boundary face is the main boundary (this is not necessary in the dumbbell case because there, it is always unique which boundary face is the main boundary in an agglutination), the argumentation is equal to the one for the dumbbell case. Examples: If $(l_1(p), l_2(p), l_3(p)) = (1, 1, 5)$, then $c_1 = (2, \langle 0, 0 \rangle)$ and $c_2 = c_3 = (2, \langle 2, 0 \rangle)$. If $(l_1(p), l_2(p), l_3(p)) = (1, 5, 5)$, then $c_2 = (2, \langle 4, 0 \rangle)$ and $c_1 = c_3 = (2, \langle 2, 0 \rangle)$.
- Subcase (2c) is possible in the sandwich case (for case 2 of Table 8.2).

Let M be a cubic map that is cut along a sandwich path p which is continued to the right where $l_2(p)$ is *even*. Let F be the resulting boundary face in the ancestor. In order to note down the canonical boundary code c of F, we must distinguish 4 cases depending on $l_1(p)$ and $l_3(p)$. Table 8.3 shows the resulting canonical boundary codes. x, y, and z can be derived from $l_1(p)$, $l_2(p)$, and $l_3(p)$: It holds $x = \lfloor\frac{l_1(p)-1}{2}\rfloor$, $y = \lfloor\frac{l_2(p)-1}{2}\rfloor$, and $z = \lfloor\frac{l_3(p)-1}{2}\rfloor$ (so that $x, y, z \geq 0$).

Case	$l_1(p)$	$l_3(p)$	$l_2(p)$	c
5	even	even	even	$(6, \langle x+y+z+3, z, 0, y, 0, x\rangle)$
6	odd	even	even	$(6, \langle x+y+z+2, x, y, 0, z, 0\rangle)$
7	even	odd	even	$(6, \langle x+y+z+2, z, y, 0, x, 0\rangle)$
8	odd	odd	even	$(6, \langle x+y+z+1, 0, x, y, z, 0\rangle)$

Table 8.3

So in order to obtain a map of $\mathcal{M}(g, n, D, b_l, b_u)$, we need to agglutinate one patch P of size $s(P) = (f, g-1, 1)$ with itself where the canonical boundary code of the boundary face F of P matches one of the 4 cases in Table 8.3. The cases 6 and 7 are disjoint because if p is canonical, then it always holds $l_1(p) \leq l_3(p)$. So in case 6 and 7, it even holds $l_1(p) + 1 \leq l_3(p)$ (because one value is even and the other one is odd), and thus it holds $x \leq z$ in case 6 while it holds $z > x$ in case 7 so that the corresponding canonical boundary codes cannot be identical.

8.6.4 Remarks

Every cubic map M which is constructed in any of the above cases must satisfy the following properties:

(1) Every face of M must have a degree $d \in D$.
(2) For every $d \in D$, it must hold $b_l(d) \leq p_d(M) \leq b_u(d)$.
(3) The candidate must be a canonical Petrie path in M.

The last property can only be verified after the agglutination, but the first two properties can be verified already before the agglutination: The first property always holds because it holds for all generated patches. The second property is verified by adding the corresponding numbers for the patches (where only inner faces are considered).

Moreover, there are some tests on M that have already been described for patches in Section 8.5:

- If loops and multiple edges are not allowed, then it must eventually be tested whether M contains a multiple edge. M must always be tested if one of its canonical ancestors is agglutinated with itself along the Petrie path p (this is the case when $|A(M,p)| < 2$ if p is a Jordan curve path, when $|A(M,p)| < 3$ if p is a sandwich path, or when p is a dumbbell path) because as in Section 8.5, p might contain an edge d whose incident vertices are both incident with a common edge e that has been an inner edge before the agglutination (if p is a dumbbell path and $|A(M,p)| = 3$, then d is an element of the second part of p). On the other hand, if two patches are agglutinated along a Jordan curve path or if three patches are agglutinated along a sandwich path, then M does not need to be tested for multiple edges.

- If only semi–polyhedral or polyhedral maps are wanted, then it must be tested if M is semi–polyhedral or polyhedral, respectively.
- If only maps that represent planar bipartite Delaney graphs are wanted, then the test described in Section 8.5 must be performed for M.

8.7 Decreasing the number of stored patches

As mentioned before, it is not possible to generate only canonical patches. Thus, many patches might be stored that will not be used during the construction of cubic maps afterwards. These patches might require so much computer memory that the construction of $\mathcal{M}(g, n, D, b_l, b_u)$ might become impossible.

In this section, we will provide so–called *patch preventing criteria* that detect and filter out non–canonical patches *before* they are stored. Moreover, we will describe how it can be prevented to store even *canonical* patches. We will say that a patch P *satisfies* a criterion if by applying the criterion, it cannot be stated that P is not canonical (so that P will not be filtered out).

From now on, a face of degree greater than 6 will be called a **large** face. A face of degree smaller than 6 will be called a **small** face. For the following considerations, as usual $f = \frac{n}{2} + 2 - 2g$ denotes the number of faces in every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$.

Let $p_d(P)$ be the number of *inner* d–gons in a patch P. Thus the definition of $p_d(P)$ corresponds to the defintion of $p_d(M)$ for a cubic map M due to the fact that in the construction, the faces of a cubic map correspond to the inner faces of a patch. Let $n' = |D|$ and $D = \{d_1, \ldots, d_{n'}\}$ where the elements are sorted such that $d_i < d_j$ if and only if $i < j$. Let $f_i := p_{d_i}(P)$, $i \in \{1, \ldots, n'\}$, be the number of *inner* d_i–gons in a patch P. Then the sequence $s_D(P) = \langle f_1, \ldots, f_{n'} \rangle$ is called the **face vector** of P (with respect to D).

Apart from the criteria described in the following sections, more patch preventing criteria for $g = 0$ can be found in [46].

8.7.1 The Petrie path criteria

Theorem 8.2 *If P is a canonical patch, then it holds:*

- *P contains at most 2 boundary faces not containing convex edges.*
- *If P contains 2 boundary faces not containing convex edges, then $s(P) = (f, g-1, 2)$ and both boundary faces have the same canonical boundary code.*

Proof: When a patch is cut along a zig–zag path, then the resulting new boundary faces always contain convex edges. The same holds when a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is cut along a sandwich or a dumbbell path. Hence new boundary faces without convex edges occur only when a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is cut along a Jordan curve path. In this case, we obtain altogether two boundary faces without convex edges in the resulting patch(es). Hence P contains at most 2 boundary faces without convex edges. If P contains 2 boundary faces without convex edges, then P must be the ancestor of a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ because in every further cut, at least one of the 2 boundary faces is cut. Hence $s(P) = (f, g-1, 2)$

as indicated in Section 8.6, and both boundary faces of P have the same degree and thus the same canonical boundary code. ■

Hence if a patch P contains more than 2 boundary faces without convex edges or 2 boundary faces without convex edges where either $s(P) \neq (f, g-1, 2)$ or the degrees of these boundary faces are not equal, then P is not canonical and does not need to be stored.

For the next criterion, we need to investigate cycles in cubic maps. Let $c_1 = \langle h_0, h'_0, \ldots, h_r, h'_r \rangle$ and $c_2 = \langle k_0, k'_0, \ldots, k_t, k'_t \rangle$ with $\{h_0, h'_0, \ldots, h_r, h'_r\} \neq \{k_0, k'_0, \ldots, k_t, k'_t\}$ be two cycles of lengths $r+1$ and $t+1$, respectively, in a map $M = (G, \Pi) \in \mathcal{M}(g, n, D, b_l, b_u)$. Then an **intersecting path** of c_1 and c_2 is a path p or $\overline{p}$ where $p = \langle h_i, h'_i, \ldots, h_{i+l-1}, h'_{i+l-1} \rangle$ is a path of length $l \geq 1$ (where all indices are expressed modulo $r+1$) such that all elements of p are incident also with c_2, but neither the edge $\{h_{i-1}, h'_{i-1}\}$ nor the edge $\{h_{i+l}, h'_{i+l}\}$ is incident with c_2, i.e. it holds $\{h_i, h'_i, \ldots, h_{i+l-1}, h'_{i+l-1}\} \subset \{k_0, k'_0, \ldots, k_t, k'_t\}$ and $\{h'_{i-1}, h_{i+l}\} \cap \{k_0, k'_0, \ldots, k_t, k'_t\} = \emptyset$. The semi–edge h'_{i-1} is incident with h_i and another semi–edge k. The semi–edge h_{i+l} is incident with h'_{i+l-1} and another semi–edge k'. Both k and k' are incident with c_2 but not with c_1. The intersecting path is called a **crossing** if and only if either both $\Pi(k) = h_i$ and $\overline{\Pi}(k') = h_{i+l}$ or both $\overline{\Pi}(k) = h_i$ and $\Pi(k') = h_{i+l}$.

Lemma 8.3 *Let $c_1 = \langle h_0, h'_0, \ldots, h_r, h'_r \rangle$ and $c_2 = \langle k_0, k'_0, \ldots, k_t, k'_t \rangle$ with $\{h_0, h'_0, \ldots, h_r, h'_r\} \neq \{k_0, k'_0, \ldots, k_t, k'_t\}$ be two cycles in a cubic planar map $M = (G, \Pi)$. Then the number of semi–edges h_i with $\Pi(h_i) = k$ for some $i \in \{0, \ldots, r\}$ and some $k \in \{k_0, k'_0, \ldots, k_t, k'_t\} - \{h_0, h'_0, \ldots, h_r, h'_r\}$ is even. Moreover, the number of semi–edges h_i with $\overline{\Pi}(h_i) = k$ for some $i \in \{0, \ldots, r\}$ and some $k \in \{k_0, k'_0, \ldots, k_t, k'_t\} - \{h_0, h'_0, \ldots, h_r, h'_r\}$ is even.*

The semi–edges having one of the above properties are exactly the start and end semi–edges of the intersecting paths of c_1 and c_2: For k and h_i denoted as above, it holds that since k is incident with c_2 and c_2 is a cycle, it must hold $h'_{i-1} \in \{k_0, k'_0, \ldots, k_t, k'_t\}$ or $h_i \in \{k_0, k'_0, \ldots, k_t, k'_t\}$. Thus the incident vertices of the considered semi–edges are the vertices where c_2 "leaves" or "enters" c_1. The idea of the proof is that in a planar map, c_1 must be entered from the same side it was left (while tracing c_2). Figure 8.13 illustrates the situation for the following proof.

Proof: Let $p = \langle h_{i-l}, h'_{i-l}, \ldots, h_{i-1}, h'_{i-1} \rangle$ (all indices expressed modulo $r+1$) be an intersecting path of c_1 and c_2 (where $i \in \{0, \ldots, r\}$ and $l \in \{1, \ldots, r\}$) with $\Pi(h_i) = k$ for some $k \in \{k_0, k'_0, \ldots, k_t, k'_t\} - \{h_0, h'_0, \ldots, h_r, h'_r\}$. Let w.l.o.g. $k \in \{k_0, k_1, \ldots, k_t\}$ (otherwise, consider $\overline{c_2}$ instead of c_2). W.l.o.g., let even $k = k_0$ (thus $k'_t = h'_{i-1}$). Now starting with k, we trace c_2 until we approach the start semi–edge of the next intersecting path of c_1 and c_2: Let m be as small as possible such that $\Pi(h_j) = k'_m$ or $\overline{\Pi}(h_j) = k'_m$ for some $j \in \{0, \ldots, r\}$. Assume that $\overline{\Pi}(h_j) = k'_m$. Then due to the Jordan curve theorem in Section 1.6, the path $\langle k_0, k'_0, \ldots, k_m, k'_m \rangle$ is incident with c_1 which is a contradiction to the condition that m is smallest possible. Hence $\Pi(h_j) = k'_m$. Thus by tracing c_2, to every *end* semi–edge h_i of an intersecting path with $\Pi(h_i) = k$ for some $i \in \{0, \ldots, r\}$ and some $k \in \{k_0, k'_0, \ldots, k_t, k'_t\} - \{h_0, h'_0, \ldots, h_r, h'_r\}$, a *start* semi–edge h_j of an intersecting path with $\Pi(h_j) = k$ for some $j \in \{0, \ldots, r\}$ and some $k \in \{k_0, k'_0, \ldots, k_t, k'_t\} - \{h_0, h'_0, \ldots, h_r, h'_r\}$ can be assigned. Hence the number of semi–edges h_i with $\Pi(h_i) = k$ for some $i \in \{0, \ldots, r\}$ and some $k \in \{k_0, k'_0, \ldots, k_t, k'_t\} - \{h_0, h'_0, \ldots, h_r, h'_r\}$ is even. In the same way, it can be shown that the number of semi–edges h_i with $\overline{\Pi}(h_i) = k$ for some $i \in \{0, \ldots, r\}$ and some $k \in \{k_0, k'_0, \ldots, k_t, k'_t\} - \{h_0, h'_0, \ldots, h_r, h'_r\}$ is even. ■

Corollary 8.4 *If c_1 and c_2 are Jordan curve paths, then the number of edges that are incident with both c_1 and c_2 is even.*

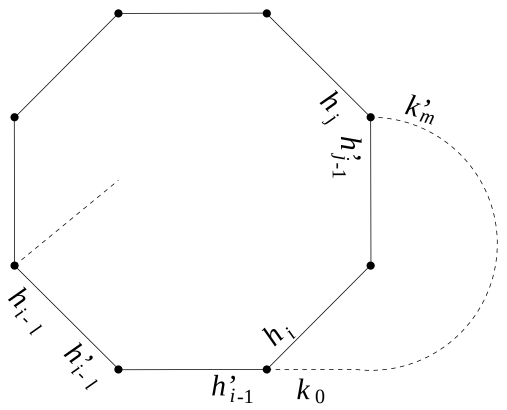

Figure 8.13: the solid edges are elements of c_1

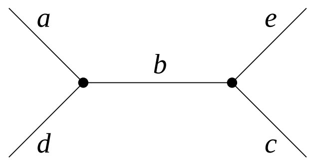

Figure 8.14

Proof: Every intersecting path of two non–equivalent Jordan curve paths has length 1 and is a crossing: Look at Figure 8.14. If a, b, and c are elements of c_1, and if b is an element of c_2, then both a and c are no elements of c_2 because otherwise c_1 and c_2 would have the same set of elements (a implies the side and the direction of b when b is rooted in order to initiate a unique Petrie path; the same holds for c) and thus be equivalent. Hence d and e are elements of c_2. Hence every intersection path is a crossing of length 1. Since every intersection path is a crossing, the number a of semi–edges h_i with $\Pi(h_i) = k$ for some $i \in \{0, \ldots, r\}$ and $k \in \{k_0, k'_0, \ldots, k_t, k'_t\}$ is equal to the number b of semi–edges h_i with $\overline{\Pi}(h_i) = k$ for some $i \in \{0, \ldots, r\}$ and $k \in \{k_0, k'_0, \ldots, k_t, k'_t\}$, and due to Lemma 8.3, both a and b are even. Since every intersecting path has length 1, the number of edges incident with both c_1 and c_2 is $\frac{a+b}{2} = \frac{2a}{2} = a$ where a is even. ■

Theorem 8.5 *If $g = 0$ and n is no multiple of 4, then for every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, it holds that M contains a sandwich path or a dumbbell path.*

Proof: Suppose we have a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ where the edges are coloured black. Let $p_0, \ldots, p_m$ be the Jordan curve paths in M. We will now inductively consider $p_0, \ldots, p_m$ one after the other. For every considered Jordan curve path p_i, we will colour the edges of p_i in red. After each step i, we will count the number r_i of red edges. We will show by induction that r_i is always even. Of course, it suffices to take one representative of each class of equivalent Jordan curve paths in order to colour every edge in M that is incident with a Jordan curve path. So let $\{q_0, \ldots, q_{m'}\} \subseteq \{p_0, \ldots, p_m\}$ such that out of each equivalence class of $\{p_0, \ldots, p_m\}$, exactly one representative is contained in $\{q_0, \ldots, q_{m'}\}$.

q_0 is the first considered Jordan curve path. The length l of q_0 is even, so after colouring q_0 it holds that $r_0 (= l)$ is even. Let now $q_0, \ldots, q_i$ $(0 \leq i \leq m')$ already be coloured red, and let r_i be even. If $i = m'$, then we are done. Otherwise we colour q_{i+1}. Let l be the length of q_{i+1}. l is even. We colour $l - x$ new edges where x is the number of elements of q_{i+1} that are already coloured red. Due to Corollary 8.4, for every $j \in \{0, \ldots, i\}$ it holds that the number of edges that are incident with both q_{i+1} and q_j is even. Moreover, if b is an edge which is incident with both q_{i+1} and q_j, then b cannot be incident with any other Jordan curve path $q' \in \{q_0, \ldots, q_i\} - \{q_j\}$: Recall Figure 8.14. Every Jordan curve path p incident with b contains either both a and c or both d and e, and hence p is equivalent either to q_{i+1} or to q_j. Hence the sets $e_j = \{e | e \text{ is incident with both } q_{i+1} \text{ and } q_j\}$, $j \in \{0, \ldots, i\}$, are pairwise disjoint and $|e_j|$ is even for every $j \in \{0, \ldots, i\}$. Hence $x = \sum_{j=0}^{i} |e_j|$ is even. Thus the number $l - x$ of new red edges is even, too, and thus $r_{i+1} = r_i + (l - x)$ is even, too.

It holds $e_{_M} = \frac{3}{2}n$. Since n is no multiple of 4, $e_{_M}$ is odd and thus $e_{_M} \neq r_{m'}$. Hence after the colouring, there is still a black edge. This edge is not an element of a Jordan curve path. Hence M contains a sandwich or dumbbell path. ∎

Theorem 8.5 can be applied in the following way: If $g = 0$ and n is no multiple of 4, then in the canonicity test, we assign a lower quality to Jordan curve paths than to sandwich paths and dumbbell paths. Since every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ contains a sandwich path or a dumbbell path, a Jordan curve path will never be the canonical Petrie path in M. Thus we do not need to store patches containing boundary faces without convex edges because as explained in the proof of Theorem 8.2, boundary faces without convex edges occur only if a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is cut along a Jordan curve path. Moreover, we do not need to agglutinate patches along Jordan curve paths (which is impossible anyway if the appropriate patches are not stored).

Theorem 8.6 *If M is an exactly 1–connected cubic map, then M contains a dumbbell path.*

Proof: Let v be a vertex in M whose deletion transforms M into a disconnected bunch of maps B. Then v is incident with an edge e whose removal transforms M into a disconnected bunch of maps, too: Let $M_1, \ldots, M_m$ $(m \geq 2)$ be the connected components of B. Since $\mathrm{val}(v) = 3$ (in M), it holds $2 \leq m \leq 3$ and there exists $i \in \{1, \ldots, m\}$ such that in M, there is only one semi–edge h whose source is v and whose target u is a vertex contained in M_i. Let h_2 and h_3 be the other semi–edges incident with v in M. Thus the targets w and x of h_2 and h_3 are contained in other components M_j, $j \in \{1, \ldots, m\} - \{i\}$. Since the deletion of v in M transforms M into a disconnected bunch of maps, every path connecting w and u contains h (and h_2 or h_3). Thus the removal of the edge e containing h as an element transforms M into a disconnected bunch of maps. In particular, e cannot be an element of a cycle (because otherwise, v and u and thus also w and u would still be connected even if e was removed in M). But each element of a Jordan curve path or a sandwich path is also an element of a cycle. Hence e is an element of a dumbbell path. ∎

Theorem 8.6 can be applied in the following way: If only exactly 1–connected cubic maps shall be accepted (e.g. if $1 \in D$ and $b_l(1) > 0$, see Section 6.1), then in the canonicity test, we assign a higher quality to dumbbell paths than to Jordan curve paths and to sandwich paths. Since every desired cubic map contains a dumbbell path, the canonical Petrie path in every desired map will be a dumbbell path. Thus we do not need to store patches containing boundary faces without convex edges because as explained in the proof of Theorem 8.2, boundary faces without convex edges occur only if a cubic map is cut along a Jordan curve path. Moreover, we do not need to perform Jordan curve agglutinations (which is impossible anyway when the appropriate patches are not stored) and sandwich agglutinations.

Remark: Since e (in the proof of Theorem 8.6) is not incident with a cycle, e is a *face–bridge*: Every edge which is not a face–bridge is incident with cycles, namely cycles that correspond to the incident faces or — if both incident faces contain face–bridges — cycles that correspond to parts of the incident faces. Thus an exactly 1–connected cubic map M always contains a face–bridge. In particular, if M is simple, then due to Lemma 2.2, M contains a face of degree at least 8. Thus if $D \cap \{3, \ldots, 7\} = \emptyset$, then $\mathcal{M}(g, n, D, b_l, b_u)$ contains no exactly 1–connected maps (so that any construction can be omitted if only exactly 1–connected maps are wanted).

Theorem 8.7 *Let $d_{\min} := \min D$ $(= \min_{d \in D} d)$, let $M \in \mathcal{M}(g, n, D, b_l, b_u)$, and let $P \in A(M, p')$ for some Petrie path p' in M. If $d_{\min} \geq 6$, $b_{_P} = 1$, and p' is not a sandwich path with $l_2(p')$ even, then $g_{_P} \geq 1$.*

Proof: Let P be a possibly canonical patch with $g_P = 0$ and $b_P = 1$. Then it holds that if P is not a tile, then $|A(P,p)| = 2$ (for every valid zig–zag path p in P) and both $g_Q = 0$ and $b_Q = 1$ for every $Q \in A(P,p)$: It holds $0 \le g_Q \le g_P = 0$, and hence $g_Q = 0$ is clear. p cannot be a linkage path because $b_P = 1$. And since $g_P = 0$, it cannot hold $|A(P,p)| = 1$ if p is an intersection path or an enclosure path (see Section 6.6.1 and 6.6.3). So $|A(P,p)| = 2$. Due to Section 6.6.1 and 6.6.3, the boundary face in P is replaced by two new boundary faces, so each patch in $A(P,p)$ contains exactly one boundary face. Thus $b_Q = 1$. In the same way, it can be shown inductively that $g_Q = 0$ and $b_Q = 1$ for every *iterated* ancestor Q of P and — if Q is not a tile — $|A(Q,q)| = 2$ for every valid zig–zag path q in Q.

Now we show by induction that if $d_{\min} \ge 6$, then every possibly canonical patch P with $g_P = 0$ and $b_P = 1$ contains at least 6 convex edges:

- Let P be a tile. Then the degree of the boundary face and thus the number of convex edges in P is at least 6.
- Let P not be a tile. Then due to the above considerations, P has two canonical ancestors Q_1 and Q_2 containing k_{Q_1} and k_{Q_2} convex edges, respectively, where it holds $k_{Q_1} \ge 6$ and $k_{Q_2} \ge 6$ due to the induction assumption. Lemma 6.17 implies $k_P = k_{Q_1} + k_{Q_2} - 6 \ge 6$.

But if $P \in A(M,p')$ with $b_P = 1$ and p' is not a sandwich path with $l_2(p')$ even, then due to Sections 8.6.1–8.6.3, the boundary face in P contains at most 4 convex edges. Thus it cannot hold $g_P = 0$. ■

Corollary 8.8 *If $d_{\min} \ge 6$, then the genus of every canonical patch is smaller than g.*

Proof: Let $M \in \mathcal{M}(g,n,D,b_l,b_u)$ and let p be the canonical Petrie path in M. If p is a Jordan curve path with $|A(M,p)| = 1$ or p is a sandwich path or a dumbbell path with $|A(M,p)| \le 2$, then the genus of every canonical ancestor of M (and hence the genus of every iterated canonical ancestor of M) is always smaller than g even if $d_{\min} < 6$ (see Sections 6.4.1, 6.4.2, and 6.4.3). In the remaining cases, the genera of the (2 or 3) canonical ancestors of M add up to g. Theorem 8.7 implies that the genus of every canonical ancestor of M is at least 1 so that the genus of every canonical ancestor of M can be at most $g-1$ in the Jordan curve case and at most $g-2$ in the sandwich or dumbbell case. ■

Thus if $d_{\min} \ge 6$, then patches of genus g need not be generated. Moreover, if a patch P is to be agglutinated with $x \ge 1$ other patches along a Petrie path, then it does not need to be searched for appropriate patches of genus 0. In particular, P cannot be agglutinated with x other patches if $g_P > g - x$.

8.7.2 The face number criterion

Numbers of faces of certain degrees are considered in the following criterion. Let b_u and b_l be determined as sharply as possible as described in Section 6.1. Then the following criterion becomes most effective. Let P be a possibly canonical patch and let $M \in \mathcal{M}(g,n,D,b_l,b_u)$. If it holds $p_d(M) < p_d(P)$ for some $d \in D$, then clearly P is not an iterated ancestor of M. So let $M \in \mathcal{M}(g,n,D,b_l,b_u)$ with $p_d(M) \ge p_d(P)$ for every $d \in D$. If P is an iterated ancestor of M, then by the agglutinations, exactly $p_d(M) - p_d(P)$ faces of degree d are added for every $d \in D$. Thus $f'_P + \sum_{d\in D}(p_d(M) - p_d(P)) = f$. So in particular, it holds that if $f'_P + \sum_{d\in D}(p_d(M) - p_d(P)) > f$, then P is not an iterated ancestor of M.

We obtain a patch preventing criterion which is independent from a concrete map M in the following way: Let $a_d(P) := \max\{0, b_l(d) - p_d(P)\}$ for every $d \in D$. Since $p_d(M) \geq b_l(d)$ for every $d \in D$ and every $M \in \mathcal{M}(g, n, D, b_l, b_u)$, it holds $a_d(P) \leq \max\{0, p_d(M) - p_d(P)\}$ for every $d \in D$ and every $M \in \mathcal{M}(g, n, D, b_l, b_u)$.

- For every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ with $p_d(M) < p_d(P)$ for some $d \in D$, it holds that P is not an iterated ancestor of M.
- For every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ with $p_d(M) \geq p_d(P)$ for every $d \in D$, it holds that $a_d(P) \leq p_d(M) - p_d(P)$ for every $d \in D$. If $f'_P + \sum_{d \in D} a_d(P) > f$, then $f'_P + \sum_{d \in D}(p_d(M) - p_d(P)) \geq f'_P + \sum_{d \in D} a_d(P) > f$ so that P is not an iterated ancestor of M.

Thus we obtain the following criterion:

Theorem 8.9 *If $f'_P + \sum_{d \in D} a_d(P) > f$, then P is not a canonical patch.*

8.7.3 The Euler criterion

Let $fd(X)$ be defined as follows: $fd(X) = \sum_{d \in D}(6 - d) \cdot p_d(X)$. X may be a cubic map or a patch. For a cubic map M of genus g, it always holds $fd(M) = 12 \cdot (1 - g)$ due to formula 6.1 on page 68. Let P be a possibly canonical patch.[1] Let k_P be the number of convex edges in P.

Lemma 8.10 *For a possibly canonical patch P, it holds $fd(P) = 12 \cdot (1 - g_P) - 6b_P - k_P$.*

Proof: We prove this lemma by induction over the size of P. Let P have minimal size, i.e. P is a tile. Let d' be the degree of P. Then $p_{d'}(P) = 1$, $p_d(P) = 0$ for all $d \neq d'$, $g_P = 0$, $k_P = d'$, and $b_P = 1$. Hence $fd(P) = \sum_{d \in D} p_d(P) \cdot (6 - d) = 1 \cdot (6 - d') = 6 - k_P = 12 \cdot (1 - 0) - 6 \cdot 1 - k_P = 12 \cdot (1 - g_P) - 6b_P - k_P$. Hence the lemma is true for all tiles. Let now P be a patch which is cut along a zig–zag path p.

- Let $A(P, p) = \{Q\}$. Since $s(Q) < s(P)$ (Lemma 6.18), it holds $fd(Q) = 12 \cdot (1 - g_Q) - 6b_Q - k_Q$ due to the induction assumption. Moreover, it holds $p_d(P) = p_d(Q)$ for every $d \in D$ (because the inner faces remain unmodified after the cut) and thus $fd(P) = fd(Q)$, and it holds $k_Q = k_P + 6$ (Lemma 6.17).
  - If p is a linkage path, then $g_Q = g_P$ and $b_Q = b_P - 1$ and hence

$$\begin{aligned} fd(P) &= fd(Q) \\ &= 12 \cdot (1 - g_Q) - 6b_Q - k_Q \\ &= 12 \cdot (1 - g_P) - 6 \cdot (b_P - 1) - (k_P + 6) \\ &= 12 \cdot (1 - g_P) - 6b_P - k_P \end{aligned}$$

  - If p is an intersection or enclosure path, then $g_Q = g_P - 1$ and $b_Q = b_P + 1$ and hence

$$\begin{aligned} fd(P) &= fd(Q) \\ &= 12 \cdot (1 - g_Q) - 6b_Q - k_Q \\ &= 12 \cdot (1 - (g_P - 1)) - 6 \cdot (b_P + 1) - (k_P + 6) \\ &= 12 \cdot (1 - g_P) - 6b_P - k_P \end{aligned}$$

[1] Here and in the following sections, the patches must be possibly canonical in order to secure that the degree of each inner face is in D. By considering $p_d(P)$ for every $d \geq 1$, one can extend the validity of most formulæ to arbitrary patches.

- Let $|A(P,p)| = 2$ with $A(P,p) = \{Q_1, Q_2\}$. Due to the induction assumption, it holds $fd(Q_j) = 12\cdot(1-g_{Q_j})-6b_{Q_j}-k_{Q_j}$ for $j \in \{1,2\}$. Moreover, it holds $p_d(P) = p_d(Q_1)+p_d(Q_2)$ for every $d \in D$, it holds $g_P = g_{Q_1}+g_{Q_2}$, $b_P = b_{Q_1}+b_{Q_2}-1$, and $k_P = k_{Q_1}+k_{Q_2}-6$. Hence

$$\begin{aligned}
fd(P) &= \sum_{d\in D} p_d(P)\cdot(6-d) \\
&= \sum_{d\in D} \big(p_d(Q_1)+p_d(Q_2)\big)\cdot(6-d) \\
&= \sum_{d\in D} p_d(Q_1)\cdot(6-d) + \sum_{d\in D} p_d(Q_2)\cdot(6-d) \\
&= fd(Q_1)+fd(Q_2) \\
&= 12\cdot(1-g_{Q_1})-6b_{Q_1}-k_{Q_1}+12\cdot(1-g_{Q_2})-6b_{Q_2}-k_{Q_2} \\
&= 12\cdot(1-g_{Q_1}-g_{Q_2})+12-6\cdot(b_{Q_1}+b_{Q_2})-(k_{Q_1}+k_{Q_2}) \\
&= 12\cdot(1-g_P)+12-6\cdot(b_P+1)-(k_P+6) \\
&= 12\cdot(1-g_P)-6b_P-k_P
\end{aligned}$$

■

Let $d_{\min} := \min D$, $d_{\max} := \max D$, and let $h_P := 12\cdot(g_P-g)+6b_P+k_P$ for every possibly canonical patch P. If P is an iterated ancestor of a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$, then it holds

$$\begin{aligned}
h_P &= 12\cdot(g_P-g)+6b_P+k_P \\
&= 12\cdot(1-g)-(12\cdot(1-g_P)-6b_P-k_P) \\
&= fd(M)-fd(P)
\end{aligned}$$

Theorem 8.11 (lower Euler criterion) *For a possibly canonical patch P, it holds:*

- *If $d_{\min} \le 5$ and $f'_P + \lceil h_P \cdot \frac{1}{6-d_{\min}} \rceil > f$, then P is not a canonical patch.*
- *If $d_{\min} \ge 6$ and $h_P > 0$, then P is not a canonical patch.*
- *If $d_{\min} \ge 7$ and $f'_P + \lfloor h_P \cdot \frac{1}{6-d_{\min}} \rfloor < f$, then P is not a canonical patch.*

Proof: Let P be an iterated ancestor of some map $M \in \mathcal{M}(g,n,D,b_l,b_u)$. Then the difference h_P between $fd(M)$ and $fd(P)$ is subdued by the agglutinations that transform P into M. We will prove that this difference cannot be subdued in the 3 above cases. Let $f' := \sum_{d\in D}\big(p_d(M)-p_d(P)\big)$. It holds $p_d(M)-p_d(P) \ge 0$ for every $d \in D$ so that $f' \ge 0$. Moreover, it holds $f'_P + f' = f$. With Lemma 8.10, it holds

$$\begin{aligned}
h_P &= fd(M)-fd(P) \\
&= \sum_{d\in D}(6-d)\cdot p_d(M) - \sum_{d\in D}(6-d)\cdot p_d(P) \\
&= \sum_{d\in D}(6-d)\cdot\big(p_d(M)-p_d(P)\big) \\
&\le \sum_{d\in D}(6-d_{\min})\cdot\big(p_d(M)-p_d(P)\big) \\
&= (6-d_{\min})\cdot\sum_{d\in D}\big(p_d(M)-p_d(P)\big) \\
&= (6-d_{\min})\cdot f'
\end{aligned}$$

- If $d_{\min} \leq 5$, then $h_P \leq (6-d_{\min}) \cdot f'$ implies $f' \geq h_P \cdot \frac{1}{6-d_{\min}}$, and since f' is an integer, it holds $f' \geq \lceil h_P \cdot \frac{1}{6-d_{\min}} \rceil$ and hence $f = f'_P + f' \geq f'_P + \lceil h_P \cdot \frac{1}{6-d_{\min}} \rceil$. So if $f'_P + \lceil h_P \cdot \frac{1}{6-d_{\min}} \rceil > f$, then P is not canonical because P is no iterated ancestor of any map $M \in \mathcal{M}(g, n, D, b_l, b_u)$.
- If $d_{\min} = 6$, then $(6 - d_{\min}) \cdot f' = 0$ and hence P cannot be canonical if $h_P > 0$.
- If $d_{\min} \geq 7$, then $h_P \leq (6-d_{\min}) \cdot f'$ implies $f' \leq h_P \cdot \frac{1}{6-d_{\min}}$, and since f' is an integer, it holds $f' \leq \lfloor h_P \cdot \frac{1}{6-d_{\min}} \rfloor$ and hence $f = f'_P + f' \leq f'_P + \lfloor h_P \cdot \frac{1}{6-d_{\min}} \rfloor$. So if $f'_P + \lfloor h_P \cdot \frac{1}{6-d_{\min}} \rfloor < f$, then P cannot be canonical. In particular, P cannot be canonical if $h_P > 0$ because then $\lfloor h_P \cdot \frac{1}{6-d_{\min}} \rfloor < 0$. ■

Theorem 8.12 (upper Euler criterion) *For a possibly canonical patch P, it holds:*

- *If $d_{\max} \geq 7$ and $f'_P + \lceil h_P \cdot \frac{1}{6-d_{\max}} \rceil > f$, then P is not a canonical patch.*
- *If $d_{\max} \leq 6$ and $h_P < 0$, then P is not a canonical patch.*
- *If $d_{\max} \leq 5$ and $f'_P + \lfloor h_P \cdot \frac{1}{6-d_{\max}} \rfloor < f$, then P is not a canonical patch.*

Proof: Let P be an iterated ancestor of a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$. As above, let $f' := \sum_{d \in D} \big(p_d(M) - p_d(P)\big)$. It holds $p_d(M) - p_d(P) \geq 0$ for every $d \in D$ so that $f' \geq 0$. Moreover, it holds $f'_P + f' = f$. With Lemma 8.10, it holds (compare above)

$$\begin{aligned} h_P &= \sum_{d \in D} (6-d) \cdot \big(p_d(M) - p_d(P)\big) \\ &\geq \sum_{d \in D} (6-d_{\max}) \cdot \big(p_d(M) - p_d(P)\big) \\ &= (6 - d_{\max}) \cdot f' \end{aligned}$$

- If $d_{\max} \leq 5$, then $h_P \geq (6 - d_{\max}) \cdot f'$ implies $f' \leq h_P \cdot \frac{1}{6-d_{\max}}$, and since f' is an integer, it holds $f' \leq \lfloor h_P \cdot \frac{1}{6-d_{\max}} \rfloor$ and hence $f = f'_P + f' \leq f'_P + \lfloor h_P \cdot \frac{1}{6-d_{\max}} \rfloor$. So if $f'_P + \lfloor h_P \cdot \frac{1}{6-d_{\max}} \rfloor < f$, then P cannot be canonical. In particular, P cannot be canonical if $h_P < 0$ because then $\lfloor h_P \cdot \frac{1}{6-d_{\max}} \rfloor < 0$.
- If $d_{\max} = 6$, then $(6 - d_{\max}) \cdot f' = 0$ and hence P cannot be canonical if $h_P < 0$.
- If $d_{\max} \geq 7$, then $h_P \geq (6 - d_{\max}) \cdot f'$ implies $f' \geq h_P \cdot \frac{1}{6-d_{\max}}$, and since f' is an integer, it holds $f' \geq \lceil h_P \cdot \frac{1}{6-d_{\max}} \rceil$ and hence $f = f'_P + f' \geq f'_P + \lceil h_P \cdot \frac{1}{6-d_{\max}} \rceil$. So if $f'_P + \lceil h_P \cdot \frac{1}{6-d_{\max}} \rceil > f$, then P cannot be canonical. ■

Remarks:

- Theorem 8.12 does not need to be considered if $d_{\max} \leq 5$ because then due to formula 1.3, it must hold $g = 0$ and $f \leq 12$. There is only a tiny number of maps M with $g_M = 0$, $f_M \leq 12$, and $\deg(F) \leq 5$ for every face F in M.
- Theorem 8.12 does not need to be considered if $g = 0$: If $g = 0$, then $g_P = 0$ for every possibly canonical patch P and hence $h_P = 6 + k_P > 0$. If $d_{\max} \leq 5$, then Theorem 8.12 is not interesting anyway as stated above, if $d_{\max} = 6$, then the condition $h_P < 0$ is never satisfied, and if $d_{\max} \geq 7$, then with $h_P > 0$, it holds $\lceil h_P \cdot \frac{1}{6-d_{\max}} \rceil < 0$ and hence the condition $f'_P + \lceil h_P \cdot \frac{1}{6-d_{\max}} \rceil > f$ is never satisfied.
- If $g = 0$, then Theorem 8.11 implies an upper limit for f'_P in a canonical patch P depending on $d_{\min}$: It holds $d_{\min} \leq 5$ (see Section 1.4) and $h_P = 6 + k_P \geq 6$ (because $b_P = 1$ and $g_P = 0$). Hence due to Theorem 8.11, P is not canonical in the following cases:
  - $f'_P > f - 6$ if $d_{\min} = 5$
  - $f'_P > f - 3$ if $d_{\min} = 4$

  - $f'_P > f - 2$ if $d_{\min} \in \{1, 2, 3\}$
- The Euler criteria need not be applied to patches P that are obtained by agglutinating one patch Q with itself:
  - If P is obtained by agglutinating one patch Q with itself along an intersection or enclosure path, then $s(Q) = (f'_Q, g_Q, b_Q) = (f'_P, g_P - 1, b_P + 1)$. Moreover, $k_P = k_Q - 6$. Hence it holds

$$\begin{aligned} h_P &= 12 \cdot (g_P - g) + 6b_P + k_P \\ &= 12 \cdot (g_Q + 1 - g) + 6 \cdot (b_Q - 1) + k_Q - 6 \\ &= 12 \cdot (g_Q - g) + 6b_Q + k_Q \\ &= h_Q \end{aligned}$$

    Since $f'_P = f'_Q$, the Euler criteria are satisfied for P if and only if they are satisfied for Q. Hence if one of the Euler criteria is not satisfied for Q, then Q is not stored, and since the patches are generated with increasing size, P is not obtained by agglutinating Q with itself at all.
  - If P is obtained by agglutinating one patch Q with itself along a linkage path, then $s(Q) = (f'_Q, g_Q, b_Q) = (f'_P, g_P, b_P - 1)$. Moreover, $k_P = k_Q - 6$. Hence it holds

$$\begin{aligned} h_P &= 12 \cdot (g_P - g) + 6b_P + k_P \\ &= 12 \cdot (g_Q - g) + 6 \cdot (b_Q + 1) + k_Q - 6 \\ &= 12 \cdot (g_Q - g) + 6b_Q + k_Q \\ &= h_Q \end{aligned}$$

    Since $f'_P = f'_Q$, the Euler criteria are satisfied for P if and only if they are satisfied for Q.

8.7.4 The extended Euler criterion

The Euler criterion can be extended by considering the limit functions b_l and b_u. In Section 8.7.3, the lower and upper bounds for f' were determined just by considering $d_{\min}$ or $d_{\max}$. The idea was that an abundance of large faces must be compensated by additional small faces and vice versa, and in order to check how many additional faces are needed at least, the largest permitted face degree $d_{\max}$ or the smallest permitted face degree $d_{\min}$ was considered. Since the number of faces having the smallest or largest permitted degree is limited, the bounds for f' can be improved by considering only the limited number of faces. We will perform this improvement in two steps.

Let b_u and b_l be determined as sharply as possible as described in Section 6.1. Let P be a possibly canonical patch. Let $a_d(P) := \max\{0, b_l(d) - p_d(P)\}$ as in Section 8.7.2. Since $a_d(P)$ is the minimum number of inner faces of degree d that *must* be agglutinated with P (via other patches) in order to obtain a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, these additional faces can already be included into the estimation of Section 8.7.3. Hence in the first step, we obtain the following theorems:

Theorem 8.13 *For a possibly canonical patch P, it holds:*

- *If $d_{\min} \leq 5$ and $f'_P + \sum_{d \in D} a_d(P) + \lceil \frac{1}{6 - d_{\min}} \cdot (h_P - \sum_{d \in D} (6 - d) \cdot a_d(P)) \rceil > f$, then P is not a canonical patch.*
- *If $d_{\min} \geq 6$ and $h_P - \sum_{d \in D} (6 - d) \cdot a_d(P) > 0$, then P is not a canonical patch.*

- *If $d_{\min} \geq 7$ and $f'_P + \sum_{d \in D} a_d(P) + \lfloor \frac{1}{6-d_{\min}} \cdot (h_P - \sum_{d\in D}(6-d)\cdot a_d(P)) \rfloor < f$, then P is not a canonical patch.*

Proof: Let P be an iterated ancestor of some map $M \in \mathcal{M}(g,n,D,b_l,b_u)$. Then the difference h_P between $fd(M)$ and $fd(P)$ is subdued by the agglutinations that transform P into M. Let $f' := \sum_{d\in D}\big(p_d(M) - p_d(P) - a_d(P)\big)$. It holds $p_d(M) \geq p_d(P)$ and $p_d(M) \geq b_l(d)$ for every $d \in D$. Moreover, it holds $p_d(P) + a_d(P) = \max\{p_d(P), b_l(d)\}$ for every $d \in D$. Thus $p_d(M) - p_d(P) - a_d(P) \geq 0$ for every $d \in D$. Moreover, $f' \geq 0$ and $f'_P + \sum_{d\in D} a_d(P) + f' = f$. With Lemma 8.10, it holds

$$\begin{aligned}
h_P - \sum_{d\in D}(6-d)\cdot a_d(P) &= fd(M) - fd(P) - \sum_{d\in D}(6-d)\cdot a_d(P) \\
&= \sum_{d\in D}(6-d)\cdot p_d(M) - \sum_{d\in D}(6-d)\cdot p_d(P) - \sum_{d\in D}(6-d)\cdot a_d(P) \\
&= \sum_{d\in D}(6-d)\cdot\big(p_d(M) - p_d(P) - a_d(P)\big) \\
&\leq \sum_{d\in D}(6-d_{\min})\cdot\big(p_d(M) - p_d(P) - a_d(P)\big) \\
&= (6-d_{\min})\cdot\sum_{d\in D}\big(p_d(M) - p_d(P) - a_d(P)\big) \\
&= (6-d_{\min})\cdot f'
\end{aligned}$$

- If $d_{\min} \leq 5$, then $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) \leq (6-d_{\min})\cdot f'$ implies $f' \geq \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}}$, and since f' is an integer, it holds $f' \geq \left\lceil \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}} \right\rceil$ and hence
$$f = f'_P + \sum_{d\in D} a_d(P) + f' \geq f'_P + \sum_{d\in D} a_d(P) + \left\lceil \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}} \right\rceil.$$
So if
$$f'_P + \sum_{d\in D} a_d(P) + \left\lceil \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}} \right\rceil > f,$$
then P cannot be an iterated ancestor of any map $M \in \mathcal{M}(g,n,D,b_l,b_u)$ and hence P cannot be canonical.
- If $d_{\min} = 6$, then $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) \leq (6-d_{\min})\cdot f' = 0$ and hence P cannot be canonical if $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) > 0$.
- If $d_{\min} \geq 7$, then $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) \leq (6-d_{\min})\cdot f'$ implies $f' \leq \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}}$, and since f' is an integer, it holds $f' \leq \left\lfloor \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}} \right\rfloor$ and hence
$$f = f'_P + \sum_{d\in D} a_d(P) + f' \leq f'_P + \sum_{d\in D} a_d(P) + \left\lfloor \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}} \right\rfloor.$$
So if
$$f'_P + \sum_{d\in D} a_d(P) + \left\lfloor \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}} \right\rfloor < f,$$
then P cannot be canonical. In particular, P is not canonical if $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) > 0$ because then $\left\lfloor \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\min}} \right\rfloor < 0$. ∎

Theorem 8.14 *For a possibly canonical patch P, it holds:*

- *If $d_{\max} \geq 7$ and $f'_P + \sum_{d\in D} a_d(P) + \lceil \frac{1}{6-d_{\max}} \cdot (h_P - \sum_{d\in D}(6-d)\cdot a_d(P))\rceil > f$, then P is not a canonical patch.*
- *If $d_{\max} \leq 6$ and $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) < 0$, then P is not a canonical patch.*
- *If $d_{\max} \leq 5$ and $f'_P + \sum_{d\in D} a_d(P) + \lfloor \frac{1}{6-d_{\max}} \cdot (h_P - \sum_{d\in D}(6-d)\cdot a_d(P))\rfloor < f$, then P is not a canonical patch.*

Proof: Let P be an iterated ancestor of some map $M \in \mathcal{M}(g,n,D,b_l,b_u)$. Let $f' := \sum_{d\in D}\big(p_d(M) - p_d(P) - a_d(P)\big)$. It holds $p_d(M) - p_d(P) - a_d(P) \geq 0$ for every $d \in D$. Moreover, $f' \geq 0$ and $f'_P + \sum_{d\in D} a_d(P) + f' = f$. With Lemma 8.10, it holds (compare above)

$$\begin{aligned} h_P - \sum_{d\in D}(6-d)\cdot a_d(P) &= \sum_{d\in D}(6-d)\cdot\big(p_d(M) - p_d(P) - a_d(P)\big) \\ &\geq \sum_{d\in D}(6-d_{\max})\cdot\big(p_d(M) - p_d(P) - a_d(P)\big) \\ &= (6-d_{\max})\cdot\sum_{d\in D}\big(p_d(M) - p_d(P) - a_d(P)\big) \\ &= (6-d_{\max})\cdot f'. \end{aligned}$$

- If $d_{\max} \leq 5$, then $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) \geq (6-d_{\max})\cdot f'$ implies $f' \leq \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}}$, and since f' is an integer, it holds $f' \leq \left\lfloor \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}} \right\rfloor$ and hence

  $$f = f'_P + \sum_{d\in D} a_d(P) + f' \leq f'_P + \sum_{d\in D} a_d(P) + \left\lfloor \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}} \right\rfloor.$$

  So if

  $$f'_P + \sum_{d\in D} a_d(P) + \left\lfloor \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}} \right\rfloor < f,$$

  then P cannot be canonical. In particular, P is not canonical if $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) < 0$ because then $\lfloor \frac{1}{6-d_{\max}} \cdot (h_P - \sum_{d\in D}(6-d)\cdot a_d(P))\rfloor < 0$.
- If $d_{\max} = 6$, then $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) \geq (6-d_{\max})\cdot f' = 0$ and hence P cannot be canonical if $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) < 0$.
- If $d_{\max} \geq 7$, then $h_P - \sum_{d\in D}(6-d)\cdot a_d(P) \geq (6-d_{\max})\cdot f'$ implies $f' \geq \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}}$, and since f' is an integer, it holds $f' \geq \left\lceil \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}} \right\rceil$ and hence

  $$f = f'_P + \sum_{d\in D} a_d(P) + f' \geq f'_P + \sum_{d\in D} a_d(P) + \left\lceil \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}} \right\rceil.$$

  So if

  $$f'_P + \sum_{d\in D} a_d(P) + \left\lceil \frac{h_P - \sum_{d\in D}(6-d)\cdot a_d(P)}{6-d_{\max}} \right\rceil > f,$$

  then P cannot be canonical. ■

Remarks:

- Like Theorem 8.12, Theorem 8.14 does not need to be considered if $d_{\max} \leq 5$.

- Like Theorems 8.11 and 8.12, Theorems 8.13 and 8.14 need not be applied to patches P that are obtained by agglutinating one patch Q with itself.

In the second step, we consider which faces can be used to compensate an abundance of large or small faces. Let $r_d(P) := b_u(d) - \max\{p_d(P), b_l(d)\}$ for every $d \in D$. In a possibly canonical patch P, it holds $r_d(P) \geq 0$ for every $d \in D$. Since $\max\{p_d(P), b_l(d)\} = p_d(P) + a_d(P)$, $r_d(P)$ is the maximum number of inner d–gons that may still be agglutinated with P (via other patches) even if $a_d(P)$ additional d–gons are already considered. Let

$$b_i(P,x) \;:=\; (6-i)\cdot x + \sum_{\{d\in D\,|\,d<i\}} (6-d)\cdot r_d(P)$$

where $i \in D$ and $x \in \{0, \dots, r_i(P)\}$. If $j \in D$ and $x \in \{0, \dots, r_j(P)\}$ such that

$$f'_P + \sum_{d\in D} a_d(P) + x + \sum_{\{d\in D\,|\,d<j\}} r_d(P) = f, \tag{8.1}$$

then $b_j(P,x)$ is the biggest possible contribution to the sum in formula 1.3 by (inner) faces that may still be agglutinated with P (and are not yet considered). Analogously, define

$$t_i(P,x) \;:=\; (6-i)\cdot x + \sum_{\{d\in D\,|\,d>i\}} (6-d)\cdot r_d(P)$$

where $i \in D$ and $x \in \{0, \dots, r_i(P)\}$. If formula 8.1 holds, then $t_j(P,x)$ is the smallest possible contribution to the sum in formula 1.3 by faces that may still be agglutinated with P (and are not yet considered). In both cases, j and x are unique.

Theorem 8.15 (extended lower Euler criterion) *Let P be a possibly canonical patch. Let $j \in D$ and $x \in \{0, \dots, r_j(P)\}$ such that formula 8.1 holds. Then it holds: If $b_j(P,x) < h_P - \sum_{d\in D}(6-d)\cdot a_d(P)$, then P is not canonical.*

Proof: Let P be an iterated ancestor of a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$. Let

$$fd' := \sum_{d\in D}(6-d)\cdot p_d(M) - \Big(\sum_{d\in D}(6-d)\cdot p_d(P) + \sum_{d\in D}(6-d)\cdot a_d(P)\Big)$$

be the difference between the contributions to the sum in formula 1.3 given by the faces of M on the one hand and by the inner faces of P as well as by the additionally considered faces on the other hand. It holds $b_j(P,x) \geq fd'$ as mentioned above. Moreover, it holds

$$\begin{aligned} fd' &= \sum_{d\in D}(6-d)\cdot p_d(M) - \Big(\sum_{d\in D}(6-d)\cdot p_d(P) + \sum_{d\in D}(6-d)\cdot a_d(P)\Big) \\ &= fd(M) - fd(P) - \sum_{d\in D}(6-d)\cdot a_d(P) \\ &= h_P - \sum_{d\in D}(6-d)\cdot a_d(P) \end{aligned}$$

So if $b_j(P,x) < fd' = h_P - \sum_{d\in D}(6-d)\cdot a_d(P)$, then P cannot be an iterated ancestor of any map $M \in \mathcal{M}(g,n,D,b_l,b_u)$, and hence P cannot be canonical. ∎

Theorem 8.16 (extended upper Euler criterion) *Let P be a possibly canonical patch. Let $j \in D$ and $x \in \{0, \dots, r_j(P)\}$ such that formula 8.1 holds. Then it holds: If $t_j(P,x) > h_P - \sum_{d\in D}(6-d)\cdot a_d(P)$, then P is not canonical.*

Proof: Let P be an iterated ancestor of a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$. Let fd' be defined as in the proof of Theorem 8.15. Then $t_j(P, x) \leq fd'$ as mentioned above. So if $t_j(P, x) > fd' = h_P - \sum_{d \in D}(6-d) \cdot a_d(P)$, then P cannot be canonical. ■

Remarks:

- Like the classical Euler criteria, the extended Euler criteria need not be applied to patches P that are obtained by agglutinating one patch Q with itself because then $h_P = h_Q$, $a_d(P) = a_d(Q)$, $b_d(P, x) = b_d(Q, x)$, and $t_d(P, x) = t_d(Q, x)$ for every $d \in D$ and every $x \in \{0, \ldots, r_d(P)\}$.
- Clearly, if a patch P does not satisfy Theorem 8.11, then P neither satisfies Theorem 8.13. Analogously, if a patch P does not satisfy Theorem 8.13, then P neither satisfies Theorem 8.15. The same holds for Theorems 8.12, 8.14, and 8.16. Hence Theorems 8.14 and 8.16 need not be considered if $g = 0$. Since Theorems 8.15 and 8.16 filter out more non–canonical patches than Theorems 8.13 and 8.14, Theorems 8.13 and 8.14 are not implemented in *CGF*. However, Theorems 8.11 and 8.12 are implemented, for the following reasons:
  - Theorems 8.15 and 8.16 can only be applied if the numbers $p_d(P)$ are known for P. In order to determine these numbers before P is stored, they must be derived from the (canonical) ancestors of P. So either the numbers must be stored for each patch, or the numbers must be derived by tracing the iterated canonical ancestors of P recursively until the tiles are reached. In *CGF*, the first possibility is realized. So for each patch, $|D|$ additional numbers, namely the *face vector*, must be stored. Since the extended Euler criteria filter out many more non–canonical patches than the Euler criteria only if many different face degrees are permitted (compare Lemma 6.1) or if b_l and b_u are very restricted, the extended Euler criteria are invoked only if the user explicitly says so or if the user provides non–trivial limits b_l and b_u (because then, the face numbers must be stored anyway).
  - Since the Euler criteria depend only on h_P (i.e. on g_P, b_P, and k_P) and f'_P but not on $p_d(P)$, whole subtrees of the patch tree can be skipped during the generation of the patches: Suppose we want to generate patches P of size $s(P) = (f'_P, g_P, b_P)$. We assume that P is obtained by agglutinating two patches Q_1 and Q_2 (because otherwise, as stated above, the Euler criteria need not be applied). Let (m_1, c_{Q_1}, x) with $m_1 = b_{Q_1}$ and $c_{Q_1} = \langle c_1^{Q_1}, \ldots, c_{m_1}^{Q_1} \rangle$ be the boundary code of Q_1, and let (m_2, c_{Q_2}, y) with $m_2 = b_{Q_2}$ and $c_{Q_2} = \langle c_1^{Q_2}, \ldots, c_{m_2}^{Q_2} \rangle$ be the boundary code of Q_2. It holds $f'_P = f'_{Q_1} + f'_{Q_2}$, $g_P = g_{Q_1} + g_{Q_2}$, $b_P = b_{Q_1} + b_{Q_2} - 1$, and $k_P = k_{Q_1} + k_{Q_2} - 6$ where $k_{Q_1} = \sum_{i=1}^{m_1} c_i^{Q_1}$ and $k_{Q_2} = \sum_{i=1}^{m_2} c_i^{Q_2}$. Hence h_P and f'_P are independent from the sequences x and y. So the Euler criteria can be applied to *all* pairs of patches that match the above boundary codes where x and y are arbitrary sequences. In the patch tree, all patches that match a boundary code (m_1, c_{Q_1}, x) with an arbitrary sequence x are found in one subtree whose tree–source is the end node of the tree–path which is given by f'_{Q_1}, g_{Q_1}, b_{Q_1}, and c_{Q_1} (compare Figure 8.8 on page 125). The same holds for the patches that match a boundary code (m_2, c_{Q_2}, y) with an arbitrary sequence y. So a big set of pairs of patches can be tested in one passage as soon as the tree–sources of the corresponding subtrees are reached while tracing the patch tree.

    Sometimes one can skip even bigger subtrees: Suppose Q_1 or at least m_1 and c_{Q_1} are fixed and it is searched for patches Q_2 that can be agglutinated with Q_1. Since $\sum_{i=1}^{j} c_i^{Q_2} \leq \sum_{i=1}^{m_2} c_i^{Q_2} = k_{Q_2}$ for every $j \leq m_2$, it holds

$$\begin{aligned} h_j := & \ 12 \cdot (g_P - g) + 6b_P + k_{Q_1} + (\sum_{i=1}^{j} c_i^{Q_2}) - 6 \\ \leq & \ 12 \cdot (g_P - g) + 6b_P + k_{Q_1} + k_{Q_2} - 6 \\ = & \ 12 \cdot (g_P - g) + 6b_P + k_P \\ = & \ h_P \end{aligned}$$

for every $j \leq m_2$, and hence if $d_{\min} \leq 5$ and $f'_P + \lceil h_j \cdot \frac{1}{6-d_{\min}} \rceil > f$, then P is not canonical. Hence sometimes it suffices to know the first j entries of c_{Q_2} in order to avoid obtaining non–canonical patches. As before, all patches of a fixed size that match the first j entries of c_{Q_2} can be found in one subtree of the patch tree.

8.7.5 Recursion

In order to construct $\mathcal{M}(g, n, D, b_l, b_u)$, every canonical patch satisfying the above patch preventing criteria must be generated. However, not every of these patches must be *stored*. Let Q be a possibly canonical patch having one of the following 3 properties:

- $f'_Q > \frac{f}{2}$
- $f'_Q = \frac{f}{2}$ and $g_Q > \frac{g}{2}$
- $f'_Q = \frac{f}{2}$ and $g_Q = \frac{g}{2}$ and $b_Q > \frac{g}{2} + 1$

If Q is agglutinated with another patch Q_2 in order to obtain a patch P, then $s(P) = (f'_Q + f'_{Q_2}, g_Q + g_{Q_2}, b_Q + b_{Q_2} - 1)$. Since $f \geq f'_P = f'_Q + f'_{Q_2}$, $g \geq g_P = g_Q + g_{Q_2}$, and $g + 1 \geq b_P = b_Q + b_{Q_2} - 1$ (see Theorem 8.1), it holds $f'_{Q_2} \leq f - f'_Q$, $g_{Q_2} \leq g - g_Q$, and $b_{Q_2} \leq g + 2 - b_Q$, and hence Q_2 does *not* have any of the above three properties:

- if $f'_Q > \frac{f}{2}$, then $f'_{Q_2} \leq f - f'_Q < \frac{f}{2}$
- if $f'_Q = \frac{f}{2}$ and $g_Q > \frac{g}{2}$, then $f'_{Q_2} \leq f - f'_Q = \frac{f}{2}$ and $g_{Q_2} \leq g - g_Q < \frac{g}{2}$
- if $f'_Q = \frac{f}{2}$ and $g_Q = \frac{g}{2}$ and $b_Q > \frac{g}{2} + 1$, then $f'_{Q_2} \leq \frac{f}{2}$, $g_{Q_2} \leq \frac{g}{2}$, and $b_{Q_2} \leq g + 2 - b_Q < \frac{g}{2} + 1$

In the same way, it can be shown that if Q is agglutinated with one or two other patches in order to obtain a cubic map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, then the other patches do not have any of the above 3 properties. Hence with the following modification, it suffices to store all patches that do *not* have any of the above 3 properties: Let $s = (\frac{f}{2}, \frac{g}{2}, \frac{g}{2} + 1)$ be a triple whose entries are not necessarily integers. s is called the **threshold size**. If s contains an entry which is not an integer, then s cannot be the size of a patch. But that is not important. First, all patches P satisfying $s(P) \leq s$ are generated and stored with increasing size as before. After that, we generate only those patches of size bigger than s whose canonical ancestors each have a size at most s, i.e. all canonical ancestors are stored. These patches can be generated with increasing size as usual. Here it holds of course that the bigger the size $s(P)$ of a patch P, the smaller the number of possibilities where all canonical ancestors of P have a size at most s. But that does not matter. For every generated patch P with $s(P) > s$, we immediately perform all agglutinations of P (with itself or other patches) that yield further possibly canonical patches or cubic maps $M \in \mathcal{M}(g, n, D, b_l, b_u)$. We will not miss any agglutination because as stated above, for all patches Q that are agglutinated with P, it holds $s(Q) \leq s$ and hence Q has been stored before. The same holds for all iterated descendants of P so that as soon as an iterated descendant P' of P is generated, all agglutinations of P' that yield further possibly canonical patches or cubic maps are immediately performed. Thus, we obtain a recursive part in the algorithm where the patches are not necessarily generated with increasing size. Since the patches are agglutinated immediately in every possible way, they need not be stored. No patch will be missed during the recursion because every canonical patch has at most one canonical ancestor whose size is bigger than s.

So in the recursive part of the algorithm, every possibly canonical patch whose size is bigger than s is generated. Moreover, every agglutination of patches along Petrie paths (in order to obtain maps of $\mathcal{M}(g,n,D,b_l,b_u)$) where the size of one patch is bigger than s is performed. Thus there is a very nice side effect: The first cubic map is constructed before the generation of the patches is complete. Hence if *CGF* supplies some map displaying program like *CaGe* [21] with maps, then the first maps will in general be provided earlier than without recursion. The users of *CaGe* can then spend the time which is needed to construct the remaining maps by investigating the first maps.

The agglutination of patches along Petrie paths where the size of each patch is at most s can be done after the patch generation is complete or before the recursive part of the patch generation takes place (thus the "side effect" is intensified). A patch P with $s(P) \leq s$, i.e. a patch that must be stored (if it is not detected to be non–canonical), is called a **permanent patch**.

8.7.6 Advanced recursion

The recursion helps to save a lot of memory because the number of patches that need to be stored becomes much smaller with recursion. The definition of the *size* of a patch (page 100) is adapted to the recursion in the following sense: When all patches up to a given size are generated, then in general, most of the patches have genus 0. On the other hand, the number of patches strongly increases with an increasing number of inner faces. Thus in order to store as few patches as possible, it is useful to give the number of inner faces in a patch P a higher priority than the genus of P when defining the size of P. Theorem 8.8 supports this argumentation.

However, by changing the definition of the size of a patch, the recursion becomes even more effective. Let $c_P := \sum_{\{d\in D|d>6\}}(d-6)\cdot p_d(P)$. c_P is called the **critical sum** of P. Analogously, we define the critical sum $c_M := \sum_{\{d\in D|d>6\}}(d-6)\cdot p_d(M)$ for every map $M \in \mathcal{M}(g,n,D,b_l,b_u)$. An upper limit $c_{\max} := \max\{c_M | M \in \mathcal{M}(g,n,D,b_l,b_u)\}$ that of course satisfies $c_{\max} \geq c_P$ for any canonical patch P can be derived when computing all possible face vectors for the maps in $\mathcal{M}(g,n,D,b_l,b_u)$ (see Section 6.1). Now the **advanced size** $s'(P)$ of a patch P with $s(P) = (f'_P, g_P, b_P)$ is defined as $s'(P) = (c_P, f'_P, g_P, b_P)$, and the patches are generated as described in Section 8.7.5 but with increasing *advanced* size. This kind of recursive patch generation is called the **advanced recursion**. In the patch tree, an additional tree–level is inserted: The i–th level becomes the $(i+1)$–th level, and the new first tree–level contains the targets of those branches that represent the different possible critical sums. These branches have a common source which is the new tree–source. As in Section 8.7.5, a patch P having one of the following 4 properties

- $c_P > \frac{1}{2}c_{\max}$
- $c_P = \frac{1}{2}c_{\max}$ and $f'_P > \frac{f}{2}$

- $c_P = \frac{1}{2}c_{\max}$ and $f'_P = \frac{f}{2}$ and $g_P > \frac{g}{2}$
- $c_P = \frac{1}{2}c_{\max}$ and $f'_P = \frac{f}{2}$ and $g_P = \frac{g}{2}$ and $b_P > \frac{g}{2} + 1$

can be agglutinated only with patches having none of these 4 properties, and hence a patch must be stored only if it has none of the 4 properties. Thus, $s' := (\frac{1}{2}c_{\max}, \frac{f}{2}, \frac{g}{2}, \frac{g}{2} + 1)$ is the **advanced threshold size** for the patch generation. As for the standard recursion, a patch P with $s'(P) \leq s'$ is called a **permanent patch**.

The advanced recursion is more effective than the standard recursion because the number of patches of a given advanced size strongly increases with increasing $c_{\max}$. The reason for this is that the maximum number of convex edges in the possibly canonical patches increases with increasing $c_{\max}$ so that in general, more possibilities for agglutinations are obtained. Moreover, due to Lemma 8.10, it holds

$$
\begin{aligned}
12 \cdot (1 - g_P) - 6b_P - k_P &= fd(P) \\
&= \sum_{d \in D} (6 - d) \cdot p_d(P) \\
&= \sum_{\{d \in D | d \leq 6\}} (6 - d) \cdot p_d(P) \ + \sum_{\{d \in D | d > 6\}} (6 - d) \cdot p_d(P) \\
&= \Big(\sum_{\{d \in D | d \leq 6\}} (6 - d) \cdot p_d(P) \Big) - c_P
\end{aligned}
$$

and hence

$$
0 \leq \sum_{\{d \in D | d \leq 6\}} (6 - d) \cdot p_d(P) \ = \ 12 \cdot (1 - g_P) - 6b_P - k_P + c_P.
$$

So if $6 \notin D$, then c_P even implies an upper limit for f'_P: It holds

$$
c_P = \sum_{\{d \in D | d > 6\}} (d - 6) \cdot p_d(P) \ \geq \sum_{\{d \in D | d > 6\}} (7 - 6) \cdot p_d(P) \ = \sum_{\{d \in D | d > 6\}} p_d(P),
$$

and with $p_6(P) = 0$, it holds

$$
\sum_{\{d \in D | d \leq 6\}} (6 - d) \cdot p_d(P) = \sum_{\{d \in D | d < 6\}} (6 - d) \cdot p_d(P) \geq \sum_{\{d \in D | d < 6\}} (6 - 5) \cdot p_d(P) = \sum_{\{d \in D | d < 6\}} p_d(P).
$$

So

$$
\begin{aligned}
f'_P &= \sum_{d \in D} p_d(P) \\
&= \sum_{\{d \in D | d > 6\}} p_d(P) \ + \sum_{\{d \in D | d \leq 6\}} p_d(P) \\
&\leq c_P + \sum_{\{d \in D | d \leq 6\}} (6 - d) \cdot p_d(P) \\
&= 12 \cdot (1 - g_P) - 6b_P - k_P + c_P + c_P \\
&\leq 6 + 2c_P.
\end{aligned}
$$

This estimation is even quite weak in most cases: If $g_P > 0$ or if e.g. $5 \notin D$, then the upper limit for f'_P becomes much smaller.

So for small critical sums c, only few patches P with $c_P = c$ are generated (especially when $6 \notin D$), and for big critical sums, the generated patches need not be stored. Thus the memory requirement of the algorithm becomes much smaller, and this makes the advanced recursion more effective than the standard recursion.

8.8 Multipatches and advanced patch testing

No matter whether the standard or the advanced recursion is applied, the bottom–up approach can be accelerated. For that, let s be the (advanced) threshold size, and let Q_1 be a patch with $s(Q_1) > s$ so that Q_1 is agglutinated with itself or with permanent patches. We consider the following two situations:

(1) If Q_2 is a permanent patch that may be agglutinated with Q_1, then with a high probability, every patch Q' having the same (advanced) size and the same boundary code as Q_2 may as well be agglutinated with Q_1. The conditions 1, 3, 4, 5, 6, and 7 in Section 8.1 are satsified automatically, the conditions 2 and 8 are satisfied with a high probability (because the number of inner faces of a certain degree in a patch Q' is not completely independent of the boundary code, and the canonicity check for the zig–zag path mainly depends on the boundary codes of Q_1 and Q'). All patches having the same (advanced) size and the same boundary code as Q_2 can be found in one list L which is assigned to some leaf in the patch tree. Let P be the patch which is obtained by agglutinating Q_1 with an entry Q' of L along a zig–zag path p (so that $A(P,p) = \{Q_1, Q'\}$). Then the (advanced) size and the boundary code of P is identical for every entry Q' of L.

(2) If Q_1 is agglutinated with itself along an intersection path or an enclosure path, then exactly one patch is obtained. But if Q_1 is agglutinated along a linkage path p, then in the resulting patch P, the boundary face F_2 containing the end vertex of p might contain several markable boundary vertices. However, the (advanced) size and the boundary code of the resulting patch P are independent from the vertex that is actually marked. All possibilities to mark a vertex incident with F_2 must be considered. If one possibility yields a canonical agglutination, then the conditions 1–7 in Section 8.1 are satisfied for every possiblitiy.

In both situations, several patches can be generated that differ only in the precise incidence structure. But since it is sometimes quite time–consuming to find patches that can be agglutinated with each other and since the agglutinations of patches mainly depend on their sizes and their boundary codes, it is useful to summarize all agglutinations in the above situations. Thus in the first situation, a pointer on L instead of a pointer on Q' is stored in $t(P)$ (see Section 8.4). In the second situation, the stored offset in $t(P)$ is interpreted as the offset between the end vertex of p and a *markable* instead of a *marked* vertex, i.e. the boundary face F_2 remains unmarked and the offset can be used to find the markable vertices in F_2. Thus $t(P)$ represents a set of patches, and whenever further agglutinations including P are performed, implicitly a set of agglutinations is performed in one passage. $t(P)$ is called a **multipatch**. The patches that are summarized in a multipatch $t(P)$ are called **implicit patches** of $t(P)$. P is a representative of the implicit patches of $t(P)$, i.e. if P_1 and P_2 are implicit patches of a multipatch $t(P_1)$, then $t(P_1) = t(P_2)$. It is useful to name a representative P of a multipatch $t(P)$ because exactly one representative P is used for further agglutinations. As soon as P or an iterated descendant of P is agglutinated along a Petrie path in order to obtain a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, implicitly a set of maps is obtained. The elements of this set are obtained by running through all implicit patches of $t(P)$, i.e. by running through all entries of L in the first situation or by running through all markable vertices in the second situation.

With this method, many agglutinations are summarized, especially if the representative of a multipatch again plays the role of Q_1 in one of the above situations (e.g. if a list L_1 with 4 patches and a list L_2 with 5 patches are considered in two different levels of the recursion, then the multipatch in the higher recursion level summarizes 20 implicit patches). But due to the summarization, some patch tests can no longer be performed because they might depend on the precise incidence structure or on the number of inner faces of certain degrees. The tests that are affected are the tests whether the conditions 2 and 8 in Section 8.1 hold and the tests described in Section 8.5. So as soon as cubic maps $M \in \mathcal{M}(g,n,D,b_l,b_u)$ are constructed, these tests must be recovered. In particular, a multipatch might imply patches that would have been detected to be non–canonical if the tests would have been performed immediately.

In order to construct an implicit patch Q of a multipatch $t(P)$, an entry of L must be selected for every patch list L that occurs during the construction, and for every linkage agglutination that is performed during the construction, a markable vertex in the boundary face containing the end vertex of the corresponding zig–zag path must be marked. Let Q be an iterated canonical ancestor of a cubic map M. Before M is constructed, the tests for Q must be recovered in the same order in which they would have been performed if Q had been constructed without summarization. That is, starting with the smallest non–permanent iterated canonical ancestor of Q, all non–permanent iterated canonical ancestors of Q (and Q itself) are traced with increasing (advanced) size, and for every such patch, the following tests must be recovered:

- It must be tested whether condition 2 in Section 8.1 is satisfied. However, if $|D| \leq 2$, then condition 2 can be verified also for the implicit patches of multipatches, i.e. the test can be performed immediately and does not need to be recovered: Every implicit patch Q of a multipatch $t(P)$ has the same (advanced) size and the same boundary code, so Lemma 8.10 on page 141 yields the same value $fd(Q)$ for every implicit patch Q of $t(P)$. Hence we can argue as in the proof of Lemma 6.1.
- It must be tested whether condition 8 in Section 8.1 is satisfied. This condition often does not depend on the incidence structure and on the marked vertices (whenever the boundary vertex having the highest quality is unique), but sometimes it does.
- All tests that are described in Section 8.5 must be considered. However, it is also possible to perform those tests only in the constructed cubic maps because it might be faster to perform one test in one big map M than in many (comparatively small) iterated canonical ancestors of Q. On the other hand, if a test fails already for a small iterated canonical ancestor Q' of Q, then all implicit patches in $t(P)$ that are iterated canonical descendants of Q' need not be considered any longer. None of the two proceedings is uniquely favoured against the other. However, in *CGF* the tests are considered already for the patches, for the following reason: The generation rate of *CGF* is in most cases much lower than 300 structures per second (on an Intel Pentium II computer with 350 MHz). If we consider the one extreme where no test fails, then during the patch generation, many obsolete tests are performed for one cubic map M, but on the other hand it does not happen too often that a cubic map M is constructed, so the additional amount of time for the tests does not matter if at least a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$ is constructed. If we now consider the other extreme where all tests fail, then it saves a lot of time to perform the tests on the lowest possible level. And here it would matter if a lot of time was spent on the construction of invalid maps.

If nearly every test fails, then one may ask if it is useful to use multipatches and postpone the above tests until actually cubic maps are constructed. In particular, if the constructed cubic maps represent Delaney graphs or if b_u is very restricted, then many tests might fail. To this end, *CGF* provides an alternative strategy called the **advanced patch testing** (APT): As before, multipatches are generated. However, whenever a multipatch $t(P)$ is generated, the tests described above are performed immediately. For that, the implicit patches of $t(P)$ must all be constructed. Now the patches that fail the tests will not be considered anymore for the descendants of P. The patches that pass the tests are stored, and these are the only patches that will be considered for the descendants of P. Figure 8.15 shows the effect of the advanced patch testing for an example. In particular, if every implicit patch of $t(P)$ fails a test, then the recursion can backtrack immediately.

The advanced patch testing is performed only if desired. Since the tests are time–consuming and the storage of those patches that pass the tests requires a few memory, the advanced patch testing should be invoked only if the majority of tests fails.

Let $t(Q)$ and $t(P)$ be multipatches. Let i be the recursion level where $t(Q)$ has been generated, and let j be the recursion level where $t(P)$ has been generated. Let $i < j$ so that the implicit patches of $t(Q)$ are iterated canonical ancestors of the implicit patches of $t(P)$. In order to obtain all implicit patches of $t(P)$, the possible variations in the higher recursion level j should be considered first (i.e. if there are two levels i and j with $i < j$ where one of the above two situations occur, then the variations in level j, i.e. choosing a patch out of a patch list or marking a markable boundary vertex, should be performed before they are performed in level i). Thus the results of the tests in the lower level i remain valid until all variations in the upper level j have been performed. This is useful especially if the advanced patch testing is not performed.

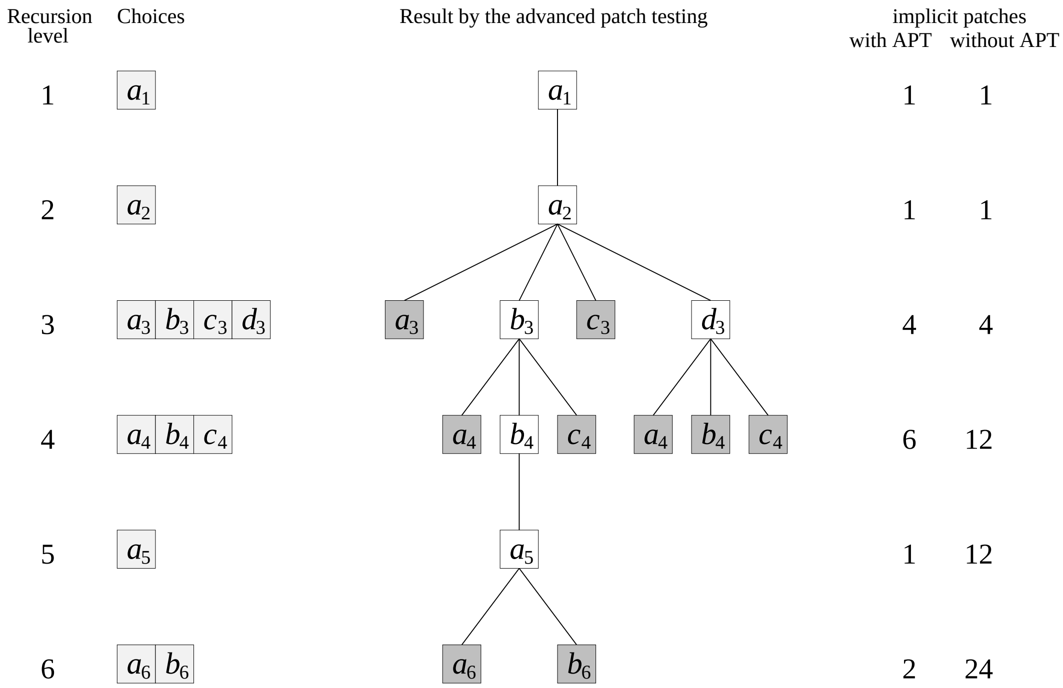

Figure 8.15: A tree–like structure derived by the advanced patch testing. a_i, b_i, c_i, and d_i denote the different possibilities to choose a patch out of a list of permanent patches or to mark a boundary vertex in the i–th recursion level. The grey nodes in the tree correspond to patches that failed a test, the white nodes correspond to patches that passed every test.

8.9 Intervals of genera and face numbers

Sometimes, not a set $\mathcal{M}(g, n, D, b_l, b_u)$ with fixed paramters g and n but a set

$$\bigcup_{\substack{g_{\min}\le g\le g_{\max}\\ n_{\min}\le n\le n_{\max}}} \mathcal{M}(g, n, D, b_l, b_u)$$

for some parameters $n_{\min}$, $n_{\max}$, $g_{\min}$, and $g_{\max}$ is to be constructed. Here it is useful to sum up all construction processes in one process in order to save time. The time can be saved during the generation of the patches: There will be only one patch generation for all parameters g and n. All patches that are possibly canonical for one pair (g,n) are generated. Many patches are canonical for several pairs so that the number of generated patches is hardly higher than the number of patches that would be generated if $\mathcal{M}(g_{\max}, n_{\max}, D, b_l, b_u)$ was to be constructed.

In order to sum up all patch generations, the patch preventing criteria must be adjusted: In order to modify b_l and b_u such that they become as sharp as possible (for the extended Euler criteria), it must be considered that the sharpest possible limits vary with g and n. Here the sharpest possible limits due to *all* pairs (g,n) must be considered. Moreover, the parameter f must be substituted by $f_{\max} := \frac{1}{2}n_{\max} + 2 - 2g_{\min}$ in the Euler criteria.

8.10 Parallelization and recovering

In this section, we describe how the knowledge of Section 4.5 can be applied to the construction of $\mathcal{M}(g, n, D, b_l, b_u)$. Recall Section 4.5.

Since the generation of the permanent patches cannot be divided into a clean forest structure, the bottom–up construction of $\mathcal{M}(g, n, D, b_l, b_u)$ can be parallelized only if recursion is applied. In this case, the set S of permanent patches remains unchanged so that in each subprocess, the patches in S must be generated and stored.

The generation in the recursive part of the algorithm can be divided into a forest structure so that we can proceed as in Section 4.5. Here the tree–levels correspond to the recursion levels. However, given a threshold level l, there might be cubic maps that are constructed in each subprocess because all agglutinated patches are permanent or generated in a lower recursion level than l. Hence we demand that these maps are accepted only in the subprocess p_0. The lower we choose l, the less is the total time consumption of all subprocesses (because the less is the number of common computations that are performed by each subprocess), but the more inhomogeneous the time comsumptions of the single subprocesses become.

In order to provide the possibility of recovering an interrupted process, there must be a possibility to avoid writing all stored patches into the dump file because this process would require too much time and too much disk space. If no recursion is applied, then in general, the generation of the patches is the most time consuming part of the whole algorithm. Hence in this case, there does not exist an efficient way to provide dump files. On the other hand, if recursion is applied, then the recovering of an interrupted process can be performed in the way described in Section 4.5. In the dump files, as usual the number of the current root of the construction forest and the sequence of branch labels representing the tree–path to the current position in the construction forest are written. Moreover, in the case of multipatches, the implicit patches must be numbered, and the numbers of the current implicit patches must be stored, too. When a process is restarted, then first the permanent patches are generated and stored anew. After that, the recursive part of the generation process continues at the position which is stored in the dump file.

The amount of information that has to be stored is quite small. In *CGF*, every 30 minutes a dump file containing about 500 bytes is stored. Hence the loss of time after a restart is at most 30 minutes plus the time which is needed to generate the permanent patches. When recursion is applied, then the generation of the permanent patches is *not* the bottleneck of the bottom–up approach.

Chapter 9

The top–down approach

In this chapter, the so–called *top–down approach* for the construction of $\mathcal{M}(g, n, D, b_l, b_u)$ will be described. The principle of the top–down approach is given by reconstructing the canonical sequences of cuts in the original direction.

9.1 The basic idea

In the *bottom–up* approach, we generate all patches that are possibly canonical and satisfy the patch preventing criteria. While generating the patches, their incidence structures are known or can at least be derived by actually constructing the patches if necessary. As a consequence, the whole patch generation takes place with a high respect to D in the sense that the degree of every inner face of every generated patch is in D.

In the *top–down* approach, we try to reconstruct canonical sequences of cuts in the original direction. Here the incidences of the inner vertices and edges in the obtained patches are almost completely unknown. We only know the valencies of the vertices contained in the boundary faces of the patches. Thus we cannot construct the patches. We can only try to construct the (possibly) canonical paths. Take Figure 9.1 as an example: Here we try to reconstruct a canonical sequence of cuts where we assume that the first canonical path is a Jordan curve path of length 14, the second canonical path is an intersection path of length 3 where the offset between the start vertex and the end vertex in the corresponding boundary face is 8, and the third canonical path is an intersection path of length 1 where the offset between the start vertex and the end vertex in the corresponding boundary face is 8. When the boundary faces containing the start vertices and the end vertices of the paths are uniquely specified, then we can construct a "scaffold" as in Figure 9.1. Here, the elements of the canonical paths are drawn. Moreover, there are some dangling edges indicating the continuations in the paths. However, we do not actually construct scaffolds during the top–down approach. Figure 9.1 is used only to illustrate the idea of the top–down approach. We will see later that it is even not always possible to construct scaffolds immediately. So the top–down approach for the construction of $\mathcal{M}(g, n, D, b_l, b_u)$ is again only a top–down *generation* rather than a *construction*.

So in the top–down approach, for every map and every patch, we specify the possible properties

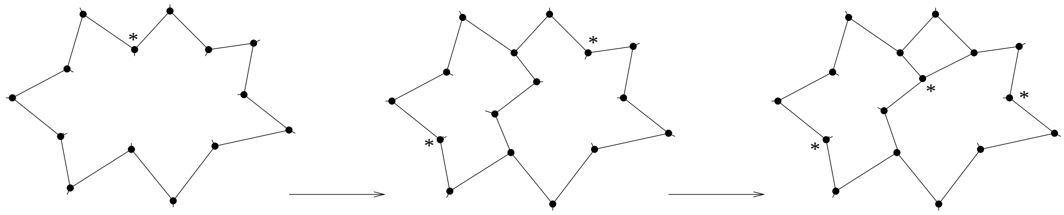

Figure 9.1

of the next canonical cut. We start with very few knowledge, so there are many possibilities for the first canonical cut: Any Petrie path being incident with at most n vertices might be the canonical Petrie path of a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$. So in order to reconstruct the canonical cut sequence of every map in $\mathcal{M}(g, n, D, b_l, b_u)$, we must consider every Jordan curve path of length at most n and every sandwich and dumbbell path of length at most $n + 1$. Suppose we consider a Petrie path p. Then we can uniquely determine the following information:

- the canonical boundary codes of the new boundary faces that are obtained after the cut (due to the Sections 8.6.1, 8.6.2, and 8.6.3)
- the positions of the marked boundary vertices in the new boundary faces (due to the marking rules in Section 7.4)

In order to reconstruct further possibly canonical cuts, we must provide the following information that cannot be derived from the previous cut:

- We must determine how many patches are obtained by the cut (1, 2, or 3 patches in the sandwich or dumbbell case, 1 or 2 patches in the Jordan curve case).
- In case we do not obtain the maximal number of patches, we must determine which obtained patch contains which boundary faces.
- We must determine the sizes of the obtained patches. Here the number of boundary faces has already been determined in the previous steps so that we only need to assign genera and numbers of inner faces to the obtained patches. The genera and the numbers of inner faces are assigned according to Section 6.4 (e.g. the sum of inner faces must be $f = \frac{n}{2} + 2 - 2g$).

Again, every possibility must be considered. Details will be given in Section 9.3. So after the first cut, we obtain 1, 2, or 3 patches whose sizes and whose boundary codes are known (the boundary codes are derived from the canonical boundary codes of the boundary faces). The incidence structure of the patches is not known. So what we obtain is similar to what we have called a *multipatch* in the bottom–up approach (Section 8.8). Let a patch size s and a boundary code C be given. Or to be precise: s is a triple that is interpreted as the size of a patch, and C is a triple (m, c, b) that matches the description of the boundary code of patches on page 120. Then (s, C) is called a **multipatch** in the top–down approach. A multipatch (s, C) represents all patches having the size s and the boundary code C (this set of patches is possibly empty). These patches are called **implicit patches** of (s, C). In order to avoid fussy sentences, we will say that a multipatch (s, C) with $s = (f', g', b')$ *indicates* the size s, the boundary code C, the genus g', the number of inner faces f', and the number of boundary faces b'. This means that all implicit patches of (s, C) have genus g', f' inner faces, and b' boundary faces. Moreover, we will say that (s, C) *indicates* a boundary face with a special property if all implicit patches of (s, C) contain a boundary face having this property. Thus, we avoid tedious sentences like “if all implicit patches of a multipatch contain a boundary face with..." and write "if a multipatch indicates a boundary face with..." instead.

Now the canonical cut sequences of all implicit patches of the obtained multipatches must be reconstructed. This is done recursively: We determine all possibilities for the first cut, and then we reconstruct all possible canonical cut sequences of the implicit patches of the obtained multipatches. As soon as a canonical sequence of cuts has been reconstructed completely, we obtain exactly one patch. In order to determine the possibilities for the first cut for a multipatch, we must choose the type and the length of the canonical zig–zag path. In case of an intersection path or a linkage path, we must also select the end vertex of the path. Then as before, the canonical boundary codes of the new boundary faces and the marked vertices in the new boundary faces can uniquely be determined. In particular, the start vertex of the canonical zig–zag path does not need to be further specified: If the multipatch indicates exactly one boundary face having the highest quality, then the canonical boundary vertex in each implicit patch is unique, and if the multipatch indicates more than one boundary face having the highest quality, then in the implicit patches, these boundary faces cannot be distinguished and exactly one of them is taken as the canonical boundary face. In particular, the canonical boundary codes of the obtained ancestors do not depend on the choice of the canonical boundary face (so that the same multipatches are obtained for each choice).

As before, we must again determine how many ancestors are obtained by the cut and which ancestor contains which boundary faces. Moreover, we must determine the sizes of the obtained ancestors. Details will be given in Section 9.4. The canonical cut sequences of the implicit patches of the obtained multipatches are reconstructed recursively. This recursion is stopped as soon as it approaches a multipatch (s, C) with $s = (1, 0, 1)$. Then it is tested if $C = (m, c, b)$ corresponds to the boundary code of a tile, i.e. $m = 1$, $c = \langle c_1 \rangle$ for some $c_1 \in \mathbb{N}$, and $b = \langle 0 \rangle^{c_1}$. If it does, then it is tested whether c_1 is a permitted degree. If both conditions hold, then the reconstruction is continued, otherwise we must backtrack to a point where a choice can be modified such that we do not obtain the invalid multipatch anymore.

Since the search for patches of a given size and with a given boundary code can be done independently from other patches, the idea of the top–down approach is better illustrated in Figure 9.2 than in Figure 9.1. Here the positions of the marks are uniquely determined by the marking rules. When an appropriate patch has been constructed, it must be rotated such that the mark of the patch matches the mark in the figure. Then the agglutination is performed with respect to the marking rules.

Altogether, the top–down approach is a huge divide and conquer algorithm, i.e. problems are partitioned into smaller subproblems. A "problem" is the reconstruction of a canonical cut sequence of a patch or a cubic map. A problem is solved using a backtrack algorithm. Figure 9.3 shows a cut–out of the search tree for $\mathcal{M}(0, 8, \{4\}, b_l, b_u)$ with $b_l(4) = b_u(4) = 6$. In the search tree, we have "and" nodes and "or" nodes. "or" nodes indicate that there are several possibilities to solve a problem. These nodes occur wherever values can be chosen. An example for that is the first cut: Here the canonical Petrie path p might be a Jordan curve path of length 4 *or* 6 *or* 8 *or* a dumbbell path with $(l_1(p), l_2(p), l_3(p)) = (3, 3, 3)$ *or* .... Of course, in order to construct *every* desired map, *every* possibility must be considered (one after the other). An "and" node indicates that all corresponding subproblems must be solved in order to solve the original problem. These nodes occur whenever a cut is specified such that we obtain more than one ancestor. In this case, the cut can be canonical only if the algorithm succeeds in reconstructing canonical sequences of cuts for all ancestors.

Whenever we try to solve a subproblem which is related to other subproblems by an "and" node, then the other subproblems will also have to be solved. Thus we obtain a stack of open

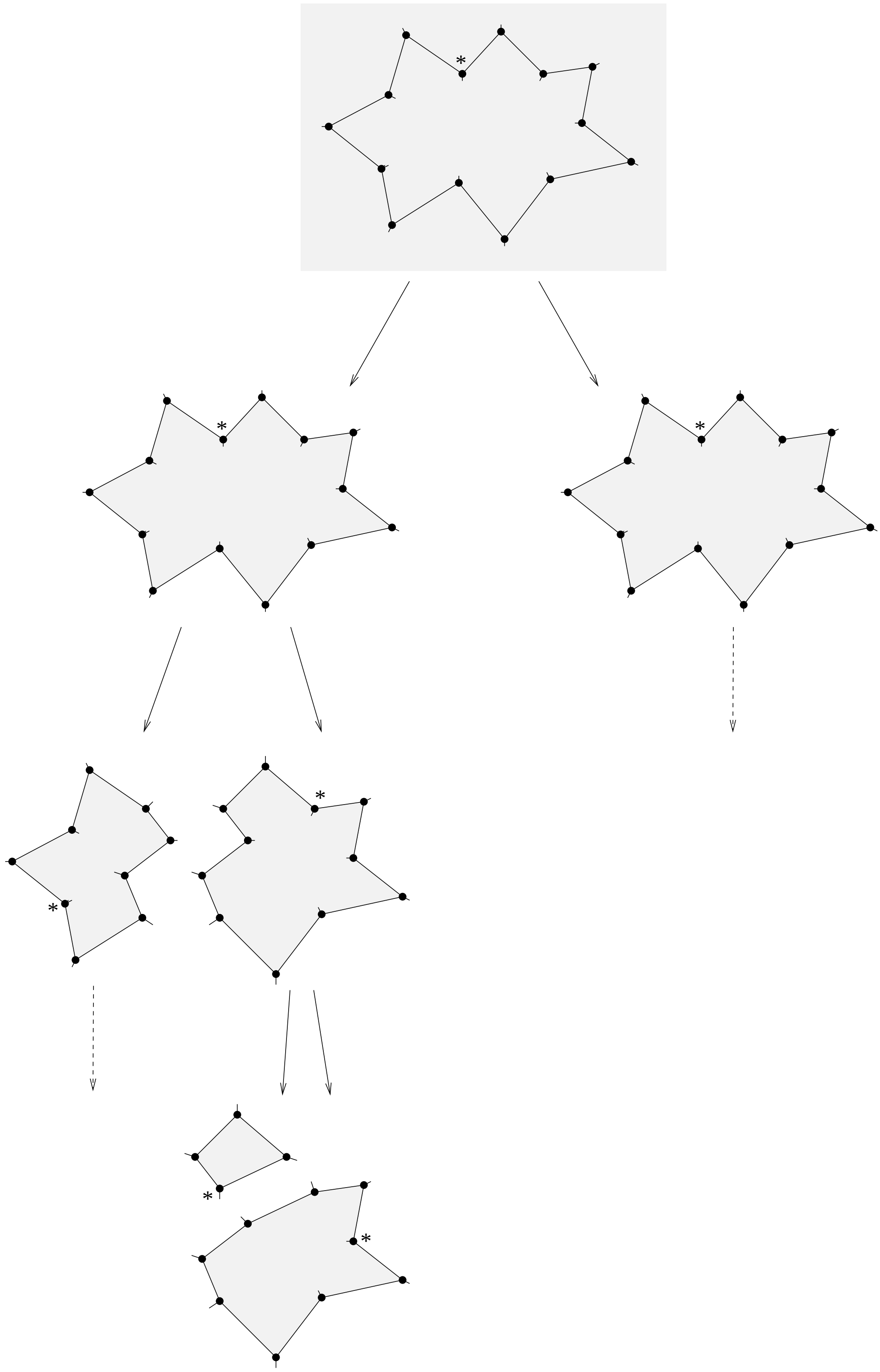

Figure 9.2

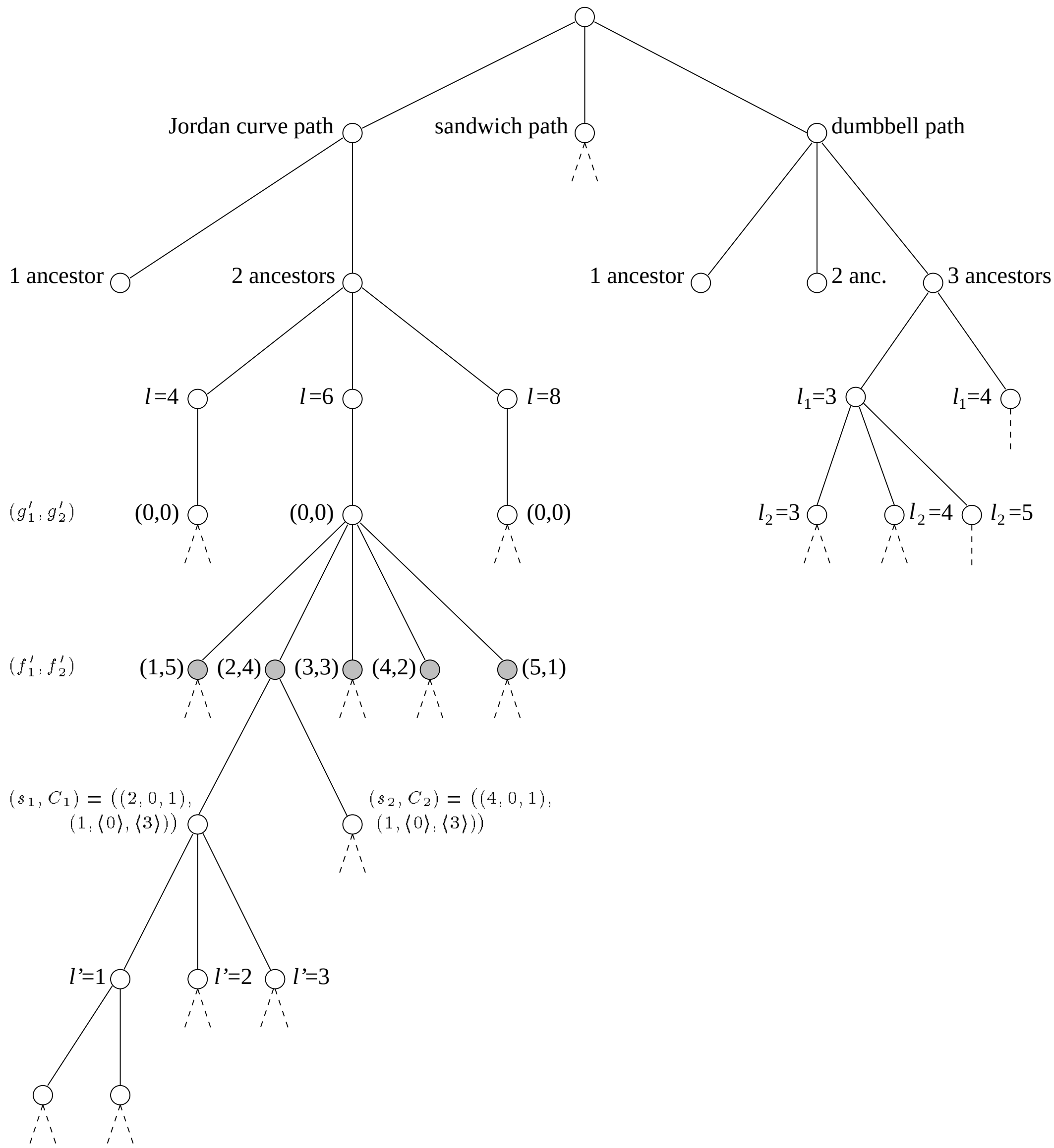

Figure 9.3: The grey nodes are "and" nodes, the white nodes are "or" nodes.

subproblems. Whenever we fail in solving a subproblem which is related to other subproblems by an "and" node, then the other subproblems will not have to be solved anymore. We cancel solving the current subproblem and all subproblems of a common "and" node and advance to the next subproblem of an "or" node. Whenever a subproblem is solved, then we backtrack successfully and continue by solving the remaining open subproblems.

Although the search tree is huge, extremal search paths often end early. For example, if we search for maps of order 100 and begin with a Petrie path which is incident with 99 vertices, then the corresponding patches contain altogether only 1 inner vertex. Hence there are very few possibilities for further cuts. Moreover, sometimes a failure can be detected before tiles are approached. This will also be discussed in the following sections.

Of course, we cannot reconstruct only canonical cut sequences. For example, not every Petrie path incident with at most 8 vertices is a canonical Petrie path in a map $M \in \mathcal{M}(0, 8, \{4\}, b_l, b_u)$ with $b_l(4) = b_u(4) = 6$ (in fact, $\mathcal{M}(0, 8, \{4\}, b_l, b_u)$ contains only one map whose canonical Petrie path is a Jordan curve path of length 6, so starting the reconstruction with a different Petrie path does not yield a canonical cut sequence). So we are only reconstructing *possibly* canonical cuts. The top–down approach will often approach subproblems that cannot be solved. But even when all subproblems are solved for the problem of reconstructing the canonical cut sequence of a cubic map, then only *possibly* a map in $\mathcal{M}(g, n, D, b_l, b_u)$ is found. In the bottom–up approach, an agglutination was called *possibly canonical* when it satisfied the 8 conditions given in Section 8.1. In the top–down approach, we cannot check all of these conditions right after a cut has been performed. Let $\mu = (s, C)$ be a multipatch that has been obtained after a cut (for the following enumeration, recall the conditions in Section 8.1):

(1) This condition cannot be verified unless μ implies a tile (in this case, μ implies no other patch).
(2) One can verify if the limit b_u is still satisfied each time a new tile is approached. If the degree d of the new tile (implied by μ) is in D but the limit $b_u(d)$ is violated by the new tile, then it must be backtracked without success.
(3) This condition will always be satisfied for μ because the numbers of inner faces will always be assigned appropriately to the patches (see Section 9.3 and 9.4).
(4) This condition will always be satisfied, too, because the genera will always be assigned appropriately to the patches (see Section 9.3 and 9.4).
(5) This condition will always be satisfied due to the proof of Theorem 8.1.
(6) This condition will of course always be satisfied because we only consider zig–zag paths that are continued to the right.
(7) Due to Lemma 6.11 and Lemma 6.16, this condition will of course always be satisfied, too.
(8) If μ does not imply a tile and μ indicates a unique boundary face having the highest quality, then the canonical boundary vertex in every implicit patch P of μ is unique and hence the next zig–zag path p that it used to cut P will be canonical, too. If μ indicates more than one boundary face having the highest quality, then the canonical boundary vertex in P is not unique. Since the mark–qualities of the possible canonical boundary vertices cannot be computed in order to distinguish these vertices, p might be non–canonical.

Thus the same problems occur that have already occurred when multipatches were introduced in the bottom–up approach. These problems will be considered in Section 9.5.

9.2 About vertices and inner faces in patches

In this section, we will derive some formulæ that yield relationships between different parameters of a possibly canonical patch P.[1] We denote the number of inner vertices of P by $n_1(P)$, the number of 2–valent boundary vertices by $n_2(P)$, and the number of 3–valent boundary vertices by $n_3(P)$. It holds $k_P = n_2(P) - n_3(P)$ (see proof of Lemma 6.17).

Lemma 9.1 *It holds* $n_1(P) = 2f'_P + 2b_P - n_3(P) - 4 \cdot (1 - g_P)$.

Proof: It holds $f'_P = \sum_{d\in D} p_d(P)$ and hence

$$\sum_{d\in D} p_d(P) \cdot (6-d) = \sum_{d\in D} p_d(P) \cdot 6 - \sum_{d\in D} p_d(P) \cdot d = 6f'_P - \sum_{d\in D} p_d(P) \cdot d.$$

Moreover, it holds $\sum_{d\in D} p_d(P) \cdot d = n_2(P) + 2n_3(P) + 3n_1(P)$: The sum $\sum_{d\in D} p_d(P) \cdot d$ is the number of vertices that are traced if every inner face of P is traced exactly once. This number is $n_2(P) + 2n_3(P) + 3n_1(P)$ because since every boundary vertex is incident with exactly one boundary face and a boundary face does not contain face–bridges, every 2–valent boundary vertex is traced once, every 3–valent boundary vertex is traced twice, and every inner vertex is traced three times. With Lemma 8.10, we have

$$\begin{aligned}
& & \sum_{d\in D} p_d(P) \cdot (6-d) &= 12 \cdot (1-g_P) - 6b_P - k_P \\
&\Leftrightarrow & 6f'_P - \big(n_2(P) + 2n_3(P) + 3n_1(P)\big) &= 12 \cdot (1-g_P) - 6b_P - k_P \\
&\Leftrightarrow & 6f'_P - n_2(P) - 2n_3(P) - 12 \cdot (1-g_P) + 6b_P + k_P &= 3n_1(P) \\
&\Leftrightarrow & 6f'_P - (n_3(P) + k_P) - 2n_3(P) - 12 \cdot (1-g_P) + 6b_P + k_P &= 3n_1(P) \\
&\Leftrightarrow & 6f'_P - 3n_3(P) - 12 \cdot (1-g_P) + 6b_P &= 3n_1(P) \\
&\Leftrightarrow & 2f'_P - n_3(P) - 4 \cdot (1-g_P) + 2b_P &= n_1(P)
\end{aligned}$$

■

Corollary 9.2 $n_1(P) + n_3(P)$ *is even.*

Proof: We have

$$\begin{aligned}
& & n_1(P) &= 2f'_P + 2b_P - n_3(P) - 4 \cdot (1-g_P) \\
&\Leftrightarrow & n_1(P) + n_3(P) &= 2f'_P + 2b_P - 4 \cdot (1-g_P)
\end{aligned}$$

The right side of the equation is obviously even. ■

Let P be cut along a zig–zag path p of length l. Let n'_1 be the number of inner vertices and let n'_3 be the number of 3–valent boundary vertices of all patches in $A(P,p)$. Then it holds $n'_1 = n_1(P) - (l-1)$ because $l-1$ inner vertices are cut (by an intersection cut as well as by a linkage cut as well as by an enclosure cut). Moreover, it holds $n'_3 = n_3(P) + (l-3)$: If p is an intersection path or a linkage path, then each of the $l-1$ inner vertices is replaced by one 3–valent and one 2–valent boundary vertex, and two 3–valent boundary vertices (the start vertex and the end vertex of p) are replaced by two 2–valent boundary vertices each. If p is an enclosure path, then each of the $l-1$ inner vertices except the end vertex of p is replaced by a 3–valent and a 2–valent boundary vertex, one 3–valent boundary vertex (the start vertex

[1] As in Section 8.7, we consider only *possibly canonical* patches in order to secure that the degree of each inner face is in D. The proofs can easily be rewritten for arbitrary patches.

of p) is replaced by two 2–valent boundary vertices, and one (3–valent) inner vertex (the end vertex of p) is replaced by three 2–valent boundary vertices.

Thus it holds $n'_1 + n'_3 = n_1(P) + n_3(P) - 2$, and hence Corollary 9.2 implies that $n'_1 + n'_3$ is even. Of course, if $|A(P,p)| = 1$, then the new values n'_1 and n'_3 are consistent with Lemma 9.1: Let $A(P,p) = \{Q\}$. If p is an intersection or enclosure path, then $f'_Q = f'_P$, $g_Q = g_P - 1$, $b_Q = b_P + 1$. If p is a linkage path, then $f'_Q = f'_P$, $g_Q = g_P$, $b_Q = b_P - 1$. In either case, it holds $2b_Q - 4 \cdot (1 - g_Q) = 2b_P - 4 \cdot (1 - g_P) - 2$ and hence

$$\begin{aligned}
n'_1 &= n_1(P) - (l-1)\\
&= 2f'_P + 2b_P - 4 \cdot (1 - g_P) - n_3(P) - (l-1)\\
&= 2f'_Q + 2b_Q - 4 \cdot (1 - g_Q) + 2 - n_3(P) - (l-1)\\
&= 2f'_Q + 2b_Q - 4 \cdot (1 - g_Q) + 2 - (n'_3 - (l-3)) - (l-1)\\
&= 2f'_Q + 2b_Q - 4 \cdot (1 - g_Q) - n'_3
\end{aligned}$$

Let $|A(P,p)| = 2$ with $A(P,p) = \{Q_1, Q_2\}$. Then $n_3(Q_1)$ and $n_3(Q_2)$ can be derived if the boundary codes of Q_1 and Q_2 are known. $n_1(Q_1)$ can be computed using Lemma 9.1 if $n_3(Q_1)$, g_{Q_1}, f'_{Q_1}, and b_{Q_1} are known. Similarly, $n_1(Q_2)$ can be computed if $n_3(Q_2)$, g_{Q_2}, f'_{Q_2}, and b_{Q_2} are known. However, if $n_1(Q_1)$ is already known, then $n_1(Q_2)$ can be computed more easily using the formula

$$n_1(P) = n_1(Q_1) + n_1(Q_2) + (l-1)$$

instead of Lemma 9.1. This formula holds because $n_1(Q_1)+n_1(Q_2) = n'_1$. If $n_1(Q_2)$ is computed this way, then it can easily be verified that the value is consistent with the one provided by Lemma 9.1. As an example, we will show this for an intersection cut of length l: It holds $n_3(Q_1) + n_3(Q_2) = n_3(P) + (l-3)$ and hence $n_3(P) - n_3(Q_1) = n_3(Q_2) - (l-3)$. Moreover, $g_P = g_{Q_1} + g_{Q_2}$, $f'_P = f'_{Q_1} + f'_{Q_2}$, $b_P = b_{Q_1} + b_{Q_2} - 1$, and $n_1(Q_1) + n_1(Q_2) = n_1(P) - (l-1)$ imply

$$\begin{aligned}
n_1(Q_2) &= n_1(P) - n_1(Q_1) - (l-1)\\
&= 2f'_P + 2b_P - n_3(P) - 4 \cdot (1 - g_P) - \big(2f'_{Q_1} + 2b_{Q_1} - n_3(Q_1) - 4 \cdot (1 - g_{Q_1})\big) - (l-1)\\
&= 2f'_{Q_2} + 2 \cdot (b_{Q_2} - 1) - \big(n_3(P) - n_3(Q_1)\big) + 4g_{Q_2} - (l-1)\\
&= 2f'_{Q_2} + 2b_{Q_2} - 2 - \big(n_3(Q_2) - (l-3)\big) + 4g_{Q_2} - l + 1\\
&= 2f'_{Q_2} + 2b_{Q_2} - n_3(Q_2) - 4 \cdot (1 - g_{Q_2})
\end{aligned}$$

Analogously, the number of inner vertices can be computed for the patches that are obtained by cutting a cubic map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ along a Petrie path p of length l: Let n' be the sum of inner vertices of all patches in $A(M,p)$. Then $n' = n - l$ if p is a Jordan curve path and $n' = n - l + 1$ if p is a sandwich path or a dumbbell path. If $A(M,p) = \{P\}$, then the number of inner vertices of P is n'. If $|A(M,p)| = 2$ with $A(M,p) = \{P_1, P_2\}$ and $n_1(P_1)$ is known ($n_1(P_1)$ can be computed using Lemma 9.1), then $n_1(P_2)$ can be computed using the formula

$$n_1(P_2) = n' - n_1(P_1).$$

If $|A(M,p)| = 3$ with $A(M,p) = \{P_1, P_2, P_3\}$ and $n_1(P_1)$ and $n_1(P_2)$ are known (using Lemma 9.1), then $n_1(P_3)$ can be computed using the formula

$$n_1(P_3) = n' - n_1(P_1) - n_1(P_2).$$

It can be verified for all cases that the computed values are consistent with Lemma 9.1.

Here is why we are interested in computing the number of inner vertices of a patch as fast as possible: Lemma 9.1 can be used to compute f'_P for a patch P out of $n_1(P)$, $n_3(P)$, b_P, and g_P. During the top–down approach, we determine possible parameters for the ancestors of a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$ or a patch P. Consider a map $M \in \mathcal{M}(g,n,D,b_l,b_u)$. If we first determine the Petrie path p which is used for the cut, then for every patch $P \in A(M,p)$, the boundary code and thus b_P and $n_3(P)$ can already be computed. As soon as we assign genera to the patches in the next step, Lemma 9.1 yields a fixed relation between $n_1(P)$ and f'_P for every patch $P \in A(M,p)$. Hence at this point, we can either partition f $(= f_M)$ or the sum n' of inner vertices of the patches in $A(M,p)$. Analogously, for a patch P, we first determine the type, the length, and the end vertex of the zig–zag path p, then we assign genera to the patches in $A(P,p)$, and then we can either partition f'_P or the sum n' of inner vertices of the patches in $A(P,p)$. For both M and P, although partitioning f_M or f'_P appears to be more intuitive, we choose partitioning n', for the following reason: Let Q be an ancestor of a map $M \in \mathcal{M}(g,n,D,l,u)$ or of a patch P. Sometimes it happens that when we assign a number f'_Q to a patch Q, then Lemma 9.1 yields a negative value for $n_1(Q)$. Similarly, when we assign a number $n_1(Q)$ to a patch, then Lemma 9.1 sometimes yields $f'_Q < 1$. In both cases, the assignment is invalid (i.e. the whole partition of f, f'_P, or n', respectively, is invalid) and must be corrected. However, in the second case the probability of an invalid assignment is smaller than in the first case: Lemma 9.1 yields $f'_Q = \frac{1}{2}(n_1(Q)+n_3(Q))+2-b_Q-2g_Q$. If $n_1(Q)$ is fixed, then b_Q+2g_Q must be bigger than $\frac{1}{2}(n_1(Q)+n_3(Q))+1$ in order to obtain an invalid value $f_Q < 1$. In particular, if $g=0$, then $g_Q=0$ and $b_Q=1$ and hence $b_Q+2g_Q = 1 \leq \frac{1}{2}(n_1(Q)+n_3(Q))+1$ for every possibly canonical patch Q, and hence all assignments to $n_1(Q)$ are valid. On the other hand, Lemma 9.1 yields $n_1(Q) = 2f'_Q+2b_Q-n_3(Q)-4\cdot(1-g_Q)$. So if f'_Q is rather small, then $2f'_Q+2b_Q+4g_Q-4 < n_3(Q)$ is possible and thus $n_1(Q) < 0$ is possible. In particular, if $g=0$, then $2b_Q+4g_Q-4=-2$ and hence $f'_Q < \frac{1}{2}(n_3(Q)+2)$ implies $n_1(Q) < 0$. Thus many verifications and corrections must be performed.

So the assignment of numbers of inner vertices to the patches yields a faster algorithm than the assignment of numbers of inner faces. Of course, an assignment of f'_Q to a patch Q is invalid also if Lemma 9.1 yields a value $f'_Q > f$ or $f'_Q > f'_P$, respectively (depending on whether Q is an ancestor of M or P). If Q is not the only patch in $A(M,p)$ or $A(P,p)$, respectively, then even $f'_Q \geq f$ or $f'_Q \geq f'_P$, respectively, indicates an invalid assignment. However, the following statements can easily be verified:

- If n' inner vertices are partitioned and assigned to the patches of $A(M,p) = \{Q_1, \ldots, Q_j\}$ (with $j = |A(M,p)| \in \{1,2,3\}$), then $\sum_{i=1}^{j} f'_{Q_i} = f$ where f'_{Q_i} is obtained by Lemma 9.1 for every i. Thus, $f'_{Q_1} > f$ is impossible if $j=1$, and for $j>1$ it holds that if $f'_{Q_i} \geq f$ for some i, then $f'_{Q_{i'}} \leq 0$ for some other i' (so that the invalid assignment can be detected by the standard verification $f_{Q_{i'}} < 1$). Moreover, if $g=0$, then no invalid assignment is possible.
- Analogously, it holds that if $n_1(P)-(l-1)$ inner vertices are partitioned and assigned to the patches of $A(P,p) = \{Q_1, \ldots, Q_j\}$ (with $j = |A(P,p)| \in \{1,2\}$), then $\sum_{i=1}^{j} f'_{Q_i} = f'_P$ where f'_{Q_i} is obtained by Lemma 9.1 for every i. Thus, $f'_{Q_1} > f'_P$ is impossible if $j=1$, and for $j=2$ it holds that if $f'_{Q_i} \geq f'_P$ for some i, then $f'_{Q_{3-i}} \leq 0$. Moreover, if $g=0$, then no invalid assignment is possible.

Thus the assignment of numbers of inner vertices is consistent to the assignment of numbers of inner faces to the patches, and f'_{Q_j} can also be obtained by one of the formulæ

$$f'_{Q_j} = f - \sum_{i=1}^{j-1} f'_{Q_i} \qquad \text{or} \qquad f'_{Q_j} = f'_P - \sum_{i=1}^{j-1} f'_{Q_i}.$$

As an example, we will verify the statement for an intersection cut p of length l in a patch P with $|A(P,p)| = 2$ and $A(P,p) = \{Q_1, Q_2\}$: It holds $n_3(P) + (l-3) = n_3(Q_1) + n_3(Q_2)$ and $n_1(P) - (l-1) = n_1(Q_1) + n_1(Q_2)$. Moreover, $g_P = g_{Q_1} + g_{Q_2}$, $f'_P = f'_{Q_1} + f'_{Q_2}$, $b_P = b_{Q_1} + b_{Q_2} - 1$, and Lemma 9.1 imply

$$\begin{aligned}
f'_{Q_2} &= \frac{n_1(Q_2) + n_3(Q_2)}{2} + 2 - b_{Q_2} - 2g_{Q_2} \\
&= \frac{n_1(P) - (l-1) - n_1(Q_1) + n_3(P) + (l-3) - n_3(Q_1)}{2} + 2 - (b_P - b_{Q_1} + 1) - 2g_{Q_2} \\
&= \frac{n_1(P) + n_3(P)}{2} - \frac{n_1(Q_1) + n_3(Q_1)}{2} - 1 + 2 - b_P + b_{Q_1} - 1 - 2 \cdot (g_P - g_{Q_1}) \\
&= f'_P - f'_{Q_1}
\end{aligned}$$

9.3 Reconstructing the first cut in a cubic map

In this section, we give details how *every* possible first cut in *every* map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is reconstructed. In order not to miss any possible first cut, every possibility must be considered (one after the other) wherever there is more than one possibility to assume some value (e.g. the genus or the number of inner vertices of a patch). For the reconstruction, we first choose the type of the Petrie path p. Here we must take 3 possibilities into consideration. They are discussed in the following sections. In order to keep the following text simple, the process of trying to reconstruct the canonical cut sequence of a patch will just be called "searching" (a patch), and after the canonical cut sequence of a patch has been completely reconstructed, the patch is "found". Let μ be a multipatch and let P be an implicit patch of μ. Then during the reconstruction, we can in most cases only make assumptions about P. The patch P is rather "virtual". As soon as a canonical cut sequence has been completely reconstructed, we obtain a concrete patch P. Similarly, throughout the following text, the symbol M does not denote a special map in $\mathcal{M}(g, n, D, b_l, b_u)$. It represents every map in $\mathcal{M}(g, n, D, b_l, b_u)$. The map M is "virtual". As soon as a canonical cut sequence has been completely reconstructed, we obtain a concrete map $M \in \mathcal{M}(g, n, D, b_l, b_u)$.

9.3.1 Jordan curve paths

First, we choose $|A(M,p)| \in \{1, 2\}$.

(1) Assume $|A(M,p)| = 1$ with $A(M,p) = \{P\}$. Due to Section 6.4.1, this is possible only if $g \geq 1$. Next, we choose the length l of p. The length l must always be even. If multiple edges are allowed, then l may vary from 2 to n, otherwise l may vary from 4 to n. The patch P has the boundary code $(2, \langle 0, 0\rangle, \langle \frac{l}{2}, \frac{l}{2}\rangle)$. It holds $n_3(P) = l$. The genus is unique: $g_P = g - 1$. The number of inner vertices is unique, too: $n_1(P) = n - l$. Now patches are searched that match the properties of P, i.e. the implicit patches of the multipatch $\mu = ((f, g-1, 2), (2, \langle 0, 0\rangle, \langle \frac{l}{2}, \frac{l}{2}\rangle))$ are searched (details will be given in Section 9.4).

(2) Assume $|A(M,p)| = 2$ with $A(M,p) = \{P_1, P_2\}$. This is possible only if $f \geq 2$. Moreover, if $d_{\min}$ $(= \min D) \geq 6$, then it must hold $g \geq 2$ because due to Theorem 8.7, it holds both $g_{P_1} \geq 1$ and $g_{P_2} \geq 1$.

Next, we choose the length l of the Jordan curve path p. The length l must always be even. If multiple edges are allowed, then it must hold $l \geq 2$, otherwise it must hold $l \geq 4$. If n is a multiple of 4, then $l \leq n$ must be satisfied, otherwise it must even hold $l \leq n-2$: Either of the two patches P_1 and P_2 has the boundary code $(1, \langle 0 \rangle, \langle \frac{l}{2} \rangle)$. Thus it holds $n_3(P_1) = n_3(P_2) = \frac{l}{2}$. So if $l = n$ and n is no multiple of 4, then both $n_3(P_1)$ and $n_3(P_2)$ are odd. Then due to Corollary 9.2, $n_1(P_1)$ and $n_1(P_2)$ must be odd, too. Thus $n_1(P_1)+n_1(P_2) \geq 2$. But $n_1(P_1)+n_1(P_2) = n-l$, so $l = n$ is impossible if n is no multiple of 4.

Next, we assign the genera to the patches. If $d_{\min} < 6$, then g_{P_1} may vary from 0 to g, otherwise g_{P_1} may vary from 1 to $g-1$ (because $g_{P_2} \geq 1$). Then it holds $g_{P_2} = g - g_{P_1}$ due to Section 6.4.1.

Next, we distribute $n-l$ inner vertices to the patches. $n_1(P_1)$ may vary from 0 to $n-l$ where $n_1(P_1)$ must be odd if and only if $n_3(P_1)$ is odd. Then $n_1(P_2)$ is implied by $n_1(P_2) = n - l - n_1(P_1)$.

Now P_1 is searched, and whenever an appropriate patch is found, then P_2 is searched. Here the face vector of P_1 must be considered, i.e. P_2 may contain at most $b_u(d) - p_d(P_1)$ inner faces of degree d for any $d \in D$. The lower limit b_l is not considered until a patch matching the properties of P_2 is found.

As soon as 1 or 2 appropriate patches (in the first or second case, respectively) are found, they must be constructed and agglutinated along p. Here every test must be recovered that depends on the complete incidence structure of a patch or of the resulting cubic map M, that is:

- The patches are constructed in a bottom–up manner starting with the tiles. After each step, it must be tested if this step really was the inverse of a canonical cut. See Section 9.5 and 8.3 for details.
- After the patches have been agglutinated along p, it must be tested whether p is a canonical Petrie path in M. Moreover, the tests described in Section 8.5 must be performed. This is done also during the construction of the patches.

In case 2, similar to the bottom–up approach (see Section 8.6.1), it can be prescribed that $s(P_1) \leq s(P_2)$ or, alternatively, that $(g_{P_1}, n_1(P_1)) \leq (g_{P_2}, n_1(P_2))$ holds (i.e. the first pair is lexicographically smaller than the second pair). The latter possibility is easier to manage because all required values are already known. If this prescription is done, then the two Petrie paths p and p' starting with h_0 and h'_0 (see Figure 6.7 on page 78), respectively, must be compared, and the path having the higher quality is taken as the *candidate* for the canonicity check. If $(g_{P_1}, n_1(P_1)) = (g_{P_2}, n_1(P_2))$, then also $s(P_1) = s(P_2)$ (due to Lemma 9.1). In this case, it must be secured explicitly that exactly one pair (P_1, P_2) or (P_2, P_1) is taken for the construction. See Section 9.5.

9.3.2 Dumbbell paths

For a dumbbell path p, we first choose the lengths $l_1(p)$, $l_2(p)$, and $l_3(p)$. It must hold $l_1(p) \geq 1$, $l_2(p) \geq 1$, $l_3(p) \geq 1$, and $l_1(p)+l_2(p)+l_3(p) \leq n+1$. If $l_1(p) = 1$, then the start side boundary is a monogon, and if $l_1(p) = 2$, then the start side boundary is a digon. Analogously, if $l_3(p) = 1$, then the end side boundary is a monogon, and if $l_3(p) = 2$, then the end side boundary is a digon. Hence if loops and multiple edges are not allowed, then it must hold $l_1(p) \geq 3$ and analogously $l_3(p) \geq 3$. If p is canonical, then $l_1(p) \leq l_3(p)$, and hence we consider this restriction when choosing $l_1(p)$ and $l_3(p)$. Thus we choose $l_1(p)$, $l_3(p)$, and $l_2(p)$ in exactly this order where

- $l_1(p)$ may vary between $\min\{d_{\min}, 3\}$ and $\frac{n}{2}$,
- $l_3(p)$ may vary between $l_1(p)$ and $n - l_1(p)$, and
- $l_2(p)$ may vary between 1 and $n + 1 - l_1(p) - l_3(p)$.

Then the patches contain altogether $n' := n + 1 - l_1(p) - l_2(p) - l_3(p)$ inner vertices. The canonical boundary codes of the three boundary faces can be computed using $l_1(p)$, $l_2(p)$, and $l_3(p)$ (see Section 8.6.2).

Next, we choose $|A(M,p)| \in \{1,2,3\}$. The cases are treated as follows:

(1) $|A(M,p)| = 1$ is possible only if $g \geq 2$ and if every boundary face contains at least one 3–valent boundary vertex (because a boundary face without 3–valent boundary vertices is the boundary face of a tile, but a tile has only one boundary face). Let $A(M,p) = \{P\}$. Then $g_P = g - 2$ and $n_1(P) = n'$. The boundary code of P is uniquely derived from the canonical boundary codes of the 3 boundary faces of P. Now patches are searched that match the properties of P. If $l_1(p) = l_3(p)$, then as explained in Section 8.6.2, two agglutinations must be performed.

(2) $|A(M,p)| = 2$ is possible only if $g \geq 1$ and $f \geq 2$. Let $A(M,p) = \{P_1, P_2\}$ and let $b_{P_1} = 1$ and $b_{P_2} = 2$. If $d_{\min} \geq 6$, then $|A(M,p)| = 2$ is possible only if $g \geq 2$ because $b_{P_1} = 1$ implies $g_{P_1} \geq 1$ due to Theorem 8.7. There are 3 subcases:
  - P_1 contains the start side boundary.
  - P_1 contains the end side boundary.
  - P_1 contains the main boundary.

  In all subcases, the boundary codes of P_1 and P_2 and hence $n_3(P_1)$ and $n_3(P_2)$ can be derived from the canonical boundary codes of the boundary faces. Every boundary face of P_2 must contain at least one 3–valent boundary vertex, otherwise the corresponding subcase is not possible.

  Now we choose the genera of the patches. If the boundary face of P_1 does not contain a 3–valent boundary vertex, then P_1 must be a tile and hence $g_{P_1} = 0$. If P_1 is not a tile, then g_{P_1} may vary as follows: If $d_{\min} < 6$, then g_{P_1} may vary between 0 and $g - 1$, otherwise g_{P_1} may vary between 1 and $g - 1$. In either case, we assign $g_{P_2} = g - 1 - g_{P_1}$. Next, we partition n': $n_1(P_1)$ must be even if and only if $n_3(P_1)$ is even. $n_1(P_2)$ must be even if and only if $n_3(P_2)$ is even. If P_1 is a tile, then it must hold $n_1(P_1) = 0$. Otherwise $n_1(P_1)$ may vary between 0 and n' if $n_1(P_2)$ must be even and between 0 and $n' - 1$ if $n_1(P_2)$ must be odd (because then $n_1(P_2) \geq 1$). In either case, $n_1(P_2)$ follows from the formula $n_1(P_2) = n' - n_1(P_1)$.

  Now P_1 is searched, and whenever an appropriate patch is found, P_2 is searched. Here the face vector of P_1 must be considered in the same way as in the Jordan curve case. For the subcases, there are the following additional rules:
  - If P_1 contains the main boundary and $l_1(p) = l_3(p)$, then two different agglutinations can be performed. If the marked vertices in P_2 are not mark–isomorphic, then both agglutinations *must* be performed, otherwise only one agglutination may be performed (compare Section 8.6.2).
  - If P_1 contains a side boundary, then the agglutination is always unique. However, if $l_1(p) = l_3(p)$, then every agglutination where P_1 contributes the start side boundary

corresponds to an agglutination where P_1 contributes the end side boundary. The correspondence is given by inversing p. Hence if P_1 contains a side boundary and $l_1(p) = l_3(p)$, then one can prescribe that P_1 contains always the start side boundary. Then after an agglutination, the path having the higher quality out of p and $\overline{p}$ must be taken as the candidate for the canonicity check.

(3) $|A(M,p)| = 3$ is possible only if $f \geq 3$. If $d_{\min} \geq 6$, then $|A(M,p)| = 3$ is possible only if $g \geq 3$ due to Theorem 8.7. Let $A(M,p) = \{P_1, P_2, P_3\}$. The values $n_3(P_1)$, $n_3(P_2)$, and $n_3(P_3)$ can be derived from the canonical boundary codes of the boundary faces. Let P_3 be the patch containing the main boundary. We choose the genera of the patches. In principle, g_{P_2} may vary between 0 and g, g_{P_1} may vary between 0 and $g - g_{P_2}$, and then $g_{P_3} = g - g_{P_1} - g_{P_2}$. However, if $d_{\min} \geq 6$, then g_{P_2} may vary only between 1 and $g-2$, g_{P_1} may vary between 1 and $g - g_{P_2} - 1$, and then $g_{P_3} = g - g_{P_1} - g_{P_2}$. Moreover, for $i \in \{1,2,3\}$ it holds that if P_i does not contain any 3–valent boundary vertex, then P_i must be a tile and hence $g_{P_i} = 0$.

Next, we partition n'. As usual, $n_1(P_i)$ must be even if and only if $n_3(P_i)$ is even ($i \in \{1,2,3\}$). Apart from that, $n_1(P_2)$ may vary between 0 and n', $n_1(P_1)$ may vary between 0 and $n' - n_1(P_2)$, and then $n_1(P_3) = n' - n_1(P_1) - n_1(P_2)$. However, if P_i ($i \in \{1,2,3\}$) is a tile, then it must hold $n_1(P_i) = 0$.

Now P_1 is searched. Whenever an appropriate patch is found, P_2 is searched, and for every patch which is found P_3 is searched. Here the face vectors of P_1 and P_2 must be considered in the same way as in the Jordan curve case. Similar to the bottom–up approach (see Section 8.6.2), it holds that if $l_1(p) = l_3(p)$, then it can be prescribed that $(g_{P_1}, n_1(P_1)) \leq (g_{P_2}, n_1(P_2))$. Then, if $(g_{P_1}, n_1(P_1)) \neq (g_{P_2}, n_1(P_2))$, the path having the higher quality out of p and $\overline{p}$ is the candidate for the canonicity check.

As soon as all required patches are found, they must be constructed and agglutinated along p. As in the Jordan curve case, every test must be performed that depends on the incidence structure of the resulting map M.

9.3.3 Sandwich paths

For a sandwich path p, we first choose the lengths $l_1(p)$, $l_2(p)$, and $l_3(p)$. It must hold $l_1(p) \geq 1$, $l_2(p) \geq 1$, $l_3(p) \geq 1$, and $l_1(p) + l_2(p) + l_3(p) \leq n+1$. If $l_1(p) + l_2(p) = 2$, then the start side boundary is a digon. If $l_2(p) + l_3(p) = 2$, then the end side boundary is a digon. And if $l_1(p) + l_3(p) = 2$, then the main boundary is a digon. Hence if multiple edges are not allowed, then at least two of the three parts of p must have a length greater than 1. Moreover, if p is canonical, then $l_1(p) \leq l_3(p)$, and hence we consider this restriction when choosing $l_1(p)$ and $l_3(p)$. Thus we choose $l_1(p)$, $l_3(p)$, and $l_2(p)$ in exactly this order where

- $l_1(p)$ may vary between 1 and $\frac{n}{2}$,
- $l_3(p)$ may vary between $l_1(p)$ and $n - l_1(p)$ (where $l_3(p) = 1$ is not allowed if $l_1(p) = 1$ and multiple edges are not allowed), and
- $l_2(p)$ may vary between 1 and $n+1-l_1(p)-l_3(p)$ (where $l_2(p) = 1$ is allowed only if multiple edges are allowed or it holds both $l_1(p) > 1$ and $l_3(p) > 1$).

Then the patches contain altogether $n' := n + 1 - l_1(p) - l_2(p) - l_3(p)$ inner vertices. The canonical boundary codes of the boundary faces can be derived from $l_1(p)$, $l_2(p)$, and $l_3(p)$ (see Section 8.6.3).

If $l_2(p)$ is odd, then the procedure is in principle the same as for the dumbbell case. However, there are the following differences:

- If $|A(M,p)| = 1$ (with $A(M,p) = \{P\}$), then it is not only possible that the canonical boundary codes of the two side boundaries are identical (if $l_1(p) = l_3(p)$), but it is also possible that the canonical boundary code of the main boundary is identical with the canonical boundary code of the start side boundary (see case 2 of Table 8.2 on page 134). In this case, it is not clear which boundary face in P is the main boundary, and since the marked vertices contained in the two possible boundary faces are not mark–isomorphic (otherwise the corresponding patch automorphism would fix the marked vertex contained in the third boundary face), both possibilities must be considered. Besides, it is impossible that the canonical boundary code of the main boundary is identical with the canonical boundary code of the end side boundary because then, the canonical boundary code of the start side boundary would be $(2, \langle 0,0 \rangle)$ (see Section 8.6.3). But this is the boundary code of a tile (there are no 3–valent boundary vertices), and P is no tile.
- If $|A(M,p)| = 2$ with $A(M,p) = \{P_1, P_2\}$ where P_2 contains the start side boundary and the main boundary, then as above, both corresponding canonical boundary codes might be identical. In this case, it must hold $l_1(p) = 1$ (see Section 8.6.3). If the two marked boundary vertices in P_2 are not mark–isomorphic, then 2 agglutinations must be performed where the roles of the boundary faces of P_2 are switched after the first agglutination. p does not need to be compared with $\overline{p}$ because p always has a higher quality than $\overline{p}$: It holds $l_1(p) < l_3(p)$ unless $l_1(p) = l_3(p) = 1$, but then the canonical boundary code of the main boundary would be $(2, \langle 0,0 \rangle)$. But this is the boundary code of a tile, and P_2 is no tile.
- If $|A(M,p)| = 2$ with $A(M,p) = \{P_1, P_2\}$ where P_2 contains the end side boundary and the main boundary, then both corresponding canonical boundary codes might be identical. In this case, it must hold $l_1(p) = 1$ and $l_2(p) = 1$, and hence P_1 is a 2–tile. Moreover, $l_3(p)$ is odd. If the two marked boundary vertices in P_2 are not mark–isomorphic, then 2 agglutinations must be performed where the roles of the boundary faces of P_2 are switched after the first agglutination. p does not need to be compared with $\overline{p}$ because $l_1(p) < l_3(p)$ unless $l_1(p) = l_2(p) = l_3(p) = 1$, but then we would have three patches instead of two, namely three 2–tiles.

If $l_2(p)$ is even, then let $A(M,p) = \{P\}$. It holds $g_P = g - 1$ and $n_1(P) = n'$. Now patches are searched that satisfy the properties of P.

9.4 Reconstructing the cut sequence of a patch

In this section, we give details how the canonical cut sequence of *every* implicit patch P of a multipatch $\mu = (s, C)$ is reconstructed. So we know the boundary code C of P, we know g_P, b_P, $n_1(P)$, and thus f'_P (Lemma 9.1). Although the reconstruction of a canonical sequence of cuts considers multipatches rather than patches (because we only consider sizes and boundary codes of patches), we refer to the patches rather than to the corresponding multipatches. This is more convenient. Moreover, this way we can add some statements about how the marked vertices are determined in the (implicit) patches.

In order not to miss any patch, every possibiliy must be considered (one after the other) wherever there is more than one possibility to assume some value. The reconstruction of a cut sequence yields a result “successful” or “not successful”: If the result is “successful”, then a patch with the required properties has been found, otherwise no patch with the required properties has been found.

For the reconstruction, we first check if μ implies a tile, i.e. $g_P = 0$ and $f'_P = 1$. If it does, then we check whether the boundary code C of P matches the boundary code of a permitted tile, i.e. $C = (1, \langle d \rangle, \langle 0 \rangle^d)$ for some $d \in D$. If this is the case and the number of all d–tiles found so far is less than $b_u(d)$, then an implicit patch P of μ is found and the result is "successful", otherwise the result is "not successful". The number of all d–tiles found so far includes all d–tiles that have been found in lower recursion levels, i.e. earlier in the reconstruction.

If μ does not imply a tile, then we must specify the properties of the next cut and search recursively for the implicit patches of the multipatches that are obtained by the cut. The next cut starts at the canonical boundary face. If μ indicates more than one boundary face having the highest quality, then at this point of the reconstruction, they cannot be distinguished in P and exactly one of them is used as the canonical boundary face $F_1 = \langle v_0, h_0, h_0, \ldots, v_{k-1}, h_{k-1}, h'_{k-1} \rangle$ where v_0 be the start vertex of the canonical zig–zag path p (in particular, v_0 is the marked vertex in F_1).

First, we decide how many patches are obtained by performing the next cut. Here 2 possibilities must be taken into consideration.

Assume $|A(P,p)| = 1$.

Next, we choose the type of the next canonical path p. Let $A(P,p) = \{Q\}$.

- An intersection path p is possible only if $g_P \geq 1$. It holds $g_Q = g_P - 1$. Next, we choose the length l of p. The length may vary between 1 and $n_1(P) + 1$. Then it holds $n_1(Q) = n_1(P) - (l-1)$. Next, we choose the end vertex of p. Every 3–valent boundary vertex v_s, $s \in \{1, \ldots, k-1\}$, may be the end vertex of p (in particular, if such a vertex does not exist, then an intersection cut is impossible). Let $C_1 = (c, \langle b_0, \ldots, b_{c'-1} \rangle)$ be the canonical boundary code of F_1 (where $c' = \max\{1, c\}$). The positions of the 3–valent boundary vertices can be derived from the canonical valency sequence of F_1 which on the other hand can be derived from C_1. Thus, with $t := k + l$, the canonical valency sequences of the two new boundary faces F_3 and F_4 in Q can be derived according to Section 6.6.1. From the canonical valency sequences, we derive the boundary codes $C_3 = (c_3, s_3)$ and $C_4 = (c_4, s_4)$ of F_3 and F_4, respectively, and thus the whole boundary code of Q. Every different possibility for the position of the end vertex of p yields different boundary codes C_3 and C_4.

  C_3 and C_4 can be derived from C_1 also without deriving the canonical *valency* sequences: Each entry b_i, $i \in \{0, \ldots, c'-1\}$, represents b_i 3–valent boundary vertices. So every 3–valent boundary vertex of F_1 can be represented by a pair (a, b) where $a \in \{0, \ldots, c'-1\}$ determines the entry of $\langle b_0, \ldots, b_{c'-1} \rangle$ and $b \in \{1, \ldots, b_a\}$ determines the 3–valent boundary vertex in the corresponding part of F_1, i.e. the part that is approached after a convex edges have been passed while tracing F_1 starting with v_0. The pair $(0,1)$ corresponds to the first 3–valent boundary vertex in the first part of F_1, i.e. to v_0. So let a pair $(a,b) \neq (0,1)$ that corresponds to the end vertex of p be given. Then after the intersection cut, all entries b_i, $i \in \{1, \ldots, c'-1\} - \{a\}$, will occur in C_3 and C_4 as shown in the tables below. As usual, the information in the tables has been derived originally by considering the valency sequences of F_1, F_3, and F_4. The boundary codes C_3 and C_4 are not necessarily canonical. In order to obtain the canonical boundary codes of F_3 and F_4, s_3 and s_4 must eventually be cyclically shifted. The vertices that have to be marked in F_3 and F_4 can uniquely be determined due to the marking rules in Section 7.4.

  Now all required properties of Q are specified (i.e. the corresponding multipatch is specified) and it can be searched for Q where the search for P is successful if and only if the search for Q is successful.

c	a	l	C_3
0	0	odd	$(3, \langle 0, \frac{l-1}{2}, b_0 - b\rangle)$
0	0	even	$(4, \langle 0, \frac{l-2}{2}, 0, b_0 - b\rangle)$
> 0	0	odd	$(c+3, \langle 0, \frac{l-1}{2}, b_0 - b, b_1, \nearrow, b_{c-1}, 0\rangle)$
> 0	0	even	$(c+4, \langle 0, \frac{l-2}{2}, 0, b_0 - b, b_1, \nearrow, b_{c-1}, 0\rangle)$
> 0	> 0	odd	$(c-a+3, \langle 0, \frac{l-1}{2}, b_a - b, b_{a+1}, \nearrow, b_{c-1}, 0\rangle)$
> 0	> 0	even	$(c-a+4, \langle 0, \frac{l-2}{2}, 0, b_a - b, b_{a+1}, \nearrow, b_{c-1}, 0\rangle)$

c	a	l	C_4
0	0	odd	$(3, \langle b-2, 0, \frac{l-1}{2}\rangle)$
0	0	even	$(2, \langle b-2, \frac{l}{2}\rangle)$
> 0	0	odd	$(3, \langle b-2, 0, \frac{l-1}{2}\rangle)$
> 0	0	even	$(2, \langle b-2, \frac{l}{2}\rangle)$
> 0	> 0	odd	$(a+3, \langle b_0 - 1, b_1, \nearrow, b_{a-1}, b-1, 0, \frac{l-1}{2}\rangle)$
> 0	> 0	even	$(a+2, \langle b_0 - 1, b_1, \nearrow, b_{a-1}, b-1, \frac{l}{2}\rangle)$

- An enclosure path p is possible only if $g_P \geq 1$. It holds $g_Q = g_P - 1$. Next, we choose the length l of p. The length may vary between (at least) 2 and (at most) $n_1(P) + 1$. Then it holds $n_1(Q) = n_1(P) - (l-1)$. The minimum length is 2 because the enclosure entry length l_e is at least 1 and the enclosure cycle length l_c is at least 1, too. If loops are not allowed, then it must hold $l_c \geq 2$ and hence $l \geq 3$. If neither loops nor multiple edges are allowed, then it must hold $l_c \geq 3$ and hence $l \geq 4$. Next, we choose l_c. The value of l_c may vary between the minimum given above and $l - 1$. Then $l_e = l - l_c$. The boundary codes C_3 and C_4 of the two new boundary faces F_3 and F_4 in Q and hence the whole boundary code of Q can be computed out of l_e, l_c, and the canonical boundary code $C_1 = (c, \langle b_0, \nearrow, b_{c'-1}\rangle)$ of F_1 (where $c' = \max\{1, c\}$):

c	l_e	l_c	C_3	C_4
0	odd	odd	$(5, \langle 0, \frac{l_e+l_c-2}{2}, 0, \frac{l_e-1}{2}, b_0 - 1\rangle)$	$(1, \langle \frac{l_c-1}{2}\rangle)$
0	even	odd	$(5, \langle 0, \frac{l_e-2}{2}, 0, \frac{l_e+l_c-1}{2}, b_0 - 1\rangle)$	$(1, \langle \frac{l_c-1}{2}\rangle)$
0	odd	even	$(4, \langle 0, \frac{l_e+l_c-1}{2}, \frac{l_e-1}{2}, b_0 - 1\rangle)$	$(2, \langle \frac{l_c-2}{2}, 0\rangle)$
0	even	even	$(4, \langle 0, \frac{l_e-2}{2}, \frac{l_e+l_c}{2}, b_0 - 1\rangle)$	$(2, \langle \frac{l_c-2}{2}, 0\rangle)$
> 0	odd	odd	$(c+5, \langle 0, \frac{l_e+l_c-2}{2}, 0, \frac{l_e-1}{2}, b_0 - 1, b_1, \nearrow, b_{c-1}\rangle)$	$(1, \langle \frac{l_c-1}{2}\rangle)$
> 0	even	odd	$(c+5, \langle 0, \frac{l_e-2}{2}, 0, \frac{l_e+l_c-1}{2}, b_0 - 1, b_1, \nearrow, b_{c-1}\rangle)$	$(1, \langle \frac{l_c-1}{2}\rangle)$
> 0	odd	even	$(c+4, \langle 0, \frac{l_e+l_c-1}{2}, \frac{l_e-1}{2}, b_0 - 1, b_1, \nearrow, b_{c-1}\rangle)$	$(2, \langle \frac{l_c-2}{2}, 0\rangle)$
> 0	even	even	$(c+4, \langle 0, \frac{l_e-2}{2}, \frac{l_e+l_c}{2}, b_0 - 1, b_1, \nearrow, b_{c-1}\rangle)$	$(2, \langle \frac{l_c-2}{2}, 0\rangle)$

As in the intersection case, the boundary codes C_3 and C_4 are transformed into canonical boundary codes by cyclically shifting the boundary sequences, and the vertices that have to be marked in F_3 and F_4 can uniquely be determined due to the marking rules in Section 7.4. Now it can be searched for Q where the search for P is successful if and only if the search for Q is successful.

- A linkage path p is possible only if $b_P \geq 2$. It holds $g_Q = g_P$. First, we choose the length l of p. The length may vary between 1 and $n_1(P)+1$. Then it holds $n_1(Q) = n_1(P) - (l-1)$. Next, we choose the end vertex of p. Every 3–valent boundary vertex in every boundary face $F \neq F_1$ of P may be the end vertex of p. However, unlike in the intersection case, some choices yield identical boundary codes for Q, and thus we would obtain the same multipatch several times if we would accept all choices. Thus if there are several boundary faces $F \neq F_1$ having the same canonical boundary code, then we accept only one of them as the boundary face F_2 containing the end vertex of p. And if the boundary face F_2 is periodical, then the end vertex must occur in one certain period of F_2, for example in the first period. Once a boundary face F_2 is chosen, the positions of the 3–valent boundary vertices can be derived from the canonical boundary code of F_2. Moreover, the boundary code C_3 of the new boundary face F_3 in Q and hence the whole boundary code of Q can be computed out of l, the end vertex of p (to be precise: the offset in F_2 between the end vertex of p and the marked vertex in F_2) and the canonical boundary codes of F_1 and F_2. Let $C_1 = (c_1, \langle b_0, \ldots, b_{c'_1-1}\rangle)$ be the canonical boundary code of F_1 (where $c'_1 = \max\{1, c_1\}$), and let $C_2 = (c_2, \langle d_0, \ldots, d_{c'_2-1}\rangle)$ be the canonical boundary code of F_2 (where $c'_2 = \max\{1, c_2\}$). As in the intersection case, let the position of the end vertex of p be represented by the pair (a, b) where $a \in \{0, \ldots, c'_2 - 1\}$ and $b \in \{1, \ldots, d_a\}$. The resulting boundary code C_3 of F_3 is listed in the table below. As above, the boundary code C_3 is transformed into a canonical boundary code by cyclically shifting the boundary sequence, and the vertex that has to be marked in F_3 can uniquely be determined due to the marking rules in Section 7.4. Now all required properties of Q are specified and it can be searched for Q where the search for P is successful if and only if the search for Q is successful.

c_1	c_2	l	C_3
0	0	odd	$(6, \langle 0, \frac{l-1}{2}, d_0-1, 0, \frac{l-1}{2}, b_0-1\rangle)$
0	0	even	$(6, \langle 0, \frac{l-2}{2}, 0, d_0-1, \frac{l}{2}, b_0-1\rangle)$
0	>0	odd	$(c_2+6, \langle 0, \frac{l-1}{2}, d_a-b, d_{a+1}, \ldots, d_{c_2-1}, d_0, \ldots, d_{a-1}, b-1, 0, \frac{l-1}{2}, b_0-1\rangle)$
0	>0	even	$(c_2+6, \langle 0, \frac{l-2}{2}, 0, d_a-b, d_{a+1}, \ldots, d_{c_2-1}, d_0, \ldots, d_{a-1}, b-1, \frac{l}{2}, b_0-1\rangle)$
>0	0	odd	$(c_1+6, \langle 0, \frac{l-1}{2}, d_0-1, 0, \frac{l-1}{2}, b_0-1, b_1, \ldots, b_{c_1-1}, 0\rangle)$
>0	0	even	$(c_1+6, \langle 0, \frac{l-2}{2}, 0, d_0-1, \frac{l}{2}, b_0-1, b_1, \ldots, b_{c_1-1}, 0\rangle)$
>0	>0	odd	$(c_1+c_2+6, \langle 0, \frac{l-1}{2}, d_a-b, d_{a+1}, \ldots, d_{c_2-1}, d_0, \ldots, d_{a-1}, b-1, 0, \frac{l-1}{2},$ $b_0-1, b_1, \ldots, b_{c_1-1}, 0\rangle)$
>0	>0	even	$(c_1+c_2+6, \langle 0, \frac{l-2}{2}, 0, d_a-b, d_{a+1}, \ldots, d_{c_2-1}, d_0, \ldots, d_{a-1}, b-1, \frac{l}{2},$ $b_0-1, b_1, \ldots, b_{c_1-1}, 0\rangle)$

In either case (intersection, enclosure, linkage), it must be tested if the resulting patch Q contains a boundary face whose canonical boundary code corresponds to the canonical boundary code of a tile. If it does, then the cut is invalid if $g_Q \neq 0$, $n_1(Q) \neq 0$, or $b_Q > 1$ (if $b_Q > 1$, then every boundary face must contain a 3–valent boundary vertex because otherwise Q would be disconnected).

Let now $|A(P,p)| = 2$. This is possible only if $f'_{P} \geq 2$. Let $A(P,p) = \{Q_1, Q_2\}$. First, we choose the type of the next canonical path p.

- Let p be an intersection path. We choose the length l of p. The length may vary between 1 and $n_1(P)+1$. Then it holds $n_1(Q_1)+n_1(Q_2) = n_1(P)-(l-1) =: n'$. Next, we choose the end vertex of p. Every 3–valent boundary vertex $u \neq v_0$ contained in F_1 may be the end vertex of p. The positions of the 3–valent boundary vertices in F_1, the canonical boundary codes of the new boundary faces F_3 (in Q_1) and F_4 (in Q_2), and the marked vertices contained in F_3 and F_4 can be determined as in the case $|A(P,p)| = 1$. Next, we specify which of the unmodified boundary faces of P will be contained in Q_1 and which boundary faces will be contained in Q_2. P contains $x := b_{P} - 1$ boundary faces besides F_1, so in principle there are 2^x possibilities to partition the unmodified boundary faces of P. However, if there are two boundary faces F and F' having identical canonical boundary codes, then at this level of the reconstruction, it makes no difference whether F is assigned to Q_1 and F' is assigned to Q_2 or vice versa. Hence only one of the two combinations is considered. In general, all combinations are considered that yield pairwise different boundary codes for Q_1 (and thus pairwise different boundary codes for Q_2). Next, we partition g_{P}. Here g_{Q_1} may vary between 0 and g_{P}, and then it holds $g_{Q_2} = g_{P} - g_{Q_1}$. Finally, we partition n'. Here, in principle, $n_1(Q_1)$ may vary between 0 and n', and then it follows $n_1(Q_2) = n' - n_1(Q_1)$. However, for both $n_1(Q_1)$ and $n_1(Q_2)$, Lemma 9.1 eventually yields a more stringent lower bound by the side conditions $f_{Q_1} \geq 1$ and $f_{Q_2} \geq 1$. As soon as $n_1(Q_1)$ and $n_1(Q_2)$ are fixed, all required properties of Q_1 are specified and it can be searched for Q_1. If an appropriate patch is found, then it is searched for Q_2. The search for P is successful if and only if the search for both Q_1 and Q_2 is successful.
- Let p be an enclosure path. First, we choose the length l of p. The length may vary between (at least) 2 and (at most) $n_1(P)+1$. Then it holds $n_1(Q_1)+n_1(Q_2) = n_1(P)-(l-1) =: n'$. If loops or multiple edges are not allowed, then the minimum for the length l_c of the enclosure cycle and hence the minimum for l is further restricted as in the case $|A(P,p)| = 1$. Next, we choose l_c. The value of l_c may vary between the minimum given above and $l-1$. The canonical boundary codes of the new boundary faces F_3 (in Q_1) and F_4 (in Q_2) and the marked vertices in F_3 and F_4 can be determined as in the case $|A(P,p)| = 1$. The rest is equal to the intersection case.

In either case, it must be tested for $i \in \{1,2\}$ if the resulting patch Q_i contains a boundary face whose canonical boundary code corresponds to the canonical boundary code of a tile. If it does, then as above, the cut is invalid if $g_{Q_i} \neq 0$, $n_1(Q_i) \neq 0$, or $b_{Q_i} > 1$.

When a patch P of a given size (f'_{P}, g_{P}, b_{P}) and with a given boundary code is searched, then it can sometimes be told in advance that an appropriate patch does not exist. For that, we perform the following computations right at the beginning of the process (before reconstructing any cut sequence): Let $D = \{d_1, \ldots, d_{n'}\}$ (where $n' = |D|$) such that $d_i < d_j$ if and only if $i < j$ for every $i, j \in \{1, \ldots, n'\}$.

- First, as described in Section 6.1, all possible face vectors of maps in $\mathcal{M}(g, n, D, b_l, b_u)$ are computed.
- For every such face vector $s = \langle f_1, \ldots, f_{n'} \rangle$, every sequence $\langle f'_1, \ldots, f'_{n'} \rangle$ with $f'_i \in \{0, \ldots, f_i\}$ for every $i \in \{1, \ldots, n'\}$ is computed. Thus, all possible face vectors of the canonical patches are computed. Let $\langle f'_1, \ldots, f'_{n'} \rangle$ with $f' := \sum_{i=1}^{n'} f'_i$ be the face vector of a canonical patch Q whose size is known (so that $f' = f'_{Q}$). Then the number k_{Q} of convex edges in Q can be derived using Lemma 8.10. So if $\langle f'_1, \ldots, f'_{n'} \rangle$ with $f' := \sum_{i=1}^{n'} f'_i$ is a *possible* face vector of

a canonical patch of size (f', g', b'), then with Lemma 8.10,

$$k = 12 \cdot (1 - g') - 6b' - \sum_{i=1}^{n'} f'_i \cdot (6 - d_i)$$

is a *possible* number of convex edges in a canonical patch of size (f', g', b').

Thus all possible numbers of convex edges in canonical patches depending on their sizes are known in advance. During the reconstruction of canonical cut sequences, whenever a patch P is searched, the size of P as well as the boundary code of P are determined explicitly. So the number k_P of convex edges in P can easily be derived applying Lemma 8.10, and it can be verified whether k_P is a possible number of convex edges in a canonical patch of size $s(P)$ at all. If not, then it does not need to be searched for appropriate patches P. This consideration is helpful for example if all permitted face degrees are multiples of 2 or 3, respectively. In these cases, due to Lemma 6.17, the number of convex edges in every canonical patch is also a multiple of 2 or 3, respectively (compare the proof of Theorem 9.6), so that there are only few different possible numbers of convex edges.

In fact, while the possible numbers of convex edges in a canonical patch P are determined and stored, it suffices to store them for the case $g_P = 0$ and $b_P = 1$: Let Q be a patch having the same face vector as P (so that $f'_Q = f'_P$). Then due to Lemma 8.10, it holds

$$\begin{aligned}
& 12 \cdot (1 - g_Q) - 6b_Q - k_Q = fd(Q) = fd(P) = 12 \cdot (1 - g_P) - 6b_P - k_P \\
\Leftrightarrow \quad & k_P = 12 \cdot (1 - g_P) - 12 \cdot (1 - g_Q) - 6b_P + 6b_Q + k_Q \\
\Leftrightarrow \quad & k_P = 12 \cdot (g_Q - g_P) - 6 \cdot (b_P - b_Q) + k_Q
\end{aligned}$$

so that with $g_P = 0$ and $b_P = 1$, it holds $k_P = k_Q + 12g_Q + 6 \cdot (b_Q - 1)$. Hence if and only if k is a possible number of convex edges in a patch Q of size (f'_Q, g_Q, b_Q), then $k + 12g_Q + 6 \cdot (b_Q - 1)$ is a possible critical sum of a patch P of size $(f'_Q, 0, 1)$. Hence it suffices to store the possible numbers of convex edges for patches of genus 0 having one boundary face, and the possible numbers for patches having a higher genus and/or more than one boundary face can be derived from the earlier ones.

9.5 The complete reconstruction

The reconstructed cut sequence of a patch P can be represented by a tree as in Figure 6.21 on page 98. The nodes of this tree correspond to the patches that occur in the canonical cut sequence of P. The leaves correspond to the tiles. The inner nodes of the tree (the nodes that are no leaves) are either 2–valent or 3–valent depending on whether the corresponding patch has 1 or 2 canonical ancestors. When a patch has 2 canonical ancestors, then the canonical cut sequence of the first ancestor will be completely reconstructed before the reconstruction of the canonical cut sequence of the second ancestor begins. The same holds for the patches that are agglutinated along a Petrie path: If more than one patch is agglutinated, then the canonical cut sequence of the first patch is completely reconstructed before the reconstruction of the canonical cut sequence of the second patch begins. So in principle, we obtain *one* tree representing the cut sequences of all patches that are agglutinated along a Petrie path. Figure 9.4 shows an example where two patches are agglutinated along a Jordan curve path. The numbers in the nodes of the tree indicate the order in which the corresponding patches are reconstructed (according to Section 1.9, the tree is drawn such that the tree–source is at the bottom of the figure; the tree–source having the number 0 corresponds to the cubic map at the top of the figure).

We have emphasized several times that in order to obtain a complete reconstruction of the canonical cut sequences of *every* map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, we must consider every possibility wherever there is more than one possibility to assume some value. For that, we can proceed as usual in a backtracking algorithm: After we have obtained and handled the (possibly) canonical cut sequence of one map $M \in \mathcal{M}(g, n, D, b_l, b_u)$, we pretend that the reconstruction on the highest level (level 22 in the example in Figure 9.4) was not successful. Then the algorithm will backtrack from the highest level and the reconstruction will be continued exactly at the point where it had stopped when M was found.

As soon as we have reconstructed a cut sequence of a cubic map completely (so that in the corresponding tree, all patches corresponding to the leaves are tiles), we are able to construct the map. However, this cannot be done in a top–down manner as shown in Figure 9.1. Although the marked vertices can always be uniquely determined, it is not always possible to determine the canonical boundary vertex and thus the start vertex of the next canonical zig–zag path in the intermediate patches: If a patch P contains more than one boundary face having the highest quality, then we must compute the mark–qualities of the marked vertices in these boundary faces in order to determine the canonical boundary vertex of P. But for that, the complete incidence structure of P must be known. When the incidence structure of P is known, then the incidence structures of the canonical ancestors of P are known, too. Hence we *must* construct P in a bottom–up manner. However, this is not difficult. All agglutinations are unique due to the marking rules, i.e. if an agglutination would be revoked, then those vertices contained in the agglutinated boundary faces would be marked that had actually been marked before the agglutination took place. Take Figure 8.7 on page 123 as an example: As in the bottom–up approach, there is exactly one agglutination of the canonical ancestors of P that does not contradict the marking rules.

Constructing maps and patches in a bottom–up manner yields even a big advantage: All tests that must be recovered as soon as a patch can be constructed can be performed as in the bottom–up approach. We have already pointed out in Section 9.1 that as soon as we have reconstructed a cut sequence completely, we have only *possibly* found a *canonical* cut sequence of a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$. In order to secure that the cut sequence is canonical and the resulting cubic map M is in $\mathcal{M}(g, n, D, b_l, b_u)$, we must recover some tests. First, we consider the tests listed in Section 8.1 and discussed in Section 9.1:

Condition 1: We do not need to recover the test whether the degree of every face of M is in D because if this was not the case, then the reconstruction would have failed and backtracked when approaching the corresponding tile.

Condition 2: We also do not need to recover the test whether $p_d(M) \leq b_u(d)$ for every $d \in D$ because this test is performed whenever a tile is approached during the reconstruction. However, we need to recover the test whether $b_l(d) \leq p_d(M)$ for every $d \in D$. If this is not the case, then $M \notin \mathcal{M}(g, n, D, b_l, b_u)$.

Condition 8: This test must be completely recovered as soon as a patch P can be constructed. For that, the patch P is constructed in a bottom–up manner. As in the bottom–up approach, after each agglutination, the start vertex v of the corresponding zig–zag path in the obtained patch Q must be marked. v is always markable. However, the test whether v is the marked vertex in Q having the highest total quality must be recovered. If it is not, then the reconstructed cut sequence is not canonical and P and thus M must be rejected.

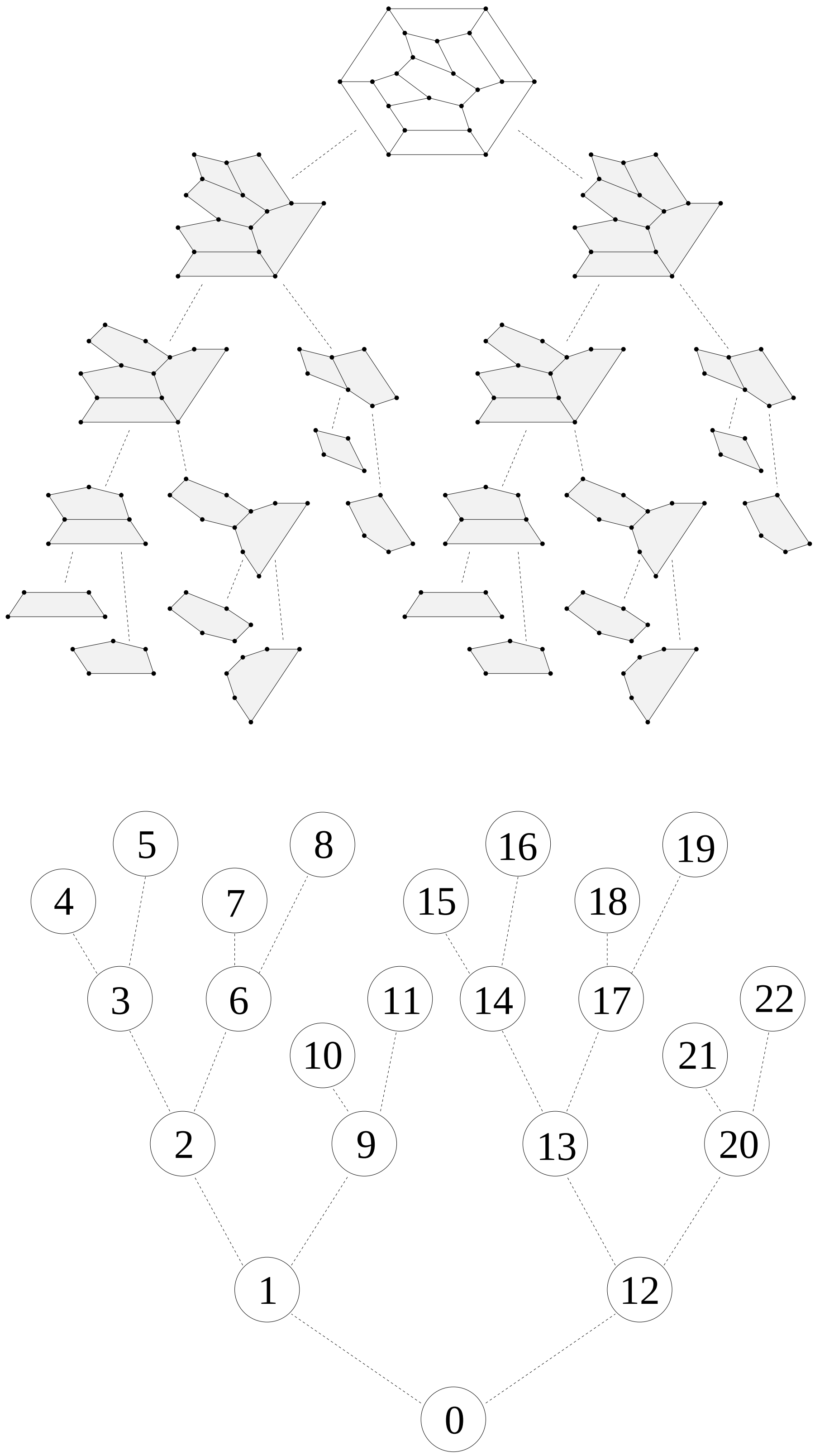

Figure 9.4

Condition 8 is verified as in the bottom–up approach. The only difference is that in the top–down approach, all tests on the intermediate patches are recovered in one passage (while in the bottom–up approach, the test can be performed immediately for every new generated patch). However, when multipatches are used in the bottom–up approach, then the tests on the intermediate patches are recovered in one passage, too.

The further tests are also performed as in the bottom–up approach:

- As soon as M is constructed, it must be tested as in the bottom–up approach whether the Petrie path in the reconstructed cut sequence is a canonical Petrie path in M. If it is not, then M must be rejected, otherwise M must be accepted.
- The tests described in Section 8.5 can be recovered as in the bottom–up approach as soon as the corresponding patches can be constructed.

Constructing maps and patches in a bottom–up manner yields also a big advantage in securing a complete reconstruction: The concepts that secure a complete construction when multipatches are used in the bottom–up approach secure a complete reconstruction also in the top–down approach. When using multipatches in the bottom–up approach, we sum up all agglutinations with the following common properties:

- the number of patches that are to be agglutinated (1 or 2)
- the sizes of the patches that are to be agglutinated
- the boundary codes of the patches that are to be agglutinated

These details are identical for every patch or every pair of patches, respectively, that is/are to be agglutinated. By performing the same agglutination for all these patches/pairs of patches, we secure that the boundary codes and the sizes of the obtained descendants are identical for each agglutination. In the top–down approach, we do the opposite: We sum up all cuts where the patches that are to be cut have a fixed size and a fixed boundary code, and the patches that are obtained have the same common properties as above. For that, we have the following restrictions to the cuts (see Section 9.4):

- If there are several boundary faces having the highest quality, then we take only one of them as the canonical boundary face F_1.
- In the linkage case, if there are several boundary faces $F \neq F_1$ having the same canonical boundary code, then we accept only one of them as F_2. And if the boundary face F_2 is periodical, then the end vertex must occur in one certain period of F_2, for example in the first period.

If these restrictions were not given, then we would have different cuts where the patches that are to be cut have a common size and a commmon boundary code, and the patches that are obtained have the same common properties as above. The corresponding multipatches would be identical for the different cuts. But with the restrictions, it holds that for every cut that is performed during the top–down approach, either the multipatch before the cut or the multipatches after the cut differ from the ones in every other cut. Thus the multipatches in the top–down approach correspond to the multipatches in the bottom–up approach, and thus in order to secure completeness, we can construct all implicit patches of a multipatch in the same way as in the bottom–up approach. In the following description, all details will be given once more (see above, Section 8.3, and Section 8.8). Let p be the (possibly) canonical zig–zag path in an implicit patch P of a multipatch t. The top–down part of the reconstruction, i.e. the reconstruction of the canonical cut sequences, is called the “first step”, and the bottom–up part, i.e. the construction of all implicit patches of a multipatch, is called the "second step".

- Let p be a linkage path. During the first step, exactly one representative out of all boundary faces having the highest quality may be taken as the boundary face F_1 containing the start vertex of p. The remaining boundary faces are partitioned into classes depending on their qualities, and out of every class, exactly one representative may be taken. The 3–valent boundary vertices of each such boundary face are also partitioned into classes depending on their qualities, and out of every class, exactly one representative may be taken.

  Let q be the quality of the new boundary face F_3 in the ancestor. During the second step, we presuppose that the canonical ancestor Q of P is given (because we are constructing bottom–up). It holds that if there exist two or more boundary faces in Q having the quality q and the same total quality, then only one of those faces may be taken as F_3 (denoted as in Section 8.2). If one boundary face F_3 is agglutinated with itself along a linkage path, then the restrictions to i and i' (denoted as in Section 8.2) are as follows: i must correspond to the last period in F_3, and for i' there are no restrictions except those given in Section 8.2. Let F_1 and F_2 be the resulting new boundary faces in P (as denoted in Section 8.2). As described in Section 8.2, there might be more than one markable boundary vertex incident with F_2. By incrementing the offset d by the length of one period (in the *valency* sequence) iteratively, all different possibilities are considered, and if some of the possibilities yield mark–isomorphic patches, then only one of those possibilities is actually taken.
- Let p be an intersection path or an enclosure path with $|A(P,p)| = 1$. During the first step, exactly one representative out of all boundary faces having the highest quality may be taken as the boundary face F_1 containing the start vertex of p.

  Let q_3 and q_4 be the qualities of the obtained new boundary faces F_3 and F_4 (denoted as in Section 8.2). During the second step, we presuppose that the canonical ancestor Q of P is given. It holds that if there exist two or more boundary faces in Q having the quality q_3 and the same total quality, then only one of those faces may be taken as F_3. To F_4, there is no further restriction except that of course $F_4 \neq F_3$. The restrictions to i and i' (denoted as in Section 8.2) are as follows: i must correspond to the last period in F_3 and i' must correspond to the last period in F_4.
- Let p be an intersection or an enclosure path with $|A(P,p)| = 2$. During the first step, exactly one representative out of all boundary faces having the highest quality may be taken as the boundary face F_1 containing the start vertex of p. Let q_3 and q_4 be the qualities of the new boundary faces F_3 and F_4 after the cut (denoted as in Section 8.2). F_3 must be assigned to the first ancestor Q_1 and F_4 must be assigned to the second ancestor Q_2. The unmodified boundary faces must be assigned to F_3 and F_4 in every possible way such that the resulting boundary codes for Q_1 are pairwise distinct (and the resulting boundary codes for Q_2 are pairwise distinct, too). Hence if there are two unmodified boundary faces F and F' having the same quality, then either F is assigned to Q_1 and F' is assigned to Q_2 or vice versa. Both possibilities yield the same boundary codes for Q_1 and the same boundary codes for Q_2.

  During the second step, we presuppose that the canonical ancestors Q_1 and Q_2 of P are given. It holds that if there exist two or more boundary faces in Q_1 having the quality q_3 and the same total quality, then only one of those faces may be taken as F_3. Moreover, if there exist two or more boundary faces in Q_2 having the quality q_4 and the same total quality, then only one of those faces may be taken as F_4. The restrictions to i and i' (denoted as in Section 8.2) are as usual: i must correspond to the last period in F_3 and i' must correspond to the last period in F_4.

As in the bottom–up approach, an *advanced patch testing* can be introduced if necessary. This patch testing works exactly as in the bottom–up approach (i.e. it works in a bottom–up manner) because as soon as the canonical cut sequence of a patch P has been reconstruted, the canonical cut sequences of its canonical ancestors have been reconstructed before, and thus these ancestors can be constructed and tested before P itself is constructed and tested. Unlike in the bottom–up approach, the cumulative face vectors can be verified with the upper limits given by b_u, that is: Let a cubic map be cut into more than one patch, e.g. into two patches P_1 and P_2. Let Q be an iterated canonical ancestor of P_2, and let an appropriate patch P_1 already have been found. If $p_d(P_1) + p_d(Q) > b_u(d)$ for some $d \in D$, then P_1 and P_2 cannot be agglutinated with each other. This verification is useful during the advanced patch testing.

In the bottom–up approach, at most one ancestor of a cubic map $M \in (g, n, D, b_l, b_u)$ is generated recursively. However, in the top–down approach, *all* ancestors are generated recursively. Let the following situation be given: The top–down approach searches for three patches P_1, P_2, and P_3. The sizes and the boundary codes c_1, c_2, and c_3 of these patches are fixed (by the corresponding multipatches), and the algorithm has already found an appropriate patch P_1. Now it is searching for a patch P_2. If it does not succeed in finding a patch P_2, then there are different possible reasons for that:

- There does not exist a possibly canonical patch of size $s(P_2)$ having the boundary code c_2.
- There exist possibly canonical patches having the size $s(P_2)$ and the boundary code c_2, but for every such patch P_2, the cumulative face vectors of P_1 and P_2 exceed the upper limits given by b_u.

Of course, in both cases it does not make sense to search for a patch P_3, and in fact the recursion does not approach the level where it is searched for P_3. However, while in the first case it does not make sense to search for another patch having the size $s(P_1)$ and the boundary code c_1, in the second case it does make sense because the corresponding face vector of P_1 may differ from the previous one. If the two cases of not finding an appropriate patch are distinguished during the reconstruction, then a lot of time can be saved everytime the first case occurs.

9.6 Top–down approach versus bottom–up approach

In this section, we want to compare the performances of the top–down approach and the bottom–up approach. For both ways of construction, it holds that it is not possible to trace only (inverses of) canonical cut sequences. Hence it is interesting to know which construction principle approaches closer to this ideal.

- The bottom–up approach has the advantage that it includes the restriction to a set of permitted face degrees in a natural way by starting only with the appropriate tiles. Moreover, restrictions that refer to the incidence structures of the maps can easily be considered as described in Section 8.5.
- In the top–down approach, restrictions to the incidence structures of maps can hardly be verified until the reconstruction approaches a tile. Only at this point, we can verify if the tile has a permitted degree $d \in D$. So essentially, the canonical cut sequence of *every* cubic map of a given genus and a given order will be reconstructed by the top–down approach. By counting the tiles that have a permitted degree and comparing the sums with b_u, and by considering the possible numbers of convex edges as indicated in Section 9.4 (page 174), some branches in the search tree will of course not have to be traced. But the performance is still much worse than in the bottom–up approach.

- On the other hand, in the top–down approach, restrictions to the possible canonical Petrie paths inside the desired cubic maps can be considered in a natural way. For example, if Theorem 8.5 on page 138 or Theorem 8.6 on page 139 can be applied, then some of the three parts of the whole reconstruction can simply be omitted. In Section 9.8, more such criteria will be provided.
- In the bottom–up approach, we must derive the boundary code of a patch P out of the boundary codes of its ancestors in order to determine whether the corresponding agglutination is possibly canonical, i.e. whether the start vertex of the corresponding zig–zag path is markable. This is not necessary in the top–down approach. Here we only consider zig–zag paths whose start vertices are markable.
- In the bottom–up approach, regardless of the question whether we use the standard size or the advanced size for patches, there is a threshold size (or an advanced threshold size, respectively) s so that every patch P with $s(P) \leq s$ or $s'(P) \leq s$, respectively, *must* be stored. Thus in the worst case, the bottom–up approach cannot be performed due to a lack of memory. In the top–down approach, no patches need to be stored. Hence the top–down approach can always be performed (if the required time is not the problem).

Considering all advantages and disadvantages of both approaches, the best performance of *CGF* is obtained when both approaches are combined. This means that the resulting **combined approach** consists of two parts. In the first part, up to a given threshold size s, all patches are generated and stored using the bottom–up approach. As usual, the stored patches are called **permanent patches**. In the second part, the top–down approach is invoked. As soon as the top–down approach searches a patch P of size $s(P) \leq s$, it does not need to continue the recursion anymore. Instead, it can look directly into the patch tree in order to find out whether a patch of size $s(P)$ having the required boundary code exists. If not, then the algorithm can backtrack.

The combined approach joins the advantages of both pure approaches: Up to a certain size, there exist very few canonical patches, and they can be generated most easily with the bottom–up approach. But for larger sizes, the number of generated patches would become so large that it is faster to continue from the top. And in the top–down part of the combined approach, the search tree (and in particular, the *depth* of the search tree) is much smaller than in the pure top–down approach.

The choice of the threshold size s is essential for the performance of the combined approach. For a fast construction, s should be approximately in the middle between the biggest and the smallest possible patch size, i.e. $s \approx (\frac{f}{2}, \frac{g}{2}, \frac{g+2}{2})$ so that s is approximately the threshold size that is used when recursion is applied in the bottom–up approach. However, the choice depends also on the amount of accessible memory. Thus sometimes s must be chosen smaller than the optimal value. But at least, the combined approach can always be performed (if the required time is not the problem).

By looking for patches in the patch tree, a phenomenon occurs that has already occurred in the recursion of the bottom–up approach: Since there might be more than one patch stored at one leaf in the patch tree (i.e. all patches have the same size and the same boundary code), situation 1 described in Section 8.8 might occur also in the second part of the combined approach. It is treated in the same way as in the bottom–up recursion.

9.7 Improving the performance of the combined approach

In this section, we give some heuristics that help to reduce the required amount of time and memory of the combined approach.

Whenever a cubic map or a patch is cut into more than one patch, the smallest of these patches should be searched first. The smaller the size of a patch, the earlier an appropriate patch can be found provided that an appropriate patch exists. If it does not exist, then the search for the other (bigger) patches can be omitted. If there exists an appropriate patch, then this indicates that the chosen combination of patch sizes and boundary codes might be realizable.

The next heuristics helps to optimize the worst case behaviour of the top–down part of the combined approach. In Figure 9.5, the possible genera and numbers of inner faces of the patches are pictured in a matrix (for an example with $g = 4$ and $f = 7$). Given a threshold size s, it can be stated that the size of every permanent patch is located either to the left of s or below s in the same column as s. In Figure 9.5, the corresponding domain is drawn in grey. In order to optimize the worst case behaviour of the top–down part, the goal is to minimize the largest possible number of cuts that must be performed in order to cut a possibly canonical patch P into permanent patches. Since $g_Q \leq g_P$ as well as $f'_Q \leq f'_P$ for every ancestor Q of P, the position of $s(Q)$ in the matrix is never located above $s(P)$ or to the right of $s(P)$. Hence if a cut sequence is traced in the matrix, then we always move to the left and to the bottom. Figure 9.6 shows two extremal paths in the matrix. While path a approaches permanent patches very quickly, it takes many steps in the case of path b. In order to improve the worst case behaviour, the domain of permanent patches should better be distributed as shown in Figure 9.7. To this end, we assign to each possible patch genus g' an upper limit $f'_{g'}$ of inner faces so that a patch P of size $s(P) = (f'_P, g', b_P)$ will be stored if and only if $f'_P \leq f'_{g'}$ (one could as well assign an upper limit for the genus to each possible number of inner faces, but in general, the maximum number f of inner faces is much higher than the maximum genus g, so in this case we would have to do many more assignments). A further improvement is given by assigning to each possible patch genus g' an upper limit $b_{g'}$ of boundary faces so that a patch P of size $s(P) = (f'_{g'}, g', b_P)$ will be stored if and only if $b_P \leq b_{g'}$. For the assignments, there are the following rules:

- if $g_1 < g_2$, then $f'_{g_1} \geq f'_{g_2}$
- if $g_1 < g_2$ and $f'_{g_1} = f'_{g_2}$, then $b_{g_1} > b_{g_2}$

If these rules are considered, then the permanent patches can be generated as usual with increasing size. The limits substitute the threshold size given before. For every patch size s, it is checked whether s lies within the given limits. If it does, then it is secured by the rules that the canonical ancestors of every (permanent) patch having size s are also permanent and have already been generated and stored.

As a final heuristics, the *advanced size* introduced in Section 8.7.6 can be used also in the combined approach: We assign to each possible patch genus g' upper limits $c_{g'}$, $f'_{g'}$, and $b_{g'}$ so that a patch P having the advanced size $s'(P) = (c_P, f'_P, g', b_P)$ is permanent if and only if

- $c_P \leq c_{g'}$
- if $c_P = c_{g'}$, then $f'_P \leq f'_{g'}$
- if $c_P = c_{g'}$ and $f'_P = f'_{g'}$, then $b_P \leq b_{g'}$

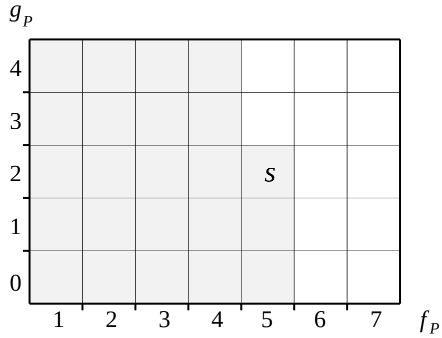

Figure 9.5

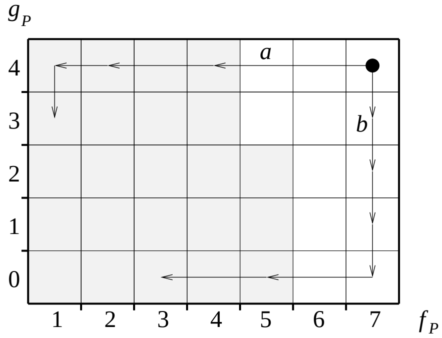

Figure 9.6

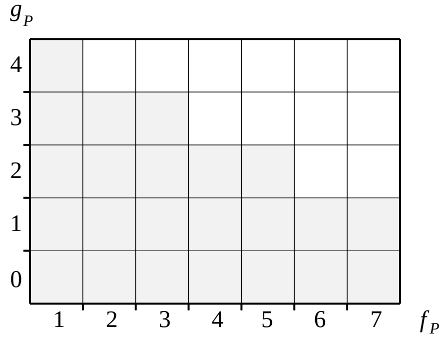

Figure 9.7

For the assignments, there are the following rules:

- if $g_1 < g_2$, then $c_{g_1} \geq c_{g_2}$
- if $g_1 < g_2$ and $c_{g_1} = c_{g_2}$, then $f'_{g_1} \geq f'_{g_2}$
- if $g_1 < g_2$, $c_{g_1} = c_{g_2}$, and $f'_{g_1} = f'_{g_2}$, then $b_{g_1} > b_{g_2}$

If the advanced size is used, then during the top–down part, the critical sum must be partitioned in every possible way wherever a cubic map or a patch is cut into more than one patch. That is:

- If a cubic map or a patch having the critical sum c is cut into two patches P_1 and P_2, then the critical sum c_{P_1} of P_1 may vary between 0 and c, and then $c_{P_2} = c - c_{P_1}$.
- If a cubic map having the critical sum c is cut into three patches P_1, P_2, and P_3, then c_{P_1} may vary between 0 and c, c_{P_2} may vary between 0 and $c - c_{P_1}$, and then $c_{P_3} = c - c_{P_1} - c_{P_2}$.

However, it is not useful to run through all values because in most cases, there are only few valid combinations. For example, due to Lemma 6.1 on page 68, the critical sum of a map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is always unique if $|D| \leq 2$, and for patches it holds:

Lemma 9.3 *If $|D| \leq 2$, then the critical sum c_P of a canonical patch P is unique for a given size $s(P)$ and a given number k_P of convex edges contained in P.*

Proof: Let $D = \{d\}$. If $d > 6$, then $c_P = (d-6) \cdot f'_P$, and if $d \leq 6$, then $c_P = 0$ so that c_P does not even depend on k_P. Let $D = \{d_1, d_2\}$ with $d_1 < d_2$. Due to Lemma 8.10, it holds

$(6-d_1)\cdot p_{d_1}(P)+(6-d_2)\cdot p_{d_2}(P)=12\cdot(1-g_P)-6b_P-k_P$. Together with the condition $p_{d_1}(P)+p_{d_2}(P)=f'_P$, we obtain

$$\begin{aligned} & (6-d_1)\cdot(f'_P-p_{d_2}(P))+(6-d_2)\cdot p_{d_2}(P)=12\cdot(1-g_P)-6b_P-k_P \\ \Leftrightarrow\quad & (6-d_1)\cdot f'_P+(d_1-d_2)\cdot p_{d_2}(P)=12\cdot(1-g_P)-6b_P-k_P \\ \Leftrightarrow\quad & p_{d_2}(P)=\frac{12\cdot(1-g_P)-6b_P-k_P-(6-d_1)\cdot f'_P}{d_1-d_2} \end{aligned}$$

so that $p_{d_2}(P)$, thus $p_{d_1}(P)=f'_P-p_{d_2}(P)$, and thus $c_P=\sum_{\{i\in\{1,2\}\mid d_i>6\}}(d_i-6)\cdot p_{d_i}(P)$ can be uniquely determined. ■

Hence it is better to perform some simple computations in advance (before performing any reconstruction) similar to the ones in Section 6.1 and Section 9.4: Let $D=\{d_1,\ldots,d_{n'}\}$ (where $n'=|D|$) such that $d_i<d_j$ if and only if $i<j$ for every $i,j\in\{1,\ldots,n'\}$.

- First, as described in Section 6.1, all possible face vectors of maps in $\mathcal{M}(g,n,D,b_l,b_u)$ are considered. For every such face vector s, the critical sum is computed. Thus all possible critical sums of cubic maps in $\mathcal{M}(g,n,D,b_l,b_u)$ are obtained.
- Then, as described in Section 9.4, all possible face vectors of the canonical patches are computed. Let $\langle f'_1,\ldots,f'_{n'}\rangle$ with $f':=\sum_{i=1}^{n'}f'_i$ be the face vector of a canonical patch Q whose size is known (it holds $f'=f'_Q$). Then the number k_Q of convex edges in Q can be derived using Lemma 8.10, and also the critical sum

  $$c_Q=\sum_{\{i\in\{1,\ldots,n'\}\mid d_i>6\}} f'_i\cdot(d_i-6)$$

  of Q can be computed. So if $\langle f'_1,\ldots,f'_{n'}\rangle$ with $f':=\sum_{i=1}^{n'}f'_i$ is a *possible* face vector of a canonical patch of size (f',g',b'), then with Lemma 8.10,

  $$k=12\cdot(1-g')-6b'-\sum_{i=1}^{n'}f'_i\cdot(6-d_i)$$

  is the corresponding number of convex edges in the patch, and thus

  $$c=\sum_{\{i\in\{1,\ldots,n'\}\mid d_i>6\}} f'_i\cdot(d_i-6)$$

  is a *possible* critical sum in a canonical patch of size (f',g',b') containing k convex edges.

Thus all possible critical sums of canonical patches depending on their sizes and their numbers of convex edges can be computed in advance. During the reconstruction of canonical cut sequences, the size of a patch is determined explicitly, and the number of convex edges can be derived applying Lemma 8.10. So the possible critical sums that need to be considered can be assigned easily to a searched patch.

In fact, while the possible critical sums for a canonical patch P are determined and stored, it suffices to store them for the case $g_P=0$ and $b_P=1$: Let Q be a patch having the same face vector as P (so that $f'_Q=f'_P$ and $c_Q=c_P$). Then it holds $k_P=12\cdot(g_Q-g_P)-6\cdot(b_P-b_Q)+k_Q$ (see Section 9.4) so that with $g_P=0$ and $b_P=1$, it holds $k_P=k_Q+12g_Q+6\cdot(b_Q-1)$. Hence if c is a possible critical sum of a patch Q of size (f'_Q,g_Q,b_Q) containing k_Q convex edges, then c is also a possible critical sum of a patch P with $s(P)=(f'_Q,0,1)$ containing $k_Q+12g_Q+6\cdot(b_Q-1)$ convex edges. Hence it suffices to store the possible critical sums for patches of genus 0 having one boundary face, and the possible critical sums of patches having a higher genus and/or more than one boundary face can be derived from the earlier ones.

9.8 Decreasing the number of constructed maps

As mentioned in Section 9.6, restrictions to the possible canonical Petrie paths inside the desired cubic maps can be considered in a natural way. For example, if Theorem 8.5 on page 138 can be applied, then no Jordan curve paths need to be considered provided that Jordan curve paths have a lower quality than sandwich and dumbbell paths. Thus one of three parts of the whole reconstruction can simply be omitted. In particular, the Jordan curve reconstruction is in general more time consuming than the sandwich or dumbbell reconstruction because after the cut, at most two patches are obtained. Hence the average size of the smallest obtained patch is larger than in the sandwich or dumbbell case, and thus the average number of cuts that must be performed in order to approach permanent patches is also larger than in the sandwich or dumbbell case.

If Theorem 8.6 can be applied, then no Jordan curve paths and no sandwich paths need to be considered.

Theorem 9.4 *If* $1 \in D$ *and* $b_l(1) > 0$*, then every map* $M \in \mathcal{M}(g, n, D, b_l, b_u)$ *contains a dumbbell path* p *with* $l_1(p) = 1$*.*

Proof: This is trivial because every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ contains a monogon, and every semi–edge which is incident with a monogon is the start semi–edge of a dumbbell path: With the notation of Section 6.3, it always holds $v_0 = v_t$ with $t = 1$. ■

This theorem can be applied in the following way: We assign a higher quality to dumbbell paths than to sandwich paths and to Jordan curve paths. Thus the canonical Petrie path in every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is a dumbbell path. Since a dumbbell path p with $l_1(p) = 1$ has a higher quality than every dumbbell path q with $l_1(q) > 1$, it is clear that the canonical Petrie path in every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is a dumbbell path with $l_1(p) = 1$. Hence it suffices to reconstruct cut sequences where the first cut is a cut along a dumbbell path whose first part has length 1. Here the boundary code of the patch P containing the start side boundary is $(1, \langle 1 \rangle, \langle 0 \rangle)$, and thus P is always a 1–tile.

Theorem 9.5 *If* $1 \notin D$ *and* $2 \in D$ *and* $b_l(2) > 0$*, then every map* $M \in \mathcal{M}(g, n, D, b_l, b_u)$ *contains a dumbbell path* p *with* $l_1(p) = 2$ *or a sandwich path* p *with* $l_1(p) = 1$ *and* $l_2(p) = 1$*.*

Proof: Look at Figure 9.8. W.l.o.g., we investigate the Petrie path p whose initial rooted edge is $(\{h_1, h_1'\}, h_1, \texttt{right})$. Thus $p = \langle h_1, h_1', h_2', h_2, h_4, \ldots \rangle$. If h_3 is incident with p, then h_3 is the end semi–edge of p and p is a sandwich path with $l_1(p) = l_2(p) = 1$. Otherwise p is a dumbbell path with $l_1(p) = 2$. ■

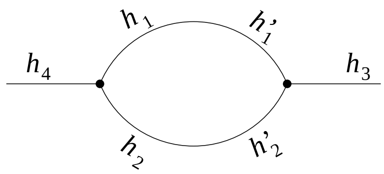

Figure 9.8

This theorem can be applied in the following way: We assign a higher quality to dumbbell paths than to sandwich paths and a higher quality to sandwich paths than to Jordan curve paths. Thus the canonical Petrie path in every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is either a dumbbell path or a sandwich path p with $l_1(p) = l_2(p) = 1$ (because p has a higher quality than any sandwich path q with $l_1(q) > 1$ or $l_2(q) > 1$). The latter case occurs only if M does not contain any dumbbell path. Hence it suffices to reconstruct cut sequences where the first cut is either a cut along a sandwich path p with $l_1(p) = l_2(p) = 1$ or a cut along a dumbbell path. In the first case (sandwich path), the patch containing the start side boundary is always a 2–tile.

If the expected number of accepted maps is very small so that it is no problem to filter out isomorphic maps after the construction, then we can further reduce the number of constructed maps as follows: We perform two constructions. During the first construction, we assign a higher quality to dumbbell paths than to sandwich paths (and a higher quality to sandwich paths than to Jordan curve paths), and we reconstruct cut sequences where the first cut is a cut along a dumbbell path p with $l_1(p) = 2$. Here the patch containing the start side boundary is always a 2–tile. During the second construction, we assign a higher quality to sandwich paths than to dumbbell paths (and a higher quality to dumbbell paths than to Jordan curve paths), and we reconstruct cut sequences where the first cut is a cut along a sandwich path with $l_1(p) = l_2(p) = 1$. Here the patch containing the start side boundary is always a 2–tile, too. Due to Theorem 9.5, every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is constructed at least once, and due to the quality assignments, every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ is also *accepted* at least once. Thus if isomorphic maps can be filtered out easily, then a complete construction can be obtained like this. An example where this kind of construction has been performed can be found in Section 10.2.

Theorem 9.6 *Let $g = 0$. If d is a multiple of 3 for every $d \in D$, then every map $M \in \mathcal{M}(g, n, D, b_l, b_u)$ contains neither sandwich paths nor dumbbell paths.*

Proof: If d is a multiple of 3 for every $d \in D$, then for every possibly canonical patch P, it holds $k_P = 12 \cdot (1 - g_P) - 6b_P - \sum_{d \in D}(6 - d) \cdot p_d(P)$ due to Lemma 8.10, and hence k_P is a multiple of 3, too. Let M be a planar cubic map containing a sandwich path or a dumbbell path p. If M is cut along p, then due to Section 6.4.2 and 6.4.3, we obtain 3 patches containing one boundary face each (because $g_M = 0$). Due to Section 8.2, at least two of the obtained boundary faces and thus at least two corresponding patches each contain either 1 or 2 convex edges. But for $\mathcal{M}(g, n, D, b_l, b_u)$, the number of convex edges in every possibly canonical patch is a multiple of 3. So $M \notin \mathcal{M}(g, n, D, b_l, b_u)$. ■

So if $g = 0$ and d is a multiple of 3 for every $d \in D$, then it suffices to reconstruct cut sequences where the first cut is a cut along a Jordan curve path. Similarly, if $g = 0$ and d is a multiple of 2 for every $d \in D$, then it suffices to consider dumbbell paths whose first and third parts each have even lengths (because otherwise, we obtain a side boundary whose number of convex edges is 1). In Section 8.6.2, the corresponding cases are the cases 1 and 5. Moreover, it suffices to consider sandwich paths whose first and third parts each have odd lengths (because otherwise, we obtain a side boundary whose number of convex edges is 1). In Section 8.6.3, the corresponding case is case 2 (the cases 5–8 do no need to be considered because $g = 0$).

9.9 Parallelization and recovering

In this section, we describe how the knowledge of Section 4.5 can be applied to the construction of $\mathcal{M}(g, n, D, b_l, b_u)$. Recall Sections 4.5 and 8.10.

If the combined construction of $\mathcal{M}(g, n, D, b_l, b_u)$ is to be parallelized, then the set S of permanent patches must be generated (in the bottom–up part) in each subprocess $p_0, \ldots, p_{k-1}$. The reconstruction which is performed in the top–down part of the combined approach can be divided into a tree structure so that we can proceed as in Section 4.5. Here the tree–levels correspond to the recursion levels as illustrated in Figure 9.4. In level 0, all choices that are described in Section 9.3 are done. There are quite many variations so that in principle, it is always useful to choose $l = 0$ as the threshold level in order to have a small number of common computations that are done by each subprocess. However, the time consumptions of the subprocesses might become extremely inhomogeneous: In general, the reconstruction of cut sequences where the Petrie path is short requires much more time than the reconstruction of cut sequences where the Petrie path is long. Hence it is useful to provide a second threshold level $l' > l$ in order to partition obstinate subprocesses into subsubprocesses. There might be cubic maps whose permanent iterated ancestors are all found before recursion level l is approached in the reconstruction (of course, this never happens if $l = 0$). Then we demand that these maps are only accepted in the subprocess p_0. Similar things hold for cubic maps whose permanent iterated ancestors are all found before level l' is approached.

The recovering of an interrupted process can be performed as described in Section 8.10. In the dump files, the sequence of branch labels representing the tree–path to the current position in the search tree is written as usual. Moreover, if a permanent patch is stored in a list of patches having the same size and the same boundary code (so that these patches are the implicit patches of a common multipatch), then these patches must be numbered, and the numbers of the current implicit patches must be stored, too. When a process is restarted, then first the permanent patches are generated and stored anew. After that, the top–down part of the process continues at the position that has been stored in the dump file.

The amount of information that has to be stored is quite small. In *CGF*, every 30 minutes a dump file containing about 500 bytes is stored. Hence the loss of time after a restart is at most 30 minutes plus the time which is needed to generate the permanent patches. The generation of the permanent patches is *not* the bottleneck of the combined approach.

Chapter 10

Results

10.1 General tendencies

The generation rate of *CGF* strongly depends on $c_{\max}$ (the maximal possible *critical sum* in the constructed maps). The reason for this is as follows: Due to Lemma 8.10, it holds

$$\begin{aligned} k_P &= 12 \cdot (1 - g_P) - 6b_P - \sum_{d \in D} (6 - d) \cdot p_d(P) \\ &= 12 \cdot (1 - g_P) - 6b_P \; - \sum_{\{d \in D \mid d > 6\}} (6 - d) \cdot p_d(P) \; - \sum_{\{d \in D \mid d \leq 6\}} (6 - d) \cdot p_d(P) \\ &\leq 12 \cdot (1 - g_P) - 6b_P \; - \sum_{\{d \in D \mid d > 6\}} (6 - d) \cdot p_d(P) \\ &= 12 \cdot (1 - g_P) - 6b_P + c_P \end{aligned}$$

for every possibly canonical patch P. Moreover, it holds $c_P \leq c_{\max}$. So the higher $c_{\max}$, the higher the upper limit for c_P, and thus the higher the theoretical upper limit for k_P. Now if k_P is high, then many possibilities to agglutinate P with itself or with other patches must be tested (see question 3 in Section 8.3: the higher k_P, the more pairs (i, i') must be considered). In particular, many of these possibilities are tested *successfully* so that many patches are generated where many of them are not canonical.

The combined approach is almost always faster than the bottom–up approach, at least if the threshold size is chosen appropriately. The bottom–up approach is faster than the combined approach if no large faces (faces of degree greater than 6) are permitted because in this case, it holds $c_P = 0$ and thus $k_P \leq 6$ for every possibly canonical patch P due to the above estimation. So the number of possible agglutinations does not explode with increasing patch size. Moreover, the bottom–up approach is faster than the combined approach if only one face degree is permitted (e.g. for $\mathcal{M}(2, 28, \{7\}, b_l, b_u)$ with $b_l(7) = b_u(7) = 12$). Here the number of possible agglutinations explodes with increasing patch size, but for small patch sizes, there are very few possibly canonical patches. For example, there is only one possibly canonical patch of size $(1, 0, 1)$ and only one possibly canonical patch of size $(2, 0, 1)$.

In order to compare the bottom–up approach with the combined approach exemplarily, we investigate the construction of $\mathcal{M}(0, 52, \{5,7\}, b_l, b_u)$ with $b_l(5) = b_l(7) = 0$ and $b_u(5) = b_u(7) = \infty$. The corresponding standard program call of *CGF* is

```
CGF -g 0 -v 52 -f 5 -f 7
```

(compare appendix A) where some additional options must be used in order to determine the kind of approach that is to be applied.

approach	stored patches	time
bottom–up, no recursion	805 647	66.4s
bottom–up, recursion	405 108	66.5s
bottom–up, advanced recursion	1 450	73.9s
combined, standard patch size, $f_0' = 17$	727 293	38.1s
combined, standard patch size, $f_0' = 16$	653 517	32.0s
combined, standard patch size, $f_0' = 15$	538 885	25.6s
combined, standard patch size, $f_0' = 14$	405 108	19.0s
combined, standard patch size, $f_0' = 13$	266 880	13.2s
combined, standard patch size, $f_0' = 12$	153 644	8.6s
combined, standard patch size, $f_0' = 11$	74 690	6.1s
combined, standard patch size, $f_0' = 10$	31 090	5.6s
combined, standard patch size, $f_0' = 9$	10 957	7.5s
combined, standard patch size, $f_0' = 8$	3 566	12.7s
combined, standard patch size, $f_0' = 7$	1 095	24.3s
combined, advanced patch size, $c_0 = 7$, $f_0' = 28$	159 826	11.1s
combined, advanced patch size, $c_0 = 6$, $f_0' = 28$	32 627	3.5s
combined, advanced patch size, $c_0 = 5$, $f_0' = 28$	6 694	3.4s
combined, advanced patch size, $c_0 = 4$, $f_0' = 28$	1 450	5.8s
combined, advanced patch size, $c_0 = 3$, $f_0' = 28$	314	10.6s
combined, advanced patch size, $c_0 = 2$, $f_0' = 28$	74	30.5s

Table 10.1: computations for $\mathcal{M}(0, 52, \{5,7\}, b_l, b_u)$ with $b_l(5) = b_l(7) = 0$ and $b_u(5) = b_u(7) = \infty$

Table 10.1 shows details concerning the number of stored patches and the required time for the construction. In the top–down approach, it was always set $b_0 = 1$ (for f_0', b_0, and c_0, see Section 9.7). Table 10.1 shows that in this example, the combined approach is much faster than the bottom–up approach. The usage of the advanced size for patches yields a faster and less memory consuming construction. One can see very well that for the combined construction, regardless whether the standard or the advanced size is used, with respect to the required time there is some optimal limit f_0' or (c_0, f_0'), respectively, for the domain of the permanent patches. In this example, the optimal limits are 10 and $(5, 28)$, respectively. If the chosen limit is higher than the optimal limit (i.e. the domain of the permanent patches is larger), then the algorithm requires too much time for the generation of the permanent patches. If the chosen limit is lower than the optimal limit, then the algorithm requires too much time for the top–down part of the combined approach. On the other hand, the memory consumption is smaller so that below the optimal limit, we have a classical time–space–tradeoff.

The required time listed in Table 10.1 refers to an Intel Pentium II computer with 350 MHz. It should be remarked that if the algorithm was implemented such that it could only construct *planar* cubic maps, then for the bottom–up approach, it would be about twice as fast as *CGF* actually is. The reason for this is that the boundary code of a patch would be much simpler and thus also the patch tree would be less complex if only planar patches had to be considered. The program *CPF* (see [46]) which constructs only planar cubic maps is about twice as fast as *CGF* (when the bottom–up approach in *CGF* is applied). Since *CPF* does not contain any top–down approach, it is not clear how much time can be saved by implementing the combined approach such that only planar maps can be constructed. However, since some important parts of the algorithm can be omitted (the partition of a genus into 2 summands, the partition of a set of boundary faces, the test whether a linkage cut is possible etc.), the resulting program would probably also be much faster than the universal program *CGF*.

For the above example, Table 10.2 shows how many patches of which size are stored when the bottom–up approach is applied. Column A contains the numbers for the bottom–up approach without recursion, column B contains the numbers for the bottom–up approach with (standard) recursion, and column C contains the numbers for the bottom–up approach with advanced recursion.

Remarks:

- No patch containing more than 22 inner faces satisfies the lower Euler criterion. Thus no patch containing more than 22 inner faces is stored.
- The estimation in Section 8.7.6 yields $f'_P \leq 14$ for possibly canonical patches whose critical sum c_P is at most 4. Since $(4, 14, 0, 1)$ is the advanced threshold size for the above example, it is clear that no patch containing more than 14 inner faces is stored if the advanced recursion is invoked.
- For the combined approach, the numbers of stored patches sorted by sizes need not be listed but can be derived from Table 10.2 because within the domain of the permanent patches, the numbers of stored patches in the combined approach correspond to the numbers of stored patches in the bottom–up approach: If the standard size for patches is used in the combined approach, then column A is relevant, and if the advanced size is used, then column C is relevant. Of course, no patches are stored whose sizes are not within the domain of the permanent patches.

Finally, for the construction of $\mathcal{M}(0, n, \{5, 7\}, b_l, b_u)$, Table 10.3 shows that the combined approach (CA) is not only faster than the bottom–up approach (BA), but even the growth of the required time with increasing vertex number n is more moderate for the combined approach than for the bottom–up approach if the memory consumptions of both approaches are identical. For the results in Table 10.3, the bottom–up approach has been applied using the advanced recursion. The combined approach has been applied using the advanced size for patches. Here, for each process, the limits for the domain of the permanent patches have been set such that exactly those patches were permanent whose advanced sizes were smaller or equal to the advanced threshold size. Thus, for each vertex number n, exactly the same set of patches was stored in the bottom–up approach and in the combined approach so that the memory consumptions of both approaches were identical.

patch size	stored patches A	B	C
(1, 0, 1)	2	2	2
(2, 0, 1)	3	3	3
(3, 0, 1)	8	8	8
(4, 0, 1)	28	28	28
(5, 0, 1)	72	72	63
(6, 0, 1)	254	254	153
(7, 0, 1)	728	728	209
(8, 0, 1)	2 471	2 471	298
(9, 0, 1)	7 391	7 391	239
(10, 0, 1)	20 133	20 133	225
(11, 0, 1)	43 600	43 600	95
(12, 0, 1)	78 954	78 954	84
(13, 0, 1)	113 236	113 236	8
(14, 0, 1)	138 228	138 228	35
(15, 0, 1)	133 777	0	0
(16, 0, 1)	114 632	0	0
(17, 0, 1)	73 776	0	0
(18, 0, 1)	45 250	0	0
(19, 0, 1)	17 767	0	0
(20, 0, 1)	9 594	0	0
(21, 0, 1)	1 411	0	0
(22, 0, 1)	4 332	0	0
(23, 0, 1)	0	0	0
(24, 0, 1)	0	0	0
(25, 0, 1)	0	0	0
(26, 0, 1)	0	0	0
(27, 0, 1)	0	0	0
(28, 0, 1)	0	0	0
total	805 647	405 108	1 450

Table 10.2: stored patches sorted by sizes

n	A	time consumption		
		BA	CA	ratio
28	1	0.0s	0.0s	
32	0	0.0s	0.0s	
36	2	0.1s	0.1s	1.00
40	0	0.5s	0.2s	2.50
44	13	2.7s	0.6s	4.50
48	0	15.9s	1.8s	8.83
52	73	73.9s	5.8s	12.74
56	5	479.6s	22.1s	21.70

Table 10.3: Time consumptions for the construction of $\mathcal{M}(0, n, \{5, 7\}, b_l, b_u)$. Column A lists the numbers of constructed maps, i.e. column A contains $|\mathcal{M}(0, n, \{5, 7\}, b_l, b_u)|$ for each n.

10.2 Applications

In this section, we refer to the applications that have been listed in Section 1.8. The following results have mostly been computed on a large cluster of computers using the parallelizing techniques described in Section 8.10 and 9.9. Since the cluster is rather heterogeneous, it is difficult to provide exact running times for the computations. Thus only approximate running times will be provided, and these approximate running times refer to a computer cluster consisting of 20 Intel Pentium I computers with 133 MHz and 45 Intel Pentium II computers with 350 MHz. This computer cluster will be called the *reference cluster* from now on.

Cubic maps of genus 2 containing only heptagons

Figures 10.1–10.6 show the 6 polyhedral cubic maps of genus 2 containing only heptagons (compare Section 1.8). Formula 1.3 on page 9 implies that each of these maps contains exactly 12 faces, and then formula 1.2 on page 9 implies that each of the maps contains exactly 28 vertices. Thus the maps can be constructed using the program call

```
CGF -g 2 -v 28 -f 7 -facecon 3
```

(see appendix A). The number of stored patches is 155, and the required time on an Intel Pentium II computer with 350 MHz is 4.8 seconds. This is one of the rare examples where the bottom–up approach is faster than the combined approach.

By omitting the option `-facecon 3`, one obtains all 56 (not necessarily polyhedral) cubic maps of genus 2 containing only heptagons. Here the number of stored patches is 206, and the required time on an Intel Pentium II computer with 350 MHz is 5.9 seconds. Thus it can be seen that the restriction to polyhedral maps is not implemented as a filter only but affects the running time and the required memory of a computation.

The 5–flow conjecture

The 5–flow conjecture has not yet been verified for maps of (orientable) genus 3. Using *CGF*,

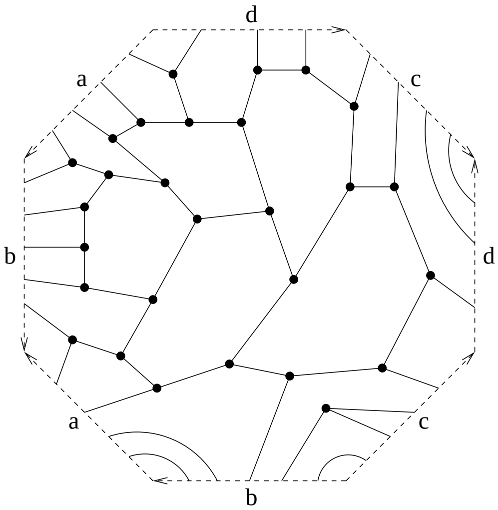

Figure 10.1

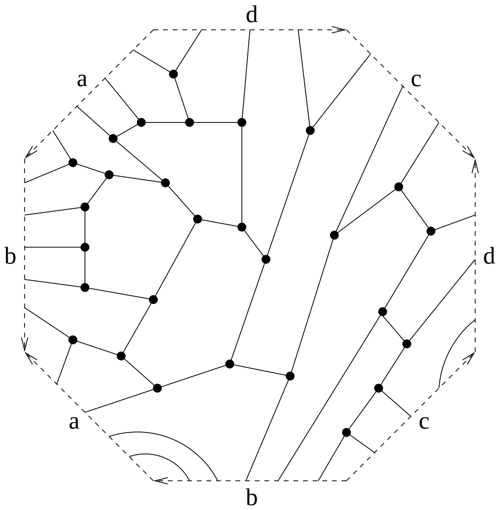

Figure 10.2

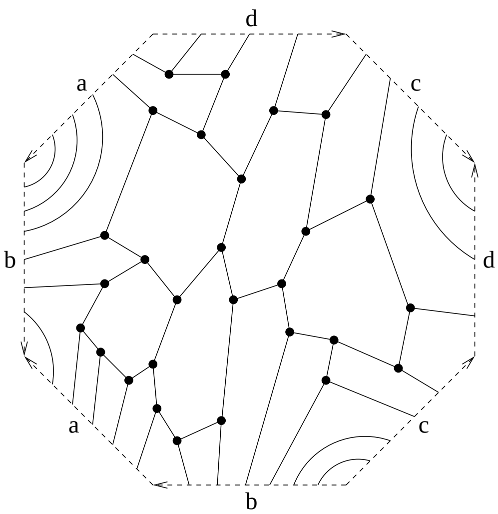

Figure 10.3

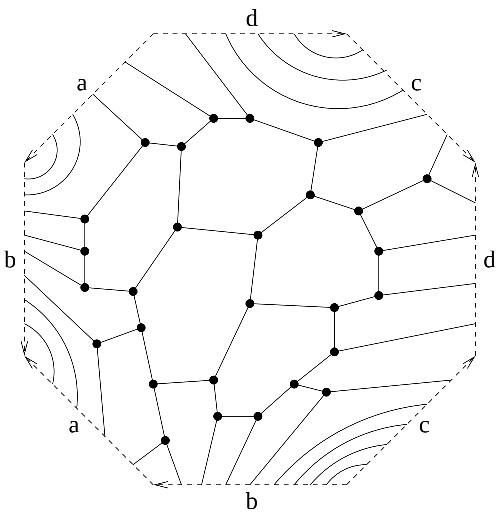

Figure 10.4

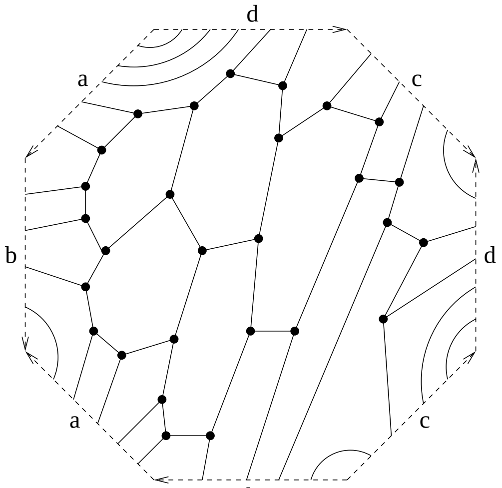

Figure 10.5

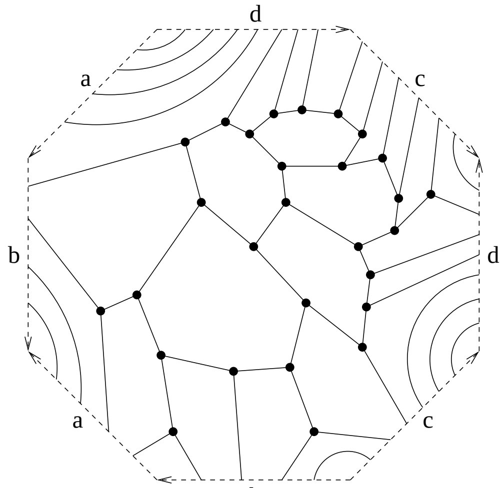

Figure 10.6

the estimated running time for the verification on the reference cluster is approximately 3 years. Thus the reference cluster is not suitable for the verification. Since customary computers become more and more powerful, the verification of the 5–flow conjecture should be performed in the future. For the verification, maps of different orders should be generated separately because depending on the order of a map, different face degrees are permitted. In principle, every face degree $d \geq 7$ is permitted. However, if M is a cubic map of genus 3 containing only faces of degree ≥ 7, then due to formula 1.3 on page 9, it holds $\deg(F) \leq 35 - \frac{1}{2}n_{_M}$ for every face F contained in M: For $g_{_M} = 3$, formula 1.2 on page 9 implies $f_{_M} = \frac{1}{2}n_{_M} - 4$. Moreover, since $p_d(M) = 0$ for every $d \in \{1, \ldots, 6\}$, it holds

$$-24 = 12 \cdot (1 - g_{_M}) = \sum_{d \geq 1} p_d(M) \cdot (6 - d) = \sum_{d \geq 7} p_d(M) \cdot (6 - d) \leq \sum_{d \geq 7} p_d(M) \cdot (-1) = -f_{_M}$$

and thus $f_{_M} \leq 24$. Let's assume that $p_{d'}(M) \geq 1$ for some $d' \geq 36 - \frac{1}{2}n_{_M}$. So

$$\begin{aligned}
\sum_{d \geq 1} p_d(M) \cdot (6 - d) &= \sum_{d \geq 7} p_d(M) \cdot (6 - d) \\
&= p_{d'}(M) \cdot (6 - d') \;+ \sum_{\{d \geq 7 \mid d \neq d'\}} p_d(M) \cdot (6 - d) \\
&\leq p_{d'}(M) \cdot (\frac{1}{2}n_{_M} - 30) \;+ \sum_{\{d \geq 7 \mid d \neq d'\}} p_d(M) \cdot (6 - d) \\
&\leq p_{d'}(M) \cdot (\frac{1}{2}n_{_M} - 30) \;+ \sum_{\{d \geq 7 \mid d \neq d'\}} p_d(M) \cdot (6 - 7) \\
&= p_{d'}(M) \cdot (f_{_M} - 26) \;- \sum_{\{d \geq 7 \mid d \neq d'\}} p_d(M) \\
&= p_{d'}(M) \cdot (f_{_M} - 26) - \big(f_{_M} - p_{d'}(M)\big) \\
&= (p_{d'}(M) - 1) \cdot f_{_M} - 25 p_{d'}(M) \\
&\leq (p_{d'}(M) - 1) \cdot 25 - 25 p_{d'}(M) \\
&= -25 \\
&< 12 \cdot (1 - g_{_M})
\end{aligned}$$

This is a contradiction to formula 1.3.

So for example, an appropriate program call to verify the 5–flow conjecture for all cubic maps of genus 3 and order 50 is

```
CGF -g 3 -v 50 -f 7 -f 8 -f 9 -f 10 -mapcon 23 -output stdout | fluss5 > no5
```

(see appendix A) where in this case, the combined approach should be applied (using the option `-topdown` or `-td_limits`). The option `-mapcon 23` restricts the construction to 2–connected maps. The accepted maps are piped to a program called *fluss5* (see Section C.1) which filters those maps not having a 5–flow. These maps are saved in a file called `no5`.

As for the maps of genus 2, the cubic maps of genus 3 containing only heptagons are of special interest in chemistry. They are obtained by the program call

```
CGF -g 3 -v 56 -f 7
```

where the estimated running time of this process on the reference cluster is approximately 35 days. The geometrical properties of such maps are investigated by Prof. R. Bruce King of the University of Georgia (see e.g. [51]).

Non–Hamiltonian maps

As mentioned in Section 1.8, $G(4,7)$ contains a non–Hamiltonian map of order 98. It is not known if this map is the smallest non–Hamiltonian map in $G(4,7)$. With the help of *CGF*, all maps of $G(4,7)$ containing up to 80 vertices have been constructed and tested for Hamiltonicity. Every constructed map was Hamiltonian, thus it can be stated:

- The smallest non–Hamiltonian map in $G(4,7)$ contains 86, 92, or 98 vertices.

This follows from the fact that due to formula 1.3, $G(4,7)$ contains no map of order 82, 84, 88, 90, 94, or 96.

Let $G_n(4,7)$ be the set of all maps in $G(4,7)$ containing n vertices. The maps of $G_n(4,7)$ can be (and have been) constructed and tested using the program call

```
CGF -v n -f 4 -f 7 -g 0 -td_limits c 10 f 13 b 1 -facecon 3 -output stdout |
    gconv -f graph6 | cubhamg6 > nonham
```

(see appendix A) where n must be substituted by the actual vertex number. The option `-facecon 3` restricts the construction to polyhedral maps. A cubic planar map is polyhedral if and only if it is 3–connected. Thus only maps that are contained in $G_n(4,7)$ are obtained. The programs *gconv* and *cubhamg6* are described in Section C.1. Finally, the file `nonham` obtains the constructed non–Hamiltonian maps.

Table 10.4 shows the computed results. Column A lists the number of non–Hamiltonian maps in $G_n(4,7)$. "Time" is the running time on an Intel Pentium II computer with 350MHz. For $n = 80$, the process has been parallelized so that the precise running time cannot be given. For $n = 86$, the estimated running time is approximately 12 days on the reference cluster, and for $n = 92$, the estimated running time is approximately 1 year on the reference cluster.

n	$\lvert G_n(4,7)\rvert$	A	time
8	1	0	0.0s
14	1	0	0.0s
20	1	0	0.0s
26	3	0	0.3s
32	7	0	4.8s
38	16	0	39.5s
44	80	0	42.2s
50	270	0	48.0s
56	1 352	0	796.9s
62	6 359	0	3319.4s
68	33 783	0	36.0h
74	176 962	0	98.3h
80	975 823	0	

Table 10.4: computations for $G(4,7)$

Face–regularity

In order to investigate the two cases enumerated in Section 1.8, $\mathcal{M}(1,n,\{5,7\},b_l,b_u)$ and $\mathcal{M}(1,n,\{3,7\},b_l,b_u)$ with $b_l(5)=b_l(7)=0$ and $b_u(5)=b_u(7)=\infty$ have been computed for $n \leq 40$. It has been tested whether the constructed maps are face–regular or *weakly face–regular*: A map M is **weakly face–regular** if and only if there exists some $d \in \mathbb{N}$ such that M contains at least one d–gon and every d–gon in M has the same number of neighbours of degree i' for every i'.

The results of the computations are shown in Table 10.5 and in Table 10.6. "Time" is the running time on an Intel Pentium II computer with 350MHz. $\mathcal{M}(1,n,\{5,7\},b_l,b_u)$ can be (and has been) constructed and tested for face–regularity by the program call

```
CGF -v n -f 5 -f 7 -g 1 -topdown -memory 14000000 -output stdout |
    faceregular > faceregmaps
```

(see appendix A) where n must be substituted by the actual vertex number. The program *face-regular* is described in Section C.1. If the maps are to be tested for weak face–regularity, then the option `w` must be inserted after the program's name `faceregular`. The file `faceregmaps` obtains the constructed (weakly) face–regular maps.

$\mathcal{M}(1,n,\{3,7\},b_l,b_u)$ can be (and has been) constructed and tested by the same program call as above where the option `-f 5` must be substituted by the option `-f 3`. However, the face–regular elements of $\mathcal{M}(1,n,\{3,7\},b_l,b_u)$ can more easily be derived from hexagonal lattices:

Theorem 10.1 *Let $M \in \mathcal{M}(1,n,\{3,7\},b_l,b_u)$ be face–regular. Then every triangle in M is incident with three heptagons, and every heptagon is incident with one triangle and six*

n	$\lvert\mathcal{M}(1, n, \{5, 7\}, b_l, b_u)\rvert$	face–regular maps	weakly face–regular maps	time
8	1	1	1	0.0s
12	1	1	1	0.0s
16	8	5	6	0.1s
20	3	0	0	0.3s
24	43	8	15	1.8s
28	17	0	0	8.3s
32	304	15	24	32.4s
36	229	3	3	146.0s
40	2698	10	20	1132.7s

Table 10.5: computations for $\mathcal{M}(1, n, \{5, 7\}, b_l, b_u)$ with $b_l(5) = b_l(7) = 0$ and $b_u(5) = b_u(7) = \infty$

n	$\lvert\mathcal{M}(1, n, \{3, 7\}, b_l, b_u)\rvert$	face–regular maps	weakly face–regular maps	time
8	1	1	1	0.0s
16	6	3	6	0.1s
24	5	3	3	1.8s
32	153	14	93	140.9s
40	219	5	79	6h 16m

Table 10.6: computations for $\mathcal{M}(1, n, \{3, 7\}, b_l, b_u)$ with $b_l(3) = b_l(7) = 0$ and $b_u(3) = b_u(7) = \infty$

heptagons.

Proof: Since M is face–regular, every i–gon is incident with the same number of j–gons for every $i, j \in D$. Let $c_{i,j}$ be this number. We must show $c_{3,7} = 3$ and $c_{7,3} = 1$. It clearly holds $c_{3,7} \in \{0, 1, 2, 3\}$. Let x be the number of edges in M that are each incident with a triangle and a heptagon. It holds $c_{3,7} \cdot p_3(M) = x = c_{7,3} \cdot p_7(M)$. Formula 1.3 implies $3p_3(M) = p_7(M)$ so that $c_{3,7} = 3c_{7,3}$. Together with $c_{3,7} \in \{0, 1, 2, 3\}$, we obtain $c_{3,7} \in \{0, 3\}$. If $c_{3,7} = 0$, then no triangle in M is incident with any heptagon. But since M is connected, this is impossible. Thus $c_{3,7} = 3$ and hence $c_{7,3} = 1$. ■

Now since $c_{7,3} = 1$, we can transform M into a hexagonal lattice by contracting every edge which is incident with a triangle. Or with the terminology of part I of this text: We can transform M into a hexagonal lattice by reducing every triangle applying ${R_1}^-$. Thus by modifying *togen7* such that R_1 is the only generating rule which may be applied, one obtains an algorithm that can be used to construct the face–regular elements of $\mathcal{M}(1, n, \{3, 7\}, b_l, b_u)$.

Tilings of the sphere

CGF has been used to fill a gap in Table 1 in [43], namely the gap for $n = 11$. With the terminology of [43], the task was to find all smallest $I(5, 11)$–fulleroids. From the combinatorial point of view, an $I(5, 11)$–fulleroid is a 3–connected planar cubic map whose faces are all pentagons or 11–gons and which has the same orientation preserving symmetries as the icosahedron. Let M' be the combinatorial representation of a smallest possible $I(5, 11)$–fulleroid, i.e. M' is a planar cubic map. Applying the argumentation of [42] (in particular page 92) and [43] to $I(5, 11)$–fulleroids, one obtains the following results:

- Since no element of the symmetry group of the icosahedron except the identity element can fix an 11–gon, $p_{11}(M')$ is a multiple of the order of the group, i.e. $p_{11}(M') = 60 \cdot c$ for some $c \in \mathbb{N}$.
- We assume $p_{11}(M') = 60$. Then due to formula 1.3, it holds $p_5(M') = 5p_{11}(M') + 12 = 312$. Thus $f_{M'} = 372$ and formula 1.2 implies $n_{M'} = 740$.[1]
- The 11–gons of M' contribute $60 \cdot 22$ vertices to the corresponding flag graph F and the 5–gons contribute $312 \cdot 10$ vertices to F. Thus, the order of F is $312 \cdot 10 + 60 \cdot 22 = 4440$. As the symmetry group I acts fixed–point free on those flags, the associated Delaney graph contains exactly $\frac{4440}{60} = 74$ vertices.
- A Delaney graph Δ is given by a quadruple $\Delta = (M, E_0, E_1, E_2)$ where M is a map and the edges of the underlying graph of M are partitioned into 3 disjoint sets E_0, E_1, and E_2. M is called the *underlying map* of Δ. We call the elements of E_0 *red edges* or 0*–edges*, the elements of E_1 *green edges* or 1*–edges*, and the elements of E_2 *blue edges* or 2*–edges*. An (i, j)*–face* ($i, j \in \{0, 1, 2\}$) in Δ is a face of M where each incident edge is either an i–edge or a j–edge. Tracing the face, one traces i–edges and j–edges alternatingly.

  Let now $\Delta = (M, E_0, E_1, E_2)$ correspond to M'. Then it holds:
  - $n_M = 74$
  - Δ contains exactly one $(0, 1)$–face of degree 22, representing the 11–gons.
  - Δ contains exactly one $(0, 1)$–face of degree 2, exactly one $(0, 2)$–face of degree 2, and exactly one $(1, 2)$–face of degree 2.
  - All remaining $(0, 2)$–faces are squares, all remaining $(1, 2)$–faces are hexagons, and the

[1] At this point, it is clear why we do not try to find the smallest $I(5, 11)$–fulleroids by constructing $\mathcal{M}(0, 740, \{5, 11\}, b_l, b_u)$ (with $b_l(5) = b_l(11) = 0$ and $b_u(5) = b_u(11) = \infty$) and filtering those maps which have the appropriate automorphism group: The order 740 is just too high.

50 vertices not contained in the $(0,1)$–faces of degree 2 and 22 are contained in five more $(0,1)$–faces of degree 10 representing the 300 pentagons which have no internal symmetry. Then formula 1.3 implies that Δ contains exactly 18 squares. It follows that Δ contains exactly 12 hexagons.

Now *CGF* can be used to construct $\mathcal{M}(0, 74, \{2,4,6,10,22\}, b, b)$ with $b(2) = 3$, $b(4) = 18$, $b(6) = 12$, $b(10) = 5$, and $b(22) = 1$. The accepted maps might be the underlying maps of the desired Delaney graphs. In this case, the edges can be coloured as follows:

- Each vertex is incident with a red edge, a green edge, and a blue edge.
- One digon contains a red edge and a green edge, one digon contains a red edge and a blue edge, and one digon contains a green edge and a blue edge.
- The squares contain red edges and blue edges.
- The hexagons contain green edges and blue edges.
- The 10–gons contain red edges and green edges.
- The 22–gon cotains red edges and green edges.

Alternatively, the faces can be coloured with the missing colour as described in Section 8.5. Thus if an accepted map is the underlying map of a desired Delaney graph, then the faces can be coloured such that one digon is blue, one digon is green, one digon is red, the squares are green, the hexagons are red, and the 10–gons and the 22–gon are blue. Every vertex is incident with a red face, a green face, and a blue face. In order to perform the corresponding tests during the construction (compare Section 8.5), the appropriate program call is

```
CGF -g 0 -v 74 -delaney -f 2 123 [3-3] -f 4 2 [18-18] -f 6 1 [12-12]
    -f 10 3 [5-5] -f 22 3 [1-1] -facecon 23 -td_limits c 12 f 37 b 1
    -patchcheck
```

(see appendix A). However, using this call, all digons may be arbitrarily coloured so that we actually obtain a superset of the desired maps.

Instead of performing the above program call directly, the results of Section 9.8 were applied: Since every constructed map contains a digon, it also contains a sandwich path p with $l_1(p) = l_2(p) = 1$ or a dumbbell path p with $l_1(p) = 2$. So the above program call could be substituted by the two calls

```
CGF -g 0 -v 74 -delaney -f 2 123 [3-3] -f 4 2 [18-18] -f 6 1 [12-12]
    -f 10 3 [5-5] -f 22 3 [1-1] -facecon 23 -td_limits c 12 f 37 b 1
    -patchcheck -types 2 -priority 231 -digon_path
```

and

```
CGF -g 0 -v 74 -delaney -f 2 123 [3-3] -f 4 2 [18-18] -f 6 1 [12-12]
    -f 10 3 [5-5] -f 22 3 [1-1] -facecon 23 -td_limits c 12 f 37 b 1
    -patchcheck -types 3 -priority 321 -digon_path
```

This way, the required time to construct the maps was approximately 10 weeks on the reference cluster (of course, the processes were parallelized). The options act on the construction as follows:

`-patchcheck` invokes the advanced patch testing. This is useful because many patches can be rejected due to the tests described in Section 8.5.

`-facecon 23` restricts the construction to semi–polyhedral maps (compare Section 8.5).

`-types 2` restricts the construction to maps that contain a sandwich path, `-priority 231` assigns a higher quality to sandwich paths than to dumbbell paths, and `-digon_path` restricts the construction to maps that contain a sandwich path p with $l_1(p) = l_2(p) = 1$.

`-types 3` restricts the construction to maps that contain a dumbbell path, `-priority 321` assigns a higher quality to dumbbell paths than to sandwich paths, and `-digon_path` restricts the construction to maps that contain a dumbbell path p with $l_1(p) = 2$.

Altogether, 7 maps have been accepted. Figures 10.7–10.13 show the accepted maps coloured appropriately yielding the Delaney graphs Δ_1–Δ_7. Each of the Delaney graphs contains exactly one $(0,1)$–face of degree 2, exactly one $(0,2)$–face of degree 2, and exactly one $(1,2)$–face of degree 2. The corresponding tilings M_{Δ_1}–M_{Δ_7} are shown (in the same order) in Figures 10.14–10.34 where to each map, a Schlegel diagram, a 3–dimensional spring embedding, and a classical tiling of the sphere is provided. Of course, the 3–dimensional embeddings are not suitable for investigating details of the tilings. They are provided in order to give the reader an impression of how the structures look like if they are interpreted as chemical structures, i.e. the edges have almost equal lengths. In the classical tilings, those faces which are in a common orbit are coloured identically.

It can be seen in Figure 10.14 that M_{Δ_1} is not 3–connected. However, all other maps M_{Δ_2}–M_{Δ_7} are 3–connected and thus represent smallest $I(5,11)$–fulleroids. Moreover, as it can be seen in Figure 10.20, M_{Δ_4} is even an $I_h(5,11)$–fulleroid (in Figure 10.31, the specular symmetry is destroyed). Thus it is shown:

Theorem 10.2 *There exist exactly* 6 *smallest* $I(5,11)$*–fulleroids and exactly one smallest* $I_h(5,11)$*–fulleroid.*

The first result is quite surprising because as it is shown in Table 1 in [43], for all $n > 6$ where the number x_n of smallest $I(5,n)$–fulleroids is known, x_n is much smaller than x_{11}, namely $x_7 = 2$ and $x_n = 1$ for $n \in \{8,9,10,12,14,15\}$.

For $n \in \{16,18,20\}$, determining x_n using *CGF* is probably easier than determining x_{11} because the corresponding Delaney graphs contain only 68, 54, and 38 vertices, respectively. However, for $n \in \{13,17,19\}$, determining x_n would take more time than determining x_{11}.

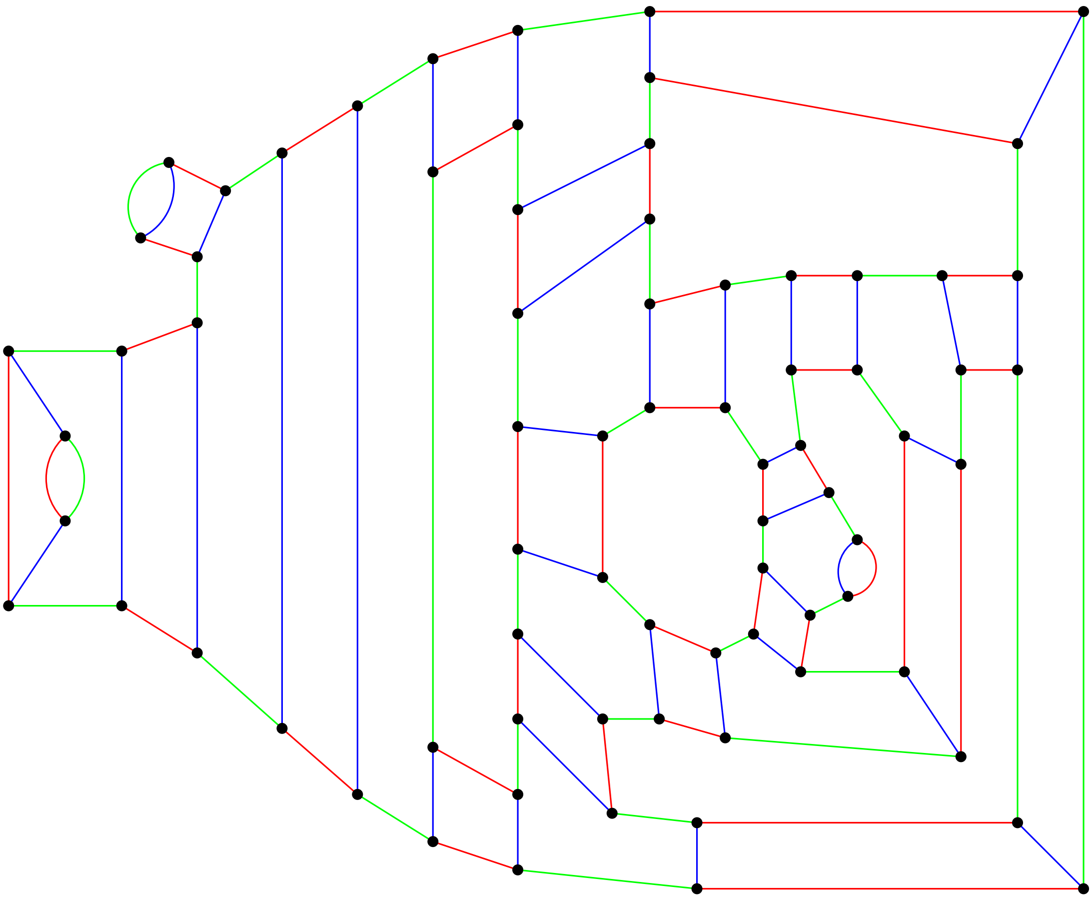

Figure 10.7: Delaney graph Δ_1

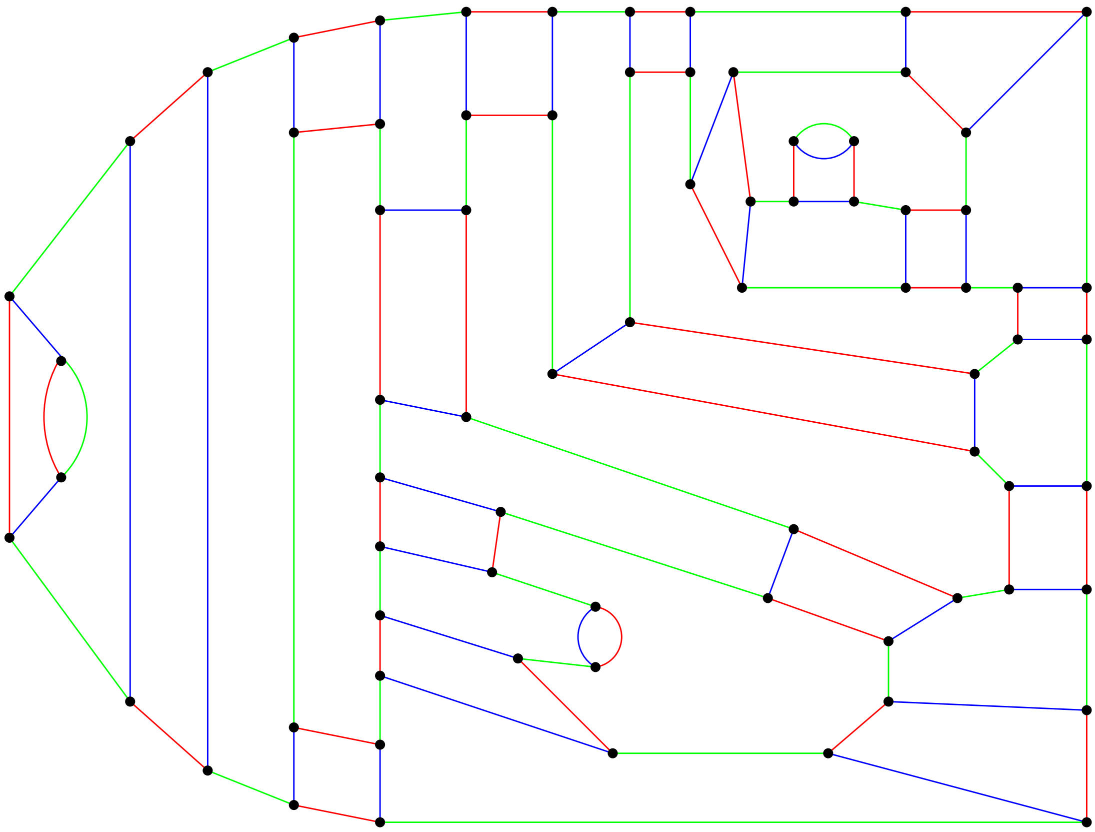

Figure 10.8: Delaney graph Δ_2

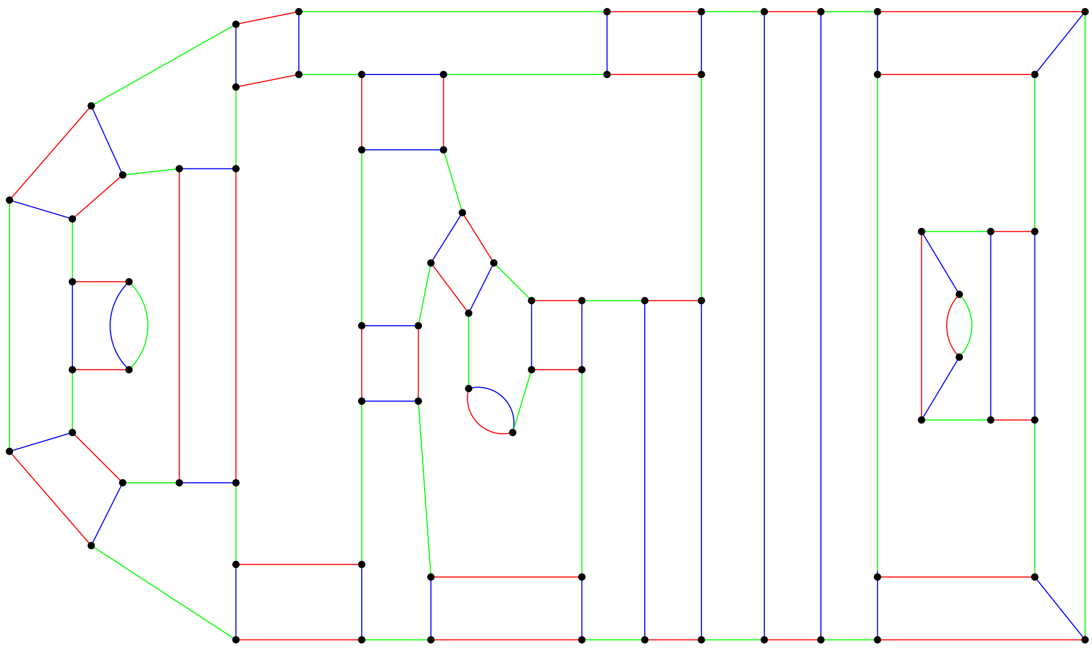

Figure 10.9: Delaney graph Δ_3

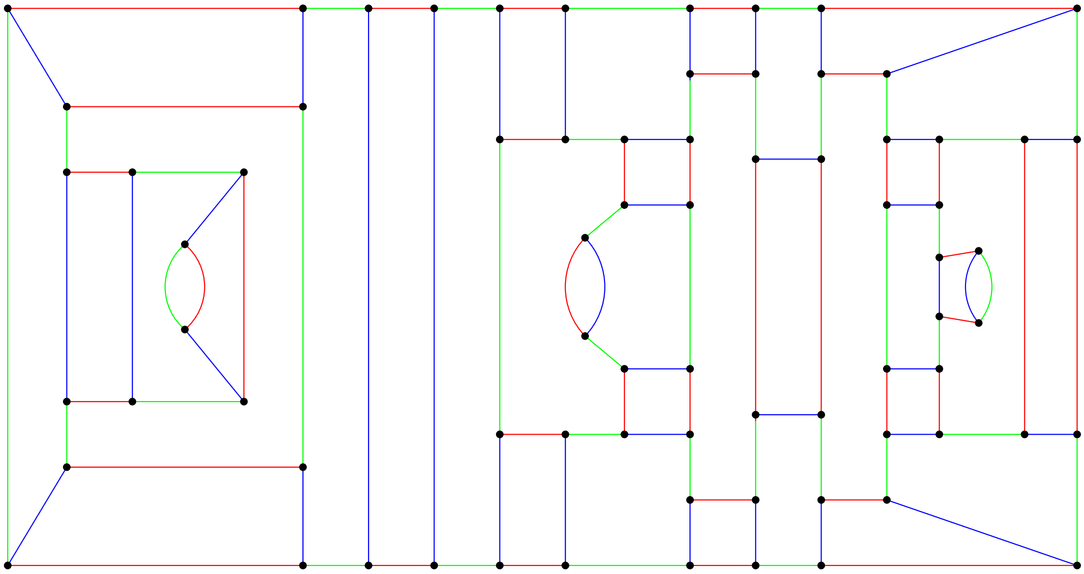

Figure 10.10: Delaney graph Δ_4

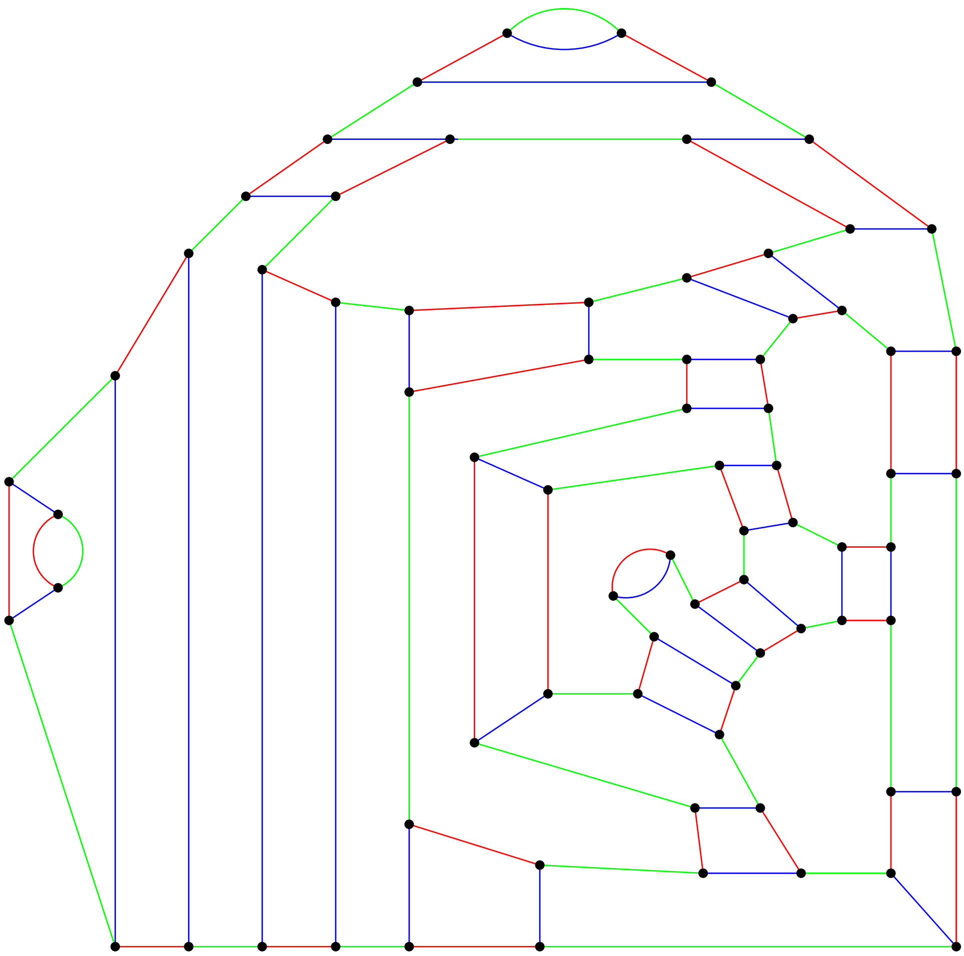

Figure 10.11: Delaney graph Δ_5

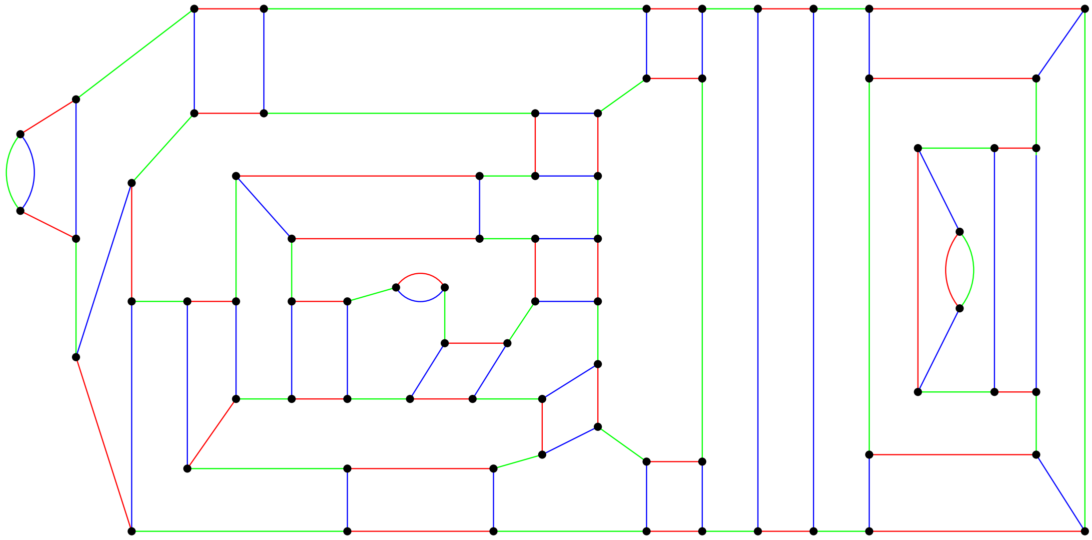

Figure 10.12: Delaney graph Δ_6

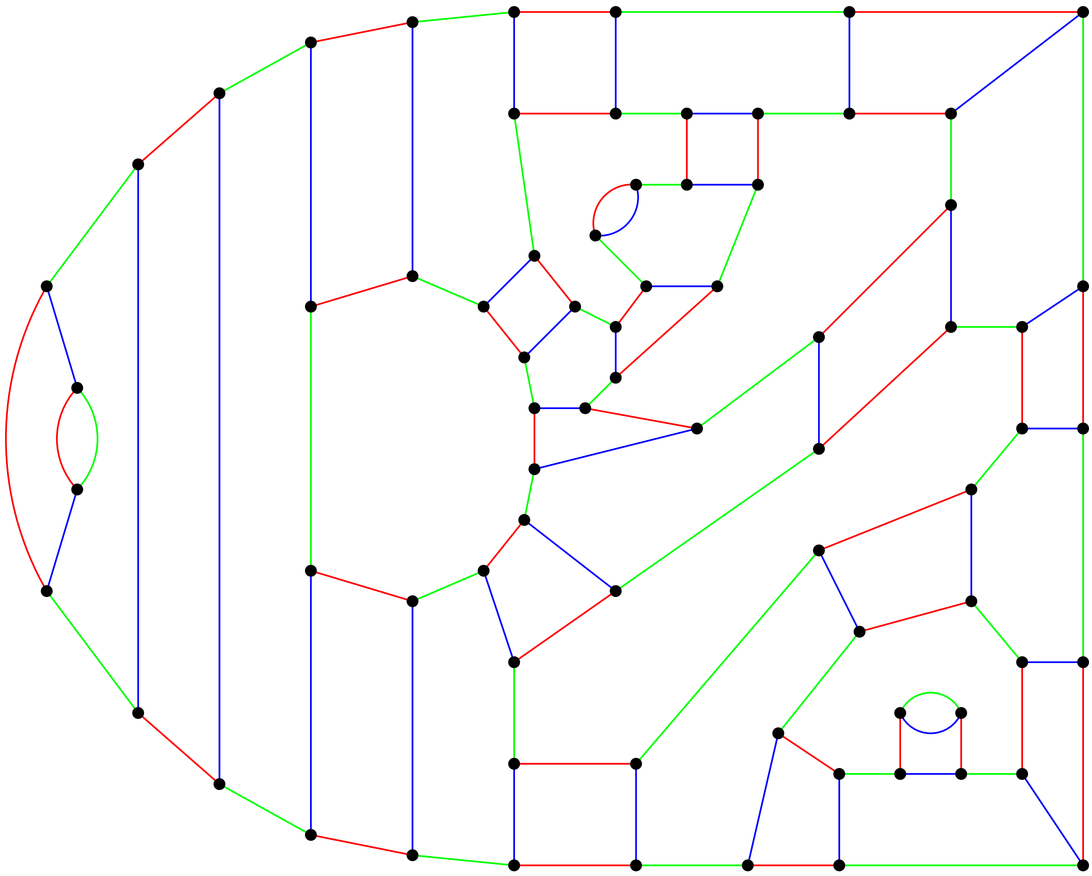

Figure 10.13: Delaney graph Δ_7

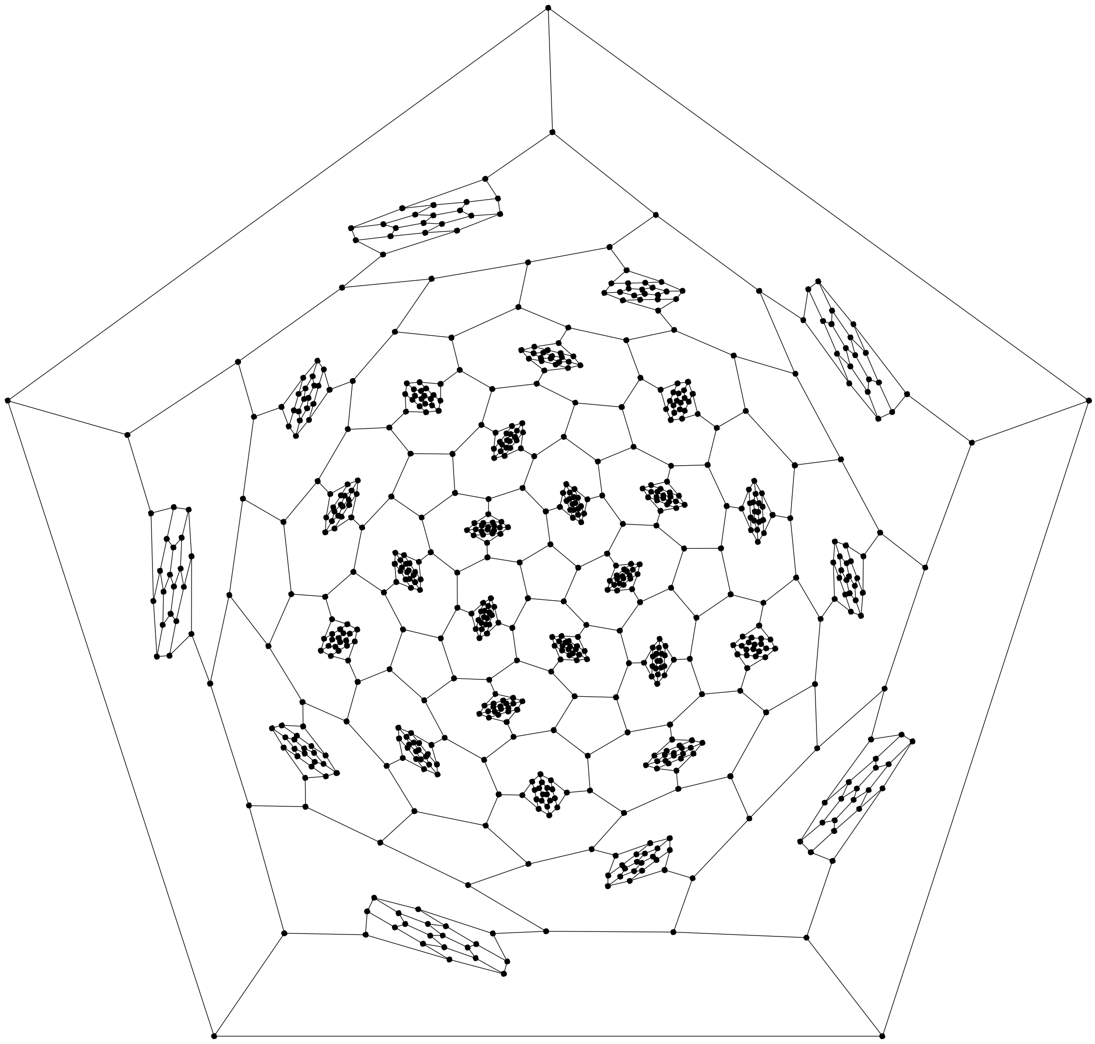

Figure 10.14: a Schlegel diagram of M_{Δ_1}

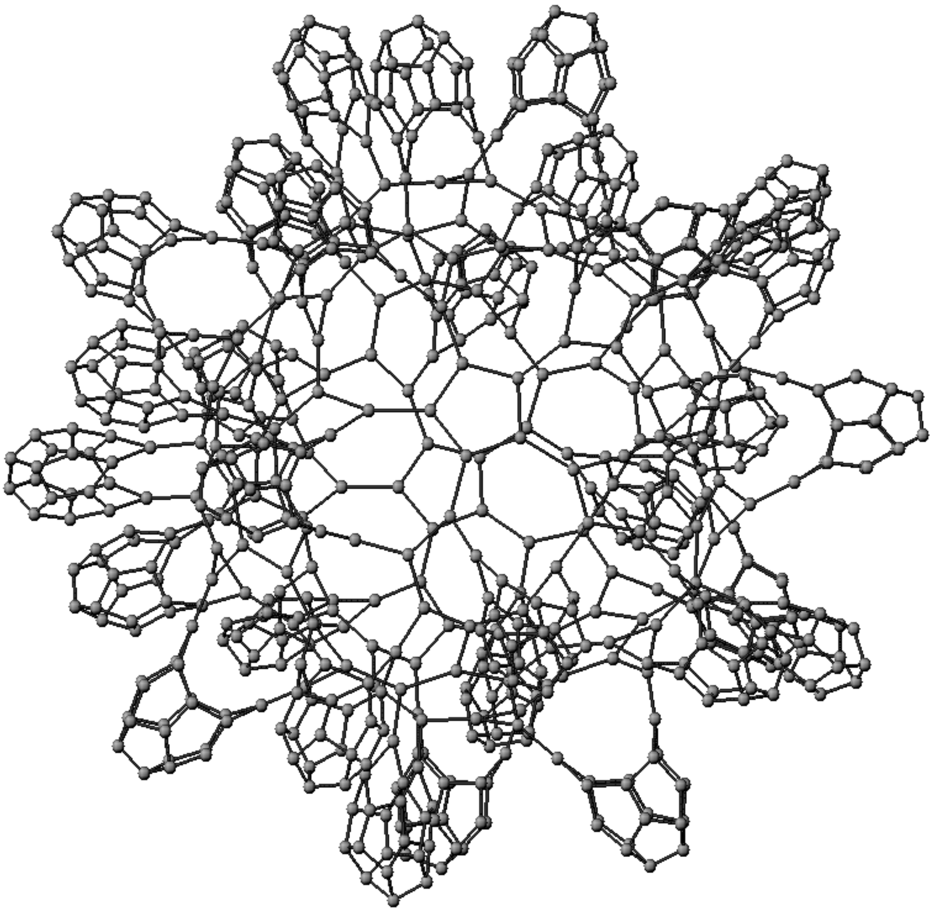

Figure 10.15: a 3–dimensional embedding of M_{Δ_1}

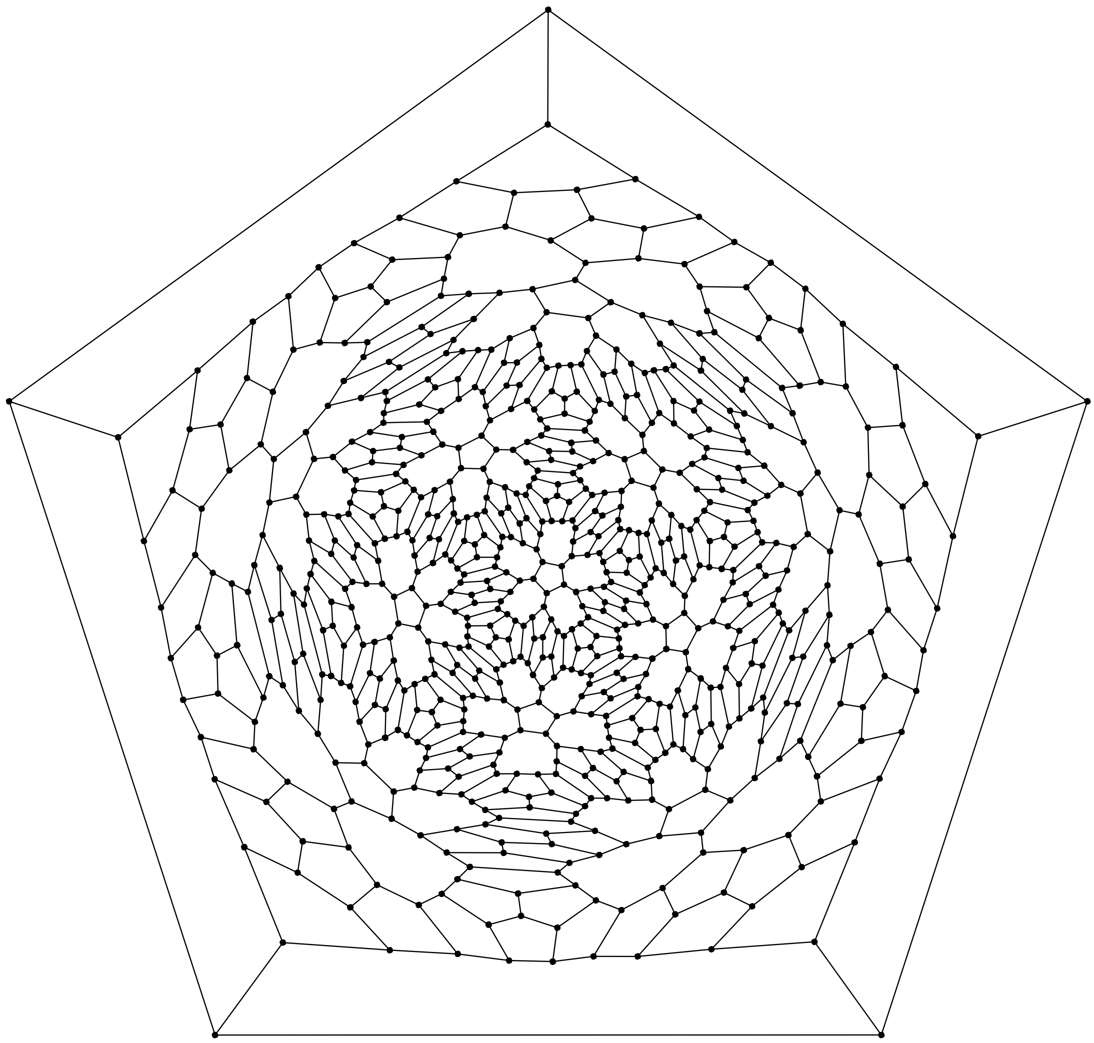

Figure 10.16: a Schlegel diagram of M_{Δ_2}

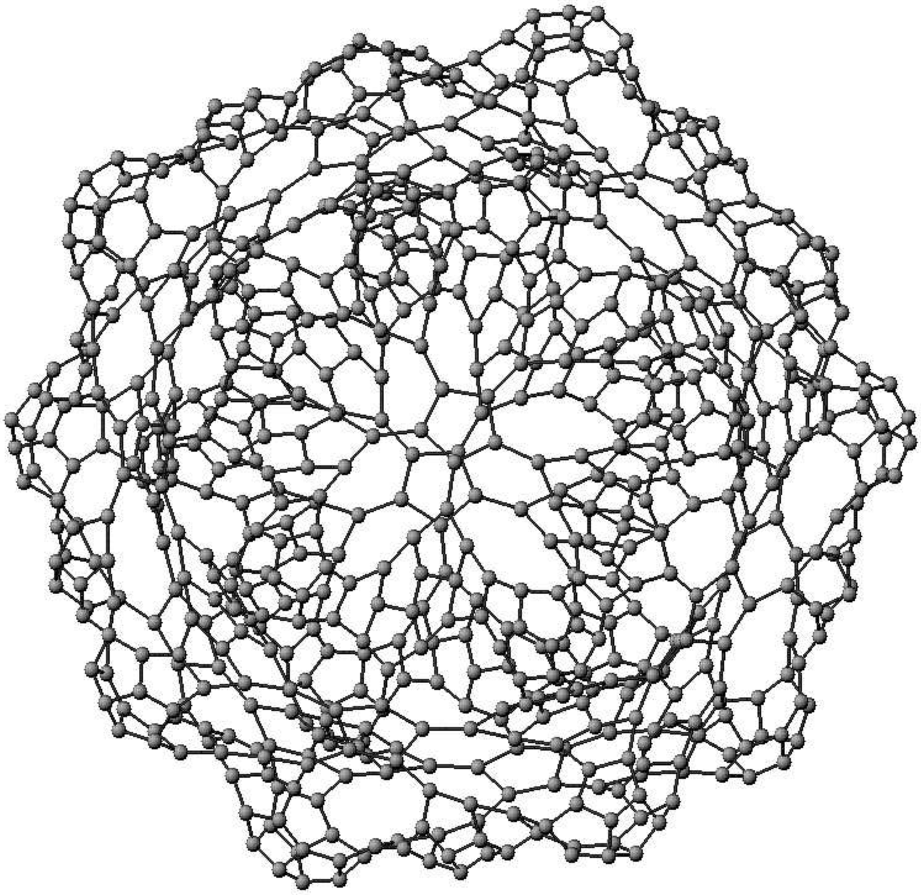

Figure 10.17: a 3–dimensional embedding of M_{Δ_2}

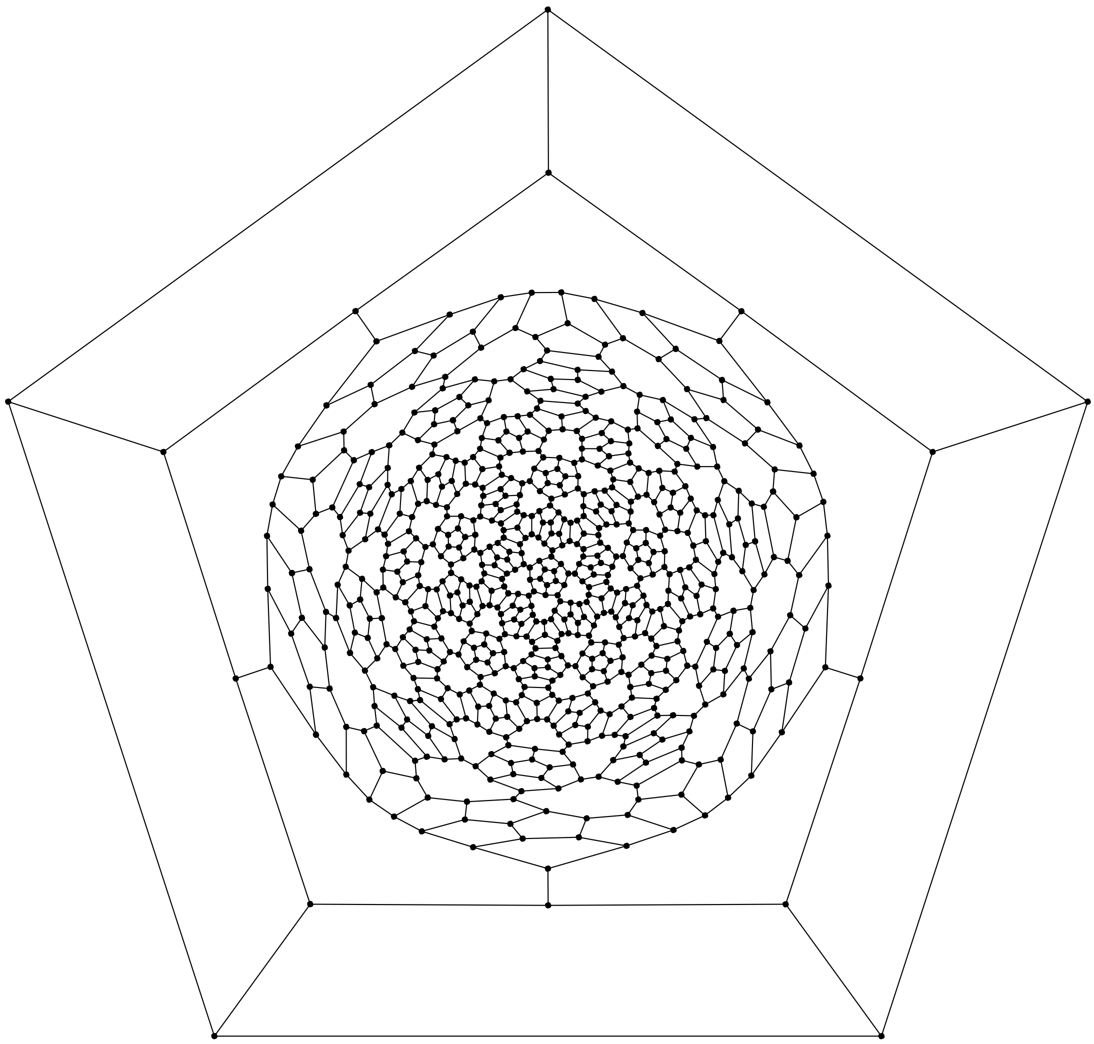

Figure 10.18: a Schlegel diagram of M_{Δ_3}

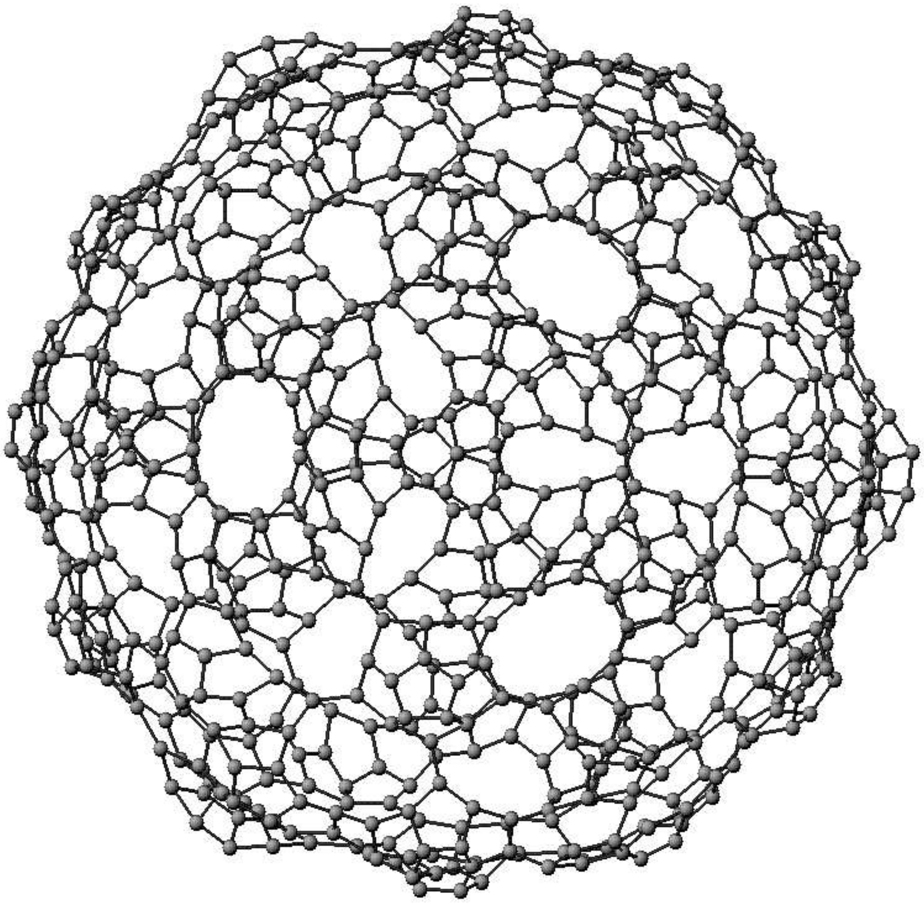

Figure 10.19: a 3–dimensional embedding of M_{Δ_3}

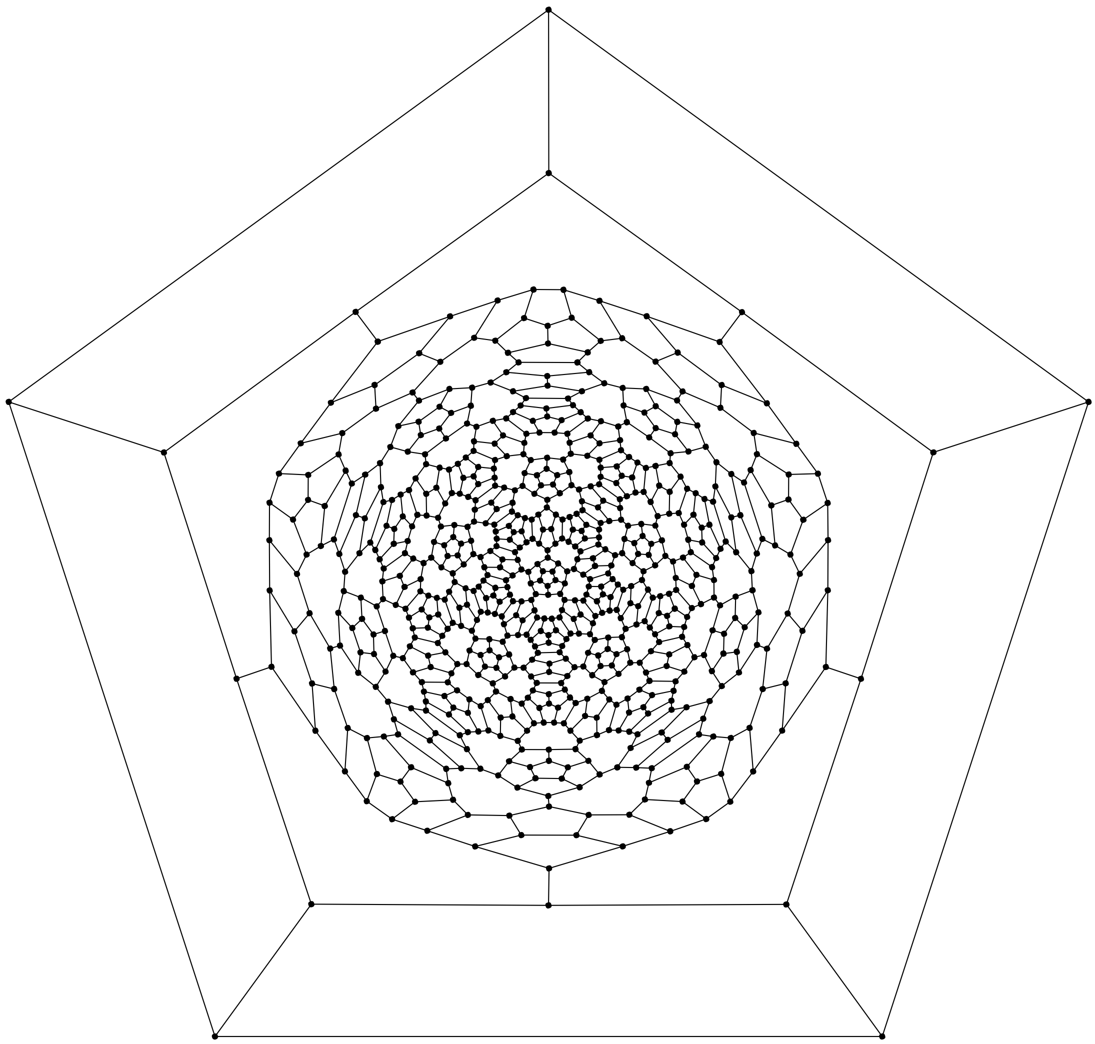

Figure 10.20: a Schlegel diagram of M_{Δ_4}

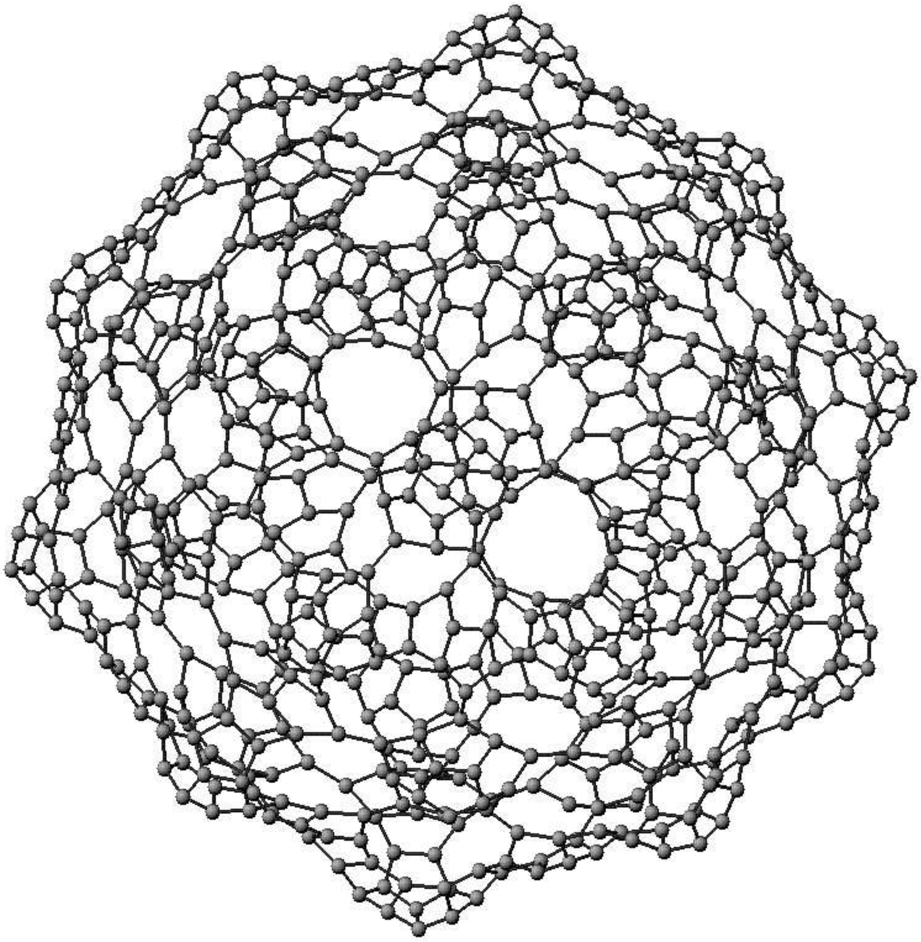

Figure 10.21: a 3–dimensional embedding of M_{Δ_4}

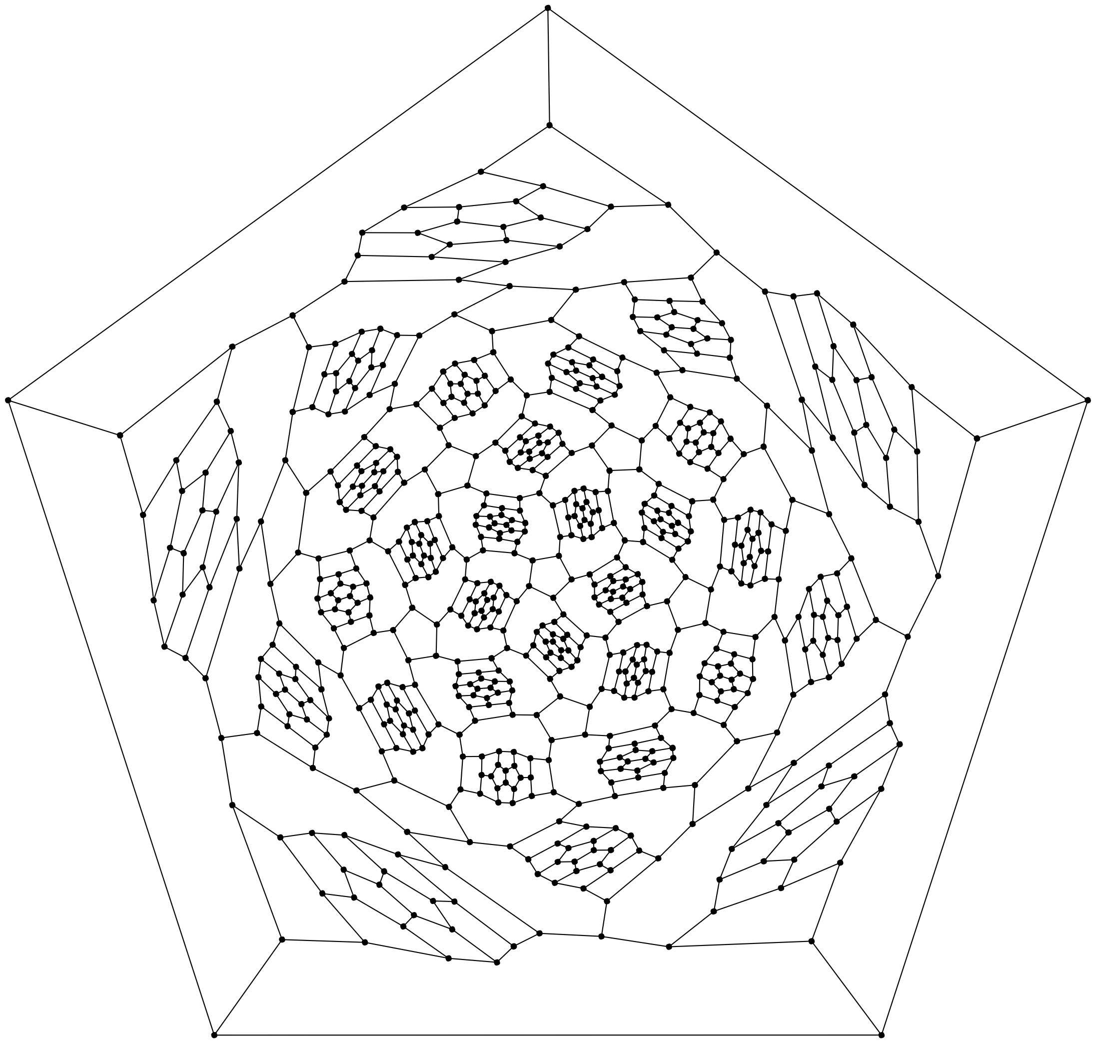

Figure 10.22: a Schlegel diagram of M_{Δ_5}

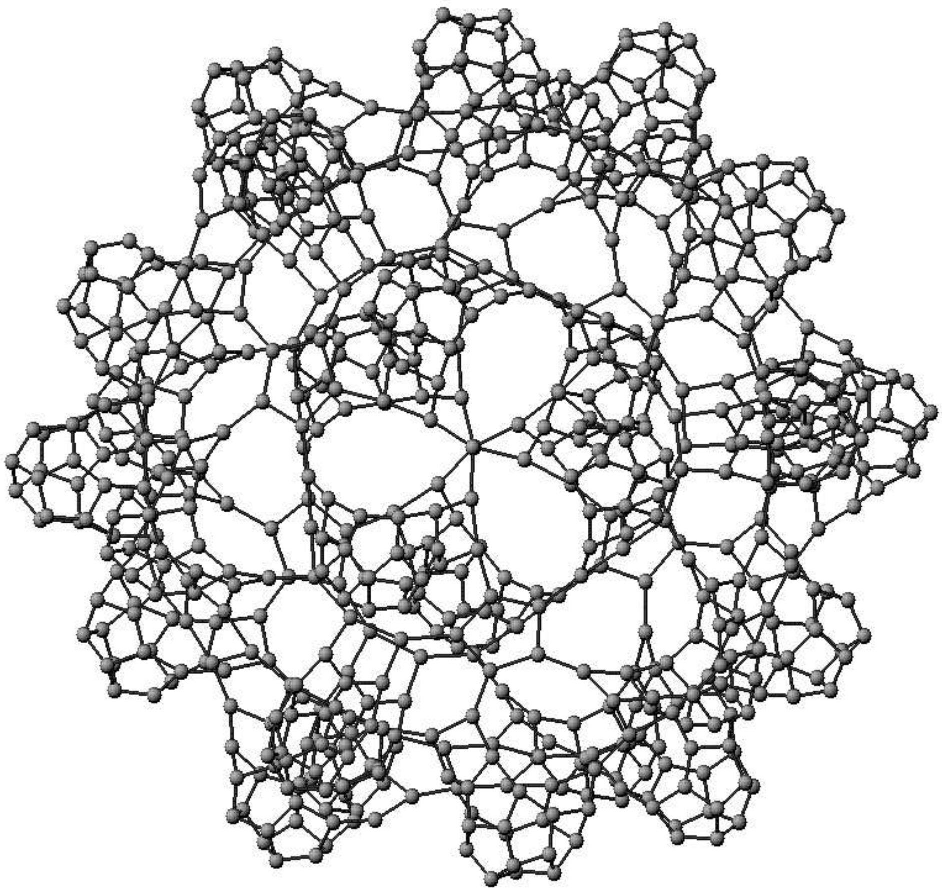

Figure 10.23: a 3–dimensional embedding of M_{Δ_5}

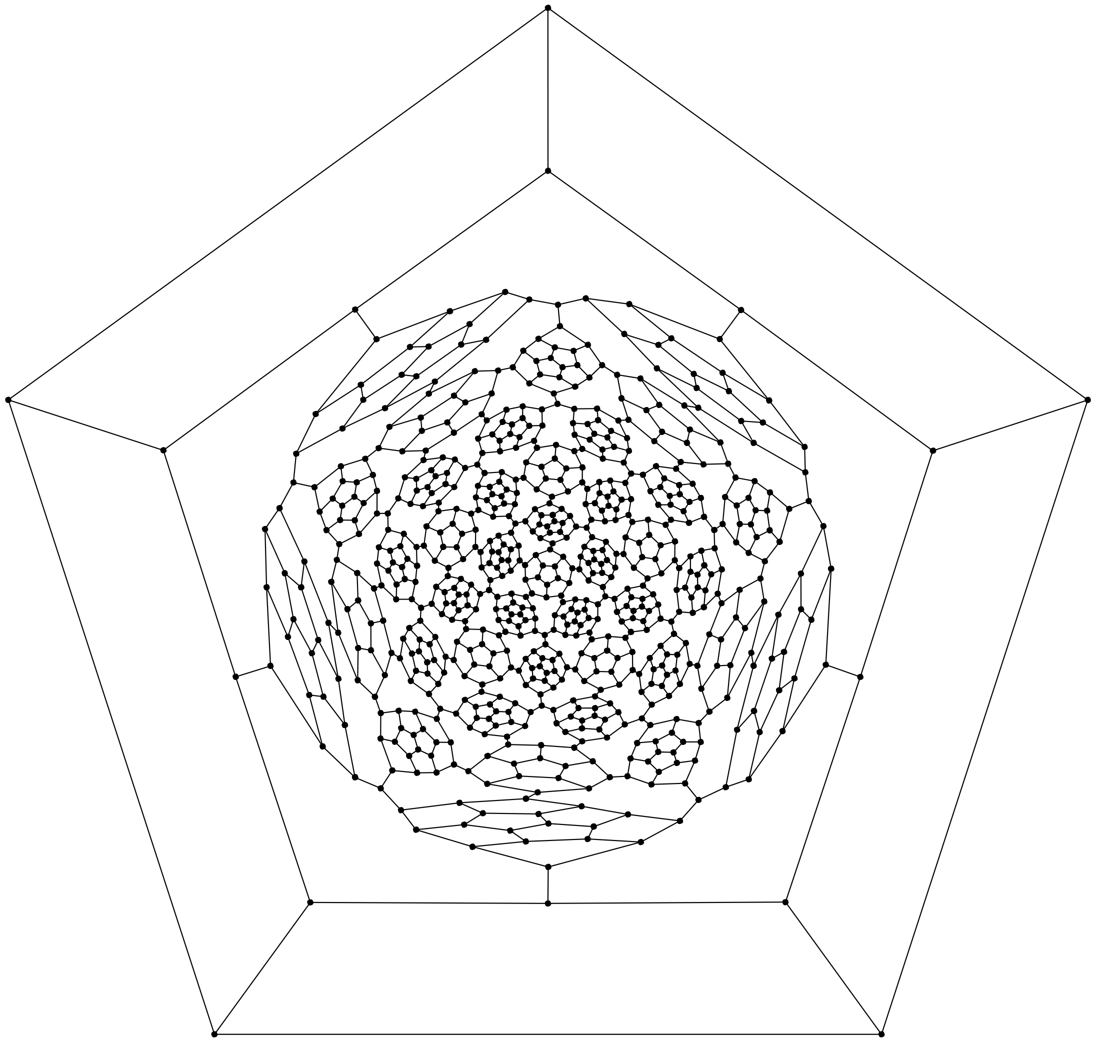

Figure 10.24: a Schlegel diagram of M_{Δ_6}

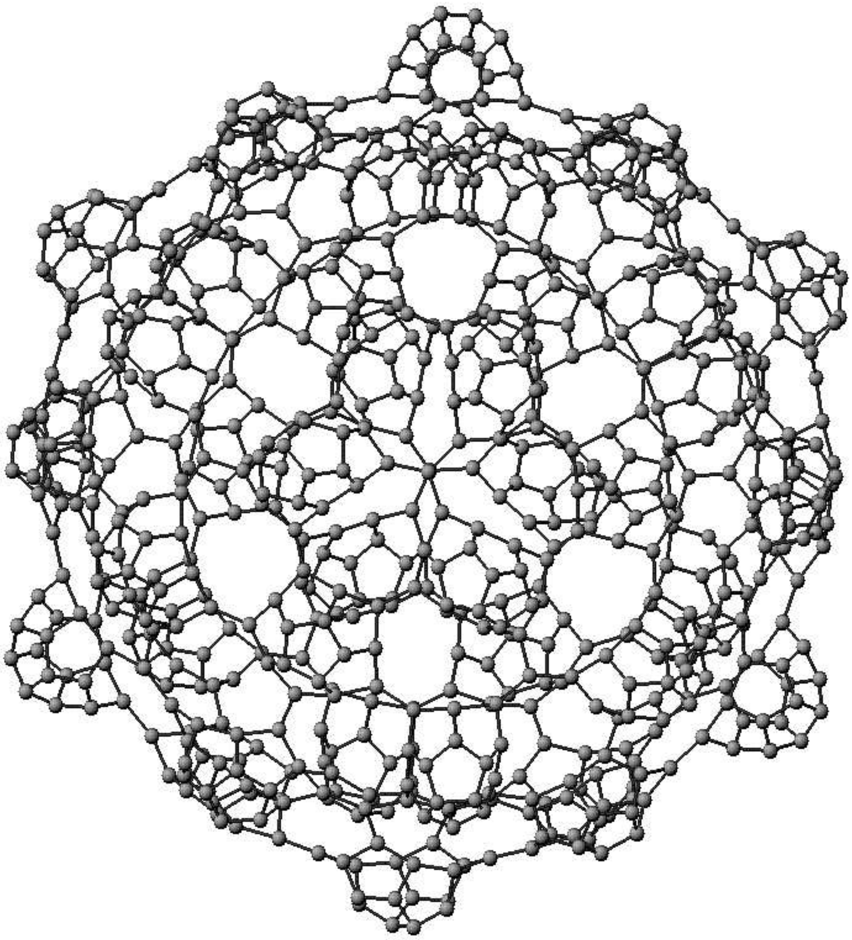

Figure 10.25: a 3–dimensional embedding of M_{Δ_6}

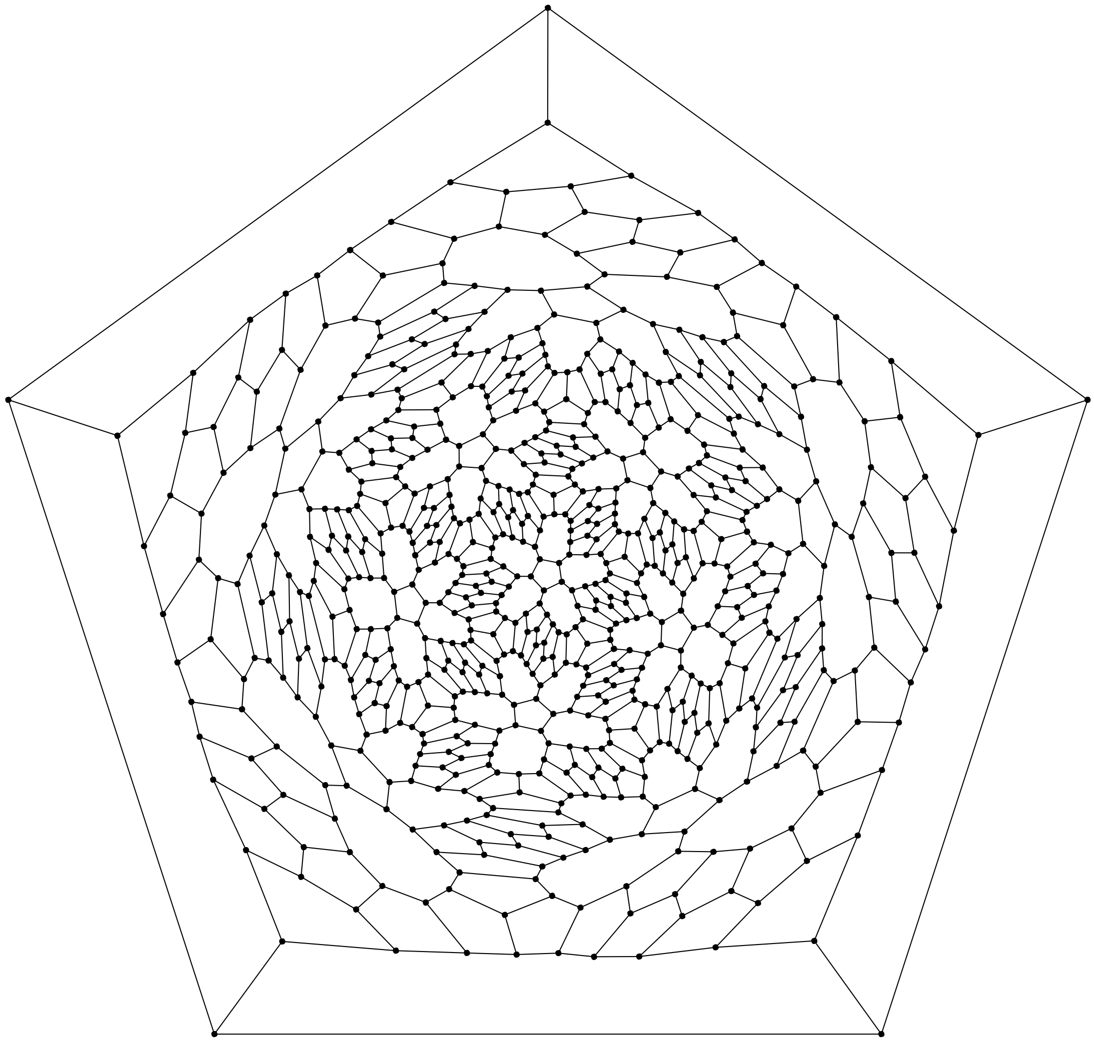

Figure 10.26: a Schlegel diagram of M_{Δ_7}

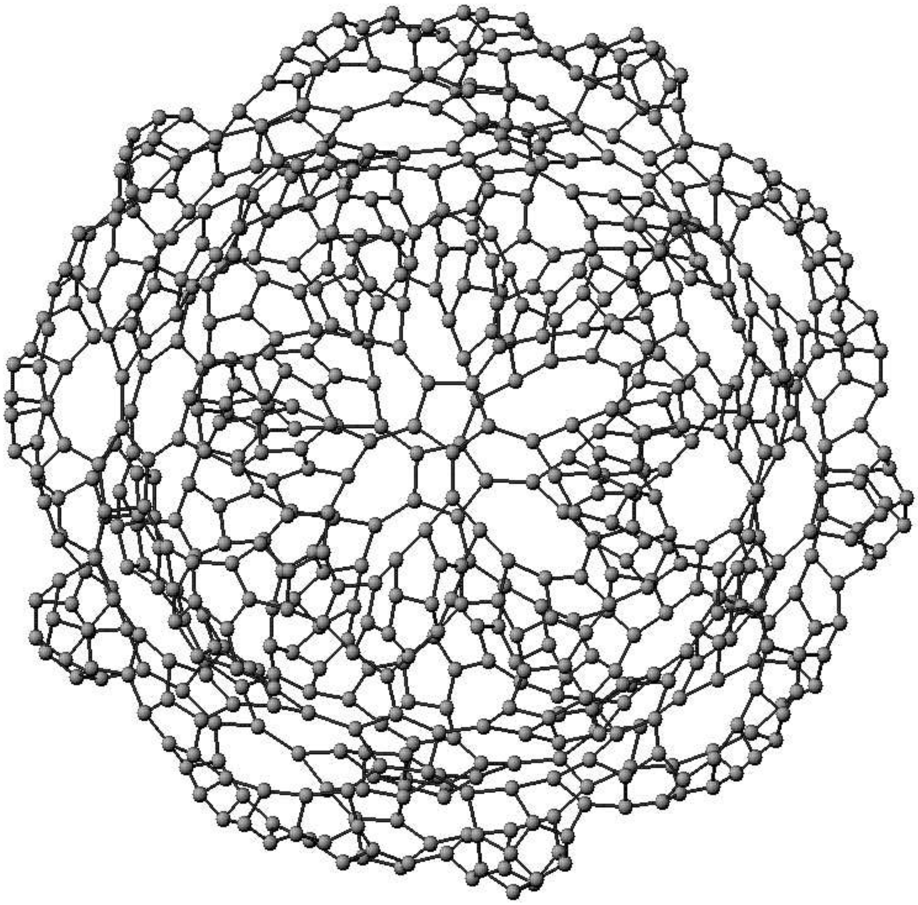

Figure 10.27: a 3–dimensional embedding of M_{Δ_7}

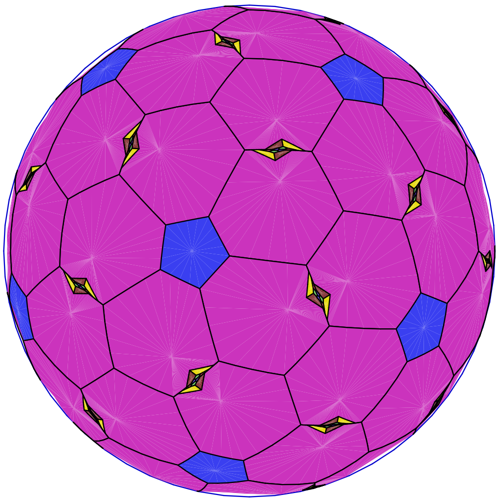

Figure 10.28: a tiling for M_{Δ_1}

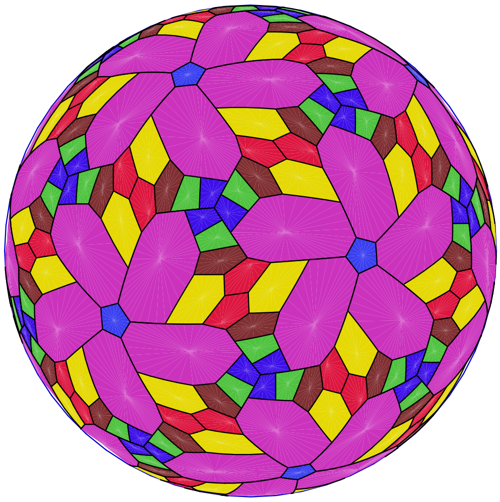

Figure 10.29: a tiling for M_{Δ_2}

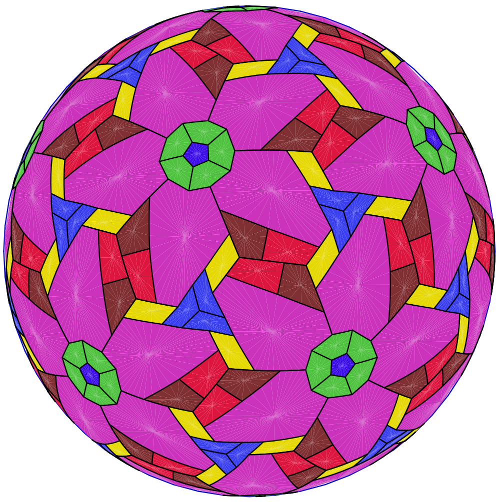

Figure 10.30: a tiling for M_{Δ_3}

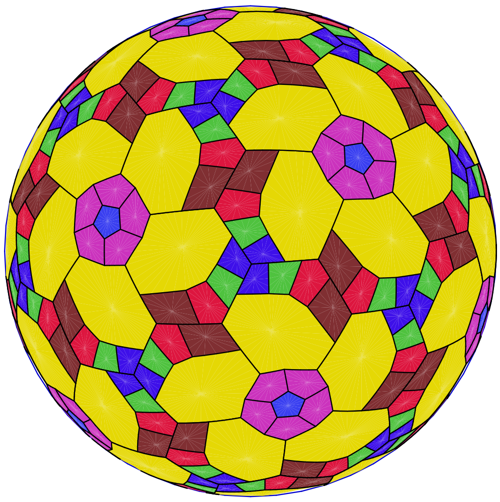

Figure 10.31: a tiling for M_{Δ_4}

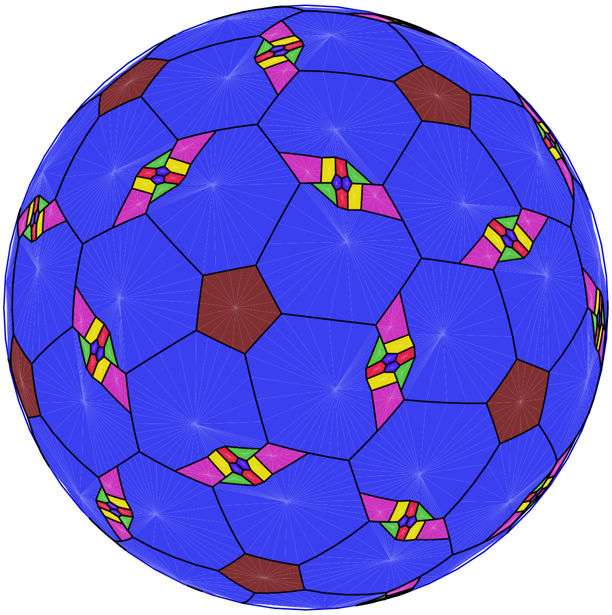

Figure 10.32: a tiling for M_{Δ_5}

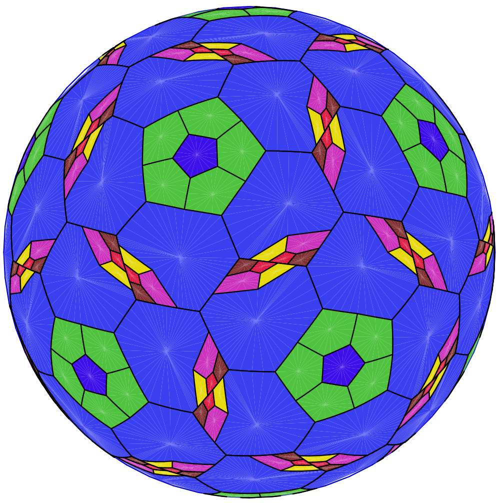

Figure 10.33: a tiling for M_{Δ_6}

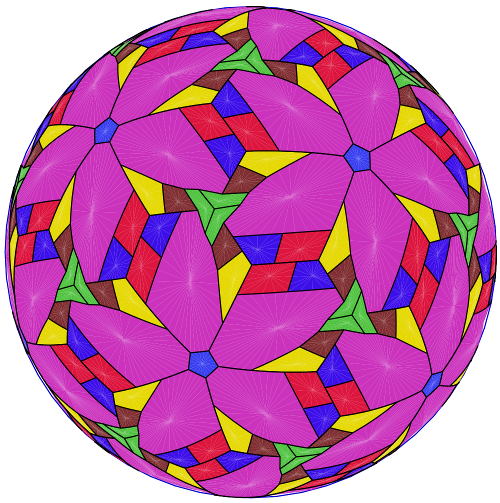

Figure 10.34: a tiling for M_{Δ_7}

Part III

Appendix

Appendix A

The program *CGF*

In order to start the program *CGF*, the program's name followed by several options must be entered into the command line. Each option begins with a minus sign "`-`". Some options require further parameters, some options do not. An example call is

```
CGF -v 52 -g 0 -f 5 -f 7
```

which causes *CGF* to construct $\mathcal{M}(0, 52, \{5, 7\}, b_l, b_u)$ with $b_l(5) = b_l(7) = 0$ and $b_u(5) = b_u(7) = \infty$. If multiple edges are not allowed, then *CGF* provides a *vertex code* of every accepted map as output. If multiple edges are allowed, then *CGF* provides an *edge code* of every accepted map as output. In both cases, a map is encoded with respect to the start edge of the canonical Petrie path. The codes are written one after another. The whole output begins with a header

```
>>embed_code le<<
```

which enables other programs to recognize the code. In case of *cubic* maps, an edge code can easily be distinguished from a vertex code: If n is the order of a cubic map, then in the vertex code, the entries vary from 1 to n, while in the edge code the entries vary from 1 to $\frac{3}{2}n$. However, in order to keep the code universal for *every* map, the *edge* code of a map is preceded by the sequence $\langle 1, 2 \rangle$ (*before* the order of the map is listed). A vertex code cannot start with this sequence of numbers because if 1 is the number of vertices, then 2 cannot be the number of a vertex. Thus a vertex code can be distinguished from an edge code by reading the first two entries.

A.1 Output files

Every *CGF* process creates the following output files:

- A **code file** containing the codes of the accepted maps. The suffix of this file is "`.ec`". If several vertex numbers n and/or genera g are permitted for the constructed maps, then an own code file is created for every pair (g, n).

- A **log file** containing the comments of *CGF*. The suffix of this file is "`.log`".
- A **dump file** containing data that can be used to recover a process that has got lost (e.g. due to a computer hangup). The suffix of this file is "`.dump`".
- A backup of the previous dump file whose suffix is "`.dump~`".
- A **warn file** whose suffix is "`.dump.bsy`". This file is created immediately before the dump file is updated, and it is removed immediately after the saving of the new dump file has finished. So if a process has got lost and the warn file exists, then the process has got lost while it was saving the dump file. This case will occur very rarely. When it occurs, the dump file is probably corrupt and one should use the backup of the previous dump file in order to recover the process. This is done automatically by *CGF* if a backup exists.

The bodies of the file names depend on the options that are used for the process. The naming of the files will be discussed in Section A.4.

By default, the dump file is updated every 30 minutes. Moreover, it must be updated whenever codes are written into the code file. Otherwise the same codes would be written once more in case the process gets lost and becomes recovered. To this end, *CGF* collects some codes before it writes them all into the code file.

When a lost process is recovered, then the new codes will be appended to the old code file and the new comments will be appended to the old log file. When a process gets lost while it is writing codes to the code file (so that the warn file remains on the disk), then — as said above — the backup of the previous dump file should be used to recover the process. In this case, the codes that have already been written will be written once more so that the code file contains isomorphic maps. In order to detect the isomorphic maps, one can rename the old code file before the lost process is recovered. Then after the recovered process has finished, there exist an old code file and a new code file where the first few codes in the new code file might be equal to the last few codes in the old code file. This is easy to find out.

A.2 Example log file

When *CGF* is called as follows

```
CGF -g 1 -f 5 -f 6 -f 7 -v 24 -output none -save 0 -mapstat
    -facestat -patchstat -alt -td_limits f 6 6 b 2 1
```

then the following comments will be written into the log file
`3reg_g1_v24_f5_f6_f7_alt_td6.2_6.1.log`:

```
Program call:  CGF -g 1 -f 5 -f 6 -f 7 -v 24 -output none -save 0 -mapstat
               -facestat -patchstat -alt -td_limits f 6 6 b 2 1

Genus:      min  1   max  1
Vertices:  min 24   max 24
Permitted face degrees:  5 6 7

Limits for faces of certain degrees (degree/min/max):
  5 :   0   6
  6 :   0  12
```

```
  7 :   0   6

Remark: The following combination is not possible due to Jendrol's theorem:
Genus 1,  12 faces:   1 5-gon, 10 6-gons, 1 7-gon

Could not find dump file 3reg_g1_v24_f5_f6_f7_alt_td6.2_6.1.dump.
I will proceed without.
Time for generating the patches: 0.2 seconds
Number of stored patches: 4727
Memory consumption: 731520 bytes

Stored patches sorted by canonical zig-zag paths:
intersection:   3509
enclosure:        41
linkage:        1174
none:              3

Patches sorted by genus and inner faces (row: #inner faces, column: genus):
    |     0     1
----+------------
  1 |     3     0
  2 |    13     0
  3 |    47     0
  4 |   221     0
  5 |   798     0
  6 |  3627    18
  7 |     0     0
  8 |     0     0
  9 |     0     0
 10 |     0     0
 11 |     0     0
 12 |     0     0


Constructed 11240 maps with genus 1 and 24 vertices along type-1 Petrie paths
 in 9.2 seconds, accepted 23.
Constructed 8060 maps with genus 1 and 24 vertices along type-2 Petrie paths
 in 13.3 seconds, accepted 319.
Constructed 1035 maps with genus 1 and 24 vertices along type-3 Petrie paths
 in 0.9 seconds, accepted 477.
=> Constructed 819 non-isomorphic maps with 24 vertices.
Total generation time: 23.6 seconds.

Maps sorted by degree sets:
5 6 7 :  769 maps
5 7 :  43 maps
6 :  7 maps
Used patches: 2711
```

Here type–1 Petrie paths are Jordan curve paths, type–2 Petrie paths are sandwich paths, and type–3 Petrie paths are dumbbell paths. "Used patches" are those stored patches that have been agglutinated with other patches during the construction of the 20335 maps.

A.3 Options

The following table lists all possible options. The keywords are printed using typewriter letters. When an option requires parameters, then the corresponding variables are printed using italic letters. When a parameter is optional, it is embraced in brackets "[" and "]".

`-v` n_1 : This option is mandatory. n_1 is the number of vertices in every constructed cubic map.

`-vs` n_2 : If this option is used, then the number of vertices in the constructed cubic maps may vary between n_2 (lower limit) and n_1 (upper limit).

`-g` g_1 : This option is mandatory. g_1 is the genus of every constructed cubic map.

`-gs` g_2 : If this option is used, then the genera of the constructed cubic maps may vary between g_2 (lower limit) and g_1 (upper limit).

`-delaney` : The constructed cubic maps are interpreted as Delaney graphs. This option may be used only if $g_1 = 0$. When the option is used, it must occur in the command line before the option `-f` occurs in the command line for the first time because the behaviour of `-f` depends on `-delaney`.

`-f` d $[c]$ $[s]$: This option includes d in D, i.e. d–gons are permitted in the constructed cubic maps. Every face degree which is not explicitly included in D is not permitted. At least one face degree must be permitted. If the option `-delaney` is used, then c is mandatory, otherwise it must be omitted. c is a number providing the information whether d–gons may be coloured red, green, or blue. The three colours are represented by the digits 1, 2, and 3. If d–gons may be coloured with a certain colour, then the corresponding digit is a part of the number c. Thus $c \in \{\texttt{1}, \texttt{2}, \texttt{3}, \texttt{12}, \texttt{13}, \texttt{23}, \texttt{123}\}$ (where permutations of the digits, e.g. `312` instead of `123`, are accepted by *CGF*). s is a string providing limits for the number of d–gons that must occur in every constructed map:

- If only a lower limit shall be provided, then s begins with the letter "`l`" or with an opening bracket "`[`" followed by the lower limit. Examples: Both "`l3`" and "`[3`" indicate $b_l(d) = 3$. If "`[1`" is provided for every permitted face degree d, then D is the degree set of all constructed maps. Otherwise the degree set D_M of a constructed map M is a subset of D, i.e. the constructed maps do not necessarily contain all permitted face degrees.
- If only an upper limit shall be provided, then s begins with the upper limit followed by the letter "`u`" or by a closing bracket "`]`". Examples: Both "`7u`" and "`7]`" indicate $b_u(d) = 7$.
- If both a lower limit and an upper limit shall be provided, then both above strings are joined by a "`-`". Examples: Both "`l3-7u`" and "`[3-7]`" indicate $b_l(d) = 3$ and $b_u(d) = 7$.

If limits are provided, then the face vector of every patch is stored.

`-output` s : By default, the codes of the accepted maps are written into a code file whose name is derived from the used options (see Section A.4). For every vertex number and every genus, an own code file is created. If the option `-output` is used, then the codes are not written into code files but somewhere else indicated by the string s which is one of `stdout` (the maps are written to the standard output channel), `none` (the maps are not written at all), or `stdout_max` (useful only if the option `-vs` is used with $n_2 < n_1$: the maps of order n_1 are written to the standard output channel, all maps of smaller order are written into code files). `stdout` must be used if the codes are to be piped into another program that uses them as input.

`-outputmem` x y : As stated in Section A.1, the codes of the accepted maps are not written into the code file immediately. Instead, some codes are collected, and when a certain limit is reached, then all collected codes are written at once. This is done in order not to write dump files too often: Whenever codes are written, the dump file must be updated so that the codes are not rewritten after a restart. So for every order that the constructed maps may have, some memory is provided to collect maps. This memory is the higher, the higher the order is (because in general, the number of accepted maps increases with increasing vertex number). The limits are provided by x and y: For maps of order n_2 (see option `-vs`), the provided memory is big enough to collect x codes. For maps of order n_1, the provided memory is big enough to collect $x + y$ codes. For maps or order n with $n_2 < n < n_1$, the memory varies linearly between x and $x + y$. As soon as the memory is used up for one vertex number, *all* collected codes are written. Defaults: $x = 200$, $y = 300$. If the maps are piped into a visualization program, then it is advisable to set $x = y = 0$ so that the first code is written as quickly as possible.

`-logfile` s : s is a string that indicates where *CGF* writes comments such as the number of accepted maps or the required time. s is one of `none` (no comments are written), `stderr` (the comments are written to the standard error channel), `file` (the comments are written into the log file), or `both` (the comments are written to the standard error channel and into the log file). Default: `both`.

`-alt` : If this option is used, then the standard size will be used to manage the patch generation. By default, the advanced size is used. The standard size sometimes yields a faster generation than the advanced size if the critical sum is restricted, for example if $\mathcal{M}(1, n, \{5, 6, 7\}, b_l, b_u)$ with $b_l(6) = 0$, $b_u(6) = \infty$, and $b_l(5) = b_l(7) = b_u(5) = b_u(7) = 3$ is to be generated.

`-topdown` : If this option is used, then the combined approach will be performed. By default, the bottom–up approach will be performed.

`-td_limits` s' : If this option is used, then the combined approach will be performed. The option must occur in the command line after the option `-g` occurs in the command line. s' is a sequence of letters and numbers that determine the levels described in Section 9.7. Between any two consecutive entries of the sequence, there must be a blank space in the command line (see Chapter 10 for examples). s' has the following structure:

- If the option `-alt` is used, then $s' = \langle \mathtt{f}, f'_0, \ldots, f'_{g_1}, \mathtt{b}, b_0, \ldots, b_{g_1} \rangle$ where the f'_i and b_i, $i \in \{0, \ldots, g_1\}$, are described in Section 9.7 (for g_1, see option `-g`).
- If the option `-alt` is not used, then $s' = \langle \mathtt{c}, c_0, \ldots, c_{g_1}, \mathtt{f}, f'_0, \ldots, f'_{g_1}, \mathtt{b}, b_0, \ldots, b_{g_1} \rangle$ where the c_i, f'_i, and b_i ($i \in \{0, \ldots, g_1\}$) are described in Section 9.7.

Alternatively, s' can be the string `auto`. In this case, *CGF* determines an appropriate sequence automatically. The defaults for the numbers are the largest possible numbers.

`-patchcheck` : If this option is used, then the *advanced patch testing* is performed.

`-facecon` x : It must hold $x \in \{\mathtt{3}, \mathtt{23}, \mathtt{123}\}$. If $x = \mathtt{23}$, then only semi–polyhedral maps are accepted. If $x = 3$, then only polyhedral maps are accepted. If $x = \mathtt{123}$, then there is no restriction related to semi–polyhedral or polyhedral maps. Default: $x = \mathtt{123}$.

`-mapcon` x : This option is used to filter out maps with a certain connectivity. It must hold $x \in \{\mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{12}, \mathtt{13}, \mathtt{23}, \mathtt{123}\}$. If the digit y is a part of the number x, then maps that are exactly y–connected are accepted. Default: $x = \mathtt{123}$. If $g_1 > 0$ (see option `-g`), then it must hold $x \in \{\mathtt{1}, \mathtt{23}, \mathtt{123}\}$ because the filter is very simple so that it cannot determine whether a map of genus $g > 0$ is exactly 2–connected or 3–connected.

`-patchcon` : Similar to polyhedral or semi–polyhedral maps (compare Section 8.5), the filter invoked by `-mapcon` can be included already into the generation of the patches. This is done only if the option `-patchcon` is used.

`-mod` x : This option is used in order to split a big process into x subprocesses $p_0, \ldots, p_{x-1}$. If this option is used, then the usage of the option `-class` is mandatory.

`-level` z : This option can be used only if the bottom–up approach is performed. z indicates the threshold recursion level where the big process must be split. The default is $z = 0$. In the combined approach, this level is always 0 and cannot be modified. However, for the combined approach, there are other options to split a process into subprocesses using a higher threshold level.

`-class` y : This option may only be used if `-mod` is used. y may vary from 0 to $x - 1$ and indicates the subprocess p_y that will be executed.

`-mod2` x_2 : This option may be used only in case the combined approach is performed. Since the subprocesses of a big process might require very inhomogeneous amounts of time (see Section 9.9), this option provides a possibility to split an obstinate subprocess p_y once more. x_2 is the number of subsubprocesses. The option `-mod2` can also be used when the option `-mod` is not used (so that the subsubprocesses are actually subprocesses).

`-level2` z_2 : This option may only be used if `-mod2` is used. z_2 indicates the threshold level where the big process must be split. It must hold $z_2 \geq 1$. Default: $z_2 = 1$.

`-class2` y_2 : This option may only be used if `-mod2` is used. y_2 may vary from 0 to $x_2 - 1$ and indicates which of the x_2 subprocesses of p_y shall be executed. Maps that are constructed before the split takes place are accepted only in the first subprocess of p_y.

`-mod3` x_3, `-mod4` x_4, `-mod5` x_5, `-mod6` x_6 : These options are similar to `-mod2`. They can be used to further split a subsubprocess.

`-level3` z_3, `-level4` z_4, `-level5` z_5, `-level6` z_6 : These options are similar to `-level2`. If $i < j$ for some $i, j \in \{2, 3, 4, 5, 6\}$, then it must hold $z_i < z_j$. Defaults: $z_3 = 3$, $z_4 = 5$, $z_5 = 8$, $z_6 = 11$.

`-class3` y_3, `-class4` y_4, `-class5` y_5, `-class6` y_6 : These options are similar to `-class2`. They can be used to further split a subsubprocess.

`-old_mod2` x'_2 : This option can be used to further split a subprocess supplementarily. If this option is used, then the option `-mod2` must be used, too, and either x_2 must be a multiple of x'_2 or it must hold $x'_2 = 0$.

- If $x'_2 = 0$, then a new splitting in level z_2 is introduced. This works as follows: Suppose *CGF* has been called with a set of options not containing the option `-mod2`, and the corresponding process (which will be called the "original process" in the following description) is running for a long time. Then one can split the process in level z_2 supplementarily by interrupting the original process and calling *CGF* again using the same set of options and the additional options `-old_mod2` 0, `-mod2` x_2, and `-class2` y_2. *CGF* must be called x_2 times where y_2 varies from 0 to $x_2 - 1$. Each new subprocess will read the dump file of the original process and introduce the new splitting.
- $x'_2 \neq 0$, then an existing splitting in level z_2 is extended. This works as follows: Suppose *CGF* has been called with a set of options containing the options `-mod2` x'_2 and `-class2` y'_2, and the corresponding process (which will be called the "original process" in the following description) is running for a long time. Then one can extend the splitting in level z_2 supplementarily by interrupting the original process and calling *CGF* again using the same set of options where the options `-mod2` x'_2 and `-class2` y'_2 are replaced by the options `-old_mod2` x'_2, `-old_class2` y'_2, `-mod2` x_2, and `-class2` y_2 where x_2 must be a multiple of x'_2. Let $x_2 = c \cdot x'_2$ for some $c \in \mathbb{N}$. *CGF* must be called c times where each value of $\{y'_2 + k \cdot x'_2 | k \in \{0, \ldots, c - 1\}\}$ must be assigned once to y_2 (for the option `-class2`). Each new subprocess will read the dump file of the original process and extend the splitting.

In both cases, the following things must be considered:

- When a process that has been called using the option `-old_mod2` gets lost, it must be recovered without using this option. Otherwise the process would reread the dump file of the original process again instead of reading the dump file of the new subprocess.

- When using the option `-old_mod2`, all new subprocesses must start at the same point, i.e. they must read the same dump file of the original process. In particular, the original process must have been stopped before (so that the corresponding dump file is not updated anymore), and subprocesses where the option `-old_mod2` is used must not be mixed with subprocesses where the option `-old_mod2` is not used.
- When using the option `-old_mod2`, the number of accepted maps so far in the original process is written into the log file. Afterwards, the counter is set to zero so that at the end of the new subprocess, only the number of *additionally* accepted maps in the subprocess is written into the log file. Thus the total number of accepted maps is obtained by adding all numbers that are written at the end of the log files of the subprocesses and the number of accepted maps in the original process. Moreover, the code files that correspond to the subprocesses do not begin with a header. So by appending the code files of the subprocesses to the code file of the original process, one obtains a list of all maps that would have been accepted if the original process had been executed completely.

`-old_mod3` x_3', `-old_mod4` x_4', `-old_mod5` x_5', `-old_mod6` x_6' : These options are similar to `-old_mod2`. They can be used for further supplementary splits.

`-old_class3` y_3', `-old_class4` y_4', `-old_class5` y_5', `-old_class6` y_6' : These options are similar to `-old_class2`. They can be used for further supplementary splits.

`-priority` x : With this option, the qualities of the three types of Petrie paths can be set. It holds $x \in \{\texttt{123}, \texttt{132}, \texttt{213}, \texttt{231}, \texttt{312}, \texttt{321}\}$. The digit "`1`" represents Jordan curve paths, the digit "`2`" represents sandwich paths, and the digit "`3`" represents dumbbell paths. The earlier the digit occurs in the number x, the higher is the quality of the corresponding type of Petrie paths. The default is $x = \texttt{321}$ due to Theorem 8.5, but sometimes it can be useful to modify x (see Section 9.8).

`-types` x : This option causes *CGF* to restrict the construction of cubic maps depending on the types of the Petrie paths. It holds $x \in \{\texttt{1}, \texttt{2}, \texttt{3}, \texttt{12}, \texttt{13}, \texttt{23}, \texttt{123}\}$. As for the option `-priority`, the digit "`1`" represents Jordan curve paths, the digit "`2`" represents sandwich paths, and the digit "`3`" represents dumbbell paths. Patches are agglutinated along Petrie paths of type i only if the digit i occurs in x. Default: $x = \texttt{123}$.

`-save` t : This option causes *CGF* to update the dump file every t seconds. Default: $t = 2000$. If $t \leq 0$, then no dump file will be created at all.

`-lazydump` : If this option is used, then the dump file is updated only every t seconds even if codes are written into the code file (because the memory for collecting codes is used up; see option `-outputmem`). This option can be used when it does not matter if codes are rewritten after a recovering of a lost process (e.g. when the maps are piped). The option helps to minimize the number of disk accesses (which can be important in large computer clusters).

`-memory` x : x is the number of bytes in the computer memory that may be used for storing the permanent patches. Default: $x = \infty$. If the memory is used up, then the behaviour of *CGF* depends on the construction approach: In the bottom–up approach, *CGF* exits. In the combined approach, *CGF* stops the generation of the permanent patches, adapts the limits that have been provided with the option `-td_limits` and continues with the top–down part of the construction. If the numbers must not be modified (this is the case if one of the options `-mod`, `-mod2`, `-mod3`, `-mod4`, `-mod5`, or `-mod6` is used because all subprocesses must refer to the same set of permanent patches), then *CGF* exits.

The option `-memory` is useful especially if the option `-topdown` is used instead of the option `-td_limits`. In this case, *CGF* first tries to store as many patches as possible, then *CGF* sets the limits for the domain of permanent patches, and then *CGF* continues with the top–down part of the construction.

`-dual` : If this option is used, then not the original maps but their *duals* will be encoded and written into the code file (vertex code or edge code depending on whether the dual is simple or not).

`-db` : If this option is used, then double edges are allowed in the accepted maps even if neither monogons nor digons are permitted.

`-digon_path` : If this option is used, then a Petrie path p is canonical only if p is a sandwich path with $l_1(p) = l_2(p) = 1$ or a dumbbell path with $l_1(p) = 2$. Compare Section 9.8. See Section 10.2 for an example how this option is used.

`-no_recover` : This option causes *CGF* not to recover the current process even if an appropriate dump file exists.

`-fileindex` s : This option causes *CGF* to include the string s into the names of the created files. Thus more individual file names can be given. See Section A.4.

`-logfilename` s : This option causes *CGF* to name the log file completely different. s is the new name of the log file.

`-shortfilename` : The file names created by *CGF* become rather long. With this option, only the most important options are included into the file names (see Section A.4).

`-mapstat` : If this option is used, then the log file contains some statistics about the number of constructed maps depending on the types of the Petrie paths.

`-facestat` : If this option is used, then the log file contains some statistics about the degree sets of the accepted maps. To this end, the face vector of every patch is stored during the construction. The option is useful only if at least three different face degrees are permitted, e.g. $D = \{5, 6, 7\}$. In this case, the degree sets of two accepted maps might differ from each other.

`-patchstat` : If this option is used, then the log file contains some statistics about the number of stored patches depending on the number of inner faces and the genus. Moreover, the log file contains some statistics about the number of stored patches depending on the type of the canonical zig–zag path.

`-pv` x : This option can be used to switch off patch preventing criteria. It should be used only for tests. The default is $x = 20$. In the bottom–up approach, no recursion will be performed if $x < 10$. In the top–down approach, no tests on the numbers of convex edges in the patches (as described in Section 9.4, page 174) will be performed if $x < 5$.

`-pid` : This option causes *CGF* to write the process ID to the standard output channel. The option is useful if *CGF* shall interact with another program.

A.4 File names

In this section, the names of the created files are discussed. Except for the code files, the bodies of all file names are identical. The bodies of the code files slightly differ due to the different pairs of vertex numbers and genera.

The body of the name of a code file begins with "`3reg_g`i`_v`j" where i is the genus and j is the order of every map whose code is written into the file. The bodies of the names of the other files begin with "`3reg_g`i_1`_gs`i_2`_v`j_1`_vs`j_2" where $i_1 = g_1$, $i_2 = g_2$, $j_1 = n_1$, and $j_2 = n_2$ as provided by the options `-g`, `-gs`, `-v`, and `-vs`. If the options `-gs` and `-vs` are not used, then the corresponding strings do not occur in the file names.

The rest of the body is identical for all files. Almost every option contributes a short string to the file name. In detail:

`-delaney`: "`_delaney`"

`-f` f $[c]$ $[s]$: "`_f`f", "`_f`f`_`c", "`_f`f`_`s", or "`_f`f`_`c`_`s" depending on which optional parameters have been used ("`l`" and "`u`" are always replaced by "`[`" and "`]`")

`-t` x: "`_t`x" if $x \neq$ `123`

`-priority` x: "`_p`x"

`-alt`: "`_alt`"

`-mapcon` x: "`_mc`x" if $x \neq$ `123`

`-facecon` x: "`_fc`x" if $x \neq$ `123`

`-patchcon`: "`_pcon`"

`-patchcheck`: "`_pch`"

`-dual`: "`_dual`"

`-db`: "`_db`"

`-fileindex` s: "`_`s"

`-pv` x: "`_pv`x"

`-topdown`: "`_td`"

`-td_limits` s': "`_tdl`" if s' is the string "`auto`" and "`_td_`s" if s' is not the string "`auto`" where $s = \langle f'_0, b_0, \ldots, f'_{g_1}, b_{g_1} \rangle$ or $s = \langle c_0, f'_0, b_0, \ldots, c_{g_1}, f'_{g_1}, b_{g_1} \rangle$ depending on whether `-alt` is used or not (the entries of s are concatenated by dots and underscores)

`-mod` x: "`_m`x`.`y" or "`_m`x`.`y`.`z" depending on whether a level z is provided explicitly (using the option `-level`) where y is the class provided by `-class`.

`-mod2` x_2: "`_M`x_2`.`y_2" or "`_M`x_2`.`y_2`.`z_2" depending on whether a level z_2 is provided explicitly where y_2 is the class provided by `-class2`.

`-mod3` x_3: "`_MM`x_3`.`y_3" or "`_MM`x_3`.`y_3`.`z_3"

`-mod4` x_4: "`_mm`x_4`.`y_4" or "`_mm`x_4`.`y_4`.`z_4"

`-mod5` x_5: "`_3m`x_5`.`y_5" or "`_3m`x_5`.`y_5`.`z_5"

`-mod6` x_6: "`_3M`x_6`.`y_6" or "`_3M`x_6`.`y_6`.`z_6"

If the option `-shortfilename` is used, then the only options that contribute a string to the file name are the options `-delaney`, `-fileindex`, `-mod`, `-mod2`, `-mod3`, `-mod4`, `-mod5`, and `-mod6`.

Appendix B

The programs *togen* and *togen7*

In order to start the program *togen*, the program's name followed by several options must be entered. Each option begins with a minus sign "-". Some options require further parameters, some options do not. An example call is

```
togen -v 18 -output none
```

which causes *togen* to construct $\mathcal{T}_{18}$ but not to write any map into a code file. Thus the maps are just counted. *togen* and *togen7* provide a *vertex code* of every accepted map as output. The codes are written one after another. The whole output begins with a header

```
>>embed_code le<<
```

which enables other programs to recognize the code.

The output files that are created by *togen* and *togen7* correspond to the files that are created by *CGF*. See Section A.1 for details. The bodies of the file names depend on the options that are used for the process. The naming of the files will be discussed in Section B.3.

B.1 Example log file

When *togen7* is called as follows

```
togen7 -v 28 -output none
```

then the following comments will be written into the log file `ct7_v28.log`

```
Program call:  togen7 -v 28 -output none

Vertices:  min 28   max 28
Time for generating the maps: 78.7 seconds
Generated 0 non-isomorphic maps with 4 vertices.
```

```
Generated 1 non-isomorphic maps with 6 vertices.
Generated 4 non-isomorphic maps with 8 vertices.
Generated 6 non-isomorphic maps with 10 vertices.
Generated 15 non-isomorphic maps with 12 vertices.
Generated 33 non-isomorphic maps with 14 vertices.
Generated 115 non-isomorphic maps with 16 vertices.
Generated 297 non-isomorphic maps with 18 vertices.
Generated 997 non-isomorphic maps with 20 vertices.
Generated 3118 non-isomorphic maps with 22 vertices.
Generated 10853 non-isomorphic maps with 24 vertices.
Generated 37618 non-isomorphic maps with 26 vertices.
Generated 133601 non-isomorphic maps with 28 vertices.
end of process
```

B.2 Options

The following table lists all possible options. The keywords are printed using typewriter letters. When an option requires parameters, the corresponding variables are printed using italic letters. When a parameter is optional, it is embraced in brackets "[" and "]".

`-g` g : It must hold $g \in \{\mathtt{0}, \mathtt{1}, \mathtt{10}\}$. If the digit "`0`" occurs in the number g, then planar maps are constructed. If the digit "`1`" occurs in g, then toroidal maps are constructed. Default: $g = \mathtt{1}$.

`-v` n_1 : This option is mandatory. If the option is used for *togen*, then *togen* constructs $\mathcal{T}_{n_1}$, $\mathcal{P}_{n_1}$, or $\mathcal{T}_{n_1} \cup \mathcal{P}_{n_1}$ depending on g. The elements of $\mathcal{T}'_{n_1}$, $\mathcal{P}'_{n_1}$, or $\mathcal{T}'_{n_1} \cup \mathcal{P}'_{n_1}$, respectively, are encoded and written into the code file(s).

If the option is used for *togen7*, then *togen7* constructs $\mathcal{U}_{n_1}$, $\mathcal{Q}_{n_1}$, or $\mathcal{U}_{n_1} \cup \mathcal{Q}_{n_1}$ depending on g. The elements of $\mathcal{U}'_{n_1}$, $\mathcal{Q}'_{n_1}$, or $\mathcal{U}'_{n_1} \cup \mathcal{Q}'_{n_1}$, respectively, are encoded and written into the code file(s).

`-vs` n_2 : If this option is used for *togen*, then all elements of $\mathcal{T}_{n_1} - \mathcal{T}_{n_2-2}$, $\mathcal{P}_{n_1} - \mathcal{P}_{n_2-2}$, or $(\mathcal{T}_{n_1} \cup \mathcal{P}_{n_1}) - (\mathcal{T}_{n_2-2} \cup \mathcal{P}_{n_2-2})$, respectively, are encoded and written into the code files (one code file for each vertex number and each genus). If the option is used for *togen7*, then all elements of $\mathcal{U}_{n_1} - \mathcal{U}_{n_2-2}$, $\mathcal{Q}_{n_1} - \mathcal{Q}_{n_2-2}$, or $(\mathcal{U}_{n_1} \cup \mathcal{Q}_{n_1}) - (\mathcal{U}_{n_2-2} \cup \mathcal{Q}_{n_2-2})$, respectively, are encoded and written into the code files.

`-output` s : By default, the codes of the accepted maps are written into a code file whose name is derived from the used options (see Section B.3). For every genus and every vertex number, an own code file is created. If the option `-output` is used, then the codes are not written into code files but somewhere else indicated by the string s which is one of `stdout` (the maps are written to the standard output channel), `none` (the maps are not written at all), or `stdout_max` (useful only if the option `-vs` is used with $n_2 < n_1$: the maps of order n_1 are written to the standard output channel, all maps of smaller order are written into code files). `stdout` must be used if the codes are to be piped into another program that uses them as input.

`-CGF` s : By default, *CGF* is invoked in order to provide the hexagonal lattices having at most n_1 vertices. Using the option `-CGF`, *togen* and *togen7* can be told the name (and the path to the appropriate directory) of *CGF* on the disk. Default: s = `CGF`.

`-inputfile` s : If the hexagonal lattices are to be read from an input code file instead of invoking *CGF*, then this option must be used. s is the name of the input file on the disk.

The option *must* be used when the option `-mod` (see below) is used.

`-logfile` s : s is a string that indicates where *togen* and *togen7* write comments such as the number of accepted maps or the required time. s is one of `none` (no comments are written), `stderr` (the comments are written to the standard error channel) , `file` (the comments are written into a log file), or `both` (the comments are written to the standard error channel and into a log file). Default: `both`.

`-facecon` x : It must hold $x \in \{3, 23, 123\}$. If $x = 23$, then only semi–polyhedral maps are written into the code files. If $x = 3$, then only polyhedral maps are written into the code files. If $x = 123$, then there is no restriction related to semi–polyhedral or polyhedral maps. Default: $x = 123$. This is a filter only. It does not affect the construction process.

`-mapcon` x : This option is used to filter out maps having a certain connectivity. If the digit y is a part of the number x, then maps that are exactly y–connected are written. It must hold $x \in \{1, 23, 123\}$ because the filter is very simple so that it cannot determine whether a toroidal map is exactly 2–connected or 3–connected. Default: $x = 123$.

`-mod` x : This option is used in order to split a big process into x subprocesses $p_0, \ldots, p_{x-1}$. If this option is used, then the usage of the option `-class` is mandatory.

`-level` z : This option may only be used if `-mod` is used. z indicates the vertex number where the big process must be split. The default is $z = \max\{6, \frac{2}{3}n_1\}$. It holds $z = m + 2$ where m is defined in Section 4.5.

`-class` y : This option may only be used if `-mod` is used. y may vary from 0 to $x - 1$ and indicates the subprocess p_y that will be executed. Maps having a smaller order than z that are to be written into a code file are written if and only if $y = 0$.

`-save` t : This option causes *togen* and *togen7* to update the dump file every t seconds. Default: $t = 2000$. If $t \leq 0$, then no dump file will be created at all.

`-dual` : If this option is used, then not the original maps but their *duals* will be encoded and written into the code files.

`-no_recover` : This option causes *togen* and *togen7* not to recover the current process even if an appropriate dump file exists.

`-fileindex` s : This option causes *togen* and *togen7* to include the string s into the names of the created files. Thus more individual file names can be given. See Section B.3.

`-logfilename` s : This option causes *togen* and *togen7* to name the log file completely different. s is the new name of the log file.

`-pid` : This option causes *togen* and *togen7* to write the process ID to the standard output channel. The option is useful if *togen* and *togen7* shall interact with other programs.

B.3 File names

In this section, the names of the created files are discussed. Except for the code files, the bodies of all file names are identical. The bodies of the code files slightly differ due to the different genera and vertex numbers.

The body of the name of a code file begins with "`cs_v`j" where j is the order of every map whose code is written into the file. s is a string containing the letter "`p`" if planar maps are written into the file and "`t`" if toroidal maps are written into the file. Moreover, the string ends with the digit "`7`" if the file is created by *togen7*. The bodies of the names of the other files begin with "`cs_v`j_1`_vs`j_2" where $j_1 = n_1$ and $j_2 = n_2$. If the option `-vs` is not used, then the bodies begin with "`cs_v`j_1".

The rest of the body of the file name is identical for all files. Almost every option contributes a short string to the file name. In detail:

`-mapcon` x: "`_c1`" if $x =$ `1` and "`_c2`" if $x =$ `23`

`-facecon` x: "`_fc`x" if $x \neq$ `123`

`-dual`: "`_dual`"

`-mod` x: "`_m`x`.`y`_l`z" where y is the class provided by `-class` and z is the threshold level.

`-inputfile` s: "`_input_'`s`'`"

`-fileindex` s: "`_`s"

Appendix C

Verifying the programs

Of course, no guarantee can be given that the programs *CGF*, *togen*, and *togen7* are correct. However, in order to detect errors, exhaustive tests have been performed. In this chapter, the possibilities to test the programs and the required tools and reference programs are described. Moreover, further tools are presented.

C.1 Tools

The following tool programs were developed and used for the tests:

facestat: The program *facestat* reads vertex codes representing cubic maps and determines the *degree sets* of the maps. If the option "full" is used, then the program writes the degree set D_M of every single map M in the same order in which the maps were read. Otherwise for every determined degree set D, the program writes down the number of maps having the degree set D. Reading maps constructed by *togen* and *togen7*, the program can be used to generate statistics for cubic toroidal maps having certain degree sets. This statistics serves as a reference when *CGF* is tested.

embedstat: The program *embedstat* does the same as **facestat**, but an own statistics is created for every genus.

togenstatadd: The results given in Sections 4.6 and 5.3 were computed using the parallelizing technique described in Section 4.5. The program *togenstatadd* was used to add the numbers of maps that were provided by the single log files. Let n and m be given as in Section 4.5. n and m must be provided as arguments when calling *togenstatadd*. The program reads the log files from the standard input file. For that, all original log files must have been merged in one file. First, the program writes down the number l of original log files that were read. In every original logfile, for every vertex number n' with $4 \leq n' \leq n$, the number of constructed maps containing n' vertices is listed. Hence for every vertex number n', there are l numbers of maps. For $n' \leq m$, *togenstatadd* verifies if all corresponding numbers of maps in the original log files are identical. For $n' > m$, *togenstatadd* adds the numbers. Finally, for every vertex number n', the corresponding sum is written.

embed: This program reads simple connected cubic graphs. For every read graph G, all pairwise non–isomorphic maps having G as the underlying graph are constructed. The resulting maps are sorted by their genera, and if desired, then they are encoded and written into code files. Here for each genus, one code file is created. Moreover, the degree set of every map is determined. At the end of the process, the degree set statistics is written as by the program *embedstat*. Using the program *minibaum* developed by Gunnar Brinkmann [52, 53], one can construct all simple connected cubic graphs of a given order n. Using these graphs as input for *embed*, one can construct all simple cubic maps of order n.

embed_extract: This program reads vertex codes of simple cubic maps. The user must give a set D (by providing the elements of D as arguments). The program writes the vertex codes of all read maps having the degree set D to the standard output channel. The current numbers of these maps are written to the standard error channel.

statadd: This program does the same as the program *togenstatadd*, but for the statistics provided by *embed*. So whenever a degree set D occurs in several statistics (related to the same genus), then the corresponding numbers of maps are added. Thus the construction of maps using *embed* can be parallelized by distributing the graphs constructed by *minibaum* to several processes. Of course, maps that are constructed by different processes are pairwise non–isomorphic because even the underlying graphs are pairwise non–isomorphic.

mapadd: This program does the same as the program *togenstatadd*, but for the numbers of maps provided by *CGF* if a big process has been split into a set of subprocesses.

facestatadd: This program does the same as the program *statadd*, but for the face degree statistics which is provided by *CGF* using the option `-facestat`. Hence like *mapadd*, the program *facestatadd* is used when a big process has been split into a set of subprocesses.

gconv: This program converts different graph and map codes into each other. Among others, the program converts edge codes and vertex codes (which are non–readable binary codes) into ASCII codes that can be read by humans. The program has been a basic tool while debugging the programs *CGF*, *togen*, and *togen7*.

non_iso_embed: This program reads vertex codes or edge codes of cubic maps. For every read map M, the program tests if M is isomorphic to a map M' that has been read before. If it is, then the current number of M' is written. Otherwise it is written that M is new. At the end, the program writes the number of read maps and the number of pairwise non–isomorphic maps. If a file name is provided as an argument, then the vertex codes of those maps that are pairwise non–isomorphic are written into that file. *non_iso_embed* has been a basic tool while debugging the programs *CGF*, *togen*, and *togen7*.

supertest_CGF: This program reads the statistics produced by *embed*. For every genus g, the corresponding degree sets in the statistics are considered. For every degree set D, the program reads the corresponding number of maps in the statistics and calls *CGF* such that *CGF* constructs $\mathcal{M}(g, n, D, b_l, b_u)$ where $b_l(d) = 1$ and $b_u(d) = \infty$ for every $d \in D$, and n is the order of the maps that have been constructed by *embed*. n must be provided as an argument when calling *supertest_CGF*. As soon as *CGF* has finished, *supertest_CGF* reads the number of accepted maps and compares it with the statistics provided by *embed*. Then the next call of *CGF* is performed. As soon as a difference occurs in the number of maps, *supertest_CGF* stops. The following options may be provided:

`n` n : This option is mandatory.

`embed` x : As long as the number of accepted maps is at most x, the maps are read and compared with the maps provided by *embed*. To this end, the program *embed_extract* is called which extracts the maps having degree set D from the appropriate file created by *embed*. The resulting list of maps is appended to the list of maps that have been accepted by *CGF*. Now *non_iso_embed* is called. While verifying the maps that have been accepted by *CGF*, *non_iso_embed* must not find pairwise isomorphic maps. But as soon as the maps constructed by *embed* are verified (with the ones read before), every map must be isomorphic to a map that has been accepted by *CGF*.

The verification of the accepted maps is quite time–consuming. Thus the default is $x = 0$.

`embedfile` s : s is the body of the name of the file containing the maps constructed by *embed*. To this body, the string "$n.g$" is appended (where g is the genus and n is the order of every stored map).

`all` : If this option is used, then for every degree set D, *six* calls of *CGF* are performed. The calls differ in the parameter which is supplied with the option `-priority`. All six possibilities are considered.

`opt` s : s is interpreted as an additional option for *CGF* (for example $s =$"`-alt`").

autosontest_CGF: This program has a similar function as the program *supertest_CGF*. However, instead of calling *CGF* directly, a batch file is created that contains a list of *autoson* calls. *autoson* is a batch system that automatically distributes and administrates the calls in a computer cluster such that all computers in the cluster are utilized optimally without hindering interactive work [54]. After all corresponding *CGF* processes have finished, the log files created by *CGF* are read by a program called *autosontestcollect_CGF*. This program reads the number of accepted maps from the log file and compares the number with the statistics provided by *embed*. If a big process has been split using the option `-mod`, then *autosontestcollect2_CGF* must be used. This program adds all numbers of maps for a given triple (n, g, D) before comparing the sum with the reference number.

supertest2_CGF: This program reads the statistics that has been written by *embed*. For every genus g, the corresponding degree sets in the statistics are considered. For every degree set D, the program reads the corresponding number of maps in the statistics and initiates two calls of *CGF* that both construct $\mathcal{M}(g, n, D, b_l, b_u)$ using the bottom–up approach where $b_l(d) = 1$ and $b_u(d) = \infty$ for every $d \in D$, and n is the order of the maps that have been constructed by *embed* (n must be provided as an argument when calling *supertest2_CGF*). For the first call, the option `-pv 4` is used so that no recursion will be invoked. For the second call, this option is not used (so that recursion will be invoked). As soon as both processes have finished, *supertest2_CGF* reads the numbers of accepted maps and compares them with each other. Then the next two calls are performed. As soon as a difference occurs in the numbers of accepted maps, *supertest2_CGF* stops. Thus the bottom–up recursion of *CGF* can be tested exhaustively. The following options may be provided:

`n` n : This option is mandatory.

`opt` s : s is interpreted as an additional option for *CGF*.

fluss5: This program reads vertex codes of simple cubic maps and tests whether the maps have a 5–flow. Maps that do not have a 5–flow are written to the standard output channel. The program has been developed by Gunnar Brinkmann.

cubhamg6: This program reads simple cubic maps and tests whether the maps are Hamiltonian. non–Hamiltonian maps are written to the standard output channel. The program has been developed by Brendan D. McKay. *gconv* can be used to convert the vertex codes provided by *CGF* into the "**graph6**" codes that are expected by *cubhamg6*.

faceregular: This program reads vertex codes of simple cubic maps and tests whether the maps are face–regular or weakly face–regular (in this case, the option "**w**" must be used). The maps that are (weakly) face–regular are written to the standard output channel. The program has been developed by Gunnar Brinkmann.

C.2 Reference results

In order to obtain and to verify reference results, the following computations have been done:

- Using *minibaum*, *embed*, and *statadd*, all simple cubic maps of order at most 12 were constructed, and the statistics for all simple cubic maps of order at most 16 has been made up.
- Using the above results, *CGF* has been verified using *supertest_CGF* and *autosontest_CGF*. *CGF* yields correct results for all sets $\mathcal{M}(g, n, D, b_l, b_u)$ where $n \leq 12$, $D \subset (\mathbb{N} - \{1, 2\})$, and $b_l(d) = 1$ and $b_u(d) = \infty$ for every $d \in D$. Moreover, *CGF* yields correct numbers for the cardinalities of all sets $\mathcal{M}(g, n, D, b_l, b_u)$ where $n \leq 16$, $D \subset (\mathbb{N} - \{1, 2\})$, and $b_l(d) = 1$ and $b_u(d) = \infty$ for every $d \in D$.
- $|\mathcal{M}(1, n, \{6\}, b_l, b_u)|$ with $b_l(6) = 0$ and $b_u(6) = \infty$ has been computed for all $6 \leq n \leq 400$. The results have been verified using a statistics provided by Prof. E.C. Kirby (Pitlochry/Scotland).
- Using *togen*, *togen7*, and *facestat*, large lists of reference results for $|\mathcal{M}(g, n, D, b_l, b_u)|$ with $g \in \{0, 1\}$, $b_l(d) = 1$, and $b_u(d) = \infty$ for every $d \in D$ have been obtained in order to verify *CGF*. Moreover, using *togen*, *togen7*, and *embed_extract*, large lists of reference results for $\mathcal{M}(g, n, D, b_l, b_u)$ with $g \in \{0, 1\}$, $b_l(d) = 1$, and $b_u(d) = \infty$ for every $d \in D$ have been obtained in order to verify *CGF*.
- Using *CGF* with $g \in \{0, 1\}$, the results of both *togen* and *togen7* could be verified for small vertex numbers.
- The program *plantri* that has been developed by Gunnar Brinkmann and Brendan D. McKay constructs planar triangulations. The duals of these triangulations are simple cubic planar 3–connected maps. The results of *plantri* were used to verify *togen* and *togen7* for the cases where planar 3–connected (polyhedral) maps were constructed.
- For $g = 0$, the results of *CGF* could be verified with the results of *CPF* described in [46].

C.3 Testing parts of *CGF* and *togen*

In addition to the comparison of results of *CGF*, *togen*, and *togen7* with reference results, parts of *CGF*, *togen*, and *togen7* can be tested independently of reference results. The following tests have been done for *togen* and *togen7*:

- The look–aheads have been omitted.
- The qualities of the submaps have been modified.

The numbers of accepted maps provided by the modified programs have been compared with the numbers provided by the original programs.

The following tests have been done for *CGF* (and can be done by every user who does not trust in the results):

- The results of the combined approach can be compared with the results of the bottom–up approach.
- The results of the combined approach using different threshold sizes can be compared with each other.
- The results of the bottom–up approach with (advanced) recursion can be compared with the results of the bottom–up approach without recursion (this can be done exhaustively using *supertest2_CGF*).
- In order to test the implementation of the Euler criteria, the Euler criteria have been switched off (this cannot be done by the user anymore).
- In order to test the implementation of the canonicity check, the option `-priority` can be used.
- In order to test the implementation of the parallelization, the parameter of the option `-level` can be modified. Moreover, the results of a parallelized process can be compared with the results of the original process.
- In general, the implementation of switches like `-patchcon` or `-patchcheck` can be tested by comparing the results that are obtained when the switch is used with the results that are obtained when the switch is not used.

When performing these tests, it is always useful to use the option `-mapstat` because not only the number of *accepted* maps but even the number of *constructed* and not necessarily accepted maps must be equal for all tests (except for the switch `-alt` because the usage of this switch has an influence on the lists of patches that are assigned to one leaf in the patch tree and thus on the number of agglutinations that cannot be pairwise summed up; see Sections 8.4, 8.6, and 8.7.6).

Bibliography

[1] Reinhard Diestel: “Graph theory”, 1997 Springer Verlag, ISBN 0–387–98211–6

[2] William S. Massey: “Algebraic Topology: An Introduction”, 1977 Springer Verlag

[3] J.L. Gross, T.W. Tucker: “Topological Graph Theory”, 1987 Wiley, ISBN 0–471–04926–3

[4] Carsten Thomassen: “The Graph Genus Problem Is NP–Complete”, Journal of Algorithms **10**, 568–576 (1989)

[5] Bojan Mohar: “Embedding graphs in an arbitrary surface in linear time”, Proceedings of the 28th annual ACM symposium on the theory of computing (Philadelphia 1996), 392–397, ACM Press 1996

[6] John Hopcroft and Robert Tarjan: “Efficient Planarity Testing”, Journal of the Association for Computing Machinery **21**, No. 4, 549–568 (1974)

[7] Carsten Thomassen: “Embeddings of Graphs with No Short Noncontractible Cycles”, Journal of Combinatorial Theory B **48**, 155–177 (1990)

[8] Hassler Whitney: “Congruent Graphs and the Connectivity of Graphs”, American Journal of Mathematics **54**, 150–168 (1932)

[9] Seiya Negami: “Uniqueness and Faithfulness of Embeddings of Toroidal Graphs”, Discrete Mathematics **44**, 161–180 (1983)

[10] Joseph Malkevitch: “Polytopal Graphs”, Graph Theory **3**, 169–188 (1988), ISBN 0–12–086203–4

[11] V. Batagelj: “Inductive Classes of Cubic Graphs”, Colloquia Mathematica Societatis Janos Bolyai, **37** (Finite and Infinite Sets), Eger (Hungary), 89–101 (1981)

[12] V. Batagelj: `http://vlado.fmf.uni-lj.si/vlado/projects/indcla.htm`

[13] Gunnar Brinkmann, Andreas W.M. Dress: “A Constructive Enumeration of Fullerenes”, Journal of Algorithms **23**, 345–358 (1997), Article No. AL960806

[14] J. Liu, H. Dai, J.H. Hafner, D.T. Colbert, R.E. Smalley, S.J. Tans, C. Dekker: “Fullerene ‘crop circles’”, Nature Vol. **385**, February 27, 1997, 780–781

[15] E.E.B. Campbell, P.W. Fowler, D. Mitchell and F. Zerbetto: “Increasing cost of pentagon adjacency for larger fullerenes”, Chemical Physics Letters **250**, 544–548 (1996)

[16] P.W. Fowler, T. Heine, D.E. Manolopoulos, D. Mitchell, G. Orlandi, R. Schmidt, G. Seifert and F. Zerbetto: “Energetics of fullerenes with four–membered rings”, Journal of Physical Chemistry **100**, 6984–6991 (1996)

[17] A. Ayuela, P.W. Fowler, D. Mitchell, R. Schmidt, G. Seifert and F. Zerbetto: “C_{62}: theoretical evidence for a non–classical fullerene with a heptagonal ring”, Journal of Physical Chemistry **100**, 15634–15636 (1996)

[18] D.E. Manolopoulos, P.W. Fowler, R. Taylor, H.W. Kroto and D.R.M. Walton: “An end to the search for the ground state of C_{84}?”, Journal of the Chemical Society/Faraday Transactions **88**, 3117–3118 (1992)

[19] P.W. Fowler, D.E. Manolopoulos, G. Orlandi and F. Zerbetto: "Energetics and isomerisation pathways of a lower fullerene: the Stone–Wales map for C_{40}", Journal of the Chemical Society/Faraday Transactions **91**, 1421–1423 (1995)

[20] P.W. Fowler, T. Heine, D. Mitchell, G. Orlandi, R. Schmidt, G. Seifert and F. Zerbetto: "Energetics of fullerenes with heptagonal rings", Journal of the Chemical Society/Faraday Transactions **92**, 2203–2210 (1996)

[21] Gunnar Brinkmann, Olaf Delgado Friedrichs, Andreas Dress and Thomas Harmuth: "CaGe — a Virtual Environment for Studying Some Special Classes of Large Molecules", Match (MATCDY) **36**, 233–237 (1997), ISSN 0340–6253

[22] François Jaeger: "Nowhere–zero Flow Problems", Graph Theory **3**, 71–95 (1988), ISBN 0–12–086203–4

[23] François Jaeger: "Flows and Generalized Coloring Theorems in Graphs", Journal of Combinatorial Theory B **26**, 205–216 (1979)

[24] P.D. Seymour: "Nowhere–zero 6–flows", Journal of Combinatorial Theory B **30**, 130–135 (1981)

[25] Richard Steinberg: "Tutte's 5–Flow Conjecture for the Projective Plane", Journal of Graph Theory **8**, 277–285 (1984)

[26] Martina Möller, Hans Georg Carstens, Gunnar Brinkmann: "Nowhere–zero Flows in Low Genus Graphs", Journal of Graph Theory **12**, No. 2, 183–190 (1988)

[27] Eckhard Steffen: "Tutte's 5–flow conjecture for graphs of nonorientable genus 5", Journal of Graph Theory **22**, No. 4, 309–319 (1996)

[28] W.T. Tutte (Ed.): "Recent Progress in Mathematics", page 343, 1969 Academic Press, New York

[29] P.R. Goodey: "Hamiltonian Circuits in Polytopes with Even Sides Faces", Israel Journal of Mathematics **22**, No. 1, 52–56 (1975)

[30] P.R. Goodey: "A Class of Hamiltonian Polytopes", Journal of Graph Theory **1**, 181–185 (1977)

[31] D.A. Holton, B. Manvel, B.D. McKay: "Hamiltonian Cycles in Cubic 3–Connected Bipartite Planar Graphs", Journal of Combinatorial Theory B **38**, 279–297 (1985)

[32] A.K. Kelmans: "Constructions of Cubic Bipartite 3–connected Graphs without Hamiltonian Cycles", American Mathematical Society Translations, Series 2, **158**, 127–140 (1994), ISSN 0065–9290

[33] Takao Asano, Nobuji Saito, Geoffrey Exoo, Frank Harary: "The smallest 2–connected cubic bipartite planar nonhamiltonian graph", Discrete Mathematics **38**, 1–6 (1982)

[34] W.T. Tutte: "On hamiltonian circuits", Journal of the London Mathematical Society **21**, 98–101 (1946)

[35] Joseph Zaks: "Non–Hamiltonian non–Grinbergian graphs", Discrete Mathematics **17**, 317–321 (1977)

[36] Joseph Zaks: "Non–Hamiltonian cubic planar graphs having just two types of faces", Annals of Discrete Mathematics **9**, 225–227 (1980)

[37] Hansjoachim Walther: "Note on two problems of J. Zaks concerning Hamiltonian 3–polytopes", Discrete Mathematics **33**, No. 1, 107–109 (1981)

[38] Peter John Owens: "Regular planar graphs with faces of only two types and shortness parameters", Journal of Graph Theory **8**, No. 2, 253–275 (1984)

[39] Michal Tkáč: "On shortness coefficients of simple 3–polytopal graphs with only one type of faces besides triangles", Discrete Mathematics **128**, No. 1–3, 407–413 (1994)

[40] Peter John Owens: "Non–Hamiltonian simple 3–polytopes whose faces are all 5–gons or 7–gons" Discrete Mathematics **36**, No. 2, 227–230 (1981)

[41] Gunnar Brinkmann, Michel Deza: "Lists of face–regular polyhedra", preprint, Ecole Normale Supérieure, Département de Mathématiques et Informatique, Liens–98–13

[42] Andreas Dress and Gunnar Brinkmann: "Phantasmagorical Fulleroids", Match **33** (MATCDY), 87–100 (1996), ISSN 0340–6253

[43] Olaf Delgado Friedrichs, Michel Deza: "More icosahedral fulleroids", preprint, Ecole Normale Supérieure, Département de Mathématiques et Informatique, Liens–98–6

[44] Gunnar Brinkmann: "Isomorphism rejection in structure enumeration programs", to appear in the Proceedings of the 1998 DIMACS Workshop on Discrete Mathematical Chemistry

[45] Brendan D. McKay: "Isomorph–Free Exhaustive Generation", Journal of Algorithms **26**, 306–324 (1998), Article No. AL970898

[46] Thomas Harmuth: "Die Generierung simpler, 3–regulärer, planarer, zusammenhängender Graphen mit vorgegebenen Flächengrößen", Diplomarbeit, Universität Bielefeld (1996)

[47] Ernest Jucovič: "On the Number of Hexagons in a Map", Journal of Combinatorial Theory **10**, 232–236 (1971)

[48] David Barnette: "On p–Vectors of 3–Polytopes", Journal of Combinatorial Theory **7**, 99–103 (1969)

[49] S. Jendrol: "On the face–vectors and vertex–vectors of maps", Colloquia Mathematica Societatis János Bolyai, **18.** Combinatorics, Keszthely (Hungary), 629–633 (1976)

[50] Branko Grünbaum: "Convex Polytopes", 271–279

[51] R. Bruce King: "Some aspects of the symmetry and topology of possible carbon allotrope structures", Journal of Mathematical Chemistry **23** (Girona 97 special issue), no. 1–2, 197–227 (1998)

[52] Gunnar Brinkmann: "Generating cubic graphs faster than isomorphism checking", Preprint 92-047, Universität Bielefeld, Sonderforschungsbereich 343 (Diskrete Strukturen in der Mathematik)

[53] Gunnar Brinkmann: "Fast Generation of Cubic Graphs", Journal of Graph Theory **23**, No. 2, 139–149 (1996)

[54] Brendan D. McKay: "`autoson` — a distributed batch system for UNIX workstation networks (version 1.3)", manual, March 1996, Australian National University, TR-CS-96-03

Index