Zur kategoriellen Beschreibung von Schichtenarchitekturen

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik

von
Georgios Lajios

Dortmund
2005

Tag der mündlichen Prüfung: 20.12.2005

Dekan: Prof. Dr. Bernhard Steffen

Gutachter: Prof. Dr. Ernst-Erich Doberkat
Prof. Dr. Peter Padawitz

Inhaltsverzeichnis

Kapitel 1

Einleitung

Schichtenarchitekturen gehören zu den am weitesten verbreiteten Architekturmustern für Softwaresysteme. Bekannte Beispiele sind das ISO/OSI-Netzwerk-Modell [Tan02, 1.4.1], die Schalen eines Betriebssystems [Tan01, 1.4.2] oder das Drei-Schichten-Modell von betrieblichen Informationssystemen [GJM03, 4.7.4]. Eine Schichtenarchitektur ist ein komponentenbasiertes Softwaresystem, bei dem die Komponenten in eine endliche Zahl von Schichten partitioniert sind. Charakteristisch für Schichtenarchitekturen sind folgende Merkmale [SG96, BMR+96, Szy98]:

- Das Softwaresystem ist aus Komponenten aufgebaut.
- Die Menge der Komponenten ist in Schichten eingeteilt.
- Die Menge der Schichten ist linear geordnet. Jede Schicht kommuniziert nur mit den unmittelbar benachbarten Schichten.
- Die Kommunikation findet über definierte Schnittstellen statt.

Schichtenarchitekturen bieten vielfältige Vorteile sowohl für Entwurf und Implementierung als auch für die Wartung eines Softwaresystems. Beim Entwurf kann man in der obersten Schicht komplexe, benutzernahe Dienste einplanen und dann von Schicht zu Schicht sukzessive auf niedrigere Abstraktionsebenen hinabsteigen. Oft ist es zwar nicht trivial, das richtige Abstraktionsniveau für eine Schichteneinteilung zu finden, denn diese legt bereits früh eine Grundstruktur für den weiteren Entwicklungprozess. Jedoch

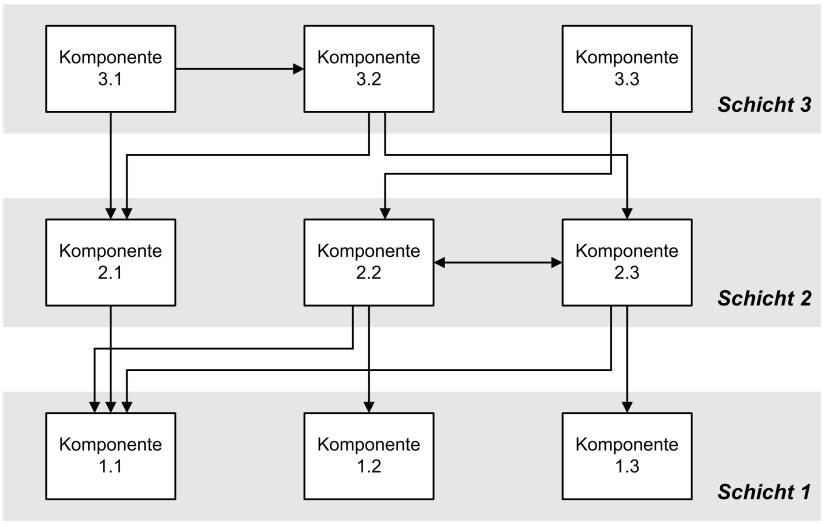


Abbildung 1.1: Beispiel eines Schichtensystems

ist eine spätere Verfeinerung der Schichten jederzeit möglich. In der Implementierungsphase kann Zeit eingespart werden, da die einzelnen Schichten unabhängig voneinander implementiert werden können. Der hierarchische Aufbau von Schichtenarchitekturen begünstigt schließlich die Wartung, da sich Änderungen nicht unkontrolliert ausbreiten können, sondern im Idealfall nur die direkten Nachbarschichten betreffen. So ist es möglich, einzelne Teile auszutauschen, ohne die Gesamtstruktur des Softwaresystems in Frage zu stellen.

Schichten können einzelne fachliche Belange kapseln und besitzen dabei selbst einen modularen Aufbau aus Komponenten. Komponenten sind in dieser Arbeit zustandsbehaftete Einheiten eines Softwaresystems, die eine definierte, abgegrenzte Funktionalität bereitstellen. Auf diese kann der Aufrufer, der in der Regel eine andere Komponente ist, über Schnittstellen zugreifen. In der Spezifikation der Schnittstellen ist festgelegt, was eine Komponente anderen Komponenten zusichert. Diese sehen nur die Schnittstelle und besitzen keinerlei Kenntnis darüber, wie die Komponente implementiert ist oder wie viele andere Komponenten zur Realisierung einer Funktionalität beitragen.

Beispielhaft zeigt Abbildung 1.1 ein System mit drei Schichten, die aus jeweils drei Komponenten bestehen. Die Kästchen stellen Komponenten dar, die Pfeile dazwischen sind die Konnektoren, die Komponenten miteinander verbinden. In Richtung der Pfeile können Aufrufe zwischen den Komponenten stattfinden. So kann Komponente 3.1 die Komponenten 3.2 und 2.1 aufrufen, nicht jedoch Komponente 1.1. Ein Zugriff auf die Ressourcen von Komponente 1.1 durch 3.1 ist nur indirekt über Komponente 2.1 möglich.

Ein typisches Szenario sieht wie folgt aus: Der Benutzer ruft Komponente 3.3 der obersten Schicht auf, die beispielsweise die graphische Benutzerschnittstelle sein könnte. Sie leitet den Aufruf weiter an Komponente 2.2. Dort findet eine teilweise Verarbeitung der Daten statt, bevor Komponente 1.1 aufgerufen wird. In Abhängigkeit vom Ergebnis dieses Unteraufrufs stellt Komponente 2.2 einen weiteren Unteraufruf an Komponente 1.2. Aus den bisher gewonnenen Informationen erstellt Komponente 2.2 schließlich das Ergebnis, das sie in aufbereiteter Form an Komponente 3.3 weiterleitet.

Ein formales Modell von Schichtenarchitekturen muss es gestatten, Systeme aus Komponenten zusammenzusetzen und diese in Schichten einzuteilen. Ferner muss es das Zusammenspiel der Komponenten bei der Verarbeitung eines Aufrufs abbilden. Als wichtige Rahmenbedingungen für die Modellierung fordern wir folgende Punkte:

- Die Komponenten, Konnektoren, Schnittstellen und Schichten sollen im Modell explizit repräsentiert sein. Nur so kann man sie beliebig komponieren und Systeme aus ihnen konstruieren.
- Eine Trennung zwischen den Komponenten selbst und der Konfiguration von Komponenten ist wünschenswert. Dazu ist eine geeignete Beschreibung der Komponentenbeziehungen nötig. Verschiedene Schichten-Einteilungen, z.B. Verfeinerungen, müssen flexibel handhabbar sein.
- Die architekturelle Sicht auf ein System ist grobgranular. Dem sollte das Abstraktionsniveau des Modells Rechnung tragen. Insbesondere impliziert das den Verzicht auf die Modellierung der Algorithmik.
- Dennoch müssen auch dynamische Aspekte berücksichtigt werden. Eine rein statische Modellierung reicht zur Charakterisierung einer Architektur nicht aus.

Unsere Vorgehensweise ist zweischrittig: Zunächst definieren wir ein statisches Modell, das Komponenten mit ihren Schnittstellen und deren Konfiguration mit Hilfe von Konnektoren umfaßt. Wir verwenden Methoden aus der Graphentheorie und der Relationalen Algebra. Danach erweitern wir das Modell um einen dynamischen Teil, der das Zusammenspiel der Komponenten abbildet. Hierfür verwenden wir Methoden aus der Kategorientheorie und interpretieren die Daten eines Aufrufs in einer lextensiven Kategorie. Wir diskutieren nun einige wichtige Aspekte im einzelnen.

1.1 Der Begriff der Komponente

Im Kontext von Architekturen ist es üblich, die Bausteine als Komponenten, und nicht etwa als Module o.ä., zu bezeichnen [SG96, 7.2.3]. Man versteht unter einer *Komponente* einen für sich abgeschlossenen Teil eines Softwaresystems, der über festgelegte Schnittstellen anderen Komponenten Dienste zur Verfügung stellt und die Dienste anderer Komponenten nutzt. Eine Komponente ist prinzipiell wiederverwendbar, jedoch steht dieser Aspekt für unsere Untersuchungen nicht im Mittelpunkt des Interesses. Ebensowenig verlangen wir, dass eine Komponente unabhängig von anderen ausgeliefert werden kann, wie dies beispielsweise in [Szy98, 4.1.1] gefordert wird.

Diese Begriffsbildung steht in einem gewissen Konflikt[1] mit dem Komponentenbegriff im Bereich des *Component-Based Software Engineering* [HC01]. Dort werden Wiederverwendbarkeit und Unabhängigkeit in der Auslieferung als oberste Axiome angenommen, und man unterscheidet zwischen Komponenten und Komponenten-Instanzen. Ein System besteht in dieser Sichtweise nicht aus Komponenten, sondern aus Komponenten-Instanzen, die durch durch eine Skriptsprache zu einem lauffähigen Programm zusammengesetzt werden können [Szy98, NA03]. Es ist bemerkenswert, dass auch die Verfechter des Component-Based Software Engineering nicht durchgehend von *Komponenten-Instanzen* sprechen, sondern bisweilen auf das kürzere Wort *Komponente* verfallen.

Demgegenüber ist es im Bereich der Software-Architekturen üblich, die Bau-

[1] Dieser Begriffskonflikt ist so tiefgreifend, dass in der 2. Auflage (2003) von [BCK03] nicht mehr von *Komponenten*, sondern von *Softwareelementen* die Rede ist. Dies hat sich jedoch bisher in der Literatur zu Softwarearchitekturen nicht durchgesetzt.

steine eines Softwaresystems als Komponenten zu bezeichnen, ohne zwischen einer abstrakten Komponente und ihrer konkreten Instanz zu unterscheiden. Diesen Komponentenbegriff findet man z.B. bei Barbosa [Bar01]:

What is understood by a software component: the specification of a state-based module, eventually acting as 'building block' of larger systems. [...] Such a component should encapsulate a number of services through a public interface which provides limited access to its internal state. Furthermore, it persists and evolves in time. [ibd., 1.1.1]

Zentrale Punkte sind die Existenz eines internen Zustands und die Möglichkeit zur Interaktion mit anderen Komponenten über Schnittstellen. Eine ähnliche Sichtweise vertritt [BS01]. Ein zustandsbehafteter Komponentenbegriff wird auch in [BMR+96] verwendet. Wir werden im folgenden (Kap. 2) einen formalen Komponentenbegriff einführen, der sich konzeptionell an Barbosa anlehnt.

1.2 Kategorientheorie und Informatik

Wir verwenden für unsere Modellierung Methoden aus der Kategorientheorie. Die Kategorientheorie ist eine mathematische Theorie, die seit den 1940er Jahren entwickelt wurde, um einen einheitlichen Begriffsrahmen zu schaffen, der es erlaubt, über verschiedene Gebiete der Mathematik zu sprechen und Transformationen zwischen ihnen zu beschreiben, insbesondere zwischen Topologie und Algebra [EM45, Mac97].

Die kategorielle Beschreibung abstrahiert von der inneren Struktur mathematischer Objekte und betrachtet nur das Zusammenspiel der *Morphismen*, die i.a. die strukturerhaltenden Abbildungen zwischen zwei Objekten sind. Da die Objekte jedoch keine Mengen sein müssen, sind auch die Morphismen nicht unbedingt Abbildungen, sondern müssen nur das wesentliche Extrakt einer Kollektion von Abbildungen erfüllen, nämlich eine assoziative Verknüpfungsstruktur mit neutralem Element darstellen[2].

[2]Eine genaue Definition findet sich in Anhang B.

Grundlegendes Beispiel ist die Kategorie **Set** der Mengen und Abbildungen. Unter Beibehaltung der Objekte erhält man zwei weitere in der Informatik wichtige Kategorien, **Par** und **Rel**, wenn man partielle Abbildungen oder Relationen als Morphismen verwendet. Ein anderes Beispiel ist die Kategorie **Grph** der Graphen und Graphhomomorphismen, von der für verschiedene Anwendungen allerlei Varianten betrachtet werden können, z.B. gerichtete/ungerichtete, markierte, Hypergraphen, etc.

Der abstrakte Blickwinkel, den die Kategorientheorie einnimmt, hat zur Konsequenz, dass Objekte in kategorieller Sprache nur bis auf Isomorphie beschrieben werden können, also unabhängig von jedweder speziellen Darstellung. Kategorielle Beschreibungen spezifizieren also und geben keine konkrete Implementierung an [EGRW95]. Diese ist gleichwohl durch den Übergang in eine spezielle Kategorie und Festlegung konkreter Objekte und Morphismen darin möglich. Die Unabhängigkeit von einer speziellen Repräsentation ist in der Kategorientheorie jederzeit gewahrt, denn auch alle Konstruktionen sind nur bis auf Isomorphie definiert. Dies entspricht dem Leitgedanken der Spezifikation, nicht mehr als nötig festzulegen. Eine konkrete Repräsentation von Objekten ist für theoretische Untersuchung meist nicht gewünscht und bringt sogar Nachteile mit sich, denn bei Transformationen muss man stets die Identität der Elemente im Auge behalten, um sicherzustellen, dass neue hinzukommende Elemente eine neue Identität besitzen und nicht mit zuvor gelöschten verwechselt werden können, etc. Da theoretische Untersuchungen in der Regel ohnehin abstrakter Natur sind und im Anwendungsfall einer Abbildung auf die Objekte der Realität bedürfen, ist es müßig, die Identität abstrakter Objekte zu verwalten. Insofern kann man sagen, dass nichtkategorielle Modellierungsansätze oftmals überspezifiziert und ihre Resultate nur deswegen verallgemeinerbar sind, weil man sie auf natürliche Weise von ihrem Gegenstand auf dessen Isomorphieklasse übertragen kann.

Der Kategorientheorie ist zu verdanken, dass eine große Zahl von Konstruktionen in verschiedenen Gebieten der Mathematik in einer einheitlichen Weise abstrakt beschrieben werden kann. So liefert der kategorielle Produktbegriff in der Kategorie der Mengen das kartesische Produkt, in der Kategorie der Gruppen das direkte Produkt und in der Kategorie der topologischen Räume das topologische Produkt. Dies ermöglicht einen einheitlichen begrifflichen Rahmen nicht nur für die mathematische Theorie, sondern auch

für die mathematische Modellbildung in der Informatik. Resultate, die unter Verwendung rein kategorieller Methoden gewonnen werden, lassen sich leicht auf einzelne Kategorien spezialisieren, sofern diese die (ebenfalls kategoriell zu formulierenden) Voraussetzungen erfüllen.

Die Spezialisierung allgemein gewonnener Resultate spiegelt ein deduktives Vorgehen wider: Aus einem allgemeinen Satz wird ein spezieller gewonnen. Umgekehrt kann die Kategorientheorie auch ein induktives Vorgehen unterstützen und dabei helfen, aus speziellen Resultaten allgemeine zu gewinnen. Insbesondere kann sie dazu beitragen, den inhaltlichen Kern, der ähnlichen Resultaten in verschiedenen Gebieten zugrunde liegt, zu bestimmen.

Der hohe Abstraktionsgrad der Kategorientheorie führt naturgemäß dazu, dass man nur sehr wenige gemeinsame Eigenschaften extrahieren kann, die für eine bestimmte Konstruktion in allen Kategorien gelten. Das beginnt schon damit, dass viele Konstruktionen, wie z.B. Produkte, nicht in allen Kategorien existieren. Auch dort, wo sie existieren, haben sie in manchen Fällen recht ungewohnte Eigenschaften. Dies hat dazu geführt, bestimmte Klassen von Kategorien zu identifizieren, in denen Existenz und „Wohlverhalten“ einer Auswahl von Konstruktionen sichergestellt ist. So hat sich jüngst herausgestellt, dass die kategoriellen Konstruktionen, die in der Theorie der Graphtransformationen verwendet werden (Doppel-Pushout-Ansatz [Roz97]), in der Klasse der adhäsiven Kategorien befriedigend funktionieren [LS04, EHPP04]. Andere spezielle Klassen von Kategorien mit besonderen Eigenschaften sind kartesische, kartesisch abgeschlossene [BW99, 5.3.9, 6.1.3], distributive, extensive [CLW93] oder algebraische Kategorien [HS73, §32] sowie Topoi, Logoi, Allegorien [Fv90]. Die Identifikation der Kategorien, in denen sich ein Problem lösen lässt, birgt sowohl Einsichten in das Problem selbst als auch in den Lösungsweg, denn man abstrahiert von akzidentellen Eigenschaften und wird an die Substanz der mathematischen Objekte geführt.

In den 1970er und 80er Jahren wurde die Verwendung kategorieller Methoden in der Informatik durch die „ADJ“-Gruppe[3] bei IBM Research (Goguen, Thatcher, Wagner, Wright) propagiert. Anwendungsgebiete waren die Semantik von Programmiersprachen und die Spezifikation abstrakter Datentypen. Eine Untersuchung der Einsatzmöglichkeiten von Kategorientheorie in

[3] Zur Namensgebung und weiteren historischen Hintergründen siehe Wagner [Wag02].

der Informatik und Hinweise zur Verwendung einzelner Konstruktionen gibt Goguen in seinem *Categorical Manifesto* [Gog91]. Speziell auf den Bereich der algebraischen Spezifikation gerichtet, untersucht [EGRW95] die Rolle der Kategorientheorie. Nicht nur zeigt sich, dass kategorielle Methoden bei der Formalisierung hilfreich sind, sondern es werden auch konkrete Beispiele von Problemen aus den Bereich der algebraischen Spezifikation aufgeführt, die mit Hilfe der Kategorientheorie gelöst werden konnten.

Coalgebren sind ein in jüngster Zeit sehr wichtiges Gebiet, in dem kategorielle Methoden zur Anwendung kommen. Zustandsbasierte dynamische Systeme, die traditionell als Transitionsssysteme oder Automaten modelliert werden, lassen sich mit dem Konzept der Coalgebra uniform beschreiben [Rut00]. Obwohl viele Untersuchungen sich auf die Kategorie **Set** der Mengen beschränken, kann man Coalgebren in einer beliebigen Kategorie $\mathcal{C}$ definieren: Eine Coalgebra zu einem Endofunktor $F : \mathcal{C} \to \mathcal{C}$ ist ein Morphismus $X \to FX$, wobei X ein beliebiges Objekt ist. Durch geeignete Wahl von F lassen sich beispielsweise deterministische und nichtdeterministische Transitionssysteme, entweder mit oder ohne Ein- bzw. Ausgaben, beschreiben [Rut00]. Auch probabilistische Modelle sind möglich [dVR97]. Bei der Untersuchung von Eigenschaften coalgebraischer Transitionssysteme stellt sich das Prinzip der *Coinduktion* als elegantes Hilfsmittel heraus. Daneben finden auch allgemeine kategorielle Konstruktionen wie Limiten und Colimiten in der Theorie der Coalgebren Verwendung [Rut00, Sect. 4].

1.3 Verwandte Arbeiten

Im folgenden diskutieren wir verwandte Arbeiten. Zunächst gehen wir speziell auf Arbeiten ein, die ebenfalls kategorielle Methoden zugrunde legen, dann erörtern wir einige alternative formale Ansätze im Bereich der Modellierung komponentenbasierter Softwaresysteme.

Kategorielle Methoden

Der hier vorgestellte Ansatz lehnt sich konzeptionell an den in [Dob03a] präsentierten an, der dem Architekturmuster *pipeline* gewidmet ist. Grundlage des Modells ist ein interpretierter Graph, dessen Knoten die *filter* und dessen

Kanten die *pipes* darstellen. Die Semantik wird in der Kleisli-Kategorie $\mathcal{C}_T$ zu einer Monade (T, η, μ) [Mac97, VI.5] in einer Kategorie $\mathcal{C}$ definiert. Diese ermöglicht nichtdeterministische und stochastische Modelle, wenn man geeignete Monaden vewendet. Den Kanten des Graphen sind Objekte aus $\mathcal{C}$ zugeordnet, und die Semantik eines Knotens v ist durch einen Morphismus $X \to TY$ gegeben, wobei X das Produkt der Objekte an den eingehenden Kanten von v und Y das Produkt der Objekte an den ausgehenden Kanten ist.

Schichtenarchitekturen unterscheiden sich dadurch von *pipelines*, dass in ihnen beim Aufruf einer Komponente in Abhängigkeit von den Eingaben dynamisch entschieden wird, welche Unteraufrufe benötigt werden, bevor eine Antwort an den Aufrufer gegeben werden kann. Daher ist eine Steuerungslogik notwendig, die darüber entscheidet, welche Komponenten aktiviert werden und in welcher Reihenfolge dies stattfindet. Es liegt also zwar eine statische Konfiguration der Komponenten, aber ein dynamischer Kontrollfluss vor. Während bei einem statischen Kontrollfluss, wie er beispielsweise bei den *pipelines* gegeben ist, durch die Kleisli-Komposition eine Verknüpfung nichtdeterministischer Berechnungen möglich ist, scheidet dies bei einem dynamischen Kontrollfluss aus, da im allgemeinen keine kanonische Projektion $TX \to X$ existiert. Es muss aber in jedem Schritt über den weiteren Kontrollfluss entschieden werden.

Man könnte zwar mithilfe von Eilenberg-Moore-Algebren [Dob04] eine *Derandomisierung* vornehmen, aber damit würde man nicht mehr in der Kleisli-Kategorie arbeiten, sondern erhielte sofort nach Komposition von Algebra und Coalgebra wieder einen gewöhnlichen Morphismus $X \to X$ und stände vor demselben Problem wie zuvor, nämlich der Frage, wie über den weiteren Kontrollfluss entschieden werden soll.

Auch Barbosa verwendet Monaden (T, η, μ) zur Modellierung von Komponenten. Die Ein- und Ausgabeobjekte I, O sind dort in den Funktor $F = T(Id \times O)^I$ kodiert, während das der Coalgebra $U \to F(U)$ zugrundeliegende Objekt den Zustand repräsentiert [Bar01, 5.1]. Für die Monade zum identischen Funktor $T = Id$ kann man eine Coalgebra als Mealy-Automaten interpretieren [Rut05, Sect. 2], andere Wahlen für T reichern das Modell um nichtdeterministische Aspekte an. Barbosas Modell, das eine Vielzahl von Variationen zulässt, stützt sich im Kern ebenfalls auf die Kleisli-Komposition

von Morphismen. Auch hier fehlt eine dynamische Entscheidung über die Aufrufreihenfolge; die Komponenten werden stets in kanonischer Reihenfolge durchlaufen.

Wermelinger und Fiadeiro [WF98] verwenden Methoden aus der Kategorientheorie, um Verbindungsmuster mobiler Programme zu modellieren, setzen jedoch auf einem rein mengentheoretischen Ansatz auf. In einer Kategorie $\mathcal{PROG}$ von Programmen und Superpositionsmorphismen werden Konstruktionen durch Bildung von Colimiten durchgeführt. So entspricht die Komposition zweier Programme, die eine gemeinsame Signatur besitzen, einem Pushout. Es zeigt sich, dass alle endlichen Colimiten existieren, so dass beliebige komplexe Operationen möglich sind. Durch Änderung der Morphismenmengen ist eine flexible Anpassung der möglichen Superpositionsmorphismen möglich. Dies wird bei der Synchronisierung von mobilen Programmen ausgenutzt. Eine ähnliche Herangehensweise findet sich in [FM96].

Distributive und extensive Kategorien, wie wir sie verwenden, haben schon verschiedentlich in der Informatik Anwendung gefunden: In [Wal92] und [KW93] werden Konzepte distributiver Kategorien zur Definition einer Semantik imperativer Programmiersprachen eingesetzt. Die Distributivität erlaubt die Konstruktion einer *if-then-else*-Anweisung unter Verwendung der eindeutigen vermittelnden Morphismen in Produkten und Coprodukten. Als Boolescher Datentyp findet das Coprodukt $I + I$ des terminalen Objekts mit sich selbst Verwendung. Jedoch werden wichtige Konstruktionen nur für den Spezialfall **Set** durchgeführt. Nur in [WKW95] werden Teile der Ergebnisse aus den anderen beiden Arbeiten unter Verwendung globaler Elemente auf einem rein kategoriellen Fundament rekonstruiert. Ein Programm mit Eingabeobjekt X und Ausgabeobjekt Y sind dort ein auf Coprodukten operierender Morphismus $e : X + U + Y \to X + U + Y$, dessen Einschränkung auf den Cofaktor Y mit dem identischen Morphismus übereinstimmt. Im Spezialfall der Kategorie **Set** der Mengen und Abbildungen kann man sich die Berechnung zu einer Eingabe $x \in X$ wie folgt vorstellen: Es wird solange iterativ e angewendet, bis das Ergebnis in Y liegt. Auf diese Weise induziert ein Programm e einen Morphismus $\bar{e} : X \to Y$, die sogenannte *Fix-Punkt-Semantik*. Dieses Vorgehen lässt sich auch allgemein kategoriell beschreiben. Es stellt sich heraus, dass in einer beliebigen distributiven Kategorie – im Gegensatz zu **Set** – weder Eindeutigkeit noch Existenz einer

Fix-Punkt-Semantik gewährleistet sind [ibd., 4.6]. Wie wir in Abschnitt 4.5 sehen werden, kann dieses Problem in lextensiven Kategorien nicht auftreten.

Die Idee, Morphismen in Coprodukte zu verwenden, geht auf Arbeiten von Elgot zur Semantik von Programmiersprachen zurück [Elg75, 4.1], in denen Iteration als Transformation eines Morphismus $f : X \to X + Y$ in einen Morphismus $f^\dagger : X \to Y$ beschrieben wird. Wir gehen darauf in Kapitel 5 näher ein.

Lextensive Kategorien finden u.a. in der kategoriellen Modellierung von Datenbanken Anwendung [JRW02, JR01]. Sie werden dort als Modelle für *Sketches* [BW99, 10.4] verwendet, welche *entity-attribute*-Diagramme repräsentieren. Außerdem wurden extensive Kategorien zur Modellierung von Datentypen herangezogen [AAG03]. Container lassen sich abstrakt in der Kategorie $\mathcal{C} \downarrow A$ der Objekte über A, einem Spezialfall der Komma-Kategorie modellieren (vgl. Anhang B). Ein Container ist dann ein Paar aus einem Objekt A und einer endlichen Menge von Objekten der Kategorie $\mathcal{C} \downarrow A$. Die Eigenschaften extensiver Kategorien können für die Konstruktion von Limiten und Colimiten derartiger Container-Kategorien ausgenutzt werden.

Manes verwendet in seinem Buch [Man92] Kategorien, die mit den extensiven eng verwandt sind. Die Bezeichnungen dort unterscheiden sich jedoch von den inzwischen üblich gewordenen. Seine zentrale Begriffsbildung, die er als Boolesche Kategorie bezeichnet, umfasst neben der Existenz von Coprodukten und initialen Objekten eine Forderung, die der Universalität von Coprodukten nahekommt. Daneben diskutiert er extensive Kategorien, bezeichnet diese jedoch als distributiv. Gegenstand von Manes' Untersuchungen ist die Definition einer kategoriellen Semantik, die sich an die bekannte *Predicate-Transformer*-Semantik nach Hoare und Dijkstra anlehnt.

Ansätze zur Modellierung von Architekturen und komponentenbasierten Systemen

Ein Möglichkeit zur Beschreibung komponentenbasierter Softwaresysteme ist ein *stream*-basiertes Vorgehen, wie es sowohl FOCUS [BS01] als auch REO [AR02] verwenden. Bei diesen Ansätzen stehen zeitliche und kausale Aspekte im Vordergrund. Wir wollen nur auf den Ansatz von Arbab und Rutten näher eingehen. Diese betrachten *timed data streams*, das sind Paare bestehend

aus einer unendlichen Folge von Elementen einer Datenmenge D und einer unendlichen Folge positiver reeller Zahlen. Die reellen Zahlen repräsentieren die Zeit, zu der die Ein- oder Ausgabe der Datenelemente erfolgt. Mithilfe des Prinzips der Coinduktion [Rut00] und von Bisimulationen können diese Folgenpaare elegant manipuliert werden. Binäre Relationen über timed data streams bezeichnen Arbab und Rutten als *channels*. Sie geben eine Reihe von *basic channels* an, aus denen durch Komposition neue entstehen, die auf ihre Äquivalenz hin untersucht werden können. Der REO-Kalkül bietet eine flexible Möglichkeit, komponentenbasierte Softwaresysteme zu definieren und zu manipulieren. Nachteile sind die Beschränkung auf einen statischen Kontrollfluss und die bereits angesprochene Festlegung auf einen Datentyp, der mit coalgebraischen Ansätzen einhergeht.

Architekturen lassen sich prinzipiell auch in der *Unified Modeling Language* (UML) beschreiben, allerdings ist diese nicht speziell darauf ausgerichtet. So bildet man Komponenten und Konnektoren durch dieselben Sprachelemente ab [MRRR02]. Außerdem sind die besonderen Bedingungen einzelner Architekturmuster der UML nicht inhärent, sondern müssen vom Modellierer beachtet werden. Dies ist nicht nur mühevoll, sondern auch fehleranfällig. Ferner müssen die Bedingungen in geeigneter Weise zusätzlich dokumentiert werden, damit sie bei späteren Erweiterungen nicht verletzt werden. Eine systematische Möglichkeit dazu bietet die *Object Constraint Language* (OCL) [ibd., Sect. 5]. Spezielle Architekturbeschreibungssprachen [SG96, DvdHT05] ermöglichen es hingegen, die Bedingungen einzelner Architekturmuster schon auf der Ebene der Syntax zu berücksichtigen.

Eine Technik zur Formulierung des Zusammenspiels von Objekten, die einander aufrufen, ist durch die UML-Sequenzdiagramme gegeben, die allerdings in der Regel nur exemplarischen Charakter besitzen und für die erst wenige Ansätze zu einer Semantik existieren [LLJ04]. Unser dynamisches Modell bietet mit den *Aufrufbäumen* einen alternativen Ansatz, der auf Methoden aus dem Compilerbau aufsetzt (vgl. Abschnitt 5.4). Da wir uns nicht auf objektorientierte Sprachen einschränken und mit der kategoriellen Spezifikationsmethode auf einer anderen Abstraktionsebene arbeiten, sind die Ansätze jedoch nicht direkt vergleichbar.

1.4 Überblick

In der vorliegenden Arbeit modellieren wir Schichtenarchitekturen. Die Modellierung ist aufgeteilt in ein statisches und ein dynamisches Modell. Das statische Modell repräsentiert die Architektur eines geschichteten Systems in ihrer Zusammensetzung aus Komponenten und Schnittstellen (Kapitel 2). Hierzu finden Methoden aus der Graphentheorie und der Relationalen Algebra Anwendung. Operationen auf Schichtensystemen untersuchen wir in Kapitel 3. Dort betrachten wir zunächst universelle Konstruktionen und definieren dann architekturspezifische Operationen mithilfe von Graphtransformationen.

Das statische Modell kann zu einem dynamischen Modell, das die Berechnung eines Aufrufs im System widerspiegelt, erweitert werden, indem jede Komponente eine Implementierung erhält.

Die Funktionalität einer Komponente modellieren wir als einen Morphismus $f : X \to A + B$, dessen Bildbereich das Coprodukt zweier Objekte A und B ist. Dieser kann nicht nur Daten transformieren, sondern auch den weiteren Kontrollfluss bestimmen, je nachdem, ob das Bild eines globalen Elements g unter f in A oder in B „liegt“. Was „liegt“ dabei kategoriell bedeutet und in welchen Kategorien man diese Frage überhaupt sinnvoll stellen kann, wird in Kapitel 4 geklärt. Es stellt sich heraus, dass Forderungen an die Eigenschaften der Coprodukte gestellt werden müssen, die auf den Begriff der lextensiven Kategorie führen. Das dynamische Modell wird dann in Kapitel 5 im Detail erläutert. Kapitel 6 ist der Untersuchung der Verhaltensäquivalenz von Schichtensystemen gewidmet. Zentral dafür ist der Begriff der Bisimulation. Die Arbeit schließt mit Schlussbetrachtungen und einem Ausblick auf weitere Forschung.

1.5 Danksagung

Die vorliegende Dissertation entstand am Lehrstuhl für Software-Technologie der Universität Dortmund auf Anregung und unter Leitung von Prof. Dr. Ernst-Erich Doberkat, dem ich für seine engagierte Betreuung herzlich danke. Er hat entscheidende Verbesserungen an Inhalt und Struktur der Arbeit bewirkt.

Bei Prof. Dr. Peter Padawitz möchte ich mich für die Übernahme des Zweitgutachtens bedanken.
Dr. Alexander Fronk danke ich für seine Hinweise zum Umgang mit Formeln in der Relationalen Algebra und und Dr. Doris Schmedding für ihre Hilfe beim Korrekturlesen der Arbeit.

Kapitel 2

Statisches Modell

2.1 Software-Architektur

Der Begriff der Software-Architektur ist in den letzten Jahren zu einem Schlagwort in der Software-Entwicklung geworden. Angesichts der zunehmenden Komplexität von Software-Projekten wird eine gut geplante Software-Architektur immer mehr zu einem entscheidenden Qualitätsfaktor für das Produkt Software. Hier ist neben Kostenvorteilen in der Entwicklung vor allem der Wartungsaspekt hervorzuheben. Aus der Fülle an Literatur, die inzwischen zu diesem Thema existiert, lassen sich einige gemeinsame Aspekte extrahieren, die von den meisten Autoren akzeptiert werden. Wir wollen hierzu drei Definitionen aufführen und festhalten, welche Aspekte sie umfassen.
Aus Beratungen der IEEE Architecture Planning bzw. Working Group zwischen 1996 und und 1999 hat sich der im Oktober 2000 veröffentliche IEEE-Standard 1471-2000 *Recommended Practice for Architectural Description of Software-Intensive Systems* entwickelt. Dort wird der Begriff der Software-Architektur wie folgt definiert:

Architecture: the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.[IEE00]

In dem vielzitierten Standardwerk über Software-Architektur von Shaw und Garlan [SG96] wird dieser Grundbegriff wie folgt definiert:

The architecture of a software system defines that system in terms of computational components and interactions among those components. [ibd, 1.1]

Der Begriff der Software-Architektur ist mithin ein abstrakter und abgeleiteter. Zugrundegelegt wird ein Komponentenbegriff, und das Zusammenspiel der Komponenten untereinander macht die Software-Architektur aus. Die folgende Definition von Bass, Clements und Kazman bringt dies auf den Punkt:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. [BCK03, Sect. 2.1]

Die Architektur-Sicht auf ein Softwaresystem ist eine abstrakte: Strukturelle Fragen stehen im Vordergrund. Da die Zusammenstellung der Komponenten ein Softwaresystem wesentlich bestimmt, sind zentrale Fragen an eine Architektur:

- Welche Komponenten werden eingesetzt?
- Wie werden diese kombiniert? *(Konfiguration)*
- Welche Schnittstellen bieten die Komponenten?

Man könnte versucht sein, die Architekturbeschreibung im wesentlichen auf die *statischen* Eigenschaften eines Softwaresystems zu beschränken. Das ist jedoch nicht ausreichend. Ein mathematisches Modell für Softwarearchitekturen muss stets auch den dynamischen Ablauf im Auge behalten. Folgendes Beispiel soll das verdeutlichen: Eine azyklische *pipeline*-Architektur lässt sich durch verhaltensneutrale Transformationen stets in eine Form bringen, in der die Filter in Schichten angeordnet sind [Dob03a]. Dies macht sie dennoch nicht zu einer Schichtenarchitektur, denn in einer solchen findet ein ganz anderer Kontrollfluss statt. Während die *pipeline*-Architektur aus zustandlosen Filtern besteht, die stets in der gleichen Reihenfolge aktiv werden, funktioniert beispielsweise die klassische Drei-Schichten-Architektur betrieblicher Informationssysteme gänzlich anders: Ruft eine Komponente den Dienst,

den eine andere anbietet, auf, so erwartet sie in der Regel nach endlicher Zeit eine Antwort darauf. Der Kontrollfluss (zumindest des aktiven *threads*) ist solange unterbrochen, bis der Dienstleister seine Schuld erfüllt hat (vgl. das Szenario aus Abbildung 1.1). Die Komponenten haben Zustände, die sich während der Bearbeitung eines Aufrufs ändern können. Der Austausch von Informationen zwischen Komponenten ist allein über die Schnittstellen entlang von Konnektoren möglich.
Unser dynamisches Modell, das wir in Kapitel 5 beschreiben, bildet die zuletzt beschriebenen Vorgänge ab. Eine rein statische Beschreibung der Konfiguration reicht also nicht aus, um eine Architektur zu definieren. Der Fokus unserer Untersuchung liegt daher in der Struktur komponentenbasierter Systeme und dem Zusammenspiel ihrer Teile.

2.2 Schnittstellen und Komponenten

Komponenten sind die Bausteine von Softwarearchitekturen. Unser Komponentenmodell lehnt sich an die Definition von Barbosa an, die wir bereits in der Einleitung zitiert haben. Einige weitere wichtige Punkte, die eine Komponente ausmachen, führt [dW99, 10.1.2] im Einzelnen auf:

A coherent package of software implementation that

(a) can be independently developed and delivered,

(b) has explicit and well-specified interfaces for the services it provides,

(c) has explicit and well-specified interfaces for services it expects from others, and

(d) can be composed with other components, perhaps customizing some of their properties, without modifying the components themselves.

Wesentliche Merkmale einer Komponente sind also ein- und ausgehende Schnittstellen, die geeignet spezifiziert sein müssen, und die Möglichkeit, sie mit anderen Komponenten zu einem Softwaresystem zusammensetzen zu können. Die Schnittstellen legen die Außensicht der Komponente fest und abstrahieren von Details, die für die Softwarearchitektur nicht von Bedeutung sind. Über Schnittstellen stellen Komponenten anderen Komponenten

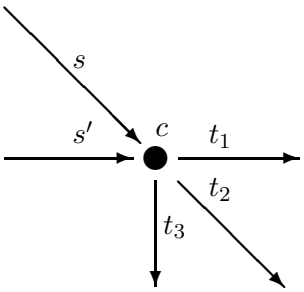
Dienste zur Verfügung und können auf Dienste anderer Komponenten zugreifen. Eine Komponente kann als Dienstanbieter, Dienstnutzer oder in beiden Rollen gleichzeitig auftreten.
Auf Architekturebene untersuchen wir die Beziehungen zwischen Komponenten in Form von angebotenen oder genutzten funktionalen Schnittstellen. Die Implementierung von Komponenten folgt dem *Geheimnisprinzip* [Par72, GJM03] und tritt aus Architekturperspektive gegenüber der Spezifikation von Komponentenschnittstellen in den Hintergrund.
In unserem formalen Modell hat jede Schnittstelle s einen eingehenden und einen ausgehenden Typ und ist durch diese in ihrer Außensicht vollständig bestimmt. Um den Ansatz möglichst allgemein zu halten, arbeiten wir jetzt und im folgenden in einer beliebigen, aber festen, Kategorie[1] $\mathcal{C}$.

Definition 2.1 *Ein Paar $s = (x, y) \in |\mathcal{C}| \times |\mathcal{C}|$ nennen wir* $\mathcal{C}$-Schnittstelle.

Wir geben den Projektionen $proj_i : |\mathcal{C}| \times |\mathcal{C}| \to |\mathcal{C}|$ besondere Namen. Die erste Projektion $proj_1$ bezeichnen wir mit τ_{in} und nennen das Bild $\tau_{in}(s)$ den eingehenden Typ von s. Die zweite Projektion $proj_2$ bezeichnen wir mit τ_{out} und nennen das Bild $\tau_{out}(s)$ den ausgehenden Typ von s. Das wesentliche einer Schnittstelle ist das Vorhandensein dieser Typisierungsabbildungen (τ_{in}, τ_{out}). Daher wollen wir auch beliebige Objekte mit Typisierungsabbildungen $\tau = (\tau_{in}, \tau_{out})$ als Schnittstellen bezeichnen.
In der Praxis besitzt jede Schnittstelle einen sie eindeutig identifizierenden Namen, der es ermöglichst, dass eine Komponenente mehrere Schnittstellen desselben Typs besitzt. Wir vernachlässigen dies in unserem Modell, um die Notation nicht unnötig kompliziert zu machen. Es wäre ohne weiteres möglich, das Modell um einen Namensraum zu erweitern und eine injektive Abbildung von der Menge aller Schnittstellen einer Komponente in diesen Namensraum zu definieren.
Auf Ebene der Spezifikation ist eine Komponente nichts anderes als die Summe ihrer Schnittstellen. Dabei müssen ein- und ausgehende Schnittstellen unterschieden werden.

Definition 2.2 *Eine* Komponente c *ist ein Paar (c^{in}, c^{out}), wobei c^{in}, c^{out} endliche Mengen von Schnittstellen sind. Wir nennen c^{in} die Menge der eingehenden und c^{out} die Menge der ausgehenden Schnittstellen.*

[1] Grundbegriffe der Kategorientheorie und einige spezielle Definitionen, die im folgenden verwendet werden, finden sich im Anhang.



$$
\begin{aligned}
c^{in} &= \{s, s'\} \\
c^{out} &= \{t_1, t_2, t_3\} \\
s &= (X, Y) \\
s' &= (U, V) \\
t_1 &= (X, V) \\
t_2 &= (X, U) \\
t_3 &= (A, B)
\end{aligned}
$$

Abbildung 2.1: Komponente c mit ein- und ausgehenden Schnittstellen

Es sind also die Schnittstellen, die eine Komponente ausmachen. Abbildung 2.2 zeigt eine Komponente mit ein- und ausgehenden Schnittstellen. Die Elemente von c^{in} und c^{out} können einem festen Universum entnommen werden und sind ansonsten beliebig[2].

2.3 Komposition von Komponenten

Bei der Erstellung eines komponentenbasierten Softwaresystems hat der Entwickler ein großes Maß an Freiheit hinsichtlich der Auswahl geeigneter Komponenten und der Art und Weise, wie er diese zusammenfügen möchte. Den dadurch entstehenden Aufbau eines Softwaresystems wollen wir im *Konfigurationsgraphen* $G = (G_V, G_E)$ darstellen, dessen Knoten die *Komponenten* und dessen Kanten die *Konnektoren* repräsentieren.

Definition 2.3 *Sei $G = (G_V, G_E, src, tar)$ ein (schwach) zusammenhängender, schlingenfreier, endlicher gerichteter Graph*[3]*. Jeder Kante $e \in G_E$ sei eine Schnittstelle $\tau(e)$ zugeordnet. Ein solches Paar (G, τ) nennen wir* Konfigurationsgraph.

Wir fordern, dass der Konfigurationsgraph zusammenhängend ist, da Komponenten, die in verschiedenen Zusammenhangskomponenten liegen, keinerlei Bezug zueinander haben und wir es de facto mit mehreren, unabhängigen

[2] Ähnlich wie es bei der Definition eines Graphen nicht darauf ankommt, wie die Knotenmenge gebildet wird, sondern allein auf das Zusammenspiel von Knoten und Kanten.

[3] Grundlegende Definitionen aus der Graphentheorie finden sich in Anhang A.1.

Softwaresystemen zu tun haben. Eine Komponente soll auf der Ebene der Architektur nicht auf sich selbst zugreifen. Inwiefern intern in der Implementierung rekursive Methodenaufrufe oder ähnliches stattfinden, ist davon nicht berührt. Die Annahme, dass der Konfigurationsgraph nur aus endlich vielen Komponenten besteht, ist eine für Softwaresysteme selbstverständliche Forderung.
In unserem statischen Modell von Schichtenarchitekturen spielen die Komponenten eine herausragende Rolle gegenüber den Konnektoren. In ihnen findet die Verarbeitung von Daten statt und sie sind Träger des lokalisierten Zustandes. In der Literatur zu Architekturen wird oft hervorgehoben, dass auch den Konnektoren eine große Bedeutung zukommt und sie als gleichwertig mit den Komponenten anzusehen sind, da sie die Art und Weise regeln, wie die Komponenten miteinander kommunizieren [SG96, 7.2]. Da unser Modell ein spezielles Architekturmuster abbildet, ist die erlaubte Art der Kommunikation so weitgehend festgelegt, dass die besondere Rolle der Konnektoren nicht zum Tragen kommt. Daher werden diese durch Kanten im Konfigurationsgraphen repräsentiert und besitzen bis auf den ein- und ausgehenden Typ keine weitere Struktur.
Bei der Zusammenstellung von Komponenten muss zwei Dingen Rechnung getragen werden:

1. Zwei Komponenten können nur entlang von Schnittstellen miteinander verbunden werden[4]. Dabei muss eine Komponente eine ausgehende und die andere eine eingehende Schnittstelle besitzen, und die beiden Schnittstellen müssen in ihrer Typisierung übereinstimmen.

2. Damit ein lauffähiges System entsteht, müssen alle ausgehenden Schnittstellen der beteiligten Komponenten mit eingehenden Schnittstellen anderer Komponenten verbunden sein. Denn über diese nimmt eine Komponente Leistungen anderer Komponenten in Anspruch, die sie für ihre Funktion benötigt. Dagegen dürfen eingehende Schnittstellen unverbunden bleiben, falls eine angebotene Funktionalität von keiner anderen Komponente genutzt wird.

[4]Eine direkte Kommunikation zwischen Komponenten ist nur über Schnittstellen möglich. Globale Zustände, wie beispielsweise die Information über die aktuell aktive Komponente, finden im dynamischen Modell in Kapitel 5 Berücksichtigung.

Wir wollen dies nun formalisieren. Ist M eine Menge von Komponenten, die in einem Softwaresystem Verwendung finden sollen, so setzen wir $G_V := M$. Damit sind die Knoten des Konfigurationsgraphen (G_V, G_E, src, tar) bestimmt.
Eine Menge G_E von Konnektoren soll die Komponenten miteinander verbinden. Diese sind durch eine Abbildung $\tau : G_E \to |\mathcal{C}| \times |\mathcal{C}|$ typisiert. Wir formalisieren die obigen Forderungen wie folgt:

1. Ist $e \in G_E$ mit $c_1 = src(e)$ und $c_2 = tar(e)$, so muss $\tau(e) \in c_1^{out} \cap c_2^{in}$ sein.
2. Für alle $v \in G_V$ und alle $s \in v^{out}$ existiert ein $e \in G_E$ mit $v = src(e)$ und $\tau(e) = s$.

Zusätzlich muss beachtet werden, dass der Graph G schlingenfrei und zusammenhängend ist.

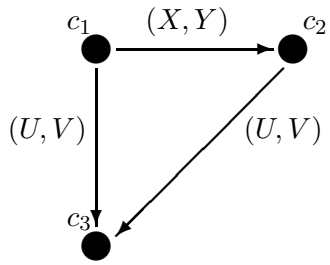
Beispiel 2.4 Drei Komponenten $c_i = (c_i^{in}, c_i^{out})$ $(i = 1, 2, 3)$ seien wie folgt gegeben:

$$\begin{aligned} c_1^{in} &= \{(X,Y)\}, & c_1^{out} &= \{(U,V),(X,Y)\}, \\ c_2^{in} &= \{(X,Y)\}, & c_2^{out} &= \{(U,V)\}, \\ c_3^{in} &= \{(U,V)\}, & c_3^{out} &= \emptyset, \end{aligned}$$

wobei $X, Y, U, V \in |\mathcal{C}|$ sind. Wir definieren auf $G_V = \{c_1, c_2, c_3\}$ die Konnektoren $G_E = \{e_1, e_2, e_3\}$ mit Typisierung

$$\tau(e_1) = (X,Y), \quad \tau(e_2) = (U,V), \quad \tau(e_3) = (U,V)$$

und legen $src, tar : G_E \to G_V$ so fest, dass wir den im folgenden dargestellten Konfigurationsgraphen G erhalten.



Komponente c_1 greift also über das Interface (U, V) auf c_3 zu, d.h. sie kann Aufrufe vom Typ U an c_3 senden und erhält Antworten vom Typ V. Man beachte, dass die Schnittstelle (U, V) von c_3 sowohl von c_1 als auch von c_2 angesprochen wird, während die von c_1 angebotene eingehende Schnittstelle (X, Y) von keiner Komponente genutzt wird.

Während jede ausgehende Schnittstellen durch einen Konnektor mit einer eingehenden Schnittstelle einer anderen Komponente verbunden sein muss, gilt dies nicht für eingehende Schnittstellen. Denn wie das Beispiel zeigt, wird nicht immer die gesamte von einer Komponente zur Verfügung gestellte Funktionalität von anderen Komponenten nachgefragt.
Dies umfasst jedoch nicht den Fall, dass Funktionalität ausgeblendet wird, wie es beispielsweise im Kontext von *JavaBeans* möglich ist, indem eine Schnittstelle in der Metainformation, die in der *BeanInfo*-Klasse abgelegt ist, auszulassen und damit versteckt wird [GT00, 5.1.5]. Verwendet man dieselbe *JavaBean*-Komponente mit verschiedenen *BeanInfo*-Klassen, so sieht sie für den Aufrufer unterschiedlich aus. In diesem Fall würden wir gemäß Definition 2.2 von zwei verschiedenen Komponenten sprechen, da sie unterschiedliche Mengen von Schnittstellen besitzen. Eine Komponente kann sich also nach außen nicht mit verschiedenen Schnittstellen präsentieren.

2.4 Eigenschaften von Konfigurationsgraphen

Ist ein Konfigurationsgraph (G, τ) gegeben, so wollen wir untersuchen, inwieweit die Menge seiner Komponenten sich in so partitionieren lässt, dass er als Schichtenarchitektur interpretiert werden kann. Dazu verwenden die relationalen Algebra, da diese sowohl zur Spezifikation von Eigenschaften, als auch zur konstruktiven Manipulation relationaler Strukturen geeignete Methoden zur Verfügung stellt[5].
Mit der Setzung $W := G_V$, $R := \{ \langle src(e), tar(e) \rangle \mid e \in G_E \}$ erhält man eine relationale Struktur, die wir im folgenden untersuchen wollen. Beim Übergang von einem Graphen zu seiner relationalen Struktur verlieren wir die Information über Mehrfachkanten zwischen denselben Knoten. Für die Frage, ob ein Konfigurationsgraph die Struktur eines Schichtensystems besitzt, ist das jedoch unproblematisch.

[5] Die wichtigsten Definitionen zur Relationalen Algebra findet der Leser in Anhang A.2.

Zwei Eigenschaften, die wir bereits allgemein für Konfigurationsgraphen gefordert haben, lassen sich kompakt in der relationalen Algebra formulieren. Schlingenfreiheit wird durch die Formel $R \subseteq \bar{\mathbb{I}}$ ausgedrückt, und der Forderung, dass der Konfigurationsgraph (schwach) zusammenhängend sein soll, entspricht die Formel $(R \cup R^T) = \mathbb{L}$. Im folgenden soll untersucht werden, welche Eigenschaften den Konfigurationsgraphen einer Schichtenarchitektur ausmachen. Wir halten folgende Charakterika von Schichtenarchitekturen fest:

C1 Es gibt eine endliche Anzahl von Schichten. Diese sind linear geordnet.

C2 Jede Schicht kommuniziert nur mit den unmittelbar benachbarten Schichten, und zwar stets von oben nach unten.

C3 Das System ist zusammenhängend, d.h. eine Schicht hat mindestens eine Schnittstelle zu jeder ihrer Nachbarschichten.

Um G zu einer Schichtenarchitektur zu machen, müssen wir die Komponenten also in Schichten einteilen. Dies geschieht durch eine Äquivalenzrelation E auf W, die wir so interpretieren, dass zwei Komponenten v, w genau dann in derselben Schicht liegen, wenn E_{vw} gilt. Eine Äquivalenzklasse $L \in W/E$ entspricht einer Schicht. Ziel ist es, eine lineare Ordnung auf der Menge der Äquivalenzklassen zu konstruieren. Nach Satz A.5 können wir die Schichten dann bijektiv und ordnungserhaltend auf ein Anfangsstück der natürlichen Zahlen abbilden, also „durchnumerieren“.
Für unser weiteres Vorgehen wird die Relation $S := R \cap \bar{E}$ eine wichtige Rolle speilen. Sie beinhaltet genau die Schnittstellen, die zwischen verschiedenen Schichten liegen; Schnittstellen innerhalb einer Schicht sind durch den Schnitt mit dem Komplement von E ausgeblendet. Aus der Struktur von S wollen wir die lineare Ordnung auf der Menge W/E der Äquivalenzklassen von E ableiten, dergestalt, dass eine Schicht $L \in W/E$ genau dann obere Nachbarschicht einer Schicht $L' \in W/E$ ist, wenn eine Schnittstelle von L nach L' existiert.
Im Hinblick auf die Konstruktion der Ordnung müssen wir einige Forderungen an S und E stellen.

- Damit die Schichten linear geordnen werden können (**C1**), darf kein Zyklus auftreten. Ein Zyklus in Bezug auf die Schichten ist dabei ein

geschlossener Weg, der nicht notwendig nur in S verläuft, sondern auch mit Elementen von E verknüpft werden kann, denn alle Komponenten innerhalb einer Schicht werden in der Äquivalenzklasse identifiziert. Dabei muss aber mindestens ein Schritt in S vorkommen. Daher verwenden wir die transitive Hülle $(E;S;E)^+$. Wegen $\mathbb{I} \subseteq E$ stellt die Komposition mit E keine Einschränkung dar. Zyklusfreiheit heißt, dass ein solcher Weg nicht in E liegen darf. Wir fordern daher:

(**S1**) $$(E;S;E)^+ \subseteq \bar{E}$$

- Schnittstellen dürfen nur von einer Schicht zur nächstunteren oder innerhalb einer Schicht existieren (**C2**). Keine Schicht darf übersprungen werden, indem z.B. eine Schicht eine direkte Schnittstelle zur übernächsten Schicht besitzt. Wir fordern, dass die Komposition eines S-Schrittes im Konfigurationsgraphen mit einem beliebigen Schritt, der nicht innerhalb einer Schicht, also nicht in E liegt, nicht in S liegen darf. In Kombination mit **S3** impliziert dies, dass keine Schicht übersprungen werden darf.

(**S2**) $$(S;\bar{E}) \cup (\bar{E};S) \subseteq \bar{S}$$

- Die Zusammenhangsforderung **C3** fassen wir wie folgt: Zwei beliebige Komponenten v, w müssen über einen Pfad in $S \cup E$ verbunden werden können. Da sie beliebig gewählt sind, kann $(S \cup E)^+_{vw}$ oder $((S \cup E)^+)^T_{vw}$ gelten. Daher ergibt sich:

(**S3**) $$(S \cup E)^+ \cup ((S \cup E)^+)^T = \mathbb{L}$$

Man beachte, dass wir zwischen den Schichten einen starken Zusammenhangbegriff verwenden, während die Komponenten eines Konfigurationsgraphen nur schwach zusammenhängend sein müssen.

Wenn S die drei Forderungen **(S1)** - **(S3)** erfüllt, wollen wir $K := (S \cup E)^+$ untersuchen und zeigen, dass K eine lineare Ordnung $\tilde{K}$ auf der Menge W/E der Äquivalenzklassen induziert, wenn man

$$\tilde{K}_{[x][y]} :\Leftrightarrow K_{xy}$$

setzt. Wir zeigen zunächst, dass $\tilde{K}$ auf wohldefiniert ist. Seien x, x', y, y' mit $E_{xx'}$ und $E_{yy'}$ gegeben. Dann gilt:

$$\begin{aligned} K_{xy} &\Leftrightarrow (S \cup E)^+_{xy} \\ &\Rightarrow (E; (S \cup E)^+; E)_{xy} \qquad (\text{da } I \subseteq E) \\ &\Leftrightarrow (S \cup E)^+_{x'y'} \\ &\Leftrightarrow K_{x'y'}. \end{aligned}$$

Nun halten wir weitere Eigenschaften fest, aus denen sich ergibt, dass $\tilde{K}$ eine lineare Ordnung auf L/E ist. K ist

- reflexiv: $E \subseteq K$
- antisymmetrisch: $K \cap K^T \subseteq E$
- transitiv: $K; K \subseteq K$ (da K als transitive Hülle definiert ist)
- linear: $K \cup K^T = \mathbb{L}$ (wegen **(S3)**)

Dies sind die Axiome für lineare Ordnung, wenn man E durch $\mathbb{I}$ ersetzt. Wir fassen also die Äquivalenzrelation E als verallgemeinerte Gleichheit auf. Da K die vier Axiome erfüllt, ergibt sich unmittelbar, dass die Relation $\tilde{K}$ eine lineare Ordnung auf der Menge der Äquivalenzklassen W/E ist.
Von den Axiomen ist allein die Antisymmetrie nicht trivial. Wir beweisen sie nun. Dabei nutzen wir aus, dass für E als Äquivalenzrelation $E^* = E^+ = E^T = E$ ist.

$$\begin{aligned} K \cap K^T &= \Big(S \cup E\Big)^+ \cap \Big((S \cup E)^+\Big)^T \\ &= \Big(E^+ \cup (E^* S)^+ E^*\Big) \cap \Big(E^+ \cup (E^* S)^+ E^*\Big)^T \qquad (\text{siehe Anhang}) \\ &= \Big(E \cup (ES)^+ E\Big) \cap \Big(E \cup ((ES)^+ E)^T\Big) \end{aligned}$$

$$\begin{aligned} &= E \cup \Big((ES)^+ E \cap ((ES)^+ E)^T \Big) \\ &\subseteq E \cup \Big((ESE)^+ \cap ((ESE)^+)^T \Big) \qquad (\text{da } E;E \subseteq E). \end{aligned}$$

Um die Antisymmetrie zu beweisen, reicht es also, $(ESE)^+ \cap ((ESE)^+)^T = O$ zu zeigen. Dazu formen wir (**S1**) um:

$$\begin{aligned} (ESE)^+ \subseteq \bar{E} &\Leftrightarrow (ESE)^+ \cap E = O \\ &\Rightarrow \Big((ESE)^+; (ESE)^+ \Big) \cap E = O \\ &\Leftrightarrow E((ESE)^+)^T \cap (ESE)^+ = O \qquad (\text{Zyklusregel}) \\ &\Leftrightarrow ((ESE)^+)^T \cap (ESE)^+ = O. \end{aligned}$$

Insgesamt haben wir damit $K \cap K^T \subseteq E$ gezeigt und den folgenden Satz bewiesen:

Satz 2.5 *Sei R eine beliebige Relation, E eine Äquivalenzrelation auf einer endlichen Menge W. Erfüllt eine Relation $S := R \cap E$ die Axiome **(S1)**, **(S2)** und **(S3)**, so ist durch $\tilde{K}_{[x][y]} :\Leftrightarrow K_{xy}$ eine lineare Ordnung auf der Menge W/E der Äquivalenzklassen definiert.* □

Die Ordnungsrelation $\tilde{K}$ ermöglicht es, die Äquivalenzklassen mit natürlichen Zahlen $1, \ldots, n$ zu kennzeichnen, wobei n die Kardinalität von W/E ist. Denn da mit W auch W/E endlich ist, ist $\tilde{K}$ sogar eine Wohlordnung. Wir haben also ein kleinstes Element C_0. Daher können wir die Äquivalenzklassen mit einer Abbildung $\bar{p} : W/E \to \{1, ..., n\}$ durchnumerieren. Induktiv definieren wir

$$\bar{p}(C_0) := 1,$$

$\bar{p}(C) := i$, falls $\bar{p}(D) = i - 1$ und C direkter Nachfolger von D ist.

Ist eine relationale Struktur (W, R) gegeben, die die Axiome $R \subseteq \bar{\mathbb{I}}$ und $(R \cup R^T) = \mathbb{L}$ für Schlingenfreiheit und Zusammenhang erfüllt, die also einem Konfigurationsgraphen zugrundeliegen kann, so fassen wir die Äquivalenzklassen als Schichten auf.
Aus $\bar{p}$ gewinnen wir eine Abbildung $p : W \to \{1, ..., n\}$ durch die Setzung $p(w) = \bar{p}([w]_E)$. Damit haben wir jeder Komponente w eine Schicht $p(w)$

zugeordnet. Wir interpretieren die n-te Schicht als die oberste (beispielsweise die Benutzerschnittstelle) und die erste Schicht als die unterste. Aufrufe sind stets nur innerhalb einer Schicht oder von einer Schicht in die direkt darunterliegende möglich.

2.5 Schichtensysteme

Während wir im letzten Abschnitt beliebige Konfigurationsgraphen auf die auf die Charakteristika von Schichtenarchitekturen hin untersucht haben, abstrahieren wir nun von diesem Entstehungsprozess und definieren unseren Grundbegriff, die Schichtenarchitektur, als eigenständigen Begriff.

Definition 2.6 *Sei (G, τ) ein Konfigurationsgraph mit $\mathcal{C}$-Interfaces, d.h. jeder (gerichteten) Kante $e \in G_E$ sei ein $\mathcal{C}$-Interface $\tau(e)$ zugeordnet. Für festes $n \in \mathbb{N}$ sei $p : G_V \to \{1, ..., n\}$ eine surjektive Abbildung, so dass für alle $e \in G_E$ die Bedingung*

$$(P) \quad 0 \leq p(src(e)) - p(tar(e)) \leq 1$$

erfüllt ist. Ein solches Tripel $\mathfrak{S} = (G, p, \tau)$ nennt man $\mathcal{C}$-Schichtensystem. *Der Graph G heißt der* dem Schichtensystem zugrundliegende Graph.

Die Abbildung p gibt an, welcher Schicht ein Element $v \in V$ angehört. Man beachte, dass jeder Graph eine derartige Abbildung p zulässt, wenn man $n = 1$ wählt. Durch Bedingung (P) wird sichergestellt, dass keine Konnektoren zwischen nicht benachbarten Schichten existieren. Wir nennen die Knoten $v \in G_V$ des Schichtensystems *Komponenten* und die Kanten $e \in G_E$ *Konnektoren* mit Schnittstelle $\tau(e)$. Über einen Konnektor e kann die Komponente $src(e)$ ein Aufruf an $tar(e)$ übermitteln. Die Antwort wird dann ebenfalls über e übermittelt. Es ist jedoch im allgemeinen nicht möglich, dass $tar(e)$ eigene Aufrufe an $src(e)$ übermittelt, es sei denn, es existiert eine weitere Kante e' mit $src(e') = tar(e)$ und $tar(e') = src(e)$.
Aus der Schichtenabbildung p ergibt sich wieder eine Äquivalenzrelation, wenn man für $v_1, v_2 \in V$

$$v_1 \sim v_2 :\Longleftrightarrow p(v_1) = p(v_2)$$

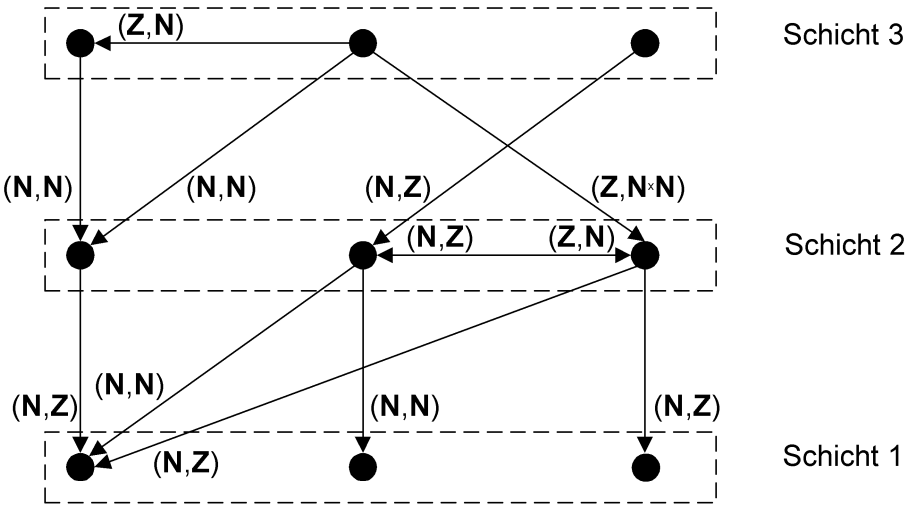


Abbildung 2.2: Schichtensystem in **Set**. An den Pfeilspitzen ist jeweils $(\tau_{\text{in}}(e),\ \tau_{\text{out}}(e))$ notiert.

setzt. Als *Schichten* definieren wir die Elemente von $V/\sim$. Man hat eine kanonische Bijektion

$$\bar{p} : V/\sim \longrightarrow \{1, ..., n\},$$

die eine lineare Ordnung auf den Schichten induziert. Wir bezeichnen

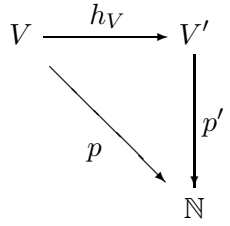
$$L_i := \bar{p}^{-1}(\{i\}) \quad (0 \leq i \leq n)$$

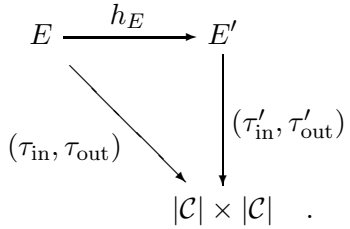
als die $i$te Schicht. Damit ist klar, dass die Begriffsbildung von Definition 2.6 äquivalent zu den im letzten Abschnitt mit relationalen Mitteln untersuchten Strukturen ist.
Durch verschiedene Abbildungen p_1, p_2 kann man verschiedene Schichteneinteilungen auf demselben Graphen G definieren. Sind Schichteneinteilungen p_1, p_2 gegeben, so sagen wir, dass p_2 eine *Verfeinerung* von p_1 ist, falls für alle $v, w \in V$ gilt: $p_2(v) = p_2(w) \Rightarrow p_1(v) = p_1(w)$.
Abbildung 2.2 zeigt den Graphen für das Beispielsystem aus der Einleitung. Als Kategorie wird $\mathcal{C} = \mathbf{Set}$ verwendet, und es kommen die Mengen der natürlichen und der ganzen Zahlen als Typen vor.
Sind $\mathfrak{S} = (G, p, \tau)$ und $\mathfrak{S}' = (G', p', \tau')$ zwei $\mathcal{C}$-Schichtensysteme mit m bzw. n Schichten, so definieren wir die Morphismen zwischen $\mathfrak{S}$ und $\mathfrak{S}'$ als die

Graph-Homomorphismen

$$h = (h_V, h_E) : G \to G',$$

die sowohl mit der Schichteneinteilung als auch mit der Typisierung verträglich sind, d.h. für die die folgenden Diagramme kommutieren:[6]





Morphismen müssen also eine Komponente auf eine Komponente der gleichen Schicht und Konnektoren auf Konnektoren der gleichen Typisierung abbilden. Auf diese Weise ist sichergestellt, dass Monomorphismen tatsächlich Einbettungen von Systemen sind, d.h. das eingebettete System sich mit der gleichen Typisierung im Bild wiederfindet. Wir erhalten die *Kategorie* $\mathcal{C}$**Syst** *der* $\mathcal{C}$*-Schichtensysteme*. Zwei Systeme $\mathfrak{S}$ und $\mathfrak{S}'$ sind genau dann isomorph, wenn die zugrundeliegenden Graphen isomorph sind und der Isomorphismus sowohl mit der Schichteneinteilung als auch der Typisierung verträglich ist. Als volle Unterkategorien definieren wir noch die Kategorien $\mathcal{C}\mathbf{Syst}_n$ der $\mathcal{C}$-Schichtensysteme mit genau n Schichten.

Ist $\mathfrak{S}$ ein beliebiges Schichtensystem, so zerfällt die Menge E_v der Konnektoren einer Komponente v in zwei disjunkte Teilmengen:

$$E_v^{in} := \{e \in E \mid tar(e) = v\},$$

[6] Paul Taylors Diagram Package für LaTeXwurde für diese und die meisten der folgenden Diagramme verwendet.

$$E_v^{out} := \{e \in E \mid src(e) = v\}.$$

Elemente von E_v^{in} sind die Kanten, welche in v hinein verlaufen, diejenigen von E_v^{out} verlaufen aus v heraus. Jene definieren die *eingehenden* Konnektoren, also solche, über die die Komponente v aufgerufen werden kann, diese die *ausgehenden*, über die sie andere Komponenten aufrufen kann. Da durchaus mehrere Konnektoren zwischen denselben Komponenten möglich sind, können i.a. auch parallele Kanten auftreten.
Drei Typen von Komponenten $v \in G_V$ können unterschieden werden:

- Anfangskomponenten: $E_v^{in} = \emptyset$ (reine Dienstnutzer)
- Endkomponenten: $E_v^{out} = \emptyset$ (reine Dienstleister)
- Innere Komponenten: $E_v^{in} \neq \emptyset \wedge E_v^{out} \neq \emptyset$ (Dienstnutzer und -leister)

Reine Dienstnutzer können nur in der obersten Schicht vorkommen. Die beiden anderen Typen von Komponenten sind dagegen in jeder beliebigen Schicht möglich. In Abbildung 2.2 gibt es beispielweise in der obersten Schicht eine innere Komponente. Anders sieht es aus, wenn man *strenge Schichtenarchitekturen* betrachtet, in denen Konnektoren nur von einer Schicht in die nächsttiefere, jedoch nicht innerhalb einer Schicht erlaubt sind. In diesen können innere Komponenten nicht in der obersten oder der untersten Schicht vorkommen.

Mit dem statischen Modell eines $\mathcal{C}$-Schichtensystems haben wir Konfiguration und Typisierung der an einem Softwaresystem beteiligten Komponenten erfasst. Im nächsten Kapitel werden wir Transformationen untersuchen, die mit diesen Schichtensystemen vorgenommen werden können.

Kapitel 3

Konstruktionen

In der Literatur zu Architekturen wird eine Vielfalt von Konstruktionsprinzipien zur ihrer Erstellung und Transformation diskutiert. Während auf der eher praxisorientierten Seite oft eine bunte Mischung von Operationen vorherrscht, die in einer Auflistung durchaus von der Komposition von Komponenten bis zu deren Auslieferung reichen kann [Tho98], ist es formalen Modellen naturgemäß eigen, sich auf bestimmte Aspekte zu beschränken. So konzentrieren sich die Modelle von Broy/Stølen [BS01] und Arbab/Rutten [AR02] auf die Modellierung zeitlicher Zusammenhänge. Andere Modellbildungen zielen auf die Untersuchung mobiler Programme [WF98] oder objektorientierte Datenstrukturen [Bar01]. Unser Ansatz stellt die freie Komposition unabhängiger Komponenten in den Vordergrund. Diese läßt sich – wie gesehen – vorteilhaft mit Mitteln der relationalen Algebra modellieren. Transformationen bereits zusammengestellter Komponenten umfassen neben der Verklebung von Architekturen vor allem Verfeinerungs- und Vergröberungsoperationen. Bei Schichtenarchitekturen kann dies sowohl die Verfeinerung der Schichteneinteilung als auch die Verfeinerung einer Komponente durch ein aus mehreren Komponenten bestehendes Teilsystem bedeuten. Die Verfeinerung der Schichteneinteilung eines Systems (G, p, τ) ist nichts anderes als die Zuweisung einer neuen Schichtenfunktion p'. Wir sind darauf bereits in Abschnitt 2.5 eingegangen.
In diesem Kapitel diskutieren wir zunächst die üblichen universellen Konstruktionen in der Kategorie $\mathcal{C}\mathbf{Syst}$. Darunter fallen Produkte, Coprodukte, Pullbacks, Pushouts, initiale und terminale Objekte. Diese können zur Definition komplexer Operationen verwendet werden, haben aber zum Teil auch

eigenständige fachliche Bedeutung. So ist das Pushout zweier Schichtensysteme nichts anderes als die *Verklebung* derselben. In Abschnitt 3.2 untersuchen wir, unter welchen Voraussetzungen das Konzept der Doppel-Pushout-Graphtransformation auf Schichtensysteme übertragbar ist. Danach zeigen wir in Abschnitt 3.3, wie sich diese Methode auf die architekturspezifischen Konstruktionen *Verfeinerung* und *Vergröberung* anwenden lässt. Außerdem erklären wir die *Verschmelzung* mehrerer Komponenten zu einer neuen. Zwei weitere Konstruktionen, bei denen das dynamische Modell mit herangezogen wird, werden wir schließlich in Abschnitt 5.9 diskutieren.

3.1 Universelle Konstruktionen

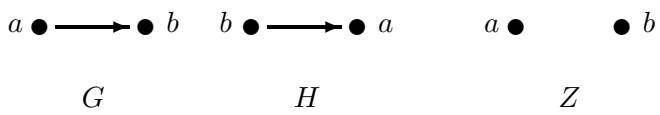
Vom kategoriellen Standpunkt aus ist ein Schichtensystem ein spezieller markierter Graph, nämlich mit einer Knotenmarkierungsfunktion p mit Bild in den natürlichen Zahlen und einer Kantenmarkierungsfunktion τ mit Bild in $|\mathcal{C}| \times |\mathcal{C}|$. Außerdem ist gefordert, dass ein Schichtensystem (schwach) zusammenhängend und schlingenfrei ist, und dass eine Kante von einer Schicht nur in dieselbe oder die nächstniedrigere Schicht führen darf.
$\mathcal{C}$**Syst**-Morphismen sind die mit der Markierungsfunktion verträglichen Graphhomomorphismen. Daher ist $\mathcal{C}$**Syst** eine volle Subkategorie der Kategorie der markierten Graphen. Limiten und Colimiten wie Produkte, Coprodukte, Pullbacks, Pushouts[1] u.a. lassen sich zurückführen auf die analogen Konstruktionen in der Kategorie der markierten Graphen. Allerdings muss stets geprüft werden, ob die Limes-Objekte auch in $\mathcal{C}$**Syst** existieren.

Produkte und Coprodukte

Die Kategorie der markierten Graphen kann man als Kategorie **Grph** $\downarrow L$ der **Grph**-Objekte über L ansehen[2], wobei L ein Graph ist, dessen Knoten die Marken repräsentieren. Damit beliebige Markierungen möglich sind, wählt man für L in der Regel einen vollständigen Graphen. Viele universelle Konstruktionen lassen sich von einer Kategorie $\mathcal{C}$ in die Kategorie der $\mathcal{C}$-Objekte über L übertragen. Nach [BW99, 13.4.4] entspricht das Produkt in $\mathcal{C} \downarrow L$ einem Pullback in $\mathcal{C}$. Konstruktiv bedeutet dies, dass man zunächst

[1] Definitionen dieser kategorientheoretischen Begriffe finden sich in Anhang B.
[2] Dies ist ein Spezialfall der Komma-Kategorie [Mac97, II.6.], vgl. Anhang B

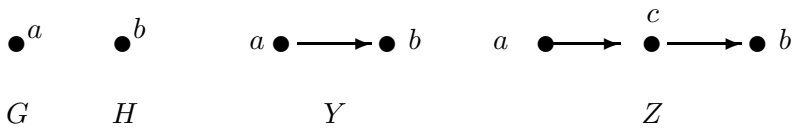
den Produktgraphen bildet und aus diesem genau diejenigen Knoten behält, die aus Knoten mit derselben Markierung hervorgegangen sind.
In $\mathcal{C}$**Syst** verlangen wir, dass der einem Schichtensystem zugundeliegende Graph zusammenhängend ist. Dies ist jedoch nicht bei allen Produkten markierter Graphen der Fall, so dass dort ein Produkt nicht immer existiert. Wir betrachten beispielhaft folgende Graphen:



Das Produkt der Graphen G und H ist in der Kategorie der markierten Graphen durch den Graphen Z gegeben. Betrachtet man G und H als Graphen, die Schichtensystemen $\mathfrak{G}, \mathfrak{H}$ zugrundeliegen, so kann Y nicht der dem Produkt $\mathfrak{G} \times \mathfrak{H}$ zugrundliegende Graph sein, da die Zusammenhangseigenschaft nicht erfüllt ist. Es gibt auch kein anderes Produktobjekt P:[3] Es ist klar, dass P mindestens einen Knoten, der mit a und einen, der mit b markiert ist, enthalten muss, denn es gibt Morphismen $Z \to G$ und $Z \to H$, also muss es nach der universellen Eigenschaft des Produktes auch einen Morphismus $Z \to P$ geben. Da P nicht zusammenhängend ist, muss noch mindestens eine Kante hinzukommen, es könnte also $P = G$ oder $P = H$ sein. Dies ist jedoch nicht möglich, da es keine Morphismen $G \to H$ oder $H \to G$ gibt, mithin keine Projektionen des Produkts existieren würden. Ebensowenig kann P eine bidirektionale Kante enthalten. Also existiert kein Produkt von G und H in $\mathcal{C}$**Syst**. Produkte von Architekturen haben jedoch, im Gegensatz beispielsweise zu Produkten von Automaten o.ä., keine herausragende fachliche Bedeutung in der Softwaretechnik. Denn die Knoten des Produktsgraphen setzen sich aus Paaren von Knoten der Ausgangsgraphen zusammen, man würde also alle Paare aus den Komponenten zweier Schichtensysteme betrachten, was man in der Praxis nie tut. Für die Existenz des

[3] Dass es kein anderes Produktobjekt gibt, muss man explizit überprüfen, da Limiten in einer Unterkategorie $\mathcal{D}$ einer Kategorie $\mathcal{C}$ durchaus anders aussehen können als in $\mathcal{C}$. Um sicherzugehen, dass der Einbettungsfunktor $\mathcal{D} \to \mathcal{C}$ Limiten erhält, muss man i.a. voraussetzen, dass $\mathcal{D}$ reflektive Unterkategorie von $\mathcal{C}$ ist [HS73, Sec. 36]. Dies ist jedoch hier nicht der Fall, da der Einbettungsfunktor von der Kategorie der Schichtensystem in die Kategorie der markierten Graphen keinen Linksadjungierten besitzt.

Produkts könnte man zwar als hinreichende Bedingung fordern, dass in G genau dann zwischen einem mit a und einem mit b markierten Knoten eine Kante existiert, wenn dies auch in H der Fall ist, aber diese Einschränkung ist sehr stark und macht den resultierenden Begriff für praktische Probleme wenig brauchbar. Die weiteren Axiome eines Schichtensystems sind im übrigen unproblematisch, d.h. falls die Zusammenhangsforderung nicht verletzt ist, existiert das Produkt: Im Produktgraph existiert nämlich genau dann eine Kante, wenn in beiden Faktoren Kanten vorkommen. Daher ist die Bedingung $0 \leq p(src(e)) - p(tar(e)) \leq 1$ stets erfüllt. Dasselbe gilt für Schlingen. Mithin steht und fällt die Existenz eines Produktsystems mit der Zusammenhangseigenschaft.
Auch Coprodukte existieren i.a. nicht. Sie entsprechen in der Kategorie der markierten Graphen einer disjunkten Vereinigung; diese ist jedoch nicht zusammenhängend. Dass keine andere Konstruktion ein Coprodukt liefert, überlegt man sich wie folgt: Es soll das Coprodukt-Objekt $G+H$ der Graphen G und H bestimmt werden, die den Schichtensystemen $\mathfrak{G}, \mathfrak{H}$ zugrundeliegen.

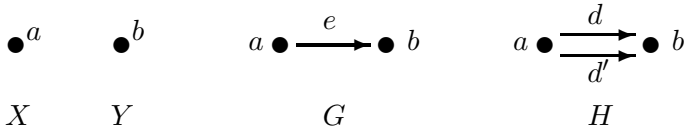


Angenommen, $G + H$ enthält nur Knoten, die mit a oder b markiert sind. Damit die Coprodukt-Inklusionen $G \to G + H$ und $H \to G + H$ existieren, muss in $G + H$ mindestens je ein mit a bzw. b markierter Knoten existieren. Da $G + H$ zusammenhängend ist, muss mindestens eine Kante zwischen einem mit a und einem mit b markierten Knoten existieren. Dann gibt es jedoch keinen Morphismus $G + H \to Z$, obwohl offensichtlich Morphismen $G \to Z$ und $H \to Z$ möglich sind.
Angenommen also, $G + H$ enthält auch Knoten, die mit einer Marke $c \neq a, b$ markiert sind. Dann gibt es jedoch keinen Morphismus $G + H \to Y$, obwohl Morphismen $G \to Y$ und $H \to Y$ existieren. Also gibt es kein Coprodukt $G + H$.
Wie in der Kategorie der Graphen ist auch in der Kategorie der markierten Graphen durch den leeren Graphen ein initiales Objekt gegeben. Dies überträgt sich nach $\mathcal{C}$**Syst**. Terminale Objekte existieren hingegen in der Regel nicht. Sie sind in der Kategorie der markierten Graphen durch den

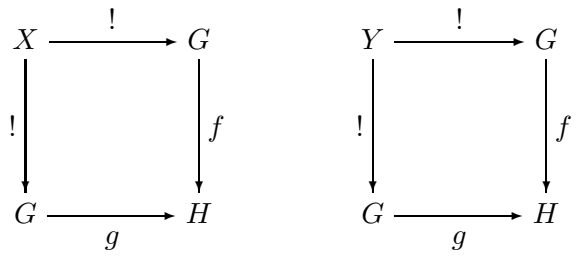
Markierungsgraphen L selbst gegeben. Für diesen wählt man in der Regel den vollständigen Graphen, dessen Knoten alle möglichen Markierungen repräsentieren. Dieser stellt jedoch i.a. kein Schichtensystem dar, da er nicht schlingenfrei ist und unzulässige, Schichten-überspringende Kanten besitzt. Es kann i.a. auch keinen anderen Graphen geben, der einem terminalen Schichtensystem zugrundeliegt, da es bei dieser bei unendlicher Markenmenge nicht endlich sein kann.

Pullbacks und Pushouts

Das Pullback einer Cospanne $A \xrightarrow{f} C \xleftarrow{g} B$ von markierten Graphen ist durch folgende Konstruktion gegeben [EHL+99]: Man bildet den Teilgraphen P des Produktgraphen, der aus denjenigen Knoten und Kanten besteht, die von f und g auf das gleiche Element abgebildet werden. Die Einschränkungen der Produktprojektionen auf P sind die Pullback-Morphismen. In der Kategorie $\mathcal{C}$**Syst** existiert das Pullback jedoch nicht immer, da die Zusammenhangseigenschaft verloren gehen kann.



Als Beispiel definieren wir Graph-Homomorphismen $f, g : G \to H$ auf den Kanten durch $f(e) = d$, $g(e) = d'$; die Fortsetzung auf die Knoten ist offensichtlich eindeutig. Wir betrachten das Pullback der Cospanne $G \xrightarrow{f} H \xleftarrow{g} G$. In der Kategorie der markierten Graphen ist das Pullbackobjekt der diskrete Graph mit zwei Knoten. In $\mathcal{C}$**Syst** existiert kein Pullback. Dass dieses auch durch kein anderes Objekt gegeben ist, zeigen wir nun. Angenommen, es existiert ein Pullback-Objekt P. Da die beiden Vierecke



kommutieren, muss es nach der Pullback-Eigenschaft Morphismen $X \to P$ und $Y \to P$ geben. Also muss P mindestens zwei Knoten enthalten, einen mit a und einen mit b markierten. Da P zusammenhängend ist, muss in P mindestens eine Kante existieren. Dies ist jedoch nicht möglich, da ein Graph, der eine Kante enthält, sich niemals mit f und g zu einem kommutativen Diagramm ergänzen lässt.
Beliebige Pushouts existieren in $\mathcal{C}$**Syst** nicht, allerdings läßt sich hier eine einfache und inhaltlich sinnvolle Bedingung für ihre Existenz formulieren. Gegenbeispiel für die allgemeine Existenz ist das Pushout zweier Einbettungen des initialen Objekts, denn dies ist nichts anderes als das Coprodukt, das, wie bereits gesehen, nicht existiert. Fordert man jedoch, dass in einer Spanne $A \xleftarrow{f} C \xrightarrow{g} B$ der Graph C nicht leer ist, tritt diese Problem nicht auf und Pushouts entstehen auf die gleiche Art und Weise wie in der Kategorie der markierten Graphen: Man bildet die disjunkte Vereinigung und identifiziert diejenigen Knoten und Kanten miteinander, die gemeinsames Bild eines Elements sind. Man bezeichnet das Pushout deshalb auch als *Verklebung*.
Wir geben diese wichtige Konstruktion explizit an: Seien $\mathfrak{S}_i = (G_i, p_i, \tau_i)$ für $i = 1, 2$ $\mathcal{C}$-Schichtensysteme, die entlang einer Menge von Knoten und Kanten der zugrundeliegenden Graphen G_1 und G_2 verklebt werden sollen. Eine Spanne

$$\mathfrak{S}_1 \xleftarrow{f} \mathfrak{A} \xrightarrow{g} \mathfrak{S}_2$$

von Morphismen zwischen einem $\mathcal{C}$-Schichtensystem $\mathfrak{A} = (A, p_A, \tau_A)$ und den Systemen $\mathfrak{S}_i$ induziert auf der disjunkten Vereinigung $G_1 + G_2$ eine kleinste Äquivalenzrelation $\approx = \langle \approx_V, \approx_E \rangle$, so dass für alle $w \in A_V$ und alle $d \in A_E$ gilt:

$$\begin{aligned} f_V(w) &\approx_V g_V(w), \\ f_E(d) &\approx_E g_E(d). \end{aligned}$$

Man identifiziert also zunächst alle Knoten bzw. Kanten miteinander, die gemeinsames Bild eines Elements aus A sind, und geht dann zu der kleinsten Äquivalenzrelation über, die diese Identifikationen umfasst. Wegen $A \neq \emptyset$ ist auch $\approx \neq \emptyset$. Unter diesen Voraussetzungen definieren wir das verklebte System $\mathfrak{S}_1 +_{\mathfrak{A}} \mathfrak{S}_2$ als $(G_V / \approx_V, G_E / \approx_E, src, tar)$, wobei sich die Abbildungen src und tar durch die Projektion $G_V \to G_V / \approx_V$ eindeutig auf $G_V / \approx_V$

übertragen. Das gleiche gilt für p und τ. Wegen $\approx_V \neq \emptyset$ ist $\mathfrak{S}_1 +_{\mathfrak{A}} \mathfrak{S}_2$ zusammenhängend.

Faktorsysteme

Eine besonders wichtige Konstruktion ist die eines Faktorsystems bezüglich eines Morphismus $f : \mathfrak{S} \to \mathfrak{S}'$. Der Morphismus f induziert durch folgende Setzung eine Äquivalenzrelation $A = (\equiv_V, \equiv_E)$ auf den Knoten und Kanten von G:

$$v \equiv_V v' :\Leftrightarrow f_V(v) = f_V(v'),$$

$$e \equiv_E e' :\Leftrightarrow f_E(e) = f_E(e').$$

Für diese Äquivalenzrelation gilt stets $\equiv_V \subseteq \sim$ (vgl. Abschnitt 2.5), denn $\mathcal{C}$**Syst**-Morphismen müssen stets die Schichteneinteilung erhalten. Ebenso ist A mit der Typisierung τ verträglich. Zwei Kanten e und e' können nur dann äquivalent sein, wenn sie unter den src und tar-Funktionen äquivalente Bilder besitzen. Es gilt eine weitere wichtige Eigenschaft: Da f Morphismus in $\mathcal{C}$**Syst** ist und $\mathfrak{S}'$ schlingenfrei ist, kann $v \equiv_V v'$ nur dann gelten, wenn v und v' nicht durch eine Kante verbunden sind. Denn sonst müsste $\mathfrak{S}'$ eine Schlinge besitzen. Eine Äquivalenzrelation A auf den Knoten und Kanten von G mit den angeführten Eigenschaften bezeichnen wir als *verträglich*.

Dem Faktorsystem $\mathfrak{S}/f$ liegt der Graph $G/f := (G_V/A, G_E/A, src, tar)$ zugrunde, wobei sich die Abbildungen src und tar eindeutig auf G_E/A übertragen, indem man für eine Äquivalenzklasse $[e] \in G_E/A$

$$\begin{aligned} src([e]) &:= [src(e)], \\ tar([e]) &:= [tar(e)]. \end{aligned}$$

setzt. Dass dies wohldefiniert ist, liegt an der Verträglichkeit, denn explizit gilt:

$$\begin{aligned} [e] = [e'] &\Leftrightarrow f_E(e) = f_E(e') \\ &\Leftrightarrow f_V(src(e)) = f_V(src(e')) \\ &\Leftrightarrow [src(e)] = [src(e')] \end{aligned}$$

Für *tar* trifft dies analog zu. Auch Schichteneinteilung und Typisierung einer Äquivalenzklasse sind durch einen beliebigen Repräsentanten definierbar, da die entsprechenden Abbildungen auf Äquivalenzklassen konstant sind. Ist von vornherein eine verträgliche Äquivalenzrelation A auf G gegeben, so schreiben wir auch $\mathfrak{S}/A$ für die obige Konstruktion.

Die Untersuchung der Limiten und Colimiten der Kategorie $\mathcal{C}$**Syst** zeigt, dass vor allem Limiten (Produkte, Pullbacks) nur unter sehr einschränkenden Bedingungen existieren, während die Situation bei den Colimiten (initiale Objekte, Coprodukte, Pushouts) besser aussieht. Wenn man Pushouts als verallgemeinerte Coprodukte ansieht, so kann man sagen, dass sie genau dann existieren, wenn sie keine Coprodukte sind, also keine disjunkte Vereinigung darstellen, sondern eine Verklebung entlang gemeinsamer Komponenten. Dies entspricht genau der fachlichen Forderung, dass zwei Schichtensysteme, um zusammenarbeiten zu können, miteinander kommunizieren müssen.

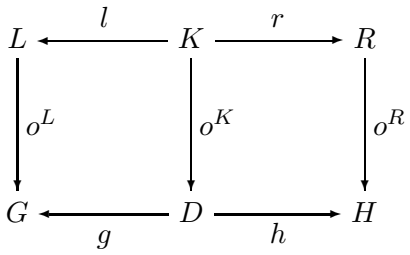
3.2 Doppel-Pushout-Transformationen

Graphtransformationen sind ein wichtiges technisches Werkzeug zur Definition und Rekonfiguration von Architekturen. So untersucht [Mét96], wie Architekturstile mithilfe von Graph-Grammatiken beschrieben werden können. Die Arbeiten [WF02, HT04] zeigen, dass die Rekonfiguration von Architekturen durch den Doppel-Pushout-Ansatz zur Graphtransformation, den wir im folgenden darstellen wollen, angemessen beschrieben werden kann. Insbesondere gilt dies auch für Verfeinerungen [HT04].
Da Pushouts in $\mathcal{C}$**Syst** unter vernünftigen Annahmen existieren, ist prinzipiell die Übertragung sowohl des Doppel- oder Einzel-Pushout-Transformationsansatzes [Roz97, 3.3, 4.2] von Graphen auf Schichtensysteme möglich. Wir wollen im folgenden den weiter verbreiteten Doppel-Pushout-Ansatz (DPO) verwenden und werden sehen, dass dieser an einigen Stellen vorteilhafter ist. In ihm wird eine Transformationsregel als eine Spanne

$$L \xleftarrow{l} K \xrightarrow{r} R$$

von Morphismen dargestellt. Um die Regel anzuwenden, muss die linke Regelseite L in ein zu transformierendes Objekt G injektiv eingebettet werden.

Das Ergebnis der Transformation wird durch den Graphen H in dem aus zwei Pushouts bestehenden Diagramm



repräsentiert. Abbildung 3.1 zeigt beispielhaft eine Regel, die zwischen zwei vorhandenen Knoten eine neue Kante einfügt. Die Graphen L und K sind diskret und besitzen zwei Knoten, der Graph R besteht aus zwei Knoten, die durch eine Kante verbunden sind. Die Beschriftung neben den Knoten legt fest, wie die Knoten a und b von den Graph-Homomorphismen l bzw. r abgebildet werden.

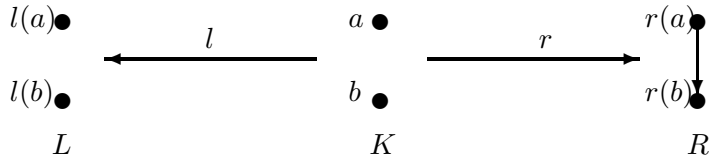


Abbildung 3.1: Transformationsregel, die eine neue Kante einfügt.

Die Anwendung einer Regel auf einen Graphen G setzt einen Graph-Homomorphismus $o^L : L \to G$ voraus, den man auch als *Ansatz* der Regel bezeichnet. Er gibt an, an welcher Stelle in G das Bild von L durch R ersetzt werden soll. Die Transformation geschieht nun in zwei Schritten: Zunächst wird der Graph D mit den Morphismen o^K und g so bestimmt, dass der linke Teil des obigen Diagramms ein Pushout wird. Man nennt D aus diesem Grund auch *Pushout-Komplement*. Die Existenz des Pushout-Komplements wird durch zwei Bedingungen sichergestellt: die *Identifikations-* und die *Kanten-Bedingung* (*identification-* und *dangling-edge-condition*) [Roz97, Prop. 3.3.4]. Erstere besagt, dass, falls o^L nicht injektiv ist, dasselbe Element (Knoten oder Kante) nicht gleichzeitig gelöscht und erhalten werden kann. Die zweite Bedingung verlangt, dass kein Knoten

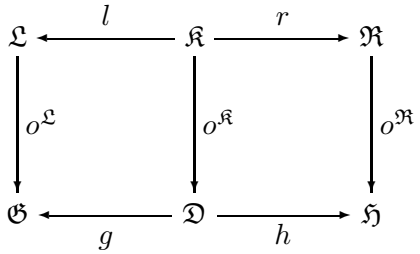
gelöscht werden darf, an dem eine Kante hängt, die nicht gelöscht werden soll. Fordert man zudem, dass der Morphismus l injektiv ist, so ist das Pushout-Komplement D bis auf Isomorphie eindeutig [ibd.]. Wir setzen im folgenden die Injektivität der Morphismen l stets voraus. Wenn die Identifikations- oder die Kanten-Bedingung nicht erfüllt ist, wollen wir sagen, dass die Regel mit dem fraglichen Ansatz nicht anwendbar ist. Der zweite Schritt besteht darin, den Graphen H als Pushoutobjekt der Spanne $D \stackrel{o^K}{\leftarrow} K \stackrel{r}{\rightarrow} R$ zu bestimmen. Der Graph H ist dann das Ergebnis der Anwendung der Regel $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ auf den Graphen G mit dem Ansatz o^L. Man schreibt $G \stackrel{p}{\Rightarrow} H$ und bezeichnet die Anwendung der Regel auch als *Produktion*.

Bemerkung 3.1 Knoten und Kanten, die nicht im Bild von o^L liegen, werden von der Transformation nicht tangiert. Man hat daher eine kanonische Einbettung $e_G^H : [G \backslash \mathrm{im}(o^L)] \rightarrow H$, wobei mit $[G \backslash \mathrm{im}(o^L)]$ der von $G_V \backslash \mathrm{im}(o_V^L)$ erzeugte Untergraph bezeichnet werde. Dies ermöglicht folgendes Vorgehen: Ist $m : L \rightarrow G$ ein Ansatz mit $\mathrm{im}(m) \cap \mathrm{im}(o^L) = \emptyset$, so lässt sich m zu einem Ansatz $e_G^H \circ m : L \rightarrow H$ umformen, d.h. dieselbe Regel ist nun auch auf das Ergebnis H der einmaligen Regelanwendung anwendbar. Da die Ansätze disjunkt sind, ist insbesondere die Bedingung für sequentielle Unabhängigkeit [Roz97, Prop. 3.4.2] erfüllt. Denn diese besagt, dass der Schnitt der Bilder der Ansätze in der Menge derjenigen Knoten enthalten sein muss, die von beiden Regeln erhalten bleiben. Da dieser Schnitt leer ist, ist die Bedingung trivial erfüllt. Damit ist sichergestellt, dass es nicht auf die Reihenfolge der Anwendung ankommt.
Wir werden dieses Vorgehen benötigen, um die Operationen *Verfeinerung* und *Vergröberung* explizit als mehrfache Anwendung von Graphtransformationsregeln darzustellen.

Der Doppel-Pushout-Ansatz ist auf rein kategorientheoretischer Ebene formulierbar [EHPP04]. Voraussetzung dafür ist, dass die fragliche Kategorie adhäsiv ist [LS04], was besondere Anforderungen an die Pushouts und Pullbacks impliziert. Da in der Kategorie $\mathcal{C}$**Syst** keine beliebigen Pushouts und Pullbacks existieren, ist sie nicht adhäsiv. Damit kann der Doppel-Pushout-Ansatz zur Graphtransformation nicht ohne Einschränkung übertragen werden. Dies ist wenig verwunderlich, denn die Doppel-Pushout-Graphersetzung ist ein allgemeines Konzept, das eine große Vielfalt von Transformationen unterstützt und daher auch disjunkte Vereinigungen, Einfügen von Schleifen

u.a. erlaubt. Neben der Identifikations- und der Kanten-Bedingung müssen wir also zusätzliche Forderungen stellen, um sicherzugehen, dass ein Transformationsschritt in $\mathcal{C}$**Syst** möglich ist. Denn eine Regel, die zwei Knoten miteinander verklebt, könnte einen schlingenfreien Graphen in einen nichtschlingenfreien transformieren. Dies müssen wir ausschließen.

Satz 3.2 *Sei* $\mathfrak{L} \xleftarrow{l} \mathfrak{K} \xrightarrow{r} \mathfrak{R}$ *eine Regel in* $\mathcal{C}$**Syst** *und* $o^{\mathfrak{L}} : \mathfrak{L} \to \mathfrak{G}$ *eine Einbettung der linken Regelseite in ein System* $\mathfrak{G}$*. In* $\mathcal{C}$**Syst** *existiert ein DPO-Diagramm*



wenn neben der Identifikations- und der Kanten-Bedingung folgende Bedingungen gelten:

(i) $\mathfrak{K}$ *ist nicht leer.*

(ii) Ist K *der* $\mathfrak{K}$ *zugrundeliegende Graph und sind* $v, w \in K_V$ *mit* $v \neq w$ *und* $r_V(v) = r_V(w)$*, so dürfen* $o^L_V(l_V(v))$ *und* $o^L_V(l_V(w))$ *nicht durch eine Kante verbunden sein.*

(iii) Sind $v, w \in K_V$ *mit* $v \neq w$ *und* $o^L_V(l_V(v)) = o^L_V(l_V(w))$*, so dürfen* $r_V(v)$ *und* $r_V(w)$ *nicht durch eine Kante verbunden sein.*

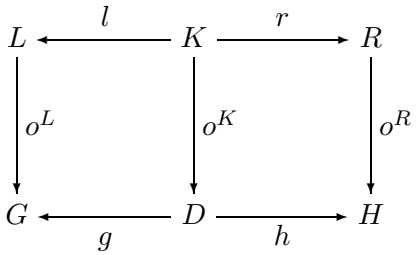
Beweis: Aufgrund der Identifikations- und der Kanten-Bedingung ist die Existenz des Doppel-Pushout-Diagramms in **Grph** sichergestellt. Damit dies auch in $\mathcal{C}$**Syst** der Fall ist, müssen wir nur prüfen, dass die Graphen Schichtensysteme darstellen, also zusammenhängend und schlingenfrei sind sowie die Bedingung $0 \leq p(src(e)) - p(tar(e)) \leq 1$ erfüllen.
Die Zusammenhangseigenschaft ist klar, da ein Pushout zusammenhängender Graphen genau dann zusammenhängend ist, wenn es von einem Paar nichtleerer Morphismen erzeugt wird. Dies wird durch Bedingung (i) sichergestellt.

Eine Schlinge kann auf zwei Weisen in einen schlingenfreien Graphen eingeführt werden: Entweder es wird eine Schlinge neu hinzugefügt, oder zwei miteinander verbundene Knoten werden verklebt, wodurch die Kante zwischen ihnen zu einer Schlinge wird.
Wir betrachten zunächst den ersten Fall. Da alle auftretenden Objekte $\mathcal{C}$**Syst**-Systeme sind, kann in L keine Schlinge vorkommen. Daher kann eine Schlinge nur dadurch neu hinzugefügt werden, dass eine Regel eine neue Kante einfügt, und dabei ein nicht-injektiver Ansatz verwendet wird. Dies ist durch Bedingung (iii) ausgeschlossen.
Der zweite Fall, dass nämlich zwei bereits durch eine Kante verbundene Knoten miteinander identifiziert werden, ist explizit durch Bedingung (ii) ausgeschlossen.
Es bleibt zu prüfen, dass keine Kanten eingefügt werden, die Komponenten aus verschiedenen, nicht direkt benachbarten Schichten verbinden. Dies wird durch den recht restriktiven Morphismenbegriff verhindert: Um eine neue Kante einzufügen, muss der Morphismus $r : \mathfrak{K} \to \mathfrak{R}$ zwei nichtverbundene Knoten v, w auf zwei durch eine Kante verbundene Knoten $r_V(v), r_V(w)$ abbilden. Dazu müssen jedoch v und $r(v)$ bzw. w und $r(w)$ dasselbe Bild unter der Schichtenabbildung p besitzen. Dies gilt ebenso für v und $l(v)$ bzw. w und $l(w)$. Da $\mathfrak{R}$ ein Schichtensystem ist, gilt dies mithin auch für das Ergebnis der Transformation. □

Der Beweis zeigt die Vorteile des gewählten Morphismenbegriffs. Da dieser sowohl die Schicht einer Komponente als auch die Typisierung erhalten muss, ist stets sichergestellt, dass die charakteristischen Eigenschaften von Schichtensystemen bei der Anwendung einer Transformationsregel erhalten bleiben. Würde man sich auf die den Systemen zugrundeliegenden Graphen beschränken und auf diese den Standard-Formalismus der Graphtransformation anwenden, so müsste man im Nachhinein überprüfen, ob das Resultat überhaupt ein Schichtensystem darstellt und festlegen, wie die Schichteneinteilung und die Typisierung definiert werden muss. Der gewählte Morphismenbegriff stellt hingegen stets eine mit der Graphstruktur verträgliche Schichteneinteilung sicher, d.h. es können keine Kanten zwischen nicht benachbarten Schichten auftreten.
Hiermit ist es möglich, innerhalb der Kategorie $\mathcal{C}$**Syst** den Doppel-Pushout-Ansatz anzuwenden. Allerdings zeigt sich, dass viele gängige Transformatio-

nen nicht formulierbar sind, da Schichtensysteme stets zusammenhängend sein müssen. Eine Regel, die zwischen zwei existierenden Knoten eine neue Kante einfügt, stellt man im DPO-Ansatz für gewöhnlich wie in Abbildung 3.1 gezeigt dar. Verwenden wir die Transformation ganz in der Kategorie der Schichtensysteme, so ist diese Regel nicht formulierbar[4]. Da wir auf die Operation, in einem beliebigen System eine neue Kante einzufügen, nicht verzichten möchten, müssen wir den DPO-Ansatz für $\mathcal{C}$**Syst** auf eine breitere Basis stellen. Hilfreich ist folgendes

Lemma 3.3 *Sei $L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ eine Regel in* **Grph***, wobei R ein zusammenhängender, nichtleerer Graph sei. Es existiere ein DPO-Diagramm*



Dann gilt: Falls G zusammenhängend und K nicht leer ist, ist auch H zusammenhängend.

Beweis: Wir zeigen, dass mit G auch H zusammenhängend ist. Seien x, y zwei beliebige Knoten in H. Falls beide in $o^R_V(R_V)$ liegen, existiert ein Weg zwischen ihnen, da R zusammenhängend ist. Wir nehmen o.E. an, dass $x \in H_V \backslash o^R_V(R_V)$ und $y \in o^R_V(R_V)$ ist. Da der Knoten x nicht im Bild von o^R_V liegt, muss er (wegen der Pushouteigenschaft) im Bild von h_V liegen. Also gibt es einen Knoten $d \in D_V$ mit $h_V(d) = x$. Nun ist $g_V(d) \in G_V \backslash o^L_V(L_V)$. Sei c ein beliebiger Knoten, der in $o^L_V(L_V)$ liegt. Ein solcher existiert, da mit K auch L nicht leer ist. Da G zusammenhängend ist, existiert ein Weg

$$g(d) = v^0, e^0, v^1, ..., e^{n-1}, v^n = c$$

[4]Man könnte das Einfügen einer neuen Kante durch eine unendliche Menge von Regeln darstellen, was jedoch die Handhabbarkeit des Graphtransformationssystems stark beeinträchtigen würde, da bereits die Prüfung, welche Regeln anwendbar sind, unangemessen aufwändig wäre. Wollte man dies dennoch tun, so müsste man alle zusammenhängenden Graphen auf der linken Regelseite durchlaufen und auf der rechten Seite jeweils „denselben“ Graphen mit einer zusätzlichen Kante verwenden. Für die im Beweis von Satz 3.2 erwähnten Fälle denke man an endliche Teilmengen der so entstehenden Regelmengen.

aus Knoten und Kanten in G. Wir bestimmen den minimalen Index i, so dass $\{v^0, v^1, ..., v^{i-1}, v^i\} \subseteq G_V \backslash o_V^L(L_V)$ und $v^{i+1} \in o_V^L(L_V)$ ist. Mittels der kanonischen Einbettung $e : [G\backslash \mathrm{im}(o^L)] \to H$ transportieren wir den Teilweg $v^0, ..., v^i$ nach H und erhalten $x = e(v^0), e(e^0), ..., e(v^i)$. Da v^i in G eine Kante zu einem Knoten in $o_V^L(L_V)$ besitzt, gibt es auch eine Kante von $e(v^i)$ zu einem Knoten a in $o_V^R(R_V)$. Da $o^R(R)$ zusammenhängend ist, gibt es ferner einen Weg von a nach y. Insgesamt haben wir so einen Weg von x nach y konstruiert.
Den Fall $x, y \in H_V \backslash o_V^R(R_V)$ kann man analog behandeln, indem man einen beliebigen Knoten z aus $o_V^R(R_V)$ auswählt und jeweils Wege von x und y zu z konstruiert. □
Das Lemma zeigt, dass nicht bei allen beteiligten Graphen Zusammenhang gefordert werden muss. Um die dadurch möglichen allgemeineren Transformationen durchführen zu können, definieren wir eine Oberkategorie von $\mathcal{C}$**Syst**, in der wir auf die Zusammenhangsforderung verzichten.

Definition 3.4 *Die Objekte der Kategorie* $\mathcal{C}$**GenSyst** *der allgemeinen* $\mathcal{C}$*-Schichtensysteme hat als Objekte Tripel* (G, p, τ)*, wobei* (G, τ) *ein Konfigurationsgraph mit* $\mathcal{C}$*-Interfaces und* $p : G_V \to \{1, ..., n\}$ *eine surjektive Schichtenabbildung ist. Die Morphismen zwischen zwei allgemeinen* $\mathcal{C}$*-Schichtensystemen* (G, p, τ) *und* (G', p', τ') *sind die Graph-Homomorphismen* $h : G \to G'$*, die sowohl mit der Schichteneinteilung* p *als auch mit der Typisierung* τ *verträglich sind.*

Mit dieser Definition wird $\mathcal{C}$**Syst** zu einer Unterkategorie von $\mathcal{C}$**GenSyst**. Wir können ein beliebiges $\mathcal{C}$-Schichtensystem als ein allgemeines $\mathcal{C}$-Schichtensystem auffassen, und umgekehrt jedes zusammenhängende allgemeine $\mathcal{C}$-Schichtensystem als $\mathcal{C}$-Schichtensystem. Für spätere Zwecke führen wir folgende Notation ein: Ist $\mathfrak{S} \in \mathcal{C}$**GenSyst**, so sei $\mathrm{Dis}(\mathfrak{S})$ die *Diskretisierung* von $\mathfrak{S}$, also das System, das durch Entfernung sämtlicher Kanten aus $\mathfrak{S}$ entsteht. Mit dieser Begrifflichkeit können wir nun die allgemeine Fassung der Doppel-Pushout-Transformation für $\mathcal{C}$-Schichtensysteme formulieren.

Satz 3.5 *Sei* $\mathfrak{L} \xleftarrow{l} \mathfrak{K} \xrightarrow{r} \mathfrak{R}$ *eine Regel in* $\mathcal{C}$**GenSyst** *und* $o^{\mathfrak{L}} : \mathfrak{L} \to \mathfrak{G}$ *eine Einbettung der linken Regelseite in ein System* G*. Die Identifikations- und die Kanten-Bedingung seien erfüllt. Weiter gelte:*

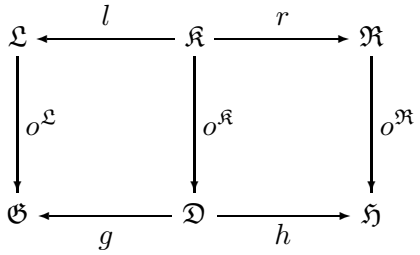
(i) $\mathfrak{K}$ *ist nicht leer.*

(ii) Ist K der $\mathfrak{K}$ zugrundeliegende Graph und sind $v, w \in K_V$ mit $v \neq w$ und $r_V(v) = r_V(w)$, so dürfen $o^L_V(l_V(v))$ und $o^L_V(l_V(w))$ nicht durch eine Kante verbunden sein.

(iii) Sind $v, w \in K_V$ mit $v \neq w$ und $o^L_V(l_V(v)) = o^L_V(l_V(w))$, so dürfen $r_V(v)$ und $r_V(w)$ nicht durch eine Kante verbunden sein.

(iv) Die Graphen G und R, die den Systemen $\mathfrak{G}$ bzw. $\mathfrak{R}$ zugrundeliegen, sind zusammenhängend.

Dann existiert genau ein DPO-Diagramm



*und es gilt: Der $\mathfrak{H}$ zugrundliegende Graph H ist zusammenhängend. Man kann daher $\mathfrak{H}$ als Objekt der Kategorie $\mathcal{C}$**Syst** auffassen.*

Beweis: Satz 3.2 und Lemma 3.3 □.

Damit ist das weite Feld der Anwendungsmöglichkeiten, das die Theorie der Graphtransformationen bietet, auch für Schichtenarchitekturen geöffnet, insbesondere auch was den Einsatz von Werkzeugen angeht [EEKR99]. Im nächsten Abschnitt werden wir Graphtransformationen einsetzen, um Konstruktionen auf Architekturen zu definieren.

3.3 Architekturspezifische Konstruktionen

In diesem Abschnitt diskutieren wir drei Konstruktionen, die spezifisch für Architekturen sind: die *Verfeinerung* und *Vergröberung* von Architekturen und die *Verschmelzung* von Komponenten. Für die ersten beiden Operationen verwenden wir den Doppel-Pushout-Ansatz, die letzte beschreiben wir direkt.

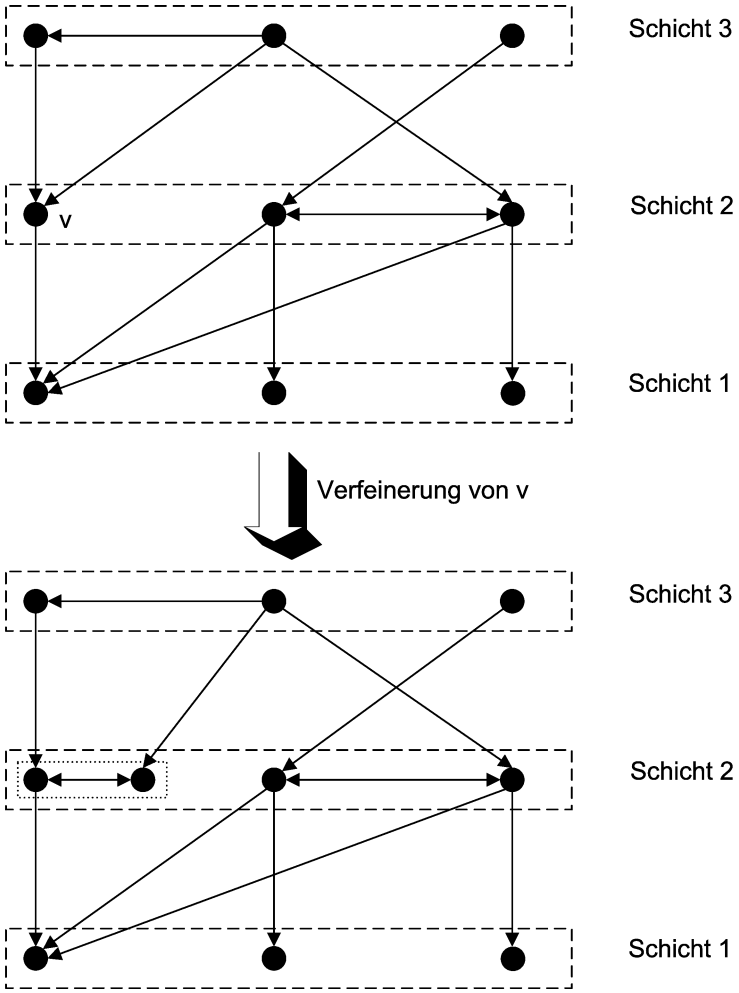
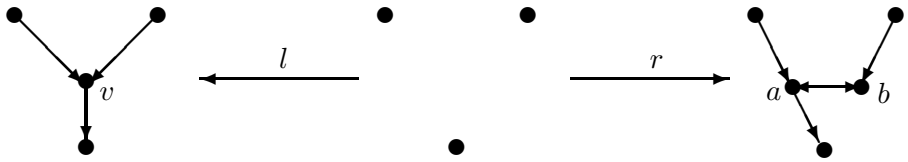


Abbildung 3.2: Verfeinerung des Knotens v

Knotenverfeinerung

Im Gegensatz zur Schichtenverfeinerung (vgl. Abschnitt 2.5) ist die Knotenverfeinerung eine Konstruktion, bei der die Anzahl der Komponenten vergrößert wird. Eine Komponente wird durch ein System von Komponenten, die durch Konnektoren untereinander verbunden sind, ersetzt. Ein Schichtensystem $\mathfrak{S}$ *verfeinert* ein System $\mathfrak{S}'$, falls es eine verträgliche Äquivalenzrelation A auf der Knotenmenge G_V von $\mathfrak{S}$ gibt, so dass $\mathfrak{S}/A$ isomorph zu $\mathfrak{S}'$ ist. Die Verträglichkeit impliziert insbesondere, dass eine Knotenverfeinerung nur innerhalb einer Schicht möglich ist. Will man die entstandenen Knoten verschiedenen Schichten zuordnen, so muss man ihr eine Schichtenverfeinerung folgen lassen. In der oben angegebenen Situation wollen wir gleichzeitig davon sprechen, dass $\mathfrak{S}'$ das System $\mathfrak{S}$ *vergröbert*.
Die Verfeinerung eines einzelnen Knotens v ist also die Ersetzung dieses Knotens durch ein Teilsystem T. Die Konnektoren, die mit v verbunden sind, müssen dabei mit Komponenten von T verbunden werden. Dazu muss festgelegt werden, welcher Konnektor mit welcher Komponente des neuen Teilsystems verbunden werden soll. Abbildung 3.2 zeigt diese Situation. Es reicht nicht, allein festzulegen, welche Komponente v durch T ersetzt werden soll, sondern die v umgebenden Komponenten müssen in die Transformationsregel mit einbezogen werden. Folgende Doppel-Pushout-Regel realisiert die Situation aus Abbildung 3.2.



Die Morphismen l und r seien dabei so definiert, wie es die geometrische Anordnung der Knoten, die in allen drei Graphen vorkommen, nahelegt. Es werde also der Knoten links oben aus dem mittleren Graphen durch l auf den Knoten links oben im linken Graphen abgebildet, etc. Diese Regel stellt die Ersetzung der Komponente v durch das aus den zwei Komponenten a, b bestehende Teilsystem dar. Man sieht, dass die v umgebenden Komponenten durch die Regel mit spezifiziert werden und diese festlegt, wie die mit

v verbundenen Konnektoren mit dem substituierten Teilsystem verbunden werden.
Die Doppel-Pushout-Graphtransformation eignet sich besonders gut für die Formulierung dieser Regel, da in ihr die Kanten-Bedingung sicherstellt, dass die Regel nicht anwendbar ist, wenn v mit einer größeren Zahl von Komponenten verbunden ist, als in der linken Regelseite spezifiziert. Denn da der Konten v gelöscht wird, darf das Bild von v im Graphen G – auf den die Regel angewandt wird – mit keinen Kanten verbunden sein, die nicht ebenfalls im Bild liegen. Somit ist stets sichergestellt, dass eine Regel dieser Form nur dann anwendbar ist, wenn das Bild von v nur mit höchstens so vielen Konnektoren verbunden ist, wie auf der linken Regelseite spezifiziert ist. Dies ist ein Vorteil gegenüber dem alternativen Single-Pushout-Ansatz, bei dem die Regel anwendbar wäre und problematische Kanten gelöscht würden [Roz97, 4.6].
Wir verallgemeinern diese Überlegungen in der folgenden Definition: Eine Ersetzungsregel $\mathfrak{L} \xleftarrow{l} \mathfrak{K} \xrightarrow{r} \mathfrak{R}$ ist ein *Verfeinerungsschritt*, falls gilt:

- $\mathfrak{L}$ ist ein System mit einer ausgezeichneten Komponente v, in dem jede Komponente über einen Konnektor direkt mit v verbunden ist.
- $\mathfrak{R}$ entsteht aus $\mathfrak{L}$ durch Ersetzung einer Komponente v durch ein (zusammenhängendes) Teilsystem $\mathfrak{T}$.
- $\mathfrak{K}$ ist das (möglicherweise diskrete) System, das aus $\mathfrak{L}$ entsteht, wenn die Komponente v und alle mit ihr verbundenen Konnektoren gelöscht werden.

Für gewöhnlich wird man $\mathfrak{L}$ so wählen, dass $\mathfrak{K}$ diskret wird. Wir fordern dies jedoch nicht explizit, da zusätzliche Konnektoren zur Formulierung differenzierterer Anwendungsbedingungen verwendet werden können.
Ist $\mathfrak{A} \subseteq \mathfrak{S}$ das von einem Teilgraphen $A \subseteq G$ erzeugte Teilsystem, so definieren wir den *Umgebungs-Abschluss* von $\mathfrak{A}$ als das Teilsystem $\mathfrak{A}^*$, das durch den Teilgraphen A^* gegeben ist, der neben A selbst alle mit einem Knoten von A verbundenen Knoten einschließlich der verbindenden Kanten enthält:

$$\begin{aligned} A^*_V &= A_V \cup \Big\{ v \in G_V \;\Big|\; \exists e \in G_E \; \exists w \in A_V \; \{src(e), tar(e)\} = \{v, w\} \Big\}, \\ A^*_E &= A_E \cup \Big\{ e \in G_E \;\Big|\; \{src(e), tar(e)\} \subseteq A_V \Big\}. \end{aligned}$$

Wir wollen nun skizzieren, wie sich eine Verfeinerung durch eine endliche Anzahl von Verfeinerungsschritten erreichen lässt. Sei $\mathfrak{S}$ ein Schichtensystem, das das System $\mathfrak{S}'$ verfeinert, d.h. es existiere eine verträgliche Äquivalenzrelation A auf dem zugrundeliegenden Graphen G von $\mathfrak{S}$, so dass $\mathfrak{S}/A$ isomorph zu $\mathfrak{S}'$ ist.
Sei $\{K_i \mid i = 1, ..., k\}$ die Menge der nichttrivialen Äquivalenzklassen in G_V/A. Wir betrachten die von diesen Äquivalenzklassen erzeugten Teilsysteme $\mathfrak{S}_i$ von $\mathfrak{S}$ und ihre Einbettungen $emb_i : \mathfrak{S}_i \to \mathfrak{S}$ in das Gesamtsystem. Jeder Äquivalenzklasse entspricht ein Verfeinerungschritt

$$\mathfrak{S}_i^*|\mathfrak{S}_i \longleftarrow [\mathfrak{S}_i^* \backslash \mathfrak{S}_i] \longrightarrow \mathfrak{S}_i^*.$$

Dabei sei $[\mathfrak{S}_i^* \backslash \mathfrak{S}_i]$ das von $\mathfrak{S}_i^* \backslash \mathfrak{S}_i$ erzeugte Teilsystem. Es werden also alle Knoten und Kanten, die in $\mathfrak{S}_i$ liegen, aus $\mathfrak{S}_i^*$ gelöscht und alle dabei eventuell entstehenden „hängenden Kanten“ entfernt. $[\mathfrak{S}_i^* \backslash \mathfrak{S}_i]$ repräsentiert daher die $\mathfrak{S}_i$ umgebenden Komponenten und Konnektoren in $\mathfrak{S}$.
Das System $\mathfrak{S}_i^*|\mathfrak{S}_i$ entstehe aus $\mathfrak{S}_i^*$, indem alle Komponenten und Konnektoren aus $\mathfrak{S}_i$ durch eine ausgezeichnete Komponente v ersetzt werden. Alle Konnektoren von $\mathfrak{S}_i$ zu anderen Komponenten werden dabei mit v verbunden.
Da $\mathfrak{S}_i$ eine Äquivalenzklasse bezüglich A ist, kann man $\mathfrak{S}_i^*|\mathfrak{S}_i$ in $\mathfrak{S}'$ einbetten. Diese Einbettung liefert den Ansatz für die Anwendung des Verefeinerungsschritts. Durch sequentielle Anwendung des Verfeinerungschritts für $i = 1, ..., k$ erhält man schließlich das Schichtensystem $\mathfrak{S}'$.

Vergröberung

Auch einzelne Vergröberungsschritte lassen sich als Graphtransformationsregeln definieren. Wir wollen zeigen, dass jede Vergröberung durch eine Hintereinanderausführung endlich vieler solcher Schritte erzielt werden kann.
Ein *Vergröberungsschritt* ist eine DPO-Regel der Form

$$\mathfrak{S} \xleftarrow{emb} \mathrm{Dis}(\mathfrak{S}) \xrightarrow{!} \mathfrak{I},$$

wobei mit $\mathfrak{I}$ das triviale System, das aus einer Komponente besteht, bezeichnet werde. Ferner sei $\mathfrak{S}$ ein Schichtensystem, das nur eine Schicht besitzt. In

der Mitte verwenden wir die Diskretisierung von $\mathfrak{S}$, um sicherzustellen, dass stets ein Morphismus $! : \mathrm{Dis}(\mathfrak{S}) \to \mathfrak{I}$ existiert. Eine Regel dieser Form ersetzt das Teilsystem $\mathfrak{S}$ durch eine einzige Komponente. Es ist klar, dass dies der Bildung eines Faktorsystems gleichkommt, bei dem alle Knoten, die im Bild von $\mathfrak{S}$ sind, in einer Äquivalenzklasse liegen und alle anderen Äquivalenzklassen einelementig sind. Folgendes Beispiel zeigt eine Regel, die ein aus zwei Komponenten bestehendes Teilsystem durch eine einzige Komponente ersetzt:

Im Gegensatz zur Verfeinerung ist es bei der Vergröberung nicht nötig, die umgebenden Komponenten in die Regel mit einzubeziehen, denn alle mit dem zu ersetzenden Teilsystem durch Konnektoren verbundenen Komponenten sind nach Anwendung der Regel mit der einzigen Komponente, die sie ersetzt, verbunden.
Sei A eine Äquivalenzrelation auf der Komponentenmenge G_V eines Systems $\mathfrak{S}$ und $\mathfrak{S}' = \mathfrak{S}/A$. Man kann $\mathfrak{S}'$ aus $\mathfrak{S}$ gewinnen, indem man so viele Vergröberungsschritte durchführt, wie es nicht-triviale Äquivalenzklassen in $\mathfrak{S}/A$ gibt. Sei $\{K_i \mid i = 1, ..., k\}$ die Menge der nichttrivialen Äquivalenzklassen in G_V. Wir betrachten die von diesen Äquivalenzklassen erzeugten Teilsysteme $\mathfrak{S}_i$ von $\mathfrak{S}$ und ihre Einbettungen $emb_i : \mathfrak{S}_i \to \mathfrak{S}$ in das Gesamtsystem. Man erhält Vergröberungschritte

$$\mathfrak{G}_i \xleftarrow{emb} \mathrm{Dis}(\mathfrak{G}_i) \xrightarrow{!} \mathfrak{I}$$

und kann als linke Ansätze die Einbettungen emb_i verwenden, die nach Bemerkung 3.1 sequentiell angewandt werden können. Führt man die Verfeinerungsschritte für $i = 1, ..., k$ an $\mathfrak{S}$ aus, so erhält man $\mathfrak{S}'$.

Verschmelzung von Komponenten

Neben der Vergröberung, die eine Operation auf der Ebene der Architekturen ist, ist auch die rein auf der konzeptionellen Ebene der Komponenten

angesiedelte Operation *Verschmelzung* von Bedeutung. Hierbei handelt es sich nicht um die Transformation eines bereits existierenden Schichtensystems, sondern um die Zusammenstellung mehrerer Komponenten zu einer neuen, die dann bei der Komposition von Systemen von vornherein als einzelne Komponente angesehen wird. Dies tritt in der Praxis insbesondere dann auf, wenn Komponenten ausgeliefert werden sollen. Während der Entwickler der Komponente die Funktionalität in seinem eigenen Entwicklungsprozess aus mehreren Teilkomponenten zusammensetzt, will der Nutzer der Komponente diese für seine Zwecke als eine Einheit verwenden. Daher muss vor der Auslieferung aus mehreren Komponenten eine neue Komponente erstellt werden.
Damit Komponenten miteinander verschmolzen werden können, müssen diese gemeinsame Schnittstellen besitzen. Der Prozess der Verschmelzung kann sukzessive geschehen, d.h. zunächst werden zwei Schnittstellen miteinander verschmolzen, dann kommen jeweils einzelne weitere hinzu. Daher formulieren wir diese Konstruktion nur für den Fall von zwei Schnittstellen. Damit Komponente c_1 mit einer anderen Komponente c_2 verbunden werden kann, muss c_1 eine ausgehende Schnittstelle $s = (s_1, s_2)$ und c_2 eine eingehende Schnittstelle $t = (s_2, s_1)$ besitzen.
Allgemein können zwei Komponenten $c_1 = (c_1^{in}, c_1^{out})$ und $c_2 = (c_2^{in}, c_2^{out})$ entlang von mehreren gemeinsamen Schnittstellen verbunden werden, wenn nichtleere linkseindeutige (injektive) Relationen $R : c_1^{in} \leftrightarrow c_2^{out}$ und $S : c_2^{in} \leftrightarrow c_1^{out}$ existieren, die angeben, welche Schnittstellen gekoppelt werden sollen. Diese Relationen müssen mit der Typisierung der Schnittstellen verträglich sein, d.h. es muss für alle $s = (s_1, s_2) \in c_1^{in}, t = (t_1, t_2) \in c_2^{out}, u = (u_1, u_2) \in c_2^{in}, v = (v_1, v_2) \in c_1^{out}$ gelten

$$R_{st} \Rightarrow s_1 = t_2 \wedge s_2 = t_1$$

und

$$S_{uv} \Rightarrow u_1 = v_2 \wedge u_2 = v_1.$$

Die verklebte Komponente $c = (c^{in}, c^{out}) = (c_1^{in}, c_1^{out}) \oplus (c_2^{in}, c_2^{out})$ ist wie folgt zusammengesetzt:

$$c^{in} = \{s \in c_1^{in} \mid \neg\exists t \in c_2^{out} : R_{st}\} \cup \{u \in c_2^{in} \mid \neg\exists v \in c_1^{out} : S_{uv}\},$$
$$c^{out} = \{v \in c_1^{out} \mid \neg\exists u \in c_2^{in} : S_{uv}\} \cup \{t \in c_2^{out} \mid \neg\exists s \in c_1^{in} : R_{st}\}.$$

Auf diese Weise ist eine partielle Verknüpfung $\oplus$ zwischen Komponenten definiert. Die wiederholte sequentielle Anwendung von $\oplus$ ermöglicht es, eine beliebige endliche Anzahl von Komponenten miteinander zu verschmelzen. Die entstandene Komponente sieht dann nach außen hin wie eine einzelne Komponente aus und kann als solche ausgeliefert werden.

Kapitel 4

Datentypen als (Co-)Produkte

Der Begriff des Datentyps wird in der Literatur meist als eine Menge von Werten definiert, die eine Variable annehmen kann oder die als Ein- bzw. Ausgabe einer Berechnung zur Verfügung stehen, zusammen mit einer Anzahl von Operationen, die auf diesen Werten definiert sind [LEW97, 1.2]. Formalisiert wird dieses Konzept mithilfe von Algebren zu einer Signatur $\Sigma = (S, OP)$ [EMC+99, 7.2-3]. Die Sorten einer Algebra konstituieren die Typen der Wertemenge, die Verknüpfungen geben die darauf möglichen Operationen an. Häufig ist man jedoch nicht an einer bestimmten Algebra interessiert, sondern möchte nur die für die Anwendung wesentlichen Eigenschaften spezifizieren. Dies führt dazu, statt konkreter Algebren Klassen von Algebren zu bestrachten, sogenannte abstrakte Datentypen [Wir90, LEW97], deren Axiome in der Regel durch Gleichungen definiert sind.
Kategorien als Basis für abstrakte Datentypen zu verwenden, ist u.a. von Wagner [Wag86] und von Walters [Wal91] vorgeschlagen worden. Man sieht in jedem Objekt eine Sorte und definiert die Operationen als Morphismen, wobei für mehrstellige Operationen Produkte herangezogen werden. Die Existenz von Produkten in der fraglichen Kategorie ist dann natürlich eine notwendige Voraussetzung. Axiome werden durch kommutative Diagramme anstatt durch Gleichungen formuliert [Poi86]. Dieses Vorgehen ist sehr flexibel, denn die Morphismenmenge kann leicht an die Bedürfnisse der Anwendung angepaßt werden. So betrachtet man die Kategorie der Graphen wahlweise mit partiellen oder totalen Morphismen [Roz97, 4.2.1, 3.3.1], oder kann sich auf die Klasse der Monomorphismen oder Teilklassen davon beschränken [EHPP04, Def. 3].

Beim Übergang von mengenbasierten Algebren zu Kategorien stößt man jedoch auf das Problem, dass die Objekte einer Kategorie nicht notwendigerweise Mengen sein müssen, und dass, selbst wenn eine Repräsentation als Menge vorliegt, die Morphismen keine Abbildungen zwischen diesen Mengen sein müssen. So ist jede partielle Ordnung eine Kategorie [BW99, 2.3.1], und man kann die Morphismen in der zu **Set** dualen Kategorie $\mathbf{Set}^{op}$ in der Regel nicht als Abbildungen interpretieren. Kategorientheorie ist im wesentlichen elementfrei formuliert, statt Gleichungen werden Diagramme verwendet. In Kategorien mit terminalen Objekten läßt sich der Begriff des *globalen Elements* [McL92, 1.4] als Ersatz heranziehen.
Der Ansatz, den wir folgenden vorstellen, setzt neben der Existenz von Produkten auch die von Coprodukten voraus. Die intuitive Vorstellung, dass ein Element eines Coprodukts aus zwei Objekten stets in genau einem der Objekte „liegen“ muss, stellt sich zwar im allgemeinen als falsch heraus, aber es lassen sich Voraussetzungen identifizieren, unter denen sie gültig ist. Wir bezeichnen diese Eigenschaft als *eindeutige Coprodukt-Faktorisierung* und bauen auf ihr in Kapitel 5 ein dynamisches Modell für Schichtensysteme auf.
Das vorliegende Kapitel ist wie folgt gegliedert: Im nächsten Abschnitt erklären wir die Idee zunächst am konkreten Beispiel der Kategorie der Mengen, und stellen dann im zweiten Abschnitt dieses Kapitels die Problematik dar, die sich in allgeminen Kategorien auftut. Danach führen wir die extensiven Kategorien ein und zeigen in den letzten beiden Abschnitten, dass in diesen die eindeutige Coprodukt-Faktorisierung gilt.

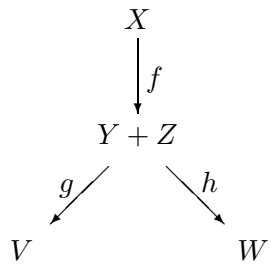
4.1 Mengen

In der Kategorie **Set** der Mengen und Abbildungen kann man mithilfe von Abbildungen, deren Wertebereich die disjunkte Summe zweier oder mehrerer Mengen ist, neben der Berechnung auch Kontrollflussaspekte modellieren. Sind beispielsweise drei Funktionen

$$\begin{aligned} f &: X \to Y + Z \\ g &: Y \to V \\ h &: Z \to W \end{aligned}$$

gegeben und hat man eine Eingabe $x \in X$, so kann man festlegen, dass der weitere Kontrollfluss eines Systems davon abhängt, ob $f(x) \in Y$ oder

$f(x) \in Z$ ist[1]. Das Ergebnis der durch f modellierten Berechnung, $f(x)$, wird im ersten Fall als Eingabe für g, im zweiten Fall für h verwendet. Es wird also entweder $g \circ f$ oder $h \circ f$ für die weitere Berechnung gebildet[2].
Bei diesem Vorgehen fließt folgende universelle Eigenschaft der disjunkten Vereinigung von Mengen ein: Für alle Eingaben $x \in X$ existiert stets entweder genau ein $y \in Y$ mit $f(x) = y$ oder genau ein $z \in Z$ mit $f(x) = z$. Man benötigt *Existenz* und *Eindeutigkeit* eines solchen Elements.



Verzweigung des Kontrollflusses

Die disjunkte Vereinigung ist das Coprodukt in der Kategorie der Mengen. Daher liegt es nahe, den vorgestellten Ansatzes auf allgemeine Kategorien durch eine Coprodukt-Konstruktion zu verallgemeinern. Es stellt sich jedoch heraus, dass nicht jede Kategorie ein Coprodukt besitzt und nicht immer das kategorientheoretische Analogon der obigen Existenz- und Eindeutigkeitsforderung erfüllt ist. Wir wollen daher in Kategorien arbeiten, die (a) Coprodukte besitzen und (b) in denen diese sich im obigen Sinne „wohlverhalten“.

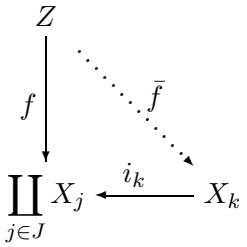
4.2 Allgemeine Kategorien

Wir wollen die Situation aus dem letzten Abschnitt kategoriell fassen. Sei J im folgenden eine endliche Indexmenge und $\mathcal{C}$ eine Kategorie mit endlichen Coprodukten.

[1]Mit $+$ bezeichnen wir die *Summe von Mengen*. Sind zwei Mengen A und B gegeben, so ist $A + B = (A \times \{1\}) \cup (B \times \{2\})$. Wir sagen, dass ein Element $x \in A + B$ entweder in A oder in B liegt und schreiben kurz $x \in A$, falls $x \in A \times \{1\}$, sonst analog $x \in B$. Mehrfache Summen von Mengen schreiben wir mit dem Summenzeichen $\sum$.

[2]Wir unterschlagen hier geflissentlich die kanonischen Injektionen $Y \to Y + Z$ und $Z \to Y + Z$ in der Notation.

Definition 4.1 *Sei $f : Z \to \coprod_{j\in J} X_j$ ein Morphismus. Wir sagen, dass f* durch den Cofaktor X_k faktorisiert*, falls es einen Morphismus $\bar{f} : Z \to X_k$ gibt, so dass folgendes Diagramm kommutiert:*



Falls sowohl der Cofaktor X_k als auch der Morphismus $\bar{f}$ eindeutig bestimmt sind, nennen wir die Faktorisierung eindeutig.

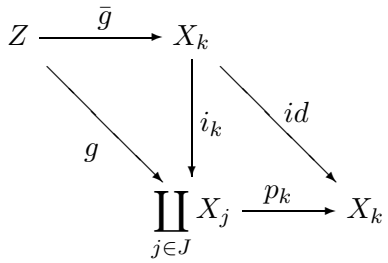
Um die Faktorisierung kompakter formulieren zu können, wollen wir sie als Fixpunkt einer natürlichen Transformation charakterisieren. Sei $\coprod_{j\in J} X_j$ ein Coprodukt nicht-leerer Elemente X_j, d.h. es existiere jeweils mindestens ein Morphismus $I \to X_j$ vom terminalen Objekt I nach X_j. Wir definieren für $k \in J$ die *Coprodukt-Projektion* $p_k : \coprod_{j\in J} X_j \to X_k$ durch folgende Vorschrift: Für $j \in J\backslash\{k\}$ sei $f_j : X_j \to X_k$ ein beliebiger Morphismus und $f_k = id_{X_k} : X_k \to X_k$ die Identität. Dann sei p_k der eindeutige vermittelnde Morphismus der f_j. Es folgt $p_k \circ i_k = id_{X_k}$, so dass p_k linksinvers zu i_k ist. Der Morphismus $\pi_k := i_k \circ p_k : \coprod_{j\in J} X_j \to \coprod_{j\in J} X_j$ induziert eine natürliche Transformation von dem kontravarianten Hom-Funktor in das Coprodukt [HS73, 13.2(9)]

$$\begin{array}{rcl} M_k : \mathrm{Hom}(-, \coprod_{j\in J} X_j) & \to & \mathrm{Hom}(-, \coprod_{j\in J} X_j) \\ g & \mapsto & \pi_k \circ g \end{array} \quad .$$

Wir bezeichen diesen Funktor im folgenden stets mit M_k. Es gilt folgende Charakterisierung:

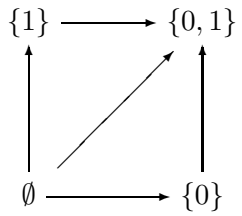
Satz 4.2 *Sei $\coprod_{j\in J} X_j$ das Coprodukt nicht-leerer Elemente X_j und $g : Z \to \coprod_{j\in J} X_j$ ein Morphismus. Dann faktorisiert g genau dann durch X_k, falls g ein Fixpunkt von M_k ist. Der Faktor ist dann $\overline{g} := p_k \circ g$.*

Beweis: Angenommen, g faktorisiert durch X_k via $\bar{g} : Z \to X_k$. Wie das Diagramm



zeigt, gilt $\bar{g} = p_k \circ g$, somit ist $\pi_k \circ g = i_k \circ p_k \circ g = i_k \circ \bar{g} = g$. Die andere Richtung ist trivial.□

Wir untersuchen einige pathologische Fälle, um die Notwendigkeit zusätzlicher Annahmen aufzuzeigen. Betrachten wir beispielsweise die Kategorie $\mathfrak{P}(\{0,1\})$. Ihre Objekte sind die Elemente der Potenzmenge von $\{0,1\}$ und ihre Morphismen die Inklusionen zwischen den Teilmengen. Es existiert also genau dann ein Morphismus $X \to Y$, falls $X \subseteq Y$ ist. Graphisch können wir uns $\mathfrak{P}(\{0,1\})$ durch folgendes Diagramm veranschaulicht denken:



Produkt und Coprodukt sind in $\mathfrak{P}(\{0,1\})$ auf besonders einfache Weise gegeben: Das Produkt zweier Teilmengen ist ihr Schnitt, das Coprodukt ihre Vereinigung (*nicht:* disjunkte Vereinigung!). Außerdem gilt: $\emptyset$ ist initiales, $\{0,1\}$ terminales Objekt.
Da das Coprodukt nicht disjunkt ist, kann die oben geforderte Eindeutigkeit nicht gewährleistet werden. Folgende Cospannen stellen nämlich Paare von Coprodukt-Inklusionen dar:

$$\begin{aligned} \{0,1\} &\longrightarrow \{0,1\} \longleftarrow \{0,1\} \\ \{0\} &\longrightarrow \{0,1\} \longleftarrow \{1\} \\ \{0,1\} &\longrightarrow \{0,1\} \longleftarrow \{1\} \end{aligned}$$

Im ersten Fall ist das terminale Objekt $\{0,1\}$ Coprodukt zweier Kopien seiner selbst, wobei die beiden Coprojektionen identisch sind. Der zweite Fall bietet das gewohnte Bild, da die beiden Cofaktoren[3] des Coprodukts disjunkt sind. Im dritten Beispiel liegt eine Mischung aus den ersten beiden Fällen vor. Hier ist interessant, dass das kleinere der beiden Objekte ($\{1\} \subseteq \{0,1\}$) zum Coprodukt-Objekt gar nichts beiträgt – es könnte genausogut durch das initiale Objekt $\emptyset$ ersetzt werden.
Ein weiteres Beispiel zeigt, dass auch die Existenzbedingung nicht in allen Kategorien erfüllt ist. Wir betrachten dazu die Produkt-Kategorie $\mathbf{Set}^2$, deren Objekte Paare $\langle A, B\rangle$ von Mengen sind. Ein Morphismus $\langle A, B\rangle \to \langle C, D\rangle$ ist ein Paar $\langle f, g\rangle$ von Abbildungen $f : A \to C$ und $g : B \to D$. Wir bezeichnen mit O und I das initale bzw. terminale Objekt in **Set**. Dann ist in folgendem Diagramm

$$\begin{array}{ccccc} & & < I, I > & & \\ & & \downarrow{\scriptstyle id_{<I,I>}} & & \\ < O, I > & \xrightarrow[! \times id_I]{} & < I, I > & \xleftarrow[id_I \times !]{} & < I, O > \end{array}$$

die untere Zeile ein Paar von Coprodukt-Inklusionen[4]. Dennoch existieren keine Morphismen $< I, I > \to < I, O >$ oder $< I, I > \to < O, I >$, da es in **Set** keine Morphismen $I \to O$ gibt. Der Morphismus $id :< I, I > \to < I, I >$ hat also als Ziel ein Coprodukt, faktorisiert jedoch durch keinen der Cofaktoren. Coprodukte in beliebigen Kategorien unterscheiden sich also in ihren Eigenschaften bisweilen sehr stark vom Prototyp **Set**. Ein weiteres Beispiel hierfür ist die Kategorie der kommutativen Ringe, in der das Coprodukt durch das Tensorprodukt gegeben ist. So ist z.B. das Coprodukt von $\mathbb{Z}/n\mathbb{Z}$ und $\mathbb{Q}$ der triviale Ring, und die Inklusionen sind keine Monomorphismen (vgl. Anhang B.6).
Durch Dualisierung kann man analoge Beispiele für Produkte finden. Vor diesem Hintergrund ist bei der Anwendung von Produkten und Coprodukten zu Modellierungszwecken in der Informatik Vorsicht geboten. Insbesondere

[3] Da der Begriff der Komponente in dieser Arbeit eine spezifische fachliche Bedeutung hat, nennen wir die Komponenten eines Coprodukts *Cofaktoren*.

[4] Zur Notation vgl. Anhang B.8

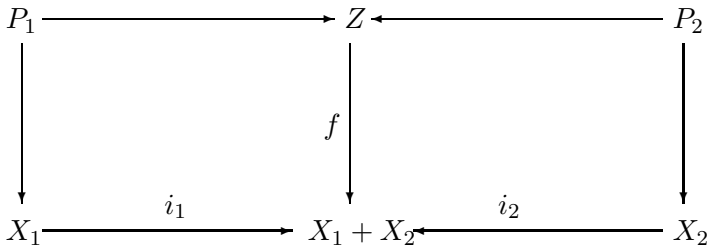
die Gleichsetzung von kategoriellen Produkten mit *Records* und von Coprodukten mit *Variant Records*, wie sie z.B. Wagner [Wag86] vorschlägt, ist problematisch. Zwar liefern diese Konstruktionen in vielen Anwendungskategorien wie Mengen oder Graphen Strukturen, die mit dem intuitiven Verständnis von (Variant) Records in Einklang sind, doch bereits in der häufig betrachteten Kategorie **Rel** der Mengen und Relationen sind sowohl Produkt als auch Coprodukt durch die disjunkte Vereinigung gegeben [BW99, 5.4.7]. Es stellt sich also die Frage, in welchen Kategorien formales und intuitives Verständnis übereinstimmen.
Im folgenden sollen geeignete Voraussetzungen untersucht werden, die Existenz und Eindeutigkeit der Coprodukt-Faktorisierung sicherstellen.

4.3 Extensive Kategorien

Um die Implementierung von Komponenten zu modellieren, bedienen wir uns des Begriffs der extensiven Kategorie [CLW93]. Kontrollflussaspekte können in solchen Kategorien mithilfe von Morphismen, die auf Coprodukten operieren, modelliert werden. In diesem Abschnitt referieren wir die wichtigsten Eigenschaften extensiver Kategorien. Alle nicht bewiesenen Behauptungen entnimmt man der Standardreferenz [CLW93].

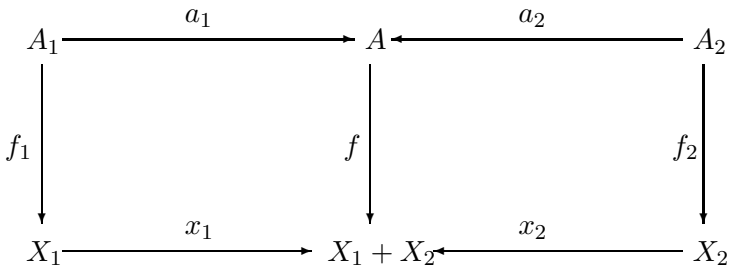
Definition 4.3 *Eine Kategorie $\mathcal{C}$ mit endlichen Coprodukten*[5] *heißt* extensiv*, falls gilt:*

(i) Ist $X_1 \xrightarrow{i_1} X_1 + X_2 \xleftarrow{i_2} X_2$ ein Coprodukt und $f : Z \to Y$ ein beliebiger Morphismus, so existiert das Pullback von f und i_1 (und aus Symmetriegründen auch das von f und i_2).



[5] Die Forderung nach endlichen Coprodukten impliziert insbesondere die Existenz nullstelliger Coprodukte, also initialer Elemente.

(ii) Das kommutative Diagramm



besteht genau dann aus zwei Pullback-Quadraten, wenn die obere Zeile ein Coprodukt darstellt.

Wir geben sofort einige Beispiele für extensive Kategorien an. Die Richtigkeit der Behauptungen wird im Laufe der Diskussion in diesem Abschnitt klar werden.

Beispiele 4.4 (i) Die Kategorie **Set** der Mengen und Abbildungen und die Kategorie **Gph** der Graphen und Graphhomomorphismen sind extensiv (vgl. Satz 4.20).

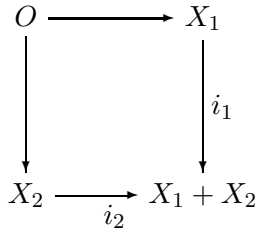
(ii) Die Kategorie **Top** der topologischen Räume und stetigen Abbildungen ist extensiv, ebenso die Kategorie **Mes** der Messräume und messbaren Abbildungen (das zeigt man unter Verwendung von Satz 4.10, s.u.).

(iii) Die endliche Kategorie $\mathfrak{P}(\{0,1\})$, deren Objekte die Teilmengen von $\{0,1\}$ und deren Morphismen die Inklusionen sind, ist nicht extensiv (das wird mit Satz 4.6 klar werden).

Viele typische Anwendungskategorien sind also extensiv. Die Konzeption der extensiven Kategorie intendiert genau dies: Während die meisten aus **Set** oder **Grph** bekannten Eigenschaften von Coprodukten in beliebigen Kategorien – wie im letzten Abschnitt gesehen – nicht gelten müssen, zeichnen sich extensive Kategorien dadurch aus, dass ihre Coprodukte diese vertrauten Eigenschaften besitzen. Was dies im einzelnen bedeutet, führen wir nun aus.

Definition 4.5 *Das Coprodukt in einer Kategorie mit Pullbacks entlang von Coprodukt-Injektionen heißt* disjunkt*, falls das Pullback der Injektionen*

$X_1 \xrightarrow{i_1} X_1 + X_2 \xleftarrow{i_2} X_2$ *ein initiales Objekt* O *ist.*

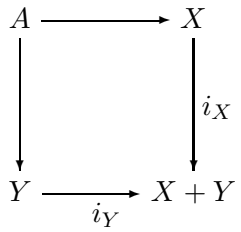


In **Set** bedeutet dies gerade, dass die Bilder von i_1 und i_2 disjunkt sind.

Satz 4.6 *In einer extensiven Kategorie sind die Coprodukte disjunkt*[6].

Initiale Objekte O in extensiven Kategorien haben eine wichtige Eigenschaft, die wir mehrfach als Beweisprinzip verwenden werden: Gibt es einen Morphismus $X \to O$, so muss $X \cong O$ gelten, also X schon selbst initial sein. Man sagt dazu, dass initiale Objekte *strikt* sind. In **Set** ist dies selbstverständlich: Initial ist dort die leere Menge, und eine Abbildung in die leere Menge kann es nur aus der leeren Menge geben. Eine erste Folgerung aus dieser Eigenschaft geben wir sofort.

Bemerkung 4.7 Die Aussage von Satz 4.6 lässt sich verschärfen: In jedem kommutativen Diagramm



ist $A \cong O$. Dies sieht man, indem man die universelle Eigenschaft des Pullbacks anwendet und einen Morphismus $A \to O$ erhält. Da initiale Objekte strikt sind, muss $A \cong O$ gelten.

Neben der Disjunktheit gibt es noch eine zweite wichtige Eigenschaft von Coprodukten in extensiven Kategorien. Wir betrachten wieder zunächst

[6] Beweise findet man in [CLW93]

die Situation in **Set**. Sind dort eine Abbildung $f : X \to Y$ und eine Coprodukt-Zerlegung $Y = Y_1 \dot\cup Y_2$ gegeben, so induziert dies eine Zerlegung $X = f^{-1}(Y_1) \dot\cup f^{-1}(Y_2)$ der Menge X. Kategoriell entspricht die disjunkte Vereinigung einem Coprodukt und die Urbildbildung einem Pullback. Man erhält so folgende

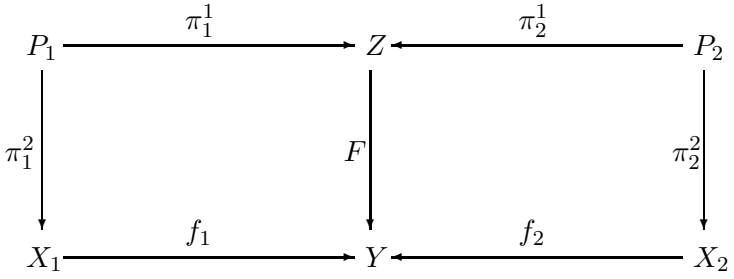
Definition 4.8 *Das Coprodukt in einer Kategorie heißt* universell*, falls es stabil unter Pullbacks ist, d.h. (doppelte) Pullbacks von Coprodukten Coprodukte sind.*

Bemerkung 4.9 In extensiven Kategorien sind Coprodukte universell (das ist gerade die *nur dann*-Richtung der Äquivalenz aus Punkt (*ii*) der Definition).

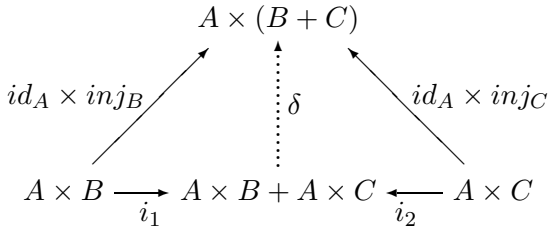
Die beiden genannten Eigenschaften, Disjunktheit und Universalität, charakterisieren extensive Kategorien bereits vollständig.

Satz 4.10 *Eine Kategorie mit endlichen Coprodukten und Pullbacks entlang ihrer Injektionen ist genau dann extensiv, wenn die Summen universell und disjunkt sind.*

Da die Disjunktheit der Coprodukte oftmals leicht zu erkennen ist, erleichtert dieser Satz in vielen Fällen den Beweis der Extensivität einer Kategorie. In **Set**, **Top** und **Mes** beispielsweise, wo das Coprodukt die disjunkte Vereinigung ist (in **Top**/**Mes** versehen mit der finalen Topologie/σ-Algebra bezüglich der Inklusionen), ist die Disjunktheit der Summen trivial. Es bleibt also für die Extensivität dieser Kategorien nur noch die Universalität der Summen zu zeigen. Diese folgt jedoch leicht, wenn man bedenkt, dass in einem Pullback-Diagramm



durch $X_1 \xrightarrow{f_1} Y \xleftarrow{f_2} X_2$ genau dann ein Coprodukt gegeben ist, wenn $Y = f_1(X_1) \dot\cup f_2(X_2)$ ist und f_1, f_2 injektiv sind, und durch $P_1 \xrightarrow{\pi_1^1} Z \xleftarrow{\pi_2^1} P_2$ genau dann, wenn $Z = \pi_1^1(P_1) \dot\cup \pi_2^1(X_2)$ ist und p_1, p_2 injektiv sind.
Man hat $P_i = \{(z, x) \mid F(z) = f_i(x)\} \subseteq Z \times X_i$ und $\pi_i^1(P_i) = \{z \in Z \mid F(z) \in f_i(X_i)\} = F^{-1}(f_i(X_i))$, und da disjunkte Zerlegungen unter inversen Bildern erhalten bleiben, folgt die Behauptung.
Eine Verschärfung des Begriffs der extensiven Kategorie ist derjenige der *lextensiven* Kategorie. Darunter versteht man eine extensive Kategorie mit allen endlichen Limiten. Diese Begriffsbildung hat bereits in der Informatik Anwendung gefunden [JRW02, JR01] und wird auch in dieser Arbeit eine zentrale Rolle spielen. Die Forderung der Existenz endlicher Limiten umfasst insbesondere alle endlichen Produkte (damit auch terminale Objekte).
In extensiven Kategorien mit Produkten haben nicht nur die Coprodukte selbst besondere Eigenschaften, sondern auch ihr Zusammenspiel mit den Produkten ist bemerkenswert.
In jeder Kategorie mit Produkten und Coprodukten gibt es einen kanonischen Morphismus $\delta : A \times B + A \times C \to A \times (B + C)$, der als vermittelnder Coprodukt-Morphismus definiert ist:



Definition 4.11 *Eine Kategorie mit endlichen Produkten und Coprodukten heißt* distributiv*, falls der kanonische Morphismus $\delta : A \times B + A \times C \to A \times (B + C)$ ein Isomorphimus ist.*

In distributiven Kategorien gilt also zwischen Produkten und Coprodukten *(Summen)* ein Distributivgesetz, das syntaktisch wie sein von den natürlichen Zahlen bekanntes Analogon aussieht.

Satz 4.12 *Jede extensive Kategorie mit endlichen Produkten ist distributiv.*

Alle bisher betrachteten extensiven Kategorien besitzen endliche Produkte, sind also distributiv. Es gibt jedoch auch distributive Kategorien, die nicht extensiv sind, und Kategorien, die weder distributiv noch extensiv sind.

Beispiele 4.13 (i) $\mathfrak{P}(\{0,1\})$ ist distributiv, jedoch, wie oben gesehen, nicht extensiv.

(ii) Die Kategorie **Rel** der Mengen und Relationen ist nicht distributiv. Denn **Rel** ist selbstdual, d.h. isomorph zu $\mathbf{Rel}^{op}$. Ein Isomorphismus ist durch den Funktor $F : \mathbf{Rel} \to \mathbf{Rel}^{\mathrm{op}}$ gegeben, der jede Menge auf sich selbst und jede Relation R auf ihre Konverse R^T abbildet. F^{-1} ist genauso definiert, aufgefasst als Funktor $\mathbf{Rel}^{\mathrm{op}} \to \mathbf{Rel}$. Daher sind Produkte und Coprodukte durch die gleiche Konstruktion gegeben (nämlich durch die disjunkte Vereinigung). Wäre **Rel** distributiv, so müsste $X \times X \times X \cong X \times (X + X) \cong X \times X + X \times X \cong X \times X \times X \times X$ sein, was für endliche nichtleere Mengen X falsch ist.

Bemerkung 4.14 Wir identifizieren in distributiven Kategorien $A \times B + A \times C$ und $A \times (B + C)$ miteinander und schreiben beispielsweise $\mathrm{inj} : A \times B \to A \times (B + C)$ für die Komposition

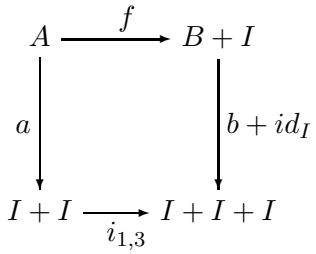
$$A \times B \xrightarrow{i_1} A \times B + A \times C \xrightarrow{\delta} A \times (B + C)$$

Sehr wichtig für die Arbeit mit Coprodukten in distributiven und extensiven Kategorien ist der folgende

Satz 4.15 *In einer distributiven Kategorie sind die Injektionen in ein Coprodukt Monomorphismen* [CLW93, 3.3].

Einen allgemeinen Überblick über distributive Kategorien bietet [Coc93]. Jede distributive Kategorie $\mathcal{C}$ läßt sich zu einer extensiven Kategorie vervollständigen. Cockett und Lack [CL01] betrachten dazu die Kategorie $\mathsf{Bool}(\mathcal{C})$ der Booleschen Ausdrücke in $\mathcal{C}$, deren Objekte Paare (A, a) sind, wobei A ein Objekt in $\mathcal{C}$ und $a : A \to I + I$ ein $\mathcal{C}$-Morphismus ist. Dabei repräsentiert das Coprodukt $I + I$ des terminalen Objekts mit sich selbst die beiden Wahrheitswerte. Morphismen zwischen (A, a) und (B, b) in $\mathsf{Bool}(\mathcal{C})$ sind die

$\mathcal{C}$-Morphismen $f : A \to B + I$, für die das Diagramm



kommutiert, wobei $i_{1,3}$ die Injektion in den ersten und dritten Cofaktor bezeichnet. Es zeigt sich, dass die Kategorie $\mathsf{Bool}(\mathcal{C})$ stets extensiv ist. Der Funktor $\mathcal{C} \to \mathsf{Bool}(\mathcal{C})$, der ein $\mathcal{C}$-Objekt A auf (A, i_1) und einen $\mathcal{C}$-Morphismus $f : A \to B$ auf dessen Komposition mit der Injektion $B \to B+I$ abbildet, erhält endliche Produkte und Coprodukte und ist für extensive Kategorien $\mathcal{C}$ eine Äquivalenz von Kategorien. In diesem Sinne ist $\mathsf{Bool}(\mathcal{C})$ eine extensive Kategorie, die $\mathcal{C}$ vervollständigt. Dies zeigt, dass sich alle Konstruktionen, die wir in extensiven Kategorien durchführen werden, prinzipiell auch in distributiven ausführen lassen, wenn man zur extensiven Vervollständigung übergeht.
Die Kategorie **SRel** der messbaren Räume und stochastischen Relationen [Pan97, 5.1] ist distributiv, da die σ-Algebren der Räume $(X + Y) \times Z$ und $X \times Y + X \times Z$ von denselben Rechtecken erzeugt werden. Sie ist jedoch nicht extensiv. Der leere Raum $\emptyset$ ist terminales Objekt, denn es gibt genau eine messbare Abbildung von einem beliebigen Raum Z in die Menge $\mathbb{P}(\emptyset)$ der Wahrscheinlichkeitsmaße; diese ordnet jedem Punkt in Z das leere Maß $\mu : \emptyset \to [0, 1]$ zu. Allerdings ist $\emptyset$ auch initiales Objekt in **SRel**. Damit besitzt **SRel** Nullobjekte. Da der Einbettungsfunktor beliebige, insbesondere also auch unäre, endliche Produkte und Coprodukte erhält, ist $\emptyset$ auch Nullobjekt in der extensiven Vervollständigung $\mathsf{Bool}(\mathbf{SRel})$. Nach Anhang B ist $\mathsf{Bool}(\mathbf{SRel})$ daher degeneriert.

Topoi

Extensive Kategorien sind Spezialfälle von Topoi. Bei diesen handelt es sich um Kategorien, die sich durch die Existenz von Potenzobjekten auszeichnen. Bekanntlich kann man in der axiomatischen Mengenlehre alle Mengen mithilfe einer kleinen Anzahl von Axiomen konstruieren [Sch93], eines von

ihnen ist das Potenzmengenaxiom: Zu jeder Menge A existiert eine Potenzmenge $\mathfrak{P}(A)$. Eine Abbildung $f : B \to \mathfrak{P}(A)$ ist dann nichts anderes als eine Relation R zwischen B und A, wenn man $R_{ba} :\Leftrightarrow a \in f(b)$ setzt. Diese Konstruktion lässt sich kategoriell verallgemeinern [Gol79, 4.7]:

Definition 4.16 *Sei $\mathcal{C}$ eine Kategorie mit Produkten. Man sagt, dass $\mathcal{C}$* Potenzobjekte besitzt*, wenn für jedes Objekt A Objekte $\mathfrak{P}(A)$ und E_A sowie ein Monomorphimus $e : E_A \to A \times \mathfrak{P}(A)$ existieren, so dass gilt: Ist $R \in |\mathcal{C}|$ ein Objekt und $r : R \to A \times B$ ein Monomorphismus (eine "Relation") für ein beliebiges Objekt B, so existiert genau ein $f : B \to \mathfrak{P}(A)$, so dass ein Pullback der Form*

$$\begin{array}{ccc} R & \xrightarrow{r} & A \times B \\ \downarrow & & \downarrow{\scriptstyle id_A \times f} \\ E_A & \xrightarrow[e]{} & A \times \mathfrak{P}(A) \end{array}$$

existiert. Das Objekt $\mathfrak{P}(A)$ heißt dann Potenzobjekt *von A.*

Beispiel 4.17 In **Set** ist das Potenzobjekt einer Menge A die Potenzmenge von A. Genauer: Man setzt $E_A := \{\langle x, X\rangle \mid x \in X, X \subseteq A\}$ und für $e : E_A \to A \times \mathfrak{P}(A)$ die Inklusion. Ist eine Relation $R \subseteq A \times B$ mit Inklusion $r : R \to A \times B$ gegeben, so bildet die Abbildung $f : B \to \mathfrak{P}(A)$ ein Element $b \in B$ auf

$$f(b) = \{a \mid \langle a, b\rangle \in R\}$$

ab. Mit Beispiel B.10 ergibt sich als Pullback-Objekt

$$\begin{aligned} P &= \left\{(x, X, a, b) \,\middle|\, e(x, X) = (id_A \times f)(a, b),\, x \in X,\, X \subseteq A,\, a \in A,\, b \in B\right\} \\ &= \left\{(x, X, a, b) \,\middle|\, x = a,\, X = \{a|\langle a, b\rangle \in R\},\, \langle a, b\rangle \in R\right\} \\ &\cong R, \end{aligned}$$

wobei im letzten Schritt durch $\cong$ Gleichmächtigkeit ausdrückt wird, also Isomorphie in **Set**.

Topoi haben viele charakteristische Eigenschaften mit **Set** gemein. Dennoch lässt sich der Begriff des Topos sehr kompakt definieren:

Definition 4.18 *Ein* Topos *ist eine endlich-vollständige Kategorie, die Potenzobjekte besitzt.*

Beispiele 4.19

*(i) Die Kategorien **Set** und **Grph** sind Topoi [LS97, Sect. 33].*

(ii) Die Kategorie der topologischen Räume und stetigen Abbildungen ist kein Topos [Bor94, Prop. 7.1.2].

Es zeigt sich, dass die Forderung eines Potenzobjekts eine Fülle von Eigenschaften zur Folge hat. Näheres findet man in [Gol79, Ch. 4]. Unter anderem sind Coprodukte in Topoi stets disjunkt [Joh77, 1.57] und universell [ibd., 1.51]. Damit gilt nach Satz 4.10:

Satz 4.20 *Jeder Topos ist eine lextensive Kategorie.*

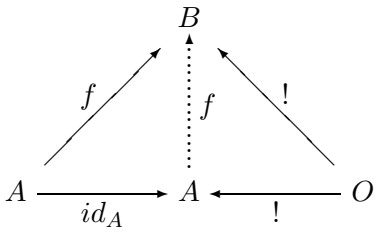
In der Toposttheorie wurden viele wichtige Konstruktionen, die in der Kategorie der Mengen möglich sind, verallgemeinert. Dennoch arbeiten wir in lextensiven Kategorien, und nicht in Topoi, da wir die Existenz von Potenzobjekten nicht benötigen und ihre Forderung nur zu einer unnötigen Einschränkung der möglichen Modelle führen würde. So ist z.B. die Kategorie der topologischen Räume extensiv, obwohl sie kein Topos ist.
Dennoch ist es möglich, Begriffe aus der Topos-Theorie nutzbringend einzusetzen, insbesondere die Idee der globalen Elemente. Da in extensiven Kategorien terminale Objekte existieren, die die einpunktigen Mengen verallgemeinern, kann man auch dort globale Elemente $g : I \to X$ betrachten, also Morphismen von dem terminalen Objekt I in ein beliebiges Objekt X. Diese werden es uns ermöglichen, Eingabedaten zu modellieren und ihren Weg durch ein System zu verfolgen.

4.4 Komplemente

Eine wichtige Frage bei der Untersuchung von Coprodukten ist die nach der Eindeutigkeit von Komplementen:

Definition 4.21 *Ein Morphismus $g : B \to C$ heißt* Komplement *zu $f : A \to C$, falls $A \xrightarrow{f} C \xleftarrow{g} B$ Coprodukt ist.*

In einer Kategorie mit Coprodukt und initialem Objekt O ist $A \xrightarrow{id} A \xleftarrow{!} O$ ein Coprodukt, da das Diagramm



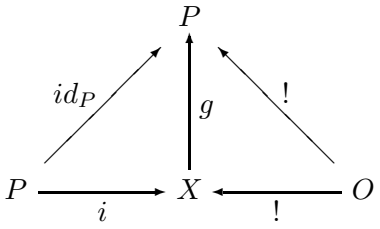
nur mit f als mittlerem Morphismus zu einem Paar kommutativer Dreiecke wird. Es lässt sich jedoch sogar noch mehr zeigen:

Lemma 4.22 *In jeder Kategorie mit Coprodukt und initialem Objekt O gilt:*

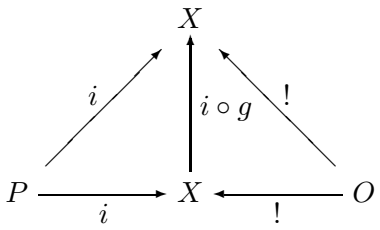
(a) (Manes) *Ist $P \xrightarrow{i} X \xleftarrow{!} O$ ein Coprodukt, so ist i ein Isomorphismus.*

(b) Ist $A_1 \xrightarrow{i_1} O \xleftarrow{i_2} A_2$ ein Coprodukt, so sind A_1 und A_2 initial.

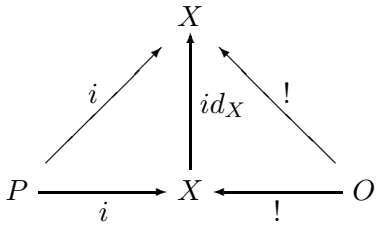
Beweis: (a) Wegen der universellen Eigenschaft des Coprodukts existiert genau ein $g : X \to P$, so dass



kommutiert. Insbesondere gilt $g \circ i = id_P$.
Wegen $(i \circ g) \circ i = i \circ (g \circ i) = i$ ist $i \circ g$ der eindeutig bestimmte vermittelnde Morphismus in dem Coprodukt-Diagramm



Andererseits kommutiert jedoch auch



Also folgt wegen der Eindeutigkeit des induzierten Coprodukt-Morphismus $i \circ g = id_X$ und ingesamt, dass g ein Isomorphismus ist.
(b) Sei X ein beliebiges Objekt. Wegen der universellen Eigenschaft des Coprodukts faktorisiert jeder Morphismus $A_i \to X$ durch O. □

Folgendes Lemma zeigt, dass Coprodukte in extensiven Kategorien in vieler Hinsicht die aus **Set** gewohnten Eigenschaften aufweisen:

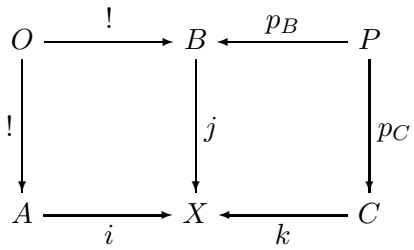
Lemma 4.23 *Folgende Aussagen gelten in einer extensiven Kategorie mit initialem Objekt O und terminalem Objekt I:*

(a) Coprodukt-Komplemente sind bis auf Isomorphie eindeutig:
Sind $A \xrightarrow{i} X \xleftarrow{j} B$ und $A \xrightarrow{i} X \xleftarrow{k} C$ Coprodukte, so existiert ein Isomorphismus $t : C \to B$ mit $j \circ t = k$.

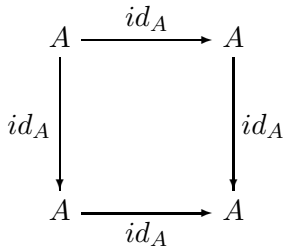
(b) Ist $A \xrightarrow{id_A} A \xleftarrow{j} B$ Coprodukt, so folgt $B \cong O$.

(c) Ist $A \xrightarrow{id_A} A \xleftarrow{id_A} A$ Coprodukt, so ist $A \cong O$.

Beweis: (a) Seien $A \xrightarrow{i} X \xleftarrow{j} B$ und $A \xrightarrow{i} X \xleftarrow{k} C$ Coprodukte. In dem Pullback-Diagramm



ist die obere Zeile ein Coprodukt und daher wegen Lemma 4.22 der Morphismus p_B ein Isomorphismus. Mit $t := p_B^{-1} \circ p_C$ gilt also $j \circ t = k$. Vertauscht man B und C und bildet das analoge Pullback-Diagramm, so erhält man ein $t' : B \to C$ mit $k \circ t' = j$. Setzt man diese Gleichungen ineinander ein, so ergibt sich $k \circ t' \circ t = k$ und $j \circ t \circ t' = j$. Da die Coprodukt-Inklusionen Monomorphismen sind, folgt $t \circ t' = id_B$ und $t' \circ t = id_C$.
(b) Da auch $A \xrightarrow{id} A \xleftarrow{!} O$ ein Coprodukt ist, folgt wegen (a) $B \cong O$.
(c) Das Diagramm



ist ein Pullback. Wegen der Disjunktheit der Coprodukte muss $A \cong O$ sein. □

Bemerkung 4.24 Die Existenz der Komplemente ist in Booleschen Topoi gewährleistet, denn dort bilden die Subobjekte eine Boolesche Algebra [Man92].

4.5 Eindeutige Faktorisierung

Wir wollen nun zeigen, dass in extensiven Kategorien unter geeigneten Voraussetzungen eine eindeutige Coprodukt-Faktorisierung eines Morphismus $f : Z \to X_1 + X_2$ existiert. Es ist klar, dass man diese nur erwarten kann,

wenn nicht schon Z selbst in ein (nichttriviales) Coprodukt zerlegt werden kann.

Definition 4.25 *Ein Objekt Objekt Z einer Kategorie $\mathcal{C}$ heißt* unzerlegbar*, wenn gilt: Ist $Z_1 \xrightarrow{i_1} Z \xleftarrow{i_2} Z_2$ ein Coprodukt, so ist i_1 oder i_2 ein Isomorphismus.*

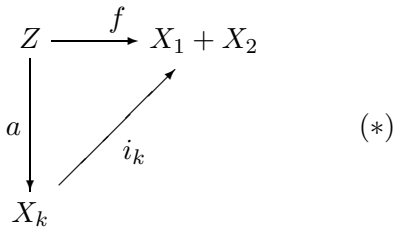
Beispiele 4.26 (i) Terminale Objekte in **Set** sind einelementige Mengen. Ist eine solche als Coprodukt, d.h. disjunkte Vereinigung zweier Mengen darstellbar, so muss eine von ihnen ebenfalls einelementig sein. Somit sind terminale Objekte in **Set** unzerlegbar.

(ii) Das terminale Objekt $\{0,1\}$ in $\mathfrak{P}(\{0,1\})$ ist nicht unzerlegbar, denn es ist Coprodukt der nicht-terminalen Objekte $\{0\}$ und $\{1\}$.

Nun können wir den zentralen Satz dieses Abschnitts formulieren:

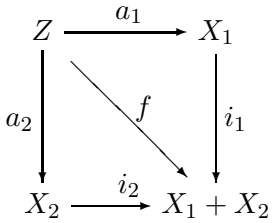
Satz 4.27 *Sei $\mathcal{C}$ eine extensive Kategorie und Z ein unzerlegbares und nichtinitiales Objekt. Dann existiert für jeden Morphimus $f : Z \to X_1 + X_2$ eine eindeutige Faktorisierung.*

Beweis: Wir müssen zeigen, dass genau ein $k \in \{1,2\}$ und ein Morphismus $a : Z \to X_k$ existiert, so dass folgendes Diagramm kommutiert:



(∗)

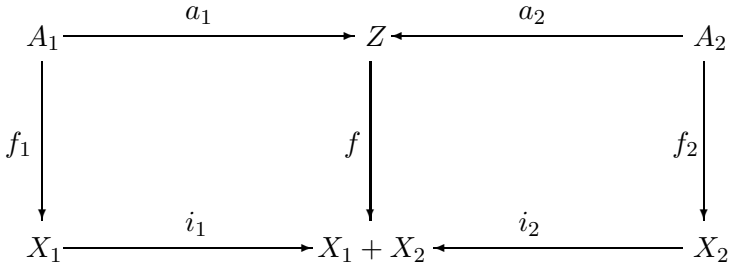
Aus der Disjunktheit der Coprodukte folgt, dass der Index $k \in \{1,2\}$ eindeutig bestimmt ist. Denn gäbe es $a_1 : Z \to X_1$ und $a_2 : Z \to X_2$, so dass beide Dreiecke in folgendem Diagramm kommutieren,



so kommutiert auch das Viereck, und da das Pullback der Injektionen $X_1 \xrightarrow{i_1} X_1 + X_2 \xleftarrow{i_2} X_2$ ein initiales Objekt O ist, existiert ein Morphismus $Z \to O$, woraus $Z \cong O$ folgt, im Widerspruch zur Annahme.

Sind für festes k zwei parallele Morphismen $a, b : Z \to X_k$ gegeben, für die das Dreieck $(*)$ kommutiert, so folgt aus $i_k \circ a = f = i_k \circ b$ und der Tatsache, dass i_k ein Monomorphismus ist, $a = b$.

Die Existenz eines $k \in \{0, 1\}$ und eines Morphismus $\bar{f} : Z \to X_k$ ergibt sich wie folgt: Aufgrund der Universalität des Coprodukts in $\mathcal{C}$, kann man das Pullback des Coprodukt-Diagramms entlang von f bilden und erhält wieder ein Coprodukt:



Da Z unzerlegbar ist, muss a_1 oder a_2 Isomorphismus sein. Sei a_k Isomorphismus und $b = a^{-1} : Z \to A_k$. Dann ist $\bar{f} := f_k \circ b : Z \to X_k$ der gesuchte Morphismus. □

Formal gesprochen bedeutet das, dass der kontravariante Hom-Funktor $\mathrm{Hom}(Z, -)$ Coprodukte respektiert. Während dies für Produkte stets gilt, ist das Resultat für Coprodukte von der Annahme einer lextensiven Kategorie und eines nicht-initalen unzerlegbaren Elements abhängig, wie die pathologischen Beispiele aus Abschnitt 4.2 verdeutlichen.

Wir arbeiten mit *globalen Elementen*, um Eingabewerte zu modellieren [McL92, 1.4]. Ist I ein terminales Objekt und $X \in |\mathcal{C}|$, so nennt man einen Morphismus $a : I \to X$ globales Element von X. Jedes globale Element ist ein Monomorphismus. In **Set** sind zwei parallele Morphismen $f, g : X \to Y$ genau dann gleich, wenn $f \circ a = g \circ a$ für alle globalen Elemente a von X ist. Dies gilt jedoch nicht in allen Kategorien, insbesondere nicht in sol-

chen, die Nullobjekte[7] besitzen, wie z.B. zahlreiche algebraische Kategorien, etwa **Grp**, **Mon** und **Vct**, aber auch **Rel**. Kategorien, die die Bedingung erfüllen, nennt man *wohlpunktiert* [Mac97, S. 291]. Da Wohlpunktiertheit jedoch eine recht starke Einschränkung darstellt (Bemerkung 4.32), wollen wir mit schwächeren Annahmen arbeiten und dabei auf Satz 4.27 aufbauen. Wir benötigen daher ein Lemma, das die Unzerlegbarkeit terminaler Objekte sicherstellt:

Lemma 4.28 *In einer nicht-degenerierten lextensiven Kategorie $\mathcal{C}$, deren leere Objekte initial sind, sind terminale Objekte unzerlegbar.*

Beweis: Wir müssen zeigen: Ist $A \xrightarrow{i} I \xleftarrow{j} B$ Coprodukt, so ist einer der Morphismen i, j ein Isomorphismus und der andere ein Nullmorphismus, faktorisiert also durch das initiale Objekt O. O.B.d.A. sei $i : A \to I$ kein Nullmorphismus. Dann ist A nicht initial (da initiale Objekte strikt sind). Für ein beliebiges globales Element $g : I \to A$ muss $i \circ g : I \to I$ der identische Morphismus sein (denn es gibt genau einen Morphismus $I \to I$). Also ist i split-epi. Da i als Coprodukt-Inklusion in einer lextensiven Kategorie ein Monomorphismus ist, folgt, dass i Isomorphismus ist. Aufgrund von Lemma 4.23 (a) muss dann $B \cong O$ sein. □

Damit ergibt sich folgender:

Satz 4.29 *Sei $\mathcal{C}$ eine nicht-degenerierte lextensive Kategorie, in der alle leeren Objekte initial sind. Dann existiert für jeden Morphimus $f : I \to \coprod_{k=1}^{n} X_k$ genau ein $1 \leq k \leq n$, so dass f Fixpunkt von M_k ist. Mit der Setzung $\Phi_f := X_k$ ist dann $p_k \circ f : I \to \Phi_f$ der eindeutige Faktor.*

Beweis: Induktiv mit Satz 4.2 und Lemma 4.28 □

Als Kategorien, in denen wir unsere Konstruktionen durchführen können, haben sich also nicht-degenerierte lextensive Kategorien $\mathcal{C}$ herauskristallisiert, in denen leere Objekte initial sind.
Im folgenden führen wir die wichtigsten Eigenschaften dieser Kategorien noch einmal explizit auf:

[7] Existiert ein Nullobjekt, so sind alle initialen und alle terminalen Objekte Nullobjekte. Daher existiert dann für jedes Objekt $X \in \mathcal{C}$ genau ein globales Element $g : I \to X$.

- $\mathcal{C}$ ist extensiv.
- Es existieren:
  - Produkte,
  - Coprodukte,
  - Pullbacks,
  - terminale Objekte,
  - initiale Objekte.
- Es existieren keine
  - Nullobjekte.
- $\mathcal{C}$ ist distributiv.
- Für $X \ncong O$ ist $\mathrm{Hom}(I, X) \neq \emptyset$.

Beispiele 4.30 Damit ergeben sich für unsere Konstruktionen folgende Anwendungskategorien, die die obigen Bedingungen erfüllen:

(i) Die Kategorie **Set** der Mengen und Abbildungen als Standardbeispiel.

(ii) Die Kategorie **Grph** der Graphen und Homomorphismen. Sie ist als Topos automatisch lextensiv. Leere Objekte in **Grph** sind die leeren Graphen $G = (\emptyset, \emptyset)$; diese sind genau die initialen Objekte.

(iii) Die Kategorie **Top** der topologischen Räume und stetigen Abbildungen ist, wie gesehen, extensiv (S. 66). Da sie zudem endliche Produkte besitzt, ist sie auch lextensiv. Die leeren und die initialen Objekte stimmen auch hier überein: Es ist die leere Menge $\emptyset$ mit der Topologie $\{\emptyset\}$.

(iv) Für die Kategorie **Mes** der Messräume und messbaren Abbildungen gelten die zu **Top** gemachten Ausführungen analog. Leer und gleichzeitig initial ist auch hier die leere Menge mit der leeren σ-Algebra.

Bemerkung 4.31 Die Voraussetzung, dass nicht-initiale Objekte nicht-leer sind, ist notwendig. Die Kategorie $\mathbf{Set}^2$, deren Objekte Paare von Mengen sind, ist ein nicht-degenerierter Topos [Gol79, 4.4], also insbesondere lextensiv. Jedoch ist das nicht-initiale Objekt $\langle O, I\rangle$ leer, da ein Morphismus $\langle f, g\rangle : \langle I, I\rangle \to \langle O, I\rangle$ einen Morphismus $f : I \to O$ in **Set** induzieren würde. Das terminale Objekt $\langle I, I\rangle$ in $\mathbf{Set}^2$ ist nicht unzerlegbar, sondern läst sich als Coprodukt

$$\langle O, I\rangle \longrightarrow \langle I, I\rangle \longleftarrow \langle I, O\rangle$$

darstellen.

Bemerkung 4.32 Die Forderung, dass jedes nicht-initiale Objekt nicht-leer sein muss, ist in der Tat schwächer als Wohlpunktiertheit. Um dies zu sehen, betrachten wir die Kategorie **IdemSet** von Mengen mit idempotenten Endofunktionen. Ein Objekt von **IdemSet** ist ein Paar (X, λ) bestehend aus einer Menge X und einer idempotenten Funktion $\lambda : X \to X$ (also $\lambda^2 = \lambda$). Ein Morphismus von (X, λ) nach (Y, μ) ist eine Abbildung $f : X \to Y$, für die das folgende Diagramm kommutiert:

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle\lambda}\downarrow & & \downarrow{\scriptstyle\mu} \\ X & \xrightarrow[f]{} & Y \end{array} \quad .$$

Diese Konstruktion ist ein Spezialfall der Kategorie[8] **M-Set** [Gol79, 4.6] und damit ein Topos, insbesondere also eine lextensive Kategorie. Terminales Objekt in **IdemSet** ist (I, id_I) für jede einelementige Menge $I = \{*\}$. Ein globales Element $g : I \to X$ muss die Bedingung $\lambda \circ g = g \circ id_I = g$ erfüllen, ist also ein Fixpunkt von λ.
Initiales Objekt O ist die leere Menge mit der leeren Abbildung. In **IdemSet** ist jedes nicht-initiale Objekt (X, λ) nicht-leer: Da eine Endofunktion genau

[8] Ist M ein Monoid, so sind die Objekte von **M-Set** Paare (X, λ), wobei X eine Menge ist und $\lambda : M \times X \to X$ eine Operation von M auf X. Morphismen sind die Funktionen $X \to Y$, die mit der Operation verträglich sind. Wählt man für M das zweielementige Monoid mit $1 * 1 = 1$ und $0 * 1 = 1 * 0 = 0 * 0 = 0$, so erhält man genau **IdemSet**.

dann idempotent ist, wenn ihr Bild $Im\lambda$ mit der Menge ihrer Fixpunkte $Fix\lambda$ übereinstimmt, existiert ein $x \in Fix\lambda = Im\lambda \neq \emptyset$ und man hat ein globales Element $g : I \to X$ mit $g(*) = x$ von X.
Um zu sehen, dass **IdemSet** nicht wohlpunktiert ist, versehen wir die dreielementige Menge $X = \{1, 2, 3\}$ mit der Endofunktion

$$\begin{aligned} \lambda : X &\to X \\ 1 &\mapsto 1 \\ 2 &\mapsto 2 \\ 3 &\mapsto 2. \end{aligned}$$

Offenbar ist durch λ selbst ein Morphismus $(X, \lambda) \to (X, \lambda)$ definiert, den wir der besseren Unterscheidbarkeit halber mit f bezeichnen. Ein weiterer, von f verschiedener Endomorphimus von (X, λ) ist die identische Funktion id_X. Nun existiert kein globales Element $g : I \to X$ mit $f \circ g \neq id_x \circ g$, da g dann kein Fixpunkt von $f = \lambda$ wäre. **IdemSet** kann also nicht wohlpunktiert sein.

Wir haben in diesem Kapitel gesehen, dass nicht-degenerierte lextensive Kategorien, in denen leere Objekte initial sind, eine eindeutige Coprodukt-Faktorisierung besitzen. Dies werden wir im nächsten Kapitel für unser dynamisches Modell ausnutzen.

Kapitel 5

Dynamisches Modell

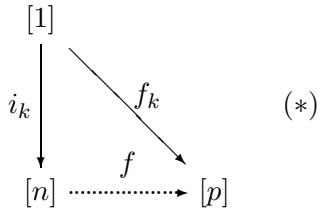
Eine Komponente kann auf einen Aufruf, der ihr von einer anderen Komponente übermittelt wird, auf zwei Weisen reagieren: Entweder sie beantwortet den Aufruf sofort, oder sie schickt sequentiell anderen Komponenten Unteraufrufe, die sie zur Beantwortung benötigt, und sendet die Antwort, sobald sie alle Informationen und Dienstleistungen anderer Komponenten empfangen hat[1]. Während dieses Prozesses verändert die Komponente ihren inneren Zustand.
Wir verwenden ein kategorielles Modell, basierend auf lextensiven Kategorien. Diese sind zwar schon verschiedentlich in der Informatik eingesetzt worden, nicht jedoch in der kategoriellen Modellierung von Softwarearchitekturen. Als kategorientheoretische Ansätze in diesem Bereich sind [Bar01], [Dob03a] und [WF98] zu nennen. Ersterer verwendet eine coalgebraische Herangehensweise, während der zweite allgemeiner in der Kleisli-Kategorie einer Monade [Mac97, VI.5] arbeitet. Beiden ist gemeinsam, dass die modellierten Architekturen einen festen Kontrollfluss besitzen, während wir von einer dynamischen Entscheidung über den Kontrollfluss ausgehen. Die letzte der erwähnten Arbeiten verwendet zwar Ideen aus der Kategorientheorie, um Verbindungsmuster mobiler Programme zu modellieren, basiert jedoch auf einem rein mengentheoretischen Ansatz. Unser Ziel ist es hingegen, vom allgemeinen Begriff der Kategorie ausgehend durch rein kategorielle Bedingungen die Voraussetzungen zu gewährleisten, die für eine Modellierung kom-

[1] Terminierungsfragen interessieren an dieser Stelle nur insoweit, wie sie das Zusammenspiel der Komponenten untereinander betreffen. Jede Komponente für sich genommen soll stets terminieren.

ponentenbasierter Systeme nötig sind. Es wird sich herausstellen, dass es im Wesentlichen genügt, vorauszusetzen, dass die zugrundeliegende Kategorie lextensiv ist.
Wir definieren die Semantik einer Komponente als einen Morphismus F, der ein Produkt als Argument hat. Ein Faktor dieses Produkts ist ein Objekt U, das den Zustand der Komponente beschreibt, der andere Faktor ist ein Coprodukt, das die Eingabe und die Rückgaben der Unteraufrufe verarbeiten kann:

$$F\colon \left(\tau_{\text{in}} + \coprod_{k=1}^{m} \tau_{\text{out}_k}\right) \times U \longrightarrow \left(\tau_{\text{out}} + \coprod_{k=1}^{m} \tau_{\text{in}_k}\right) \times U.$$

Erhält F als Eingabe einen Datenwert x aus τ_{in} und den Zustand u der Komponente, so ist das Ergebnis ein Datenwert y und der neue Zustand u'. Je nachdem, in welchem Cofaktor das Element y liegt, entscheidet sich der weitere Kontrollfluss innerhalb des Systems zwischen $m+1$ Möglichkeiten. Liegt y in τ_{out}, so wird der Aufruf sofort beantwortet und y ist der Rückgabewert. Liegt y in einem der Objekte τ_{in_k}, so wird der k-te von m möglichen Unteraufrufen initiiert.
Ein ähnliches Vorgehen wurde von Elgot vorgeschlagen [Elg75, Blo82], aufbauend auf der dualen Variante der Algebraischen Theorien im Sinne von Lawvere [Law63]. Elgot betrachtet Kategorien T, deren Klasse von Objekten die Menge aller $[n]$ mit $n \in \mathbb{N}$ ist. Er fordert die Existenz ausgezeichneter Morphismen $i_k : [1] \to [n]$ $(i = 1, ..., n)$, die $[n]$ zu einem n-fachen kategoriellen Coprodukt von [1] machen. Genauer: [Elg75, 2.2]

> Sei $[p]$ ein beliebiges Objekt. Dann existiert für jede Familie von Morphismen $f_k : [1] \to [p]$, $k = 1, \ldots, n$, genau ein Morphismus $f : [n] \to [p]$, so dass $f \circ i_k = f_k$ für jedes k gilt, also folgendes Diagramm kommutiert:
>
> 
>
> $(*)$

Eine solche Kategorie T bezeichnet man nach Eilenberg und Wright als *Theorie* [EW67]. Obwohl als Objekte einer Theorie stets durch die natürlichen

Zahlen $\mathbb{N}$ repräsentiert werden, können die entstehenden Kategorien stark variieren. Dem liegt der – häufig geäußerte – Gedanke zugrunde, dass die Morphismen, und nicht die Objekte, die entscheidenden Größen einer Kategorie sind [EM45, Gog91]. Man hat dann im allgemeinen keine unmittelbare Interpretation der Morphismen als Abbildungen zwischen den Objekten, wie dies in konkreten Kategorien der Fall ist, sondern die Objekte dienen primär dazu, die Morphismen in Morphismenmengen $\mathrm{Hom}([m],[n])$ zu strukturieren. Elgot gibt als Beispiel die Kategorie $[R]$ der Matrizen mit Einträgen in einem kommutativen Ring R mit Einselement. Die Menge der Morphismen zwischen $[m]$ und $[n]$ besteht aus den $m \times n$-Matrizen[2].

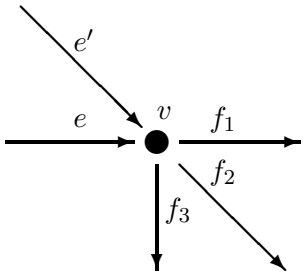
Das wichtigste Beispiel für eine Theorie ist jedoch das folgende: Für eine feste Menge X seien die Morphismen $[m] \to [n]$ alle Abbildungen zwischen den Mengen $X \times \{1,\ldots,m\}$ und $X \times \{1,\ldots,n\}$. Die ausgezeichneten Morphismen $i_k : X \to X \times \{1,\ldots,n\}$ bilden $x \in X$ auf (x,k) ab. Da man in **Set** das n-fache Coprodukt $\coprod_{k=1}^{n} X$ mit $X \times \{1,\ldots,n\}$ identifizieren kann, ist Bedingung (*) erfüllt. In seiner Arbeit [Elg71] gibt Elgot eine Semantik für Flussdiagramme in dieser Theorie an. Sie beruht im wesentlichen auf der Idee, dass für eine Eingabe $x \in X$ neben dem Berechnungsergebnis durch einen Morphismus $[1] \to [p]$ auch eine Entscheidung zwischen p möglichen Zweigen des weiteren Kontrollflusses getroffen wird. Ist $f(x) \in X \times \{k\}$ für ein $k \in \{1,\ldots,p\}$, so wird der k-te Zweig ausgewählt.

Unser Modell verzichtet auf die Verwendung natürlicher Zahlen und verwendet statt dessen Coprodukte. Neben der Unabhängigkeit von $\mathbb{N}$ ermöglicht dies insbesondere eine größere Flexibilität, da wir Coprodukte aus verschiedenen Cofaktoren benutzen können, d.h. der Datentyp kann je nach gewähltem Zweig verschieden sein. Verzichten wir gedanklich einen Moment auf diese Flexibilität und beschränken uns auf Coprodukte der Form $\coprod_{k=1}^{p} X$, so sehen wir, wie unser Ansatz sich in den Rahmen der Theorien im Sinne von Eilenberg und Wright einfügt. Setzen wir nämlich $[p] = \coprod_{k=1}^{p} X$ und nehmen für $i_k : [1] \to [p]$ die k-te Coprodukt-Injektion, so folgt die Existenz eines f, für das das Diagramm $(*)$ kommutiert, unmittelbar aus der Coprodukt-Eigenschaft B.4.

[2] Dies sind natürlich genau die R-Modul-Homomorphismen $R^m \to R^n$, aber für die Struktur der Kategorie ist es irrelevant, ob man die Objekte als natürliche Zahlen oder als Moduln R^n ansieht.

5.1 Implementierung von Komponenten

Die Funktionalität, die eine Komponente v anderen Komponenten zur Verfügung stellt, wird durch ihre eingehenden Schnittstellen bestimmt. Wenn für jede eingehende Schnittstelle festgelegt ist, wie die Komponente auf Eingaben reagiert, so ist sie vollständig implementiert. Insofern ist eine Komponente durch ihre Schnittstellen bestimmt. Gleichzeitig gehen wir davon aus, dass Komponenten zustandsbehaftet sind. So ist es beispielweise möglich, dass über eine Schnittstelle ein Zugriff erfolgt, der den Zustand so verändert, dass beim Zugriff über eine andere Schnittstelle eine veränderte Rückgabe entsteht. In einem Schichtensystem sind die Schnittstellen der Komponenten durch Konnektoren miteinander verbunden. Daher besteht die Implementierung einer Komponente darin, jedem eingehenden Konnektor eine Semantik zuzuweisen, die den Zustand der Komponente vor der Ausführung berücksichtigt und den Zustand nach der Ausführung verändern kann. Wir wollen diese Überlegungen im folgenden formalisieren.
Sei $\mathcal{C}$ eine nicht-degenerierte lextensive Kategorie, deren leere Objekte initial sind, und (G, p, τ) ein $\mathcal{C}$-Schichtensystem. Ferner sei $v \in G_V$ eine Komponente und $e \in E_v^{in}$ eine eingehender Konnektor von v. Mit e soll eine Funktionalität F_e assoziiert werden, die v aufrufenden Komponenten anbietet.



$$E_v^{in} = \{e, e'\}$$
$$E_v^{out} = \{f_1, f_2, f_3\}$$

Definition 5.1 *Sei $v \in G_V$ eine Komponente und $E_v^{out} = \{f_1, ..., f_m\}$. Sei $U_v \in |\mathcal{C}|$ ein Objekt, $u_v : I \to U_v$ ein globales Element. Für jeden Konnektor $e \in E_v^{in}$ existiere ein Morphismus F_e der Form*

$$F_e \colon \left(\tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)\right) \times U_v \longrightarrow \left(\tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k)\right) \times U_v.$$

Dann heißt $\mathfrak{I} = (U_v, u_v, (F_e)_{e \in E_v^{in}})$ Implementierung *von v.*

Das Objekt U_v repräsentiert die Menge der inneren Zustände der Komponente bezüglich der Implementierung $\mathfrak{I}$, und $u_v : I \to U_v$ ist der initiale Zustand. Das Coprodukt $\tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)$ besteht aus zwei Teilen: Der Cofaktor $\tau_{\text{in}}(e)$ verarbeitet die Eingaben, die der Aufrufer über den Konnektor e übermittelt. Die übrigen Summanden können Rückgabewerte aufnehmen, die die Komponente v von anderen Komponenten erhält, die sie zur Bearbeitung ihreres eigenen Aufrufs aufgerufen hat. In beiden Fällen handelt es sich um Daten, die eine Berechnung anstoßen, die der Morphismus F_e repräsentiert. Das Coprodukt im Bildbereich von F_e gliedert sich ebenfalls in zwei Teile: Der Ausgabe-Cofaktor $\tau_{\text{out}}(e)$ enthält die Rückgabe an den Aufrufer, die Cofaktoren $\tau_{\text{in}}(f_k)$ Werte, die für Unteraufrufe benötigt werden. Besitzt in einem $\mathcal{C}$-Schichtensystem (G, p, τ) jede Komponente eine Implementierung, so nennen wir (G, p, τ) *implementiert*.

Bemerkung 5.2 Ist für die Implementierung einer Komponente $v \in G_V$ das Zustandsobjekt U_v ein terminales Objekt, so sind die Projektionen

$$\Big(\tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)\Big) \times U_v \longrightarrow \tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)$$

und

$$\Big(\tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k)\Big) \times U_v \longrightarrow \tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k)$$

Isomorphismen und wir schreiben der Einfachheit halber

$$F_e\colon\ \tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k) \longrightarrow \tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k).$$

5.2 Aufrufe

Wir zeigen zunächst exemplarisch in der Kategorie **Set** der Mengen und Abbildungen, wie ein Aufruf berechnet wird. Dort können wir mit gewöhnlichen Elementen arbeiten statt mit globalen.
Jede Komponente $v \in G_V$ besitzt eine Menge U_v von Zuständen, die sie annehmen kann. Durch $u_v : I \to U_v$ sei der Zustand gegeben, in dem sich v befindet. Jedem Konnektor $e \in E_v^{in}$ ist eine Abbildung

$$F_e\colon\ \Big(\tau_{\text{in}}(e) + \sum_{k=1}^{m} \tau_{\text{out}}(f_k)\Big) \times U_v \longrightarrow \Big(\tau_{\text{out}}(e) + \sum_{k=1}^{m} \tau_{\text{in}}(f_k)\Big) \times U_v$$

zugeordnet, wobei $f_1, ..., f_m$ die Elemente von E_v^{out} sind, also die Konnektoren, die aus v herausführen[3]. Als Eingaben akzeptiert F_e Paare, deren erstes Element entweder aus $\tau_{\text{in}}(e)$ oder aus einem der $\tau_{\text{out}}(f_k)$ mit $1 \leq k \leq m$ ist. Das zweite Element ist aus der Zustandsmenge U_v. Die Ausgabe entstammt entweder $\tau_{\text{out}}(e)$ oder einem der $\tau_{\text{in}}(f_k)$, jeweils zusammen mit einem Element aus U_v.
Wenn Komponente v sich im Zustand u_v befindet und über Konnektor e aufgerufen werden soll, so muss ihr eine Eingabe $z \in \tau_{\text{in}}(e)$ übermittelt werden. Zunächst gibt es zwei Möglichkeiten:

1. Fall Komponente v kann den Aufruf allein beantworten und benötigt keine Zusammenarbeit mit weiteren Komponenten. In diesem Fall ist $F_e(z, u_v) \in \tau_{\text{out}}(e) \times U_v$. Die Rückgabe an den Aufrufer ist in $\tau_{\text{out}}(e)$ kodiert, während U_v den neuen Zustand von v darstellt.

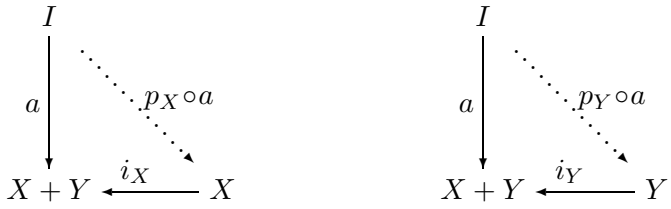
2. Fall Komponente v muss eine Unteraufruf an eine andere Komponente $tar(f_k)$ senden. In diesem Fall ist $F_e(z, u_v) \in \tau_{\text{in}}(f_k) \times U_v$ für ein $k \in \{1, ..., n\}$.

Im ersten Fall ist die Bearbeitung der Aufrufs mit ihrer Beantwortung abgeschlossen. Im zweiten Fall ist $(proj_1 \circ F_e)(z, u_v)$ ein Element von $\tau_{\text{in}}(f_k)$ und kann als Eingabe über Konnektor f_k an eine andere Komponente v' gesendet werden.
In beiden Fällen kann sich durch die Bearbeitung des Aufrufs auch der Zustand von v ändern. Durch $(proj_2 \circ F_e)(z, u_v)$ ist der neue Zustand definiert.

Diese Konstruktion verallgemeinern wir von **Set** auf nicht-degenerierte lextensive Kategorien, deren leere Objekte initial sind. Für diese Kategorien gilt die eindeutige Faktorisierungseigenschaft, wie wir in Abschnitt 4.5 gesehen haben. Statt der Summe, die in **Set** durch die disjunkte Vereinigung gegeben ist, verwenden wir Coprodukte. Elemente werden durch globale Elemente ersetzt. Während wir in **Set** von einem Element $a \in X + Y$ sagen, dass entweder $a \in X$ oder $a \in Y$ ist, müssen wir im allgemeinen Fall die Coprodukt-Faktorisierung überprüfen:

[3]Da wir den Spezialfall $\mathcal{C} = \text{Set}$ behandeln, schreiben wir $\sum$ statt $\coprod$. Zudem vernachlässigen wir der Übersichtlichkeit halber die Injektionen in die disjunkte Vereinigung.

Ist $a : I \to X + Y$ ein globales Element, so kommutiert nach Satz 4.29 genau eines der folgenden Diagramme:



Diese Eigenschaft nutzen wir, um zu entscheiden, in welchem Cofaktor des Coprodukts das globale Element liegt.

Definition 5.3 *Sei $v \in V$ und $e \in E_v^{in}$. Eine* Eingabe *von F_e ist ein globales Element*[4] *$x : I \to \tau_{\text{in}}(e)$. Ist $u : I \to U_v$ ein Startzustand, so bezeichnen wir $g = (x, u) : I \to \tau_{\text{in}}(e) \times U_v$ als* Aufruf *der Komponente v über den Konnektor e.*
Für einen Aufruf, der bei zur Verarbeitung eines anderen Aufrufs entsteht, verwenden wir auch die Sprechweise Unteraufruf.

Wie die dynamische Verarbeitung eines Aufrufs kategoriell modelliert wird, demonstrieren wir zunächst anhand der ersten Verarbeitungsschritte. Eine formale Definition geben wir im Anschluss (Abschnitt 5.5).
Für jedes Element $e \in G_E$ einer Komponente $v \in G_V$ existiere eine Implementierung

$$F_e \colon \left(\tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)\right) \times U_v \longrightarrow \left(\tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k)\right) \times U_v.$$

Den Zustand der Komponente v bezeichnen wir mit u_v. Ein Aufruf von Konnektor e mit Eingabe $x : I \to \tau_{in}$ besteht aus dem Produkt

$$g = (x, u_v) : I \to \tau_{\text{in}}(e) \times U_v,$$

das sowohl den Zustand von v als auch die übermittelten Daten enthält. Um die Signatur von g für die Komposition mit F_e anzupassen, bilden wir die Komposition

[4] Zur Notation (x, u) vgl. Definition B.3

$$\tilde{g} : I \xrightarrow{g} \tau_{\text{in}}(e) \times U_v \xrightarrow{inj.} \Big(\tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)\Big) \times U_v.$$

Die Inklusion hatten wir für $\mathcal{C} = \mathbf{Set}$ nicht explizit aufgeführt, da wir ein Element eines Cofaktors gleich als Element des Coprodukts aufgefaßt haben. Das globale Element $\tilde{g}$ hat nun die geeignete Codomäne, um als Eingabe für den Morphismus F_e zu fungieren, der die Funktionalität von e repräsentiert. Verknüpfen wir $\tilde{g}$ mit F_e, so erhalten wir ein globales Element

$$F_e \circ \tilde{g} : I \to \Big(\tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k)\Big) \times U_v.$$

Dieses repräsentiert das Ergebnis des ersten Verarbeitungsschrittes des Aufrufs g an e. Der in U_v gelegene Faktor $proj_2 \circ F_e \circ \tilde{g}$ gibt den neuen Zustand von v an. In dem globalen Element

$$y := proj_1 \circ F_e \circ \tilde{g} : I \to \tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k)$$

befindet sich die Information über den weiteren Ablauf und die zu übermittelnden Daten. Ein solches globales Element eines $m+1$-fachen Coprodukts liegt nach Satz 4.29 in genau einen Cofaktor. Um diesen zu bestimmen, verwenden wir die natürlichen Transformationen M^{out} und M_k^{in} ($k \in \{1, ..., m\}$) von

$$\text{Hom}\Big(- \,,\ \tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k) \Big)$$

in sich selbst (vgl. Satz 4.2) und die Coprodukt-Projektionen, die wir mit p^{out} bzw. p_k^{in} bezeichnen. Entweder ist y Fixpunkt von M^{out} oder von genau einem der M_k^{in}.

1. Fall: y ist Fixpunkt von M^{out}. Dann ist $a := p^{out} \circ y : I \to \tau_{out}(e)$ die resultierende Arbeit der Komponente.

2. Fall: y ist Fixpunkt von M_k^{in} für ein $k \in \{1, ..., m\}$. In diesem Fall interpretieren wir $a := p_k^{in} \circ y : I \to \tau_{\text{in}}(f_k)$ als Startdatum für einen Unteraufruf. Das globale Element a wird dazu der Komponente v' über Schnittstelle f_k

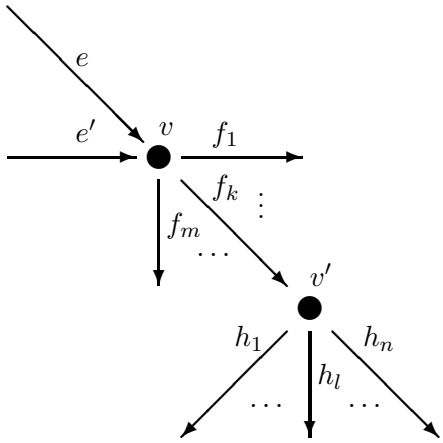


Abbildung 5.1: Ausschnitt aus dem Konfigurationsgraphen

als Eingabe gesendet (Abbildung 5.1). Ist $u_{v'} : I \to U_{v'}$ Zustand von v', so bilden wir analog zu dem eben geschilderten Vorgehen

$$g' := (a, u')$$

und untersuchen dann

$$z := proj_1 \circ F_{f_k} \circ \tilde{g}' : I \to \tau_{\text{out}}(f_k) + \coprod_{l=1}^{n} \tau_{\text{in}}(h_l),$$

wobei $E^{out}_{v'} = \{h_1, ..., h_n\}$ die Menge der ausgehenden Konnektoren von f_k ist. Hier findet wieder eine Fallunterscheidung statt, die entweder zu einer Antwort auf den Aufruf g' oder zu einem weiteren Unteraufruf führt.
Wir wollen hier für unsere beispielhafte Darstellung annehmen, dass z ein Fixpunkt von M^{out} ist und damit der Aufruf von v' sofort beantwortet wird. Das dann entstehende globale Element $b := p^{out} \circ z : I \to \tau_{out}(f_k)$ wird mit dem Zustand u_v zu $g'' := (z, u_v)$ kombiniert, es wird

$$\tilde{g}'' : I \to \left(\tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)\right) \times U_v$$

gebildet und $w := proj_1 \circ F_e \circ \tilde{g}''$ betrachtet. Wir nehmen weiter beispielhaft an, dass w Fixpunkt von M^{out} ist. Dann ist durch

$$p^{out} \circ w$$

die endgültige Antwort auf den ursprünglichen Aufruf gegeben.
Im allgemeinen Fall sind beliebig viele Unter-Aufrufe denkbar. Wenn die Folge der Aufrufe terminiert, ergibt sich aus der Komposition der Morphismen die Gesamt-Arbeit des Systems für einen Aufruf.

Bemerkung 5.4 Man beachte das Vorgehen bei der Bestimmung des Konnektors c, dessen implementierender Morphismus F_c den nächsten Berechnungsschritt vornimmt. Es gelte weiter die Situation aus Abbildung 5.1. Wird Komponente v über Konnektor e angesprochen, so wird F_e zur Berechnung herangezogen. Der Unteraufruf an Komponente v' über f_k wird durch F_{f_k} verarbeitet. Die Antwort von v', die über f_k an v übermittelt wird, wird dann *jedoch wieder von F_e verarbeitet*, da v ja ursprünglich über e aufgerufen wurde und dieser Aufruf den Unteraufruf von v' initiiert hatte.

5.3 Ein einfaches Beispielsystem in Set

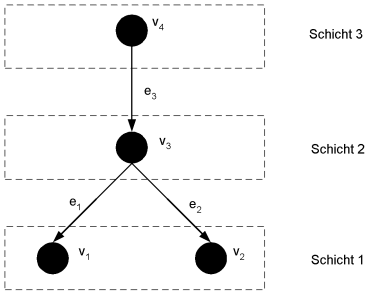


Abbildung 5.2: Ein Beispielsystem

Um die Berechnung eines Aufrufs zu illustrieren, betrachten wir nun ein einfaches Beispielsystem in der Kategorie der Mengen. Sei $G = (G_V, G_E)$ der Graph in Abbildung 5.2. Weiter seien $\tau_{\text{in}}(e_1) = \tau_{\text{in}}(e_2) = \mathbb{N}$ und $\tau_{\text{in}}(e_3) = \tau_{\text{out}}(e_1) = \tau_{\text{out}}(e_2) = \tau_{\text{out}}(e_3) = \mathbb{Z}$. Die Schichteneinteilung p ist aus dem Bild ersichtlich. Dann ist (G, p, τ) ein **Set**-Schichtensystem.
Die Implementierungen der Konnektoren e_1 und e_2 sind Funktionen[5] $F_{e_1}, F_{e_2} : \mathbb{N} \to \mathbb{Z}$. Über die Komponente v_4 kommuniziert der Benutzer

[5]Hier verwenden wir die Schreibweise gemäß Bemerkung 5.2

mit dem System. Eine Implementierung von e_3 ohne innere Zustände ist von der Form:

$$F_{e_3} : \mathbb{Z} + (\mathbb{Z} + \mathbb{Z}) \to \mathbb{Z} + (\mathbb{N} + \mathbb{N})$$

Wir wollen zwei verschiedene Implementierungen betrachten.

1. Fall: Wir setzen

$$F_{e_3}(z) = \begin{cases} j_1(z) & \text{für} \quad z \in i_0(\mathbb{Z}) \wedge z \geq 0, \\ j_2(-z) & \text{für} \quad z \in i_0(\mathbb{Z}) \wedge z < 0, \\ j_0(z) & \text{für} \quad z \in i_1(\mathbb{Z}) \cup i_2(\mathbb{Z}). \end{cases}$$

Dabei bezeichnen wir die drei Injektionen in das Coprodukt $\mathbb{Z} + (\mathbb{Z} + \mathbb{Z})$ mit i_0, i_1, i_2 und die in das Coprodukt $\mathbb{Z} + (\mathbb{N} + \mathbb{N})$ mit j_0, j_1, j_2. Außerdem fassen wir $\mathbb{N}$ als Teilmenge von $\mathbb{Z}$ auf. Anstelle von globalen Elementen verwenden wir in **Set** gewöhnliche.
Die resultierende Arbeit hängt von dem Aufruf z ab. Ist $z \geq 0$, so erhält man $(F_{e_3} \circ i_1 \circ F_{e_1} \circ p_1 \circ F_{e_3})(z) = F_{e_1}(z)$, ansonsten $(F_{e_3} \circ i_2 \circ F_{e_2} \circ p_2 \circ F_{e_3})(z) = F_{e_2}(-z)$, wobei wir mit p_i die Coprodukt-Projektionen bezeichnen. Somit ist die Gesamtarbeit durch

$$G(z) = \begin{cases} F_{e_1}(z) & \text{für} \quad z \geq 0, \\ F_{e_2}(-z) & \text{für} \quad z < 0. \end{cases}$$

gegeben. Der Morphismus F_{e_3} modelliert also eine Fallunterscheidung zwischen den Morphismen F_{e_1} und F_{e_2}, verbunden mit einem Vorzeichenwechsel für negative Eingaben. Das System terminiert in jedem Fall.

2. Fall: Wir setzen

$$F_{e_3}(z) = \begin{cases} j_1(|z|) & \text{für} \quad z \in i_0(\mathbb{Z}) \cup i_2(\mathbb{Z}), \\ j_2(|z|) & \text{für} \quad z \in i_1(\mathbb{Z}). \end{cases}$$

In diesem Fall wird ein über e_3 eintreffender Aufruf zunächst nach e_1 weitergeleitet. Dann läuft der Aufruf zwischen v_1 und v_2 hin und her, ohne zu terminieren. Wir erhalten also einen unendlichen Aufrufbaum, jedoch mit endlicher Tiefe (Abbildung 5.3). Bei immer neuen Unteraufrufen (zwischen zwei Komponenten) kann auch das Phänomen unendlicher Tiefe auftreten, wie wir in Beispiel 5.11 sehen werden. Diese Beispiele zeigen, dass bereits recht einfache Systeme eine komplexe Dynamik entwickeln können.

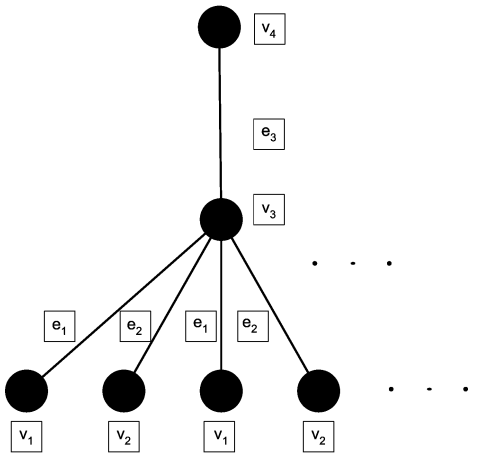


Abbildung 5.3: Ein unendlicher Aufrufbaum.

5.4 Aufrufsequenzen

Bei der Berechnung eines Aufrufs, wie wir sie bisher informell durchgeführt haben, mussten wir uns stets merken, von welcher Komponente ein Subaufruf angestoßen wurde – zu dieser kehrt der Kontrollfluss einer terminierenden Berechnung irgendwann zurück. Dies ist sowohl Anforderung an eine korrekte Berechnung als auch eine ausschließende Bedingung: *Nur* als Antwort auf offene Aufrufe kann in der Schichtenhierachie von unten nach oben kommuniziert werden.
Im Compilerbau verwendet man Aktivierungsbäume [ASU86, 7.1], um den Kontrollfluss während des Programmablaufs zu repräsentieren. Es handelt sich dabei um endliche Wurzelbäume mit Ordnungsrelationen auf den Kindern jedes Knotens, die den zeitlichen Ablauf wie folgt kodieren [ibd.]:

1. Jeder Knoten repräsentiert die Aktivität einer Prozedur.
2. Die Wurzel respräsentiert die Aktivität das Hauptprogramms.
3. Ein Knoten v ist genau dann Vater eines Knotens w, wenn die Aktivität von von v durch einen Unteraufruf an w übergeht.
4. Sind zwei Knoten v, w Kinder desselben Knotens, so liegt v genau dann in der Ordnung vor w, wenn v zeitlich vor w aktiv wird.

Der Kontrollfluss entspricht daher einer ordnungsrespektierenden Tiefensuche im Aktivierungsbaum. In der Praxis ist es nicht nötig, den gesamten Kontrollfluss eines Programmablaufs zu repräsentieren, sondern es genügt, im *control stack* abzuspeichern, über welche Zwischenschritte ein Aufruf den aktiven Programmteil erreicht hat. Der Tiefe des Baums entspricht die Größe des benötigten Stacks zur Verwaltung der Unteraufrufe. Da diese beliebig verschachtelt sein können, muss die Kontrollinstanz dafür Sorge tragen, dass stets ein hinreichend großer Stack zur Verfügung steht.
Um die explizite Konstruktion eines Stacks und der damit verbundenen Operationen zu vermeiden, greifen wir die Idee des Aktivierungsbaums auf. Wir unterscheiden formal zwischen den Elementen des Baumes und den Systemkomponenten und beschreiben die Berechnung eines Aufrufs durch einen Homomorphismus von einem Baum in den Systemgraphen. Für den Baum definieren wir durch Tiefensuche den kanonischen Überdeckungspfad, der einen Aufrufpfad im Systemgraphen induziert. Insofern repräsentieren die Bäume eine *globale Kontrolle* des Aufrufs: Jede Komponente entscheidet zwar lokal über den nächsten Aufruf, aber die Baumstruktur stellt sicher, dass die Antwort auf einen Aufruf nicht übergangen werden kann.
Der Zustand des Systems während der Verarbeitung eines Aufrufs besteht also einerseits aus dem Produkt der Zustände U_v der Komponenten $v \in G_V$ und andererseits aus dem Aufrufbaum. Dieser gibt die aktuell aktive Komponente wieder und enthält darüberhinaus die Information über den Stack der offenen Aufrufe und Unteraufrufe.
Die Annahme einer globalen Kontrolle schränkt die Verwendung unsres Ansatzes auf solche Softwaresysteme ein, die über eine zentrale Steuerungsinstanz verfügen. Dies gilt im allgemeinen nicht für verteilte Systeme. Für diese sind jedoch auch andere Aspekte unserer Modellierung inadäquat, wie etwa die Annahme einer festen Konfiguration der Komponenten untereinander, so dass tiefgreifende Anpassungen erforderlich wären, um unseren Ansatz in dieser Richtung auszudehnen, die jedoch hier nicht weiter verfolgt werden sollen.
Die Kategorie **ORTree** (für engl. *ordered rooted trees*) der geordneten Wurzelbäume hat als Objekte Paare

$$(T, (<_v)_{v \in T_V})$$

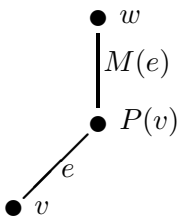
wobei $T = (T_V, T_E, w)$ ein abzählbarer Baum mit ausgezeichneter Wurzel

$w \in T_V$ ist und $(<_v)_{v \in V}$ eine Familie von Wohlordnungsrelationen. Dabei ist $<_v$ eine (strikte) Wohlordnung der Kinder des Knotens $v \in V$; für Blätter ist $<_v$ die leere Relation. Warum wir eine Wohlordnung verlangen, wird in Kürze ersichtlich.

Morphismen zwischen $(T, (<_v)_{v \in V})$ und $(T', (<'_v)_{v \in V})$ in **ORTree** sind alle Graphhomomorphismen $f : T \to T'$, die das Wurzelelement erhalten und monoton bezüglich der Wohlordnungen sind, d.h. für zwei Kinder v_1, v_2 eines Knotens $v \in V$ mit $v_1 < v_2$ sei $f(v_1) < f(v_2)$.

Wir betrachten die Kanten in Wurzelbäumen als von der Wurzel hin zu den Blättern orientiert, und können somit Wurzelbäume als gerichtete Graphen auffassen. Dabei schließen wir auch unendliche Bäume nicht aus.

Man beachte, dass in einem Wurzelbaum mit Wurzel w nicht nur jeder Knoten $v \neq w$ einen *Vaterknoten* $P(v)$ besitzt, sondern auch jede Kante e mit $src(e) \neq w$ eine *Mutterkante* $M(e)$, nämlich die – wegen der Baumeigenschaft – eindeutig bestimmte Kante mit $tar(M(e)) = src(e)$.



Jeder geordnete Wurzelbaum T besitzt einen kanonischen *Überdeckungspfad*, den wir durch Tiefensuche definieren. Wir durchlaufen dabei einen Wurzelbaum mithilfe einer Variante der gewöhnlichen Tiefensuche [AHU74, 5.2]. Während normalerweise die Kinder eines Knotens in beliebiger Reihenfolge durchlaufen werden, ist hier die Ordnungsrelation $<_v$ für die Reihenfolge ausschlaggebend. In Abbildung 5.4 wird der Algorithmus zur Bestimmung der Knotenfolge in Pseudo-Code beschrieben.

Um die durch die Tiefensuche definierten Pfade zu notieren, betrachten wir die formale Umkehrung $\bar{e}$ einer Kante $e \in E$. Die Menge aller $\bar{e}$ bezeichnen wir mit $\bar{E}$. Beginnend bei der Wurzel werden rekursiv die Kinder in der Reihenfolge ihrer Ordnung besucht. Ist man in einem Blatt angekommen, läuft man mit *backtracking* wieder zurück, hat man alle Blätter erreicht, geht es wieder zur Wurzel. So entsteht ein Pfad

$$(v^{(1)}, e^{(1)}, v^{(2)}, e^{(2)}, \ldots)$$

der Teilfolgen $(v^{(i)})_{i \geq 1}$ und $(e^{(i)})_{i \geq 1}$ von Elementen aus V bzw. $E \cup \bar{E}$ besitzt. Die Tiefensuche in einem endlichen Wurzelbaum T wird durch die Ordnungsrelation $<_v$ auf den Kindern jedes Knotens v zu einer eindeutigen Vorschrift,

```
program ordered-depth-first

procedure SEARCH(v)
  begin
    mark v old;
    while there exists a vertex in K[v] marked new do
      begin
        u ← first vertex in K[v] marked new;
        add(v,u) to T;
        SEARCH(u)
      end
  end

begin
  T ← ∅;
  for all v in V do mark v new;
  SEARCH(w)
end
```

Abbildung 5.4: Algorithmus zur geordneten Tiefensuche in Pseudocode. Gegeben sei ein geordneter Baum T mit Wurzel w. Mit $K[v]$ bezeichnen wir die Kinder eines Knotens v. Wir gehen davon aus, dass auf jeder der Mengen $K[v]$ eine Ordnung existiert und bezeichnen mit `first vertex` das kleinste Element bezüglich dieser Ordnung.

mit der die Knoten von T durchlaufen werden. Der dadurch eindeutig bestimmte Pfad, der T vollständig überdeckt, ist der *kanonische Überdeckungspfad* von T.

Jetzt wird klar, warum wir $<_v$ als Wohlordnung voraussetzen. Ein Knoten v in einem Aufrufbaum kann abzählbar-unendlich viele Kinder besitzen. Wären diese beispielsweise mit einer Ordnung versehen, die zu $(\mathbb{Z}, <)$ isomorph ist, gäbe es kein kleinstes Element und damit wäre nicht klar, welcher Knoten Nachfolger von v im kanonischen Aufrufpfad ist. In einer Wohlordnung existiert zu jeder Teilmenge ein kleinstes Element.

Zur Modellierung des Begriffs der Aufrufsequenz benötigen wir homomorphe

Abbildungen zwischen den Kategorien **ORTree** und **Grph**. Diese lassen sich formal als Morphismen einer geeigneten Komma-Kategorie [Mac97, II.6.] definieren. Zunächst müssen wir zwei geeignete Funktoren einführen.
Da geordnete Wurzelbäume als Wurzelbäume angesehen werden können, wenn man von der Ordnungsstruktur absieht, gibt es einen Vergissfunktor

$$U : \mathbf{ORTree} \longrightarrow \mathbf{RTree}$$

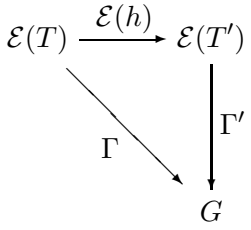
in die Kategorie **RTree** der Wurzelbäume. Morphismen in **RTree** sind dabei alle Graphmorphismen, die das Wurzelelement respektieren. Außerdem hat man einen Einbettungsfunktor *emb*, der einen Wurzelbaum als Objekt in der Kategorie **Grph** der gerichteten Graphen auffasst, wobei das Wurzelelement zum ausgezeichneten Element wird.
Sei $\mathfrak{S} = (G, p, \tau) \in |\mathcal{C}\mathbf{Syst}_n|$. Für die Modellierung der Aufrufsequenzen verwenden wir den Spezialfall $\mathcal{E} \downarrow G$ einer Komma-Kategorie [Mac97, II.6.], wobei

$$\mathcal{E} : \mathbf{ORTree} \xrightarrow{U} \mathbf{RTree} \xrightarrow{emb} \mathbf{Grph}$$

die Komposition des Vergiss-Funktors und der Einbettung ist. Die Objekte von $\mathcal{E} \downarrow G$ heißen *Aufrufsequenzen*. Ist $\Gamma : A \to G$ eine Aufrufsequenz, so heißt A *Aufrufbaum*. Wir fassen dies zusammen in folgender

Definition 5.5 *Die Kategorie* $\mathbf{Call}_{\mathfrak{S}}$ *der Aufrufsequenzen eines Schichtensystems* $\mathfrak{S} = (G, p, \tau)$ *ist die Komma-Kategorie* $\mathcal{E} \downarrow G$.

Ein Objekt von $\mathbf{Call}_{\mathfrak{S}}$ ist mithin ein geordneter Wurzelbaum T zusammen mit einem Graph-Homomorphismus $\Gamma : \mathcal{E}(T) \to G$. Morphismen zwischen $\Gamma : \mathcal{E}(T) \to G$ und $\Gamma' : \mathcal{E}(T') \to G$ $\mathbf{Call}_{\mathfrak{S}}$ sind alle Homomorphismen $h : T \to T'$ zwischen den Bäumen T und T', die mit den Aufrufsequenzen verträglich sind. Es muss also gelten:



.

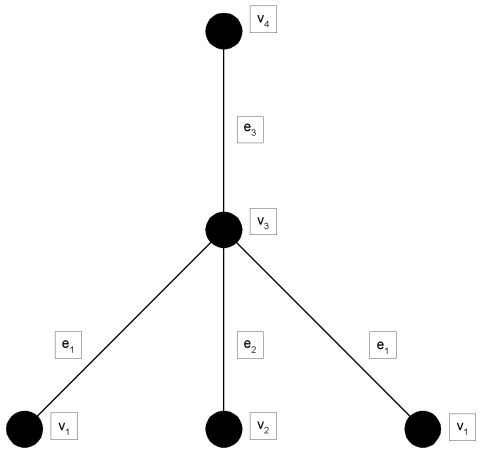


Abbildung 5.5: Ein möglicher Aufrufbaum für das Beispielsystem aus Abbildung 5.2. Die Bilder des Homomorphismus Γ werden in Kästchen direkt an den Baum notiert.

Dabei muss Γ ein Homomorphismus von *gerichteten* Graphen sein, also die Orientierung der Kanten erhalten. Im folgenden lassen wir den Funktor $\mathcal{E}$ weg und identifizieren einen Baum $T \in \mathbf{ORTree}$ mit seiner Einbettung in **Grph**.
Der kanonische Überdeckungspfad von T wird durch die Aufrufsequenz $\Gamma : T \to G$ auf einen Pfad im Systemgraphen abgebildet. Dieser gibt an, in welcher Reihenfolge die Komponenten des Schichtensystems durchlaufen werden.

Definition 5.6 *Sei $\Gamma : T \to G$ eine Aufrufsequenz. Dann definieren wir den* kanonischen Aufrufpfad *als das Bild*

$$\Big(\Gamma(v^{(1)}),\ \Gamma(e^{(1)}),\ \Gamma(v^{(2)}),\ \dots\ ,\ \Gamma(e^{(n)}),\ \Gamma(v^{(n+1)})\Big)$$

das kanonischen Überdeckungspfades von T unter Γ.

In der graphischen Darstellung eines Aufrufbaumes zeichnen wir die Wurzel als obersten Knoten und die Unterbäume gemäß ihrer Ordnung von links

nach rechts. Die Bilder unter Γ schreiben wir in Kästchen direkt an die Knoten und Kanten des Baums, wie in Abbildung 5.5 zu sehen.
Die Bäume repräsentieren eine globale Kontrolle des Aufrufs. Dadurch ist es für unsere formale Konstruktion nicht nötig, einen Stack von offenen Unteraufrufen einzuführen. Die aktive Komponente entscheidet über den nächsten Aufruf, und die globale Kontrolle stellt sicher, dass keine unzulässigen Sprünge im Kontrollfluss auftreten, wie etwa der direkte Übergang der Aktivität von einem Blatt zur Wurzel unter Auslassung dazwischenliegender Komponenten.

5.5 Iterative Berechnung von Aufrufsequenzen

Eine formale Definition der Berechnung einer Aufrufsequenz geben wir nun. Sei $\mathfrak{S} = (G, p, \tau)$ ein Schichtensystem, $v \in G_V$ eine Komponente und $f \in E_v^{out}$ ein Konnektor, der v mit einer Komponente v' verbindet. $\mathfrak{S}$ sei implementiert mit implementierenden Morphismen

$$F_e\colon \Big(\tau_{in}(e) + \coprod_{f \in E^{out}_{tar(e)}} \tau_{out}(f)\Big) \times U_v \longrightarrow \Big(\tau_{out}(e) + \coprod_{f \in E^{out}_{tar(e)}} \tau_{in}(f)\Big) \times U_v$$

zu jedem $e \in E_c^{in}$ und $c \in G_V$.
Wir definieren die Aufrufsequenz zu einem Aufruf der Komponente v' über einen Konnektor f mit Hilfe einer Transformation auf $\mathbf{Call}_{\mathfrak{S}}$, die iteriert angewendet wird. Ausgehend vom Startelement $\Gamma^{(0)}$ konstruieren wir durch mehrfache Anwendung der Transformation $\Gamma^{(j)} \mapsto \Gamma^{(j+1)}$ die Aufrufsequenz $\Gamma^{(n)}$. Dabei bezeichnet n den Index, für den die Iteration terminiert. Die Transformation operiert auf Aufrufsequenzen $\Gamma^{(j)} : A^{(j)} \to G$ mit zwei ausgezeichneten Elementen: einem Knoten $a^{(j)} \in T_V^{(j)}$ und einem Konnektor $i^{(j)} \in T_E^{(j)}$. Dabei fordern wir stets $a^{(j)} \in \{src(i^{(j)}), tar(i^{(j)})\}$. Anhand der Elemente $a^{(j)}$ und $i^{(j)}$ können wir unterscheiden, ob im j-ten Iterationsschritt ein neuer (Unter-)Aufruf erzeugt oder ein Aufruf beantwortet wird, und an welcher Stelle dies geschieht.
Da die Bäume $A^{(j)}$ nur Hilfskonstruktionen sind, gehen wir für eine inhaltliche Interpretation der beiden Elemente $a^{(j)}, i^{(j)}$ zu ihren Bildern $\Gamma^{(j)}(a^{(j)})$ und $\Gamma^{(j)}(i^{(j)})$ über und interpretieren diese Elemente als die aktuell *aktive Komponente* und den von dieser *angesprochenen Konnektor*. Angesprochen

kann hier sowohl bedeuten, dass die Komponente $\Gamma^{(j)}(a^{(j)})$ über den Konnektor eine andere Komponente aufruft, als auch, dass sie darüber einen Rückgabewert übermittelt.
Wir gehen iterativ vor und konstruieren ausgehend vom Startelement $\Gamma^{(0)}$ durch Transformationen $\Gamma^{(j)} \mapsto \Gamma^{(j+1)}$ die Aufrufsequenz $\Gamma^{(n)}$. Als Startelement verwenden wir die Aufrufsequenz $\Gamma^{(0)} : A^{(0)} \to G$, wobei $A^{(0)}$ der Baum mit Wurzel w und einem einzigen Blatt u ist (siehe Abbildung 5.6):

$$A^{(0)}{}_V = \{w, u\},$$

$$A^{(0)}{}_E = \{h\},$$

mit $w \neq u$ und $src(h) = w$, $tar(h) = u$. Die Aufrufsequenz $\Gamma^{(0)} : A^{(0)} \to G$ definieren wir auf den Knoten durch $\Gamma^{(0)}(w) = v$ und $\Gamma^{(0)}(u) = tar(f)$. Wegen der Homomorphie-Eigenschaft von $\Gamma^{(0)}$ ist klar, dass $\Gamma^{(0)}(h) = f$ sein muss. Wir setzen $a^{(0)} := w$ und $i^{(0)} := h$. Zu Beginn ist also Komponente $v = \Gamma^{(0)}(a^{(0)})$ aktiv und f der Konnektor, der als nächster angesprochen wird. Eine Eingabe $x^{(0)} : I \to \tau_{in}(f)$ sei gegeben. Jede Komponente v von G befinde sich im Startzustand $u_v^{(0)}$.
Mithilfe der nun zu definierenden Transformation bauen wir iterativ die Aufrufsequenz auf. Dabei stellen die Zwischenschritte den Zustand des Systems während der Verarbeitung des Aufrufs dar.

Wir benötigen eine Hilfsgröße $r^{(j)}$, die sich jedoch direkt aus $a^{(j)}$ und $i^{(j)}$ berechnen lässt. In Abhängigkeit davon, ob im aktuellen Schritt ein neuer Aufruf gestartet wird oder die Antwort auf einen Aufruf übermittelt wird, muss der Konnektor $e^{(j)}$ bestimmt werden, dessen implementierender Morphismus $F_{e^{(j)}}$ für den nächsten Schritt zuständig ist (vgl. Bemerkung 5.4). Da wir für die Konstruktionen die Elemente des Baumes $A^{(j)}$ benötigen, bestimmen wir den Repräsentanten $r^{(j)}$ im Baum, also diejenige Kante, für die $\Gamma(r^{(j)}) = e^{(j)}$ ist. Wir unterscheiden zwei Fälle:

(a) $a^{(j)} = src(i^{(j)})$, d.h. Komponente $\Gamma(a^{(j)})$ stellt einen neuen Aufruf über den Konnektor $\Gamma(i^{(j)})$. Hier setzen wir $r^{(j)} := i^{(j)}$.

(b) $a^{(j)} = tar(i^{(j)})$, d.h. Komponente $\Gamma(a^{(j)})$ leitet eine Antwort via Konnektor $\Gamma(i^{(j)})$ an $\Gamma(src(i^{(j)}))$. Diese Komponente befindet sich in der Bearbeitung des Aufrufs, die sie selbst über den Konnektor $\Gamma(M(i^{(j)}))$ erhalten

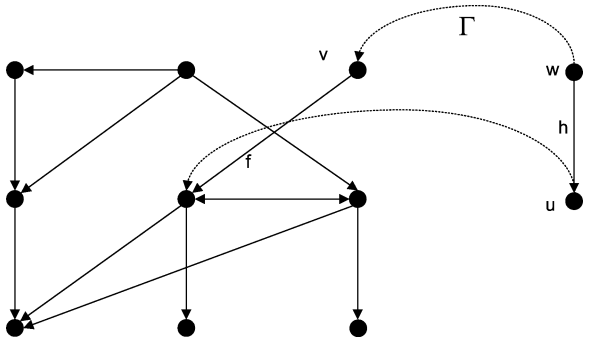


Abbildung 5.6: Starthomomorphismus $\Gamma^{(0)} : A^{(0)} \to G$. Rechts ist der Startbaum $A^{(0)}$ mit dem Wurzelelement w und dem Knoten u gezeichnet und links der Systemgraph G. Die punktierten Bögen zeigen an, wie Γ die Knoten von $A^{(0)}$ nach G abbildet.

hat, wobei $M(i^{(j)})$ die Mutterkante von $i^{(j)}$ bezeichnet. Daher setzen wir $r^{(j)} := M(i^{(j)})$.

Für das Startelement $\Gamma^{(0)} : A^{(0)} \to G$ ergibt sich offensichtlich $r^{(0)} = i^{(0)} = h$ und $e^{(0)} = \Gamma^{(0)}(r^{(0)}) = f$ (da ein neuer Aufruf vorliegt).

Nun definieren wir den Iterationsschritt. Jede Komponente v von G befinde sich im Zustand $u_v^{(j)}$. Die Aufrufsequenz $\Gamma^{(j)} : A^{(j)} \to G$ sei bereits gegeben, $\Gamma^{(j)}(a^{(j)})$ sei die aktive Komponente und $\Gamma^{(j)}(i^{(j)})$ der angesprochene Konnektor.
Wir berechnen zunächst $r^{(j)}$ nach der soeben erklärten Vorschrift und setzen $e^{(j)} := \Gamma^{(j)}(r^{(j)})$ sowie

$$g^{(j)} := (x^{(j)}, u^{(j)}_{tar(e^{(j)})}),$$

$$\mathbb{F}^{(j)} := F_{e^{(j)}} \circ \widetilde{g^{(j)}} : I \to \Big(\tau_{out}(e^{(j)}) + \coprod_{d \in E^{out}_{tar(e^{(j)})}} \tau_{in}(d)\Big) \times U_{tar(e^{(j)})}.$$

Die Faktorisierung dieses globalen Elements wird im folgenden entscheidend sein. Bei der Konstruktion von $\Gamma^{(j+1)} : A^{(j+1)} \to G$ aus $\Gamma^{(j)} : A^{(j)} \to G$

unterscheiden wir zwei Fälle[6]:

1. *$proj_1 \circ \mathbb{F}^{(j)}$ faktorisiert durch $\tau_{out}(e^{(j)})$.*

   In diesem Fall ist $A^{(j+1)} = A^{(j)}$ und $\Gamma^{(j+1)} = \Gamma^{(j)}$. Wir setzen $i^{(j+1)} := i^{(j)}$ und $a^{(j+1)} := tar(i^{(j)})$.

2. *$proj_1 \circ \mathbb{F}^{(j)}$ faktorisiert durch eines der $\tau_{in}(d)$ mit $d \in E^{out}_{tar(e)}$.*

   In diesem Fall werden ein neuer Knoten u' und eine neue Kante e' mit $tar(e^{(j)}) = src(e')$ und $tar(e') = u'$ eingefügt, also $A_V^{(j+1)} = A_V^{(j)} \dot\cup \{u'\}$ und $A_E^{(j+1)} = A_E^{(j)} \dot\cup \{e'\}$. Wir setzen $\Gamma^{(j+1)}(e') = d$, außerdem $i^{(j+1)} := e'$ und $a^{(j+1)} := u'$.

In beiden Fällen gelte:

- Der neue Zustand der Komponente $tar(e^{(j)})$ ist $u^{(j+1)}_{tar(e^{(j)})} := proj_2 \circ \mathbb{F}^{(j)}$. Die Zustände aller anderen Komponenten bleiben unverändert.
- Das globale Element $x^{(j+1)} = proj_1 \circ \mathbb{F}^{(j)}$ dient als Eingabe für den nächsten Schritt

Die beschriebene Transformation wird auf den Starthomomorphismus $\Gamma^{(0)} : A^{(0)} \to G$ iteriert angewendet. Sie terminiert, falls eine Aufrufsequenz $\Gamma^{(n)} : A^{(n)} \to G$ entsteht, für die gilt:

$$\Gamma^{(n)}(i^{(n)}) = f \quad \text{und} \quad \Gamma^{(n)}(a^{(n)}) = tar(f).$$

Diese Bedingung besagt, dass die Antwort auf den ursprünglichen Aufruf von f übermittelt wird. Die *Aufrufsequenz zur Eingabe x* ist dann $\Gamma^{(n)} : A^{(n)} \to G$. Das entstandene globale Element $x^{(n)} : I \to \tau_{out}(f)$ bezeichnen wir mit $\bar{x}$.

Beispiel 5.7 Die Berechnung einer exemplarischen Aufrufsequenz ist in Abbildung 5.7 gezeigt. Komponente v der obersten Schicht ruft über den Konnektor f eine Komponente der mittleren Schicht auf. Im rechten Bereich von Schritt j ist jeweils der Baum $A^{(j)}$ zu sehen, links der Systemgraph G. Die gestrichelten Pfeile geben an, wie der Homomorphismus $\Gamma^{(j)}$ auf den Knoten

[6] Dies sind genau die beiden Fälle, die bereits in Abschnitt 5.2 erörtert wurden.

operiert; auf den Kanten ist er dadurch eindeutig bestimmt, da G in diesem Beispiel keine Mehrfachkanten besitzt.

In Schritt 0 sieht man den Startgraphen $A^{(0)}$ und die Aufrufsequenz $\Gamma^{(0)}$. Die in diesem Schritt aktive Komponente ist v, ihr Repräsentant $a^{(0)}$ im Baum ist w. Der angesprochene Konnektor ist f, er wird durch h repräsentiert, daher setzen wir $i^{(0)} := h$. Wie stets im Startelement ist $r^{(0)} = i^{(0)}$, und damit $f = \Gamma^{(0)}(r^{(0)})$ wie vorgesehen der Konnektor, der die Eingabe übernimmt.

In Schritt 1 ist die mittlere Schicht aktiv, die aktive Komponente wird durch $a^{(1)}$ repräsentiert. Wir nehmen an, dass ein Subaufruf an eine Komponente der untersten Schicht übermittelt wird. Der angesprochene Konnektor der untersten Schicht wird durch $i^{(1)}$ repräsentiert. Wieder ist $r^{(1)} = i^{(1)}$, da es sich um einen neuen Aufruf handelt.

In Schritt 2 ist Komponente $\Gamma^{(2)}(a^{(2)})$ der untersten Schicht aktiv. Sie gibt die Antwort auf den Aufruf zurück. Daher ist der angesprochene Konnektor derselbe wie im vorherigen Schritt, also $i^{(2)} = i^{(1)}$, und wir haben jetzt $r^{(2)} = M(i^{(2)}) = h$.

In Schritt 3 wird ein weiterer Unteraufruf an eine andere Komponente der untersten Schicht gestellt. Der angesprochene Konnektor der untersten Schicht wird durch $i^{(3)}$ repräsentiert und es ist $r^{(3)} = i^{(3)}$, da ein neuer Aufruf stattfindet.

Schritt 4 verläuft analog zu Schritt 2: Es ist $i^{(4)} = i^{(3)}$ und $a^{(4)} = tar(i^{(4)})$. Außerdem haben wir $r^{(4)} = M(i^{(4)}) = h$.

In Schritt 5 kann Komponente a den Aufruf von w beantworten. Es ist $a^{(5)} = u$ und $i^{(5)} = h$, damit ist das Terminierungskriterium erfüllt.

Eine inhaltliche Interpretation für dieses Beispiel geben wir in Abschnitt 5.7.

Definition 5.8 *Sei $\mathfrak{S} = (G, p, \tau)$ ein implementiertes $\mathcal{C}$-Schichtensystem im initialen Zustand.*

(a) Entsteht aus einer Eingabe $x : I \to \tau_{\text{in}}(e)$ ein globales Element $\bar{x} : I \to \tau_{\text{out}}(e)$, so heißt $\bar{x}$ die resultierende Arbeit *von e für x.*

(b) Ist $\mathcal{A} : \tau_{\text{in}}(e) \to \tau_{\text{out}}(e)$ ein Morphismus, so dass für alle globalen Elemente $x : I \to \tau_{\text{in}}(e)$ gilt: $\bar{x} = \mathcal{A} \circ x$, dann heißt $\mathcal{A}$ die Gesamtarbeit

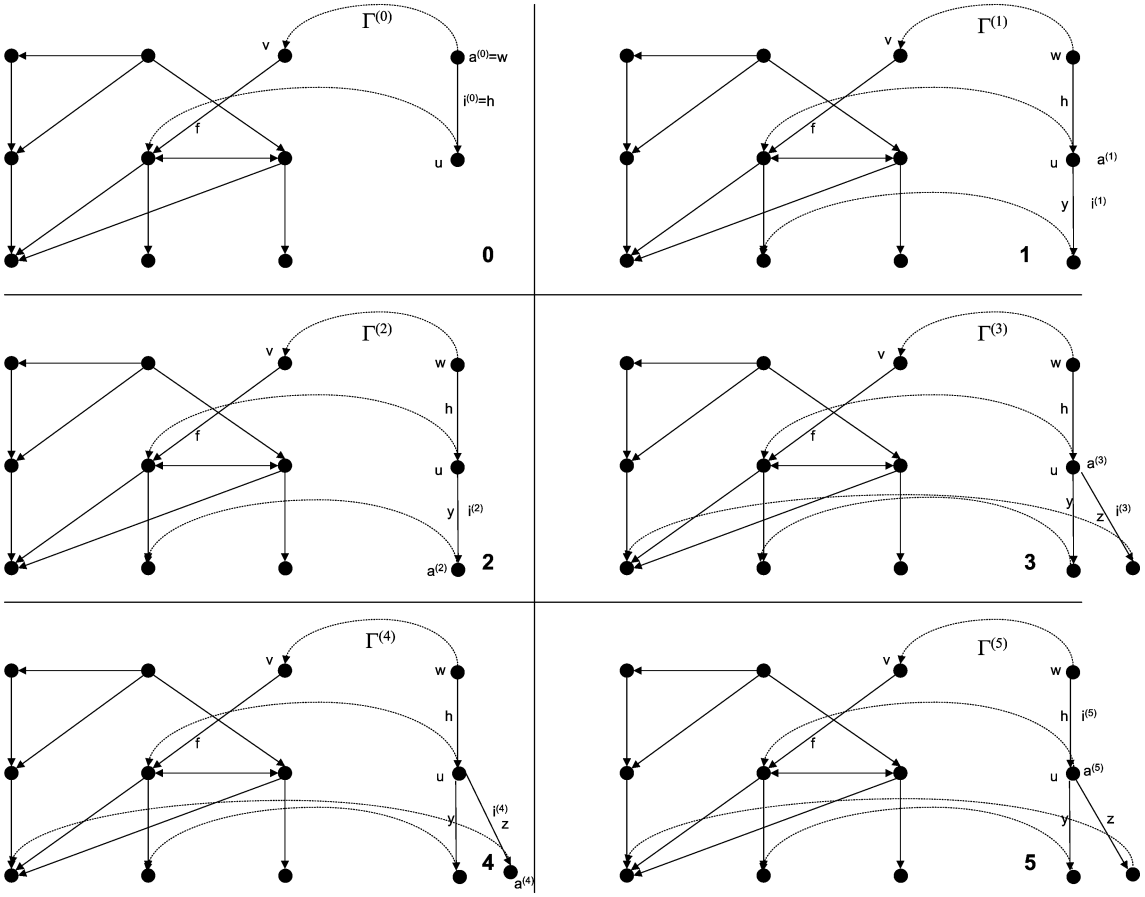
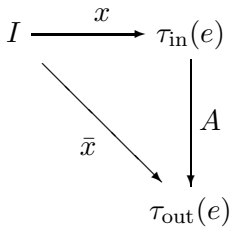


Abbildung 5.7: Schritte zur Berechnung der Aufrufsequenz

von e.



(c) Sind zwei verschiedene Implementierungen eines Schichtensystems gegeben und ist für jeden Aufruf die Gesamtarbeit in beiden Implementierungen die gleiche, so heißen die beiden implementierten Schichtensysteme dynamisch äquivalent.

Bemerkung 5.9 Wenn die Folge der $\Gamma^{(j)}$ für eine Eingabe x nicht terminiert, so existiert die resultierende Arbeit für x nicht. Damit existiert auch die Gesamtarbeit des entsprechenden Konnektors nicht. Im nächsten Abschnitt werden wir Terminierungsfragen näher untersuchen.

Beispiel 5.10 Man betrachte das System aus Abbildung 5.2. Für F_{e_3} übernehmen wir die Implementierung vom 1. Fall aus Kapitel 5.3. Weiter definieren wir $F_{e_1} : \mathbb{N} \to \mathbb{Z}$ als $F_{e_1}(n) = n!$ und $F_{e_2}(n) = 2^n$. Dann ist die Gesamtarbeit von e_3 gegeben durch

$$A(z) = \begin{cases} z! & \text{für} \quad z \geq 0, \\ 2^{-z} & \text{für} \quad z < 0. \end{cases}$$

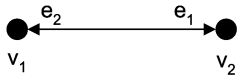
Ein dynamisch äquivalentes System erhielte man z.B. durch Vertauschung der Rollen von e_1 und e_2.

5.6 Terminierung

Die Terminierung der iterativen Berechnung des Aufrufbaums eines Aufrufs ist im allgemeinen nicht garantiert, wie wir in Abschnitt 5.3 gesehen haben. Dort ist der entstehende Aufrufbaum zwar in der Tiefe beschränkt, verzweigt jedoch unendlich. Auch Bäume unendlicher Tiefe sind möglich. Sie entstehen, wenn zwei Komponenten derselben Schicht einander im Wechsel aufrufen.

Beispiel 5.11:

Sei $G = (G_V, G_E)$ der Graph mit zwei Knoten $G_V = \{v_1, v_2\}$ und zwei Kanten $G_E = \{e_1, e_2\}$, für die $src(e_1) = tar(e_2) = v_1$ und $src(e_2) = tar(e_1) = v_2$ ist.



Es gelte $p(v_1) = p(v_2) = 1$. Abbildungen $\tau_{in}, \tau_{out} : E \to |\mathcal{C}|$ seien gegeben. Für

$$F_{e_1} : \tau_{\text{in}}(e_1) + \tau_{\text{out}}(e_2) \to \tau_{\text{out}}(e_1) + \tau_{\text{in}}(e_2)$$

gelte: $F_{e_1} \circ \tilde{x}$ faktorisiert für jedes $x : I \to \tau_{\text{in}}(e_1)$ durch $\tau_{\text{in}}(e_2)$. Ebenso faktorisiere die Komposition von

$$F_{e_2} : \tau_{\text{in}}(e_2) + \tau_{\text{out}}(e_1) \to \tau_{\text{out}}(e_2) + \tau_{\text{in}}(e_1)$$

und $\tilde{x}$ für jedes $x : I \to \tau_{\text{in}}(e_2)$ durch $\tau_{\text{in}}(e_1)$. Die Komponenten v_1 und v_2 rufen einander also stets gegenseitig auf. Jeder Aufruf von v_1 läuft unendlich oft zwischen v_1 und v_2 hin und her und es ergibt sich ein Aufrufbaum *unendlicher Tiefe*.

In einem wichtigen Spezialfall können wir die Terminierung allerdings zeigen. Wir verlangen von den Komponenten, dass sie zur Beantwortung eines Aufrufs, der ihnen über einen ihrer Konnektoren übermittelt wird, nur eine endliche Zahl von Unteraufrufen stellen. Dann können wir zeigen, dass jeder Aufruf an $\mathfrak{S}$ zu einem endlichen Aufrufbaum führt, mithin, dass die Berechnung der Aufrufsequenz nach endlicher Iteration terminiert. Wir gehen dabei von der zusätzlichen Voraussetzung einer *strengen Aufrufhierachie* aus, in der ein Konnektor eine Komponente mit einer anderen nur dann verbinden darf, wenn diese in einer tieferen Schicht liegt als jene. Die surjektive Schichtenabbildung $p : V \to \{1, ..., n\}$ erfülle also $p(\text{src}(e)) > p(\text{tar}(e))$ für alle $e \in G_E$.

Satz 5.12 *Jede Komponente v einer strengen Schichtenarchitektur $\mathfrak{S}$ erfülle folgende Annahme: Jeder Aufruf von v wird nach einer endlichen Anzahl von Subaufrufen beantwortet. Dann existiert zu jedem Aufruf ein endlicher Aufrufbaum.*

Beweis: Die endliche Zahl der Schichten und die strenge Aufrufhierarchie begrenzen die Tiefe des Aufrufbaums.
Wegen der strengen Aufrufhierachie ist der wechselseitige Aufruf zweier Komponenten ausgeschlossen. Die endliche Zahl der Schichten gewährleistet daher, dass der Aufrufbaum von endlicher Tiefe ist. Unendliche Verzweigungen sind durch die Annahme einer endlichen Zahl von Subaufrufen ausgeschlossen. Ein Baum endlicher Tiefe mit endlicher Verzweigung ist endlich. Daher terminiert die Transformation $\Gamma^{(j)} \mapsto \Gamma^{(j+1)}$ nach endlich vielen Schritten.□

Die Forderung, dass jede Komponente nur eine endliche Zahl von Subaufrufen zur Beantwortung eines Aufrufs stellen darf, ist wegen des Halteproblems [LP97, Sect. 5.3] sicher nicht algorithmisch nachprüfbar. Dass wir Terminierungsresultate nur unter derart starken Annahmen erhalten, ist vor dem Hintergrund der Mächtigkeit des Ansatzes jedoch nicht überraschend, denn wir lassen beliebige Morphismen in der jeweiligen Kategorie zu. Die nichtterminierende Zusammenstellung von einzeln terminierenden Komponenten, die in unserem Modell möglich ist, spiegelt den auch in der Praxis vorkommenden Fall wider, dass Komponenten einander durch gegenseitigen Aufruf blockieren und ein Programm daher nicht terminiert. Bereits ein einfaches Beispiel verdeutlicht die Schwierigkeiten, mit denen man bei der Analyse solcher Systeme konfrontiert ist. Man betrachte das System in Abbildung 5.8. Wir setzen

$$F_{e_1} : \mathbb{N} \to \mathbb{N}$$

$$F_{e_1}(n) = \begin{cases} n/2 & \text{falls} \quad n \quad \text{gerade,} \\ 3n+1 & \text{falls} \quad n \quad \text{ungerade} \end{cases}$$

und

$$F_{e_2} : \mathbb{N} + \mathbb{N} \to \mathbb{N} + \mathbb{N}$$

$$F_{e_2}(n) = \begin{cases} i_1(n) & \text{falls} \quad n = 1, \\ i_2(n) & \text{falls} \quad n \neq 1. \end{cases}$$

Dabei bezeichnet $i_k : \mathbb{N} \to \mathbb{N} + \mathbb{N}$ für $k = 1, 2$ die $k$te Injektion in das Coprodukt. Das System terminiert genau dann, wenn die Folge $x^{(j)}$ der Berechnungsergebnisse den Wert 1 annimmt. Die Frage, ob das System für jeden

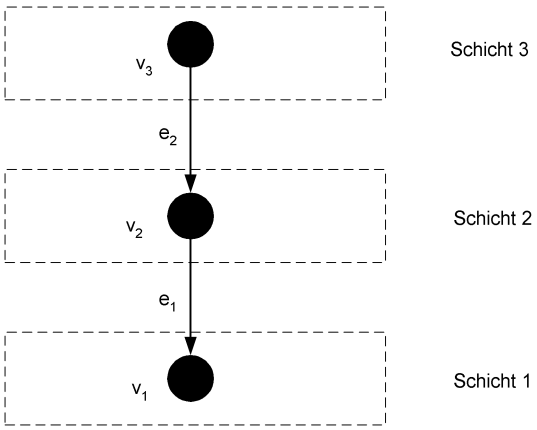


Abbildung 5.8: System mit drei Komponenten

Aufruf von e_2 mit $n \in \mathbb{N}$ als Eingabe terminiert, ist das bislang ungelöste $3n+1$-*Problem* [Lag97]. Es existiert also nicht nur kein allgemeines Verfahren zur Entscheidung, ob ein gegebenes System terminiert, sondern wir können sogar ein konkretes Beispielsystem geben, für das die Terminierungsfrage bis dato ungeklärt ist.

Existenz der Gesamtarbeit

Nachdem wir im vorherigen Abschnitt iterativ die resultierende Arbeit $\bar{x}$ für einzelne Eingaben x an einen Konnektor e in einem $\mathcal{C}$-Schichtensystem $\mathfrak{S}$ bestimmt haben, untersuchen wir nun die Frage, wie und unter welchen Voraussetzungen die Gesamtarbeit von e als Morphismus $\mathcal{A} : \tau_{\text{in}}(e) \to \tau_{\text{out}}(e)$ darstellbar ist. Der Existenz eines Morphismus $\mathcal{A}$, so dass $\mathcal{A} \circ x = \bar{x}$ für alle Eingaben x ist, stehen a priori mehrere mögliche Hindernisse entgegen:

(1) Das System $\mathfrak{S}$ muss nicht für alle Eingaben terminieren.

(2) Das iterative Verfahren zur Bestimmung der resultierenden Arbeit ordnet den Eingaben in einer solchen Weise Ausgaben zu, dass dies keinem Morphismus in $\mathcal{C}$ entspricht.

(3) Schließlich ist es in nicht-wohlpunktierten Kategorien denkbar, dass $\mathcal{A}$ nicht eindeutig bestimmt ist.

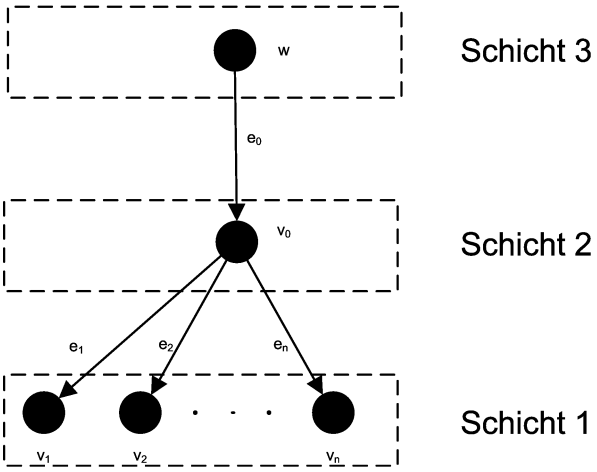


Abbildung 5.9: Schichtensystem

Wir berücksichtigen diese Punkte wie folgt: Die Terminierung ist eine notwendige Bedingung, um die Gesamtarbeit $\mathcal{A}$ zu bestimmen, deswegen wollen wir die Voraussetzungen von Satz 5.12 im folgenden generell fordern. Um sicherzustellen, dass wir keine Zuordnungen konstruieren, die der Morphismenbegriff in der Kategorie nicht abdeckt, konstruieren wir die Gesamtarbeit vollständig innerhalb der Kategorie, durch geeignete Komposition von Morphismen. Dies führt im folgenden zu recht komplexen Zusammenstellungen von Projektionen und Vertauschungen von Faktoren in Produkten. Die notwendige Fallunterscheidung zwischen den Eingaben, deren Verarbeitung verschieden strukturierte Aufrufbäume nach sich ziehen, treffen wir dadurch, dass wir die Menge aller Eingaben in ein Coprodukt zerlegen und auf jedem Cofaktor die Gesamtarbeit bestimmen. Der vermittelnde Coprodukt-Morphismus liefert dann die Gesamtarbeit für alle Eingaben. Der dritte Punkt ist bei diesem Vorgehen wegen der Eindeutigkeit (bis auf Isomorphie) des Coprodukt-Morphismus automatisch mit erledigt.
Um die Betrachtung nicht zu kompliziert werden zu lassen, untersuchen wir zunächst exemplarisch das in Abbildung 5.9 gezeigte System und führen danach beliebige strenge Schichtensysteme auf dieses zurück. Die Zustandsräume der Komponenten v_i seien durch Objekte U_i $(i = 0, \ldots, n)$ gegeben. Wir schreiben im folgenden kurz F_i statt F_{e_i} für die implementierenden Morphismen und wählen die Objekte $\tau_{in}(e_i)$ und $\tau_{out}(e_i)$ so, dass gilt:

$$F_0 : \Big(C + \coprod_{i=1}^{n} B_i\Big) \times U_0 \to \Big(D + \coprod_{i=1}^{n} A_i\Big) \times U_0,$$

$$F_i : A_i \times U_i \to B_i \times U_i \quad (i = 1, \ldots, n).$$

Ziel ist es, die Gesamtarbeit $\mathcal{A} : C \to D$ des Konnektors e_0 zu bestimmen – so sie existiert. Dabei denken wir uns Zustände $u_0 : I \to U_0$, …, $u_n : I \to U_n$ als gegeben, beispielsweise durch die Startzustände der Komponenten. Um die Veränderung der Zustände während der Berechung zu berücksichtigen, bilden wir in jedem Berechnungschritt das Produkt $\prod_{i=0}^{n} U_i$ aller Zustandsräume mit ab. Wir konstruieren daher zunächst einen Morphismus $\mathcal{A}^{\mathcal{U}} : \prod_{i=0}^{n} U_i \times C \to \prod_{i=0}^{n} U_i \times D$, aus dem wir mithilfe der Startzustände später $\mathcal{A}$ gewinnen.

Den Vorbereich von $\mathcal{A}^{\mathcal{U}}$ zerlegen wir in Objekte $M_{i_1 \ldots i_k}$ mit $i_1, \ldots, i_k \in \{1, \ldots, n\}$ und $k \in \mathbb{N}_0$. Dabei lassen wir auch $k = 0$ zu und schreiben dafür M_0. Die Idee ist, dass M_0 diejenigen Aufrufe repräsentiert, die direkt von F_0 beantwortet werden, ohne dass es zu einem Unteraufruf kommt. Mit M_i für $1 \leq i \leq n$ kennzeichnen wir die Aufrufe von e_0, die genau einen Unteraufruf von e_i zu ihrer Beantwortung benötigen. Analog steht dann $M_{i_1 \ldots i_k}$ für alle Aufrufe, die zunächst einen Unteraufruf von e_{i_1} nach sich ziehen, dann von e_{i_2}, usw., und schließlich als letzten Unteraufruf vor der Beantwortung des Aufrufs den Konnektor e_{i_k} ansprechen.

Technisch nutzen wir zur Durchführung der genannten Zerlegung die Tatsache aus, dass Coprodukte in lextensiven Kategorien universell sind, d.h. eine Coprodukt-Zerlegung $Y_1 + Y_2$ des Nachbereichs Y eines Morphismus $f : X \to Y$ sich in eine Coprodukt-Zerlegung $X_1 + X_2$ des Vorbereichs X „zurückziehen“[7] lässt. Wir gehen induktiv vor und definieren erst M_0, dann den Induktionsschritt $M_{i_1 \ldots i_k} \mapsto M_{i_1 \ldots i_k j}$.

Zunächst bestimmen wir also mit M_0 diejenigen Aufrufe, die keine Unteraufrufe nach sich ziehen. Dazu definieren wir f als folgende Komposition von Morphismen:

$$f : \prod_{i=0}^{n} U_i \times C \xrightarrow{proj_0 \times id_C} U_0 \times C \xrightarrow{twist} C \times U_0 \xrightarrow{inj} \Big(C + \coprod_{i=1}^{n} B_i\Big) \times U_0 \xrightarrow{F_0} \Big(D + \coprod_{i=1}^{n} A_i\Big) \times U_0.$$

[7] Hier bedienen wir uns ausnahmsweise der wörtlichen Übersetzung von *pull-back*, da sie eine gute intuitive Vorstellung vermittelt.

Der Morphismus f transformiert das Eingabeobjekt $\prod_{i=0}^{n} U_i \times C$ zunächst durch Projektion, Vertauschung der Faktoren des Produkts *(twist*[8]*)* und Injektion so, dass es an die Signatur von F_0 angepasst wird, und wendet F_0 dann an. Da dabei die Zustände $U_1, \ldots, U_n$ „verloren gehen", müssen wir ein Produkt von Morphismen bilden, das diese mit berücksichtigt. Dies geschieht in Φ_0:

$$\Phi_0 : \prod_{i=0}^{n} U_i \times C \xrightarrow{(proj_1 \times \ldots \times proj_n, f)} \prod_{i=1}^{n} U_i \times \left(D + \coprod_{i=1}^{n} A_i\right) \times U_0 \xrightarrow{twist} \prod_{i=0}^{n} U_i \times \left(D + \coprod_{i=1}^{n} A_i\right)$$

Dabei bezeichnet *twist* wieder die Vertauschung der Faktoren des Produkts. Nun bilden wir ein doppeltes Pullback

$$\begin{array}{ccccc}
M_0 & \xrightarrow{\eta_0} & \prod_{i=0}^{n} U_i \times C & \xleftarrow{\bar{\eta}_0} & \bar{M}_0 \\
\downarrow{\scriptstyle \mathcal{A}_0} & & \downarrow{\scriptstyle \Phi_0} & & \downarrow{\scriptstyle R_0} \\
\prod_{i=0}^{n} U_i \times D & \xrightarrow[i_1]{} & \prod_{i=0}^{n} U_i \times \left(D + \coprod_{i=1}^{n} A_i\right) & \xleftarrow[i_2]{} & \prod_{i=0}^{n} U_i \times \coprod_{i=1}^{n} A_i
\end{array}$$

Wir interpretieren M_0 als das Objekt der Aufrufe, die von e_0 sofort beantwortet werden, und $\bar{M}_0$ als ihr Coprodukt-Komplement, denn wegen der Universalität des Coprodukts ist auch die obere Zeile ein Coprodukt. Der Morphismus $\mathcal{A}_0 : M_0 \to \prod_{i=0}^{n} U_i \times D$ ist der Anteil der Gesamtarbeit, der auf die sofort beantworteten Aufrufe entfällt. Der Morphismus $R_0 : \bar{M}_0 \to \prod_{i=0}^{n} U_i \times \coprod_{i=1}^{n} A_i$ stellt den ersten Berechnungsschritt der übrigen Aufrufe dar.

Nachdem wir mit M_0 den Induktionsanfang gemacht haben, definieren wir nun induktiv die Objekte $M_{i_1 \ldots i_k}$ und $\bar{M}_{i_1 \ldots i_k}$ für $i_1, \ldots, i_k \in$

[8] Wir lehnen unsere Notation an [Wal91, Ch. 2, §5] an.

$\{1,\ldots,n\}$ und $k \in \mathbb{N}$. Sie repräsentieren die Aufrufe mit Aufrufpfad $e_0, e_{i_1}, e_0, e_{i_2}, \ldots, e_0, e_{i_k}, e_0$.
Seien also $M_{i_1\ldots i_k}$ und $\bar{M}_{i_1\ldots i_k}$ gegeben. Wir wollen für beliebiges $j \in \{1,\ldots,n\}$ die Objekte $M_{i_1\ldots i_k j}$ und $\bar{M}_{i_1\ldots i_k j}$ definieren. Dazu zerlegen wir $\bar{M}_{i_1\ldots i_k}$ in die Objekte $\bar{M}^j_{i_1\ldots i_k}$ für $j = 1, \ldots n$, die sich aus der Faktorisierung von $R_{i_1\ldots i_k}$ bestimmen. Dies geschieht durch Pullback-Bildung:

$$\begin{array}{ccc}
\bar{M}_{i_1\ldots i_k} & \xleftarrow{\zeta^j_{i_1\ldots i_k}} & \bar{M}^j_{i_1\ldots i_k} \\
\downarrow{\scriptstyle R_{i_1\ldots i_k}} & & \downarrow{\scriptstyle R^j_{i_1\ldots i_k}} \\
\prod_{i=0}^{n} U_i \times \coprod_{i=1}^{n} A_i & \xleftarrow{inj} & \prod_{i=0}^{n} U_i \times A_j
\end{array}$$

Das Objekt $\bar{M}^j_{i_1\ldots i_k}$ repräsentiert diejenigen Eingaben, die als nächstes von e_j verarbeitet werden müssen. Der Morphismus $\zeta^j_{i_1\ldots i_k}$ bettet $\bar{M}^j_{i_1\ldots i_k}$ in $\bar{M}_{i_1\ldots i_k}$ ein. Wegen der Universalität des Coprodukts haben wir

$$\bar{M}_{i_1\ldots i_k} = \coprod_{j=1}^{n} \bar{M}^j_{i_1\ldots i_k}.$$

Im nun folgenden Schritt wird zunächst der Morphismus F_j durchlaufen. Das Ergebnis wird dann in jedem Fall wieder von F_0 verarbeitet, so dass wir diese direkt in einem gemeinsamen Morphismus

$$\Phi^j_{i_1\ldots i_k} : M^j_{i_1\ldots i_k} \to \prod_{i=0}^{n} U_i \times \left(D + \coprod_{i=1}^{n} A_i\right)$$

zusammenfassen. Dazu sind wieder Projektionen und Vertauschungen von Produkt-Faktoren nötig. Die genaue Komposition dieser Morphismen ist aus Abbildung 5.10 ersichtlich. Dabei bezeichnet *twist* jeweils die Vertauschung von Faktoren des Produkts. Um die Notation nicht unnötig zu verkomplizieren, verzichten wir darauf, durch obere und untere Indizes genauer zu bezeichnen, welche Faktoren jeweils vertauscht werden, da dies aus den Objekten im Vor- und Nachbereich eindeutig hervorgeht.
Nun können wir die durch $\Phi^j_{i_1\ldots i_k}$ geleistete Berechnung bestimmen und damit $\bar{M}^j_{i_1\ldots i_k}$ in das Objekt $M_{i_1\ldots i_k j}$ der Eingaben, die in diesem Schritt beantwortet werden, und das Objekt $\bar{M}_{i_1\ldots i_k j}$ der Eingaben, die eine weitere

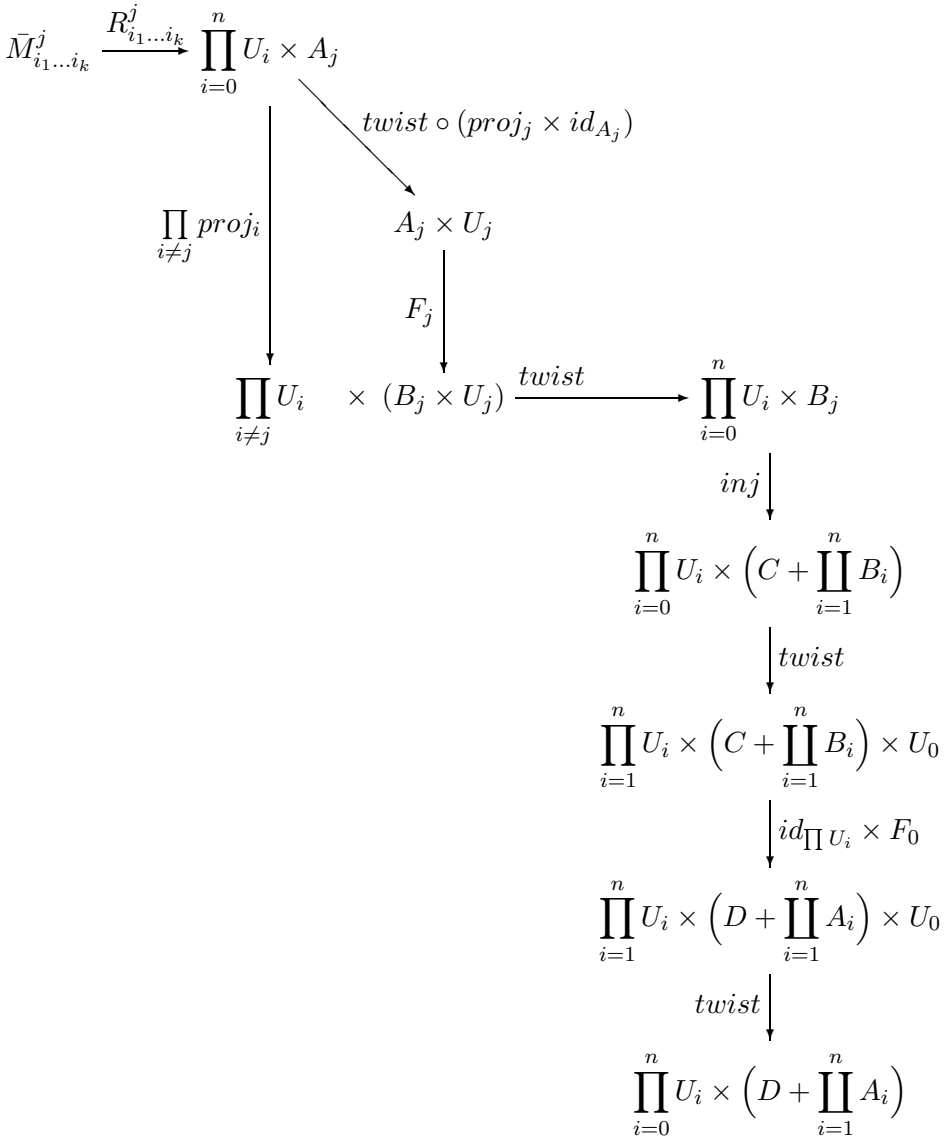
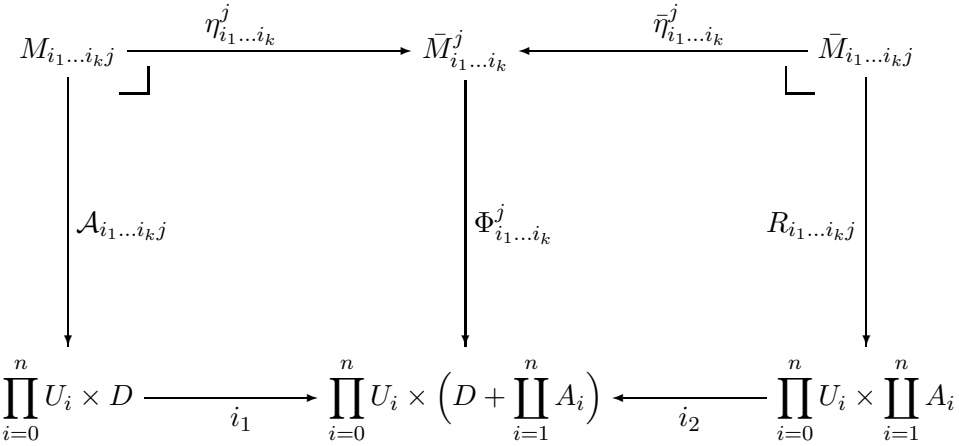


Abbildung 5.10: Der Morphismus $\Phi^j_{i_1\ldots i_k} : M^j_{i_1\ldots i_k} \to \prod_{i=0}^{n} U_i \times (D + \coprod_{i=1}^{n} A_i)$ entsteht als eine Komposition, deren wesentliche Berechnungschritte in den Morphismen F_j und F_0 liegen.

Unteraufrufe generieren, zerlegen. Du diesem Zweck bilden wir abermals ein doppeltes Pullback, das eine Coprodukt-Zerlegung der obersten Zeile liefert:

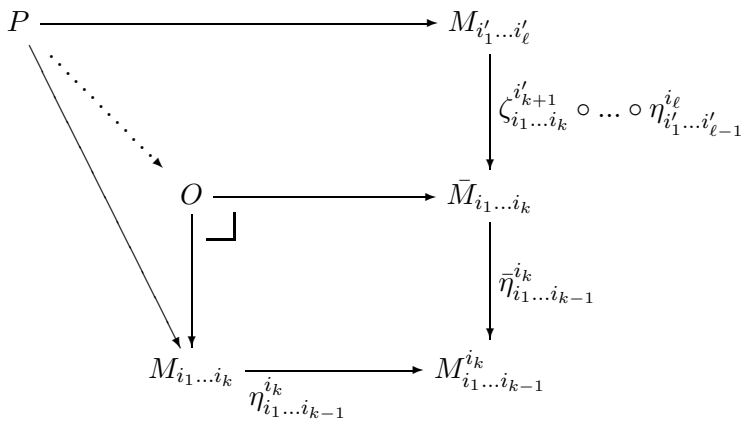


Diese Pullback-Konstruktion interpretieren wir wie folgt: Das Objekt $\bar{M}^j_{i_1\ldots i_k}$ repräsentiert die Aufrufe, die schon den Aufrufpfad $e_0, e_{i_1}, e_0, e_{i_2}, \ldots, e_0, e_{i_k}, e_0$ durchlaufen haben und als nächstes an den Konnektor e_j geleitet werden. Dieses Objekt zerlegen wir in Abhängigkeit von dem Ergebnis der durch $\Phi^j_{i_1\ldots i_k}$ geleisteten Berechnung. Liegt das Ergebnis in $\prod_{i=0}^n U_i \times D$, so ist die Berechnung des Aufrufs abgeschlossen. Den Aufrufen, für die dies gilt, entspricht das Objekt $M_{i_1\ldots i_k j}$. Liegt das Ergebnis jedoch in $\prod_{i=0}^n U_i \times \coprod_{i=1}^n A_i$, so ist ein weiterer Unteraufruf nötig. Diese Aufrufe werden mit $\bar{M}_{i_1\ldots i_k j}$ bezeichnet. Auch den Pullback-Morphismen geben wir eine inhaltliche Interpretation: $\mathcal{A}_{i_1\ldots i_k j}$ ist die auf dem Objekt $M_{i_1\ldots i_k j}$ geleistete Gesamtarbeit, $R_{i_1\ldots i_k j}$ der durch die bisherigen Berechnungsschritte gegebene Anteil der Gesamtarbeit auf $\bar{M}_{i_1\ldots i_k j}$. Die Morphismen $\eta^j_{i_1\ldots i_k}$ und $\bar{\eta}^j_{i_1\ldots i_k}$ sind die Coprodukt-Injektionen.

Die so definierten Morphismen $\mathcal{A}_{i_1\ldots i_k j} : M_{i_1\ldots i_k j} \to \prod_{i=0}^n U_i \times D$ sollen zu einem Morphismus $\mathcal{A}^{\mathcal{U}} : \prod_{i=0}^n U_i \times C \to \prod_{i=0}^n U_i \times D$ zusammengefasst werden. Dazu müssen wir noch überlegen, dass die so definierten Objekte $M_{i_1\ldots i_k}$ paarweise disjunkt sind, d.h. das Pullback zweier Einbettungen in $\prod_{i=0}^n U_i \times C$ das initiale Objekt O ist.

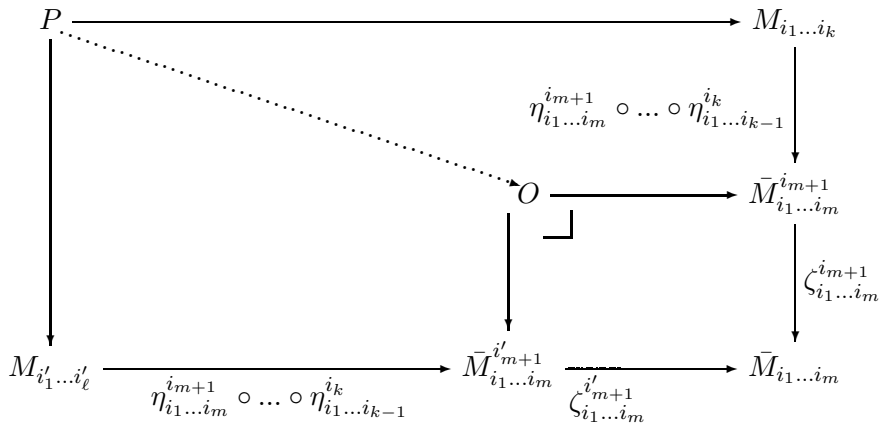
Seien $M_{i_1\ldots i_k}$ und $M_{i'_1\ldots i'_\ell}$ gegeben, wobei natürlich $i_1 \ldots i_k \neq i'_1 \ldots i'_\ell$ vorausgesetzt werde. Wir müssen zwei Fälle unterscheiden: Entweder ist die eine Indexfolge (o.B.d.A. die erste) ein echtes Teilstück der zweiten, oder aber dies gilt nicht.

Wir betrachten zunächst den ersten Fall. In dem Diagramm



ist das initiale Objekt O Pullback von $M_{i_1\dots i_k} \longrightarrow X \longleftarrow \bar{M}_{i_1\dots i_k}$, da Coprodukte in lextensiven Kategorien disjunkt sind. Ist P das Pullbackobjekt von $M_{i_1\dots i_k} \longrightarrow X \longleftarrow M_{i'_1\dots i'_\ell}$, so gibt es wegen der Pullbackeigenschaft einen Morphismus $P \to O$, und da initiale Objekte strikt sind, folgt $P \cong O$.

Nun kommen wir zum zweiten Fall. Da keine der Indexfolgen $i_1 \dots i_k$ und $i'_1 \dots i'_\ell$ echte Anfangsfolge der anderen ist, ist das gemeinsame Anfangsstück ein echtes Anfangsstück beider Folgen (und kann evtl. leer sein). Wir bezeichen mit m den größten Index, für den $i_1 \dots i_m \neq i'_1 \dots i'_m$ ist. Dann sind $M_{i_1\dots i_k}$ und $M_{i'_1\dots i'_\ell}$ beide (mit Zwischenschritten) aus $\bar{M}_{i_1\dots i_m}$ hervorgegangen.



Im obenstehenden Diagramm ist das initiale Objekt O Pullback der Einbettungen von $M_{i_1\ldots i_m}^{i_{m+1}}$ und $M_{i_1\ldots i_m}^{i'_{m+1}}$ in $\bar{M}_{i_1\ldots i_m}$, da Coprodukte disjunkt sind. Die Argumentation ist dieselbe wie zuvor: Ist P das Pullbackobjekt von $M_{i_1\ldots i_k} \longrightarrow X \longleftarrow M_{i'_1\ldots i'_\ell}$, so gibt es wegen der Pullbackeigenschaft einen Morphismus $P \to O$, und da initiale Objekte strikt sind, folgt $P \cong O$.

Da die entstehenden Objekte $M_{i_1\ldots i_k}$ mit ihren Einbettungen disjunkt sind, können wir aus ihnen ein Coprodukt bilden und den vermittelnden Coprodukt-Morphismus der bereits bestimmten Morphismen $\mathcal{A}_{i_1\ldots i_k}$, die die Arbeit auf den Objekten $M_{i_1\ldots i_k}$ darstellen, bilden. Dazu müssen wir jedoch verschärfende Annahmen treffen, für die wir zwei Möglichkeiten haben:

- Unter der verschärfenden Annahme, dass F_0 für jede beliebige Eingabe höchstens n Unteraufrufe erzeugt, haben wir für $\prod_{i=1}^{n} U_i \times C$ folgende Coprodukt-Zerlegung

$$\begin{aligned}\prod_{i=0}^{n} U_i \times C &= M_0 + \bar{M}_0 \\ &= M_0 + M_1 + \ldots + M_n + \bar{M}_1 + \ldots + \bar{M}_n \\ &= \ldots \\ &= \coprod_{k=0}^{n} M_{i_1\ldots i_k},\end{aligned}$$

  und wir können die Gesamtarbeit als den vermittelnden Coprodukt-Homomorphismus darstellen:

$$\mathcal{A}^{\mathcal{U}} := [\mathcal{A}_{i_1\ldots i_k}]_{k=0}^{n} : \coprod_{k=0}^{n} M_{i_1\ldots i_k} \to \prod_{i=0}^{n} U_i \times D.$$

- Wollen wir auf die Annahme der Terminierung nach höchstens n Schritten verzichten, so müssen wir eine zusätzliche Annahme an die Kategorie stellen. Wir verlangen, dass sie nicht nur endliche, sondern auch abzählbare Coprodukte besitzt. Dies ist beispielsweise für **Set** oder **Top** der Fall [HS73, 23.9(3)], gilt aber nicht für beliebige lextensive Kategorien, wie das Gegenbeispiel **FinSet** der Kategorie der endlichen

Mengen zeigt (vgl. [Gol79, 4.4(2)], [HS73, 23.9(1)]). Unter dieser Voraussetzung genügt es, anzunehmen, dass jede Eingabe nach endlich vielen Unteraufrufen terminiert und man hat

$$\prod_{i=0}^{n} U_i \times C = \coprod_{k\in\mathbb{N}} M_{i_1\ldots i_k},$$

$$\mathcal{A}^{\mathcal{U}} := [\mathcal{A}_{i_1\ldots i_k}]_{k\in\mathbb{N}} : \coprod_{k\in\mathbb{N}} M_{i_1\ldots i_k} \to \prod_{i=0}^{n} U_i \times D.$$

Sind $u_i : I \to U_i$ die Startzustände der Komponenten v_i, so ist die Gesamtarbeit $\mathcal{A}$ durch folgende Komposition gegeben:

$$\mathcal{A} : C \xrightarrow{\cong} I \times \ldots I \times C \xrightarrow{u_1\times\ldots u_n\times id_C} \prod_{i=0}^{n} U_i \times C \xrightarrow{\mathcal{A}^U} \prod_{i=0}^{n} U_i \times D \xrightarrow{proj} D.$$

Wir fassen die bisherigen Überlegungen zusammen in folgendem

Satz 5.13 *Sei $\mathfrak{S}$ ein $\mathcal{C}$-Schichtensystem, das zu dem in Abbildung 5.9 gezeigten isomorph ist, und e_0 der eindeutig bestimmte Konnektor zwischen der obersten und der mittleren Schicht. Die Kategorie $\mathcal{C}$ besitze abzählbare Coprodukte. Alle Komponenten von $\mathfrak{S}$ sollen für jeden Aufruf terminieren. Dann ist die Gesamtarbeit von e_0 durch einen Morphismus $\mathcal{A} : C \to D$ darstellbar.* □

Das Resultat lässt sich auf beliebige strenge Schichtensysteme verallgemeinern:

Satz 5.14 *Sei $\mathfrak{S}$ ein strenges $\mathcal{C}$-Schichtensystem in einer lextensiven Kategorie $\mathcal{C}$ mit abzählbaren Coprodukten. Jeder Komponente seien zur Beantwortung eines Aufrufs nur endlich viele Unteraufrufe erlaubt. Dann ist die Gesamtarbeit eines beliebigen Konnektors e durch einen Morphismus $\mathcal{A} : \tau_{in}(e) \to \tau_{out}(e)$ darstellbar.*

Beweis: Wir gehen induktiv vor. Seien e der Konnektor, dessen Arbeit berechnet werden soll und $e_1, \ldots, e_n$ die Elemente von $E^{out}_{tar(e)}$. Angenommen, die Gesamtarbeit von $f_1, \ldots, f_n$ ist durch die Morphismen $\mathcal{A}_{e_1}, \ldots, \mathcal{A}_{e_n}$ gegeben. Dann können wir nach Satz 5.13 die Gesamtarbeit von e durch einen

Morphismus $\mathcal{A}$ darstellen, denn die Situation ist auf die des Beispielsystems aus Abbildung 5.9 isomorph. Die dortigen Morphismen F_i (i=1, ..., n) stellen nämlich genau die Gesamtarbeit der Schnittstellen e_i dar.
Nun müssen wir nur noch die Gesamtarbeit $\mathcal{A}_i$ der n Schnittstellen e_i bestimmen. Dabei gehen wir genauso vor wie im ersten Schritt und betrachten die Elemente von $E^{out}_{tar(e_i)}$. So kann man iterativ durch das Schichtensystem gehen. Da das System streng ist, führt jeder Schritt in eine tiefere Schicht. Die endliche Zahl der Schichten begrenzt die Zahl dieser Schritte, so dass der Prozess nach endlich vielen Schritten terminiert. □

Insgesamt sehen wir also, dass sich bei strengen Schichtensystemen die Gesamtarbeit als Morphismus $\mathcal{A}$ darstellen lässt, wenn mindestens eine der beiden folgenden Annahmen erfüllt ist:

- Es gibt eine natürliche Zahl $n \in \mathbb{N}$, so dass jeder Aufruf maximal n Unteraufrufe produzieren kann.
- Jeder Aufruf produziert höchstens endlich viele Unteraufrufe. Die Kategorie $\mathcal{C}$ besitzt nicht nur endliche, sondern auch abzählbare Coprodukte.

In der Regel wird man die zweite Annahme treffen, da sie eine geringere Einschränkung für das Schichtensystem selbst darstellt und statt dessen eine Forderung an die zugrundeliegende Kategorie ist. In der mathematischen Literatur zur Kategorientheorie wird zwar meist die Frage aufgeworfen, ob eine Kategorie endliche oder beliebige Colimiten besitzt; der – dazwischen angesiedelte – abzählbare Fall ist jedoch speziell in der Informatik von Interesse, beispielsweise im Zusammenhang mit der Semantik von Programmen [MA86, Sect. 3.2], in der Typtheorie [MP98] und zur Modellierung des *Call-by-Push-Value*-Paradigmas [Lev02]. Wie diese Arbeiten zeigen, kommt die Forderung nach abzählbaren Coprodukten häufig dann ins Spiel, wenn iterative Aspekte kategoriell formuliert werden sollen, ganz analog zu unserem Vorgehen bei der Bestimmung der Gesamtarbeit als Morphismus.

5.7 Anwendungsbeispiel in der Kategorie ATGrph

Wir wollen nun ein Anwendungsbeispiel in der Kategorie der attributierten getypten Graphen vorstellen. Die Kategorie der attributierten getypten Graphen **ATGrph** (vgl. Anhang A.1) ist adhäsiv [EPT04] und besitzt strikte initiale Objekte, denn der leere Graph O ist initial und ein Graphhomomorphismus $G \to O$ kann nur existieren, wenn $G \cong O$ ist. Mithin ist **ATGrph** eine extensive Kategorie [LS04, Lemma 11], ja sie ist sogar lextensiv, da alle endlichen Limiten existieren.

Dies sieht man wie folgt: Gemäß [EHL$^+$99] existieren Pullbacks und binäre Produkte, denn man kann nach [EPT04, Theorem 1] die Kategorie **ATGrph** als eine Kategorie von Algebren zu einer Graph-Signatur auffassen. Da durch den Typgraph, der mit der finalen Σ-Algebra versehen ist, ein terminales Objekt gegeben ist, existieren sogar beliebige endliche Produkte. Nach [HS73, 23.7] existieren damit alle endlichen Limiten. Es ist offensichtlich, dass die leeren Objekte genau die initialen sind.

Attributierte getypte Graphen lassen sich übersichtlich in der Notation von UML-Klassen- und Objektdiagrammen darstellen. Wir verwenden diese Notation auch im folgenden für den Typgraphen unseres Beispiels.

Wir modellieren einen Ausschnitt aus einem einfachen System, das aus drei Schichten besteht: Benutzerschnittstelle, Anwendungsschicht und Datenbank. Der Anwendungsfall, den wir modellieren, ist der Zugriff auf eine Projektdatendank. Eine Komponente der Benutzerschnittstelle sendet die Eingaben des Benutzers an die mittlere Schicht, die bis zur endgültigen Antwort die Unteraufrufe koordiniert. Zunächst sendet die angesprochene Komponente der mittleren Schicht eine Anfrage an den Autorisierungsserver, der die Zugriffsrechte auf Operationen verwaltet. Stimmt dieser einem Zugriff zu, stellt die mittlere Schicht die Anfrage über die Projekte an die Projektdatenbank, die sie beantwortet. Schließlich leitet die Komponente der mittleren Schicht diese Antwort weiter an die Benutzerschnittstelle zur Anzeige. Als Typisierung verwenden wir einen Ausschnitt aus dem Problembereichsmodell der Anwendung. Der Typgraph umfaßt also u.a. *Benutzer*, *Projekt* und *Mitarbeiter* als Knotentypen, und Kanten zwischen ihnen, falls zwischen den jeweiligen Typen eine Assoziation besteht (Abb. 5.11).

Nun wollen wir dem bereits bei der Berechnung der Aufrufsequenzen angesprochenen Beispiel 5.7 eine inhaltliche Interpretation geben. Die Kom-

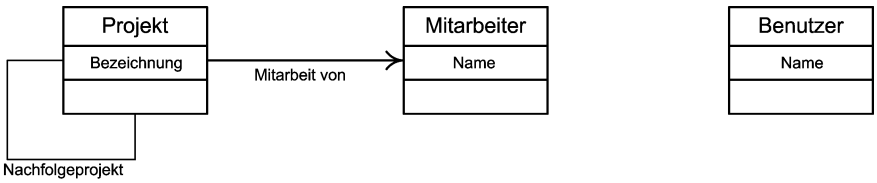


Abbildung 5.11: Aussschnitt aus dem Typgraphen

ponente v der Benutzerschnittstellenschicht leitet die Daten des Aufrufs, die als attributierter Graph kodiert sind, an die Komponente der mittleren Schicht weiter (Schritt 1 in Abb. 5.7). Dabei ist die Identität des Benutzers als Knoten repräsentiert, der seine Kennung als Markierung enthält, und das fragliche Projekt als weiterer Knoten mit der Nummer des Projekts. Schritt 2 zeigt, dass die Komponente der mittleren Schicht einen Unteraufruf an die Datenbankschicht übermittelt, deren Antwort in Schritt 3 zurückgegeben wird. Da die Autorisierung erfolgreich war, wird in Schritt 4 die Anfrage nach den Projektdaten an die Projektdatenbank gestellt. Diese liefert als Rückgabe einen Graph, der das Projekt, seine Mitarbeiter und die Vorgänger- und Nachfolgerprojekte in einer graphischen Kodierung darstellt (Schritt 5). Dieser Graph wird schließlich an die Benutzerschnittstelle weitergereicht (Schritt 6).

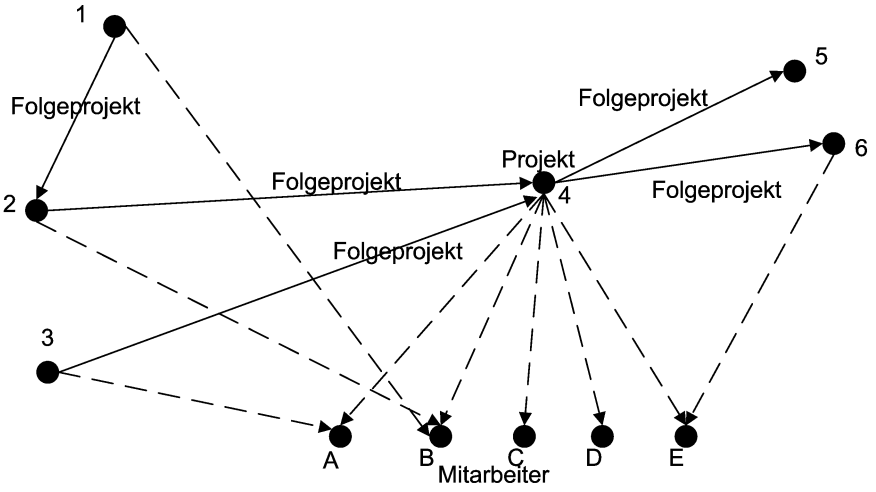


Abbildung 5.12: Mögliches Ergebnis des Aufrufs in **ATGrph**

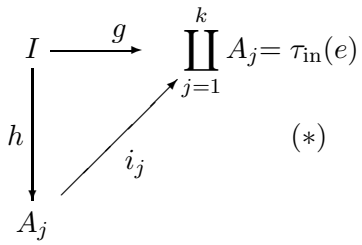
Eine mögliche Kodierung des Ergebnisses ist in Abb. 5.12 zu sehen. Allgemein kann jede Relation $R \subseteq M \times N$ als Graph dargestellt werden, indem die Elemente von M und N als Knoten darstellt und zwischen zweien eine Kante einfügt wird, falls diese vermöge R in Relation stehen. Dies ermöglicht eine direkte Kodierung der Daten relationaler Datenbanken als Graphen, die der Inzidenzmatrix entspricht. Man hat somit einen Inklusions-Funktor $\mathfrak{Rel} \hookrightarrow$ **Grph**, der R auf den Graphen $(M \dot{\cup} N, R, proj_1, proj_2)$ abbildet. $\mathfrak{Rel}$ ist sogar eine reflektive Subkategorie [Mac97, IV.3.] von **Grph** (siehe Anhang A.2). Dies ermöglichst es, trotz der Tatsache, dass $\mathfrak{Rel}$ nicht extensiv ist, auch relationale Datentypen zu modellieren.

5.8 Beispiel: Fallunterscheidung

Innere Komponenten können dazu eingesetzt werden, aufgrund logischer Kriterien zwischen der Benutzung mehrerer Komponenten tieferer Schichten zu entscheiden. Eine (abstrakte) Fallunterscheidungskomponente ist eine zustandslose innere Komponente v, die folgender Spezifikation genügt:

- Es ist $E_v^{in} = \{e\}$ und $E_v^{out} = \{f_1, ..., f_k\}$ mit $k > 0$.
- Es existieren Objekte $A_1, ..., A_k$ mit $\tau_{\text{in}}(e) = \coprod_{j=1}^{k} A_j$ und Monomorphismen $m_j : A_j \to \tau_{in}(f_j)$.
- Es existieren Monomorphismen $n_j : \tau_{out}(f_j) \to \tau_{out}(e)$.

Dann gilt nach Satz 4.29: Für jedes globale Element $g : I \to \tau_{\text{in}}(e)$ existiert genau ein $j \in \{1, ..., k\}$ und ein globales Element $h : I \to A_j$, so dass



kommutiert. Der Morphismus

$$F_e : \tau_{\text{in}}(e) + \coprod_{j=1}^{k} \tau_{\text{out}}(f_j) \longrightarrow \tau_{\text{out}}(e) + \coprod_{j=1}^{k} \tau_{\text{in}}(f_j)$$

erfülle folgende Eigenschaften:

- F_e muss die Coprodukt-Zerlegung $\coprod_{j=1}^{k} A_j$ mittels der Monomorphismen m_j in die Coprodukt-Zerlegung $\coprod_{k=1}^{m} \tau_{\text{in}}(f_k)$ überführen. Es wird also von F_e gefordert: Ist $g : I \to \tau_{\text{in}}(e)$ ein globales Element, so sei $F_e \circ \tilde{g}$ genau dann Fixpunkt von M_j^{in}, wenn Diagramm $(*)$ kommutiert. In diesem Fall sei $p_j^{in} \circ F_e \circ \tilde{g} = m_j \circ h$.

- Rückgabewerte werden vermöge der Monomorphismen n_j in das Datenobjekt $\tau_{out}(e)$ eingebettet, d.h.: Ist $a : I \to \tau_{\text{out}}(f_j)$ gegeben, so sei $F_e \circ \tilde{a}$ Fixpunkt von M^{out} und $p^{out} \circ F_e \circ \tilde{a} = n_j \circ a$.

Der Konnektor e leitet also alle Daten einer Eingabe unbearbeitet weiter und entscheidet nur darüber, an welche Komponente sie weitergegeben werden. Falls $k = 1$ ist, hat man keine echte Fallunterscheidung, da es nur einen Fall gibt. Die Komponente ist dann eine reine Verbindungskomponente, die z.B. benötigt wird, um eine Schicht zu überbrücken.
Ein Beispiel für eine Fallunterscheidungskomponente haben wir bereits in Abschnitt 5.3 (1. Fall) gesehen. Dort nimmt Komponente v_3 über e_3 Aufrufe von Komponente v_4 entgegen und leitet sie entweder an v_1 oder v_2 weiter. Durch rein kategorielle formulierbare Forderungen an den Morphismus F_e lassen sich also konkrete Eigenschaften auf der Ebene des dynamischen Modells abstrakt beschreiben. Diese können verwendet werden, um das Verhalten eines Systems genauer zu verstehen.

5.9 Transformation in Schichtenarchitekturen

Transformationen von Architekturen sind in den letzten Jahren mit verschiedenen formalen Methoden untersucht worden. Die meisten Arbeiten betrachten Rekonfigurationen innerhalb eines Architekturstils, also Änderungen in der Konfiguration eines Systems, die konsistent mit dem bereits verwendeten Architekturstil sind [HM01]. In [WF02] werden als fundamentale Operationen das Hinzufügen und Entfernen von Komponenten oder Konnektoren identifiziert und mittels Graph-Grammatiken formalisiert. Verfeinerungsoperationen im Bereich der dynamischen Architekturen, wie sie insbesondere bei mobilen System vorkommen, modelliert [HT04]. Auch dort wird ist der Formalismus der Graph-Transformation eingesetzt, und es wird speziell untersucht, welche Transformationen das Verhalten des Systems unverändert

lassen. Eine Arbeit, die Modifikationen an der Architektur bereits ausgelieferter Systeme untersucht und auf die Wichtigkeit der Werkzeugunterstützung in diesem Bereich hinweist, ist [KLV05]. Als Voraussetzung für den Einsatz von Werkzeugen ist eine formale Analyse hilfreich.
Wir wollen hier allein den Fall betrachten, dass eine vorhandene komponentenbasierte Architektur in eine Schichtenarchitektur transformiert werden soll. Die notwendigen Bedingungen, um eine Schichteneinteilungsfunktion auf einem gegebenen Konfigurationsgraphen definieren zu können, haben wir in Abschnitt 2.4 bestimmt.
In der Softwareentwicklung ist die Zuordnung von Komponenten zu Schichten in der Regel aus fachlichen Erwägungen gegeben. Daher interessiert nicht primär, ob eine beliebige Einteilung in n Schichten möglich ist oder welche Veränderungen dazu vorgenommen werden müssen. Statt dessen gehen wir davon aus, dass aus fachlicher Sicht schon eine Einteilung der Komponenten vorgenommen wurde (etwa in Benutzerschnittstelle, Datenbank, etc.) und diese Einteilung sich nicht als Schichtenarchitektur auffassen lässt. Der Interpretation als Schichtenarchitektur können zwei Hindernisse entgegen stehen:

- Es gibt Komponenten, die Fachlichkeit aus mehreren Schichten enthalten. Diese müssen in mehrere Komponenten aufgeteilt werden.
- Es existieren Konnektoren zwischen Komponenten in nicht unmittelbar benachbarten Schichten (*layer bridging* [BCKW98]). Beispiel: GUI greift direkt auf Datenbank zu. Hier müssen vermittelnde Komponenten eingeführt werden, die zunächst nur weiterleitende Funktion haben, aber durchaus später mit schichtspezifischer Funktionalität betraut werden können.

Schwerwiegender ist der erste Fall. Die fachliche Analyse lässt sich jedoch nicht formalisieren, so dass wir das erste Problem rein formal auf das zweite reduzieren: Die fragliche Komponente wird willkürlich der obersten Schicht, zu der sie Konnektoren besitzt, zugeordnet (und hat dann in der Regel illegale Konnektoren).
Wir gehen also von einem implementierten System aus, dessen Schichteneinteilung $p : G_V \rightarrow \{1, ..., n\}$ illegale Kanten besitzt, also Konnektoren zwischen Komponenten nicht benachbarter Schichten. Das System soll so angepasst werden, dass ein gültiges Schichtensystem ensteht.

Auflösung illegaler Kanten

Sei $G = (G_V, G_E)$ ein Graph und $p : V \to \{1, ..., n\}$ eine surjektive Abbildung, so dass eine illegale Kante $e \in G_E$ existiert. Wir betrachten den Graphen $G' = (G'_V, G'_E)$, der wie folgt aus G hervorgeht:
Für jedes i mit $p(tar(e)) < i < p(src(e))$ definieren wir eine neue Komponente v^i mit $p(v^i) = i$. Ferner führen wir für $p(tar(e)) \leq i < p(src(e))$ neue Kanten e^i ein, für die wir folgende Funktionen src' und tar' definieren:

$$src'(e^i) = \begin{cases} v^{i+1}, & \text{falls} \quad p(tar(e)) \leq i < p(src(e)) - 1, \\ src(e), & \text{falls} \quad i = p(src(e)) - 1, \end{cases}$$

$$tar'(e^i) = \begin{cases} v^i, & \text{falls} \quad p(tar(e)) < i < p(src(e)), \\ tar(e), & \text{falls} \quad i = p(tar(e)). \end{cases}$$

Der Graph G' ist also gegeben durch

$$G'_V = G_V \cup \{v^i \mid p(tar(e)) < i < p(src(e))\}$$

und

$$G'_E = G_E \backslash \{e\} \cup \{e^i \mid p(tar(e)) \leq i < p(src(e))\}.$$

Abbildung 5.9 zeigt ein Beispiel. Wir erweitern τ durch die Setzung $\tau_{\text{in}}(e^i) = \tau_{\text{in}}(e)$ und $\tau_{\text{out}}(e^i) = \tau_{\text{out}}(e)$ für $p(tar(e)) \leq i < p(src(e))$ auf G'_V. Die Funktionalität der neuen Konnektoren e^i ist trivial: Wir setzen $U_{v^i} = I$ für $p(tar(e)) < i < p(src(e))$; die neuen Komponenten sind also zustandslos. Ferner sei

$$F_{e^i} : \tau_{\text{in}}(e) + \tau_{\text{out}}(e) \to \tau_{\text{out}}(e) + \tau_{\text{in}}(e)$$

für $p(tar(e)) < i < p(src(e))$ der eindeutig bestimmte Morphismus, der die Cofaktoren miteinander vertauscht. Der Konnektor e^i leitet also die Daten nur weiter. Der Morphismus

$$F_{e^{tar(e)}} : \left(\tau_{\text{in}}(e) + \coprod_{k=1}^{m} \tau_{\text{out}}(f_k)\right) \times U_{tar(e)} \longrightarrow \left(\tau_{\text{out}}(e) + \coprod_{k=1}^{m} \tau_{\text{in}}(f_k)\right) \times U_{tar(e)}$$

ist mit dem ursprünglichen Morphismus F_e identisch. Er stellt sicher, dass die Funktionalität des ursprünglichen Konnektors e weiterhin im System vorhanden ist.

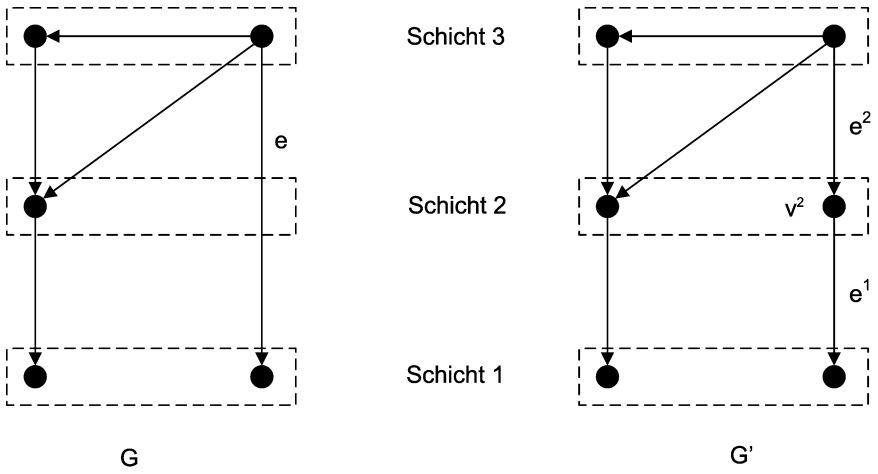
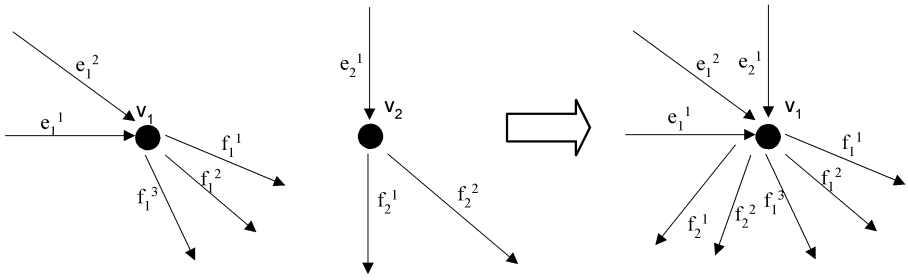


Abbildung 5.13: Eliminierung der problematischen Kante e

Damit ist die problematische Kante e eliminiert und die Zahl der problematischen Kanten um 1 vermindert. Besitzt ein Graph mehrere problematische Kanten, so wendet man das beschriebene Verfahren wiederholt an und erhält schließlich einen Graphen mit einer gültigen Schichteneinteilung.

Fusion von Komponenten

In einem zweiten Schritt können Komponenten einer Schicht, die nicht durch Konnektoren miteinander verbunden sind, fusioniert werden. Die Anzahl der Konnektoren ändert sich dadurch nicht. Eventuell entstehen bei dieser Prozedur Mehrfachkanten zwischen zwei Komponenten. Diese entsprechen verschiedenen Konnektoren derselben Komponente.



Seien (G, p, τ) ein $\mathcal{C}$-Schichtensystem und $v_1, v_2 \in G_V$ zwei Komponenten mit $p(v_1) = p(v_2)$ und $\neg\exists e \in E : \{src(e), tar(e)\} = \{v_1, v_2\}$. Dann definieren wir ein neues System (G', p', τ') wie folgt: $G'_V = G_V \backslash \{v_1, v_2\} \cup \{v\}$, $G'_E = G_E$. Wir definieren die Abbildungen src' und tar': Für alle $e \in G_E$ mit $src(e) \in \{v_1, v_2\}$ setzen wir $src'(e) = v$, für alle mit $tar(e) \in \{v_1, v_2\}$ sei $tar'(e) = v$ und für alle anderen $e \in G_E$ bleibt $src'(e) = src(e)$ und $tar'(e) = tar(e)$. Die Morphismen $F'_{e_i^j}$ der fusionierten Komponente haben als Zustandsobjekt das Produkt $U_{v_1} \times U_{v_2}$:

$$F'_{e_i^j}: \left(\tau_{\text{in}}(e_i^j) + \coprod_{f \in E_{v_i}^{out}} \tau_{\text{out}}(f)\right) \times (U_{v_1} \times U_{v_2}) \longrightarrow \left(\tau_{\text{out}}(e_i^j) + \coprod_{f \in E_{v_i}^{out}} \tau_{\text{in}}(f)\right) \times (U_{v_1} \times U_{v_2})$$

Diese Morphismen sind wie folgt definiert:

$$F'_{e_1^j} := F_{e_1^j} \times id_{U_{v_2}},$$
$$F'_{e_2^j} := twist \circ (F_{e_2^j} \times id_{U_{v_1}}) \circ twist,$$

wobei *twist* die Vertauschung der beiden Faktoren U_{v_1} und U_{v_2} bezeichne. Der Morphismus $F'_{e_i^j}$ erhält also den nicht benötigten Faktor des Zustandsobjekts $U_{v_1} \times U_{v_2}$ unverändert und ist ansonsten wie $F_{e_i^j}$ definiert. Das Verhalten des Schichtensystems ist unverändert, da alle Konnektoren nach wie vor mit derselben Semantik versehen sind.
Hingegen kann die Fusion zweier Komponenten, die durch einen Konnektor verbunden sind, i.a. das Terminierungsverhalten des Systems verändern. Wie Beispiel 5.11 zeigt, ist bereits in einem Teilsystem mit zwei Komponenten eine unendliche Folge von Aufrufen möglich. Es ist klar, dass sich das dynamische Zusammenspiel der Komponenten v_1 und v_2 aus Beispiel 5.11

nicht durch die Operation einer einzelnen Komponente ersetzen lässt, da jede Einzelkomponente notwendig terminiert.

Kapitel 6

Bisimulationen und Verhaltensäquivalenz

Der Begriff der Bisimulation hat sich in vielen formalen Bereichen der Informatik als bedeutsam herausgestellt, wenn die Verhaltensähnlichkeit von Systemen untersucht werden soll. Unser dynamisches Modell mit der Aufrufsequenz als zentralem Begriff wirft die Frage auf, wie der statische Konfigurationsgraph abgewandelt werden kann, ohne dass die Menge der beobachtbaren Aufrufsequenzen sich ändert, wenn alle denkbaren Implementierungen durchlaufen werden. Da Aufrufsequenzen typkonforme Graphhomomorphismen von Bäumen in den Konfigurationsgraph sind, führt diese Frage zu der Untersuchung von Graphen, die ähnliche Homomorphismen zulassen. Dazu verwenden wir Bisimulationen.

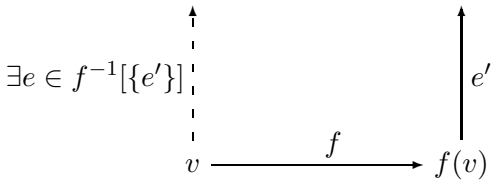
6.1 Gerichtete Graphen

Da Schichtensysteme interpretierte Graphen sind, wird ein Bisimulations-Begriff für Schichtensysteme sich konzeptionell eng an ein adäquates Pendant für gerichtete Graphen anlehnen. Daher führen wir zunächst einen Bisimulationsbegriff für gerichtete Graphen ein und erweitern diesen im folgenden auf Schichtensysteme. Als Hilfsbegriff definieren wir Simulationen. Eine Spanne von Simulationen ergibt eine Bisimulation [JNW96, Rut00]. Der Zwischenbegriff der Simulation wird nötig, da bei einem Graph-Homomorphismus eine Kante, deren Ursprungsknoten im Bild liegt, selbst nicht im Bild liegen

muss und der Begriff des Graph-Homomorphismus daher einer Verschärfung bedarf.
Seien $G = (G_V, G_E, src, tar)$ und $G' = (G'_V, G'_E, src', tar')$ zwei Graphen. Ein Graph-Homomorphismus $f = (f_V, f_E) : G \to G'$ heißt *Simulation*, falls für alle $e' \in G'_E$ gilt:

$$\forall v \in G_V \left[f_V(v) = src'(e') \Longrightarrow \exists e \in G_E \; f_E(e) = e' \wedge src(e) = v \right].$$

Folgendes Diagramm veranschaulicht diese Bedingung:



Während jeder Graph-Homomorphismus f Kanten *erhält*, also eine Kante $e \in G_E$ mit $src(e) = v$, $tar(e) = w$ auf eine Kante $e' = f_E(e)$ mit $src'(e') = f_V(v)$ und $tar'(e') = f_V(w)$ abbildet, muss eine Simulation Kanten auch *reflektieren*, d.h. zu jeder Kante $e' \in G'_E$ mit $src(e') = f_V(v)$ für ein $v \in G_V$ und $tar'(e') = w'$ gibt es eine Kante e mit $e' = f_E(e)$. Diese Forderung ist analog zum Homomorphie-Begriff in coalgebraischen Transitionssystemen [Rut00, 2.1]: Rutten betrachtet Coalgebren zu dem Funktor $B(X) = \mathfrak{P}(A \times X)$, wobei A eine Menge von Marken ist. Eine Coalgebra $src : X \to B(X)$ interpretiert er durch die Setzung

$$s \xrightarrow{a} s' \Leftrightarrow \langle a, s' \rangle \in src(s)$$

als Transitionssystem und zeigt, dass die Coalgebra-Homomorphismen zwischen bei Coalgebren (X, src) und (Y, β) gerade die Abbildungen $X \to Y$ sind, die Transitionen erhalten und reflektieren. Dies entspricht genau unserer Forderung, wenn man A einelementig wählt.
Die Herangehensweise, den Morphismenbegriff in einer Kategorie durch zusätzliche Forderungen zu verschärfen, findet sich bei diversen speziellen Morphismenbegriffen wie *zig-zag*-Morphismen, *p*-Morphismen, etc. (einen Überblick gibt [JNW96]). Wir kommen nun zum zentralen Begriff dieses Kapitels.

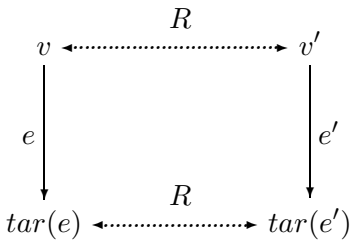
Definition 6.1 *Sei $s : G \xleftarrow{f} B \xrightarrow{g} G'$ eine Spanne von Simulationen mit $B \neq \emptyset$. Dann heißt die Relation $R \subseteq G_V \times G'_V$ mit*

$$(v, v') \in R :\Longleftrightarrow \exists w \in B_V \; f(w) = v \wedge g(w) = v'$$

die von s erzeugte Bisimulation. *Eine beliebige Relation* $R \subseteq G_V \times G'_V$ *heißt* Bisimulation*, falls es eine Spanne* s *von Simulationen gibt, so dass* R *die von* s *erzeugte Bisimulation ist.*

Wir fordern, dass eine Bisimulation nicht-leer ist, denn die leere Bisimulation ist trivial und würde zwei beliebige Graphen bisimilar machen. Es gilt folgender charakterisierender

Satz 6.2 *Eine Relation* $R \subseteq G_V \times G'_V$ *ist genau dann eine Bisimulation, wenn für alle* $(v, v') \in R$ *gilt:*
1) Ist $e \in G_E$ *mit* $src(e) = v$*, so existiert* $e' \in G'_E$ *mit* $src'(e') = v'$ *und es gilt* $\langle tar(e), tar(e')\rangle \in R$*.*
2) Ist $e' \in G'_E$ *mit* $src'(e') = v'$*, so existiert* $e \in G_E$ *mit* $src(e) = v$ *und es gilt* $\langle tar(e), tar(e')\rangle \in R$*.*



Beweis: Sei $G \xleftarrow{f} B \xrightarrow{g} G'$ eine Spanne von Simulationen und R die erzeugte Relation. Seien ferner $v \in G_V, v' \in G'_V$ mit $(v, v') \in R$ gegeben, d.h. es existiert ein $w \in B_V$ mit $f_V(w) = v$ und $g_V(w) = v'$.
Wir zeigen zunächst Eigenschaft 1). Ist $e \in G_E$ mit $src(e) = v$, so existiert $b \in B_E$ mit $f_E(b) = e$ und $src(b) = w$, da f Simulation ist. Für $e' := g_E(b)$ gilt

$$src'(e') = src'(g_E(b)) = g_V(src(b)) = g_V(w) = v'.$$

Von Eigenschaft 2) überzeugt man sich wie folgt: Ist $e' \in G'_E$ mit $src'(e') = v'$, so existiert $b \in B_E$ mit $g_E(b) = e'$ und $src(b) = w$, da g Simulation ist. Für $e := f_E(b)$ gilt

$$src(e) = src(f_E(b)) = f_V(src(b)) = f_V(w) = v.$$

Wir zeigen nun die Umkehrung. Sei $R \subseteq G_V \times G'_V$ eine Relation, die 1) und 2) erfüllt. Wir müssen einen Graphen $B = (B_V, B_E, src_B, tar_B)$ konstruieren.

Wir setzen

$$B_V = R,$$
$$B_E = \Big\{(e, e') \in G_e \times G'_E \mid \langle src(e), src(e')\rangle \in R, \langle tar(e), tar(e')\rangle \in R\Big\},$$
$$src_B(e, e') = \langle src(e), src(e')\rangle,$$
$$tar_B(e, e') = \langle tar(e), tar(e')\rangle.$$

Außerdem müssen wir Homomorphismen $f : B \to G$ und $g : B \to G'$ erklären. Wir setzen $f := proj_1$ und $g := proj_2$. Es ist klar, dass die von $G \xleftarrow{f} B \xrightarrow{g} G'$ erzeugte Relation wieder R ist. Es bleibt zu zeigen, dass f und g Simulationen sind. O.B.d.A. zeigen wir dies für f. Sei $e \in G_E$ beliebig und $w \in B_V$, so dass $f_V(w) = src(e)$ ist. Dann existiert $e' \in G'_E$ mit $src(e') = g_V(w)$ und $\langle tar(e), tar(e')\rangle \in R$. Mit $b := \langle e, e'\rangle$ ist ein Element von B_E gefunden, für das $f(b) = e$ ist und für das gilt

$$src(b) = \langle src(e), src(e')\rangle = \langle f_V(w), g_V(w)\rangle = \langle proj_1(w), proj_2(w)\rangle = w.$$

Damit ist f Simulation. □

Wir bezeichnen zwei Graphen G, G' bisimilar, wenn eine Bisimulation $R \subseteq G_V \times G'_V$ existiert. Die dadurch erklärte Relation zwischen Graphen nennen wir *Bisimilarität*. Man beachte, dass eine Bisimulation die Knotenmengen zweier Graphen in Beziehung setzt, während Bisimilarität auf der Abstraktionsebene der Graphen selbst erklärt ist. Da wir die leere Relation nicht als Bisimulation anerkennen, erhalten wir zunächst ein negatives Resultat:

Bemerkung 6.3 *Bisimilarität ist keine Äquivalenzrelation, denn sie ist nicht transitiv.*

Abbildung 6.1 zeigt zwei Spannen von Simulationen. Die Beschriftung definiert die Homomorphismen f, g, h, k auf den Knotenmengen; auf den Kantenmengen sind sie damit wegen der Homomorphieeigenschaft eindeutig festgelegt. Es existiert keine Bisimulation zwischen G und G'', da der einzige Kandidat $R = G_V \times G''_V$ die Bedingungen aus Satz 6.2 nicht erfüllt. Durch eine zusätzliche Forderung läßt sich Transitivität erreichen:

Definition 6.4 *Eine Bisimulation heißt bitotal, falls sie von einer Spanne surjektiver Simulationen erzeugt wird.*

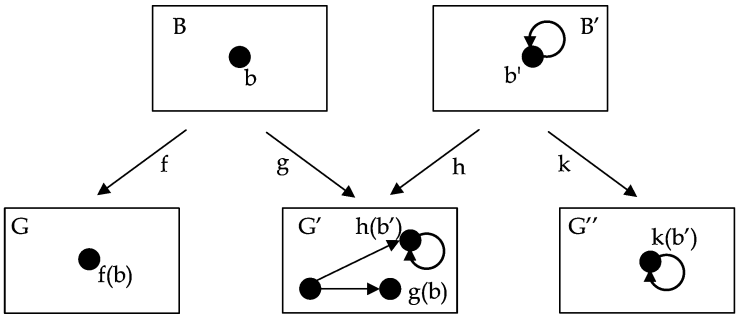


Abbildung 6.1: Bisimilarität ist nicht transitiv

Die zusätzliche Forderung, dass die Simulationen surjektiv sein müssen, impliziert, dass die erzeugte Bisimulation eine bitotale, also links- und rechtstotale Relation ist. Die analoge Relation auf der Menge aller Graphen nennen wir *bitotale Bisimilarität*[1].

Satz 6.5 *Bitotale Bisimilarität ist eine Äquivalenzrelation.*

Beweis: Nur die Transitivität ist nicht trivial. Sind zwei Spannen

$$G \xleftarrow{f} B \xrightarrow{g} G'$$

und

$$G' \xleftarrow{h} B' \xrightarrow{k} G''$$

gegeben, so bildet man das Pullback von $B \xrightarrow{g} G' \xleftarrow{h} B'$. [EHL+99, Appendix] entnimmt man, dass der von

$$P_V := \{(b, b') \in B_V \times B'_V \mid g(b) = h(b')\}$$

erzeugte Subgraph P von $B \times B'$ zusammen mit den Morphismen $g' = proj_B \circ i_P$ und $h' = proj_{B'} \circ i_P$ das Pullback von $B \xrightarrow{g} G' \xleftarrow{h} B'$ ist. Da die Simulationen g und h surjektiv sind, ist $P_V \neq \emptyset$ und damit P nicht der leere Graph. □

Eine weitere Möglichkeit, den Bisimulationsbegriff so abzuwandeln, dass Bisimilarität transitiv wird, besteht darin, mit punktierten Graphen zu arbeiten, also Paaren (G, v_0) mit $v_0 \in G_V$, und von Morphismen zu fordern, dass

[1]Die Bezeichnung sollte nicht darüber hinwegtäuschen, dass nicht die Bisimilarität zwischen Graphen bitotal ist, sondern die zugrundeliegenden Bisimulations-Relationen.

sie die ausgezeichneten Knoten erhalten. In einer solchen Kategorie $\mathbf{Grph}^*$ kann das Pullback zweier Morphismen nie ein leerer Graph sein.

6.2 Erweiterung auf Schichtensysteme

Wir erweitern den Simulationsbegriff nun von Graphen auf Schichtensysteme. Seien dazu $\mathfrak{S} = (G, p, \tau)$ und $\mathfrak{S}' = (G', p', \tau')$ $\mathcal{C}$-Schichtensysteme. Eine *Simulation zwischen* $\mathfrak{S}$ *und* $\mathfrak{S}'$ ist ein $\mathcal{C}$**Syst**-Morphismus

$$f = (f_V, f_E) : G \to G',$$

so dass für alle $e' \in G'_E$ gilt:

$$\forall v \in G_V \Big[f_V(v) = src'(e') \Longrightarrow \exists e \in G_E \ (\tau(e) = \tau(e') \wedge f_E(e) = e' \wedge src(e) = v) \Big].$$

Wir fordern also zusätzlich, dass das gesuchte Element $e \in G_E$ dasselbe Bild wie e' unter $\tau = (\tau_{in}, \tau_{out}) : G_E \to |\mathcal{C}| \times |\mathcal{C}|$ hat. Eine Bisimulation von Schichtensystemen ist dann die von einer Spanne $s : \mathfrak{S} \xleftarrow{f} \mathfrak{B} \xrightarrow{g} \mathfrak{S}'$ von Simulationen erzeugte Relation $R \subseteq G_V \times G'_V$ auf den Komponentenmengen. Satz 6.2 gilt analog für Bisimulationen von Schichtensystemen.
Das beobachtbare Verhalten eines Schichtensystems $\mathfrak{S}$ ist durch Aufrufsequenzen in $\mathfrak{S}$ gegeben. Daher charakterisieren wir *Verhaltensäquivalenz* zweier Systeme $\mathfrak{S}$, $\mathfrak{S}'$ dadurch, dass sich jede Aufrufsequenz in $\mathfrak{S}$ in eine Aufrufsequenz in $\mathfrak{S}'$ transformieren lässt, in dessen Bild dieselben Objekte liegen. Die zuletzt genannte Forderung führt auf folgenden Begriff:

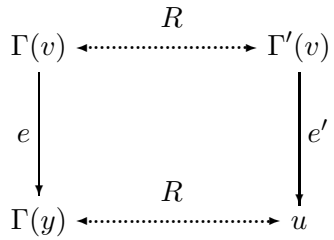
Definition 6.6 *Sei (T, w) ein Aufrufbaum. Wir nennen zwei Aufrufsequenzen $\Gamma : T \to G$ und $\Gamma' : T \to G'$ typkonform, falls für alle $e \in T_E$ gilt:*

$$\tau(\Gamma_E(e)) = \tau'(\Gamma'_E(e)).$$

Bisimilare Schichtensysteme sind in ihrem beobachtbaren Verhalten äquivalent, da sie typkonforme Aufrufsequenzen von denselben Aufrufbäumen zulassen.

Satz 6.7 *Seien $\mathfrak{S} = (G, p, \tau)$, $\mathfrak{S}' = (G', p', \tau')$ zwei $\mathcal{C}$-Schichtensysteme und (T, w) ein Aufrufbaum. Ist R eine Bisimulation zwischen $\mathfrak{S}$ und $\mathfrak{S}'$ und $\Gamma : T \to G$ eine Aufrufsequenz mit $\{v \mid (v, \Gamma(w)) \in R\} \neq \emptyset$, so existiert eine typkonforme Aufrufsequenz $\Gamma' : T \to G'$.*

Beweis: Sei $r \in G'_V$ mit $(r, \Gamma(w)) \in R$. Dann setzen wir $\Gamma'(w) = r$. Ist $v \in T$ ein Knoten, für den Γ' bereits definiert ist, so ist die Menge K_v der Kinder von v wohlgeordnet durch die Relation $<_v$. Wir durchlaufen diese mit transfiniter Induktion [Sri98, 1.7].
Sei Γ' für alle $x \in K_v$ mit $x <_v y$ definiert. Wir bezeichnen die Kante zwischen $\Gamma(v)$ und $\Gamma(y)$ mit e. Dann existiert ein Knoten $u \in G'_V$ und eine Kante $e' \in G'_E$ mit $src'(e') = \Gamma'(v)$ und $tar'(e') = u$,



so dass $\tau(e) = \tau(e')$ ist. Setze $\Gamma'(y) := u$. Ebenso durchlaufen wir den Baum T induktiv in die Tiefe. Dann sind Γ und Γ' typkonforme Aufrufbäume. □

Nur die oberste Schicht L_n eines $\mathcal{C}$-Schichtensystems erlaubt Zugriffe von außen, d.h. es muss $\Gamma(w) \in L_n$ gelten. Daher genügt es, wenn die Bisimulation auf dieser bitotal ist:

Corollar 6.8 *Seien G, G' bisimilare $\mathcal{C}$-Schichtensysteme, (T, w) ein Aufrufbaum und $\Gamma : T \to G$ ein Aufrufbaum. Es gelte für jedes $v \in L_n$*

$$\{v' \mid (v, v') \in R\} \neq \emptyset.$$

Dann existiert ein zu Γ typkonformer Aufrufbaum $\Gamma' : T \to G'$.□

Bisimulation ist also ein Begriff, der Verhaltensäquivalenz impliziert. Die Forderung an $\mathcal{C}$**Syst**-Bisimulationen, die Schichteneinteilung zu respektieren, ist im Beweis von Satz 6.7 nicht eingeflossen. Verzichten wir darauf, so ergibt sich ein allgemeinerer Bisimulations-Begriff, den wir *A-Bisimulation* nennen. Da wir Konnektoren sowohl zwischen Komponenten in benachbarten Schichten als auch innerhalb derselben Schicht zulassen, können verhaltensäquivalente Systeme verschiedene Schichteneinteilungen besitzen:

Beispiel 6.9 *Verhaltensäquivalenz hängt nicht von der Schichteneinteilung ab* Es gibt nämlich A-bisimilare Systeme mit verschieden vielen Schichten.

Dies zeigt bereits das einfache Beispiel zweier Schichtensysteme mit je genau zwei Komponenten und genau einem Konnektor dazwischen, wobei einmal beide Komponenten in einer Schicht liegen und das andere mal in zwei benachbarten Schichten.

Die Schichteneinteilung kann also ohne Änderung der Funktionalität gewissen Änderungen unterworfen werden und ist somit ein Konzept, das zwar ein strukturiertes Vorgehen beim Design einer Architektur unterstützt, aber weder allzu enge Restriktionen auferlegt, noch Folgerungen bezüglich des beobachtbaren Verhaltens zulässt.
Insgesamt zeigt sich, dass der Bisimulationsbegriff für $\mathcal{C}$-Schichtensysteme zwar Folgerungen bezüglich typkonformer Morphismen zulässt, jedoch keine weitergehenden Aussagen erlaubt. Über das Ergebnis konkreter Aufrufe können wir auf Basis des Bisimulationsbegriffs keinerlei Aussagen ableiten, anders als z.B. bei bisimilaren Transitionssystemen [Rut00] . Dies ist auf die Tatsache zurückzuführen, dass in den Morphismen F_e die Berechnung und auch Aspekte des Kontrollflusses festgelegt sind. Da diese nicht in den Bisimulationsbegriff einfließen, können wir nur Aussagen über die Mengen möglicher Aufrufsequenzen erwarten.

Kapitel 7

Schlussbetrachtungen und Ausblick

Ziel dieser Arbeit war es, einen kategoriellen Modellierungsansatz für komponentenbasierte Architekturen, inbesondere Schichtenarchitekturen, zu entwickeln und zu untersuchen. Unser Modell gliedert sich im wesentlichen in einen statischen und einen dynamischen Teil. Der statische Teil stellt mit graphentheoretischen Mitteln die Konfiguration eines Softwaresystems dar und verwendet kategorientheoretische Methoden nur bei der Definition des Schnittstellenbegriffs, mit Blick auf das zu entwicklende dynamische Modell. Dabei erweist sich die relationale Algebra als nützliches Werkzeug zur Beschreibung und Analyse von Eigenschaften der Konfigurationsgraphen, wenn man sie auf die relationale Struktur anwendet, die diese induzieren. Im Vergleich mit modaler Logik [BdRV01] ist vor allem die Ausdrucksstärke der relationalen Algebra bei der Beschreibung globaler Eigenschaften eines Systems ein entscheidender Vorteil. Wichtige Eigenschaften wie Irreflexivität und Zusammenhang lassen sich in der gewöhnlichen Modallogik nicht ausdrücken [ibd., Ex. 3.15]. Einen Ausweg böte hier nur der Übergang zu den technisch komplizierten hybriden Modallogiken. Außerdem ist die relationale Algebra sehr flexibel, wenn es um den konstruktiven Umgang mit Relationen geht, wie die Definition einer Ordnung auf einer Menge von Äquivalenzklassen in Abschnitt 2.4 zeigt.

Unser Komponentenmodell ist abstrakt und reduziert die statische Spezifikation einer Komponente auf ihre Schnittstellen. Diese stehen bei der archi-

tekturellen Sicht auf ein System im Mittelpunkt. Dennoch kann eine Architekturmodellierung nicht bei der Beschreibung der statischen Konfiguration stehenbleiben. Die Art und Weise, wie das dynamische Zusammenspiel der Komponenten in einem Softwaresystem organisiert ist, hat entscheidende Bedeutung für die Architektur. So reicht es für eine Schichtenarchitektur nicht aus, wenn die Komponenten in Schichten angeordnet sind. Dies lässt sich beispielsweise für azyklische *pipeline*-Architekturen durch eine Transformation stets erreichen [Dob03a]. Dennoch handelt es sich dabei nicht um Schichtenarchitekturen, da der Kontrollfluss gänzlich anders organisiert ist, nämlich durch die Konfiguration vollständig determiniert, während in Schichtenarchitekturen dynamisch darüber entschieden wird.

Unser Ziel war es, ein kategorielles dynamisches Modell aufzustellen, das in verschiedenen Kategorien spezialisiert werden kann, wie beispielsweise in [Dob03a, Bar01]. Die Kategorientheorie erlaubt es, in einer einheitlichen Sprechweise über verschiedenste Gebiete der Mathematik zu reden. So kann man beinahe jede mathematische Struktur zumindest zu einem gewissen Grad auch kategoriell betrachten. Dieser enorme Vorzug ist jedoch zugleich einer der entscheidensten Nachteile der Kategorientheorie: Ohne weitere Annahmen über die Eigenschaften einer Kategorie kann man nur sehr wenig Aussagen treffen; es gibt eine Vielzahl von pathologischen Sonderfällen, die sich völlig anders verhalten als die meisten gewohnten Strukturen. So sind auch wir bei dem Versuch, mithilfe von Coprodukten Entscheidungen zu modellieren, auf das Problem gestoßen, dass dies in vielen Kategorien nicht durchführbar ist. Wir haben die Klasse der lextensiven Kategorien als diejenige identifiziert, die im wesentlichen, d.h. bis auf geringe Zusatzforderungen, alles bietet, was für unsere Modellierung vonnöten ist. Entscheidende Voraussetzungen waren die besonderen Eigenschaften der Coprodukte (Disjunktheit und Universalität) und ihr verträgliches Zusammenspiel mit den Produkten (Distributivität). Bemerkenswert ist der Zusammenhang zwischen lextensiven und adhäsiven Kategorien, die sich jüngst als geeigneter Rahmen erwiesen haben, in dem die Theorie der Doppel-Pushout-Graphtransformationen kategoriell verallgemeinert werden kann [EHPP04]. Während in den extensiven Kategorien das Zusammenspiel von Coprodukten und Pullbacks im Vordergrund steht, definieren sich adhäsive Kategorien durch ein besonderes Verhältnis zwischen Pushouts und Pullbacks, das sich im sogenannten *Van-Kampen-Quadrat* [LS04] widerspiegelt. Beide Begriffe sind eng verwandt:

Eine adhäsive Kategorie ist genau dann extensiv, wenn sie ein striktes initiales Objekt besitzt [ibd., Lemma 11]. Es stellt sich daher die Frage, ob hier ein gemeinsamer Kern von Anforderungen extrahiert werden kann, der für die Modellierung von Berechnungen nötig ist.

Die Berechnung eines Aufrufs definieren wir mithilfe von Homomorphismen zwischen einem Aktivierungsbaum und dem Systemgraphen, wobei wir formal eine Komma-Kategorie verwenden. Die Bäume repräsentieren eine *globale Kontrolle* des Aufrufs (Abschnitt 5.4). So entspricht die Tiefe des Baums der Größe des benötigten Stacks, der die Unteraufrufe verwaltet. Durch die globale Kontrolle vermeiden wir es, in jeder Komponente einen Stack von offenen Unteraufrufen zu verwalten. Jede Komponente entscheidet selbst über den nächsten Aufruf, aber die globale Kontrolle sorgt dafür, dass die Antwort auf einen Aufruf nicht einfach übergangen werden kann.

Hier zeigt sich, warum beispielsweise eine Modellierung mit Petri-Netzen an ihre Grenzen stoßen würde: Da das Schaltverhalten von Petri-Netzen eine lokale Eigenschaft ist, läßt eine globale Kontrolle sich nur schwer mit einem Petri-Netz-basierten Ansatz darstellen. Ein weiter möglicher Ansatz zur Architekturmodellierung besteht in der Verwendung von Kleisli-Kategorien, wie beispielsweise in [Dob03a, Bar01] durchgeführt. Die Begriffsbildung der Kleisli-Kategorie für eine Monade jedoch ist mit einem inhärenten Nichtdeterminismus verbunden, der im Falle eines statischen Kontrollflusses unproblematisch ist [Dob03a], jedoch mit einer dynamischen Entscheidung darüber nicht kompatibel ist. Denn bei einem dynamischen Kontrollfluss muss nach jedem Berechnungschritt in Abhängigkeit vom Berechnungsergebnis der weitere Kontrollfluss bestimmt werden. Denkbar ist zwar, dies mithilfe von Eilenberg-Moore-Algebren zu tun [Dob04], aber damit wäre man nicht mehr in der Kleisli-Kategorie, sondern hätte sofort nach Komposition von Algebra und Coalgebra wieder einen gewöhnlichen Morphismus $X \to X$. Zudem ist nicht klar, wie eine Algebra, die aus einen nichtdeterministischen Berechnungsergenis ein deterministisches erzeugt, sich inhaltlich in ein Komponentenmodell einfügen soll.

Weitere Forschung wird sich auf die nähere Analyse der Möglichkeiten, unseren Ansatz auf stochastische Modelle auszuweiten, richten. Hierzu ist es nötig, eine geeignete lextensive stochastische Kategorie zu finden, bzw. bekannte Kategorien so abzuwandeln, dass sich interessante Modelle ergeben.

Fraglich ist, ob der Weg über die extensive Vervollständigung distributiver Kategorien in anderen Fällen zu fruchtbareren Ergebnissen führt als im Falle von **SRel**.

Ein weiteres Gebiet ist die Modellierung mobiler Architekturen. Diese lassen sich aufgrund der Annahme einer globalen Kontrolle mit unserem Ansatz nicht erfassen. Die Grundidee, Entscheidungen mithilfe von Morphismen in Coprodukten zu modellieren, ist jedoch nicht das entscheidende Hindernis. Vielmehr wird damit die dynamische und lokale Entscheidung über den weiteren Kontrollfluss gut abgebildet. Es stellt sich also die Frage nach einer anderen Modellierung der Kontrollstruktur.

Unser Modellierungsansatz kann auch auf nebenläufige Systeme erweitert werden. Unabhängige *threads* könnten durch mehrere parallele Aktivierungsbäume abgebildet werden, für eine beliebige Interaktion von Prozessen ist jedoch sicherlich ein leichter handhabbarer Kalkül nötig.

Die Entwicklung einer Beschreibungssprache für die Komposition allgemeiner komponentenbasierter Systeme ist ein weiterer offener Aspekt. Für die Zwecke unserer Arbeit hat sich die relationale Algebra als Mittel der Wahl herausgestellt. Ihr Hauptnachteil, der Verlust der Information über Mehrfachkanten eines Graphen, hat für die Fragestellungen unserer Arbeit keine Rolle gespielt. Es stellt sich die Frage, ob dies auch für andere Architekturen gilt, oder ob dort andere Logiken, wie z.B. hybride Modallogiken, vorteilhafter sind. Darüberhinaus ist auch die Frage nach semi-formalen Beschreibungen von Interesse. Existierende Architekturbeschreibungsprachen [SG96, 7.1] könnten durch unser dynamisches Modell mit einer formalen Semantik versehen werden.

Anhang A

Graphen und Relationen

A.1 Graphen

Definition A.1 *Ein* (gerichteter) Graph G *ist ein Quadrupel* (G_V, G_E, src, tar) *bestehend aus einer Menge* G_V *von Knoten,* G_E *von Kanten und Abbildungen* $src : G_V \to G_E$, $tar : G_V \to G_E$, *die einer Kante ihren Anfangs- und Endknoten zuordnen.*

Man beachte, dass ein solcher gerichteter Graph durchaus mehrere Kanten zwischen zwei Knoten haben kann. Wir nennen einen gerichteten Graphen *schlingenfrei*, wenn keine Kante $e \in G_E$ mit $src(e) = tar(e)$ existiert.
Ein Graphhomomorphismus $f : G \to H$ ist ein Paar $< f_V, f_E >$ von Abbildungen $f_V : G_V \to H_V$, $f_E : G_E \to H_E$, für die $src \circ f_E = f_V \circ src$ und $tar \circ f_E = f_V \circ tar$ gilt. Ein Graphhomomorphismus $f : G \to H$, für den ein Graphhomomorphismus $g : H \to G$ existiert, so dass $f \circ g = id_H$ und $g \circ f = id_G$ ist, heißt Isomorphismus. Die Kategorie der Graphen und Graphhomomorphismen bezeichnen wir mit **Grph**.
Ein Graph $G' = (G'_V, G'_E, src', tar')$ mit $G'_V \subseteq G_V$ und $G'_E \subseteq G_E$ heisst *Teilgraph* von G, wenn src' und tar' die Einschränkungen von src, tar auf G'_E sind. Besteht G'_E aus *allen* Kanten $e \in E$, für die $\{src(e), tar(e)\} \subseteq G'_V$ ist, so heisst G' der *von* G'_V *erzeugte Untergrap*h von G.
Ein *markierter Graph* über den Markenmengen D_V, D_E ist ein Graph $G = (G_V, G_E)$ mit Funktionen $l_V : G_V \to D_V$ und $l_E : G_V \to D_E$, den Knoten- bzw. Kantenmarkierungsfunktionen. Wir bezeichnen die Kategorie der markierten Graphen mit **LGrph**. Homomorphismen in **LGrph** sind

Graphhomomorphismen, die die Markierung respektieren.
Ein *getypter Graph* über dem Typgraph TG ist ein Homomorphismus $g : G \to TG$ von einem Instanzgraph G in TG. Ein Morphismus zwischen den getypten Graphen g und h ist ein Homomorphismus f zwischen den Instanzgraphen, so dass $h \circ f = g$ ist. Die Kategorie der markierten, getypten Graphen bezeichnen wir mit **LTGrph**.
Ein azyklischer Graph heißt *Baum*. Die Kategorie der Bäume bezeichnen wir mit **Tree**. Homomorphismen in **Tree** sind beliebige Graphhomomorphismen zwischen den Bäumen. Ein Baum zusammen mit einem ausgezeichneten Element, der Wurzel, heißt *Wurzelbaum*. Homomorphismen von Wurzelbäumen sind Graphhomomorphismen, die das Wurzelelement erhalten. Die entstehende Kategorie heißt **RTree** und ist eine Unterkategorie der Kategorie $\mathbf{Grph}^*$ der punktierten Graphen.
Wir verwenden einen schwachen Zusammenhangsbegriff. Ein gerichteter Graph ist demnach zusammenhängend, wenn je zwei Knoten durch einen ungerichteten Pfad verbunden werden können, d.h. die einzelnen Kanten müssen nicht notwendig in der Richtung, in die sie zeigen, durchlaufen werden.
Als Anwendungskategorie verwenden wir attributierte typisierte Graphen, die wir hier kurz definieren. Sowohl Knoten als auch Kanten können Attribute tragen, die Werte in einer beliebigen Σ-Algebra[1] annehmen können. Die Elemente der Algebra werden als spezielle Knoten aufgefaßt; die Zuweisung der Attribute geschieht durch spezielle Kanten, die sowohl Knoten als auch Kanten mit Attributknoten verbinden können. Wir benötigen einen Zwischenbegriff:

Definition A.2 (E-graph) *Ein* E-graph $G = (V_1, V_2, E_i, s_i, t_i), i = 1, 2, 3$ *besteht aus*

- V_1*, der Menge der Graph-Knoten und* V_2*, der Menge der Attributknoten,*
- E_1*, der Menge der Graph-Knoten,* E_2*, der Menge der Knoten-, und* E_3*, der Menge der Kanten-Attribut-Zuweisungen*

sowie Funktionen

[1] Für die Grundlagen der algebraischen Spezification verwiesen wir auf [Wir90].

- $s_1 : E_1 \to V_1$ *und* $t_1 : E_1 \to V_1$,
- $s_2 : E_2 \to V_1$ *und* $t_2 : E_2 \to V_2$ *und*
- $s_3 : E_3 \to E_1$ *und* $t_3 : E_3 \to V_2$.

Ein E-Graph-Morphismus $f{:}G_1{\to}G_2$ *ist ein Quintupel* $(f_{V_1}, f_{V_2}, f_{E_1}, f_{E_2}, f_{E_3})$ *mit* $f_{V_i} : G_{1,V_i} \to G_{2,V_i}$ *and* $f_{E_j} : G_{1,E_j} \to G_{2,E_j}$ *für* $i = 1, 2$ *und* $j = 1, 2, 3$, *so dass* f *mit allen* s_i *und* t_i *kommutiert.* [EPT04]

Die Kanten E_1 konstituieren die Graphstruktur auf V_1, wohingegen E_2 und E_3 die Attributierung von Knoten und Kanten darstellen.

Definition A.3 (Attributierter Graph) *Sei* $\Sigma = (S, OP)$ *eine Signatur. Ein* attributer Graph *besteht aus einem E-Graph* $G = (V_1, V_2, E_i, s_i, t_i)$ *und einer* Σ*-algebra* D, *so dass* $\dot{\bigcup}_{s \in S} D_s = G_{V_2}$. *Ein* attributierter Graph-Morphismus *ist ein Paar* $f = (f_G, f_A)$ *bestehend aus einem E-Graph-Morphismus* f_G *und einem* Σ*-Algebra-Homomorphismus* f_A, *so dass das folgende Diagramm für alle* $s \in S$ *ein Pullback (vgl. Def. B.9) ist:*

$$\begin{array}{ccc} D_{1,s} & \xrightarrow{f_{A,s}} & D_{1,s} \\ \subseteq \downarrow & & \downarrow \subseteq \\ G_{1,V_2} & \xrightarrow[f_{G,V_2}]{} & G_{2,V_2} \end{array} \quad .$$

Die Kategorie der attributierten Graphen bezeichen wir mit **AGrph**. Analog zu den markierten Graphen kann man auch die attributierten Graphen einer Kategorie von getypten Graphen zugrundelegen und erhält die Kategorie **ATGrph** der attributierten getypten Graphen. Dabei wird für den Typgraphen die finale Σ-Algebra verwendet.

Definition A.4 (Getypte attributierte Graphen) *Sei* $ATG = (TG, Z)$ *ein fester attributierter Graph, der Typgraph. Dabei sei* Z *die finale* Σ*-Algebra. Ein über* ATG getypter attributierter Graph (AG, t) *ist ein attributierter Graph* AG *zusammen mit einem attributierten Graph-Morphismus* $t : AG \to ATG$. *Ein* getypter attributierter Graph-Morphismus *ist ein attributierter Graph-Morphismus* $f : (AG_1, t_1) \to (AG_2, t_2)$ *mit* $t_2 \circ f = t_1$.

A.2 Relationen

Um Eigenschaften von Graphen zu analysieren, verwenden wir die Sprache der relationalen Algebra. Diese erweist sich als sehr flexibel und ausdrucksstark, solange es genügt, die relationalen Eigenschaften von Graphen zu betrachten. Dies bedeutet insbesondere, dass man beim Übergang von einem zu Graphen seiner zugrundeliegenden relationalen Struktur die Information über Mehrfachkanten verliert.
Seien M, N Mengen. Eine Teilmenge $R \subseteq M \times N$ bezeichnet man als Relation zwischen M und N und schreibt $R : M \leftrightarrow N$. Die Menge aller Relationen zwischen M und N bezeichnet man mit $\mathfrak{R}_{MN}$.
Zwei Relationen $R : M \leftrightarrow N$ und $S : N \leftrightarrow P$ kann man komponieren zu einer Relation

$$R;S := \{\langle m,p\rangle \in M \times P \mid \exists n \in N : \langle m,n\rangle \in R \wedge \langle n,p\rangle \in S\}.$$

Weiter hat man das Komplement

$$\bar{R} := \{\langle m,n\rangle \in M \times N \mid \langle m,n\rangle \notin R\}$$

und die Konverse

$$R^T := \{\langle n,m\rangle \in N \times M \mid \langle m,n\rangle \in R\}.$$

einer Relation R. Wichtige spezielle Relationen sind die Nullrelation

$$\mathbb{O} := \emptyset \subseteq M \times N$$

und die Universalrelation

$$\mathbb{L} := M \times N.$$

Als weitere spezielle Relation in $\mathfrak{R}_{MM}$ hat man die Identität

$$\mathbb{I} := \{\langle m,m\rangle \mid m \in M\}.$$

Die Operationen $\cup$ und $\cap$ ergeben sich aus der Mengenlehre, ebenso die Inklusion $\subseteq$. Damit wird $(\mathfrak{R}_{MN}, \cup, \cap, \bar{\cdot})$ zu einer Booleschen Algebra. Diese bildet gemeinsam mit den Operationen $.^T$ und ; und der Identität $\mathbb{I}$ die relationale Algebra. Näheres findet sich in [SS99].

Die Kategorie der relationalen Strukturen $\mathfrak{Rel}$ besitzt als Objekte Paare (X, R), wobei X eine Menge ist und R eine Relation auf X. Morphismen $(X, R) \to (Y, S)$ sind die monotonen Abbildungen $f : X \to Y$, d.h. es muss für alle $x, y \in X$ gelten:

$$\langle x, y\rangle \in R \Rightarrow \langle f(x), f(y)\rangle \in S.$$

Diese Kategorie gilt es streng von der Kategorie **Rel** der Mengen und Relationen zu unterscheiden, die als Objekte Mengen und als Morphismen Relationen besitzt.
Jede Relation $R : M \leftrightarrow M$ kann als Graph dargestellt werden, indem man die Elemente von M als Knoten darstellt und zwischen ihnen eine Kante einfügt, falls sie in R liegen. Man hat somit einen Inklusions-Funktor $K : \mathfrak{Rel} \hookrightarrow \mathbf{Grph}$, der R auf den Graphen $(M, R, proj_1, proj_2)$ abbildet.
Diese Inklusion besitzt einen Linksadjungierten $L : \mathbf{Grph} \to \mathfrak{Rel}$, gegeben durch die Vorschrift, die einem Graphen diejene Relation auf seiner Knotenmenge zuweist, die genau die durch Kanten verbundenen Knoten umfasst. Es ist klar, dass $L \circ K(R)$ eine zu R isomorphe relationale Struktur liefert. Damit ist $\mathfrak{Rel}$ eine reflektive Subkategorie von **Grph** und der Inklusions-Funktor erhält Limiten [HS73, Sec. 36]. Wendet man umgekehrt auf einen Graphen G mit parallelen Mehrfachkanten erst den Funktor L und dann K an, so erhält man einen nicht isomorphen Graphen G', der anstelle der Mehrfachkanten nur einzelne besitzt.

Wir führen nun noch einige wichtige Eigenschaften von Relationen auf, die wir benötigen:

Eine Relation R heißt

- *reflexiv*, falls $\mathbb{I} \subseteq R$ ist,
- *transitiv*, falls $R; R \subseteq R$ ist,
- *symmetrisch*, falls $R = R^T$ ist,
- *antisymmetrisch*, falls $R \cap R^T \subseteq \mathbb{I}$ ist,
- *linkstotal*, falls $\mathbb{I} \subseteq R; R^T$ ist,
- *rechtstotal*, falls $\mathbb{I} \subseteq R^T; R$ ist,

- *bitotal*, falls R links- und rechtstotal ist,
- *Ordnung*, falls sie reflexiv, transitiv, antisymmetrisch ist [SS99, 3.1.4],
- *Striktordnung*, falls sie transitiv und irreflexiv ist,
- *linear*, falls $R \cup R^T = \mathbb{L}$ ist [SS99, 3.1.5],
- *Wohlordnung*, falls R lineare Ordnung ist und für jede Teilmenge $M \subseteq X$ ein kleinstes Element $x \in M$ existiert,
- *Äquivalenzrelation*, falls R reflexiv, symmetrisch und transitiv ist.

Satz A.5 *Jede endliche linear geordnete Menge M mit n Elementen ist isomorph zu $(\{1, \ldots, n\}, <)$ und damit wohlgeordnet.*

Beweis: Isomorphie bedeutet, dass eine bijektive ordnungserhaltende Abbildung

$$h : (\{1, \ldots, n\}, <) \to M$$

existiert. Wir setzen $h(1) = \min M$ und für $i > 1$

$$h(i) = \min\, M \backslash \{h(1), \ldots, h(i-1)\}.$$

Dann ist h bijektiv und ordnungserhaltend. □

Die transitive Hülle R^+ einer Relation ist durch

$$R^+ = \bigcup_{i=1}^{\infty} R^i$$

definiert, die transitiv-reflexive Hülle durch

$$R^+ = \bigcup_{i=0}^{\infty} R^i.$$

Für diese gilt die Rechenregel [SS99, 3.2.6 (ii)]:

$$(R \cup S)^+ = R^+ \cup (R^* S)^+ R^*.$$

Anhang B

Kategorien

Definition B.1 *Eine* Kategorie $\mathcal{C}$ *besteht aus*

(i) einer Klasse $|\mathcal{C}|$ *von Objekten ,*

(ii) einer Klasse paarweise disjunkter Mengen $\mathrm{Hom}_{\mathcal{C}}(A, B)$ *zu jedem Paar* (A, B) *von Objekten. Die Elemente von* $\mathrm{Hom}_{\mathcal{C}}(A, B)$ *heißen Morphismen von* A *nach* B *und wir schreiben* $f : A \to B$*.*

(iii) einer Komposition von Morphismen, d.h. einer Abbildung

$$\circ : \mathrm{Hom}_{\mathcal{C}}(B, C) \times \mathrm{Hom}_{\mathcal{C}}(A, B) \to \mathrm{Hom}_{\mathcal{C}}(A, C)$$

für jedes Tripel (A, B, C) *von Objekten.*

Dabei müssen folgende Bedingungen erfüllt sein:

- *Assoziativität: Für* $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$*,* $g \in \mathrm{Hom}_{\mathcal{C}}(B, C)$ *und* $h \in \mathrm{Hom}_{\mathcal{C}}(C, D)$ *ist* $h \circ (g \circ f) = (h \circ g) \circ f$*.*
- *Identitäten: Für jedes Objekt* B *gibt es einen identischen Morphismus* id_B*, für den*

$$id_B \circ f = f, \quad g \circ id_B = g$$

für alle $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$ *und* $g \in \mathrm{Hom}_{\mathcal{C}}(B, C)$ *gilt.*

Wichtige Beispiele sind die Kategorie **Set** der Mengen und Abbildungen und die Kategorie **Grph** der Graphen und Graphhomomorphismen.

Definition B.2 *Seien $\mathcal{C}, \mathcal{D}$ Kategorien. Eine Abbildung $F : \mathcal{C} \to \mathcal{D}$ heißt Funktor, falls gilt:*

- *F bildet Objekte aus $\mathcal{C}$ auf Objekte in $\mathcal{D}$ ab*
- *F bildet Morphismen $f : X \to Y$ auf Morphismen $Ff : FX \to FY$ ab.*
- *$F(g \circ f) = F(g) \circ F(f)$ für $f \colon A \to B$ und $g \colon B \to C$*
- *$F(id_A) = id_{F(A)}$ für $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$.*

Manchmal bezeichnet man diese Funktoren auch als kovariante Funktoren und betrachtet zusätzlich kontravariante Funktoren, die Morphismen $f : X \to Y$ auf Morphismen $Ff : FY \to FX$ abbilden.
Wichtige Beispiele für Funktoren sind die *Hom-Funktoren*. Sei $X \in |\mathcal{C}|$. Dann ist für $f : Y \to Z$ durch

$$\mathrm{Hom}(X, f) : \mathrm{Hom}(X, Y) \to \mathrm{Hom}(X, Z)$$

$$g \mapsto f \circ g$$

der kovariante Hom-Funktor $\mathrm{Hom}(X, -) : \mathcal{C} \to \mathbf{Set}$ erklärt, und durch

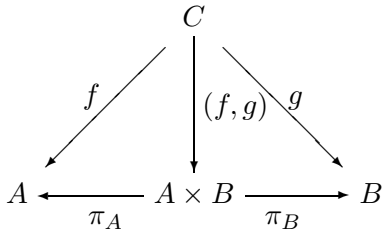
$$\mathrm{Hom}(f, X) : \mathrm{Hom}(Z, X) \to \mathrm{Hom}(Y, X)$$

$$g \mapsto g \circ f$$

der kontravariante Hom-Funktor $\mathrm{Hom}(-, X) : \mathcal{C} \to \mathbf{Set}$.

Definition B.3 *Das Produkt eines Paares (A, B) von Objekten einer Kategorie $\mathcal{C}$ ist ein Tripel $(A \times B, \pi_a, \pi_B)$, wobei $A \times B$ ein Objekt und $\pi_A : A \times B \to A$, $\pi_B : A \times B \to B$ Morphismen in $\mathcal{C}$ sind, so dass gilt:*

Für jedes Objekt $C \in |\mathcal{C}|$ und Morphismen $f : C \to A$, $g : C \to B$ existiert genau ein Morphismus $(f, g) : C \to A \times B$, so dass das Diagramm

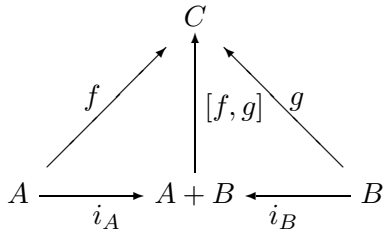


kommutiert.

Den dazu dualen Begriff liefert folgende

Definition B.4 *Das Coprodukt eines Paares (A, B) von Objekten einer Kategorie $\mathcal{C}$ ist ein Tripel $(A + B, i_a, i_B)$, wobei $A + B$ ein Objekt und $i_A : A \to A + B$, $i_B : B \to A + B$ Morphismen in $\mathcal{C}$ sind, so dass gilt:*

Für jedes Objekt $C \in |\mathcal{C}|$ und Morphismen $f : A \to C$, $g : A \to C$ existiert genau ein Morphismus $[f, g] : A + B \to C$, so dass das Diagramm



kommutiert.

Coprodukte werden bisweilen auch als Summen bezeichnet. Wir verwenden unterschiedslos beide Begriffe.

Beispiel B.5

Kategorie	**Objekte**	**Morphismen**	**Produkt**	**Coprodukt**
Set	Mengen	Abbildungen	kartesisches Produkt	disjunkte Ver.
Grp	Gruppen	Gruppenhomom.	direktes Produkt	freies Produkt
CRng	komm. Ringe	Ringhomom.	direktes Produkt	Tensorprodukt

Bemerkung B.6 Nicht in jeder Kategorie existiert zu jedem Paar von Objekten ein Produkt bzw. Coprodukt. Falls die Konstrukte existieren, müssen die Projektionen aus einem Produkt nicht immer surjektiv, die Injektionen in ein Coprodukt nicht immer injektiv sein.
So ist in **Set** die Projektion $\mathbb{N} \times \emptyset \to \mathbb{N}$ die leere Abbildung und nicht surjektiv, in der Kategorie der kommutativen Ringe ist $\mathbb{Q} \otimes_{\mathbb{Z}} \mathbb{Z}/n\mathbb{Z} = 0$ und damit die Injektion $\mathbb{Q} \to \mathbb{Q} \otimes_{\mathbb{Z}} \mathbb{Z}/n\mathbb{Z}$ nicht injektiv[1] (denn für jede $\mathbb{Z}$-bilineare Abbildung B: $\mathbb{Q} \times \mathbb{Z}/n\mathbb{Z} \to R$ in einen kommutativen Ring R ist $B(q, i) = nB(q/n, i) = B(q/n, ni) = B(q/n, 0) = 0$).

[1] Mit 0 bezeichnen wir den trivialen Ring $(0 = 1)$, das terminale Objekt in **CRng**.

Von großer Bedeutung für unsere Arbeit sind spezielle Objekte.

Definition B.7 *(i) Ein Objekt $O \in |\mathcal{C}|$ heißt initial, falls für jedes Objekt X genau ein Morphismus von O nach X existiert.*

(ii) Ein Objekt $I \in |\mathcal{C}|$ heißt terminal, falls für jedes Objekt X genau ein Morphismus von X nach I existiert.

(iii) Ein Objekt heißt Nullobjekt, falls es zugleich initial und terminal ist[2]*.*

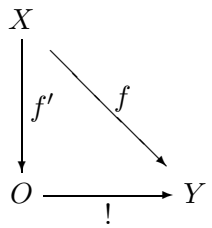
Initiale, terminale und Nullobjekte sind bis auf Isomorphie eindeutig. Daher gilt insbesondere: Existiert in $\mathcal{C}$ ein Nullobjekt, so sind alle initialen und alle terminalen Objekte Nullobjekte.

Bezeichnung B.8 *Für die eindeutig bestimmten Morphismen schreibt man in der Fachliteratur oft einfach $! : O \to X$ und $! : X \to O$. Wir schliessen uns dieser Konvention an.*

Für ein beliebiges Objekt $X \in |\mathcal{C}|$ gelten folgende Isomorphien:

$$X \cong X \times I, \quad X + O \cong X$$

Die Isomorphismen sind dabei durch $< id_X, ! >$ bzw. $[id_X, !]$ gegeben.
Falls ein Morphismus $I \to O$ existiert, so sind I und O Nullobjekte, denn $! : O \to I$ ist invers. In diesem Fall sind wegen der Isomorphie alle initialen und terminalen Objekte Nullobjekte.
In extensiven Kategorien sind initiale Objekte *strikt*, d.h. allgemein folgt aus $X \to O$ dass $X \cong O$ gilt. Ein Morphismus $f : X \to Y$, der durch O faktorisiert, heißt *Nullmorphismus*:



[2]Die gängigen englischen Bezeichnungen lauten *initial*, *terminal* und *zero object*. Bisweilen wird jedoch das terminale Objekt als *null object* und das initiale als *conull object* bezeichnet. Wir verwenden jedoch *Nullobjekt* in jedem Fall als deutsche Übersetzung von *zero object*.

Da initiale Objekte strikt sind, folgt, dass extensive Kategorien mit Nullobjekten *degeneriert* sind: Da O zugleich initial und terminal ist, existiert zu jedem A ein $A \to O$, und damit gilt $A \cong O$.

Man sagt, dass eine Kategorie *endliche (Co-)Produkte* besitzt, wenn sie binäre (Co-)Produkte und terminale Objekte (bzw. initiale Objekte) besitzt (letztere werden machmal auch als *unäre (Co-)produkte* bezeichnet).

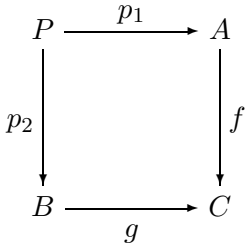
Der Begriff der Kategorie ist sehr abstrakt gefaßt. Es gibt keinen allgemeinen Elementbegriff, da die Objekte nicht notwendig Mengen sein müssen (auch wenn dies in vielen Standardbeispielen der Fall ist).

Als Ersatz verwendet man sogenannte globale Elemente. Dies sind Morphismen $f : I \to X$ von einem terminalen Objekt in ein Objekt X. In **Set** entspricht dies Abbildungen von einer einelementigen Menge $\{*\}$ in eine Menge X, und es gibt eine kanonische Bijektion zwischen globalen Elementen und gewöhnlichen, nämlich $x = f(*)$.

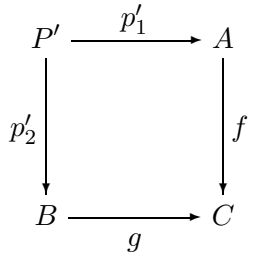
In **Set** sind zwei parallele Morphismen $f, g : X \to Y$ genau dann gleich, wenn $f \circ a = g \circ a$ für alle globalen Elemente a von X ist. Diese Eigenschaft gilt jedoch nicht in allen Kategorien. Wenn sie zutrifft, heißt eine Kategorie *wohlpunktiert*. Es ist klar, dass die Existenz von Nullobjekten ein 'knock-out'-Kriterium für die sinnvolle Verwendung von globalen Elementen darstellt, denn sie impliziert, dass es zu jedem Objekt genau ein globales Element gibt. Jedes globale Element $f : I \to X$ ist split-mono, da mit $! : X \to I$ gilt: $! \circ f = id_I$.

Ein Objekt $A \in |\mathcal{C}|$ heißt *nicht-leer*, wenn es globale Elemente besitzt, d.h. wenn mindestens ein Morphismus $I \to A$ existiert. Es ist klar, daß initiale Objekte in einer extensiven Kategorie $\mathcal{C}$ nur dann nicht-leer sein können, wenn $\mathcal{C}$ degeneriert ist.

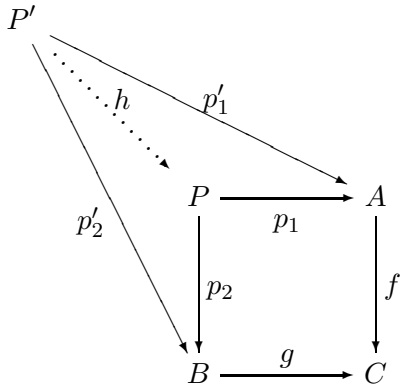
Definition B.9 *Ein kommutatives Diagramm*



heißt Pullback, falls für jedes kommutative Diagramm der Form



genau ein Morphismus $h : P' \to P$ existiert, so dass



kommutiert. In dieser Situation sagt man bisweilen, das p_2 ein Pullback von f entlang g ist (und analog p_1 ein Pullback von g entlang f ist).

Beispiel B.10 In **Set** konstruiert man das Pullback zweier Funktionen $f : A \to C$ und $g : B \to C$ wie folgt [BW99, 9.3.1]: Man setzt

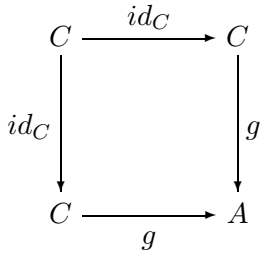
$$P = \{\langle a, b\rangle \mid f(a) = g(b)\}$$

und $p_1(a, b) = a$, $p_2(a, b) = b$.
Wir betrachten noch den wichtigen Spezialfall, dass $B \subseteq C$ und g die Inklusion ist. Dann gilt

$$\begin{aligned} P &= \{\langle a, c\rangle) \mid f(a) = c, c \in B\} \\ &= \{\langle a, f(a)\rangle \mid f(a) \in B\} \\ &\cong \{a \mid f(a) \in B\} \\ &= f^{-1}(B), \end{aligned}$$

wobei $\cong$ eine Bijektion ist, also ein Isomorphismus in **Set**. Die Pullback-Objekte in **Set** entsprechen in diesem Spezialfall also *Urbildern*.

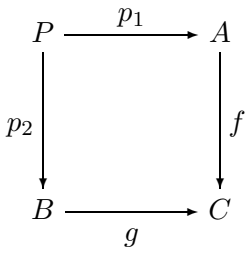
Lemma B.11 *Das Diagramm*



ist genau dann ein Pullback, wenn g Monomorphismus ist.

Beweis: Sei g Monomorphismus und Morphismen $\alpha, \beta : D \to C$ mit $g \circ \alpha = g \circ \beta$ gegeben. Dann ist $\alpha = \beta$ und wir können jeden der beiden als vermittelnden Morphismus auswählen. Ist umgekehrt das Diagramm ein Pullback und Morphismen $\alpha, \beta : D \to C$ mit $g \circ \alpha = g \circ \beta$ gegeben, so existiert ein vermittelnder Morphismus $\gamma : D \to C$ mit $id_C \circ \gamma = \alpha$ und $id_C \circ \gamma = \beta$. Daraus folgt $\alpha = \beta$, also ist g Monomorphismus. □

Lemma B.12 *(Mono-Lemma für Pullbacks) Falls in einem Pullback-Diagramm*



$g : B \to C$ ein Monomorphismus ist, so ist auch p_1 mono.

Beweis: Seien g ein Monomorphismus und $k, h : X \to P$ Morphismen mit $p_1 \circ h = p_1 \circ k =: \alpha$, wobei X ein beliebiges Objekt ist. Wir müssen zeigen, dass dann $h = k$ ist. Es gilt

$$f \circ p_1 \circ h = f \circ p_1 \circ k,$$

und wegen der Kommutativität des Quadrats ist

$$g \circ p_2 \circ h = g \circ p_2 \circ k,$$

also, da g Monomorphismus ist,

$$p_2 \circ h = p_2 \circ k =: \beta.$$

Da $f \circ \alpha = f \circ p_1 \circ h = g \circ p_2 \circ h = g \circ \beta$ ist, gibt es nach der Pullback-Eigenschaft *genau einen* Morphismus $\gamma : X \to P$ mit

$$(i)\ p_1 \circ \gamma = \alpha, \qquad (ii)\ p_2 \circ \gamma = \beta.$$

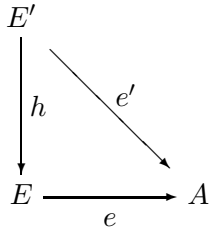
Sowohl h als auch k erfüllen die Bedingungen (i) und (ii). Wegen der Eindeutigkeit von γ folgt $h = k = \gamma$. □

Vor dem Hintergrund dieses Lemmas kann man Pullbacks entlang von Monomorphismen als Urbildkonstruktionen interpretieren: Da g und p_1 Monomorphismen sind, ist p_2 die Einschränkung von f auf das Unterobjekt P von A, das durch die Einbettung p_1 gegeben ist. Man schreibt dann wie in **Set** $f^{-1}(B)$ für P [McL92, 4.3].
In Kategorien mit endlichen Produkten läßt sich der Begriff des Pullbacks auf den des Equalizers zurückführen:

Definition B.13 *Seien $f, g : A \to B$ zwei parallele Morphismen in einer Kategorie $\mathcal{C}$. Ein Paar (E, e) aus einem Objekt E und einem Morphismus $e : E \to A$ heißt Equalizer von f und g, falls gilt:*

(i) $f \circ e = g \circ e$

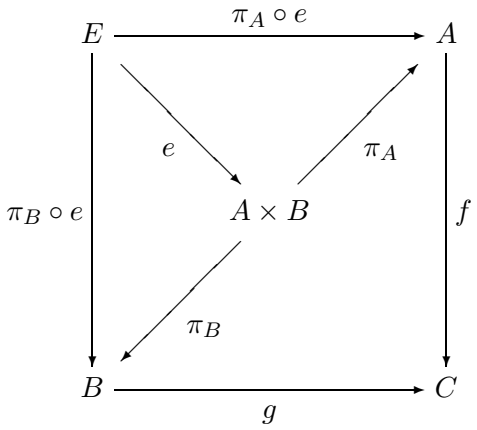
(ii) Für jeden Morphismus $e' : E' \to A$ mit $f \circ e' = g \circ e'$ existiert genau ein Morphismus $h : E' \to E$, so dass das Diagramm



kommutiert.

In **Set** ist der Equalizer durch die Menge $E = \{a \in A \mid f(a) = g(a)\}$ und die Einbettung $e : E \to A$ gegeben. Das Pullback läst sich in **Set** als Equalizer des Produkts konstruieren, eine kanonische Konstruktion, die in allen Kategorien mit Produkten und Equalizern möglich ist:

Satz B.14 *Sei* $A \xrightarrow{f} C \xleftarrow{g} B$ *ein Paar von* $\mathcal{C}$*-Morphismen. Ist* $(A \times B, \pi_A, \pi_B)$ *ein Produkt von* (A, B) *und* (E, e) *ein Equalizer von* $f \circ \pi_A$ *und* $g \circ \pi_B$*, so stellt das äußere Quadrat ein Pullback dar:*



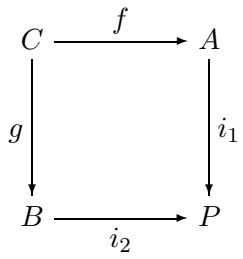
Beweis: Kommutativität des Diagramms ist klar. Morphismen $r : Q \to A$ und $s : Q \to B$ mit $f \circ r = g \circ s$ induzieren aufgrund der Produkteigenschaft einen eindeutigen Morphismus $h : Q \to A \times B$. Wegen der Equalizer-Eigenschaft faktorisiert dieser eindeutig über E. □

Es ergibt sich daraus, dass eine Kategorie mit Produkten und Equalizern Pullbacks besitzt. Da man auch umgekehrt den Equalizer eines Paars von Morphismen als Pullback von

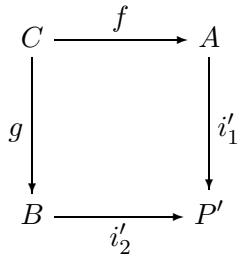
$$A \xrightarrow{(id_B, f)} A \times B \xleftarrow{(id_B, g)} A$$

konstruieren kann, gilt: Eine Kategorie mit Produkten hat genau dann Pullbacks, wenn sie Equalizer hat. Da die Konstruktion eines Equalizers einfacher ist als die eines Pullbacks, hilft diese Betrachtungsweise bisweilen weiter.
Nicht jede Kategorie besitzt Pullbacks: Die Kategorie der Standard-Borel-Räume mit Wahrscheinlichkeitsmaßen, in der die Morphismen surjektive und maßerhaltende Abbildungen sind, besitzt i.a. keine Pullbacks [Dob03b]. Der zum Pullback duale Begriff ist der des Pushouts.

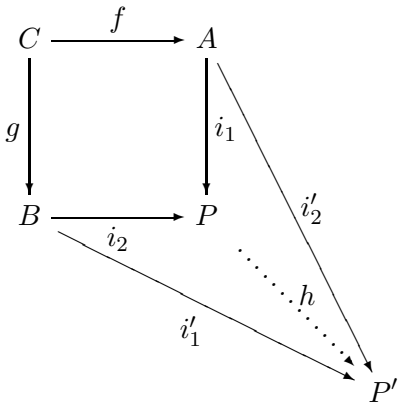
Definition B.15 *Ein kommutatives Diagramm*



heißt Pushout, falls für jedes kommutative Diagramm der Form



genau ein Morphismus $h : P \to P'$ existiert, so dass



kommutiert.

Beispiel B.16 In **Set** existiert das Pushout von zwei Morphismen $f : C \to A$ und $g : C \to B$ stets. Das Pushout-Objekt lässt sich aus dem Coprodukt-Objekt $A + B$ gewinnen, indem man den Quotienten bezüglich der kleinsten Äquivalenzrelation $\approx$ bildet, für die $f(c) \approx g(c)$ für alle $c \in C$ ist.

In **Grph** bildet man das Pushout zweier Morphismen $f : C \to G$ und $g : C \to H$ wie folgt: Zunächst bildet man komponentenweise die Pushouts P_V von $f_V : C_V \to G_V$ und $g_V : C_V \to H_V$ bzw. P_E von $f_E : C_E \to G_E$ und $g_E : C_E \to H_E$. Nun sind noch src, tar und eventuelle Markierungsfunktionen l_V, l_E zu definieren. Diese ergeben sich eindeutig aus der Pushouteigenschaft und der Homomorphiebedingung.

Unter dem allgemeinen Begriff des Limes lassen sich die bisherigen Konstruktionen wie Produkte, Equalizer und Pullback zusammenfassen [HS73, §20]. Es gilt dann: Jeder (endliche) Limes läßt sich aus (endlichen) Produkten und Equalizern konstruieren. Daher nennt man eine Kategorie mit endlichen Produkten und Equalizern auch *endlich-vollständig*.

Eine wichtige Konstruktion, mit der man aus gegebenen Kategorien neue gewinnt, ist die Bildung der *Komma-Kategorie* [Mac97, II.6.]. In der allgemeinsten Fassung hat man Kategorien $\mathcal{E}, \mathcal{C}, \mathcal{D}$ und Funktoren $S : \mathcal{D} \to \mathcal{C}$, $T : \mathcal{E} \to \mathcal{C}$ gegeben und definiert die Komma-Kategorie $T \downarrow S$ wie folgt: Objekte sind alle Tripel $\langle E, D, f\rangle$ mit $E \in |\mathcal{E}|$, $D \in |\mathcal{D}|$ und $f \in \mathrm{Hom}_{\mathcal{C}}(TE, SD)$, Morphismen $\langle E, D, f\rangle \to \langle E', D', f'\rangle$ sind Paare $\langle k, h\rangle$ von $\mathcal{C}$-Morphismen $k : E \to E'$, $h : D \to D'$, so dass $f' \circ Tk = Sh \circ f$ ist.
Wichtig sind vor allem die Spezialfälle, in denen für $\mathcal{E}$ oder $\mathcal{D}$ die Kategorie $\mathbf{1}$ eingesetzt wird, die aus einem einzigen Objekt $*$ und einen einzigen Morphismus $id : * \to *$ besteht. Ein Objekt $B \in |\mathcal{C}|$ induziert dann einen Funktor $F_B : \mathbf{1} \to \mathcal{C}$, indem man $F_B(*) = B$ und $F_B(id) = id_B$ setzt. Setzt man $\mathcal{E} = \mathbf{1}$, $\mathcal{D} = \mathcal{C}$, $T = F_B$ und nimmt man für S den identischen Funktor, so notiert man die entstehende Kategorie auch als $B \downarrow \mathcal{C}$ und bezeichnet sie als die *Kategorie der Objekte unter* B. Vertauscht man die Rollen von $\mathcal{E}$ und $\mathcal{D}$, so erhält man die *Kategorie* $\mathcal{C} \downarrow B$ *der Objekte über* B.

Index

Literaturverzeichnis

[AAG03] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Categories of containers. In *Proceedings of FOSSACS 2003*, pages 23–38, 2003.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[AR02] Farhad Arbab and Jan J. M. M. Rutten. A coinductive calculus of component connectors. Technical report, SEN-R0216, CWI, Amsterdam, 2002.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers — Principles, Techniques, and Tools*. Addison Wesley, 1986.

[Bar01] Luís Manuel Barbosa. *Components as Coalgebras*. PhD thesis, Universidade do Minho, 2001.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998, 2nd edition 2003.

[BCKW98] Lionel C. Briand, S. Jeromy Carrière, Rick Kazman, and Jürgen Wüst. A comprehensive framework for architecture evaluation. Technical Report IESE Report No. 46.98/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, 1998.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*. Cambridge University Press, 2001.

[Blo82] Stephen L. Bloom. Elgot's analysis of monadic computation. *Fundamenta Informaticae*, 2:171–186, 1982.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern–Oriented Software Architecture*, volume 1. John Wiley and Sons, 1996.

[Bor94] Francis Borceux. *Handbook of Categorical Algebra 2.* Number 51 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems.* Springer, 2001.

[BW99] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Les Publications CRM, Montreal, 1999.

[CL01] J. Robin B. Cockett and Stephen Lack. The extensive completion of a distributive category. *Theory and Applications of Categories, vol. 8, No. 22, 2001, 541-554*, 2001. Available online under `http://www.tac.mta.ca/tac/`.

[CLW93] Aurelio Carboni, Stephen Lack, and Robert F.C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra 84 (1993), 145-158*, 1993.

[Coc93] J. Robin B. Cockett. Introduction to distributive categories. *Mathematical Structures in Computer Science (1993), vol. 3, 277-307*, 1993.

[Dob03a] Ernst-Erich Doberkat. Pipelines: Modelling a software architecture through relations. *Acta Informatica 40, 37-79*, 2003.

[Dob03b] Ernst-Erich Doberkat. Semi-pullbacks and bisimulations in categories of stochastic relations. In J. Parrow J.C.M. Baeten, J.K. Lenstra and G.J. Woeginger, editors, *Proceedings ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*, pages 996–1007. Springer, 2003.

[Dob04] Ernst-Erich Doberkat. Derandomizing probabilistic semantics through Eilenberg-Moore algebras for the Giry monad. Technical Report 149, Lehrstuhl für Software-Technologie, Universität Dortmund, 2004.

[DvdHT05] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. A comprehensive approach for the development of modular software architecture description languages. *ACM Trans. Softw. Eng. Methodol.*, 14(2):199–245, 2005.

[dVR97] Erik P. de Vink and Jan J. M. M. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 460–470, London, UK, 1997. Springer-Verlag.

[dW99] Desmond Francis d'Souza and Alan Cameron Wills. *Objects, Components and Frameworks with UML.* Addison Wesley, 1999.

[EEKR99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation: applications, languages, and tools*, volume 2. World Scientific, River Edge, NJ, USA, 1999.

[EGRW95] Hartmut Ehrig, Martin Große-Rhode, and Uwe Wolter. On the role of category theory in the area of algebraic specification. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *COMPASS/ADT*, volume 1130 of *Lecture Notes in Computer Science*, pages 17–48. Springer, 1995.

[EHL+99] Hartmut Ehrig, Reiko Heckel, Mercè Llabrés, Fernando Orejas, and Julia Padberg. Basic properties of double-pullback graph transitions. Technical report, No. 99-02, Techn. Univ. Berlin, FB 13, 1999.

[EHPP04] Hartmut Ehrig, Annegret Habel, Julia Padberg, and Ulrike Prange. Adhesive high-level replacement categories and systems. In Hartmut Ehrig, editor, *Proceedings of the Second International Conference on Graph Transformations*, volume 3256 of *LNCS*, pages 144–160. Springer, 2004.

[Elg71] Calvin C. Elgot. Algebraic theories and program schemes. In Erwin Engeler, editor, *Symposium on Semantics of Algorithmic Languages*, pages 71–88. Springer-Verlag, 1971.

[Elg75] Calvin C. Elgot. Monadic computation and iterative algebraic theories. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, pages 175–230. North-Holland Publishers, 1975.

[EM45] Samuel Eilenberg and Saunders MacLane. General theory of natural equivalences. *Transactions of the American Mathematical Society, 58, 231-294*, 1945.

[EMC+99] Hartmut Ehrig, Bernd Mahr, Felix Cornelius, Martin Große-Rhode, and Philip Zeitz. *Mathematisch-Strukturelle Grundlagen der Informatik*. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[EPT04] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graph transformation. In *ICGT*, pages 161–177, 2004.

[EW67] Samuel Eilenberg and Jesse Wright. Automata in general algebras. *Information and Control*, 11:452–470, 1967.

[FM96] José L. Fiadeiro and Tom Maibaum. A mathematical toolbox for the software architec. In *IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design*, page 46, Washington, DC, USA, 1996. IEEE Computer Society.

[Fv90] Peter Freyd and Andre Ščedrov. *Categories, Allegories*. North Holland, 1990.

[GJM03] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[Gog91] Joseph A. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.

[Gol79] Robert Goldblatt. *Topoi. The Categorial Analysis of Logic*. North-Holland Publishing Company, 1979.

[GT00] Volker Gruhn and Andreas Thiel. *Komponentenmodelle*. Addison-Wesley, 2000.

[HC01] George T. Heineman and William T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[HM01] Dan Hirsch and Ugo Montanari. Two graph-based techniques for software architecture reconfiguration. *Electr. Notes Theor. Comput. Sci.*, 51, 2001.

[HS73] Horst Herrlich and George Strecker. *Category Theory.* Allyn and Bacon, 1973.

[HT04] Reiko Heckel and Sebastian Thöne. Behavior-preserving refinement relations between dynamic software architectures. In José Luiz Fiadeiro, Peter D. Mosses, and Fernando Orejas, editors, *WADT*, volume 3423 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2004.

[IEE00] IEEE. Recommended practice for architectural description of software-intensive systems. IEEE Computer Society, New York, 2000.

[JNW96] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. 127(2):164–185, June 1996. LICS '93 special issue. RS-94-7.

[Joh77] Peter T. Johnstone. *Topos Theory.* Academic Press, 1977.

[JR01] Michael Johnson and Robert Rosebrugh. Coproducts in categorical information system specification. In *Proceedings of SCI 2001*, volume XIV, pages 145–150, 2001.

[JRW02] Micheal Johnson, Robert Rosebrugh, and R.J. Wood. Entity-relationship-attribute designs and sketches. *Theory and Applications of Categories, vol. 10, No. 3, 2002, 94-112*, 2002. Available online under `http://www.tac.mta.ca/tac/`.

[KLV05] A. Steven Klusener, Ralf Lämmel, and Chris Verhoef. Architectural modifications to deployed software . *Sci. Comput. Program.*, 54(2-3):143–211, 2005.

[KW93] Wafaa Khalil and Robert F.C. Walters. An imperative language based on distributive categories II. *Informatique théorique et Applications, vol. 27, no. 6, 503-522*, 1993.

[Lag97] Jeffrey C. Lagarias. The $3x + 1$ problem and its generalizations. In J. Borwein, editor, *Proceedings of the Organic Mathematics Workshop*, volume 20 of *Canadian Mathematical Society Conference proceedings*, pages 305–334. American Mathematical Society, 1997.

[Law63] Bill Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences of the United States of America*, 50(1):869–872, 1963.

[Lev02] Paul Blain Levy. Adjunction models for call-by-push-value with stacks. *Electr. Notes Theor. Comput. Sci.*, 69, 2002.

[LEW97] Jacques Loeckx, Hans-Dieter Ehrich, and Markus Wolf. *Specification of abstract data types*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[LLJ04] Xiaoshan Li, Zhiming Liu, and He Jifeng. A formal semantics of UML sequence diagrams. In *Proceedings Australian Software Engineering Conference (ASWEC'04)*, page 168, 2004.

[LP97] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, Upper Saddle River, NJ, USA, 1997.

[LS97] Bill Lawvere and Stephen Schanuel. *Conceptual Mathematics: a First Introduction to Categories*. Cambridge University Press, 1997.

[LS04] Steven Lack and Paweł Sobociński. Adhesive categories. *Proceedings of FOSSACS 2004*, LNCS 2987, pages:273–288, 2004.

[MA86] Ernest G. Manes and Michael A. Arbib. *Algebraic approaches to program semantics*. Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[Mac97] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, 1997.

[Man92] Ernest G. Manes. *Predicate Transformer Semantics.* Cambridge University Press, 1992.

[McL92] Colin McLarty. *Elementary Categories, Elementary Toposes.* Clarendon Press, 1992.

[MP98] Gianfranco Mascari and Marco Pedicini. Types and dynamics in partially additive categories. In J. Gunawardena, editor, *Idempotency*, volume 11 of *Publications of the Newton Institute*, pages 112–132. Cambridge University Press, 1998.

[MRRR02] Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. Modeling software architectures in the unified modeling language. *ACM Trans. Softw. Eng. Methodol.*, 11(1):2–57, 2002.

[Mét96] Daniel Le Métayer. Software architecture styles as graph grammars. In *SIGSOFT '96: Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, pages 15–23. ACM Press, 1996.

[NA03] Oscar Nierstrasz and Franz Achermann. A calculus for modeling software components. In *FMCO 2002 Proceedings*, volume 2852 of *Lecture Notes in Computer Science*, pages 339–360. Springer, 2003.

[Pan97] Prakash Panangaden. Stochastic techniques in concurrency. Technical report, BRICS Report, 1997.

[Par72] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[Poi86] Axel Poigné. Algebra categorically. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Programming*, volume 240 of *LNCS*, pages 76–102. Springer Verlag, 1986.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: foundations*, volume 1. World Scientific, River Edge, NJ, USA, 1997.

[Rut00] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.

[Rut05] Jan J. M. M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. Technical report, SEN-R05, CWI, Amsterdam, 2005.

[Sch93] J.R. Schoenfield. Axioms of set theory. In Jon Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, Amsterdam, 8th edition, 1993.

[SG96] Mary Shaw and David Garlan. *Software Architecture – Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[Sri98] Sashi Mohan Srivastava. *A Course on Borel Sets*. Springer, 1998.

[SS99] Gunther Schmidt and Thomas Ströhlein. *Relations and Graphs – Discrete Mathematics for Computer Scientists*. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[Szy98] Clemens Szyperski. *Component Software*. Addison-Wesley, 1998.

[Tan01] Andrew Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2001.

[Tan02] Andrew Tanenbaum. *Computer Networks*. Prentice Hall, 2002.

[Tho98] Craig Thompson. Workshop report. Technical Report available under http://www.objs.com/workshops/ws9801/report.html, Workshop on Compositional Software Architectures, Monterey, USA, 1998.

[Wag86] Eric G. Wagner. Categories, data types and imperative languages. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Programming*, volume 240 of *LNCS*, pages 143–162. Springer Verlag, 1986.

[Wag02] Eric G. Wagner. Algebraic specifications: some old history and new thoughts. *Nordic J. of Computing*, 9(4):373–404, 2002.

[Wal91] Robert F.C. Walters. *Categories and Computer Science*, volume 18 of *Cambridge Computer Science Texts*. Cambridge University Press, 1991.

[Wal92] Robert F.C. Walters. An imperative language based on distributive categories. *Mathematical Structures in Computer Science (1992), vol. 2, 249-256*, 1992.

[WF98] Michel Wermelinger and José Luiz Fiadeiro. Connectors for mobile programs. *IEEE Trans. Softw. Eng.*, 24(5):331–341, 1998.

[WF02] Michel Wermelinger and José Luiz Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Sci. Comput. Program.*, 44(2):133–155, 2002.

[Wir90] Martin Wirsing. Algebraic specification. *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 675–788, 1990.

[WKW95] Eric G. Wagner, Wafaa Khalil, and Robert F.C. Walters. Fixpoint semantics for programs in distributive categories. *Fundamenta Informaticae, 22, 187-202*, 1995.