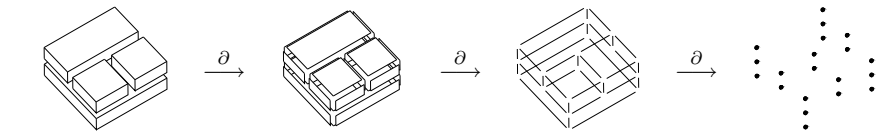
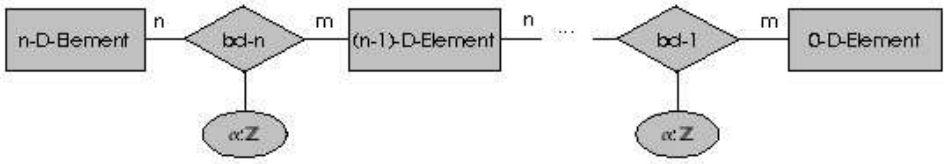
Topologie als Grundlage für Gebäudeinformationssysteme

Patrick Erik Bradley, Norbert Paul

29. Februar 2008





Zusammenfassung

Dieser technische Bericht fasst die erzielten Ergebnisse und durchgeführten Arbeiten im DFG-Projekt „Architektonische Komplexe“ ausführlich zusammen.

Ausgehend von der Annahme, dass der architektonische Entwurf auch eine endliche Zerlegung des Raums nach gewissen Regeln ist, findet Architektur über Kettenkomplexe, oder allgemeiner über endliche topologische Räume, Eingang in die Datenbanktheorie. Dies kommt von der Anforderung, Architektur am Computer zu modellieren und in diesen Datenmodellen möglichst viele räumliche Eigenschaften geeignet abzuspeichern. Die sogenannte *kategorielle* Sichtweise erleichtert dabei, gewisse relationale Datenbankschemata mit räumlichen Strukturen zu vergleichen und somit die Ausprägungen dieser Schemata selbst als Räume zu betrachten.

Mit der Kategorie $\mathcal{DK}et\mathcal{K}omp$ können Kettenkomplexe derart in Datenbanken modelliert werden, dass zu Grunde liegende topologische Komplexe mit relativ wenig Informationsverlust gespeichert werden und Beziehungen zwischen architektonischen Räumen vermöge Morphismen modelliert werden können.

Die Kategorie $\mathcal{DT}op$ erlaubt, jede beliebige Topologe für endliche Mengen verlustfrei mit relationalen Datenbanken zu modellieren und für die Grundbegriffe der Topologie Datenbankabfragen zu entwickeln. Damit ergibt sich ein allgemeines Datenmodell zum topologischen Modellieren mit relationalen Datenbanken.

1 Einleitung

Die Entwicklung von räumlich und zeitlich konsistenten Produktmodellen für den Lebenszyklus von Gebäuden führt immer auch zu einer Beschäftigung mit deren topologischen Eigenschaften. Topologie ist somit eine Grundlage der Modellierung und ist auch zumindest implizit in allen bisherigen Ansätzen zur Gebäudemodellierung vorhanden.

In der Mathematik werden topologische Räume mit bestimmten algebraischen Eigenschaften als Komplex bezeichnet. Diese sind zur Beschreibung von Architektur (oder ähnlich strukturierten Räumen wie z.B. Schiffe oder geologische Schichtungen) geradezu prädestiniert. Im DFG-Projekt „Architektonische Komplexe“ sollten architektonische Räume als dreidimensionale Komplexe der Form

$$\text{Volumen} \xrightarrow{\partial_V} \text{Flächen} \xrightarrow{\partial_F} \text{Kanten} \xrightarrow{\partial_K} \text{Punkte}$$

formal beschrieben werden, um daraus ein Datenmodell zu entwickeln und in einem Prototyp den Ablauf Aufmass → Zeichnen → Planen zu integrieren. Der hier vorliegende Bericht ist eine Zusammenfassung der wichtigsten Erkenntnisse aus diesem Projekt.

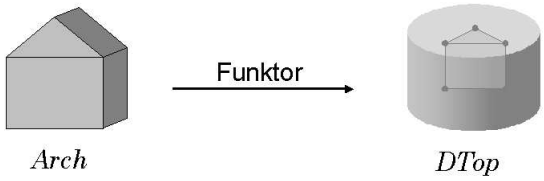


Abbildung 1: Funktor von Architektur in die topologischen Datenbanken.

Die erzielten Ergebnisse lassen sich als Entwicklung der Kategorie der topologischen Datenbanken und Beiträge zu Funktoren von Architektur als Kategorie in diese topologischen Datenbanken beschreiben. Abbildung 1 illustriert dies, und die folgenden Abschnitte erläutern dies ausführlich.

2 DKetKomp

Formaler Ausgangspunkt für die Betrachtung architektonischer Räume ist die Theorie der endlichen CW-Komplexe. Im Folgenden wird erörtert, wie im Projektverlauf gesehen wurde, dass CW-Komplexe allgemein in relationalen Datenbanken gehalten werden können und wie aus Architektur CW-Komplexe entstehen.

2.1 Von CW-Komplexen zu Kettenkomplexen

Wir skizzieren der Vollständigkeit halber, wie aus CW-Komplexen Kettenkomplexe gemacht werden. Dies ist ein Standardverfahren der algebraischen Topologie und kein origineller Projektbeitrag. Es ist auch in [4, §4.3] ausführlich beschrieben (Siehe auch [8, S. 137ff]).

Sei X ein CW-Komplex mit Zellen $\mathfrak{X}$ bzw. n-Zellen $\mathfrak{X}_n \subseteq \mathfrak{X}$. Die X zugeordneten Räume von n-Ketten $C_n(X)$ sind die formalen Summen der n-Zellen:

$$\alpha_1 b_1 + \cdots + \alpha_m b_m \qquad (b_i \in \mathfrak{X}_n,\ \alpha_1, \ldots, \alpha_m \in \mathbb{Z}). \tag{1}$$

Der nun zu definierende Randoperator $d_{n+1}\colon C_{n+1} \to C_n$ ordnet jeder n+1-Zelle c eine Kette $d_{n+1}(c)$ zu, die analog (1) beschaffen ist, und deren Koeffizienten $\alpha_i = (c : b_i)$ sich aus den Verklebeabbildungen des topologischen Zellenrands ∂c in das n-Skelett X^n des CW-Komplexes

$$\gamma_c\colon \partial c \cong S^n \to X^n$$

wie folgt ergeben: Der Zellenrand ∂c ist zunächst homöomorph zur n-Sphäre S^n. Kontraktion des Komplements der n-Zelle $b = b_i$ im n-Skelett ergibt einen topologischen Raum, der wiederum zur n-Sphäre homöomorph ist. Verkettung der Abbildung γ_c und der Kontraktionsabbildung $X^n \to S^n$ ergibt eine stetige Abbildung

$$f_{bc}\colon\ S^n \xrightarrow{\gamma_c} X^n \xrightarrow{\text{kontr.}} S^n\ .$$

Für den Fall $n = 1$ ist dies in Abbildung 2 illustiert.

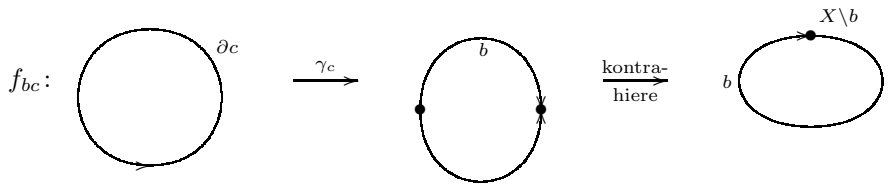


Abbildung 2: Der Grad von f_{bc} ist der Matrixeintrag $(c : b)$.

Der Koeffizient $(c : b_i)$ ist dann der Grad $\deg f_{b_i c}$ der Abbildung $f_{b_i c} : S^n \to S^n$. In Abbildung 2 beispielsweise ergibt sich auf Grund der Orientierungsumkehr als Grad von f_{bc} eine negative ganze Zahl, falls f_{bc} surjektiv ist (und 0 andernfalls).

Bezüglich der Basis $\mathfrak{X}$ für den Kettenmodul $C(X) = \bigoplus_{n\in\mathbb{N}} C_n(X)$ ist der sich aus den Randoperatoren $d_{n+1}\colon C_{n+1}(X) \to C_n(X)$ ergebende Differentialoperator $d\colon C(X) \to C(X)$ durch die quadratische ganzzahlige Matrix

$$M_d := (\deg f_{bc})_{b,c\in\mathfrak{X}}$$

gegeben.

2.2 Von Architektur zu CW-Komplexen

Architektur ist in seiner allgemeinsten Form eine Gestaltung eines wie auch immer gearteten Raumes X. Dabei wird unter X in der Regel der dreidimensionale euklidische Raum $\mathbb{R}^3$ verstanden. Es ist jedoch hier nicht notwendig, sich auf eine bestimmte Dimension festzulegen. Zudem muss es sich bei X noch nicht einmal um einen euklidischen Raum handeln, weshalb wir zunächst nur annehmen, dass X ein beliebiger topologischer Raum sei. Nun beobachten wir, dass Architektur den Raum X in endlich viele Teile (z.B. geometrische Figuren) partitioniert. Anstelle der einzelnen Punkte des Raumes X betrachten wir nun die Menge der „Zusammenfassungen“ dieser Punkte zu „Figuren“ (bzw. Äquivalenzklassen), i.a.W. den sog *Quotientenraum*. Damit ordnet Architektur dem topologischen Raum X den endlichen topologischen Quotientenraum $\bar{X}$ zu, welcher durch diejenige Äquivalenzrelation $\sim$ auf X gebildet wird, welche die durch die Architektur gegebene Partitionierung auf X induziert. Die Äquivalenzrelation $\sim$ nennen wir eine *endliche Darstellung* des Raums X.

Wir setzen nun voraus, dass X kompakt sei. Eine *architektonische Darstellung* von X ist dann eine endliche Darstellung $\sim$, für welche gilt:

Jede Äquivalenzklasse für $\sim$ ist homöomorph zu einer zusammenhängenden offenen Teilmenge eines euklidischen Raumes $\mathbb{R}^q$ mit $q \in \mathbb{N}$.

Hierbei nennen wir die Zahl q die *Dimension* der fraglichen Äquivalenzklasse. So hat z.B. die geometrische Figur „Wandoberfläche“ Dimension 2, und eine Figur „Gebäudekante“ hat Dimension 1.

Es ist nun folgendes Postulat [4, Postulate 1] durchaus naheliegend:

Postulat 1. *Architektonisches Entwerfen ist die Festlegung einer architektonischen Darstellung eines kompakten Raums X.*

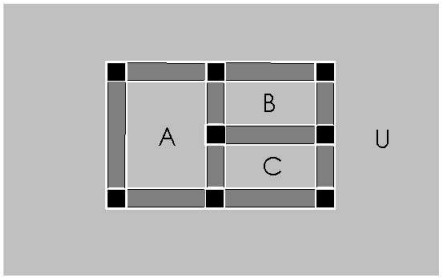


Abbildung 3: Grundriss als architektonische Darstellung.

Hierbei ist der Raum X meist ein kompakter Teilraum von $\mathbb{R}^3$. In Abbildung 3 beispielsweise definiert der Grundriss mit den Innenräumen eine architektonische Zerlegung (bzw. Darstellung) eines ebenen Rechtecks.

Gegebenenfalls bieten sich für X auch Kompaktifizierungen von $\mathbb{R}^n$ an, damit der „Außenraum“ etwa in Form einer ausgezeichneten Äquivalenzklasse mit berücksichtigt wird.

Die Kompaktheit selbst wird eigentlich erst für das nachfolgende Postulat [4, Postulate 2] benötigt:

Postulat 2. *Eine architektonische Darstellung in der Praxis lässt sich stets derart verfeinern, dass dem dadurch gegebenen Quotientenraum in natürlicher Weise die Struktur eines endlichen CW-Komplexes auferlegt werden kann.*

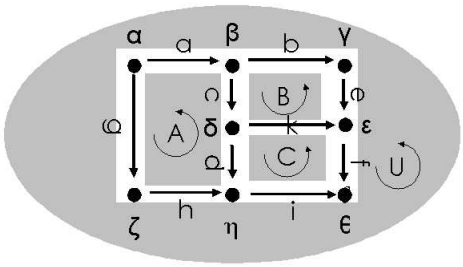


Abbildung 4: Grundriss als CW-Komplex.

Eine Verfeinerung wie in Postulat 2 unterteilt dabei die q-dimensionalen Äquivalenzklassen in p-Zellen, deren Dimension p kleiner oder gleich q sein kann. Damit können Verklebeabbildungen für einen CW-Komplex definiert werden. Stößt etwa die Achsebene einer leichten Trennwand (2-Zelle) an die einer massiven Wand (weitere 2-Zelle), dann bildet dies noch keinen CW-Komplex, da die Ebene der Trennwand als 2-Zelle ja höchstens an 1-Zellen angeheftet werden darf, die der massiven Wand aber ebenfalls 2-Zelle ist. Jedoch kann man sich diese Ebene der massiven Wand in drei Teile unterteilt denken: Jeweils ein Ebeneneabschnitt links bzw. rechts vom Stoß und eine gedachte Verbindungskante (1-Zelle), an der diese zwei Ebenenabschnitte mit der Trennwand, verbunden sind. Die Verklebeabbildung heftet die Ebenen an der Verbindungskante zusammen.

Abbildung 4 zeigt den Grundriss von Abbildung 3 als CW-Komplex mit orientierten Zellen. Dabei sei die euklidische Ebene per Einpunktkompaktifizierung erweitert, damit auch der Außenraum als Zelle U angesehen werden kann.

2.3 Die Kategorie der relationalen Kettenkomplexe

Die ursprüngliche Intention der nun folgenden relationalen Datenbankschemata war, ein einfaches Datenbankschema für Komplexe zu definieren, auf das sich dann alle weitere Datenstrukturen zur topologischen Modellierung reduzieren lassen sollten. Damit eignet es sich als ein Bezugsmodell zur Untersuchung der Effektivität aller weiteren Modelle und muss dazu nicht notwendigerweise effizient sein.

In Abschnitt 2.1 wurde zu jedem CW-Komplex ein Kettenkomplex zugeordnet. Tabellen 1 und 2 zeigen die Matrizen M_1 und M_2 der zugehörigen Ran-

M_1	a	b	c	d	e	f	g	h	i	k
α	-1	0	0	0	0	0	-1	0	0	0
β	1	-1	-1	0	0	0	0	0	0	0
γ	0	1	1	0	-1	0	0	0	0	0
δ	0	0	0	-1	0	0	0	0	0	-1
ε	0	0	0	0	1	-1	0	0	0	1
ζ	0	0	0	0	0	0	1	-1	0	0
η	0	0	0	1	0	0	0	1	-1	0
θ	0	0	0	0	0	1	0	0	1	0

Tabelle 1: Matrix M_1 for Randoperator d_1.

doperatoren $d_1 \colon C_1 \to C_0$ und $d_2 \colon C_2 \to C_1$ des in Abbildung 4 gezeigten CW-Komplexes.

M_2	A	B	C	U
a	-1	0	0	1
b	0	-1	0	1
c	-1	1	0	0
d	-1	0	1	0
e	0	-1	0	1
f	0	0	-1	1
g	1	0	0	-1
h	-1	0	0	-1
i	0	0	1	-1
k	0	1	-1	0

Tabelle 2: Matrix M_2 für Randoperator d_2.

Es ist also naheliegend, dasselbe relationale Datenbankschema für CW-Komplexe und für Kettenkomplexe (mit fester Basis) zu definieren: eine Tabelle für die Zellen mit dem Schlüsselattribut *Cell*, dem Attribut Dimension und ggf. noch weiteren Attributen, sowie eine Tabelle für die Matrix M_d des Differentialoperators mit den Schlüsselattributen *Cell_1, Cell_2*, den Matrixkoeffizienten als Attributen und ggf. noch weiteren Attributen.

Die Matrix M_d bei Kettenkomplexen weist eine Blockstruktur auf und hat damit viele Nulleinträge. Für unser Grundrissbeispiel etwa ist

$$M_d = \begin{pmatrix} 0 & M_1 & 0 \\ 0 & 0 & M_2 \\ 0 & 0 & 0 \end{pmatrix}$$

bei geeigneter Reihenfolge der Zellen. Dies legt nahe zu versuchen, die Anzahl der Datenbankeinträge für M_d zu minimieren. In der Tat lässt sich dies mit *partiellen Matrizen* durchführen. Eine partielle ganzzahlige Matrix ist eine partielle Abbildung

$$A :\subseteq I \times J \to \mathbb{Z}$$

mit den Indexmengen I und J. Sind I und J Basen für freie Moduln V und W, so wird durch die partielle Matrix A eine *partielle lineare Abbildung*

$$a : V_{\subseteq} \to W_{\subseteq}$$

erklärt. Man beachte, dass eine partielle lineare Abbildung selbst eine totale Abbildung von den partiellen Abbildungen $V_{\subseteq} := \{v :\subseteq I \to \mathbb{Z}\}$ zu $W_{\subseteq} := \{w :\subseteq J \to \mathbb{Z}\}$ ist.

Für eine kurze Einführung in die partielle lineare Algebra verweisen wir auf den Anhang A, wo sich auch die naheliegenden Rechenregeln für partielle Matrizen finden. Diese erlauben schließlich die Definition des relationalen Kettenkomplexes:

Definition. *Ein* relationaler Kettenkomplex *ist ein Tripel* (M, B, Δ) *bestehend aus einem freien Modul*

$$M = \bigoplus_{n\in\mathbb{N}} M_n$$

mit Basis $B = \bigcup_{n\in\mathbb{N}} B_n$*, sodass* B_n *Basis für* M_n *ist, sowie einer partiellen Matrix*

$$\Delta :\subseteq B \times B \to \mathbb{Z}$$

mit $\Delta \cdot \Delta = 0$ *und* $\delta(M_{n+1}) \subseteq M_n$ *für die durch* Δ *gegebene partielle lineare Abbildung* $\delta : M_{\subseteq} \to M_{\subseteq}$*.*

Hier bedeutet 0 eine partielle Nullmatrix, d.h. alle definierten Einträge sind Null. Damit hat Δ bei geeigneter Reihenfolge der Basisvektoren folgende Blockstruktur:

$$\Delta = \begin{pmatrix} * & \Delta_1 & & & \\ & * & \Delta_2 & & \\ & & \ddots & \ddots & \\ & & & \ddots & \Delta_m \\ & & & & * \end{pmatrix} \tag{2}$$

mit ganzzahligen partiellen Matrizen $\Delta_1, \ldots, \Delta_m$ als den Blöcken und Einträgen 0 oder $\uparrow$ (= undefiniert) außerhalb der Blöcke.

Ein relationaler Kettenkomplex zu einem CW-Komplex X wird nun wie folgt definiert: Zunächst gibt es eine partielle Ordnung $\leq$ auf den Zellen $\mathfrak{X}$ von X:

$$b \leq c \quad \Leftrightarrow \quad b \text{ liegt im Abschluss von } \{c\}.$$

Dann erklären wir

$$\Delta :\subseteq \mathfrak{X} \times \mathfrak{X} \to \mathbb{Z}$$

als partielle Matrix, welche in (b, c) genau dann definiert ist, wenn $b \neq c$ und b maximal bezüglich $\leq$ mit der Eigenschaft $b \leq c$ ist. In diesem Fall setzen wir

$$\Delta(b, c) := D(b, c),$$

wobei $D(b, c)$ der an der Stelle (b, c) vorkommende Matrixeintrag des Randoperators d für den Kettenkomplex zu X ist. Auf diese Weise wird jede Redundanz vermieden, und jede Berührеigenschaft von Zellen im CW-Komplex ist per Abfrage rekonstruierbar.

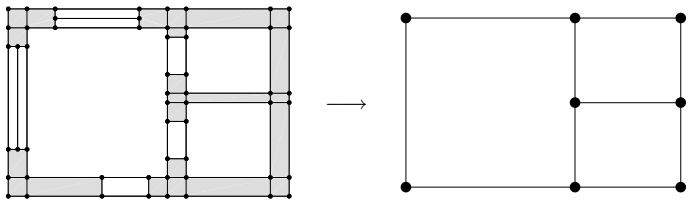


Abbildung 5: Zellluläre Abbildung σ: Werkplan $\to$ Entwurfsplan.

Es ist von großer Bedeutung für Modellierung und Datenhaltung, verschiedene Detaillierungsgrade in Beziehung setzen zu können. Dies erreichen wir mit stetigen Abbildungen und deren relationalen Entsprechungen. Genauer gesagt ist eine kategorielle Betrachtungsweise von Nöten. Hierzu hat jede Kategorie *Objekte* und *Morphismen*, welche die Objekte in Beziehung setzen.

In der Kategorie $\mathcal{CWK}omp$ sind die Objekte die CW-Komplexe und die Morphismen die zellulären Abbildungen. Letztere sind stetige Abbildungen, welche n-Zellen in Zellen von höchstens Dimension n abbilden. Als Beispiel betrachten wir die Abbildung

$$\sigma\colon \text{Werkplan} \to \text{Entwurfsplan}$$

aus Abbildung 5. Ein Entwurfsplan sei in der Gestalt eines CW-Komplexes gegeben (Abbildung 5 rechts). Bei der Gestaltung des Werkplans (Abbildung 5 links) wird aus jeder Zelle b des Entwurfsplans ein eindeutig bestimmter Teilkomplex σ_b des Werkplans erstellt. Dies geschieht normalerweise dergestalt, dass die Zuordnung

$$c \mapsto b, \quad \text{falls } c \in \sigma_b$$

eine zelluläre Abbildung σ definiert, welche wir *Skizzierung* nennen. Das praktische Handeln beim Entwurf geht nun in die andere Richutung: Jedem Element des Entwurfsplans ist eine ausführlicher detaillierte konstruktive Lösung im Werkplan zugeordnet. Das Urbild $\sigma^{-1}(e)$ eines Elements e im Entwurfsplan könnte man also als das *Detail* von e bezeichnen. Da sich nun ein solches Detail oft wiederholt, erscheint es sinnvoll, es nicht jedesmal explizit im Werkplan zu speichern, sondern jeweils nur ein Exemplar anzugeben, das dann aus der Skizze heraus referenziert wird. Das Zusammensetzen dieser Details zu einem Werkplan anhand einer derartigen Skizze entspricht dem *Gleichheitsverbund* aus der Datenbanktheorie bzw. dem *Faserprodukt* in der Kategorientheorie und wird im Anschnitt 3.5 noch ausführlich behandelt.

Die Kategorie $\mathcal{K}et\mathcal{K}omp$ der Kettenkomplexe hat als Morphismen lineare Abbildungen, welche n-Ketten auf n-Ketten abbilden. Bei festgelegten Basen bekommt man für jede lineare Abbildung eine eindeutig bestimmte Matrix. Damit

können wir die Kategorie $\mathcal{DKetKomp}$ definieren: die Objekte sind die relationalen Kettenkomplexe und die Morphismen sind die partiellen Matrizen, die zu partiellen linearen Abbildungen zwischen partiellen Kettenkomplexen gehören, welche n-Ketten auf n-Ketten abbilden. Anhang A erklärt die hierzu nötige partielle lineare Algebra.

Insbesondere gibt es auch eine Datenbankentsprechung für zelluläre Abbildungen zwischen CW-Komplexen. Auf die explizite Angabe der partiellen Matrix für die Skizzierung σ in Abbildung 5 verzichten wir jedoch.

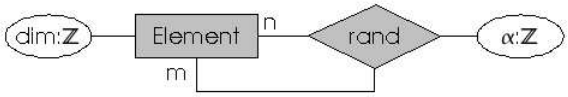


Abbildung 6: ER-Schema für relationale Kettenkomplexe.

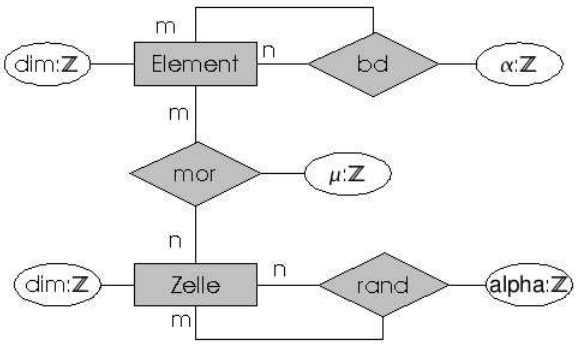


Abbildung 7: ER-Schema für relationale Kettenkomplexmorphismen.

Mit der Kategorie $\mathcal{DKetKomp}$ wurde also ein recht einfaches Datenbankschema für architektonische Räume gefunden. Es handelt sich dabei um ein Datenbankschema, bei dem die Dimension ein Attribut ist. Damit ist die Dimension eine dynamische Eigenschaft.

Hiermit werden Komplexe beliebiger endlicher Dimension nach einem einfachen, einheitlichen Datenbankschema darstellbar. Die einzige Voraussetzung hierfür ist, neben der Endlichkeit der Zellenanzahl, die Berechenbarkeit der transitiven Hülle per Abfrage, falls die Dimension nicht beschränkt ist [4, 14].

Die zu den relationalen Kettenkomplexen und den relationalen Kettenkomplexmorphismen gehörigen ER-Schemata sind in den Abbildungen 6 und 7 wiedergegeben.

2.4 Vergleich der Informationsverluste

Beim Übergang von einer Kategorie in eine andere treten im Allgemeinen Informationsverluste auf. Wir beschreiben im Folgenden für verschiedene Übergänge die wichtigsten im Projektverlauf vorgefundenen Arten derartiger Verluste und diskutieren diese.

2.4.1 Verluste von Architektur zu CW-Komplexen

Der Übergang von Architektur zu CW-Komplexen entsteht nach Theorem 2 durch Verfeinerung einer nach dem Postulat gegebenen architektonischen Darstellung eines Kompaktums X. Nach Verfeinerung ist die Information, welche Zellen des aus X gebildeten CW-Komplexes ursprünglich eine gemeinsame Äquivalenzklasse bildeten, in der Regel verloren. Abhilfe kann indirekt dadurch geschaffen werden, dass nach Übergang zu $\mathcal{DKetKomp}$ jede Zelle ein Attribut „architektonische Klasse“ erhält.

2.4.2 Verluste von CW-Komplexen zu Kettenkomplexen

Beim Übergang von $\mathcal{CWKomp}$ zu $\mathcal{KetKomp}$ gibt es zwei Arten von Informationsverlusten.

1. Art. Ein Verlust erster Art entsteht dadurch, dass eine der von den Anheftungsabbildungen induzierte Abbildung $f_{bc}\colon S^n \to S^n$ nicht surjektiv ist. Dann ist nämlich der entsprechende Matrixeintrag $(c : b)$ des Randoperators d gleich Null. Ein wichtiges Beispiel dafür ist die Anheftung einer $n+1$-Zelle b mit ihrem Rand an eine 0-Zelle c — nach Kontraktion ist dann nämlich das Bild von f_{bc} ein Punkt. In diesem Fall kann in der Matrix M_d für d nicht mehr abgelesen werden, an welcher 0-Zelle die Zelle b geheftet wurde. Ein Beispiel von nicht isomorphen Graphen mit isomorphen Kettenkomplexen findet sich in [4, Ex. 6.15] oder [16, Bsp. 5.2].

2. Art. Ein Verlust zweiter Art rührt daher, dass bei Kettenkomplexen die Reihenfolge der n-Zellen c_i im Rand einer $n+1$-Zelle b keine Rolle spielt:

$$db = c_1 \pm \cdots \pm c_m \quad \text{(die } c_i \text{ nicht notwendig verschieden).}$$

Bei zwei CW-Komplexen X, X' mit isomorphen Kettenkomplexen ist die Reihenfolge beim Durchlauf des topologischen Randes von b gemäß gewählter Orientierung jedoch von Bedeutung. Es kann nämlich $X \ncong X'$ dennoch eintreten. [4, Example 6.19] interpretiert dahingehend ein Beispiel von Hatcher [8, Example 2.38]

2.4.3 Verluste von CW-Komplexen zu relationalen Komplexen.

Zunächst halten wir fest, dass jeder Informationsverlust, der beim Übergang $\mathcal{CWKomp} \to \mathcal{DKetKomp}$ auftritt, ebenfalls beim Funktor $\mathcal{CWKomp} \to \mathcal{KetKomp}$ auftritt. Haben nämlich zwei verschiedene CW-Komplexe X, X' isomorphe relationale Kettenkomplexe, so stimmen deren partielle Differentialoperatoren

überein, mithin ihre durch Nullen aufgefüllten totalen Differentialoperatoren in *KetKomp*. Ähnliches gilt für die partiellen/totalen Matrizen, welche von zellulären Abbildungen $f\colon X \to Y$ und $f'\colon X' \to Y'$. herrühren.

Auf Grund der Wahl der partiellen Matrizen tritt kein Verlust der ersten Art mehr auf beim Übergang $\mathcal{CWK}omp \to \mathcal{DK}et\mathcal{K}omp$. Den Verlust zweiter Art gibt es jedoch auch hier. Dieser ist für viele Anwendungen aber unerheblich.

2.5 Architektonische Komplexe in $\mathcal{DK}et\mathcal{K}omp$

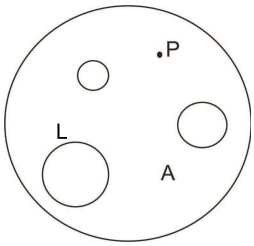


Abbildung 8: Eine 2-Region mit mehreren Randkomponenten.

Die Verwendung von partiellen anstelle von totalen Matrizen führte, wie wir sahen, bei CW-Komplexen zur Reduzierung topologischer Informationsverluste. Es stellte sich heraus, dass $\mathcal{DK}et\mathcal{K}omp$ geeignet ist, allgemeinere topologische Komplexe darzustellen, die wir *architektonische Komplexe* nennen.

Abbildung 8 illustriert eine gelochte und punktierte Kreisscheibe als architektonischen Komplex: eine 2-Region A mit 4 Randkomponenten, die 1-Sphären sind und einer Randkomponente, die ein Punkt P ist. Somit ist der partielle Differentialoperator Δ in $\mathcal{DK}et\mathcal{K}omp$ gegeben durch

$$\Delta(x,y) = \begin{cases} 0, & \text{falls } x \neq A \text{ und } y = A \\ \uparrow, & \text{sonst.} \end{cases}$$

Für eine geeignete Reihenfolge der Basisvektoren ist also die zu A gehörige Spalte in der partiellen 6×6-Differentialmatrix Δ gleich

$$\Delta A = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \uparrow \end{pmatrix}.$$

Die 1-Sphäre L hingegen hat keinen Rand. Deshalb ist die Spalte ΔL leer:

$$\Delta L = \begin{pmatrix} \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \end{pmatrix}.$$

Es ergeben sich charakterisierende Abweichungen vom relationalen Kettenkomplex eines CW-Komplexes z.B. durch undefinierte Spalten.

Ist $C = (M, B, \Delta)$ ein relationaler Kettenkomplex, so definieren wir folgenden gewichteten Graph $\Gamma(C)$: Die Knotenmenge sei B, und $b \in B_n$ habe Knotengewicht n. Die Kanten seien die Paare $e = (a, b) \in B \times B$, für die $\Delta(a, b)$ definiert ist. Das Gewicht der Kante e sei die ganze Zahl $\alpha_e = \Delta(a, b)$. Den gewichteten Graph $\Gamma(C)$ nennen wir *architektonischen Komplex*, die Elemente von B_n heißen *n-Regionen*. Die Adjazenzmatrix A von $\Gamma(C)$ ist offenbar die mit Nullen aufgefüllte Matrix Δ und erfüllt die Gleichung $A^2 = 0$.

Mit Morphismen $f\colon \Gamma(C) \to \Gamma(C')$ der unterliegenden Graphen, für welche $f(B_n) \subseteq B'_{m'}$ mit $m \leq n$ gilt, bekommen wir eine Kategorie $\mathcal{A}rch\mathcal{K}omp$, die sich von $\mathcal{DK}et\mathcal{K}omp$ im Wesentlichen nur in den Morphismen unterscheiden.

Als architektonische Beispiele hierfür bieten sich beispielsweise Kirchenschiffe mit Säulen oder Fassaden mit Fenster an. In [5] wird mit diesem Ansatz die Entwicklung von CAD-GIS-Systemen angestrebt. Eine Anwendung im Kontext raumzeitlicher urbaner Daten ist in [3] angedacht.

2.6 Euleroperatoren in $\mathcal{DK}et\mathcal{K}omp$

Offenbar gilt die Euler-Poincaré-Formel in $\mathcal{K}et\mathcal{K}omp$ (vgl. Beweis von [8, Theorem 2.44]): für einen Komplex C freier Moduln C_n mit endlichen Basen B_n der Kardinalität c_n gilt die Gleichung

$$\sum_{n\in\mathbb{N}} (-1)^n c_n = \sum_{n\in\mathbb{N}} (-1)^n b_n, \tag{3}$$

wobei b_n die n-te Bettizahl von C ist. Eine an die partielle lineare Algebra angepasste Definition der Homologie ergibt für jeden relationalen Kettenkomplex $\mathcal{C}$:

$$H_n(\mathcal{C}, \mathbb{Z}) \cong H_n(\mathcal{C}^{\mathrm{tot}}, \mathbb{Z}),$$

wobei $\mathcal{C}^{\mathrm{tot}}$ der zu $\mathcal{C}$ gehörige totale Kettenkomplex ist (Lemma A.9). Die Bettizahlen von $\mathcal{C}$ und $\mathcal{C}^{\mathrm{tot}}$ stimmen also überein. Da sich die Kardinalitäten c_n beim Übergang zu $\mathcal{C}^{\mathrm{tot}}$ nicht ändern, gilt somit die Euler-Poincaré-Formel auch in $\mathcal{DK}et\mathcal{K}omp$.

Gleichung (3) beschreibt eine Hyperebene $\mathcal{H}$ in $\mathbb{Z}^{2m+2}$, wenn m die Dimension des Komplexes C ist. $\mathcal{H}$ heißt die *Euler-Poincaré-Ebene*. Von Interesse sind die Lösungen von (3) mit nicht negativen Koordinaten. Diese bilden den

Teilraum $\mathcal{H}_+$. Jeder Punkt $x \in \mathcal{H}$ bestimmt eine Klasse von Kettenkomplexen mit denselben Zellen- und Bettizahlen. Gleichzeitig operiert $\mathcal{H}$ auf sich durch Translation. In diesem Sinne heißen die Elemente von $\mathcal{H}$ auch *Euler-Poincaré* [1, §2.3.2] oder *Euler-Poincaré-Operatoren* [2]. Es gibt offenbar eine Basis $\mathcal{B}$ von $\mathcal{H}$, die in $\mathcal{H}_+$ liegt. Die Euler-Poincaré-Operatoren aus $\mathcal{B}$ nennen wir *elementar*. Diese wiederum lassen sich algebraisch aus den *fundamentalen* Operatoren folgendermaßen zusammen bauen: Es bezeichne X^i einen Operator auf $\mathbb{Z}^{2m+2}$, der im Fall $0 \leq n \leq m$ die Größe c_i und im Fall $m+1 \leq n \leq 2m+1$ die Bettizahl $b_{n-(m+1)}$ um Eins erhöht. Dann sind durch $1 - X^m$ und $1 + X^{m+1}$ Operatoren gegeben, welche jeweils einen Knoten hinzufügen und eine m-Zelle entfernen bzw. eine Komponente hinzufügen. Beachte, dass $1 + X^{m+1}$ für jede Dimension m ein Euler-Poincaré-Operator ist, jedoch $1 - X^m$ genau dann die Euler-Poincaré-Formel respektiert, wenn m gerade ist. Es gilt gemäß [2, §6]:

Satz. *Die Euler-Poincaré-Operatoren*

$$\begin{aligned} A_i &:= X^i \cdot (1 - X^m), & i &= 1, \ldots, m \\ B_j &:= X^j \cdot (1 + X^{m+1}), & j &= 0, \ldots, m \end{aligned}$$

bilden einen vollständigen Satz elementarer Euler-Poincaré-Operatoren, d.h. jeder Euler-Poincaré-Operator aus $\mathcal{H}_+$ ist eine $\mathbb{Z}$-Linearkombination der A_i, B_j.

Euler-Poincaré-Operatoren lassen sich also wahlweise als ganzzahlige Polynome in der Variablen X oder vermöge der elementaren Operatoren A_i, B_j ausdrücken. Umrechnungsformeln zwischen beiden Darstellungsarten werden in [2, §6] angegeben. Dabei muss aus technischen Gründen zwischen m gerade und ungerade unterschieden werden.

Die Bedeutung in der Anwendung lässt sich folgendermaßen beschreiben: Jede Änderung eines Kettenkomplexes $C \in \mathcal{H}_+$ zu einem Kettenkomplex C' muss die Konsistenzregel $d \circ d = 0$ erfüllen. Äquivalent hierzu ist, dass es eine Folge von elementaren Euler-Poincaré-Operatoren $E_1, \ldots, E_r \in \mathcal{B}$ gibt, sodass

$$C' = C \pm E_1 \pm \cdots \pm E_r$$

gilt. Dasselbe gilt in *DKetKomp*. Es ist leicht einsehbar, dass die Konsistenzregel $\delta \circ \delta = 0$ als Gleichung partieller linearer Operatoren äquivalent zu einer Folge von Euler-Poincaré-Operatoren ist, die einen gegebenen relationalen Kettenkomplex in einen anderen überführen. Folglich können in *DKetKomp* lokale Änderungen vermöge einer lokalen Konsistenzbedingung überprüft werden, was für eine effiziente Implementierung von Transaktionen von Bedeutung ist.

Bei der Implementierung der elementaren Euler-Poincaré-Operatoren wird die Konsistenzregel $\delta \circ \delta = 0$ lokal überwacht: Wird eine Zelle mit `addCell(...)` einem Komplex hinzugefügt, dann ist dabei eine Zelle `c` und deren Rand δ(`c`) anzugeben, die Methodensignatur ist also `addCell(Cell c, Boundary<Cell> dc)`. Die Methode testet, ob δ(`dc`) $= 0$ gilt und heftet in diesem Fall die Zelle entsprechend an den Komplex oder erzeugt andernfalls eine Ausnahme. Das Entfernen einer Zelle `c` ist mit `removeCell(Cell c)` nur dann zulässig, wenn

c nicht selbst Rand einer anderen Zelle ist. Dabei erhöht sich entweder die Bettizahl $b_{\dim \mathsf{c}-1}$ um eins, oder es vermindert sich $b_{\dim \mathsf{c}}$ um eins. Auch hier erzeugt ein unzulässiger Aufruf eine Ausnahme.

Im allgemeinen Fall der relationalen Kettenkomplexe haben die Bettizahlen keine topologische Interpretation. Eine Ausnahme bilden diejenigen, welche sich durch Anheften gewisser Mannigfaltigkeiten analog den CW-Komplexen aufbauen lassen. Diese Mannigfaltigkeiten werden in [2, §8] *topologische Regionen* genannt, die entstehenden Räume X heissen dort *topologische architektonische Komplexe*. Ihnen kann wie bei CW-Komplexen ein (relationaler) Kettenkomplex $\mathfrak{X}$ zugeordnet werden. Dabei musste erneut die Konsistenzregel $\partial \circ \partial = 0$ nachgewiesen werden [2, Prop. 8.5]. Also hat X die Bettizahlen $b_i^{\text{top}}(X)$ mit der üblichen topologischen Interpretation, aber auch die von $\mathfrak{X}$ herrührenden Bettizahlen $b_i(X) = b_i(\mathfrak{X})$. Es konnte gezeigt werden, dass stets die Ungleichung

$$b_i(X) \leq b_i^{\text{top}}(X)$$

gilt [2, Cor. 8.12].

Indem die Regionen topologischer architektonischer Komplexe sukzessive mit einer Zellstruktur versehen werden, lassen sich die topologischen Bettizehlen algorithmisch berechnen [2, Alg. 8.16]. Ein Spezialfall liegt vor, wenn lediglich die Randkomponenten einer topologischen Region zu einer einzigen Zusammenhangskomponente verbunden werden müssen. Beispielsweise handele es sich dabei um eine Lochfassade oder um einen Grundriss mit Innenhöfen. Hierfür wird in [2, Ex. 8.19, 8.22] der Algorithmus in expliziter Weise beschrieben.

An dieser Stelle sei angemerkt, dass Euler-Poincaré-Operatoren nur für die Objekte der Kategorie *DKetKomp* definiert sind. Auf dieselbe Weise können sie wie in [2] für die Objekte von *ArchKomp* definiert werden. Eine Entsprechung für Morphismen in *DKetKomp* oder *ArchKomp* steht hingegen noch aus.

3 DTop

Architektonische Darstellungen ergeben endliche topologische Räume. Diesen können in einfacher Weise Datenbanken zugeordnet werden, welche die Topologie verlustfrei repräsentieren.

3.1 Architektonische Komplexe in topologischen Datenbanken

Es stellt sich nun die Frage, wie die Topologie der relationalen Kettenkomplexe definiert ist. Nehmen wir einen relationalen Kettenkomplex (M, B, Δ), und definieren für alle Paare $(a, b) \in X \times X$, deren Matrixeintrag $\Delta(a, b)$ definiert ist, dass b im topologischen Rand von $\{a\}$ liege.

Damit definiert die partielle Matrix Δ eine Relation R_Δ in X, nämlich

$$(a, b) \in R_\Delta \quad \Leftrightarrow \quad \Delta(a, b) \neq \uparrow .$$

Diese Relation ist azyklisch, da für die Paare $(a, b) \in R_M$ gilt:

$$a \in B_m,\ b \in B_n \implies m < n.$$

Somit ist die transitive reflexive Hülle $(R_M)^*$ von R_M eine partielle Ordnung, und ihre Ordnungstopologie ist per Definition die Topologie der Regionen eines architektonischen Komplexes. Für CW-Komplexe erhalten wir so die Quotiententopologie auf den Zellen.

Allgemein definiert jede (nicht notwendig azyklische) Relation R in X folgende Topologie $\mathcal{T}_R$ auf X: Für eine Teilmenge A von X bezeichnen wir als *Stern von A bezüglich R* die Menge aller Punkte a in X, die mit einem $b \in A$ in der Beziehung R stehen:

Definition. *Für eine Menge X, eine Relation $R \subseteq X \times X$ und eine beliebige Teilmenge $A \subseteq X$ ist*

$$\mathrm{St}_R(A) := \{s \in X \mid sRa, a \in A\}$$

der Stern von A bezüglich R.

Die Menge der Teilmengen A von X, die ihren eigenen Stern umfassen, bildet eine Topologie $\mathcal{T}_R := \{A \subseteq X \mid \mathrm{St}_R(A) \subseteq A\}$, die wir als als die *von R erzeugte Topologie* bezeichnen. Insbesondere kann $\mathrm{St}_R(A) \subseteq A$ in relationaler Algebra ohne transitive Hülle entschieden werden: Ohne Einschränkung sei $\{a\}$ Schlüssel von X, $\mathcal{A}$ Attributmenge von X und seien x, y die Attribute der Relation R und $x, y \notin \mathcal{A}$. Dann ist für $A \subseteq X$

$$\mathrm{St}_R(A) = \pi_{\mathcal{A}}(X \bowtie \beta_{a \leftarrow x}(R) \bowtie \beta_{y \leftarrow a}(\pi_a(A)))$$

und

$$\pi_a(\mathrm{St}_R(A)) = \pi_a(\beta_{a \leftarrow x}(R) \bowtie \beta_{y \leftarrow a}(A)).$$

Dann gilt

$$\pi_a(\mathrm{St}_R(A)) \setminus \pi_a(A) = \emptyset$$

genau dann, wenn A offen in $(X, \mathcal{T}_R)$ ist.

Umgekehrt hat jede endliche Topologie $\mathcal{T}$ eine derartige erzeugende Relation. Dies steht hinter der Aussage von [4, Theorem 7.6]. Somit gibt es für jede endliche Topologie $\mathcal{T}_X$ auf einer Menge X eine erzeugende Relation R und umgekehrt erzeugt jede Relation R in X eine derartige Topologie für X. Damit ergibt sich sofort eine (asymptotische) obere Schranke des Speicherbedarfs $b = b(\mathcal{T}_X)$ einer Topologie $\mathcal{T}_X$ in Abhängigkeit von der Größe $|X|$ der zugrundeliegenden Punktmenge X: Es gilt $b \in \mathcal{O}(|X|^2)$ [14, §4.3].

Da (X, R) ein simpler gerichteter Graph ist, ergibt sich eine denkbar einfache Realisierung mit relationalen Datenbanken. Wir bezeichnen ein solches Paar als *topologischen Datentyp* und eine Familie von topologischen Datentypen als *topologische Datenbank* und betrachten dabei jeweils den topologischen Raum $(X, \mathcal{T}_R)$ mit der von R erzeugten Topologie.

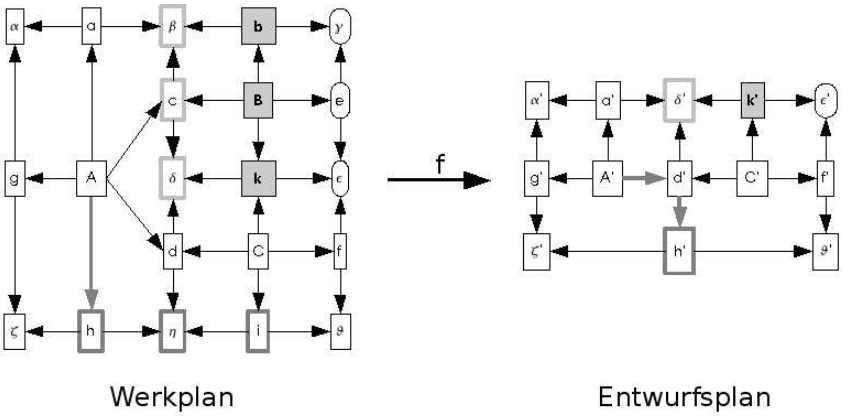


Abbildung 9: Eine stetige Abbildung f: Werkplan $\rightarrow$ Entwurfsplan in $\mathcal{DT}op$.

3.2 Die Kategorie $\mathcal{DT}op$

Bei topologischen Datenbanken stellt sich zunächst die Frage, wie stetige Abbildungen zu charakterisieren sind. Es zeigt sich, dass für zwei topologische Datentypen (X, R) und (Y, S) eine Abbildung $f\colon X \rightarrow Y$ genau dann stetig von $(X, \mathfrak{T}_R)$ zu $(Y, \mathfrak{T}_S)$ ist, wenn für alle Paare (a, b) in R das Paar $(f(a), f(b))$ in der transitiven reflexiven Hülle S^* von S enthalten ist. Letztere Eigenschaft definiert die Morphismen in der Kategorie $\mathcal{DT}op$ der topologischen Datentypen, und wir haben nun die Äquivalenz von $\mathcal{DT}op$ mit der Kategorie der so genannten *Alexandrovräume* [4, Theorem 7.6]. Abbildung 9 illustriert einen Morphismus in $\mathcal{DT}op$: dem Pfeil $A \rightarrow h$ im Werkplan entspricht der Weg $A' \rightsquigarrow h'$ im Entwurfsplan, da die Abbildung zwischen den zugehörigen topologischen Räumen f stetig ist.

Es lässt sich nun folgern, dass die Entscheidung der Stetigkeit einer Abbildung im Allgemeinen die Möglichkeit erfordert, zu entscheiden, ob ein Paar (a, b) in der transitiven Hülle R^+ einer Relation R liegt [4, Theorem 8.25]. Dies ist von Bedeutung, da mit den in der Literatur gängigen relationalen Abfragesprachen dies meist nicht der Fall ist [11]. Die Definition der relationalen Algebra in [6, S. 140] hingegen beinhaltet ausdrücklich die Berechnung der transitive Hülle, welche dort als *recursive join* bezeichnet wird. Manche kommerzielle Anbieter haben eine entsprechend erweiterte Abfragesprache. Oracle SQL ermöglicht dies z.B. mit `start with...connect by...nocycle`.

Allgemein konnte gezeigt werden, dass $\mathcal{DT}op$ assymptotisch effizienteste Datenstruktur für endliche topologsiche Räume ist [14, §4.3].

3.3 Von Architektur nach $\mathcal{DT}op$

Definieren wir nun die Kategorie $\mathcal{A}rch^+$, deren Objekte die Paare $(X, \sim)$ sind, wobei X Kompaktum und $\sim$ architektonische Darstellung von X ist, und deren

Morphismen $(X, \sim) \to (X', \sim')$ die stetigen Abbildungen $X \to X'$ sind, für welche die Faser $f^{-1}([x'])$ jeder Äquivalenzklasse $[x']$ in $(X', \sim')$ die Vereinigung von Äquivalenzklassen in $(X, \sim)$ ist. Dann haben wir einen Funktor

$$F^+ \colon \mathcal{A}rch^+ \to \mathcal{DT}op,$$

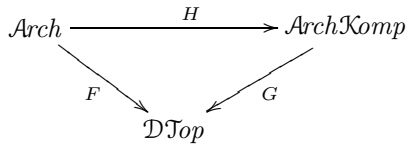
welcher einer architektonischen Darstellung $(X, \sim)$ den zum Quotientenraum $X/\sim$ gehörigen topologischen Datentyp zuordnet. Ebenso sahen wir in Abschnitt 3.1 einen Funktor

$$G \colon \mathcal{A}rch\mathcal{K}omp \to \mathcal{DT}op,$$

welcher einem architektonischen Komplex den zu Grunde liegenden Graph (als topologischen Datentyp) zuordnet.

In $\mathcal{A}rch^+$ gibt es die Teilkategorie $\mathcal{A}rch$ mit denselben Objekten wie bei $\mathcal{A}rch^+$ und deren Morphismen $(X, \sim) \to (X', \sim')$ die zusätzliche Eigenschaft haben, dass n-dimensionale Äquivalenzklassen für $\sim$ in m-dimensionale Klassen für $\sim'$ mit $m \leq n$ abgebildet werden. Bezeichnet nun F die Einschränkung von F^+ auf $\mathcal{A}rch$, so gilt:

Lemma. *Es gibt einen Funktor $H \colon \mathcal{A}rch \to \mathcal{A}rch\mathcal{K}omp$, sodass folgendes Diagramm kommutiert:*



Beweis. Sei $(X, \sim)$ eine architektonische Darstellung. Dann nehmen wir als Menge R_q der q-Regionen die q-dimensionalen Äquivalenzklassen. Ein Paar $(a, b) \in R_q \times R_p$ sei eine Kante, falls b im topologischen Rand von a im Quotientenraum $X/\sim$ liegt. Setzen wir das Kantengewicht Null, so erhalten wir einen architektonischen Komplex. Mit Morphismen wird ähnlich verfahren. □

3.4 Topologische Konstruktionen in $\mathcal{DT}op$

In der Topologie gibt es eine Vielzahl von Konstruktionen, die aus gegebenen topologischen Räumen neue erzeugen. Das zugrundeliegende Prinzip ist stets das selbe: Führe mengentheoretische Operationen mit den Punktmengen gegebener Räume durch und versehe die Ergebnismenge mit einer geeigneten Topologie. Dabei gibt es für diese Operationen gewisse charakteristische Abbildungen. Die Topologie für die Ergebnismenge ist dann so zu wählen, dass diese Abbildungen gerade noch stetig sind. Dabei sind zwei Fälle zu unterscheiden:

Fall 1. Man nehme für eine Indexmenge I topologische Räume $(X_i, \mathcal{T}_i)$ mit $i \in I$, eine Menge Y und für jedes $i \in I$ eine Abbildung $f_i \colon X_i \to Y$ in die Menge Y hinein. Dann wird die größte (d.h. feinste) Topologie $\mathcal{T}_f$ gesucht, sodass alle $f_i \colon (X_i, \mathcal{T}_i) \to (Y, \mathcal{T}_f)$ stetig sind. Da $\mathcal{T}_f$ am Ende der Abbildungspfeile sitzt, heisst diese Topologie *Finaltoplogie*.

Fall 2. Hier gehen die Abbildungen $g_i \colon Y \to X_i$ umgekehrt von der Menge Y aus in die topologischen Räume $(X_i, \mathcal{T}_i)$. Gesucht wird die kleinste (d.h. gröbste) Topologie $\mathcal{T}_g$, so dass alle $g_i \colon (Y, \mathcal{T}_g) \to (X_i, \mathcal{T}_i)$ stetig sind. $\mathcal{T}_g$ sitzt nun am Anfang der Abbildungspfeile und heisst deshalb *Initialtoplogie*.

Beispiele.

1. Sind $(X, \mathcal{T}_X)$ und $(Y, \mathcal{T}_Y)$ topologische Räume, dann ist die Produkttopologie $\mathcal{T}_{X\times Y} := \{A \times B \mid A \in \mathcal{T}_X, B \in \mathcal{T}_Y\}$ für das kartesische Produkt $X \times Y$ die Initialtopologie der Projektionsabbildungen $\pi_X \colon X \times Y \to X,\ (x, y) \mapsto x$ und $\pi_Y \colon X \times Y \to Y,\ (x, y) \mapsto y$. Der topologische Raum $(X, \mathcal{T}_X) \times (Y, \mathcal{T}_Y) := (X \times Y, \mathcal{T}_{X\times Y})$ heisst Produktraum.

2. Ist $(X, \mathcal{T}_X)$ ein topologischer Raum und $A \subseteq X$ Teilmenge von X, dann ist die Spurtopologie $\mathcal{T}|_A := \{A \cap B \mid B \in \mathcal{T}_X\}$ die Initialtopologie der kanonischen Injektion $i : A \to X, a \mapsto a$.

3. Ist $(X, \mathcal{T}_X)$ ein topologischer Raum und $f \colon X \to Y$ beliebige Abbildung, dann ist $\mathcal{T}_f = \{A \subseteq Y \mid f^{-1}(A) \in \mathcal{T}_X\}$ die Finaltopologie von f.

Die obigen Beispiele wurden nicht beliebig gewählt, denn jedes dieser Beispiele hat eine Entsprechung zu den üblichen Operatoren der relationalen Algebra:

1. Für topologische Datenbanken (X, R) und (Y, S) erzeugt der Ausdruck $(\Delta_X \otimes S) \cup (R \otimes \Delta_Y)$ die Produkttopologie von $\mathcal{T}_R$ und $\mathcal{T}_S$ [4, Corollary 8.20], wobei wir das direkte Produkt $A \otimes B$ zweier Relationen $A \subseteq X^2$ und $B \subseteq Y^2$ definieren als $A \otimes B := \{((a, x), (b, y)) \mid ((a, b), (x, y)) \in A \times B\}$ und $\Delta_X \subseteq X^2$ die Diagonale (oder Gleichheitsrelation in X) ist. In relationaler Algebra existiert kein Unterschied zwischen direktem Produkt und kartesischem Produkt.

2. Für eine Selektion $\sigma_P(X)$ und Relation R in X erzeugt $R^+ \cap (\sigma_P(X) \times \sigma_P(X))$ die Spurtopologie von $\mathcal{T}_R$ in $\sigma_P(X)$ [4, Lemma 8.6]. Dabei ist R^+ die transitive Hülle von R. Es gilt also

$$\mathcal{T}_R|_{\sigma_P(X)} = \mathcal{T}_{R^+\cap(\sigma_P(X)\times\sigma_P(X))}.$$

   Damit haben wir eine topologische Variante der Selektion:

$$\sigma_P(X, R) := (\sigma_P(X), R^+ \cap (\sigma_P(X) \times \sigma_P(X)).$$

3. Sei (X, R) eine topologische Datenbank, und ohne Einschränkung sei X eine Relation mit Schlüssel $\{\mathrm{id}\}$ und weiteren Attributen, und R sei Relation mit Schlüssel $\{a, b\}$, wobei a und b auch Fremdschlüssel von $X.\,\mathrm{id}$ seien. Für Projektionen $\pi_{\mathcal{A}}(X)$ einer Relation X auf eine Attributmenge $\mathcal{A}$ ist die Projektion

$$\pi_{X.\mathcal{A},Y.\mathcal{A}}(X \bowtie_{X.\,\mathrm{id}=R.a} R \bowtie_{Y.\,\mathrm{id}=R.b} \beta_Y(X))$$

eine Relation, welche die Finaltopologie für $\pi_{\mathcal{A}}(X)$ erzeugt. Hierbei sei $X.\mathcal{A} := \{X.a \mid a \in \mathcal{A}\}$ die wie in SQL implizit umbenannten Attribute von X und analog $\beta_Y(X)$ die Umbenennung $\beta_{Y.\mathcal{A} \leftarrow \mathcal{A}}(X)$. Die Finaltopologie stimmt mit der Quotiententopologie für die Urbilder von $\pi_{\mathcal{A}}$ überein, ist also die Topologie des `group_by`-Operators in SQL [4, Remark 8.15]. Bei anders lautenden oder mehrstelligen Schlüsseln und Fremdschlüsseln sind die Ausdrücke in entsprechend naheliegender Weise zu ändern.

4. Für die topologische Datenbank (X, R) kann R trivialerweise auch als Relation der Finaltopologie der Umbenennung $\beta_{(A_i \leftarrow B_i)_{i \in I}}(X)$ betrachtet werden.

Weitere Konstruktionen (z.B. Verkleben topologischer Datenbanken) lassen sich mit obigen Mitteln ebenfalls ausführen [4, Corollary 8.19].

Insgesamt konnte gezeigt werden, dass im Allgemeinen zur Berechnung der Initialtopologie die Berechnung der transitive Hülle einer Relation notwendig ist [4, Theorem 8.26]. Dies ist jedoch nur eine theoretische Einschränkung, da die in der Praxis zu modellierenden Räume meist Spezialfälle sind, in denen die transitive Hülle in wenigen Iterationsschritten erreicht wird — ein Iterationsschritt pro Dimension des modellierten Raums. Einige Sonderfälle von Initialtopologien wie z.B. die Produkttopologie oder die Initialtopologie einer einzelen surjektiven Abbildung sind generell auch mit relationaler Algebra ohne transitive Hülle berechenbar.

Damit ist das mengentheoretische topologische Datenmodell im Wesentlichen eine Modifikation des relationalen Datenmodells mit entsprechend angepassten Abfrageoperatoren.

3.5 Entwurf Topologischer Datenbanken

In der Datenbanktheorie ist eine Unterteilung in eine Theorie des Datenmodells als solches und in eine Entwurfslehre üblich. In dieser werden Konsistenzregeln, wie etwa Normalformen oder referentielle Integritäten, behandelt. Analog dazu wurde auch nach topologischen Konsistenzregeln gesucht, die für den Entwurf von topologiebasierten Planungssystemen sinnvoll sind.

Dabei wurde festgestellt, dass „Skizzierungen“ genau die monotonen stetigen Abbildungen im Sinne der Definition von Kuratowski [10] sind. Danach wird eine stetige Abbildung $f : (X, \mathcal{T}) \to (Y, \mathcal{S})$ genau dann als *monoton* bezeichnet, wenn das Urbild $f^{-1}(B)$ einer in $(Y, \mathcal{S})$ zusammenhängenden Menge B zusammenhängend in $(X, \mathcal{T})$ ist. Andere Autoren verwenden eine schwächere Definition von (topologischer) Monotonie und fordern nur, dass das Urbild eines einzelnen Punktes zusammenhängend ist.

Im Projekt wird unter „monoton“ stets diese Monotonie im Sinne von Kuratowski verstanden, denn diese echt stärkere Definition hat folgende vorteilhafte Eigenschaften:

- Die Komposition von monotonen Abbildungen ist ebenfalls eine monotone Abbildung [14, Satz 4.26].

- Monotonie von Abbildungen ist eine sinnvolle Konsistenzregel für Detaildatenbanken [14, §4.5.5].

Wegen der ersten Eigenschaft sind die monotonen Abbildungen Morphismen einer Kategorie der topologischen Räume, denn die identische Abbildung ist natürlich ebenfalls monoton. Die besondere Eignung von Monotonie als Konsistenzregel für Detaildatenbanken ergibt sich aus folgender Beobachtung:

Theorem (Faserprodukt bei monotonen Abbildungen). *Seien E, Y und I alexandroffsche Räume und $p : Y \to I$ eine surjektive monotone Abbildung. Sei zudem die Toplologie von I final bezüglich p, und es sei $d : E \to I$ ebenfalls eine stetige Abbildung. Für das Faserprodukt $E \times_I Y$ bezüglich p und d gilt dann: Die Projektion $\pi_E : E \times_I Y \to E$ ist ebenfalls monoton und surjektiv.*

Das Theorem ist eine äquivalente Umformulierung von [14, Satz 4.28]. Dabei ist das Faserprodukt die topologische Version des relationalen Gleichheitsverbundes (Equi-Join). Es wird das beschrieben, was passiert, wenn bei einer Planung eine Entwurfsskizze ausdetailliert wird. Dabei ist E die Entwurfsskizze, I ist ein Verzeichnis (Index) von Detailbezeichnern als topologischer Raum. Dabei legt die Topologie von I fest, *ob* Details untereinander verbunden werden können. Y ist ein topologischer Raum, der die Ausführung der in I aufgeführten Details (als Fasern bezüglich p) festlegt und zudem bestimmt, *wie* die in I ermöglichten Verbindungen von Details auszuführen sind. Die Abbildung $d : E \to I$ legt fest, welches Detail $d(e)$ ein Entwurfselement $e \in E$ verwendet. Der Verbund $E \bowtie_{d=p} Y$ ist genau das obige Faserprodukt und ergibt den detaillierten Plan von E ebenfalls als topologischen Raum. Die Stetigkeit von d bedeutet, dass zwei miteinander „verbundene“ Entwurfselemente in E nur solche Details in I verwenden dürfen, für die ebenfalls eine Verbindung besteht. Wegen der Monotonie von p sind diese Details dann ebenfalls in Y verbunden. Es kann nun gezeigt werden, dass die Vereinigung der Fasern dieser Entwurfselemente zusammenhängend im Faserprodukt (i.a.W. im Equi-Join) ist. Dies ist eine charakterisierende Eigenschaft der Monotonie surjektiver Abbildungen bei alexandroffschen Räumen und somit auch in den topologischen Datenbanken [14, Satz 4.25].

Man beachte, dass die Detailverwendung $d : E \to I$ eine totale Abbildung ist, dass also stets gefordert wird, dass *jedem* Entwurfselement immer auch ein Detail zugeordnet ist. In der Praxis wird jedoch beim Planen immer zunächst ein skizzenhaftes Konzept erarbeitet und die Detaillierung wird in dieser Planungsphase noch offengelassen. Dann wäre die Detailverwendung jedoch eine partielle Abbildung $f :\subseteq E \to I$ und anstelle des Faserproduktes (bzw. eines inneren Verbundes) wäre die Detaillierung ein äußerer Verbund. Es konnte jedoch gezeigt werden, dass ein äußerer Verbund (outer join) bei den topologischen Datenbanken keine sinnvolle Definition hat, da dessen Topologie im Allgemeinen nicht eindeutig bestimmbar ist. Vielmehr hat ein äußerer Verbund oft mehrere gleichermaßen geeignete Topologien.

Es gibt jedoch stets die Möglichkeit, einen äußeren Verbund zu vermeiden und durch einen praktisch gleichwertigen inneren Verbund zu ersetzen, indem

man Vorgabeobjekte einführt. Dies geschieht, indem in einer Detailbibliothek ein „Nicht“-Detail n eingeführt wird, auf das ein Entwurfsobjekt immer dann verweist, wenn die Detaillierung für das Objekt noch nicht feststeht. Dies entspricht einer Fortsetzung der partiellen Abbildung f zu einer totalen Abbildung $f' : E \to I \cup \{n\}$ und einer Fortsetzung von p zu $p' : Y \cup \{n\} \to I \cup \{n\}$ mit $p'(n) = n$ und für alle y mit $p(y) = \uparrow$ gilt $p'(y) = n$. Damit ist der links-äußere Verbund $E \rtimes_{p=f} Y$ als Menge im Wesentlichen gleich dem inneren Verbund $E \bowtie_{p'=f'} (Y \cup \{n\})$, für den nun zudem eine Topologie eindeutig definiert ist. Diese ist übrigens eine der mehreren möglichen Topologien für $E \rtimes_{p=f} Y$ und ist festgelegt durch die Fortsetzung der Topologie für Y auf $Y \cup \{n\}$.

Weitere Entwurfsregeln wurden nicht untersucht. Die Vorgehensweise bei der Betrachtung von Monotonie kann jedoch als Vorbild für die Entwcicklung weiterer topologischen Entwurfsregeln dienen.

4 Implementierung

Vor der ersten formellen Spezifikation des Prototyps wurde ein erstes experimentelles Java-Programm geschrieben, um mit dem Datenmodell vertraut zu werden. Mit den dabei gemachten Erfahrungen wurde dann begonnen, den endgültigen Prototypen zu programmieren.

4.1 Erster Prototyp

Es wurde zunächst ein Datenbankschema nach *DKetKomp* angelegt und eine entsprechende Klassenstruktur in Java entwickelt.

Die Zuordnung von Randoperator und Zellen geschah zunächst einfach durch eine Benennugskonvention: Ist '`<K>`' der Name des Komplexes, dann heißt die Tabelle mit den Zellen '`<K>_cells`' und der Randoperator ist '`<K>_boundary`'.

Der Komplex hat folgendes Relationenschema:

```
create table <K>_cells(
       cell integer not null,
       dim integer not null,
       primary key(cell)
);
create table <K>_boundary(
       cell1 integer not null,
       cell2 integer not null,
       value integer not null,
       primary key(cell1,cell2),
       foreign key(cell1) references <K>_cells,
       foreign key(cell2) references <K>_cells
);
```

Hier ist die Zelldimension ein Attribut, d.h. sie wird dynamisch behandelt.

Es gibt noch weitere Relationen, beispielsweise `<K>_0D-properties`, die Koordinaten der 0-Zellen im $\mathbb{R}^n$. Für den ersten Prototyp ist dies als Modellierung der geometrischen Einbettung des Komplexes ausreichend, da der Prototyp fürs Erste auf Polytope beschränkt bleiben soll.

Zudem wurde für obiges Datenbankschema die Java-Schnittstelle `Complex` als abstrakter Datentyp definiert, und die Klasse `DBComplex` implementiert diese Schnittstelle. Zur geometrischen Einbettung dient die Schnittstelle `GeoComplex`, implementiert durch `DBGeoComplex`.

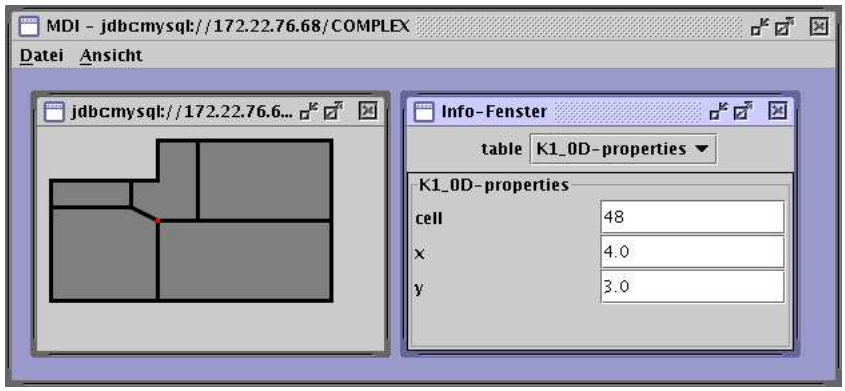


Abbildung 10: Der erste Java-Client (Screenshot).

Die Java-Implementierung folgt der MVC-Architektur[1]:

1. Das Modell ist Klasse `DBGeoComplex` beschrieben. Seine Schnittstelle sind die in `GeoComplex` definierten Operationen.

2. Mit der Programmierung des View-Control-Paars unter Verwendung von Swing-Komponenten wurde begonnen. Es gibt ein Fenster zum Betrachten eines 2D-Komplex und Auswählen einzelner Zellen sowie einen Dialog zum Bearbeiten der zusätzlichen Zellenattribute. Abbildung 10 zeigt ein Screenshot einer Anwendung des Clients.

Die Arbeiten am Prototyp erschienen vielversprechend. Abbildung 11 zeigt das UML-Diagramm der grundlegenden Modellklassen des vorläufigen Prototyps. Die vollständige Dokumentation (javadoc) und Quellcode können von der Projekthomepage heruntergeladen werden.

Allerdings zeigten sich im Verlauf der Arbeit einige Probleme, die eine gründliche Überarbeitung des Konzepts erforderten.

4.2 Erste Probleme

Ein relationaler Komplex besteht im Wesentlichen aus zwei Komponenten: einer Menge von Zellen und einer Matrix, die den Randoperator für diese Zellenmen-

[1] Model-View-Control

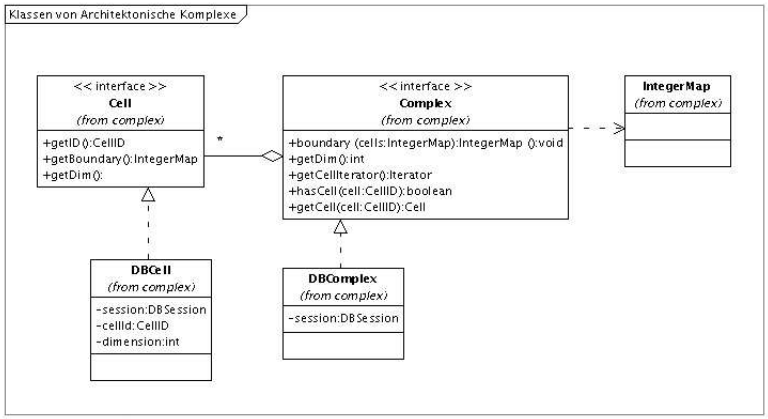


Abbildung 11: UML-Diagramm der grundlegenden Modellklassen.

ge definiert. Alle Methoden des Komplex operieren auf diesen zwei Objekten. Zum Zugriff auf die Zellen wurden zunächst Methoden, die ähnlich denen der Schnittstelle `Set` für Mengen sind, für `Complex` definiert. Zudem wurde eine Erweiterung der Schnittstelle `Map` (jeweils aus `java.util`) für Matrizen definiert. Im Laufe der Zeit stellte sich heraus, dass jede der Implementierungen der Schnittstelle `Complex` stets so modifiziert werden kann, dass ein Komplex auf die Zellenmenge und den Randoperator als Matrix nur als abstrakte Datentypen zugreift und über deren Implementierung nichts wissen muss. Damit sind aber spezialisierte Schnittstellen, wie etwa der im ersten Prototypen definierte `DBComplex` für eine Datenbankrealisierung, unnötig. Vielmehr ist es ausreichend, dem Komplex die Zellenmenge und die Matrix als Parameter zu übergeben, was ermöglicht, sich auf eine einzige Implementierung von `Complex` zu beschränken.

Weitere abgeleitete Klassen betrafen den Zelltyp des Komplex oder sogar die Eingenschaft, ob der Komplex veränderbar ist oder nicht. Insgesamt ist so sehr umfangreicher und schwer handhabbarer Code entstanden, der die weitere Entwicklung stark behinderte. Die Situation wurde übrigens dadurch erschwert, dass im Projektverlauf drei „Generationen“ von wissenschaftlichen Hilfskräften am Code arbeiteten die jeweils neu eingearbeitet werden mussten.

Das Benennungsschema für die Datenbanktabellen wurde im Laufe des Projektes als zu unflexibel gesehen.

4.3 Der endgültige Ansatz

Zur Vermeidung der oben genannten Probleme wurde zur Programmierung ein neuer Ansatz gewählt: Es gibt nur noch einen abstrakten Typ `Complex`. Der Zelltyp wird dabei als Parameter zu übergeben, was durch die in Java relativ neue Technik der parametrisierten Klassen (Generics) möglich ist. Die Zellen und der Randoperator werden dem Komplex als entsprechend typparametrisierte `Set`

und `Matrix` Objekte zugeordnet.

Dies führte zu einem neuen, einleuchtenderem Konzept der Definition einer typparametrisierten Schnittstelle `Complex<T extends Cell>` mit Zellenmenge `Set<T> cellSet` und Randoperator `Matrix<T,T,Integer> boundary`. Dieses Konzept wurde auf der eCAADe 2007 in Frankfurt vorgestellt [12].

Es folgte die sehr zeitaufwändige Refaktorierung des gesamten Projekts auf das neue Konzept. Insbesondere muss zugegeben werden, dass der angestrebte Prototyp nicht fertiggestellt werden konnte. Er enthält jedoch einige interessante Bestandteile, die nun ausführlich erläutert werden.

Wenn im Folgenden von „Zellen“ die Rede ist, dann sind damit ganz allgemein die Basiselemente $b \in B$ von relationalen Kettenkomplexen (M, B, Δ) gemeint.

4.3.1 Spezifikation

Es wurde im weiteren Projektverlauf formeller gearbeitet und Spezifikationsdokumente angefertigt, um den weiteren Projektverlauf zu steuern. Dazu wurde ein Pflichtenheft [13] und ein Dokument zur Sammlung von Umsetzungsideen angefertigt. Für die Programmierung wurde Wert auf den Einsatz von JUnit-Tests gelegt. Eine umfassende Niederschrift der Spezifikation ist in Arbeit [15].

4.3.2 Dynamische und statische Dimension

Es gibt prinzipiell zwei Arten der Realisierung eines relationalen Komplexes: zum Einen mit *dynamischer* und zum Anderen mit *statischer* Dimension.

Im ersten Fall ist die Dimension der Zellen eines relationalen Kettenkomplexes einfach ein Attribut. Dies erlaubt die Auflistung aller Zellen in einer einzigen Tabelle, und die Dimension des Komplexes ist nicht nach oben beschränkt. Die Tabelle für den partiellen Randoperator δ mit partieller Matrix $\Delta :\subseteq B \times B \to \mathbb{Z}$ gibt den Matrixeintrag für Paare von Zellen beliebiger Dimensionen an, soweit sie von $\uparrow$ verschieden sind.

Im zweiten Fall gibt es für jede Zelldimension eine eigene Zelltabelle. Dies erlaubt es, in jeder Dimension unterschiedliche Attribute ohne `null`-Werte einzuführen. Die erlaubte Komplexdimension übersteigt jedoch nicht die höchste Dimension, zu der eine Zelltabelle existiert. Hier ist es oft zweckmäßig, die partielle Matrix Δ in die Bestandteile $\Delta_{mn} :\subseteq B_m \times B_n \to \mathbb{Z}$ zu zerlegen. Dem entspricht jeweils eine eigene Tabelle für jedes auftretende Paar von Zelldimensionen. Für viele Zwecke genügt es jedoch, Tabellen lediglich für die partiellen Teilmatrizen $\Delta_{m,m-1} :\subseteq B_m \times B_{m-1} \to \mathbb{Z}$ anzulegen.

4.3.3 Komplexkatalog

Die Zuordnung der Zellenmengen zu den Randmatrizen durch eine Namenskonvention wurde ersetzt durch einen Katalog zur Auflistung der Komplexe. Ein erster Entwurf in ER-Notation wurde verworfen, denn die Problemstellung ist schwer in der Form von „Entitäten“, die zueinander in „Beziehungen“ stehen

fassbar. Daher wurde eine andere Technik zum relationalen Entwurf des Katalogs bevorzugt:

Zunächst bestimmt man alle notwendigen Attribute zu Speicherung der benötigten Information und legt dann die funktionalen Abhängigkeiten in dieser Attributmenge fest. Es gibt einen bekannten Synthesealgorithmus [9, S. 180], der anhand der funktionalen Abhängigkeiten eine Überdeckung dieser Attributmenge durch kleinere Attributmengen bestimmt[2]. Jede dieser Attributmengen ergibt dann ein Relationenschema und die Überdeckung insgesamt ein verlustfreies Datenbankschema in dritter Normalform.

Folgende Attribute wurden für einen Komplexkatalog als notwendig erachtet:

`ComplexID`. Eine eindeutig Referenznummer für einen Komplex.

`ComplexName`. Der (ebenfalls eindeutige) Name des Komplex.

`ComplexIsStaticDim`. Ein boolescher Wert, der angibt, ob der Komplex mit „statischer“ Dimension ist, jeweils also eine eigene Zellentabelle pro Dimension hat, oder ob es sich um einen Komplex mit „dynamischer“ Dimension handelt, in dem alle Zellen beliebiger Dimension in einer einzigen Tabelle zusammen mit einem Dimensionsattribut gespeichert werden.

`ComplexCellsTable`. Der Name der Tabelle, die die Zellen (ggf. einer bestimmten Dimension) enthält. Bei statischer Dimension kann es mehrere derartiger Zellen je Dimension geben.

`ComplexCellKeyAttribute`. Ein Attributname eines Primärschlüsselattributs einer `ComplexCellsTable`.

`ComplexCellsDimension`. Die Dimension der Zellen einer `ComplexCellsTable`. Der Wert ist nur bei statischer Dimension von Bedeutung.

`ComplexCellsDimAttribute`. Das Dimensionsattribut, welches sich in einer Tabelle `ComplexCellsTable` befindet. Der Wert ist nur bei dynamischer Dimension von Bedeutung.

`BoundaryTable`. Der Name einer Tabelle des Randoperators.

`BoundaryRowKeyAttribute`. Der Name eines Attributes, der eine Zeile in der Matrix des Randoperators definiert.

`BoundaryColKeyAttribute`. Der Name eines Attributes, der eine Spalte in der Matrix des Randoperators definiert.

`BoundaryCoefficientAttribute`. Der Name des Attributes, der den Eintrag in der Matrix des Randoperators definiert.

[2] Dies wird in der Literatur häufig als „Zerlegung“ bezeichnet, ist aber keine Zerlegung im mathematischen Sinn.

Bestimmt man nun die funktionalen Abhängigkeiten (functional dependencies, Fd) jeweils für den Fall eines Komplexes mit statischer und eines Komplexes mit dynamischer Dimension, dann ergeben sich für die Variante mit statischer Dimension folgende funktionale Abhängigkeiten (sog. Fd-Menge):

{`ComplexID`} → {`ComplexName`, `ComplexIsStaticDim`}: Dies sind Eigenschaften des Komplex an sich.

{`ComplexName`} → {`ComplexID`}: Der Komplexname ist eindeutig.

{`ComplexID`, `ComplexCellsDimension`} → {`ComplexCellsTable`}: Der Komplex hat für jede Dimension eine eigene Zellentabelle.

{`ComplexID`, `ComplexCellsDimension`} →
{`BoundaryTable`, `BoundaryCoefficientAttribute`}: Der Komplex hat für jede Dimension einen eigenen Randoperator als Tabelle und diese ein Attribut, unter dem der Matrixeintrag gespeichert ist. Dies ist eine Vereinfachung, die z.B. bei den sog. Tensorproduktkomplexen nicht gegeben ist. Dort ist der Matrixeintrag aus mehreren Attributwerten zu berechnen. Es wird jedoch darauf verzichtet derartige Komplexe explizit abzuspeichern.

{`ComplexID`, `ComplexCellsDimension`, `ComplexCellKeyAttribute`} →
{`BoundaryRowKeyAttribute`, `BoundaryColKeyAttribute`}: Jedes Primärschlüsselattribut einer Zellentabelle ist jeweils Teil eines Spaltenindex und eines Zeilenindex der entsprechenden Matrix.

Bei dynamischer Dimension entfällt das Attribut `ComplexCellsDimension` und es wird das Attribut `ComplexCellsDimAttribute` verwendet. Dies ergibt folgende Fd-Menge:

{`ComplexID`} → {`ComplexName`, `ComplexIsStaticDim`}

{`ComplexName`} → {`ComplexID`}

{`ComplexID`} → {`ComplexCellsTable`}

{`ComplexID`} →
{`BoundaryTable`, `BoundaryCoefficientAttribute`}

{`ComplexID`, `ComplexCellKeyAttribute`} →
{`BoundaryRowKeyAttribute`, `BoundaryColKeyAttribute`}

{`ComplexID`} → {`ComplexCellsDimAttribute`}

Um für die Varianten dynamische und statische Dimension einen gemeinsamen Katalog anlegen zu können, wird das Attribut `ComplexCellsDimAttribute` für alle Komplexe vorgesehen und kann im Falle dynamischer Dimension den Wert `null` haben. Umgekehrt wird ein Komplex mit dynamischer Dimension als Komplex mit statischer Dimension modelliert, dessen `ComplexCellsDimension` stets den konstanten Wert 0 annimmt. Dies ergibt eine gemeinsame Fd-Menge für beide Varianten der Modellierung der Dimension. Diese Fd-Menge gleicht der für statische Dimension, erweitert um die zusätzliche funktionale Abhängigkeit

{ComplexID, ComplexCellsDimension} → {ComplexCellsDimAttribute}

Wendet man nun den genannten Synthesealgorithmus [9, S. 180] auf diese Fd-Menge an, ergibt sich ein verlustloses Fd-treues Datenbankschema in dritter Normalform. In unserem Falle entsteht dabei folgendes Schema:

```
Complexes (ID, name, isStaticDimension)

ComplexTables (complex, cellDimension, cellTable,
     cellDimensionAttribute, boundaryTable,
     boundaryCoefficientAttibute)

ComplexBoundary (complex, cellDimension, cellKeyAttribute,
     rowKeyAttribute, colKeyAttribute
```

Die vom Synthesealgorithmus erzeugte Relation `ComplexTables` wurde aufgeteilt in `ComplexCellTables` und `ComplexBoundaryTables`, um die Sequenz der Randoperatoren über Fremdschlüsselbeziehungen definieren zu können:

```
ComplexCellTables (complex, cellDimension, cellTable,
     cellDimensionAttribute)

ComplexBoundaryTables (complex, cellDimension, boundaryTable,
     boundaryCoefficientAttibute, nextCellDimension)
```

Wäre nun `nextCellDimension` stets um eins kleiner als `cellDimension`, dann wäre dies ein Verstoß gegen die zweite Normalform. Ein derartiger Verstoß liegt aber nicht vor, denn diese funktionale Abhängigkeit zwischen `cellDimension` und `nextCellDimension` gibt es höchstens im Falle statischer Dimension. Bei dynamischer Dimension wird hingegen vereinbart, dass `cellDimension` konstant 0 ist. Also ist `nextCellDimension` vom Primärschlüssel voll funktional abhängig, und an dieser Stelle wird nicht gegen NF2 verstoßen.

In Java ist eine ensprechende Klasse `DBComplexCatalog` definiert. In dieser sind die oben erwähnten Tabellen und Attributbezeichner des Katalogs als Konstanten deklariert. Der Konstruktor eines `DBComplexCatalog` erhält als Parameter ein `java.sql.Connection` Objekt und erstellt daraus ein Verzeichnis der Komplexe. Eine Erweiterung des Katalogs um Geometrieinformationen wurde ebenfalls begonnen aber nicht mehr fertig gestellt.

4.3.4 Hilfsklassen: Mengen, Ketten und Matrizen

Das relationale Datenmodell baut auf der Mengenlehre auf. Daher lag es nahe, die Schnittstellen `Set` und `Map` aus der Java-Collections Bibliothek zu verwenden, oder sich zumindest daran zu orientieren, wobei Tabellen wahlweise als Implementierungen von `Set` oder als `Map` betrachtet wurden. Diese Herangehensweise bereitete etwas Schwierigkeiten, da sich das Iterieren von Tabellen nach dem JDBC-Standard mit dem Iterieren von `Set`-Objekten nicht problemlos in Einklang bringen lässt. Der im Antrag versprochene „formale Weg“ von der Theorie zur Software hatte hier ein überraschendes Hindernis.

Die Klassen für die im Projekt entwickelte besondere Variante der linearen Algebra mit `null`-Werten wurden in einem Paket `linAlg` zusammengefasst. Eine ganzzahlige Linearkombination (Kette) von Objekten einer Klasse `T` wurde dort als `IntegerMap<T>` realisiert und $M \times N$-Matrizen waren Objekte vom Typ `IntegerMap<Pair<M,N>>`. Matrizen sind also nichts anderes als Linearkombinationen von Paaren bzw. von Instanzen einer sehr einfachen Klasse `Pair<X,Y>`. Eine `IntegerMap<T>` ist eine Schnittstelle, welche `Map<T,Integer>` um die Verknüpfungen der linearen Algebra erweitert. Auch für Matrizen wurde eine eigene Schnittstelle `Matrix<M,N>` definiert. Dabei wird stets auf eine konsistente Behandlung von `null`-Werten und deren Unterschied zum Wert 0 geachtet. Dies geschieht durch eine Utility-Klasse `Integers` mit statischen Methoden `plus(Integer,Integer)` zur Addition und `mul(Integer,Integer)` zur Multiplikation und vielen weiteren statischen Methoden (`minus`, `abs`, `signum`, etc.), die von der `Complex`-Schnittstelle benutzt werden. Es wurde durchgehend mit der `Integer`-Klasse gearbeitet und nicht mit dem primitiven Typ `int`, da es bei `int` keine `null`-Werte gibt.

Zur Modellierung der Komplexe wurden anhand der Einträge des Katalogs jeweils ein `Set`-Objekt für eine Zellentabelle und ein `Matrix`-Objekt für den jeweiligen Randoperator definiert.

4.3.5 Modellklassen: Komplexe

Eine Zelle ist einfach eine Instanz der Schnittstelle `Cell`, welche vermittels eines `id`-Attributs zur Identifizierung (als Methode `getId()`) die Schnittstelle `Comparable` erweitert. Durch die so definierte lineare Ordnung für `Cell`-Objekte, ist es möglich, die effizienten Bibiliotheksklassen für geordnete Mengen (z.B. `TreeSet`) zu verwenden.

Ein Komplex ist Instanz der Schnittstelle `Complex<Z extends Cell>` mit den beiden Elementen `cellSet()` vom Typ `Set<Z>` und `getBoundary()` vom Typ `Matrix<Z,Z>`. Die Methode `boundary(IntegerMap<Z> c)` gibt den Rand der Linearkombination `c` von Zellen als Linearkombination zurück und ist äquivalent zu `getBoundary().multiply(c)`.

Ein editierbarer Komplex ist an der Eigenschaft `isModifyable()` zu erkennen. Für diese sind die optionalen Methoden `addCell(Z, IntegerMap<Z>)`, `makeCell()` und `removeCell(Z)` definiert. Bei nicht editierbaren Komplexen erzeugen diese Methoden eine `UnsupportedOperationException`. Dabei entsprechen `addCell` und `removeCell` den bekannten Euleroperatoren der Volumenmodellierung: `addCell(z,dz)` heftet die Zelle `z` an den angegebenen Rand `dz`. Dazu muss `dz` in der Tat ein Rand sein, das heißt, der Rand von `dz`, also `boundary(dz)`, darf nur 0 oder `null`-Werte enthalten. Andernfalls erzeugt die Methode eine `IllegalArgumentException`. Diese Methode entspricht den Euleroperatoren MEJC (Make Edge Join Components), MEML (Make Edge Make Loop) oder MVMC (Make Vertex Make Component), bzw. deren höherdimensionalen Äquivalente. Umgekehrt darf `removeCell(z)` nur für eine Zelle `z` aufgerufen werden, die nicht selbst Rand ist und entspricht somit den entsprechend inversen Euleroperatoren (Kill Edge Split Component, Kill Edge Kill

Loop oder Kill Vertex Kill Component usw.). Es zeigt sich zudem, dass es bis auf Dimensionsverschiebung im Wesentlichen nur zwei verschiedene Euleroperatoren gibt [2, §6].

Bei der Implementierung eines Komplexes müssen nur die Zellenmenge und die Matrix des Randoperators als Parameter übergeben werden. Insbesondere sind dies diejenigen Zellenmengen und Matrizen, wie sie durch die Einträge im Komplexkatalog definiert sind. Zudem muss ein editierbarer Komplex die Methode `makeCell()` implementieren. Dazu existiert auch eine Schnittstelle `CellFactory<Z extends Cell>`, mit nur dieser einen Methode, die als Zellenfabrik verwendet werden kann [7]. Ein Beispiel für eine derartige Herangehensweise ist die Klasse `ComplexImpl<Z extends Cell>`.

4.3.6 Weitere Funktionen

Es folgt eine Beschreibung weiterer Funktionen der im Projekt entwickelten Schnittstelle `Complex<Z extends Cell>`.

`boolean addCells(Set<Z> cells, Matrix<Z,Z> chBd).` Addiert den durch die Parameter angegebenen Komplex zum aufgerufenen Komplex. Insbesonder wird eine `IllegalArgumentException` ausgelöst, falls `chBd` kein Rand ist.

`IntegerMap<Z> coBoundary(IntegerMap<Z> chain).` Der Korand der angegebenen Kette, i.a.W. der Rand dieser Kette im Kokomplex. Dies ist der durch die transponierte Matrix definierte Komplex.

`Set<Z> getStar(Set<Z> s).` Der (topologische) Stern um die Menge `s`, i.a.W. die kleinste offene Menge `sts`, so dass gilt `sts.conatinsAll(s)`.

`Set<Z> getClosure(Set<Z> s).` Die (topologische) Hülle um die Menge `s`, i.a.W. die kleinste abgeschlossene Menge `cls`, so dass `cls.conatinsAll(s)` gültig ist.

`IntegerMap<Z> makeBoundary(Set<Z> s).` Erzeugt eine (von mehreren möglichen) nichttriviale(n) Linearkombination(en) von Zellen aus der angegebenen Menge, dessen Rand eine partielle Nullkette ist. Dies geschieht durch Lösen des durch die Randmatrix definierten homogenen linearen Gleichungssystems mit den Elementen in `s` als Variablen. Ist die Dimension des Lösungsraums größer als 1, enthält s also mehrere unabhängige Zykeln, dann wird eine Ausnahme ausgelöst.

4.3.7 Geometrie

Für die geometrische Realisierung von Komplexen verblieb dem Projekt nicht mehr viel Zeit. Zunächst wurde beabsichtigt, Java3D einzusetzen. Da aber die im Projekt eingesetzte Hardware nicht den hohen Anforderungen von Java3D

genügte, wurde auf dessen hohen Programmierkomfort verzichtet. Der maßgebliche Grund für diese Entscheidung war, dass die Verwendung von Java3D offensichtlich zu einer starken Abhängigkeit von der verwendeten Plattform führt. Die aktuelleren Versionen von Java3D sind diesbezüglich gutmütiger, so dass diese Entscheidung heute vermutlich anders gefallen wäre.

Der Versuch einer Eigenentwicklung der benötigten Funktionalitäten war natürlich ausgeschlossen, da es nicht Ziel des Projekts war, den Stand der Technik zu reproduzieren. Daher wurde auf Funktionen, wie etwa das Berechnen von verdeckten Elementen, verzichtet. Einige Funktionen aus dem `javax.vectmath`-Paket von Java3D konnten dennoch eingesetzt werden. Die Darstellung am Bildschirm erfolgt hingegen konventionell auf Komponenten des Abstract Windowing Toolkit (AWT) von Java.

Allgemeines Geometriekonzept. Komplexe werden auch bei der Modellierung von Volumenkörpern (solid modeling) verwendet. Dabei ist beim BRep-Verfahren die Geometrie eines solchen „Solid“ stets durch die Festlegung dessen Randes bestimmt. Es handelt sich dabei um ein Polytop mit ebenen Randfacetten, die damit jeweils in einer Hyperebene des $\mathbb{R}^3$ liegen und deren Geometrie in dieser Hyperebene durch ihren Rand festgelegt ist.

Der Rand wiederum besteht aus geraden Kanten jeweils in eindimensionalen Unterräumen der Hypereben — also in Hyperebenen von Hyperebenen. Es gibt daher zu jedem Eckpunkt p mehere „Türme“ von Hyperebenen

$$\{p\} \leq H_l \leq H_f \leq H_v = \mathbb{R}^3,$$

sogenannte *Fahnen*.

Dabei wird für die Eckpunkte die Lage im $\mathbb{R}^3$ explizit festgelegt, und die Geometrie der anderen n-Zellen ergibt sich aus der Topologie des Komplexes. Dieser Ansatz wurde konsequent verfolgt: Eine Klasse `AffineSpace` legt ein m-dimensionales Koordinatensystem in einem n-dimensionalen affinen Raum mit $n \geq m$ fest. In diesem können sich weitere affine Unterräume befinden. Davon abgeleitet ist die Klasse `Hyperplane`, ein $n-1$-dimensionaler affiner Unterraum eines n-dimensionalen `AffineSpace` Objekts. Auf eine Festlgung der Dimension wurde bei der Definiton dieser Klassen verzichtet. Da dieser Teil des Projekts nicht fertiggestellt wurde ist nicht sicher, ob der Ansatz zu aufwändig ist.

Für die lineare Algebra wurde das Open-Source-Projekt `JLinAlg` verwendet, das keine Festlegungen bezüglich der Dimension macht. Hier zeigte sich übrigens ein interessanter Fehler in dieser Bibliothek: Es kam bei diesem allgemeinen Ansatz zu der Situation, dass ein lineares Geichungssystem vom Typ „0 Gleichungen über n Variablen“ gelöst werden musste — dieses hat einen n-dimensionalen Lösungsraum. Der Gleichungslöser von `JLinAlg` erzeugt in dieser Situation eine Ausnahme: Die übergebene Matrix m wird nämlich als Fabrik für das Nullelement des entsprechenden Körpers „missbraucht“, indem der `LinSysSolver` das `FieldElement` $m_{1,1}$ auswählt und anhand dessen das Nullelement ermittelt. Der Index $1,1$ liegt aber außerhalb einer $0 \times n$-Matrix und führt dann zu einer `ArrayIndexOutOfBoundsException`. Daraus ließe sich die

Lehre ziehen, dass es zur objektorientierten Programmierung von algebraischen Strukturen meist sinnvoller ist, die Verknüpfungen nicht auf den Klassen der zu verknüpfenden Elemente selbst zu definieren, sondern mit zwei kooperierenden Klassen zu arbeiten: eine Klasse definiert die algebraischen Verknüpfungen (und Fabrikmethoden) und eine weitere Klasse steht für die verknüpften Elemente, etwa eine Klasse `Field<T>` für einen Körper vom Typ `T` mit Methoden, wie `T add(T a, T b)`, und Fabrikmethoden `T zero()` und `T one()` zur Erzeugung des Null- und Einselements. Dies gilt insbesondere, seit auch in Java mit Typparametern gearbeitet werden kann.

Für Zellen wurde im Projekt die Schnittstellenerweiterung `GeoCell` mit einer Methode `getGeometry()` verwirklicht. Diese Geometrie ist rekursiv definiert durch die Geometrien der Randzellen, bzw., im Falle von 0-Zellen, durch explizite Angabe der Lage im Raum. Diese werden durch eine AWT und eine Java3D-Transformationsmatrix auf die Ebene projiziert und als AWT-Shape dargestellt (mit `GeoCell.toShape(Transform3D t3d)`). Durch Angabe von zwei Matrizen erreicht man eine unabhängige Darstellung der Lage des Komplexes relativ zum Betrachter (3D) und der Transformation des `Canvas` (2D) für Zoom und Pan-Operationen.

Räumlicher Index. Einige Zeit wurde im Projekt das Problem von räumlichen Indices verfolgt. Da beabsichtigt war, die Grundlagen für ein Planungssystem zu erarbeiten, erschien es auch wichtig, effizient auf räumliche Daten zugreifen zu können. Zu diesem Zweck sind geeignete Indizierungsverfahren wichtig. Insbesondere sollen diese auch dann noch funktionieren, wenn sich beim Bearbeiten der Pläne die räumliche Situation ändert. Ein Index muss sich also entsprechend anpassen. Hier wurden jedoch keine neuen Erkenntnisse gewonnen, die über den Stand der Technik hinausweisen. Eine Idee wurde dabei vertieft: Ein n-dimensionaler Raum wird durch das direkte Produkt von n AVL-Bäumen indiziert, je ein Baum pro Dimension. Die Sortierordnung ist dabei jeweils durch die (lineare) lexikographische Ordnung der Koordinaten festgelegt, wobei für jeden Baum die Koordinaten jeweils rotiert werden. Ein Punkt $p = (x, y, z)$ wird also in den x-Baum anhand des Wortes xyz, in den y-Baum mit dem Wort yzx und in den z-Baum mit dem Wort zxy einsortiert. Das Suchen eines Punktes geschieht dann durch *paralleles* Absuchen des Baumes, analog zu einem Octree. Jeder Knoten in dem Produktbaum enspricht einem Tupel (n_x, n_y, n_z) aus je einem Knoten des entsprechenden Oktalbaums. Das Paket `indexes` enthält ein Programm `GUITest`, in dem dies für den zweidimensionalen Fall (also ein Quadtree) ausprobiert werden kann: Mit der Maus können Punkte erzeugt werden. Der Wurzelpunkt des x-Baums ist rot, der des y-Baums grün eingefärbt und farbige Linien zeigen den jeweiligen Baum an. Wird ein Punkt „gefunden“, dann wird dies durch gelbe Einfärbung angezeigt. Anspruchsvollerer Suchfunktionen, wie etwa das Absuchen oder Verschneiden von Bereichen wurden nicht mehr angegangen.

A Etwas partielle lineare Algebra

Wir stellen im Folgenden einige Begriffe der linearen Algebra vor, die mit partiellen linearen Abbildungen arbeiten.

Eine Abbildung $f\colon M \to N$ zwischen $\mathbb{Z}$-Moduln heißt linear, wenn für alle $x, y \in M$ und für alle $a \in \mathbb{Z}$ gilt:

$$f(x+y) = f(x) + f(y)$$

Sind M und N freie $\mathbb{Z}$-Moduln, dann existiert jeweils eine Basis A für M und eine Basis B für N. Und M ist isomorph zu $\mathbb{Z}^A$, die Menge aller Abbildungen $m : A \to \mathbb{Z}$ und N ist entsprechend isomorph zu $\mathbb{Z}^B$. Für eine solche Abbildung $m : A \to \mathbb{Z}$ ist der Wert $m(a)$ die a-Koordinate von m bezüglich A. Ist die Basis das kartesische Produkt zweier Mengen $R \times C$, dann heißt $\mathbb{Z}^{R\times C}$ auch *Matrix* und die Menge R heißt dann *Zeilenindex* und C heißt *Spaltenindex* von $\mathbb{Z}^{R\times C}$.

Eine solche Abbildung m kann als Relation mit dem Schema $\mathcal{M}(\underline{a} : A, c : \mathbb{Z})$ gespeichert werden: dies ist eine Tabelle mit einer Schlüsselspalte a für die Koordinatenindizes der Menge A und eine Spalte c für die entsprechenden Koordinatenwerte. Jeder Datensatz hat die Form $\langle i{:}a, m(i){:}c\rangle$.

Nun ist es möglich die Verknüpfungen der linearen Algebra mittels Datenbankabfragen für derartige Tabellen durchzuführen. Die Summe zweier Elemente x und y, abgespeichert als Tebellen in einer relationalen Datenbank kann mit SQL wie folgt berechnet werden:

```
select a, sum(k) as k
from   mPlusn
group by a;
```

wobei mPlusn durch eine Abfrage

```
create view mPlusn as
select 0 as ref, a, k from m
union all
select 1 as ref, a, k from n;
```

entsteht.

Auch für das Matrixprodukt gibt es derartige Abfragen:

```
create view MprodN as
select M.r as r, N.c as c, sum(M.k * N.k) as k
from   M inner join N on (M.c = N.r)
group by M.r, N.c;
```

Damit ist auch die Multiplikation einer Matrix mit einem Vektor möglich.

Will man nun einen solchen Vektor m als Abbildung $m :\subseteq A \to \mathbb{Z}$ platzsparend abspeichern, dann kann man in der entsprechenden Tabelle auch Einträge, deren Koordinatenwert 0 ist, weglassen. Damit erhält man eine *partielle* Abbildung $m' :\subseteq A \to \mathbb{Z}$. Die Menge aller partiellen Abbildungen bezeichnen wir hier mit $\mathbb{Z}^A_\subseteq$, den *partiellen freien $\mathbb{Z}$-Modul* mit Basis A.

Wenn wir nun die obigen Datenbankabfragen auf derartige partielle Vektoren und Matrizen anwenden, dann sind unsere Ergebnisvektoren und Matrizen höchstens an den Stellen nicht definiert, die ansonsten den Wert 0 hätten. Damit erhalten wir auch die partiellen linearen Abbildungen. Das sind die Abbildungen $f\colon M_\subseteq \to N_\subseteq$ zwischen *partiellen* $\mathbb{Z}$-Moduln bei denen für alle $x, y \in M_\subseteq$ gilt:

$$f(x+y) = f(x) + f(y)$$

Man beachte, dass f selbst stets eine totale Abbildung ist.

Werden also in der linearen Algebra partielle Abbildungen verwendet, so muss mit dem undefinierten Wert $\uparrow$ algebraisch gerechnet werden. Aus den Datenbankabfragen ergeben sich nun folgende Rechenregeln mit den Koordinatenwerten:

$$\uparrow \notin \mathbb{Z} \tag{4}$$

$$\uparrow + x = x + \uparrow = x \quad \text{für } x \in \mathbb{Z} \cup \{\uparrow\}, \tag{5}$$

$$\uparrow \cdot \alpha = \alpha \cdot \uparrow = \uparrow \quad \text{für } \alpha \in \mathbb{Z}. \tag{6}$$

Das *Bild* einer partiellen linearen Abbildung $f\colon M_\subseteq \to N_\subseteq$ wird definiert als

$$\operatorname{Bild} f := f(M_\subseteq).$$

Der *Kern* von f ist $\ker(f) := \{x \in M_\subseteq \mid \operatorname{im}_B f(x) \subseteq \{0\}\}$.

Hierbei ist das *Bild* $\operatorname{im}_B v$ eines partiellen Vektors $v :\subseteq B \to \mathbb{Z}$ bezüglich einer Basis B einfach die Menge der Basiskoordinaten, die in v vorkommen:

$$\operatorname{im}_B v = v(B) \subseteq \mathbb{Z}.$$

Es sei darauf hingewiesen, dass das Bild $\operatorname{im}_B f(x)$ eines partiellen Vektors $f(x)$ niemals einen Wert $\uparrow$ als Element enthält. Ist der partielle Vektor $f(x) :\subseteq B \to \mathbb{Z}$ an einer Stelle $b \in B$ nicht definiert, gilt also $f(x)(b) = \uparrow$, dann wird b im Bild $\operatorname{im}_B f(x)$ nicht aufgeführt.

Wir erinnern uns, dass $x \in M_\subseteq$ und $f(x)$ selbst wiederum partielle Abbildungen sind, die einigen Basiselementen jeweils einen Koordinatenwert zuordnen. Damit ist $\operatorname{im}_B f(x)$ die Menge aller Koordinatenwerte (bezüglich einer vorher fest gewählten Basis), die im Vektor $f(x)$ vorkommen.

Bemerkung A.1. *Die Multiplikation partieller Matrizen $M :\subseteq A \times B \to \mathbb{Z}$ und $N :\subseteq B \times C \to \mathbb{Z}$ ergibt sich aus den Rechenregeln (5) und (6).*

Wir definieren nun ganz allgemein, dass ein *partieller $\mathbb{Z}$-Modul* ein Tripel $\mathscr{P} = (P, P_\subseteq, +)$ ist mit den Eigenschaften:

1. $(P, +)$ ist ein $\mathbb{Z}$-Modul.
2. $(P_\subseteq, +)$ ist ein Monoid mit 0 als neutralem Element.
3. $(P, +)$ ist Untermonoid von $(P_\subseteq, +)$.

4. Es gilt für $x \in P_\subseteq$, dass $(-1) \cdot x$ ein wohldefiniertes Element von $P_\subseteq$ ist mit $(-1) \cdot x = -x$, falls $x \in P$. Entsprechend wird für natürliche $n \geq 1$ erklärt:
$$(-n) \cdot x := \underbrace{(-1) \cdot x + \cdots + (-1) \cdot x}_{n \text{ mal}} \in P_\subseteq$$

Bemerkung A.2. *Die vierte Eigenschaft besagt nicht, dass in $P_\subseteq$ notwendig*
$$x + (-1) \cdot x = 0$$
gelten muss. In P gilt dies jedoch.

Beispiel A.3. *Jeder $\mathbb{Z}$-Modul definiert in trivialer Weise einen partiellen $\mathbb{Z}$-Modul. Zur Abgrenzung sprechen wir in diesem Fall auch von* totalen *$\mathbb{Z}$-Moduln.*

Beispiel A.4. *Das Tripel $(\mathbb{Z}, \mathbb{Z}_\subseteq, +)$ mit dem freien partiellen $\mathbb{Z}$-Modul $\mathbb{Z}_\subseteq = \mathbb{Z} \cup \{\uparrow\}$ ist ein partieller Modul.*

Beispiel A.5. *Allgemein ist für jeden freien Modul $\mathbb{Z}^A$ das Tripel $(\mathbb{Z}^A, \mathbb{Z}^A_\subseteq, +)$ ein partieller $\mathbb{Z}$-Modul.*

Sei nun $\mathscr{Q} = (Q, Q_\subseteq, +)$ ein weiterer partieller $\mathbb{Z}$-Modul, für welchen Q eine Untergruppe von P, $Q_\subseteq$ ein Untermonoid von $P_\subseteq$ ist und für jedes $x \in Q_\subseteq$ $(-1) \cdot x$ für beide partielle $\mathbb{Z}$-Moduln denselben Wert liefert. Dann nennen wir $\mathscr{Q}$ einen *partiellen Teilmodul* von $\mathscr{P}$. In diesem Fall definiert für $x, y \in P_\subseteq$
$$x \equiv y \mod Q_\subseteq \Leftrightarrow x + (-1) \cdot y \in Q_\subseteq$$
eine Äquivalenzrelation auf $P_\subseteq$. Der Quotientenraum bezüglich dieser Relation werde mit $P_\subseteq / Q_\subseteq$ bezeichnet und dessen Elemente mit $[x]_\subseteq$ für $x \in P_\subseteq$.

Lemma A.6. *Es ist $\mathscr{P}/\mathscr{Q} := (P/Q, P_\subseteq/Q_\subseteq, +)$ ein partieller $\mathbb{Z}$-Modul, wobei $+$ vertreterweise erklärt ist:*
$$[x]_\subseteq + [y]_\subseteq := [x + y]_\subseteq$$

Beweis. Die Wohldefiniertheit von $+$ auf $P_\subseteq/Q_\subseteq$ ist leicht zu verifizieren, und damit ist $P_\subseteq/Q_\subseteq$ ein Monoid mit neutralem Element $[0]_\subseteq$. Zudem ist $(-1) \cdot [x]_\subseteq := [(-1) \cdot x]_\subseteq$ unabhängig von der Wahl des Vertreters der Klasse $[x]_\subseteq$.

Nun gibt es die natürliche Abbildung
$$j \colon P \to P_\subseteq/Q_\subseteq,\ x \mapsto [x]_\subseteq.$$
Es ist offenbar $\ker j = Q$. Damit induziert j eine injektive Abbildung $P/Q \hookrightarrow P_\subseteq/Q_\subseteq$. Es ist klar, dass $+$ die Addition von P/Q fortsetzt. □

Sei $\mathscr{P}$ derart, dass $P_\subseteq = \mathbb{Z}^B$ ein freier $\mathbb{Z}$-Modul ist (dann nennen wir auch $\mathscr{P}$ frei mit Basis B). Dann gibt es die Projektion
$$\pi_\uparrow \colon P_\subseteq \to P,\ \text{„}\uparrow \mapsto 0\text{“},$$
welche jedem partiellen Vektor $x \colon \subseteq B \to \mathbb{Z}$ den durch Null fortgesetzten Vektor $\pi_\uparrow(x) \colon B \to \mathbb{Z}$ zuordnet: jedes $b \in B \setminus \operatorname{im}_B x$ bekommt dabei die Koordinate $0 \in \mathbb{Z}$ zugewiesen.

Lemma A.7. *Seien $\mathscr{Q}, \mathscr{S}$ partielle Teilmoduln des freien partiellen $\mathbb{Z}$-Moduls $\mathscr{P}$ mit Basis B. Weiter sei $0 \neq Q \subseteq S$. Dann ist $S_\subseteq/Q_\subseteq \cong S/Q$. Insbesondere ist $\mathscr{S}/\mathscr{Q}$ ein (totaler) $\mathbb{Z}$-Modul.*

Beweis. Es ist offenbar auch $Q_\subseteq \subseteq S_\subseteq$, weshalb $S_\subseteq/Q_\subseteq$ existiert.

Wir zeigen, dass für jedes $x \in S_\subseteq$ stets

$$0_\uparrow := x + (-1) \cdot \pi_\uparrow(x) \in Q_\subseteq$$

gilt. Es ist nämlich $0_\uparrow \colon \subseteq B \to \mathbb{Z}$ der partielle Vektor, der jedem $b \in \operatorname{im} x$ den Wert $0 \in \mathbb{Z}$ zuweist. Damit ist $0_\uparrow \in Q_\subseteq$.

Somit folgt, dass die Injektion

$$\iota \colon S/Q \to S_\subseteq/Q_\subseteq,\ [x] \mapsto [x]_\subseteq$$

surjektiv ist. Mithin ist $\iota(-[x]) = -[x]_\subseteq$. Hieraus ergibt sich, dass $S_\subseteq/Q_\subseteq$ ein $\mathbb{Z}$-Modul ist, der zu S/Q isomorph ist. □

Bemerkung A.8. *In Lemma A.7 ist es wesentlich, dass $Q \neq 0$ ist. Für den freien Modul 0 gilt nämlich $0_\subseteq = 0$. Dann gilt aber für $S \neq 0$:*

$$S/0 \cong S \not\cong S_\subseteq \cong S_\subseteq/0_\subseteq,$$

also ist die Aussage des Lemmas in diesem Fall nicht gültig.

Schließlich wollen wir obige Theorie auf relationale Kettenkomplexe anwenden. Es sei dazu

$$X \colon \cdots \longrightarrow C_{n,\subseteq} \xrightarrow{\delta_n} C_{n-1,\subseteq} \longrightarrow \cdots$$

ein relationaler Kettenkomplex. Dann ist die n-te Homologie von X erklärt als

$$H_n(X, \mathbb{Z}) = \operatorname{Kern} \delta_n / \operatorname{Bild} \delta_{n+1}.$$

Dies ist wohldefiniert, da $\delta_{n+1} \circ \delta_n = 0$ in der Tat $\operatorname{Bild} \delta_{n+1} \subseteq \operatorname{Kern} \delta_n$ impliziert.

Sei $X^{\mathrm{tot}} = (C, d)$ der Kettenkomplex, der aus X dadurch entsteht, dass alle $\uparrow$-Werte im Differentialoperator δ durch Nullen ersetzt werden. Dann erlaubt folgendes Lemma den Vergleich von Homologien:

Theorem A.9. *Für die Homologien von X und X^{tot} gilt:*

$$H_n(X, \mathbb{Z}) \cong H_n(X^{\mathrm{tot}}, \mathbb{Z}).$$

Beweis. Bezeichnen wir den Randoperator von X^{tot} mit ∂_n, so lässt sich unschwer nachweisen, dass $(\ker \partial_n)_\subseteq = \ker \delta_n$ und $(\operatorname{Bild} \partial_{n+1})_\subseteq = \operatorname{Bild} \delta_{n+1}$ gilt. Dann folgt die Aussage unmittelbar aus Lemma A.7. □

Literatur

[1] Bangartz, Hans-Joachim, Michael Griebel und Christoph Zenger: *Einführung in die Computergraphik*. vieweg, 1996.

[2] Bradley, Patrick Erik: *Euler-Poincaré Operators for architectural complexes*. Eingereicht bei *International Journal of Computational Geometry & Applications*.

[3] Bradley, Patrick Erik und Martin Behnisch: *A topological database for urban space-time data*. in Arbeit.

[4] Bradley, Patrick Erik und Norbert Paul: *Using the relational model to capture topological information of spaces*. Eingereicht bei *The Computer Journal*.

[5] Breunig, Martin, Patrick Erik Bradley, Norbert Paul und Andreas Thomsen: *Modellierung und Verwaltung der Topologie in CAD-Systemen*. Antragstext, 2007.

[6] Codd, Edgar F.: *The relational model for database management*. Addison-Wesley, 1990.

[7] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Design patterns. Elements of reusable object-oriented software*. Addison-Wesley, 28. Auflage, 2004.

[8] Hatcher, Allen: *Algebraic Topology*. Cambridge University Press, 2002. Verfügbar auch unter http://www.math.cornell.edu/~hatcher/.

[9] Kemper, Alfons und André Eickler: *Datenbanksysteme*. Oldenbourg, 5., aktualisierte und erw. Auflage, 2004.

[10] Kuratowski, Kazimierz: *Topology*, Band II. Acad. Pr., 1968.

[11] Maier, David: *The theory of relational databases*. Pitman, 1983.

[12] Paul, Norbert: *A Complex-Based Building Information System*. In: *Proc. 24th Conf. on Education in Computer Aided Arch. Design in Europe*, 2007.

[13] Paul, Norbert: *Pflichtenheft zum Prototypen des Forschungsprojekts Architektonische Komplexe*. Technischer Bericht 2008-02, Institut für Industrielle Bauproduktion, Universität Karlsruhe, 2008.

[14] Paul, Norbert: *Topologische Datenbanken für Architektonische Räume*. Doktorarbeit, Universität Karlsruhe, eingereicht.

[15] Paul, Norbert und Patrick Erik Bradley: *A specification of The COMP*. in Arbeit.

[16] Paul, Norbert und Patrick Erik Bradley: *Relationale Datenbanken für die Topologie architektonischer Räume*. In: *Forum Bauinformatik 2005. Junge Wissenschaftler forschen*, 2005.