

Transformation objektorientierter Systeme basierend auf algebraischen Graph-Transformationen

vorgelegt von
Diplom-Informatiker (FH)
Christoph Schulz
aus Kattowitz

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Sabine Glesner, Technische Universität Berlin
Gutachter: Prof. Dr. Hartmut Ehrig, Technische Universität Berlin
Prof. Dr. Gabriele Taentzer, Universität Marburg
Prof. Dr. Michael Löwe, FHDW Hannover

Tag der wissenschaftlichen Aussprache: 10. Mai 2010

Berlin 2010
D 83

Diplom-Informatiker (FH) Christoph Schulz
Christoph.Schulz@fhdw.de

Transformation objektorientierter Systeme basierend auf algebraischen Graph-Transformationen

31. Mai 2010

Zusammenfassung

Die Anpassung bestehender Software an neue Anforderungen ist seit dem Beginn der Software-Entwicklung ein wichtiges Thema. Die Erweiterung von Software um neue Funktionalität wird erleichtert oder gar erst ermöglicht durch das Software-Refactoring: die strukturelle Verbesserung von Software unter Beibehaltung des Verhaltens, mit dem Ziel, zusätzliche Funktionalität danach einfacher umsetzen zu können. Leider werden die Transformation objektorientierter Daten und die Transformation objektorientierter Software für gewöhnlich getrennt voneinander betrachtet. In dieser Arbeit wird stattdessen die Transformation ganzer objektorientierter *Systeme*, also von Programmen, laufenden Programm-Ausprägungen (Prozessen) und zugehörigen Daten, in den Fokus gerückt. Dazu wird im ersten Schritt ein konzeptionelles Modell zur angemessenen Beschreibung objektorientierter Systeme entwickelt. Dieses Modell wird im zweiten Schritt durch eine Spezifikation mit Prädikaten und positiven Horn-Formeln formalisiert. Darauf aufbauend werden Transformationen von Systemen – analog zu Graph-Transformationen – als Spans von Homomorphismen modelliert. Das zentrale Ergebnis der Arbeit ist die kanonische Ableitung der Migration von Daten, Programmen und Prozessen aus der Transformation des zugrunde liegenden Schemas, so dass eine System-Transformation durch die Transformation des Schemas eindeutig festgelegt werden kann. Die Behandlung der Komposition von Transformationen und ein Ausblick auf mögliche Erweiterungen des Modells runden die Arbeit ab.

Abstract

Since the beginning of software development, making existing software conformant to new demands has always been an important topic. The extension of software by new functionality is facilitated or even made possible by software refactoring: improving the structure of software without changing its behaviour in order to implement additional functionality more easily. Unfortunately, the transformation of object-oriented data and the transformation of object-oriented software is usually examined separately. This work focuses on the transformation of whole object-oriented *systems*, i. e. of programs, running program instantiations (processes), and associated data. In order to achieve this goal, first a conceptual model for an appropriate description of object-oriented systems is developed. This model is then formalised by a specification with predicates and positive Horn formulas. Analogously to graph transformations, system transformations are then modelled by spans of homomorphisms. The main result of this work is the canonical deduction of the migration of data, programs, and processes from the transformation of the underlying schema, such that the transformation of a whole system can be uniquely determined by the transformation of the schema only. The work is rounded off by the treatment of the composition of transformations and an outlook of possible model extensions.

Inhaltsverzeichnis

Teil I Problemstellung

1	Einleitung	3
1.1	Transformation von Datenschemata	4
1.2	Migration der Daten	5
1.3	Migration der Software	7
1.4	Aufbau der Arbeit	8
2	Themenbezogene Arbeiten	11
2.1	Mathematisches Modell für objektorientierte Daten	11
2.2	Mathematisches Modell für objektorientierte Software	13
2.2.1	Das <i>Java</i> -Hyperkanten-Modell	13
2.2.2	Das <i>TAAL</i> -Modell	14
2.3	Mathematisches Modell für Transformationen	15
2.4	Ableitung von Ausprägungs-Migrationen aus Schema-Transformationen	17
2.5	Komposition von Schema-Transformationen	20
2.6	Migration von Daten, Programmen und Prozessen	20
3	Fallstudie: Problemstellung	23
3.1	Ausgangssituation	23
3.2	Phase 1: Hinzunahme eines neuen Versicherungsprodukts	23
3.3	Phase 2: Hinzunahme ausländischer Kunden	24
3.4	Phase 3: Konzentration auf Geschäftskunden	25
3.5	Diskussion	25

Teil II Konzeptionelles Modell

4	Transformation von Software und Datenbanken	31
4.1	Software	31
4.2	Datenbanken	33
4.3	Systeme und Welten	33
4.4	Transformationen	34
4.5	Refactorings	34
4.5.1	Software-Refactorings	34

IV	Inhaltsverzeichnis	
	4.5.2 Datenbank-Refactorings	37
	4.5.3 Erweiterungen	38
	4.5.4 Löschungen	39
	4.5.5 Definitionen	39
	4.6 Entwicklungen	40
	4.7 Induzierte Migrationen	40
5	Objektorientierte Systeme	43
5.1	Datenbanken	44
5.1.1	Klassen und Objekte	44
5.1.2	Assoziationen und Verknüpfungen	46
5.1.3	Primitive Datentypen und Werte	47
5.1.4	Verweise	48
5.1.5	Die <i>Null</i> -Klasse und <i>Null</i> -Objekte	49
5.2	Software	50
5.2.1	Grundlegende Konzepte	51
5.2.1.1	Variablen	51
5.2.1.2	Konstanten	52
5.2.1.3	Operationen und Parameter	52
5.2.1.4	Nachrichten und Argumente	54
5.2.1.5	Methoden	56
5.2.1.6	Polymorphie	57
5.2.1.7	Abarbeitung einer Nachricht	57
5.2.2	Beispiele	60
5.2.2.1	Beispiel für Daten-Manipulation: Die Zuweisung	61
5.2.2.2	Beispiel für Anweisungen: Die Fallunterscheidung	63
5.2.2.3	Beispiel für Ausdrücke: Die Ganzzahl-Addition	63
6	Fallstudie: Konzeption	67
6.1	Ausgangssituation	67
6.2	Phase 1: Hinzunahme eines neuen Versicherungsprodukts	67
6.3	Phase 2: Hinzunahme ausländischer Kunden	71
6.4	Phase 3: Konzentration auf Geschäftskunden	73
<hr/>		
Teil III Mathematisches Modell		
<hr/>		
7	Überblick	79
8	Schemata, Daten und Prozesse	81
8.1	Klassen und Objekte	81
8.2	Assoziationen und Verknüpfungen	82
8.3	Vererbung	83
8.4	Komponenten	87
8.5	Datentypen und Werte	90
8.6	Operationen und Nachrichten	93
8.7	System-Spezifikation	95
9	Kategorielle Strukturen, Übergänge und Eigenschaften	99
9.1	Die Kategorien $\mathbf{Alg}(MP)$ und $\mathbf{Alg}(M')$	99

9.2	Der Übergang von $\mathbf{Alg}(M')$ zu $\mathbf{Alg}(M)$	106
9.3	Pullbacks, Pushouts und spezielle Homomorphismen	109
10	Programme	111
10.1	Aktionen und DPO-Transformationen	112
10.2	Konsistente Aktionen und Erhaltung der Axiome	116
10.2.1	Vervollständigende Homomorphismen	116
10.2.2	Kanten zurückziehende Spans	118
10.2.3	Konsistente Aktionen	119
10.3	Finden eines Ansatzes	124
10.3.1	Partielle Ordnung von Methoden	124
10.3.2	Partiell injektive Ansätze	125
11	Transformationen	129
11.1	Grundlagen	129
11.2	Migration von Daten und Prozessen	131
11.3	Migration von Programmen	136
11.3.1	Migration konsistenter Aktionen	139
11.3.2	Migration von Ansätzen	141
11.3.3	Migration von Pushouts	142
11.3.3.1	Pullback-Funktor	142
11.3.3.2	Kompositions-Funktor	144
11.3.3.3	Epirefektor	144
11.3.3.4	Gesamte Migration	146
11.3.4	Zusammenfassung	147
11.4	Komposition	147
12	Umsetzung des konzeptionellen Modells	157
12.1	Datenbanken	157
12.1.1	Klassen und Objekte	157
12.1.2	Assoziationen und Verknüpfungen	157
12.2	Software	158
12.2.1	Prozesse	158
12.2.2	Programme	158
12.3	Transformationen	160
12.3.1	Klassen und Vererbung	162
12.3.1.1	Erzeugen einer eigenständigen Klasse	162
12.3.1.2	Erzeugen einer Oberklasse	163
12.3.1.3	Erzeugen einer Unterklasse	164
12.3.1.4	Verschmelzen zweier verwandter Klassen	165
12.3.1.5	Verschmelzen zweier nicht verwandter Klassen	166
12.3.1.6	Verschieben einer Oberklasse entlang einer Vererbungsbeziehung ..	167
12.3.1.7	Verschieben einer Unterklasse entlang einer Vererbungsbeziehung ..	168
12.3.1.8	Löschen einer Klasse	168
12.3.2	Assoziationen	169
12.3.2.1	Erzeugen einer Assoziation zu einer Klasse	170
12.3.2.2	Erzeugen einer Assoziation zu einer Operation	170
12.3.2.3	Verschieben des Anfangs einer Assoziation zu einer Oberklasse ..	171
12.3.2.4	Verschieben des Endes einer Assoziation zu einer Oberklasse	171

12.3.2.5 Löschen einer Assoziation	172
12.3.3 Attribute	173
12.3.3.1 Erzeugen eines Attributs	173
12.3.3.2 Verschieben eines Attributs zu einer Oberklasse	174
12.3.3.3 Löschen eines Attributs	174
12.3.4 Operationen	175
12.3.4.1 Erzeugen einer eigenständigen Operation	175
12.3.4.2 Erzeugen einer Oberoperation	176
12.3.4.3 Erzeugen einer Unteroperation	177
12.3.4.4 Verschmelzen zweier verwandter Operationen	177
12.3.4.5 Verschmelzen zweier nicht verwandter Operationen	178
12.3.4.6 Verschieben einer Oberoperation entlang einer Vererbungsbeziehung	179
12.3.4.7 Verschieben einer Unteroperation entlang einer Vererbungsbeziehung	180
12.3.4.8 Löschen einer Operation	180
12.3.5 Parameter	181
12.3.5.1 Erzeugen eines Parameters	181
12.3.5.2 Verschieben des Anfangs eines Parameters zu einer Oberoperation	182
12.3.5.3 Verschieben des Endes eines Parameters zu einer Oberoperation	183
12.3.5.4 Löschen eines Parameters	184
13 Fallstudie: Formalisierung	185
13.1 Ausgangssituation	185
13.2 Phase 1: Hinzunahme eines neuen Versicherungsprodukts	191
13.3 Phase 2: Hinzunahme ausländischer Kunden	197
13.4 Phase 3: Konzentration auf Geschäftskunden	203
13.5 Komposition von Transformationen	206

Teil IV Ausblick

14 Einschränkungen und Erweiterungen	211
14.1 Datenbank- und Software-Einschränkungen	212
14.1.1 Abstrakte Klassen	212
14.1.2 Kompositionen	214
14.1.3 Mindestens-eins-Beschränkungen	215
14.1.4 Vollständige Objekte	216
14.1.5 Zusammenhängende Objekte	217
14.1.6 Direkte vs. indirekte Vererbung	217
14.1.7 Typisierung von Daten in Software	218
14.1.8 Adhäsive HLR-Kategorien	218
14.2 Transformations-Einschränkungen	220
14.2.1 Symmetrie von Transformationen	220
14.2.2 Verallgemeinerung von Daten-Parametern	220
14.2.3 Mehrdeutigkeit nach Prozess-Transformation	222
14.2.4 Erhaltung der Semantik von Programmen	223
14.2.5 Daten-getriebene Schema-Transformationen	223
14.2.6 Formalisierung von Refactorings	224

15 **Fazit** 227

Teil V Anhang

A Mehrsortige algebraische Systeme 231

- A.1 Signaturen und Systeme 231
- A.2 Terme und Auswertung 233
- A.3 Spezielle Homomorphismen und Systeme 235
- A.4 Limiten 239
- A.5 Kolimiten 242
- A.6 Implikative Spezifikationen und Modelle 247
- A.7 Morphismen, Limiten und Kolimiten in implikativ spezifizierten Unterkategorien 250
- A.8 Horn-Formel-spezifizierte Unterkategorien und Konstruktion der Epireflexion 251
- A.9 Systeme und Algebren 253

B Algebraische Graph-Strukturen 255

- B.1 Graphstrukturen 255
 - B.1.1 Einfache Graphstrukturen 256
 - B.1.2 Attributierte Graphstrukturen 256
 - B.1.2.1 Grundlegendes 256
 - B.1.2.2 Konstruktion von Pushouts 257
 - B.1.2.3 Eigenschaften von Pushouts und Pullbacks 259
 - B.1.3 Typisierte Graphstrukturen 260
 - B.1.4 Typisierte attributierte Graphstrukturen 261
- B.2 Graphstruktur-Transformationen 261
 - B.2.1 Das grundlegende DPO-Modell 261
 - B.2.2 DPO-Transformationen von attributierten Graphstrukturen 263
 - B.2.3 DPO-Transformationen von typisierten Graphstrukturen 265

C Beweise 267

- C.1 Beweise aus Kapitel 9 267
 - C.1.1 Beweis von Lemma 9.12 267
 - C.1.2 Beweis von Lemma 9.13 271
 - C.1.3 Beweis von Lemma 9.14 274
 - C.1.4 Beweis von Satz 9.17 285
 - C.1.5 Beweis von Lemma 9.21 296
- C.2 Beweise aus Kapitel 10 297
 - C.2.1 Beweis von Lemma 10.24 297
 - C.2.2 Beweis von Lemma 10.25 309
 - C.2.3 Beweis von Lemma 10.26 317
 - C.2.4 Beweis von Lemma 10.28 319
 - C.2.5 Beweis von Satz 10.30 321
 - C.2.6 Beweis von Satz 10.31 327
 - C.2.7 Beweis von Satz 10.32 329
- C.3 Beweise aus Kapitel 11 337
 - C.3.1 Beweis von Lemma 11.9 337
 - C.3.2 Beweis von Lemma 11.17 340
 - C.3.3 Beweis von Lemma 11.18 342

VIII Inhaltsverzeichnis

C.3.4 Beweis von Lemma 11.20	345
C.3.5 Beweis von Lemma 11.21	357
C.3.6 Beweis von Lemma 11.28	358
C.3.7 Beweis von Lemma 11.29	359
C.3.8 Beweis von Lemma 11.32	361
C.3.9 Beweis von Satz 11.33	364
C.3.10 Beweis von Lemma 11.42	380
C.3.11 Beweis von Lemma 11.44	381
C.3.12 Beweis von Lemma 11.47	382
C.3.13 Beweis von Lemma 11.48	385
Literaturverzeichnis	393

Problemstellung

Einleitung

“Das einzig Konstante ist der Wandel.” Nirgendwo anders trifft diese Erkenntnis mehr zu als in der Informatik. Hardware und Software entwickeln sich unaufhörlich weiter, werden leistungsfähiger, sicherer und benutzerfreundlicher. Dabei muss insbesondere die Software – genauer: die in der Software programmierten *Abläufe* – ständig an die sich verändernde Realität angepasst werden. Jede Software bildet aber nicht nur Abläufe ab, sondern arbeitet vor allem mit und auf *Daten*. Diese Daten stellen in Computer-unterstützten Anwendungsfällen den wesentlichen Teil dar, den Kern also, um den sich alle Prozesse im Großen (Geschäftsprozesse) und im Kleinen (einzelne Funktionen einer Anwendung) drehen. Insbesondere sind Daten – verglichen mit Hard- und Software – in der Regel einzigartig und stellen das eigentliche Kapital eines Unternehmens dar. Man denke beispielsweise an Kunden- und Vertragsdaten, an Testergebnisse aus Forschung und Entwicklung oder an das Know-How, das in die Herstellungsprozesse von Produkten einfließt.

Die Binsenweisheit “Wissen ist Macht” wird also zu “Daten sind Macht”, vorausgesetzt es existieren geeignete Werkzeuge zum Extrahieren der gewünschten Informationen aus den Daten. Hierzu ist aber eine gewisse *Struktur* der Daten erforderlich. Die Art und Weise, wie Daten strukturiert sind – auch *Datenschema* genannt – beeinflusst nachhaltig die Effizienz der auf den Daten stattfindenden Aktionen (wie Suchen, Ersetzen, Löschen, Aggregieren, Sortieren usw.) und unterscheidet sich deshalb typischerweise für jede einzelne Anwendung.

Datenbestände sind noch viel häufiger *Veränderungen* unterworfen als Soft- oder gar Hardware.¹ Vorgänge wie die Aktualisierung von Lagerbeständen, das Einpflegen von Adressänderungen, das Anfertigen von Berichten und Statistiken – nur um einige Beispiele zu nennen – erfordern eine Auseinandersetzung mit alten existierenden und neuen hinzukommenden Daten in der verarbeitenden Anwendung. Die zugehörigen Abläufe bleiben aber die gleichen, so dass hierfür keine Anpassung der Software notwendig ist. Folglich schlagen sich Veränderungen der Realität als erstes in den Datenbeständen nieder.

¹ “Experience has shown us that instances, because they represent changing objects in the real world, tend to evolve much faster than the concepts themselves.” [82, Abschnitt 5.2]

Gelegentlich erfordert das Hinzufügen, Verändern oder Löschen von Daten *Veränderungen am Datenschema selbst*. Der häufigste Fall ist die Erweiterung des Datenbestandes um eine zusätzliche Information, die bis dato nicht vorgesehen war und deshalb vom Datenschema nicht unterstützt wird (etwa die Hinzunahme der E-Mail-Adresse zu den Kundendaten). In einer solchen Situation muss im ersten Schritt das Datenschema geeignet angepasst werden, bevor im zweiten Schritt die neuen Daten aufgenommen werden können. Dabei ergeben sich verschiedene Fragestellungen:

- (1) Ist das Datenschema flexibel genug, um die anstehende Veränderung der Daten abbilden zu können? Wenn nicht, lässt es sich geeignet anpassen, *refaktorisieren*?
- (2) Lässt sich das Refactoring des Datenschemas ohne Datenverluste bewerkstelligen? Führt die Veränderung des Datenschemas automatisch zu einer *Migration der alten Daten* in das neue Schema, oder beinhaltet die Migration manuelle Teilprozesse?
- (3) Führt die Änderung des Datenschemas automatisch zu einer *Migration der auf den Daten operierenden Software*?

Die folgenden Abschnitte gehen auf diese Fragen ein und erläutern, inwiefern die vorliegende Arbeit bei der Lösung der angerissenen Probleme weiterhilft.

1.1 Transformation von Datenschemata

Die Flexibilität eines Datenschemas lässt sich daran messen, wie leicht oder schwer neue Anforderungen an in dem Schema typisierten Daten ohne eine Schema-Änderung umgesetzt werden können. Praktisch wird die Flexibilität eines Datenschemas durch seinen Abstraktionsgrad bestimmt: Je abstrakter ein Schema ist, desto stärker kann es sich neuen Anforderungen anpassen. Durch zuviel Abstraktion kann jedoch die Aussagekraft eines Schemas stark nachlassen. In Abb. 1.1 sind zwei Schemata einer recht einfachen Adressverwaltung dargestellt.

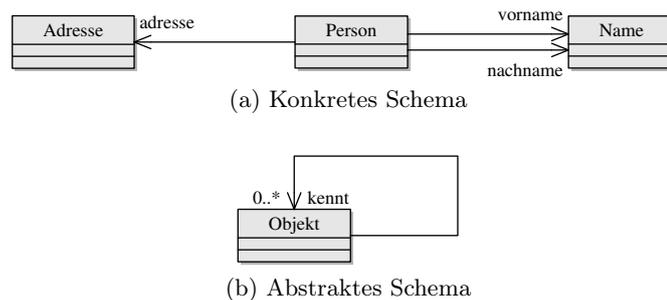


Abb. 1.1: Schemata mit unterschiedlichem Abstraktionsgrad

Beide Schemata können dafür dienen, die Struktur von Adressdaten abzubilden. Während diese Struktur in Abb. 1.1a jedoch gut zu erkennen ist, ist sie in Abb. 1.1b auf das Minimum reduziert

worden. Das sehr abstrakte Schema in Abb. 1.1b hat neben der geringen Ausdruckskraft auch den Nachteil, dass inkorrekte Daten darin typisiert sein können, etwa Personen ohne Namen. Das wird durch das konkretere Schema in Abb. 1.1a verhindert. Allerdings muss dieses Schema angepasst werden, wenn weitere Anforderungen hinzukommen, etwa der Wunsch, einer Person mehrere Adressen zuzuordnen; das abstrakte Schema muss hingegen nicht geändert werden.

Diese Kluft zwischen flexibler, aber wenig aussagekräftiger Abstraktion und bedeutungsvoller, aber einengender Konkretheit wird in der Objektorientierung mit Hilfe von Spezialisierung und Generalisierung überbrückt. Durch Spezialisierung kann “nah an den Daten” modelliert werden, um wichtige Informationen und Konsistenz-Bedingungen abzubilden und nicht zu verlieren. Durch Generalisierung können Abstraktionen geschaffen werden, um zukünftige Anpassungen des Schemas einfacher gestalten zu können. Da jedoch selten von Anfang an die geeigneten Abstraktionen gewählt werden, und auch im Laufe der Zeit neue Abstraktionen an Bedeutung gewinnen oder bestehende an Relevanz verlieren, ist es gerade in der Anpassung von Schemata wichtig, sich zwischen verschiedenen Abstraktionsgraden bewegen zu können – im extremen Fall also von dem Schema in Abb. 1.1a zu dem Schema in Abb. 1.1b zu gelangen oder umgekehrt.

Die vorliegende Arbeit analysiert in Kapitel 3 anhand eines Beispiels typische Schema-Transformationen, die bei der Entwicklung objektorientierter Systeme auftreten. Teil II präsentiert auf dieser Grundlage ein konzeptionelles Modell von Transformationen, Refactorings und objektorientierten Systemen. Dieses Modell wird schließlich in Teil III auf eine formale Grundlage gestellt.

1.2 Migration der Daten

Ist die Entscheidung gefallen, ein Datenschema zu verändern, so stellen sich zwei Fragen:

- (1) *Wie* werden die Daten zum alten Schema in das neue Schema übernommen, *migriert*? Dazu existieren im Großen und Ganzen die folgenden Ansätze:
 - (a) *Manuelle Migration*. Die Daten werden manuell migriert, d. h. durch den Einsatz von menschlicher Arbeitskraft im Rahmen eines “Ablese, umwandeln, eintippen”-Prozesses transformiert.
 - (b) *Halbautomatische Migration*. Anhand von Informationen über die durchgeführte Schema-Veränderung werden mit Hilfe von Datenbank-Werkzeugen Skripte generiert, welche die Daten aus dem alten Datenbank-Schema in das neue Datenbank-Schema kopieren und dabei geeignet anpassen.
 - (c) *Vollautomatische Migration*. Die Daten werden vom System automatisch migriert, ohne dass es für den Benutzer sichtbar ist oder administrative Tätigkeiten erforderlich sind.
- (2) *Wann* wird die Migration durchgeführt? Hier existieren prinzipiell zwei Möglichkeiten:

- (a) *Alles auf einmal*. Alle Daten werden “in einem Rutsch” migriert, und zwar vor der ersten Benutzung unter dem neuen Schema.
- (b) *Bei Bedarf (On demand)*. Es werden nur die Daten migriert, die bei einem Zugriff benötigt werden.

Die manuelle Migration ist nur bei kleineren Datenbeständen praktikabel. Der Vorteil dieses Ansatzes ist die hohe Flexibilität bei der Datentransformation, weil auch unsystematische Transformationen nach entsprechender Einweisung der Mitarbeiter möglich sind. Insbesondere können Menschen bei auftretenden Fehlern oder Inkonsistenzen im zu migrierenden Datenbestand eigenverantwortlich individuell entscheiden, welche Korrektur angemessen ist. Dafür ist das Fehlerpotential deutlich höher als bei den anderen Ansätzen, da Menschen etwa durch Unterbrechungen bei der Arbeit leicht Fehler unterlaufen können. On-Demand-Migration existiert wenn überhaupt nur sehr selten, da die Motivation bei den Mitarbeitern, die in einem System benötigten und erfolgreich abgefragten Daten vor der Weiterverarbeitung erst in ein anderes System einzupflegen, verständlicherweise sehr gering ausfällt.

Die halbautomatische Migration ist der Ansatz, der in der Praxis heutzutage am häufigsten gewählt wird. Dabei werden die Daten auf dem Abstraktionsgrad der verwendeten Datenbank (z. B. Tabellen bei relationalen oder Objekte bei objektorientierten Datenbanken) restrukturiert. Über entsprechende Skripte, die entweder manuell oder mit Unterstützung der Datenbank-Werkzeuge erstellt werden, können die Daten migriert werden. Bei komplexeren Migrationen ist es jedoch häufig notwendig, die migrierten Daten manuell nachzubearbeiten. Auch müssen die Migrations-Skripte mit Hilfe von Testdaten ausgetestet werden. Die Generierung entsprechend realer Testdaten ist wiederum keine triviale Angelegenheit. Insofern ist auch hier an mehreren Stellen ein gewisses Fehlerpotential vorhanden. On-Demand-Migration ist in begrenztem Umfang möglich, wenn die Datenbank in der Lage ist, Lese- und Schreibzugriffe auf die Daten abzufangen und an ein Migrations-Programm weiterzuleiten. Interessanterweise hat sich die On-Demand-Migration in der Praxis aus mehreren Gründen kaum durchgesetzt:

- befürchtete Komplexität durch Koexistenz mehrerer Versionen desselben Schemas
- befürchtete schleichende Inkonsistenz der Daten durch fehlerhafte Migrations-Skripte
- befürchtete Performanz-Einbußen beim Zugriff auf die Daten

Schließlich sollte bedacht werden, dass die Migrations-Skripte nach durchgeführter Migration nicht mehr benötigt werden. Der vergleichsweise hohe Aufwand zum Erstellen und Anpassen der Skripte, verbunden mit der kurzen Einsatzzeit, ergibt, dass das Verfolgen eines solchen Ansatzes nicht besonders wirtschaftlich ist.

Die vollautomatische Migration löst alle Migrations-Probleme, indem jede Veränderung am Schema direkt zu einer Migration der entsprechenden Daten führt, wobei diese Migration von der Datenbank automatisch und systematisch aus der Schema-Änderung abgeleitet wird. Unter

der Voraussetzung, dass die Ableitung der Migration aus der Schema-Veränderung *korrekt* ist, kann es – bei korrekter Implementierung – nicht zu Konsistenzproblemen kommen. On-Demand-Migration ist in demselben Maße möglich wie bei halbautomatischer Migration. Durch die garantierte Korrektheit fällt aber der Nachteil möglicher Konsistenzprobleme durch fehlerhafte Migrationsskripte weg. Die Nachteile der vollautomatischen Migration sind zum einen, dass sie auf Grund fehlender manueller Eingriffe nicht so flexibel wie die anderen Formen der Migration sein kann. Zum anderen stehen Schema-Änderung und Migration in einer viel engeren Beziehung, was dazu führen kann, dass eine – auf den ersten Blick einfache – Schema-Änderung nicht durchgeführt werden kann, weil die zugrunde liegende Systematik daraus keine vernünftige Migration ableiten kann.

Es gilt also den Graben zwischen hoher Flexibilität und deterministischer Korrektheit zu überbrücken. Die vorliegende Arbeit setzt an diesem Punkt an und stellt in Teil III eine Konstruktion vor, die es ermöglicht, bestimmte Transformationen auf der Schema-Ebene kanonisch in korrekte Migrationen auf der Daten-Ebene umzusetzen. Dabei werden formal alle Daten auf einmal migriert, allerdings ist es möglich, diese Migration auch bei laufenden Prozessen durchzuführen (siehe hierzu den nächsten Abschnitt). On-Demand-Migration von Daten wird im Rahmen dieser Arbeit nicht weiter diskutiert, hier liegt aktuell Forschungsbedarf vor.

1.3 Migration der Software

Änderungen am Datenschema passieren in der Regel nicht im luftleeren Raum. Fast immer steht dahinter der Wunsch nach einer verbesserten oder erweiterten Dienstleistung. Beispielsweise kann die Aufnahme der E-Mail-Adresse eines Kunden in Hinblick darauf geschehen, dem Kunden zukünftig schneller wichtige Informationen zukommen zu lassen, Angebote zuzustellen oder andere Dienstleistungen anzubieten als mit Hilfe herkömmlicher Briefpost. Das Datenschema und die dazugehörigen Daten sind folglich mit Prozessen verknüpft, die diese Daten auslesen, verändern oder anderweitig verarbeiten.

Teile dieser Prozesse sind häufig automatisierbar und somit ideale Kandidaten für Software. Diese Software greift natürlich auf irgendeine Weise auf den verfügbaren Datenbestand zu und arbeitet damit. Somit wird die Software abhängig vom verwendeten Datenschema. Wenn sich dieses Datenschema jedoch ändert – etwa durch eine gewünschte Transformation –, dann ist mit einer erfolgreich durchgeführten Migration der Daten nur der halbe Weg zum gewünschten Ziel zurückgelegt worden. Denn die verfügbare Software muss ebenfalls an das neue Datenschema angepasst werden. Hier ergeben sich ähnliche Fragestellungen wie bei der Datenmigration:

- (1) *Wie* kann die alte Software *migriert* werden, um Daten zum neuen Schema verarbeiten zu können?
- (2) *Wann* erfolgt die Migration?

Der erste Punkt erfordert, dass die Software dem System in einer Repräsentation vorliegt, die – genauso wie die Daten bei einer Migration – in ihrer Struktur manipuliert werden kann. Dafür sind Programme, die im Quelltext vorliegen, nur bedingt geeignet. Es ergibt sich also automatisch die Fragestellung nach einer geeigneten – und vor allem verlustlosen! – Umwandlung des Quelltextes in eine die Semantik betonende Repräsentation der Software und wieder zurück. Moderne Software-Entwicklungs-Werkzeuge wie *eclipse* bieten schon heute für verbreitete Programmiersprachen wie *Java* Mittel zum Refactoring an [87], die intern auf einem geeigneten semantischen Modell der Programmiersprache aufsetzen. Die Herausforderung liegt hier vor allem darin, ein solches Modell mit dem Datenschema zu einem einheitlichen Modell zu verknüpfen, um Refactorings synchron durchführen zu können.

Beim zweiten Punkt liegt aktuell Forschungsbedarf vor, weil die Frage nach dem “Wann?” der Migration der verarbeitenden Software heutzutage in der Regel mit “nur wenn keine laufenden Prozesse existieren” beantwortet wird.² Die besprochene On-Demand-Migration von Daten lässt sich nur ungleich schwerer auf die Migration von Software oder Software-Komponenten übertragen. Neben den technischen Herausforderungen³ stellt sich die Frage nach dem Korrektheitsbegriff, insbesondere bei der Migration laufender Prozesse (Programm-Ausprägungen). Denn ein Prozess, der migriert wird, gehört zu verschiedenen Zeitpunkten verschiedenen Prozess-Modellen an. Durch diese hinzukommende zeitliche Komponente entstehen Prozesse, die weder vollständig zum alten noch vollständig zum neuen Prozess-Modell gehören. Im Rahmen dieser Arbeit wird Korrektheit bei der Migration von Software auf den Nachweis reduziert, dass jeder mögliche Programmablauf vor einer Migration auch nach der Migration durchführbar ist.⁴

Teil III geht auf die erste Fragestellung ein und zeigt, dass Transformation unter bestimmten Bedingungen in der Lage sind, Programme und Prozesse so an Schema-Änderungen anzupassen, dass gewisse Aspekte der Software-Semantik erhalten bleiben. Auf On-Demand-Migration von Programmen und Prozessen wird im Rahmen dieser Arbeit nicht weiter eingegangen, vielmehr werden Programme und Prozesse komplett und “in einem Rutsch” migriert.

1.4 Aufbau der Arbeit

Teil I führt in das Problem ein. In Kapitel 2 werden existierende themenbezogene Arbeiten vorgestellt und ihre Gemeinsamkeiten und Unterschiede zum hier vorgestellten Ansatz diskutiert.

² Im begrenzten Rahmen erlauben Debugging-Werkzeuge moderner Entwicklungsumgebungen das Verändern von übersetzten Programmen während der Laufzeit (“Edit-and-Continue“-Technik), etwa Microsoft Visual C++ [79].

³ Eine solche On-Demand-Migration benötigt entweder eine flexible Laufzeit-Umgebung, die in der Lage ist, Software-Komponenten oder Teile hiervon dynamisch zur Laufzeit zu verändern oder auszutauschen, oder die Manipulation muss für das Programm vollkommen transparent geschehen, indem zur Laufzeit der Prozess angehalten und die Programm-Repräsentation direkt geändert wird.

⁴ vgl. [22]; siehe hierzu auch Anmerkung 11.36 sowie die Abschnitte 14.2.3 und 14.2.6

In Kapitel 3 werden die Anforderungen an Schema-Transformationen und Migrationen von Datenbeständen, Programmen und Prozessen anhand einer exemplarischen Fallstudie analysiert.

In Teil II werden die grundlegenden Konzepte, auf denen diese Arbeit aufbaut, vorgestellt. In Kapitel 4 werden die zentralen Konzepte Software, Datenbank, System und Transformation eingeführt. Kapitel 5 beschreibt ein informelles Modell zur Modellierung objektorientierter Daten, Programme und Prozesse, das die Grundlage für die Formalisierung in Teil III bildet. Dieses Modell wird in [76] im Detail behandelt. Schließlich wird in Kapitel 6 beschrieben, wie die Elemente der Fallstudie im konzeptionellen Modell dargestellt werden.

Teil III ist der Hauptteil der Arbeit und formalisiert in einem algebraisch-kategoriellen Rahmen das konzeptionelle Modell aus Teil II. Nach einem kurzen Überblick in Kapitel 7 wird in Kapitel 8 eine ganzheitliche Spezifikation M erarbeitet, die der mathematischen Modellierung von Datenbanken, Programmen und Prozessen dient. Diese Spezifikation basiert auf mehrsortigen algebraischen Systemen und spezifiziert die gewünschten Eigenschaften durch positive Horn-Formeln, die über Operationssymbolen und Prädikaten formuliert sind. Die dafür benötigte Theorie kombiniert Ergebnisse der (einsortigen) universellen Algebra mit der Erweiterung um mehrere Sorten [21, 25] und um Prädikate [53] und wird in [77] behandelt.

Kapitel 9 legt die Grundlagen für die weitere kategorielle Behandlung von Programmen, Prozessen und Transformationen, indem die Eigenschaften der Kategorie $\mathbf{Alg}(M)$ aller Systeme und zugehöriger Homomorphismen untersucht werden und diese Kategorie mit anderen Kategorien in Beziehung gesetzt wird. Hier werden auch die für die späteren Resultate wichtigen Epireflexionen untersucht, mit denen nicht zur Spezifikation M konforme Systeme frei in M -Systeme transformiert werden. Das wichtigste Ergebnis dieses Kapitels sind die Sätze 9.15 und 9.17, die besagen, dass die Konstruktion des angepassten Systems sich auf einige wenige Axiome konzentrieren kann und somit die Konstruktion insgesamt einfach zu überschauen ist.

Kapitel 10 formalisiert Programme als eine Menge von Transformations-Regeln und beschreibt, wie durch die sukzessive Anwendung dieser Regeln laufende Prozesse abgebildet werden können. Dabei beschreibt jede einzelne Transformations-Regel einen einzigen Abarbeitungsschritt im Programmablauf, der semantisch dem Aufruf einer Methode entspricht. Mathematisch wird jede dieser Transformations-Regeln über eine DPO-Regel⁵ mit bestimmten Eigenschaften formuliert.⁶ Das wichtigste Ergebnis des Kapitels ist Theorem 10.33, das besagt, dass die Ausführung von Methoden alle M -Axiome erhält.

Kapitel 11 befasst sich mit der formalen Definition von Transformationen auf der Schema- und auf der Ausprägungsebene. Über die Definition geeigneter Funktoren wird dargestellt, wie Schema-Transformationen automatisch zu eindeutigen Migrationen der zugehörigen Daten, Programme und Prozesse fortgesetzt werden können, so dass alle gewünschten Eigenschaften der Datenbank

⁵ “DPO” steht für “Doppel-Pushout” und beschreibt einen weit verbreiteten Ansatz, um Graphstruktur-Transformationen in einem algebraischen Rahmen zu formalisieren.

⁶ Die Theorie algebraischer Graphstruktur-Transformationen basierend auf dem DPO-Ansatz wird in [20] behandelt, die elementaren Konstruktionen und alle nötigen Beweise sind in [75] nachzulesen.

und der Software erhalten bleiben. Schließlich wird die Komposition von Schema-Transformationen und induzierten Ausprägungs-Migrationen untersucht. Die wichtigsten Ergebnisse des Kapitels sind zum einen Theorem 11.12 und Theorem 11.35, die besagen, dass die aus einer Schema-Transformation induzierte Migration der Ausprägung alle M -Axiome sowie die besondere Struktur von Methoden erhält. Zum anderen sagt Theorem 11.50 aus, dass die von einer komponierten Schema-Transformation induzierte Migration denselben Effekt hat wie die Komposition der zwei einzelnen induzierten Migrationen, falls bestimmte Bedingungen erfüllt sind.

Kapitel 12 beschreibt Details in der Umsetzung konzeptioneller Strukturen in das mathematische Modell sowie Randbedingungen, die das mathematische Modell zusätzlich zu den formalisierten Axiomen erfüllen muss, damit objektorientierte Modelle ordentlich umgesetzt werden. Weiter enthält dieses Kapitel einen Katalog von Schema-Transformationen, die als formale Grundlage für komplexe Schema-Veränderungen dienen. Schließlich behandelt Kapitel 13, wie die Elemente der Fallstudie im mathematischen Modell dargestellt werden und wie sich die beschriebenen Schema-Transformationen darauf auswirken.

Teil IV fasst in den Kapiteln 14 und 15 die Ergebnisse zusammen und gibt einen Ausblick auf mögliche weitere Forschungsgebiete. Teil V bildet den Anhang der Arbeit. Er enthält in den Kapiteln A und B die in dieser Arbeit benötigten Definitionen und Sätze aus den Bereichen “mehrsortige algebraische Systeme” und “algebraische Graphstrukturen”. In Kapitel C sind schließlich alle längeren Beweise zusammengestellt.

Themenbezogene Arbeiten

Die Idee, Daten-Schemata zu verändern, ohne dass bestehende Ausprägungen ungültig werden, ist nicht neu. Im Kontext von Datenbanken werden solche Schema-Transformationen unter dem Begriff *Schema-Evolution* zusammengefasst und sind seit den achtziger Jahren Gegenstand der Forschung (siehe die Zusammenfassung in [69] und [70]). Der in dieser Arbeit vorgestellte Ansatz hat jedoch Eigenschaften, die sich von vorhandenen Arbeiten unterscheiden:

- ein mathematisches Modell auf Grundlage der Kategorientheorie und universellen Algebra, das Daten, Programme, Prozesse und Transformationen einheitlich beschreibt,
- die automatische Ableitung korrekter und eindeutiger Ausprägungs-Migrationen aus komplexen Schema-Transformationen,
- eine inkrementelle Vorgehensweise durch Zusammenbau mehrerer kleinerer Schema-Transformationen zu einer großen Transformation und
- die kombinierte Migration von Daten, Programmen und Prozessen.

Diese Eigenschaften werden in den folgenden Abschnitten genauer unter die Lupe genommen und mit existierenden Arbeiten verglichen.

2.1 Mathematisches Modell für objektorientierte Daten

Das in Teil III entwickelte Modell nutzt mehrwertige algebraische Systeme zur Modellierung der Elemente objektorientierter Datenbanken. Eigenschaften werden mit Hilfe von positiven Horn-Formeln als Axiome formuliert. Dies hat zum einen den Vorteil, dass Eigenschaften über Faktorisierungsprozesse auf minimale und eindeutige Weise erzwungen werden können. Zum anderen existiert eine wohlfundierte Theorie, deren Ergebnisse für die Erhaltung von Eigenschaften genutzt werden kann.⁷ Eine weitere Besonderheit des Modells ist die Darstellung von

⁷ Beispielsweise erfüllen sowohl ein Pullback-System als auch ein Pushout-Komplement-System alle Axiome einer Spezifikation *Spec*, wenn die Ausgangssysteme ebenfalls Modelle dieser Spezifikation sind. Dies wird unter anderem in Satz 10.19 und Lemma 11.9 ausgenutzt.

Objekten durch Komponenten, die aus einzelnen, über Vererbungsverknüpfungen miteinander verbundenen Partikeln bestehen. Dies bringt auf Grund der strukturellen Gleichheit von Schema und Ausprägung viele Vorteile mit sich. Unter anderem wird dadurch ermöglicht, Ausprägungen in Schemata über gewöhnliche Homomorphismen zu typisieren.

In [20, Kap. 13] wird ein Datenmodell mit Vererbung auf der Grundlage typisierter attributierter Graphstrukturen entwickelt. Vererbungskanten werden dabei über einen zweiten, vom eigentlichen Schema unabhängigen Graph definiert, wobei der Schema-Graph und der Vererbungsgraph dieselben Knoten enthalten. Weitere Eigenschaften der Vererbungsstruktur wie Antisymmetrie werden jedoch nicht gefordert. Das Modell in [20] unterscheidet sich auch in der Typisierung von Objekten und Verknüpfungen in einem Schema. Da auf der Ausprägungs-Ebene attributierte Graphstrukturen *ohne* Vererbung zur Modellierung von Objekten und Verknüpfungen genutzt werden, müssen spezielle Morphismen, nämlich *attribuierte Clan-Morphismen*, definiert werden, damit die Typisierung “bis auf Vererbung” formuliert werden kann. Schließlich werden zum Zweck der Anwendung weiterer graphentheoretischer Konstruktionen und Sätze die Vererbungsgraphen wieder “flachgeklopft”, so dass die Vererbungsstruktur nicht über ein permanentes Modellierungskonzept repräsentiert wird. Das in dieser Arbeit entwickelte Modell hingegen nutzt zum einen zwei Prädikate *underC* und *underO* zur expliziten Modellierung der Vererbungsstruktur über binäre Relationen, wobei diese Relationen auch unter Schema- und Ausprägungs-Transformationen nicht verloren gehen. Zum anderen wird über diese Prädikate die Vererbungsstruktur sowohl zwischen Klassen als auch innerhalb von Objekten modelliert, so dass das Modell mit gewöhnlichen Typisierungs-Homomorphismen auskommt. Schließlich erlaubt die Modellierung als Prädikat auf einfache Weise, Vererbung zyklensfrei als partielle Ordnung zu spezifizieren.

In [9, 10] wird das Datenmodell aus [20] um Kompositionen (*containment edges*) erweitert. Dabei werden die Kompositionskanten als Teilmenge aller Assoziationskanten definiert, was in mehrwertigen Systemen äquivalent zu der Definition eines einstelligen Prädikats auf der Sorte der Assoziationen ist. Weiter wird nachgewiesen, dass gewisse Eigenschaften der Kompositionen – Kompositionen sind azyklisch, es existiert höchstens ein Behälter für jedes Objekt – bei Veränderung durch Graphstruktur-Transformationen erhalten bleiben. Im Gegensatz zum Ansatz dieser Arbeit werden diese Eigenschaften jedoch nie verletzt, somit wird nicht diskutiert, wie Graphen an diese Eigenschaften angepasst werden können. Dies ist jedoch ein wichtiger Teil dieser Arbeit (Abschnitt 9.2), weil durch Schema-Transformationen gewisse Eigenschaften des mathematischen Modells aus Teil III verloren gehen. Schließlich werden an die Exemplar-Ebene und an das Schema unterschiedliche Forderungen gestellt: Kompositionskanten können im Typ-Graphen zyklisch sein, während dies in der Ausprägung verboten ist. Dies ist für die Modellierung der Komposition sinnvoll und notwendig. In dieser Arbeit ist es aber an vielen Stellen wichtig, Schema- und Ausprägungs-Systeme nicht voneinander unterscheiden zu müssen, um innerhalb einer Kategorie zu bleiben und somit kategorielle Konstruktionen einfacher durchführen zu können. Der Ansatz, Eigenschaften nicht zu erzwingen, sondern Konstruktionen so einzuschränken, dass sie die Eigenschaften erhalten, ist in dieser Arbeit in Abschnitt 10.2 aufgegriffen worden, in dem

gezeigt wird, dass Programmabläufe durch die Anwendung von Graphstruktur-Transformationen die Axiome der Spezifikation objektorientierter Systeme erhält.

In [55] wird ein Graph-Modell für Klassenstrukturen und Software eingeführt, um Refactorings durch Graph-Transformationen zu modellieren. Dieses Modell beschreibt jedoch nur die für die ausgewählten Refactorings nötigen Elemente eines objektorientierten Systems. Insbesondere fehlt eine Modellierung des *Programmcodes*,⁸ so dass dieses Modell nicht zur Simulation von Programmen durch entsprechende Prozesse herangezogen werden kann.

2.2 Mathematisches Modell für objektorientierte Software

Die Idee, Programme als Graphen zu kodieren und den Ablauf von Programmen durch Graph-Transformationen zu beschreiben, ist nicht neu. Im Folgenden wird das in Teil III ausgearbeitete Modell mit anderen Modellen verglichen.

2.2.1 Das Java-Hyperkanten-Modell

In [18] wird ein Modell beschrieben, welches in der Lage ist, eine Untermenge der Programmiersprache *Java* abzubilden, so dass einfache *Java*-Programme kodiert und mit Hilfe von Graph-Transformationen ausgeführt werden können. Dabei werden Nachrichten als spezielle Hyperkanten kodiert. Folgende Gemeinsamkeiten zu dem in dieser Arbeit erarbeiteten Modell sind zu verzeichnen:

- Ausdrücke, Anweisungen und Methoden (inklusive Konstruktoren) werden durch Graph-Transformationen dargestellt (vgl. 5.2.1.5).
- Es gibt – ähnlich dem *Processor*-Objekt – ein *GO*-Element, welche die nächste zu verarbeitende Nachricht markiert und während der Ausführung des Programms weitergereicht wird (vgl. 5.2.1.4).
- Werte zu primitiven Datentypen werden über eine Indirektion angebunden (vgl. 5.2.1.1).
- Es werden die folgenden Operatoren und Anweisungen unterstützt: Arithmetische und relationale Operatoren, lesender und schreibender Zugriff auf lokale Variablen und Objekte, Erzeugung von Objekten, Fallunterscheidung und Rücksprung (vgl. 5.2.2).
- Der Zugriff auf Verknüpfungen eines Objekts wird generell über Lese- und Schreib-Methoden realisiert (vgl. 5.2.2).

Es gibt jedoch auch wesentliche Unterschiede in der Ausdruckskraft:

⁸ Dafür werden so genannte “B”-Knoten verwendet (“B” für “Body”), die von dem Rumpf einer Methode abstrahieren.

- Bedingt durch die Löschung von Nachrichten bei deren Verarbeitung verändert die Ausführung von Programmen den Graph auf *destruktive* Weise. Dies führt dazu, dass das Modell aus [18] keine Schleifenkonstrukte direkt unterstützt; stattdessen müssen Schleifen durch rekursive Aufrufe simuliert werden. Im Gegensatz dazu beeinflusst die Abarbeitung von Programmen im vorliegenden Modell nicht die operationale Struktur der Prozesse, so dass Nachrichten problemlos mehrfach ausgewertet werden können und Schleifen somit ohne zusätzlichen Aufwand unterstützt werden.
- Vererbung und abhängige Sprachkonstrukte⁹ werden nicht unterstützt.¹⁰

2.2.2 Das TAAL-Modell

Das in [38] und [39] vorgestellte Modell zur Abbildung der Semantik der objektorientierten Programmiersprache TAAL ist ein fortgeschrittenes Modell, das viele wichtige Aspekte von Programmiersprachen abdeckt. Programme werden dabei durch zwei Graphstrukturen, den *abstrakten Syntax-Graphen* und den *Kontrollfluss-Graphen* dargestellt. Dabei enthält der Syntax-Graph die Struktur des Programms, zu der die Definition der Typen, Operationen und Methoden gehören, wobei die Definition der Methoden die Anweisungen und Ausdrücke umfasst, die in ihnen vorkommen. Der Kontrollfluss-Graph enthält die Information über die auszuführenden Anweisungen und auszuwertenden Ausdrücke sowie über deren Reihenfolge und ist in gewisser Hinsicht redundant, weil er aus den Methoden-Definitionen des abstrakten Syntax-Graphen gewonnen wird.¹¹ Prozesse werden durch zwei weitere Graphen beschrieben, den *Daten-Graphen*, der die Objekte enthält, mit denen der Prozess arbeitet, und den *Rahmen-Graphen*, der die Aufrufe und lokale Variablen während der Ausführung des Prozesses verwaltet.

Trotz der Allgemeinheit der verwendeten Konzepte finden sich gerade in Hinblick auf die Zielsetzung der vorliegenden Arbeit jedoch einige Nachteile:

- Prozesse und Programme sind nicht in einem Schema typisiert. Dies erschwert ein automatisches “Nachziehen” von Programmen und Prozessen bei Änderungen des Schemas, gleichgültig ob die Änderung im Datenbank- oder im Programm-Schema stattfindet.
- Das TAAL-Modell stellt Objekte in einer “flachen” Struktur dar. Somit wird die Vererbungshierarchie auf der Ausprägungs-Ebene nicht durch Partikel reflektiert. Dies verhindert jedoch eine ordentliche Typisierung von Objekten in der Typ-Hierarchie.

⁹ etwa statisch gebundene Aufrufe geerbter Methoden oder Typanpassungen

¹⁰ Die Autoren beschreiben in dem Papier, wie eine mögliche Unterstützung von Vererbung aussehen könnte, ohne jedoch konkrete Beispiele zu formulieren.

¹¹ “The flow graph is implicit in the abstract syntax graph; it can be derived from the abstract syntax graph by appropriately interpreting the various kinds of statements and expressions.” [39, Abschnitt 6.2]

- Es ist im *TAAL*-Modell nicht möglich, mit Werten zu primitiven Datentypen umzugehen.¹² Dies ist jedoch wesentlich, um die Semantik von Programmen komplett durch Graphstruktur-Transformationen ausdrücken zu können. Besonders schwer wiegt dabei die Unmöglichkeit, Fallunterscheidungen korrekt zu simulieren, weil die Wahrheitswerte *true* und *false* nicht verwendet werden können, um zwischen dem *then*- und dem *else*-Zweig unterscheiden zu können.¹³

2.3 Mathematisches Modell für Transformationen

Das in Teil III ausgearbeitete Modell für Schema- und Ausprägungs-Transformationen kombiniert die Ausdruckskraft der universellen Algebra mit der Flexibilität der Kategorientheorie. Insbesondere die Eindeutigkeit und die Komponierbarkeit der Konstruktionen profitieren erheblich von der kategoriellen Darstellung. Schließlich erlaubt die kategorielle Formulierung, aus Schema-Transformationen korrespondierende Migrationen auf der Ausprägungsebene abzuleiten.

Das in dieser Arbeit vorgestellte Modell baut auf den Modellen in den Vorarbeiten [51] und [52] insofern auf, als dass Schema-Transformationen über Spans von Morphismen formuliert werden. In [51] wird noch gefordert, dass Komponenten (entsprechen Klassenhierarchien im Kontext dieser Arbeit) vollständig ausgeprägt werden, damit die Existenz und Eindeutigkeit von Doppel-Pullback-Diagrammen gesichert ist. In [52] entfällt diese Einschränkung durch die Einführung schwacher Typisierungen und abstrakter Faltungen. Die Konstruktion des migrierten Systems auf der rechten Seite einer Schema-Transformation wird analog zum Ansatz in dieser Arbeit über eine geeignete Faktorisierung durchgeführt. Das Transformations-Modell in dieser Arbeit unterscheidet sich insofern von dem Transformations-Modell in [52], als dass die Eigenschaften von abstrakten Faltungen nicht untersucht werden, weil das zugrunde liegende Daten-Modell (ausgedrückt über die Spezifikation M) wesentlich komplexer als das Graph-Modell in [51] und [52] ist.

In [55] werden Refactorings von Programmen durch Graph-Transformationen ausgedrückt und die Erhaltung des Programmverhaltens sowie weiterer Eigenschaften untersucht. Dazu werden ein Graph-Modell für die Klassenstruktur und für Methoden eingeführt sowie Graph-Transformationen in diesem Modell definiert, um Refactorings wie “Methode hochschieben” zu beschreiben. Wie die Refactorings sich auf vorhandene Prozesse auswirken, wird nicht diskutiert. Zur Komposition von Refactorings wird lediglich im Ausblick erwähnt, dass diese noch zu untersuchen ist.¹⁴

¹² “Graph transformation is good for modelling references and their manipulation, but not primitive values and their operations. [...] Therefore, [...] we will not simulate primitive operations.” [39, Abschnitt 7.3.1]

¹³ “One interesting consequence is that we cannot simulate the effect of conditional statements precisely (since the value of the boolean expression cannot be computed); therefore the simulation becomes *non-deterministic* at this point.” [39, Abschnitt 7.3.1]

¹⁴ “In the longer run, we want to investigate combinations of refactorings.” (Kap. 5).

In [20, 23, 24] werden so genannte Modell-Transformationen betrachtet, bei denen die Ausprägungen zwischen zwei Modellen zu *unterschiedlichen* Meta-Modellen transformiert werden. Dabei werden die Meta-Modelle als stabil betrachtet. Ein in diesen Arbeiten häufig zitiertes Beispiel ist die Transformation von Zustandsdiagrammen in Petri-Netze, wobei mit Hilfe von Graph-Transformationen Elemente aus dem Ausgangsmodell in Elemente zum Zielmodell umgeformt werden. Diese Vorgehensweise ist sinnvoll im Zusammenhang mit Modell-getriebener Entwicklung, wo zwischen einem Plattform-unabhängigen Modell “PIM”¹⁵ und einem Plattform-abhängigen Modell “PSM”¹⁶ vermittelt werden muss. Sie lässt sich jedoch nur bedingt übertragen, denn die Anforderung dieser Arbeit ist gerade die Annahme, dass das Schema, das in den beschriebenen Ansätzen dem Meta-Modell entspricht und als stabil angesehen wird, häufigen Veränderungen unterworfen ist. Werden die Modell-Transformationen hingegen so gewählt, dass sie die Veränderungen eines Schemas zu einem festen Meta-Modell beschreiben, wie es in dieser Arbeit geschieht, dann bleibt in [20] die Frage offen, wie Ausprägungen der veränderten Modelle übertragen werden. In [23, 24] wird dafür ein Konzept entwickelt, das im nächsten Abschnitt angerissen wird.

In [26] werden Transformationen eines Zustandsdiagramms durch Graph-Transformationen beschrieben und mit Hilfe des AGG-Werkzeugs [80] modelliert und durchgeführt. Dabei wird das Nachziehen von vorhandenen Ausprägungen – in diesem Fall von laufenden Simulationen zu dem Zustandsdiagramm – jedoch nicht thematisiert. Auch ist es nötig, mehrere Transformationsregeln zur Umsetzung der gewünschten Modell-Änderung (im zitierten Beispiel ist es die Einführung eines initialen Pseudo-Zustands) zu entwickeln und in einer bestimmten Reihenfolge anzuwenden. Die Komposition dieser Regeln ist nicht ohne Weiteres möglich.¹⁷ Derselbe Ansatz mit denselben Einschränkungen wird in [63] angewandt, um die Refactorings “Methode verschieben” und “Feld kapseln” zu beschreiben.

In [73, 74] wird ein formaler Ansatz für allgemeine Modell-Transformationen beschrieben, der ebenfalls auf der Kategorientheorie aufbaut. Im Vergleich zu dem Modell in Teil III werden die Transformationen direkt auf den Ausprägungen über Regeln und Ansätze ähnlich zu Graphstruktur-Transformationen formuliert, wobei je nach Transformation Ansätze ins Ausgangssystem in Ansätze ins angepasste Zielsystem übertragen werden. Es wird somit keine direkte Beziehung zwischen dem Ausgangsschema und dem Zielschema formuliert, wie es in dieser Arbeit geschieht. Dafür ist der Ansatz insofern allgemeiner, als dass er Modell-Transformationen zwischen Modellen zu unterschiedlichen Meta-Modellen erlaubt.

In [33] werden Transformationen von Typ-Graphen mit Vererbung betrachtet. Es wird gezeigt, dass unter der Bedingung, dass die Morphismen einer Transformationsregel Vererbung reflektieren, eine schwache adhäsive HLR-Kategorie definiert werden kann, innerhalb derer Typ-Graphen mit Hilfe

¹⁵ Abkürzung für “Platform Independent Model”

¹⁶ Abkürzung für “Platform Specific Model”

¹⁷ “The AGG tool does not provide a satisfactory control structure for organizing and combining rules, and the supplied mechanisms for composing rules were not sufficient to describe model refactorings.” (Kap. 5).

von Graph-Transformationsregeln transformiert werden können. Die Vererbung reflektierenden Morphismen sind vergleichbar mit den vervollständigenden Homomorphismen, die in Teil III eingeführt werden, um Programme als Graph-Transformationsregeln formulieren zu können. Auf Schema-Ebene haben Transformationen in dieser Arbeit diese Einschränkung nicht und sind somit allgemeiner, denn gerade Änderungen der Klassen-Struktur durch Hinzufügen und Entfernen von Ober- und Unterklassen sind häufig notwendig und sinnvoll, wie auch die Fallstudie im nächsten Kapitel belegt.

2.4 Ableitung von Ausprägungs-Migrationen aus Schema-Transformationen

Die vorliegende Arbeit zielt darauf ab, Ausprägungs-Migrationen formal aus Schema-Transformationen abzuleiten, so dass sie *korrekt* und *eindeutig* sind. Neu ist, dass das Modell auch für komplexere Schema-Transformationen automatisch eine entsprechende Ausprägungs-Migration zur Verfügung stellt. Unter komplexeren Schema-Transformationen werden im Folgenden Transformationen verstanden, die über ein Erzeugen, Verändern oder Löschen eines einzelnen Schema-Elements (Klasse, Assoziation etc.) hinausgehen und ein Zusammenspiel *mehrerer* Elemente voraussetzen. Dazu gehören etwa das Aufspalten und Zusammenlegen von Klassen und das Verschieben von Assoziationen entlang von Vererbungsbeziehungen.

Eine der ersten Untersuchungen zu Typ-Änderungen in objektorientierten Datenbanken ist in [78] zu finden. Es wird vorgeschlagen, dass jede Schema-Änderung eine neue Version des Schemas erzeugt, wobei Objekte immer zu genau einer Schema-Version gehören.¹⁸ Die Migration von Objekten zwischen verschiedenen Versionen desselben Schemas wird als sinnvoll erachtet¹⁹, eine weiter gehende Betrachtung dieser Thematik erfolgt jedoch nicht.

Die meisten Arbeiten zur Schema-Evolution (etwa [6, 7, 41, 46, 62, 89]) betrachten lediglich sehr einfache Schema-Transformationen, die auf der Ausprägungs-Ebene entweder zu keinen Effekten führen (z. B. bei Umbenennung von Schema-Komponenten) oder bei denen die durchzuführenden Aktionen trivial sind (Hinzufügen von Null-Werten bei der Erzeugung neuer Attribute und Löschen von Werten bei der Entfernung bestehender Attribute). Fast alle dieser Arbeiten basieren auf der Taxonomie von Schema-Transformationen der objektorientierten Datenbank ORION, die als erstes in [6, 7] veröffentlicht wurde und als erste grundlegende atomare Schema-Transformationen wie das Erstellen, Umbenennen und Löschen von Klassen, Attributen und Methoden sowie das Verändern der Vererbungsstruktur katalogisiert. Komplexere Transformationen, die das Verteilen bestehender Assoziationen und Attribute zwischen Klassen ermöglichen, werden in

¹⁸ “An object is bound throughout its life to a single version of a type; its properties and operations are defined by that one particular version of its type.” [78, Abschnitt 4.1.2]

¹⁹ “It may be desirable to convert instances of one version into those of another version of the same type.” [78, Abschnitt 4.1.2]

diesen Arbeiten nicht untersucht. Insbesondere lassen sich Verschiebe-Operationen nur durch Löschen und (Neu-)Erstellen der betreffenden Assoziationen und Attribute ausdrücken, was unumgänglich Datenverluste auf der Ausprägungs-Ebene nach sich zieht [89, Kapitel 3].

In [48] und [47] werden erstmals komplexere Schema-Transformationen ermöglicht, die nicht zu Datenverlusten führen (etwa das oben erwähnte Verschieben einer Assoziation). Die Migration der Objekte auf Ausprägungs-Ebene erfolgt durch einen Algorithmus, der entweder eine explizite Beschreibung der Beziehungen zwischen Elementen im alten und neuen Schema benötigt [48] oder der durch heuristische Verfahren diese Beziehungen selbst herstellt [47]. Diese Vorgehensweise ist sehr flexibel. Sie hat im ersten Fall jedoch den Nachteil, dass der Transformationsprozess zusätzlich zur Schema-Änderung redundant beschrieben werden muss²⁰, und Redundanzen generell vermieden werden sollten. Im zweiten Fall können die mit heuristischen Verfahren gewonnenen Beziehungen zwischen Elementen im alten und neuen Schema inkorrekt sein²¹. Somit ist immer eine manuelle Kontrolle des Migrationsalgorithmus' seitens des Datenbank-Nutzers notwendig.²² Generell verfolgen die beiden Arbeiten den Ansatz des Schema-Matchings [66], um zwei unterschiedliche Schemata miteinander zu verknüpfen und daraus eine Ausprägungs-Migration abzuleiten²³, wobei manuelle Teilprozesse bewusst in Kauf genommen werden. Die vorliegende Arbeit hingegen hat eine *voll automatische* und *eindeutige* Migration der Ausprägungs-Ebene bei gegebener Schema-Transformation zum Ziel. Insbesondere behandelt die vorliegende Arbeit Schema-Transformationen als "First Class"-Objekte, anstatt sie durch Schema-Matching aus altem und neuen Schema zu rekonstruieren.

Einige Arbeiten [45, 46, 65, 88] verfolgen den Ansatz, Schema-Evolution durch *Datenbank-Sichten* ("Views") zu erreichen. Dabei existiert in der Regel ein Basis-Schema, welches neue Schema-Elemente aufnehmen kann, aus dem aber keine Schema-Elemente gelöscht werden. Veränderungen und Löschungen im Schema werden durch intelligente Sichten simuliert, indem die entsprechenden Elemente ausgeblendet werden. Dieser Ansatz hat den Vorteil, dass überhaupt keine Migration der Ausprägungs-Ebene stattfinden muss. Der Nachteil ist, dass das Basis-Schema und die Daten selbst nicht restrukturiert werden und somit nach mehreren Refactorings die generierten Sichten immer komplexer und weniger durchschaubar werden. Weiterhin werden in den Arbeiten komplexere Schema-Transformationen in der Regel nicht unterstützt. [65] basiert auf den Schema-Modifikationen aus [72], die jedoch keine Operationen zum Verschieben von Elementen innerhalb der Klassenhierarchie enthalten (etwa das Verschieben eines Attributs zu einer Ober-

²⁰ "To provide general database reorganization [...] the database administrator must describe the relationships among objects in the old version of the database and those in the new. OTGen provides a tabular notation in which this is done. The table [...] indicates how instances of that class should be transformed." [48, Kapitel 3]

²¹ "The disadvantage is that the system must now infer how the types have changed instead of being explicitly told. [...] Of course, it is possible that the algorithms will make incorrect inferences." [47, Kapitel 2]

²² "As a result, it is important for the maintainer to be involved in the type comparison process." [47, Kapitel 2]

²³ vgl. hierzu auch [56]

klasse). Die Arbeiten [46, 88] beziehen sich explizit auf die ORION-Schema-Transformationen aus [7]. [45] erweitert das RiBS-Modell um komplexere Restrukturierungen von Aggregaten, etwa das Verschieben von Attributen entlang von Kompositionsbeziehungen, und diskutiert auch die Problemstellung, wie bestehende Datenbank-Abfragen nach einer Schema-Transformation angepasst werden müssen. Allerdings wird nicht deutlich, ob die präsentierten komplexen Transformationen auch auf Vererbungshierarchien übertragbar sind. Andere Arbeiten, die explizit komplexere Schema-Transformationen untersuchen [11–13], betrachten nicht das Problem der Daten-Migration.²⁴

In [71, 81] werden amalgamierte Graph-Transformationsregeln vorgestellt. Die Idee ist hierbei, eine Regel, die eine gewünschte Transformation der Ausprägung formuliert, so oft es möglich ist, anzuwenden, wobei Konflikte über eine gemeinsame Unterregel gelöst werden. Dieser Ansatz erlaubt somit die Formalisierung einer Migration über abstrakte Ausprägungs-Regeln. Im Gegensatz zu dem Modell in dieser Arbeit werden diese Regeln nicht automatisch aus einer Schema-Transformation abgeleitet, vielmehr müssen die Regeln für gewünschte Schema-Transformationen (einmal) manuell entwickelt werden. Weiter bestehen Graph-Transformationsregeln in der Regel aus injektiven Morphismen, so dass keine Partikel aufgespalten und zusammengelegt werden können, was für das Daten- und Prozessmodell in Teil III wesentlich ist. Schließlich bleibt die Frage, wie die amalgamierten Transformationsregeln typisiert sind: Da die linke Seite einer amalgamierten Migration im alten und die rechte Seite im neuen Schema typisiert sein muss, kann die Typisierung einer kompletten Transformationsregel nur dann erfolgen, wenn ein drittes Schema verwendet wird, in dem sowohl das alte als auch das neue Schema eingebettet sind (vgl. [20, Kap. 13]). Hier stellt sich dann das Problem der Korrespondenz identischer Elemente im alten und neuen Schema, die dann wiederum über Morphismen oder ähnliche Konzepte hergestellt werden muss. Genau diese Korrespondenz ist hingegen inhärenter Bestandteil der Schema-Transformationen, die in Teil III dieser Arbeit vorgestellt werden. Somit kommt das Transformations-Modell dieser Arbeit insgesamt mit weniger Informationen aus, die es für eine gewünschte Ausprägungs-Migration zu spezifizieren gilt.

In [23, 24] wird untersucht, wie bei der Änderung eines Meta-Modells abhängige Modelle und Modell-Transformationen, die als Graph-Transformationsregeln repräsentiert werden, angepasst werden müssen. Dies entspricht in dieser Arbeit der Migration von Programmen und Prozessen bei Schema-Änderungen. Im Papier wird die Änderung des Meta-Modells, die selbst wieder eine Graph-Transformation ist, einfach als Regel auf die Graphen der Modell-Transformationen angewandt. Dieselbe Idee wird auch in dieser Arbeit verfolgt, nur dass Schema-Änderungen und deren Übertragung auf Programme und Prozesse nicht über Ansätze einer Graph-Transformationsregel in der Ausprägung, sondern über eine Typisierung der Programme und Prozesse in der Schema-Transformation formalisiert sind. Dies vereinfacht den Transformationsprozess, weil keine Ansätze

²⁴ “Due to limited space, we did not address the notion of database update following a schema change.” [11, Kapitel 4]

gesucht werden müssen und weil die Migration durch die Anwendung des Pullback-Funktors und des Epirefektors “in einem Rutsch” stattfinden kann.

In [17] werden Modelländerungen zwischen zwei Modellen auf ein drittes Modell automatisiert übertragen. Übertragen auf die Zielsetzung dieser Arbeit bedeutet dies, eine so genannte “Verwebung” zwischen dem Ausgangsschema und der zugehörigen Ausprägung über spezielle Web-Verknüpfungen herzustellen, so dass im zweiten Schritt Schema-Änderungen zwischen Ausgangs- und Zielschema auf die Ausprägung zum Ausgangsschema automatisiert übertragen werden können. Es bleibt zu untersuchen, ob solche Web-Modelle automatisch aus der Typisierung von Ausprägungen in einem Schema gewonnen werden können.

2.5 Komposition von Schema-Transformationen

Die Synthese mehrerer kleiner Schema-Transformation zu einer großen Schema-Transformation ist ein wichtiges Thema in der vorliegenden Arbeit. Ein Grund liegt in der verbesserten Performanz: Anstatt mehrere Einzel-Migrationen durchzuführen, wird die aus der komponierten Schema-Transformation abgeleitete Gesamt-Migration genau einmal durchgeführt [54, Abschnitt 3.1.2]. Ein weiterer Grund ist die Möglichkeit, Systeme und Transformationen als Kategorie zu formulieren, was vielversprechend ist und ein weites Feld für weitere Forschungsaktivitäten darstellt.

Einige Arbeiten (etwa [60, 68]) beschreiben zusammengesetzte Refactorings und weisen nach, dass die Korrektheit des komponierten Refactorings aus der Korrektheit der einzelnen Refactorings folgt. Dabei werden komponierte Refactorings jedoch nicht direkt und ohne Rückgriff auf die Einzelbestandteile dargestellt, sondern es ist immer die Ausführung jedes einzelnen Refactorings in der gewünschten Reihenfolge notwendig. In dieser Arbeit werden komponierte Transformationen explizit konstruiert, und es wird der Nachweis erbracht, dass deren Anwendung mit der Anwendung der Einzel-Transformationen übereinstimmt.

Der Transformations-Ansatz in [73, 74] berücksichtigt die explizite Konstruktion komponierter Transformationen, indem die Regeln mit Hilfe von Pushouts miteinander verschmolzen werden. Die kategorielle Herangehensweise ähnelt dem Ansatz zur Komposition von Schema-Transformationen in Teil III, wobei in dieser Arbeit Pullbacks statt Pushouts verwendet werden, um Transformations-Spans zu komponieren. Die Ansätze unterscheiden sich jedoch voneinander, weil in [73, 74] alle zu komponierenden Regeln dasselbe Ausgangs- und dasselbe Zielschema besitzen, während in dieser Arbeit eine komponierte Transformation sich aus zwei Transformationen zusammensetzt, bei denen das Ausgangsschema der zweiten Transformation das Zielschema der ersten darstellt.

2.6 Migration von Daten, Programmen und Prozessen

Das in dieser Arbeit vorgestellte Modell erlaubt es, objektorientierte Daten *und* Software (Programme und zugehörige Prozesse) als Ausprägung eines kombinierten Daten- und Software-Schemas zu

modellieren. Schema-Transformationen führen automatisch zu einer Migration der Daten und der darauf operierenden Software. Dem Autor ist kein weiterer Ansatz aus der Literatur bekannt, der die Transformation von objektorientierten Daten, Programmen und Prozessen derart kombiniert, dass Schema-Änderungen kanonisch und eindeutig auf Daten und Software fortgesetzt werden. Die in [18] und [38, 39] vorgestellten Modelle sind zwar in der Lage, objektorientierte Programme und Prozesse zu beschreiben. Die Transformation des zugrunde liegenden Schemas und die sich anschließende Migration der im Schema typisierten Programme und Prozesse werden in den Arbeiten jedoch nicht thematisiert.

Fallstudie: Problemstellung

Dieses Kapitel motiviert die Fragestellung, warum die Veränderung objektorientierter Systeme wichtig und sinnvoll ist. Anhand eines Beispiels werden in den Abschnitten 3.1 bis 3.4 einige typische Evolutionsprozesse in der Software-Entwicklung vorgestellt. Die sich in Abschnitt 3.5 anschließende Diskussion erörtert die in dem Beispiel skizzierten Verfahren und führt die für die weitere Behandlung der Thematik zentralen Konzepte wie Transformation, Refactoring und Migration informell ein. Die Darstellung der Klassen- und Objektdiagramme in diesem Kapitel erfolgt mit Hilfe der UML-Notation [29, 64].

3.1 Ausgangssituation

Eine Versicherungsgesellschaft bietet als Versicherungsprodukt eine Haftpflichtversicherung für Privat- und Geschäftskunden an. Kunden werden mit Namen und einer Liste von Anschriften sowie ihrem Haftpflichtversicherungsprodukt in einer objektorientierten Datenbank verwaltet. In Abb. 3.1 sind das Schema sowie beispielhafte Daten zu diesem Schema dargestellt.

3.2 Phase 1: Hinzunahme eines neuen Versicherungsprodukts

Nun plant die Versicherungsgesellschaft, eine Rechtsschutzversicherung als zweites Versicherungsprodukt in ihre Produktlinie aufzunehmen. Das Schema muss dementsprechend angepasst werden. Dazu wird zunächst von der Haftpflichtversicherung abstrahiert, indem eine Oberklasse *Versicherung* eingeführt wird. Im zweiten Schritt wird das Attribut *vertragsnummer* in diese neue Klasse verschoben. Als drittes wird die von der Klasse *Kunde* ausgehende Assoziation *produkt* auf die neue Klasse *Versicherung* gerichtet. Schließlich wird eine neue Klasse *Rechtsschutzversicherung* als Unterklasse der Klasse *Versicherung* ergänzt. Das sich ergebende Schema ist in Abb. 3.2 zu sehen. Auf die Ausprägungen haben diese Schema-Änderungen keine Auswirkungen, somit entfällt eine Abbildung der Ausprägungs-Ebene.

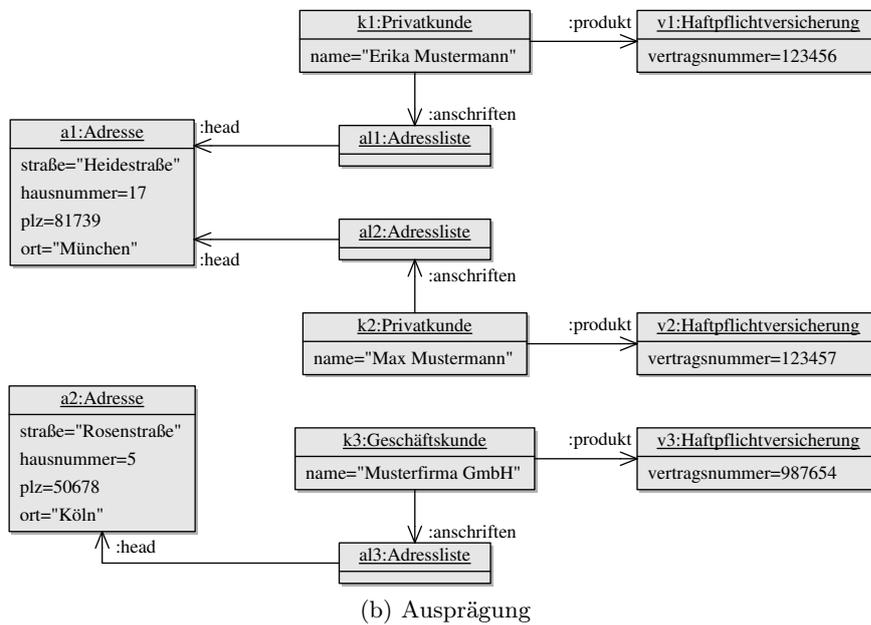
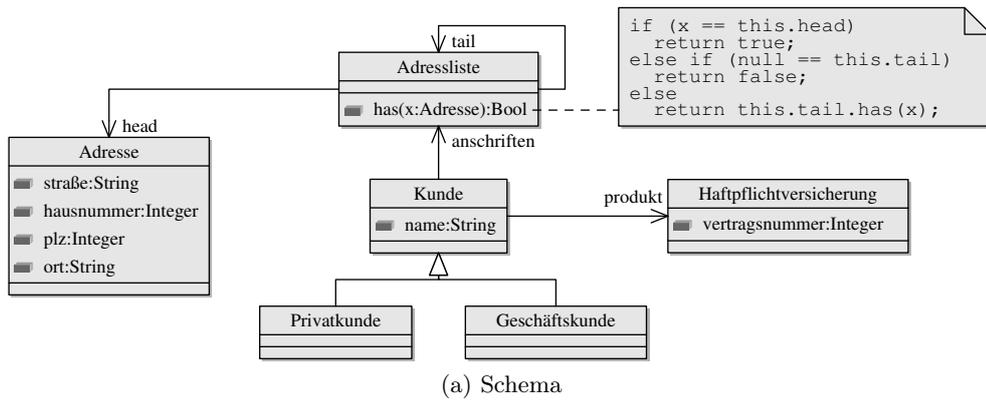


Abb. 3.1: Ausgangssituation

3.3 Phase 2: Hinzunahme ausländischer Kunden

Bisher waren alle Kunden der Versicherungsgesellschaft inländische Kunden. Jetzt kommt ein ausländischer Kunde hinzu. Hierfür ist das bisherige Schema jedoch nicht vorbereitet, weil nur inländische Adressen abgebildet werden können. Um auch ausländische Anschriften zu unterstützen, wird die bisherige Klasse *Adresse* zuerst in *Inlandsadresse* umbenannt. Dann werden analog zur letzten Phase eine neue Oberklasse *Adresse* und eine neue Geschwisterklasse *Auslandsadresse* erstellt, wobei letztere eine Adresse in Freitextform akzeptiert. Weiter wird die von der Klasse *Adressliste* ausgehende Assoziation *head* auf die neue Klasse *Adresse* gerichtet. Schließlich wird der Typ des Parameters *x* der Operation *has* der Klasse *Adressliste* zur neuen Klasse *Adresse*

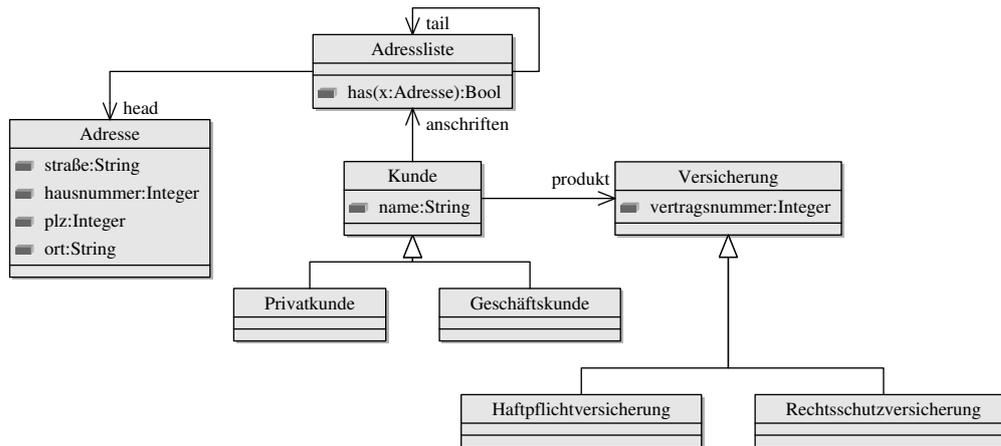


Abb. 3.2: Erweiterung des Versicherungs-Modells

verallgemeinert. Das resultierende Schema ist in Abb. 3.3a zu sehen. Die einzige Änderung auf der Ausprägungs-Ebene ist die Umbenennung der Klasse der Adress-Objekte (Abb. 3.3b).

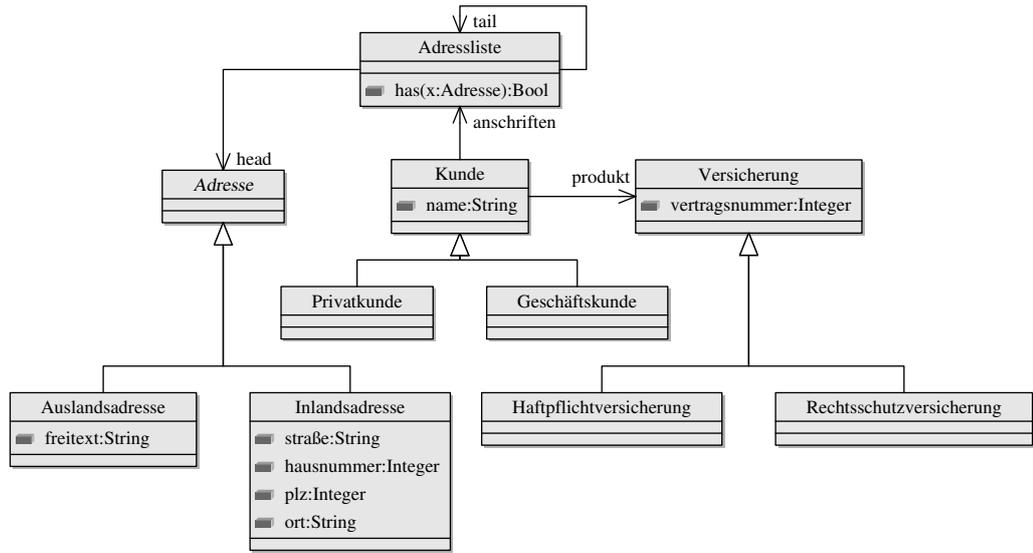
3.4 Phase 3: Konzentration auf Geschäftskunden

Die Geschäftsführung der Versicherungsgesellschaft hat entschieden, sich künftig auf Geschäftskunden zu konzentrieren und das Privatkundengeschäft aufzugeben. Nachdem die Verträge mit Privatkunden ausgegliedert worden sind, ist das Schema entsprechend anzupassen. Dazu wird zunächst die Klasse *Privatkunde* gelöscht. Im zweiten Schritt werden die Klassen *Kunde* und *Geschäftskunde* zu einer Klasse *Geschäftskunde* zusammengefasst. Das resultierende Schema sowie die angepasste Ausprägung sind in Abb. 3.4 dargestellt. Eine Änderung der Ausprägungs-Ebene ist notwendig, um Ausprägungen zu nicht mehr vorhandenen Schema-Elementen (hier: Objekte zur Klasse *Privatkunde*) zu löschen.

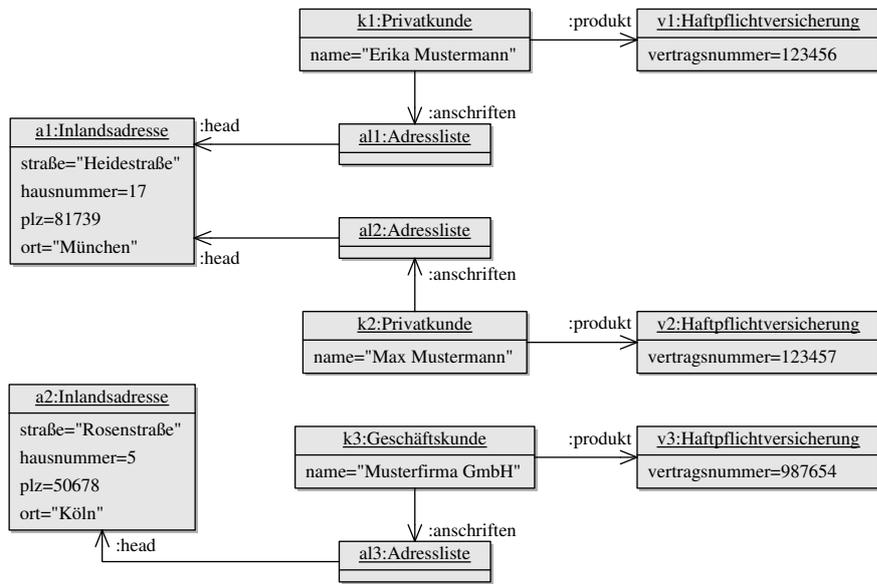
3.5 Diskussion

In dem aufgezeigten Beispiel hat das Schema mehrere Veränderungen erfahren. Diese Veränderungen lassen sich in einem ersten Schritt grob in fünf Kategorien einteilen:

- (1) Erstellung neuer Elemente (Beispiel: Erstellung der Klasse *Rechtsschutzversicherung* in Phase 1)
- (2) Aufspalten von Elementen (Beispiel: Aufspalten der alten Klasse *Adresse* in die beiden neuen Klassen *Adresse* und *Inlandsadresse* in Phase 2)

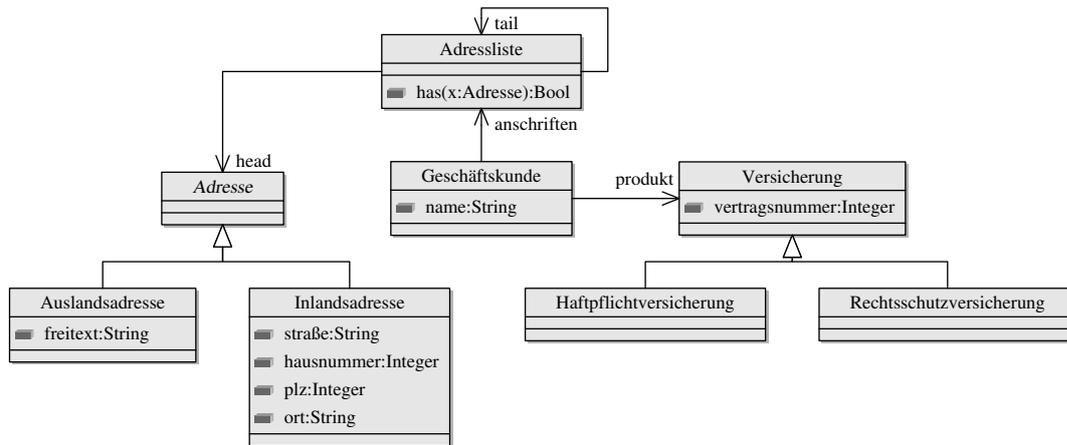


(a) Schema

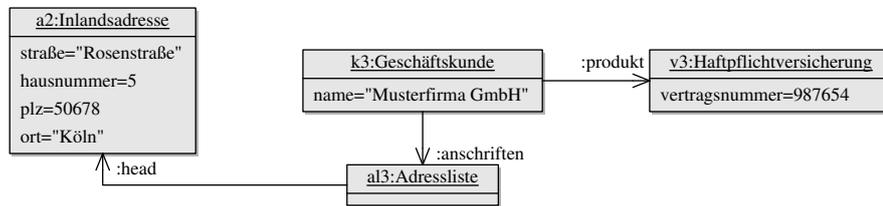


(b) Ausprägung

Abb. 3.3: Erweiterung des Adress-Modells



(a) Schema



(b) Ausprägung

Abb. 3.4: Reduktion des Kunden-Modells

- (3) Zusammenfassen von Elementen (Beispiel: Zusammenfassen der beiden alten Klassen *Kunde* und *Geschäftskunde* zur neuen Klasse *Geschäftskunde* in Phase 3)
- (4) Verschieben von Elementen in der Klassenhierarchie (Beispiel: Verschieben des Attributs *vertragsnummer* und des Ziels der Assoziation *produkt* von der Klasse *Haftpflichtversicherung* zur neuen Klasse *Versicherung* in Phase 1)
- (5) Löschung bestehender Elemente (Beispiel: Löschung der Klasse *Privatkunde* in Phase 3)

Es ist einleuchtend, dass diese Veränderungen unterschiedliche Auswirkungen auf die Datenwelt haben. Beispielsweise führen Löschungen im Schema unweigerlich zum Verlust aller Daten, die in den gelöschten Schema-Elementen typisiert sind. Das Erstellen von neuen Schema-Elementen hingegen wirkt sich in der Regel überhaupt nicht auf existierende Daten aus. Die Betrachtung der Situation bei Programmen, die gegen das Schema programmiert sind, führt zu ähnlichen Ergebnissen: Nicht alle Schema-Veränderungen sind so “vernünftig”, dass bestehende Programme nach der Veränderung weiter funktionieren.

Im Folgenden soll eine erste vorläufige Klassifizierung der verschiedenen Arten von Veränderungen in Bezug auf ihre “Vernünftigkeit” und auf ihren “Einsatzort” durchgeführt werden. Eine detaillierte

Betrachtung der Begriffe erfolgt in Teil II in Kapitel 4, eine mathematische Formalisierung geschieht in Teil III.

- Eine *Transformation* ist eine beliebige Veränderung eines Schemas, der Ausprägungen, der Programme oder Prozesse eines Systems.
- Transformationen eines Schemas werden *Restrukturierungen*, Transformationen von Daten, Programmen und Prozessen werden *Migrationen* genannt. Geht eine Migration als unmittelbare Fortsetzung einer Restrukturierung auf der Ausprägungs-Ebene hervor, so wird die Migration durch die Restrukturierung *induziert*.
- Ein *Refactoring* ist eine “vernünftige” Transformation in dem Sinne, als dass kein Informationsverlust eintritt.

In den restlichen Teilen der Arbeit werden diese Begriffe und Prozesse Schritt für Schritt verfeinert. Dies gipfelt in Teil III, in dem die Formalisierung soweit vorangetrieben ist, dass mit Hilfe mathematischer Modelle Restrukturierungen und induzierte Migrationen beschrieben und deren Effekte auf Angemessenheit überprüft werden können. Dabei zieht sich die Fallstudie durch die Arbeit und zeigt in Teil II in Kapitel 6 sowie in Teil III in Kapitel 13, wie sich das präsentierte Beispiel erst im konzeptionellen und dann im mathematischen Modell formulieren lässt.

Konzeptionelles Modell

Transformation von Software und Datenbanken

Dieses Kapitel befasst sich im Detail mit den zentralen Transformationskonzepten dieser Arbeit. Zuerst werden in den Abschnitten 4.1, 4.2 und 4.3 die zu transformierenden Einheiten analysiert und definiert. Die Abschnitte 4.4 bis 4.6 definieren die Begriffe *Transformation*, *Refactoring* und *Entwicklung*, sowohl auf Software- als auch auf Datenbank-Ebene, und stellen einen Bezug zu den in der Literatur zu findenden Konzepten her. Schließlich behandelt Abschnitt 4.7 *induzierte Migrationen*: eine Unterklasse von Transformationen der Ausrägungsebene, die allein durch eine *Restrukturierung* des Schemas ausgedrückt werden können und aus denen eine *Migration* der Datenbanken und Software zu diesem Schema abgeleitet werden kann. Kapitel 11 greift diese Klasse von Transformationen auf und stellt anhand vieler kleiner und praktischer Beispiele ihre Vielfalt und Nützlichkeit bei der Transformation objektorientierter Systeme dar.

4.1 Software

Definition 4.1 (Programm). *Ein Programm ist eine endliche Sammlung von Regeln, die eine Sequenz von Anweisungen beschreiben, um ein bestimmtes Problem zu lösen. Typischerweise benötigt ein Programm zur Lösung seiner Aufgabe Eingaben (“input”) und produziert zu den Eingabedaten entsprechende Ausgaben (“output”). Ein Programm ist in einer unmissverständlichen Weise notiert, üblicherweise in einer Programmiersprache (angelehnt an [42, Abschnitt 1.1]).*

Definition 4.2 (Prozess). *Ein Prozess ist ein sich in Ausführung befindliches Programm [83, Abschnitte 1.5.1 und 2.1.1]. Er besitzt einen Zustand, der sich aus Daten und Ausführungskontext zusammensetzt. Die Daten des Prozesses bestehen zum einen aus den Eingabedaten, die vor und während des Programmablaufs an den Prozess übermittelt werden, und zum anderen aus Daten, die während des Programmablaufs durch Verarbeitung anderer Daten entstanden sind. Der Ausführungskontext speichert alle Informationen, die mit der Ausführung des Programms “an sich” zusammenhängen und vom jeweiligen Modell der virtuellen Maschine²⁵ der verwendeten*

²⁵ Die virtuelle oder abstrakte Maschine einer Programmiersprache repräsentiert die Semantik der Ausführung von Programmen einer Programmiersprache und ist nicht zwingend ein rein virtuelles oder

Programmiersprache abhängen. Dazu gehören etwa die Rücksprungadressen für Methodenaufrufe²⁶, Informationen über Ausnahmebehandler²⁷, Wiedereintrittspunkte von Koroutinen²⁸ etc.

Im objektorientierten Paradigma besteht ein Programm aus einem Satz von Methoden, welche Anweisungen und/oder Ausdrücke enthalten und damit die Bedeutung der jeweiligen Methode gemäß der Semantik der verwendeten Programmiersprache festlegen. Die Anweisungen und Ausdrücke führen beim Abarbeiten des Programms in einem Prozess zu der Entstehung von Nachrichten, die an Objekte geschickt werden und dort zur Ausführung entsprechender Methoden führen. Die gesamte Dynamik eines objektorientierten Prozesses lässt sich also durch ein System von miteinander über Nachrichten interagierenden Objekten begreifen, wobei die existierenden Objekte die Daten und die noch zu verarbeitenden Nachrichten den Ausführungskontext des Prozess-Zustands bilden.

Kennzeichnend für die weitere Betrachtung von Programm- und Prozess-Transformationen in der vorliegenden Arbeit ist die Annahme, dass Programme und Prozesse in demselben *Schema* strukturiert sind, wobei das Schema die Typ- und Operationsstruktur widerspiegelt und in der Software-Entwicklung *Entwurf* genannt wird. In objektorientierten Programmen besteht das Schema aus *Klassen*, die durch Vererbung und Assoziationen miteinander in Beziehung stehen, und aus *Operationen* mit Ein- und Ausgabeparametern und weiteren Eigenschaften²⁹. Das Schema schränkt zum einen die Struktur der Objekte, Verknüpfungen, Werte und Nachrichten in Prozessen und die Struktur der Methoden in Programmen ein. Zum anderen fordert es die Existenz von Elementen in einem Programm. So muss in einem objektorientierten Programm für jede Operation einer nicht abstrakten Klasse eine Methode vorliegen.

Das Dreiergespann “Schema, Programm, Prozess” führt zu der in der vorliegenden Arbeit gebrauchten Definition von Software:

Definition 4.3 (Software). *Der Begriff Software steht in dieser Arbeit zusammenfassend für ein Programm inklusive eventuell laufender zugehöriger Prozesse, wobei das Programm und die Prozesse alle in demselben Schema typisiert sind.*

Die Typisierung von Programmen und Prozessen in einem Schema erlaubt es, Transformationen auf Schema-Ebene zu formulieren und diese auf beliebige Programme und Prozesse zu diesem Schema anzuwenden. Mehr dazu steht in Abschnitt 4.7.

abstraktes Konzept. Beispielsweise entsprechen sich virtuelle und konkrete Maschine bei Maschinsprachen vollständig.

²⁶ imperative Programmiersprachen wie z. B. *C*, *C++*, *Java*

²⁷ Programmiersprachen mit Ausnahmebehandlung wie z. B. *Java*, *C++*, *Python*

²⁸ z. B. *Simula-67*, *Modula-2*

²⁹ etwa Ausnahmespezifikationen

4.2 Datenbanken

Eine *Datenbank* ist nach [86] “eine Sammlung von Daten, welche Fakten über eine spezielle Anwendung der realen Welt repräsentiert” [86, Abschnitt 1.3]. Eine solche Datensammlung ist immer einem *Daten-Schema* unterworfen.³⁰ Dieses Daten-Schema beschreibt die invariante *Struktur* der Datensammlung und trennt damit erlaubte Daten mit dieser Struktur von verbotenen Daten ohne diese Struktur. Die Daten zu einem Daten-Schema heißen in dem Daten-Schema *typisiert*.

Zusammengefasst ergibt sich:³¹

Definition 4.4 (Datenbank, Schema, Ausprägung). *Eine Datenbank ist im Folgenden ein Schema zusammen mit darin typisierten Daten, die eine Ausprägung des Schemas darstellen.*

Einem Daten-Schema liegt ein bestimmtes *Datenmodell* zugrunde. Dieses Datenmodell legt fest, welche Abstraktionskonzepte ein Daten-Schema verwenden kann und wie sie miteinander in Beziehung stehen. Bekannte Datenmodelle umfassen das *hierarchische*, *Netzwerk-*, *Entity-Relationship-*, *relationale* und *objektorientierte* Datenmodell. In dieser Arbeit werden ausschließlich Daten-Schemata zum objektorientierten Datenmodell behandelt. Somit sind alle betrachteten Datenbanken objektorientiert.

4.3 Systeme und Welten

Prozesse können ohne Daten nicht existieren, und Daten ohne verarbeitende Prozesse nützen niemandem. Das Erkennen dieser gegenseitigen Abhängigkeit ist wichtig für die Betrachtung der Transformation von Daten und Software, führt es doch dazu, die Transformation von Daten und die Transformation von Software nicht isoliert zu betrachten, sondern den Fokus auf die Transformation von *beidem* zu legen. Dies motiviert die folgende Definition eines *Systems*:

Definition 4.5 (System). *Ein System bezeichnet im Folgenden eine Zusammenfassung von Software und Datenbanken, wobei die Software und die Datenbanken alle in demselben Schema typisiert sind.*

Weiterhin ist es häufig nützlich, ein Schema samt zugehörigen Ausprägungen als Ganzes zu betrachten. In der vorliegenden Arbeit wird dafür der Begriff *Welt* verwandt. Typische Anwendungen sind Datenwelten (= Datenbanken), Programmwelten (= typisierte Programme) und Prozesswelten (= typisierte Prozesse).

Definition 4.6 (Welt). *Eine Welt bezeichnet im Folgenden ein Schema samt zugehöriger Ausprägungen.*

³⁰ Dies gilt im gewissen Sinne auch für *semi-strukturierte Datenbanken* [1, 15], denn diese können als Daten zu einem nicht oder nur sehr wenig einschränkenden Schema angesehen werden.

³¹ vgl. zu Definition einer Datenbank auch [51]

4.4 Transformationen

Definition 4.7 (Transformation, Restrukturierung, Migration). *Unter einer Transformation wird eine Veränderung verstanden, die keinen semantischen Bedingungen (etwa Erhaltung der Verhaltenssemantik) genügen muss. Schema-Transformationen werden Restrukturierungen, Ausprägungs-Transformationen Migrationen genannt.*

Transformationen sind Veränderungen eines betrachteten Elements, die keinerlei Beschränkungen unterliegen. Sie können zum einen nach dem zu transformierenden Element unterschieden werden; daraus ergibt sich die Einteilung in *Software-, Datenbank- und System-Transformationen*. Zum anderen kann auch die Ebene betrachtet werden, auf welche eine Transformation angewandt wird; daraus ergibt sich die Einteilung in *Schema-, Ausprägungs- und Welt-Transformationen*. Schließlich können Unterkategorien gebildet werden, die bestimmte Eigenschaften aufweisen, etwa Refactorings (siehe nächsten Abschnitt).

4.5 Refactorings

Der Begriff *Refactoring* wird in der Literatur fast ausschließlich in Hinblick auf das Restrukturieren von Software gebraucht. Der Begriff wurde geprägt von *William Opdyke*, der in seiner Dissertation [60] Refactorings als Programm-Restrukturierungen einführt unter Beibehaltung der Semantik der zu transformierenden Software. Nur wenige Publikationen haben diesen Begriff auf Datenbanken ausgeweitet, etwa [3]. Im Bereich von Datenbanken wird stattdessen von *Schema-Evolution* gesprochen [70]. Dieser Abschnitt beleuchtet zunächst Software-Refactorings und untersucht dann, wie der Begriff unter Berücksichtigung von Erkenntnissen aus dem Bereich Schema-Evolution sinnvoll auf Datenbanken übertragen werden kann.

4.5.1 Software-Refactorings

Software-Refactorings als semantikerhaltende Programm-Transformationen sind seit der grundlegenden Arbeit von William Opdyke [60] in aller Munde. Neben einer Vielzahl von Publikationen [19, 27, 30, 36, 40, 55, 61, 67, 84, 85] wurden auch viele Software-Entwicklungsumgebungen mit der Fähigkeit zum Refactoring von Software ausgestattet, etwa *eclipse* [87]. Kaum eine professionelle Software-Entwicklungsumgebung kann heutzutage auf dem Markt bestehen, ohne grundlegende Funktionen zum Software-Refactoring zu beinhalten. Die große Nachfrage der Industrie nach Unterstützung von Software-Refactoring ist auch durch die zunehmend verbreitete *agile Software-entwicklung* [8] zu erklären, welche eine rasche Reaktion auf sich ändernde Anforderungen und Randbedingungen im Verlauf der Software-Entwicklung fordert.

Um die Eigenschaften von Software-Refactorings besser einordnen zu können, ist es sinnvoll, zunächst Eigenschaften von Programmen zu betrachten. Programm-Eigenschaften, die im Kontext von Refactorings eine wichtige Rolle spielen, sind beispielsweise:

- *Korrektheit*: Das Programm liefert für die in der Spezifikation definierten Eingaben die entsprechend definierten Ausgaben.
- *Wartbarkeit*: Das Programm ist so strukturiert, dass es leicht an neue Anforderungen angepasst werden kann und bestehende Fehler schnell gefunden und eliminiert werden können.
- *Effizienz*: Das Programm geht mit den ihm zur Verfügung stehenden Ressourcen sparsam um.

Ein (beliebiger) Transformationsprozess, der ein Programm in ein anderes Programm umwandelt, kann nun alle, einige oder keine der oben genannten Eigenschaften des Quellprogramms im Zielprogramm erhalten (Korrektheit) bzw. verbessern oder verschlechtern (Wartbarkeit, Effizienz). Um festzustellen, welche Eigenschaften durch ein Refactoring erhalten bleiben, welche verbessert und welche eventuell verschlechtert werden, hilft ein Blick in die Definitionen von Software-Refactoring in der existierenden Literatur:

- “This thesis defines a set of program restructuring operations (refactorings) that support the design, evolution and reuse of object-oriented application frameworks. [...] Refactorings do not change the behavior of a program; that is, if the program is called twice (before and after a refactoring) with the same set of inputs, the resulting set of output values will be the same. Refactorings are behavior preserving so that, when their preconditions are met, they do not ‘break’ the program.” [60, Abschnitt 1.2]
- “Refactorings are behavior-preserving program transformations which can aid in the creation of new designs and the restructuring of legacy designs.” [85, Abschnitt 1.3]
- “Refaktorisieren ist der Prozess, ein Softwaresystem so zu verändern, dass das externe Verhalten nicht geändert wird, der Code aber eine bessere interne Struktur erhält.” [27, Vorwort]
- “Refaktorisierung [ist] eine Änderung an der internen Struktur einer Software, um sie leichter verständlich zu machen und einfacher zu verändern, ohne ihr beobachtbares Verhalten zu ändern.” [27, Abschnitt 2.1]
- “Refactorings are software transformations that restructure an object-oriented program while preserving its behaviour. [...] If applied well, refactorings improve the design of software, make software easier to understand, help to find bugs, and help to program faster.” [55, Kapitel 1]

Für Software-Refactorings ist offensichtlich zum einen die Erhaltung der *Korrektheit* eines Programms charakteristisch. Der funktionale Aspekt eines Programms wird durch ein Refactoring nicht verändert: Dieselben Eingaben führen vor und nach der Transformation zu denselben Ausgaben. Dadurch repräsentieren Refactorings in gewisser Weise “sichere” Programm-Transformationen.

Formalisiert wird das Beibehalten von Verhalten u. a. in [55]. Dort werden drei wichtige Unteraspekte definiert:

- Beibehaltung von Lese-Zugriffen auf Objekte (“Access preservation”) oder Erhaltung der *Navigation* zwischen Objekten
- Beibehaltung von Schreib-Zugriffen auf Objekte (“Update preservation”) oder Erhaltung der *Modifizierbarkeit* von Objekten³²
- Beibehaltung von Methoden-Aufrufen (“Call preservation”)

Während die letzte Regel nur auf Software-Refactorings angewandt werden kann, sind die ersten beiden Regeln auch bei Datenbank-Refactorings sinnvoll anwendbar, wie im nächsten Abschnitt gezeigt wird.

Der zweite wichtige Aspekt, der sich in den oben zitierten Definitionen abzeichnet, ist die Verbesserung der *Wartbarkeit* des Programms durch ein Refactoring. Dies ist die eigentliche Motivation von Software-Refactoring: Programme werden so umgestaltet, dass es nach der Transformation leichter fällt, sie geeignet zu erweitern oder anzupassen. Durch Software-Refactoring konnte sich somit die Methode, Software zu entwickeln, wandeln: Weniger Zeit und Ressourcen müssen in die Planung investiert werden, denn die Software kann durch konsequentes Refactoring permanent wartbar und somit veränderbar, erweiterbar und korrigierbar gehalten werden. Dadurch wird die Entwicklung beschleunigt, es kann früher getestet werden, da die ausführbare Software früher vorliegt, und der Auftraggeber kommt schneller in Kontakt mit (vorläufigen Versionen) der Software, was eine raschere Anpassung der Software an die tatsächlichen Bedürfnisse des Kunden ermöglicht.

Korrektheit und Wartbarkeit sind jedoch nicht die einzigen Kriterien, welche das Verhalten eines Programms ausmachen. Ein weiteres und insbesondere für den Kunden häufig ausschlaggebendes Kriterium ist die *Effizienz* eines Programms. Ein Beispiel mag dies verdeutlichen: Innerhalb eines Programms wird ein langsamer Algorithmus durch einen schnelleren mit derselben Funktionalität ersetzt. Dadurch ändert sich an der Funktion des Programms erst einmal nichts: dieselben Eingaben führen weiterhin zu denselben Ausgaben. Jedoch ist das Laufzeitverhalten des Programms ein anderes, denn das transformierte Programm mit dem schnelleren Algorithmus verbraucht weniger Prozessorzeit und somit weniger System-Ressourcen als das alte Programm mit dem langsameren Algorithmus. Durch die Transformation ist die Performanz des Programms erhöht worden, die Nutzung der Ressource “Prozessor” ist somit effizienter. Solche Veränderungen von Programmen fallen unter den Oberbegriff der *Programm-Optimierung*. In dieser Arbeit wird die Veränderung der Effizienz durch Refactorings jedoch nicht weiter betrachtet. Refactoring von Software, deren Ressourcen begrenzt sind oder die garantierte Reaktionszeiten aufweisen muss, erfordert weitergehende Forschung, die über den Kontext der vorliegenden Arbeit hinausgeht.

³² dies schließt die Löscharkeit von Objekten ein

Schließlich muss angemerkt werden, dass der Aspekt der verbesserten Wartbarkeit – wie sehr er auch die Ursache für das Durchführen eines Refactorings maßgeblich ist – nur schwer formalisierbar und messbar ist. Denn zueinander komplementäre Transformationen können je nach Kontext durchaus beide die Wartbarkeit eines Programms erhöhen. Beispielsweise kann das Hinzufügen einer Abstraktion die Erweiterung der Anwendung um eine neue Funktion ermöglichen; im Gegenzug kann das Löschen eben dieser Abstraktion – sofern sie nicht mehr gebraucht wird – die Lesbarkeit und Verständlichkeit des Programms verbessern. Das Beispiel verdeutlicht, dass globale und vor allem messbare Kriterien für die Wartbarkeit eines Programms bzw. im vorliegenden Fall für die Veränderung der Wartbarkeit durch eine Programm-Transformation schwierig und ohne Berücksichtigung des Kontextes sogar unmöglich sind. Für eine allgemeine Definition von Refactoring im Kontext dieser Arbeit ist es somit günstig, auf die Forderung nach verbesserter Wartbarkeit durch ein Refactoring zu verzichten.

4.5.2 Datenbank-Refactorings

Analog zu Software-Refactorings sind Datenbank-Refactorings Transformationen von Datenbanken mit bestimmten positiven Eigenschaften. Während bei Software-Refactorings gefordert wird, dass die Semantik des zu transformierenden Programms erhalten bleibt, ist es bei Datenbank-Refactorings entscheidend, dass der Informationsgehalt der Datenbank nicht reduziert wird.

[3] definiert ein Datenbank-Refactoring als “a simple change to a database schema that improves its design while retaining both its behavioral and informational semantics” [3, Kapitel 2], [4, S. 39]. Der erste Aspekt, das Beibehalten der Verhaltenssemantik, wird in [55] genauer unter die Lupe genommen (siehe letzte Aufzählung). Für Datenbank-Refactorings sind dabei die ersten beiden Regeln, die Erhaltung von Lese- und Schreib-Zugriffen auf Objekte, wesentlich, da diese Regeln genau die Operationen abdecken, die auf Datenbanken möglich sind. Zugriffe auf Objekte in objektorientierten Systemen geschehen letztlich immer über entsprechende Verknüpfungen zwischen Objekten: Nur wenn ein Objekt ein anderes über eine (eventuell längere) Kette von Verknüpfungen erreichen kann, ist ein lesender oder schreibender Zugriff auf dessen Informationen möglich. Solche Verknüpfungen über beliebig viele Zwischen-Objekte hinweg werden *Pfade* genannt.³³ Mit anderen Worten erhält ein Refactoring genau dann die Semantik bzgl. Lese- und Schreib-Zugriffen auf die Datenbank, wenn Pfade in der ursprünglichen Datenbank auf Pfade in der transformierten Datenbank typverträglich abgebildet werden. Dabei müssen die *Pfad-Ausdrücke*, welche die Pfade mit Bezug aufs Schema beschreiben, eventuell angepasst werden [45, Kapitel 4].

Der zweite Aspekt, das Beibehalten der Informations-Semantik, wird in [3] nicht formal definiert, jedoch umschrieben als die Eigenschaft des Refactorings, den Informationsgehalt der Daten soweit beizubehalten, dass Klienten mit den neuen Daten ebenso gut arbeiten können wie mit den alten.

³³ Pfade spielen bei Abfragen in objektorientierten Datenbanken eine wichtige Rolle, vgl. [55, Abschnitt 4.1] und [45, Abschnitt 2.1].

Intuitiv dürfen durch ein Refactoring keine Informationen verloren gehen, die für Klienten wesentlich sind. Dies schließt aber eine geeignete Anpassung von Klienten an das neue Daten-Format nicht aus.³⁴ Eine Möglichkeit, das Beibehalten der Informations-Semantik zu formalisieren, wird in [57–59] vorgestellt: Diese Arbeiten prägen den Begriff der *Informationskapazität* als Maß aller möglichen Ausprägungen zu einem Schema und untersuchen Schema-Transformationen, welche die Informationskapazität nicht verringern. Solche Schema-Transformationen wären ideale Kandidaten für die formale Definition von Refactorings. Im Rahmen dieser Arbeit wird dieser Ansatz jedoch nicht aufgegriffen. Abschnitt 14.2.6 gibt einen Ausblick, wie das in dieser Arbeit beschriebene mathematische Modell erweitert werden muss, damit Refactorings die Informationskapazität erhalten.

4.5.3 Erweiterungen

Zu der Frage, ob *Erweiterungen* des Schemas Refactorings sind, gibt es in der Literatur unterschiedliche Ansichten. Offenkundig sind Erweiterungen immer informationserhaltend,³⁵ allerdings wird dies nicht immer als ausreichendes Kriterium für Refactorings gesehen. Während in [60] das Erstellen von neuen Programm-Elementen als Refactoring angesehen wird [60, Abschnitt 5.1], werden in [3] Datenbank-Erweiterungen nicht als Refactorings angesehen. Dort wird argumentiert, dass Erweiterungen den vorhandenen Datenbank-Entwurf *verändern* und nicht *verbessern*, was als wesentliches Kriterium für Refactorings gesehen wird [3, Kapitel 1].

Auf der einen Seite verbessern Erweiterungen sicherlich nicht die Wartbarkeit eines Programms oder einer Datenbank. Auf der anderen Seite verändern Schema-Erweiterungen in der Regel *nicht* die Funktionalität bestehender Programme oder die Informationsmenge bestehender Datenbanken. Deshalb werden Erweiterungen um Klassen, Assoziationen und Attribute in dieser Arbeit zur Klasse der Refactorings gezählt.

Eine Ausnahme bildet im Rahmen des Software-Refactorings das Erzeugen neuer Operationen, denn dies erfordert das Erstellen (“Programmieren”) entsprechender Methoden. Automatisches Erstellen von Methoden zu einer Operation führt jedoch über die Zielsetzung dieser Arbeit hinaus. In dieser Arbeit wird angenommen, dass Methoden nicht automatisiert entwickelt werden. Somit werden Erweiterungen um neue Operationen als Entwicklungen (4.6) verstanden, die eine zusätzliche nicht-triviale Interaktion mit der zu verändernden Software erfordern.³⁶

³⁴ “You must rework any source code that works with changed aspects of your database schema to maintain the existing functionality.” [4, S. 39]

³⁵ Nach [57] kann jeder Ausprägung zum alten Schema eine eindeutige Ausprägung zum erweiterten Schema zugeordnet werden, indem beispielsweise für die fehlenden Daten Default-Werte angenommen werden.

³⁶ Ein alternativer Ansatz ist, neue Operationen automatisch durch Methoden zu implementieren, die auf alle Eingaben mit einem Fehler (z. B. mit einem *Null*-Wert, einer Ausnahme o. ä.) reagieren.

4.5.4 Löschungen

Analog zu der Frage, ob Schema-Erweiterungen Refactorings sind, kann die Fragestellung diskutiert werden, ob das *Löschen* von Schema-Elementen ein Refactoring darstellt. [60] konzentriert sich allein auf den funktionalen Aspekt von Programm-Refactorings und behandelt Löschungen folglich als Refactorings genau dann, wenn die zu löschenden Programm-Elemente nicht von anderen Elementen referenziert werden. In [3] hingegen werden Datenbank-Schema-Löschungen aller Art als Refactorings angesehen. Allerdings sind Löschungen offensichtlich keine Umgestaltungen existierender Elemente im alten Schema. Insbesondere verändert sich durch das Löschen von Schema-Elementen sowohl die Funktionalität bestehender Programme als auch die Informationsmenge bestehender Datenbanken, sofern die gelöschten Schema-Elemente ausgeprägt sind. Aus diesem Grund werden Löschungen in dieser Arbeit nicht zu den Refactorings, sondern zur Gruppe der Entwicklungen (4.6) gerechnet.

4.5.5 Definitionen

Die Überlegungen der vorigen Abschnitte führen zu den folgenden Definitionen:

Definition 4.8 (Software-Refactoring). *Ein Software-Refactoring transformiert ein Programm in ein anderes Programm dergestalt, dass dessen beobachtbares Verhalten nicht verändert wird. Dabei wird unter beobachtbarem Verhalten der funktionale Aspekt des Programms verstanden, also die Reaktion einer Programm-Ausprägung auf Eingabedaten durch Produktion zugehöriger Ausgabedaten. Eingabe- und Ausgabedaten sind hierbei nicht auf Benutzer-Interaktion beschränkt, sondern erfassen auch den beobachtbaren Zustand der Umgebung, innerhalb derer das Programm ausgeführt wird.³⁷ Zu dem transformierten Programm existierende Prozesse werden in Prozesse transformiert, die äquivalent bezüglich Verhaltenssemantik sind.*

Definition 4.9 (Datenbank-Refactoring). *Ein Datenbank-Refactoring transformiert eine Datenbank in eine andere Datenbank dergestalt, dass die Fähigkeit zur Modifikation und Löschung von Objekten und zur Navigation zwischen Objekten erhalten bleibt.*

Definition 4.10 (System-Refactoring). *Ein System-Refactoring transformiert ein System in ein anderes System durch Anwendung entsprechender Software- und Datenbank-Refactorings auf die Software und die Datenbanken, aus denen das System besteht.*

Zu beachten ist, dass diese Definitionen nicht fordern, dass ein Datenbank-Refactoring eine *atomare* oder *elementare*, nicht mehr weiter unterteilbare Operation darstellt. Das liegt daran, dass die vorliegende Arbeit die Komposition beliebiger Refactorings untersucht, wobei es unwesentlich ist, ob die komponierten Refactorings atomar sind oder nicht. Dies entspricht auch der kategoriellen

³⁷ etwa Inhalte von Dateien, Kommunikation über ein Netzwerk etc.

Sichtweise auf die Hintereinanderschaltung von Morphismen, bei der es ebenfalls unwichtig ist, ob die betrachteten Morphismen selbst Kompositionen anderer Morphismen sind oder nicht. Nichtsdestoweniger ist die Analyse der Fragestellung, ob eine kleine oder gar kleinste Menge von Refactorings existiert, aus der alle anderen Refactorings konstruiert werden können, sicherlich interessant und wichtig und Gegenstand künftiger Forschung.

4.6 Entwicklungen

Gelegentlich ist es nützlich, Transformationen zu betrachten, die keine Refactorings sind. In dieser Arbeit wird dafür der Begriff *Entwicklung* benutzt, um anzudeuten, dass durch die Transformation eine (Weiter-)Entwicklung stattfindet, die nicht allein eine Umstrukturierung darstellt. Typische Beispiele sind das Erzeugen neuer Methoden und das Löschen bestehender Objekte.

Definition 4.11 (Entwicklung). *Eine Transformation wird Entwicklung genannt, wenn sie kein Refactoring ist. Sämtliche Transformationen eines bestimmten Elements sind somit entweder Entwicklungen oder Refactorings.*

4.7 Induzierte Migrationen

Transformationen von Datenbanken, Software und Systemen sind *ganzheitlich*, da sie sich sowohl auf die Schema- als auch auf die Ausprägungs-Ebene beziehen. Ein typisches Beispiel ist das Löschen einer Klasse im Datenbank-Schema, das die Löschung aller zu der Klasse gehörenden Objekte impliziert. Ein Gegenbeispiel ist das Erstellen einer neuen Klasse im Schema: Hier muss keine Veränderung der Ausprägungs-Ebene durchgeführt werden, um eine konsistente Welt zu erhalten.³⁸

Im Mittelpunkt der vorliegenden Arbeit stehen Transformationen von Welten, die allein durch die Transformation des Schemas vollständig bestimmt sind, bei denen also die Ausprägungs-Transformation sich automatisch aus der Schema-Transformation ableitet. Solche Transformationen sind vor allem aus zwei Gründen interessant:

- Schema-Transformationen lassen sich einfacher beschreiben als Welt-Transformationen, weil sie sich nur auf das Schema beziehen und Schemata in der Regel wesentlich kleiner als die zugehörigen Ausprägungen sind.
- Schema-Transformationen lassen sich einfacher verstehen als Welt-Transformationen, weil sie nur diejenige Struktur modifizieren, die das fachliche Modell, die Problemdomäne der jeweiligen Anwendung, abbildet.

³⁸ Das gilt natürlich nur, wenn in dem verwendeten Modell die Existenz einer Klasse nicht die Existenz eines ausgezeichneten *Default-Objektes* auf der Ausprägungs-Ebene nach sich zieht.

Schema-Transformationen haben jedoch auch Grenzen. Insbesondere bei der Spezialisierung von Klassen können sie nicht alle nützlichen Szenarien darstellen. Ein Beispiel ist die Ersetzung eines Attributs *istMännlich* einer Klasse *Person* vom Typ *Bool*³⁹ durch zwei Unterklassen *Mann* und *Frau*. Dies lässt sich nicht allein auf dem Schema formulieren. Denn die Geschlechter “männlich” und “weiblich” sind nur auf der Ausprägungs-Ebene durch Verknüpfungen zu verschiedenen Ziel-Objekten zu unterscheiden.

³⁹ Der primitive Datentyp *Bool* repräsentiert Wahrheitswerte, die einzigen Werte zu diesem Datentyp sind *true* (wahr) und *false* (falsch); vgl. 5.1.3.

Objektorientierte Systeme

Dieses Kapitel stellt ein einheitliches Modell für objektorientierte Systeme, bestehend aus Daten, Programmen und Prozessen, vor. Dieses Modell wird *konzeptionelles* Modell genannt, weil seine Beschreibung ohne mathematische Formalismen auskommt.

Abschnitt 5.1 befasst sich mit dem Aufbau von *Datenbanken*, sowohl auf Schema- als auch auf Ausprägungs-Ebene. Es wird dargestellt, wie Klassen und Objekte, primitive Datentypen und Werte, Assoziationen und Verknüpfungen sowie Attribute und Attributwerte modelliert werden und welche Eigenschaften dieser Elemente unterstützt werden.

In Abschnitt 5.2 geht es um die Darstellung von *Software*. Auf der Schema-Ebene umfasst dies die Modellierung von Operationen und ihren Parametern. Auf der Ausprägungs-Ebene schließt dies sowohl *Programme* als auch *Prozesse* zu diesen Programmen ein.

Prozesse bestehen aus Nachrichten und zugehörigen Argumenten, wobei jede Nachricht in einer Operation und jedes Argument einer Nachricht in einem Parameter der zur Nachricht gehörenden Operation typisiert ist. Eine Nachricht kann *aktiv*, d. h. für die unmittelbar anstehende Verarbeitung markiert sein. Der Fortgang eines Prozesses wird modelliert, indem eine aktive Nachricht durch die Ausführung der zu ihrer Operation gehörenden *Methode* (s. u.) *verarbeitet* wird. Dabei werden die aktive Nachricht deaktiviert und neue, noch zu verarbeitende Nachrichten erzeugt, von denen die gemäß der Ausführungsreihenfolge nächste Nachricht aktiviert wird. Ein Prozess ist zu Ende, wenn keine Nachricht mehr aktiv ist.

Programme sind zusammengesetzt aus einzelnen *Methoden*. Jede Methode gehört zu einer Operation und legt eine Prozess-Transformation fest. Diese Transformation beschreibt die Veränderung einer Ausgangssituation im Prozess, die aus einer aktiven Nachricht, ihren Argumenten und eventuell weiterem Kontext besteht, in eine Zielsituation im Prozess, in der diese Nachricht verarbeitet ist. Allgemein werden durch die Verarbeitung der Prozess und/oder die zugrunde liegenden Daten in der Datenbank verändert.⁴⁰ Die konkrete Verarbeitung einer Nachricht ist dabei abhängig von der Semantik der zugrunde liegenden Operation; typischerweise werden Ver-

⁴⁰ Das Schema bleibt während der Abarbeitung von Programmen unangetastet.

knüpfungen “umgehängt” (etwa bei Variablenzuweisungen, Objektmodifikationen und -selektionen oder Typanpassungen) oder Werte berechnet (etwa bei arithmetischen Rechenoperationen, bei Zeichenketten-Manipulationen oder bei Vergleichen).

Anmerkung 5.1 (Notation). Die Darstellung von Klassen im konzeptionellen Modell erfolgt mit Hilfe der UML-Notation [29, 64]. Die Darstellung von Objekten ist an die UML angelehnt, weicht im Detail jedoch ab, um die Partikelstruktur von Objekten (Abschnitt 5.1.1) zu verdeutlichen. Der Darstellung von Operationen und Nachrichten sowie von Parametern und Argumenten liegt eine Erweiterung der UML-Notation zu Grunde, die in Abschnitt 5.2 im Detail beschrieben wird. □

5.1 Datenbanken

In diesem Abschnitt werden die Daten-orientierten Konstrukte des konzeptionellen Modells beschrieben. Die Eigenschaften der grundlegenden Elemente Klasse und Objekt (5.1.1), Assoziation und Verknüpfung (5.1.2) sowie primitiver Datentyp und Wert (5.1.3) werden erläutert. Weiter werden die besonderen Verweisklassen präsentiert (5.1.4), die für die Anbindung der Datenbank an die Software-Schicht unentbehrlich sind. Schließlich werden die Struktur und die Funktionen der speziellen *Null*-Klasse und der zugehörigen *Null*-Objekte beschrieben (5.1.5).

5.1.1 Klassen und Objekte

Das Modell unterstützt die Abbildung von Klassen mit endlicher und zyklensfreier Vererbungsstruktur. Es wird nicht zwischen konkreten und abstrakten Klassen unterschieden. Klassen werden als rechteckige Kästen dargestellt. Abb. 5.1 zeigt eine kleine Vererbungshierarchie mit einer Klasse *Person* und zwei direkten Unterklassen *Mann* und *Frau*.

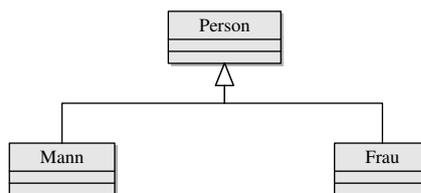


Abb. 5.1: Darstellung von Klassen und Vererbung

Objekte reflektieren Ausschnitte der Klassen-Hierarchie auf der Ausprägungs-Ebene. Im Gegensatz zu der üblichen Sichtweise, in der ein Objekt immer “aus einem Guss” ist, auch wenn dessen Klasse über Spezialisierungsbeziehungen mehrere Typen umfasst, werden Objekte als *Komposition* von einzelnen Objekt-Teilen modelliert, die im Folgenden (*Objekt-)*Partikel genannt werden. Dabei gehört jedes einzelne Partikel zu genau einer Klasse im Schema. Die Vererbungsbeziehungen zwischen den Klassen werden als so genannte *Vererbungsverknüpfungen* ausgeprägt und erlauben

auf der Ausprägungs-Ebene die Navigation zwischen den einzelnen Partikeln und somit letztlich den Zugriff auf geerbte Verknüpfungen.⁴¹ Ein Objekt zu einer Klasse C besitzt genau ein Partikel zu der Klasse C sowie zu jeder direkten und indirekten Oberklasse von C . Dadurch wird garantiert, dass die Klassen-Hierarchie, beginnend bei der Klasse des Objekts und sich zu generelleren Klassen ausbreitend, komplett für jedes Objekt in die Ausprägungs-Ebene reflektiert wird und somit alle Assoziationen typgerecht ausgeprägt werden können.

Abb. 5.2 zeigt eine mögliche Ausprägung zu dem Schema in Abb. 5.1. Es enthält zwei Objekte *mann:Mann* und *frau:Frau*, die beide jeweils aus zwei einzelnen Partikeln zu den Klassen *Mann* und *Person* (Objekt *mann*) bzw. *Frau* und *Person* (Objekt *frau*) bestehen.

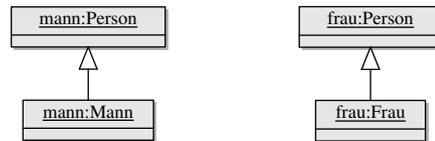


Abb. 5.2: Darstellung von Objekten

Objekte besitzen über ihre Partikelstruktur hinaus keine separate Identität. Ein Objekt wird genau durch die über Vererbungsverknüpfungen verbundenen Partikel charakterisiert, weitere Kriterien existieren nicht. Werden im Schema Klassenstrukturen durch das Entfernen von Vererbungsbeziehungen aufgebrochen, so kann in der Ausprägung ein Objekt in zwei Objekte zerfallen, wenn dessen Partikelstruktur in zwei nicht über Vererbungsverknüpfungen verbundene Teile aufgeteilt wird (Abb. 5.3).

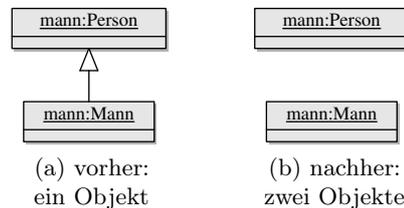


Abb. 5.3: “Aus eins mach zwei”: Aufspaltung von Objekten

⁴¹ Derselbe Ansatz wird in [44] gewählt, um Mehrfachklassifizierung, dynamische Reklassifizierung und Restrukturierung von Objekten zu ermöglichen; dort werden Partikel “implementation objects” (Implementationsobjekte) genannt, die zu einem “conceptual object” (konzeptionellen Objekt) zusammengefasst werden. Eine weitere Anwendung dieses Modells ist die Abbildung von Objekten in relationalen Datenbanken nach dem Muster “Class Table Inheritance” (eine Tabelle pro Klasse) [28, S. 285]. Schließlich werden Objekte in jenen Programmiersprachen in einzelne Partikel aufgeteilt, in denen Vererbung durch Implementierung von Schnittstellen und Delegation simuliert wird (etwa *Visual Basic 6*).

5.1.2 Assoziationen und Verknüpfungen

Eine *Assoziation* ist eine Beziehung einer Klasse zu einem anderen Schema-Element. Sie wird als ein durchgezogener Pfeil dargestellt. Es werden binäre gerichtete Assoziationen zu Klassen, primitiven Datentypen (5.1.3) und Operationen (5.2.1.3) unterstützt (Abb. 5.4).⁴² Assoziationen zu Klassen und primitiven Datentypen sind vom Typ (0..*, 1). Assoziationen zu Operationen sind vom Typ (0..*, 0..1). Alle anderen Typen von Assoziationen (etwa Assoziationen mit mehr als zwei involvierten Klassen, qualifizierte Assoziationen, mehrwertige Assoziationen, Aggregationen und Kompositionen) können nicht direkt abgebildet werden. Auf Grund der vorausgesetzten Multiplizitäten werden sie im Folgenden nicht mehr explizit notiert. Optionale Assoziationen zu Klassen, deren Ziel auf Ausprägungs-Ebene also nicht existieren muss, können simuliert werden, indem eine Verknüpfung zu einem *Null*-Objekt (5.1.5) angelegt wird. Mehrwertige Assoziationen vom Typ (0..*,0..*) können durch listenartige Strukturen simuliert werden, die nur aus (0..*,1)-Assoziationen bestehen (Abb. 5.5).

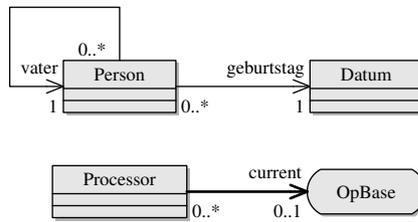


Abb. 5.4: Darstellung von einwertigen Assoziationen

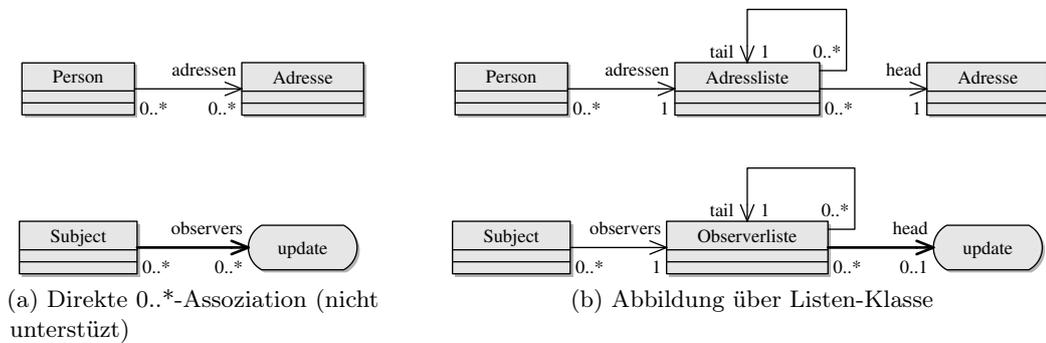


Abb. 5.5: Darstellung von mehrwertigen Assoziationen

In der Regel verweisen Assoziationen auf Klassen. Assoziationen zu Operationen können verwendet werden, um eine Brücke zwischen Daten und Software zu schlagen. Dies wird genutzt, um innerhalb eines Prozesses die zur Verarbeitung anstehenden Nachrichten zu markieren; dies wird in Abschnitt 5.2.1.4 näher erläutert. Assoziationen zu primitiven Datentypen werden verwendet, um primitive

⁴² Das Element *OpBase* ist eine Operation und wird in Abschnitt 5.2.1.3 beschrieben.

Datentypen an Hüllenklassen anzubinden (5.1.3) und um zwischen dynamisch und statisch gebundenen Nachrichten zu unterscheiden (5.2.1.4). Assoziationen zu Hüllenklassen werden *Attribute* genannt.

Verknüpfungen werden schließlich als Verbindung zwischen Objekt-Partikeln, zwischen einem Objekt-Partikel und einem Nachrichten-Partikel oder zwischen einem Objekt-Partikel und einem Wert dargestellt. Verknüpfungen, die in Attributen typisiert sind, werden *Attributwerte* genannt. Abbildung 5.6 zeigt eine beispielhafte Ausprägung zu dem Schema in Abb. 5.4.

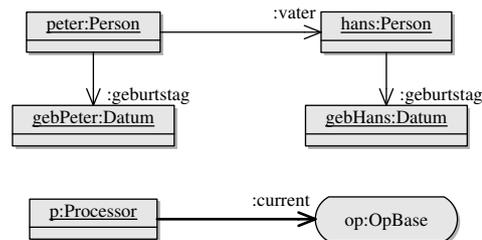


Abb. 5.6: Darstellung von Verknüpfungen

5.1.3 Primitive Datentypen und Werte

Primitive Datentypen sind wie Klassen Schema-Elemente, die eine Menge von Ausprägungen (Werte genannt, s. u.) besitzen. Sie sind jedoch keine Klassen im eigentlichen Sinn: Primitive Datentypen können nicht andere Klassen oder primitive Datentypen spezialisieren und dürfen weder ausgehende Assoziationen noch Attribute besitzen.⁴³ Auch können sie nicht direkt den Typ eines Parameters einer Operation (5.2.1.3) bilden. Dieser Unterschied wird deutlich gemacht, indem primitive Datentypen im Gegensatz zu Klassen durch Kästen mit leicht abgerundeten Ecken dargestellt werden. Die folgenden primitiven Datentypen werden unterstützt: *Bool* für Wahrheitswerte, *Integer* für ganze Zahlen, *Character* für einzelne Zeichen und *String* für Zeichenketten.

Um primitive Datentypen besser in das restliche objektorientierte Modell zu integrieren, existiert zu jedem primitiven Datentyp *Type* eine gleichnamige Klasse. Diese Klasse besitzt eine Assoziation namens *value* zu dem primitiven Datentyp und fungiert als Hülle um den primitiven Datentyp. Aus diesem Grund werden diese Klassen *Hüllenklassen* genannt. Durch die zwischengeschalteten Hüllenklassen ist es nun möglich, primitive Datentypen wie Klassen zu behandeln und insbesondere Operationen auf ihnen zu definieren (5.2.1.3).

In Abb. 5.7 wurde Abb. 5.1 um dem primitiven Datentyp *String* und die zugehörige Hüllenklasse *String* erweitert.

Ausprägungen eines primitiven Datentyps werden *Werte* genannt. Analog zum Schema werden Werte nicht direkt verwendet, sondern von einem Objekt der entsprechenden Hüllenklasse,

⁴³ Eingehende Assoziationen sind hingegen erlaubt.

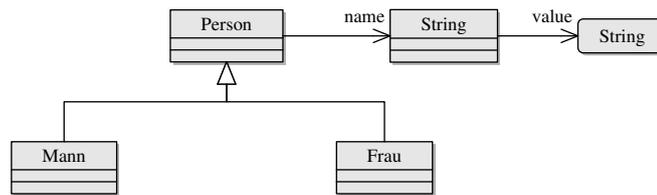


Abb. 5.7: Darstellung von primitiven Datentypen

Hüllenobjekt genannt, gekapselt. Die Verbindung zwischen Hüllenobjekt und Wert geschieht über eine Verknüpfung zur Assoziation *value*. Im Gegensatz zu Objekten fallen Inhalt und Identität bei Werten zusammen: Zwei gleiche Werte sind somit ununterscheidbar. Deswegen ist es erlaubt, von *der* Zahl 5 statt von *einer* Zahl 5 zu sprechen.⁴⁴

Abbildung 5.8 stellt zwei Zeichenketten-Werte dar, die jeweils über ein zwischengeschaltetes Hüllenobjekt mit einem Partikel der Klasse *Person* verbunden sind.

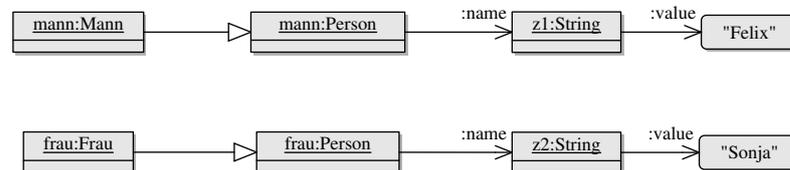


Abb. 5.8: Darstellung von Werten

5.1.4 Verweise

Jeder benutzerdefinierten Klasse X wird im Modell eine zweite Klasse $\uparrow X$ an die Seite gestellt, eine so genannte *Verweisklasse*.⁴⁵ Sie besitzt eine Assoziation namens *ref*⁴⁶ zur Klasse X und repräsentiert *Verweise* auf Objekte der Klasse X (Abb. 5.9).⁴⁷ Alle Verweisklassen spezialisieren die vordefinierte Verweisklasse \uparrow , deren Bedeutung in Abschnitt 5.2.1.3 erläutert wird. Objekte zu einer Verweisklasse werden *Verweisobjekte* genannt.⁴⁸

Verweisklassen werden eingeführt, um in der Software-Schicht Variablen abbilden zu können. Die von Verweisklassen ausgehende *ref*-Assoziation ist genau diejenige Indirektion, die im Software-Modell benötigt wird, um die Inhalte von Variablen durch das “Umbiegen” von Verknüpfungen verändern zu können. Mehr hierzu ist in den Abschnitten 5.2.1.1 und 5.2.2.1 zu finden.

⁴⁴ Gelegentlich werden in Diagrammen gleiche Werte mehrfach dargestellt; dies dient lediglich der Übersicht und soll nicht suggerieren, dass mehrere Werte mit gleichem Inhalt existieren können.

⁴⁵ Eine solche Verweisklasse wird auch für die Klasse *Null* (siehe Abschnitt 5.1.5), jedoch nicht für Hüllen- und andere Verweisklassen erzeugt.

⁴⁶ Abkürzung für “reference” (dt. “Verweis”)

⁴⁷ In Diagrammen wird aus technischen Gründen anstelle des Pfeils der Zirkumflex benutzt.

⁴⁸ Verweisobjekte werden in diversen Programmiersprachen häufig mit den Begriffen *Zeiger* (z. B. in *C++*) oder *Referenz* (z. B. in *Java*) bezeichnet.

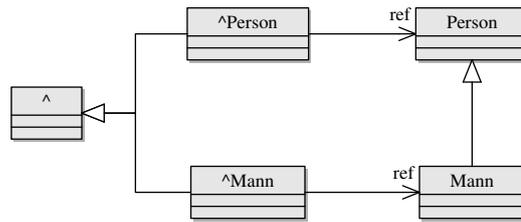
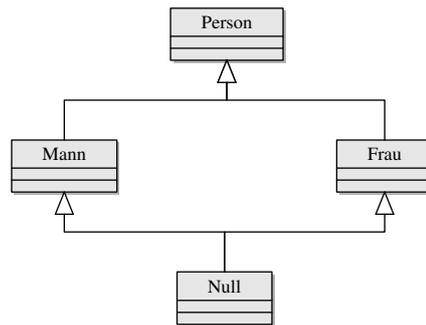


Abb. 5.9: Darstellung von Verweisklassen

5.1.5 Die *Null*-Klasse und *Null*-Objekte

Die Klassenhierarchie aller benutzerdefinierten Klassen⁴⁹ ist jeweils nach unten hin durch eine spezielle Klasse, *Null* genannt, abgeschlossen. Objekte zu dieser Klasse erfüllen unterschiedliche Aufgaben, unter anderem können sie Platzhalter zur Anzeige fehlender Informationen oder Fehlerobjekte zur Darstellung von während der Programmausführung entstehenden Fehlern sein. Abb. 5.10 zeigt die Klassenhierarchie aus Abb. 5.1 zusammen mit der die Hierarchie abschließenden *Null*-Klasse. Im Folgenden wird die *Null*-Klasse in den Klassendiagrammen aus Gründen der Übersichtlichkeit nicht mehr explizit aufgeführt.

Abb. 5.10: *Null*-Klasse als Abschluss der Vererbungshierarchie

Ein *Null*-Objekt reflektiert die Klassenhierarchie des Schemas auf der Ebene der Ausprägung. Gemäß der Struktur von Objekten besitzt ein *Null*-Objekt zu der Klasse *Null* und zu jeder direkten und indirekten Oberklasse ein Partikel, einschließlich der zugehörigen Vererbungsverknüpfungen. Des Weiteren reflektiert ein *Null*-Objekt auch alle Assoziationen zu benutzerdefinierten Klassen. Sind zwei Klassen im Schema über eine Assoziation miteinander verbunden, so existiert in jedem *Null*-Objekt eine Verknüpfung zu dieser Assoziation zwischen den entsprechenden Partikeln. Schließlich reflektiert ein *Null*-Objekt auch alle Attribute. Besitzt eine Klasse ein Attribut, so wird dieses Attribut in jedem *Null*-Objekt durch einen Attributwert ausgeprägt, der über ein Hüllenobjekt auf einen Wert verweist. Dieser Wert ist implementierungsspezifisch belegt und

⁴⁹ Dies schließt Hüllen- und Verweisklassen aus.

repräsentiert einen Fehler- oder Initialwert zum jeweiligen primitiven Datentyp.⁵⁰ Abb. 5.11 zeigt ein mögliches *Null*-Objekt zu der Klassenhierarchie aus Abb. 5.7.

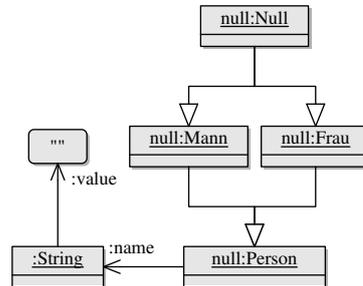


Abb. 5.11: *Null*-Objekt

Null-Objekte erfüllen zwei wichtige Funktionen. Zum einen stellen sie *Platzhalter* dar, wenn nicht genügend Informationen vorhanden sind, um ein vollständiges Objekt einer Klasse zu erzeugen. In dieser Funktion treten sie vor allem in der Initialisierung objektwertiger Variablen auf (5.2.1.1).⁵¹ Zum anderen können *Null*-Objekte zum Anzeigen einer Fehlersituation bei der Programmausführung verwendet werden. Weil *Null*-Objekte jedes andere Objekt zu einer benutzerdefinierten Klasse vertreten können, ist jede Methode, deren Rückgabetyt eine benutzerdefinierte Klasse ist, in der Lage, in einem Fehlerfall ein *Null*-Objekt an den Aufrufer zurückzugeben.⁵² Außerdem können Methoden auf Verknüpfungen und Attributwerte eines *Null*-Objekts ebenso zugreifen wie auf Verknüpfungen und Attributwerte eines Objekts zu einer benutzerdefinierten Klasse, weil jedes *Null*-Objekt Verknüpfungen und Attributwerte zu allen Assoziationen und Attributen genau einmal ausprägt. Der Unterschied ist lediglich, dass alle referenzierten Partikel wiederum Teil des *Null*-Objekts sind und somit das *Null*-Objekt nicht “verlassen” wird. Zusammen mit der Reimplementierung aller Operationen als *Null*-Objekte zurückliefernde Methoden wird folglich *strikte Fehlerfortpflanzung* im Empfänger-Objekt jeder Nachricht realisiert.⁵³

5.2 Software

Dieser Abschnitt beschreibt die Teile des konzeptionellen Modells, die für die Abbildung von Programmen und Prozessen benötigt werden. Der erste Abschnitt 5.2.1 erläutert die Abbildung der

⁵⁰ Diese Werte hängen von dem semantischen Modell der abzubildenden Programmiersprache ab. Übliche Initialwerte sind 0 für Ganzzahl-Variablen, die leere Zeichenkette für Zeichenketten-Variablen und *false* für Boole’sche Variablen.

⁵¹ *Null*-Objekte bzw. *Null*-Werte sind in Programmiersprachen und Datenbanken weit verbreitet und stehen oft für fehlende oder unvollständige Informationen.

⁵² Falls der Rückgabetyt eine Hüllklasse ist, wird ein Hüllenobjekt mit dem Initialwert des zugehörigen primitiven Datentyps (etwa *false* beim Typ *Bool*) zurückgeliefert.

⁵³ *Strikte Fehlerfortpflanzung* liegt vor, wenn ein Fehler nicht verarbeitet wird und somit aus dem System verschwindet, sondern unverändert an den Aufrufer weitergereicht wird. In diesem Fall bedeutet dies, dass alle Nachrichten an ein *Null*-Objekt immer ein *Null*-Objekt zum Ergebnis haben.

grundlegenden Konzepte Operation, Nachricht, Parameter, Argument, Konstante und Variable. Er stellt dar, wie Methoden durch das sinnvolle Verknüpfen einer Menge von Nachrichten entstehen, und beschreibt den Weg vom Erstellen über das Verschicken einer Nachricht bis hin zum Abarbeiten der zugehörigen Methode. In dem zweiten Abschnitt 5.2.2 wird an Beispielen erläutert, wie Anweisungen und Ausdrücke über Operationen und zugehörige Methoden formuliert werden.

5.2.1 Grundlegende Konzepte

Dieses Kapitel beschreibt die grundlegenden Teile, die für die Abbildung von Programmen und Prozessen benötigt werden. Es erläutert die Abbildung von Variablen (5.2.1.1), Konstanten (5.2.1.2), Operationen samt Parametern (5.2.1.3) und Nachrichten samt Argumenten (5.2.1.4). Weiter stellt es dar, wie Methoden durch das sinnvolle Verknüpfen einer Menge von Nachrichten entstehen (5.2.1.5) und wie Polymorphie formuliert wird (5.2.1.6). Schließlich beschreibt es den Weg vom Erstellen über das Verschicken einer Nachricht bis hin zum Abarbeiten der zugehörigen Methode (5.2.1.7). Alle höherwertigen Konstrukte werden auf diese grundlegenden Elemente zurückgeführt. Dies wird in den folgenden Abschnitten 5.2.2.1, 5.2.2.3 und 5.2.2.2 dargestellt.⁵⁴

5.2.1.1 Variablen

Jedes Verweis- oder Hüllenobjekt stellt eine *Variable* dar. *Objektwertige Variablen* entsprechen den Verweisobjekten und zeigen auf Partikel benutzerdefinierter Klassen. *Primitive Variablen* entsprechen den Hüllenobjekten und referenzieren Werte primitiver Datentypen. Analog werden Verweis- und Hüllenklassen zu der Menge der *Variablenklassen* zusammengefasst, wobei Verweis-klassen *objektwertige* und Hüllenklassen *primitive Variablenklassen* darstellen. Der *Typ* einer Variable ist immer eine Variablenklasse und entspricht der Klasse des jeweiligen Verweis- oder Hüllenobjekts.

Variablen bilden das zentrale Element der Datenverarbeitung. Alle Methoden erwarten Eingaben in Form von Variablen und geben Ergebnisse in Variablen zurück (5.2.1.3). Der Grund dafür ist die in Variablenklassen eingebaute Indirektion (*ref*- bzw. *value*-Assoziation): Sie erlaubt es, die Modifikation einer Variable über ein einfaches “Umhängen” einer Verknüpfung zu realisieren (5.2.2.1).

Alle Variablen werden initialisiert. Eine objektwertige Variable verweist anfangs auf ein *Null*-Objekt. Eine primitive Variable zeigt initial auf einen implementierungsspezifischen Wert, der dem Initialwert des primitiven Datentyps in jedem *Null*-Objekt entspricht (5.1.5).

⁵⁴ Insbesondere werden alle Kontrollstrukturen, die in Programmiersprachen typischerweise als Anweisungen kodiert werden, als Nachrichten zu vordefinierten Operationen verstanden (5.2.2.2). In dieser Hinsicht ähnelt das konzeptionelle Modell dem Modell der Programmiersprache *Smalltalk*: Dort werden Kontrollstrukturen wie Fallunterscheidungen oder Schleifen auch durch entsprechende Nachrichten dargestellt [5, Abschnitte 5.3.3 und 5.4.2].

5.2.1.2 Konstanten

Konstanten sind die unveränderlichen Werte eines primitiven Datentyps. In Programmen können Konstanten so jedoch nicht direkt verwendet werden. Stattdessen werden sie in entsprechend vorbelegten primitiven Variablen gekapselt (vgl. Abb. 5.8).

5.2.1.3 Operationen und Parameter

Eine *Operation* wird als ein Kasten mit abgerundeten Seiten dargestellt und kann eine beliebige Anzahl von *Parametern* besitzen. Operationen können wie Klassen in einer Vererbungsbeziehung stehen.⁵⁵ Die Darstellung der Vererbungsbeziehung geschieht analog zur Vererbung zwischen Klassen.

Jede Klasse, zu der Objekte existieren sollen, muss zu jeder ihrer Operationen eine Methode (5.2.1.5) besitzen, wobei diese nicht unbedingt von der Klasse selbst bereitgestellt werden muss, sondern auch von einer direkten oder indirekten Oberklasse geerbt werden kann. Fehlt zu einer Operation die entsprechende Methode, und wird in einem Prozess an ein Objekt einer solchen unvollständigen Klasse eine Nachricht zu dieser Operation versandt, so liegt ein Fehler vor (vgl. 5.2.1.7).

Jeder Pfeil, der von einer Operation ausgeht, stellt einen *Parameter* dar. Es sind einerseits Parameter zu Variablenklassen (*Daten-Parameter* genannt) und andererseits zu Operationen⁵⁶ (*Operations-Parameter* genannt) möglich.⁵⁷ Daten-Parameter werden je nach Art der zugeordneten Variablenklasse *objektwertig* oder *primitiv* genannt. Parameter werden zur verbesserten visuellen Unterscheidung in Diagrammen gestrichelt dargestellt. Operations-Parameter sowie Assoziationen zu Operationen werden zusätzlich etwas dicker gezeichnet.

Hat eine Operation Funktionscharakter und somit genau einen Rückgabewert, so wird der zuständige Ausgabeparameter *result*⁵⁸ genannt, so dass der Rückgabewert visuell von anderen Parametern unterschieden werden kann. Ein weiterer besonderer Parameter, der in Anlehnung an die Programmiersprachen *Java* und *C++* *this*⁵⁹ genannt wird, ordnet eine Operation einer bestimmten Klasse zu. Dieser Parameter ist insofern ausgezeichnet, als dass eine dynamische Bindung von Nachrichten an Methoden über diesen Parameter erfolgen kann (5.2.1.4). Der Parameter ist jedoch optional: Es ist ebenfalls möglich, *freie Operationen* ohne eine besondere

⁵⁵ Dies erlaubt, gemeinsame Eigenschaften von Operationen auszulagern und somit Redundanz zu vermeiden.

⁵⁶ Dies wird hauptsächlich für die Abbildung der Fallunterscheidung (5.2.2.2) und für die Kodierung der Abarbeitungsreihenfolge (s. u.) genutzt.

⁵⁷ Die Beschränkung auf Variablen bei der Ein- und Ausgabe von Daten ermöglicht eine einheitliche Behandlung von Objekten und Werten innerhalb einer Methode und hat zur Folge, dass die Übergabe von Argumenten an eine Methode grundsätzlich eine Übergabe als Referenz ist (*call by reference*).

⁵⁸ dt. "Ergebnis"

⁵⁹ dt. "dieses (Objekt)"

Bindung an eine bestimmte Klasse abzubilden. Insbesondere wird auch die Formulierung von Operationen erlaubt, die überhaupt keine Parameter besitzen.⁶⁰

Abbildung 5.12 zeigt zwei beispielhafte Operationen *gibName* und *gibVater* mit den Daten-Parametern *this* und *result*.

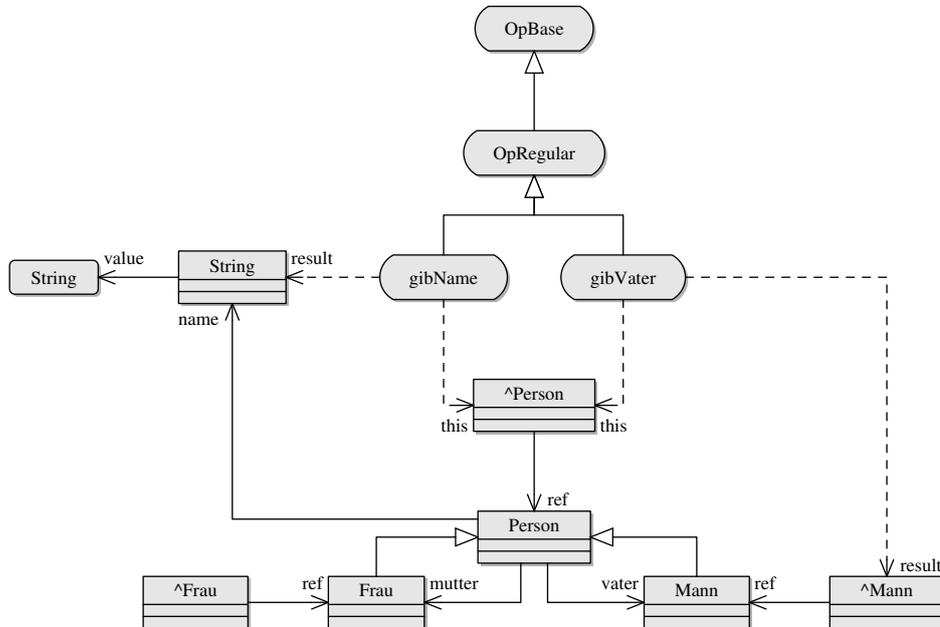


Abb. 5.12: Darstellung von Parametern

Zwei Operationen besitzen eine besondere Bedeutung. Jede Operation spezialisiert die vordefinierte Operation *OpBase*, die somit die allgemeinste Operation in der gesamten Operations-Hierarchie bildet. Diese Operation dient zum einen der Markierung der aktiven Nachricht (als Ausprägung einer Operation, s. nächsten Abschnitt) in einem Prozess. Die vordefinierte Klasse *Processor*⁶¹ repräsentiert eine verarbeitende Komponente, der zu jedem Zeitpunkt eine Nachricht zugeordnet ist. Diese Zuordnung geschieht im Schema über eine Assoziation *current*⁶², die von der Klasse *Processor* zu der Operation *OpBase* verläuft.

Zum anderen wird die Operation *OpBase* genutzt, um zwischen dynamischer und statischer Bindung einer Nachricht an eine Methode zu unterscheiden. Dazu besitzt die Operation *OpBase* einen Parameter namens *dynamic* vom Typ *Bool* sowie eine eingehende Assoziation *bound* von der allgemeinen Verweisklasse \uparrow . Der Wert des einer Nachricht zugeordneten *dynamic*-Arguments unterscheidet zwischen dynamischer und statischer Bindung: Ist dieser Wert *true*, so ist die Nachricht dynamisch gebunden. Andernfalls ist sie statisch gebunden; die gewählte Methode

⁶⁰ Dies wird für die Modellierung der leeren Anweisung und der Ende-Anweisung genutzt, vgl. [76, Abschnitt 3.3].

⁶¹ dt. "Prozessor, Verarbeiter"

⁶² dt. "aktuell"

wird über ein entsprechend typisiertes Verweisobjekt auf das Empfängerobjekt bestimmt. Die Bedeutung der beiden Bindungen und deren Umsetzung wird im nächsten Abschnitt detailliert erläutert.

Eine weitere vordefinierte Operation ist die Operation *OpRegular*. Diese Operation erbt von der Operation *OpBase* und enthält einen Operations-Parameter *next*⁶³ zur Operation *OpBase*, der auf Prozessebene genutzt wird, um die sequentielle Ordnung von Nachrichten auszudrücken. Der Parameter *next* geht nicht von der allgemeinen Operation *OpBase* aus, weil nicht jede Nachricht genau eine definierte Nachfolger-Operation hat.⁶⁴

Auf Prozessebene existiert zu jedem Ablaufplan eine Ausprägung der vordefinierten Klasse *Processor*. Das Modell legt die Anzahl der Prozessoren, die gleichzeitig in einem Prozess existieren können, nicht fest. Somit können sowohl Prozesse, die aus einem einzigen Ablaufplan bestehen, als auch nebenläufige Prozesse modelliert werden. Die Verarbeitung einer Nachricht ist in Abschnitt 5.2.1.7 beschrieben.

Abbildung 5.13 zeigt die beiden vordefinierten Operationen *OpBase* und *OpRegular* sowie die vordefinierte Klasse *Processor*.

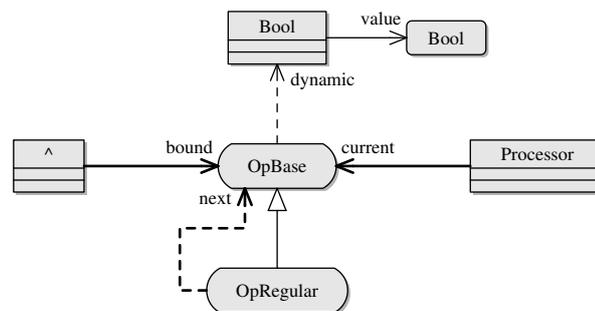


Abb. 5.13: Vordefinierte Operationen und assoziierte Klassen

5.2.1.4 Nachrichten und Argumente

Eine *Nachricht* ist die Anforderung an ein Objekt, eine spezielle Operation durchzuführen. Insofern sind Nachrichten und Operationen gleichermaßen strukturiert: Während eine Operation über Parameter die erwarteten Eingaben und zu liefernden Ausgaben spezifiziert, beinhaltet eine Nachricht über Argumente die tatsächlichen Eingaben sowie Objekte, welche die Ausgaben aufnehmen. Auf Grund dieser Korrespondenz werden Nachrichten im Modell als Ausprägungen der zugehörigen Operationen aufgefasst.

Spezialisiert die Operation einer Nachricht andere Operationen, so besteht die Nachricht analog zu Objekten aus mehreren Partikeln, die *Nachrichten-Partikel* genannt werden. Eine Nachricht

⁶³ dt. "nächste(r)"

⁶⁴ Die Fallunterscheidung (5.2.2.2) besitzt beispielsweise zwei mögliche Nachfolger-Operationen, von denen eine zur Laufzeit ausgewählt wird.

zu einer Operation O besitzt somit genau ein Partikel zu der Operation O sowie zu jeder direkten und indirekten Oberoperation von O . Weil die vordefinierte Operation $OpBase$ an der Spitze der Operations-Hierarchie steht, besitzt somit jede Nachricht ein Partikel zu dieser Operation. Dies ermöglicht für jede Nachricht das Festlegen der Bindung (dynamisch oder statisch, s. u.) sowie das Markieren der Nachricht als aktiv/inaktiv.

Die *Argumente* einer Nachricht sind Pfeile, die Variablen (bei Daten-Parametern) und Nachrichten (bei Operations-Parametern) an die Nachricht binden und die in den Parametern der zugehörigen Operation typisiert sind. Analog zu Parametern gibt es *Daten-Argumente*, die sich in *objektwertige* und *primitive* Argumente aufteilen, und *Nachrichten-Argumente*. Eine Nachricht muss zu jedem Parameter ihrer Operation ein entsprechend typisiertes Argument besitzen. Die Argumente werden typgerecht demjenigen Nachrichten-Partikel zugeordnet, zu dessen Operation der zum Argument gehörende Parameter gehört. Wie Parameter werden Argumente in Diagrammen gestrichelt dargestellt. Des Weiteren werden Nachrichten-Argumente und Verknüpfungen zu Nachrichten etwas dicker gezeichnet.

Eine Nachricht ist *aktiv*, wenn ein *Processor*-Objekt auf sie verweist, andernfalls ist sie *inaktiv*. Abbildung 5.14 zeigt ein Beispiel zu dem Schema in Abb. 5.12 mit einer aktiven und einer inaktiven Nachricht.

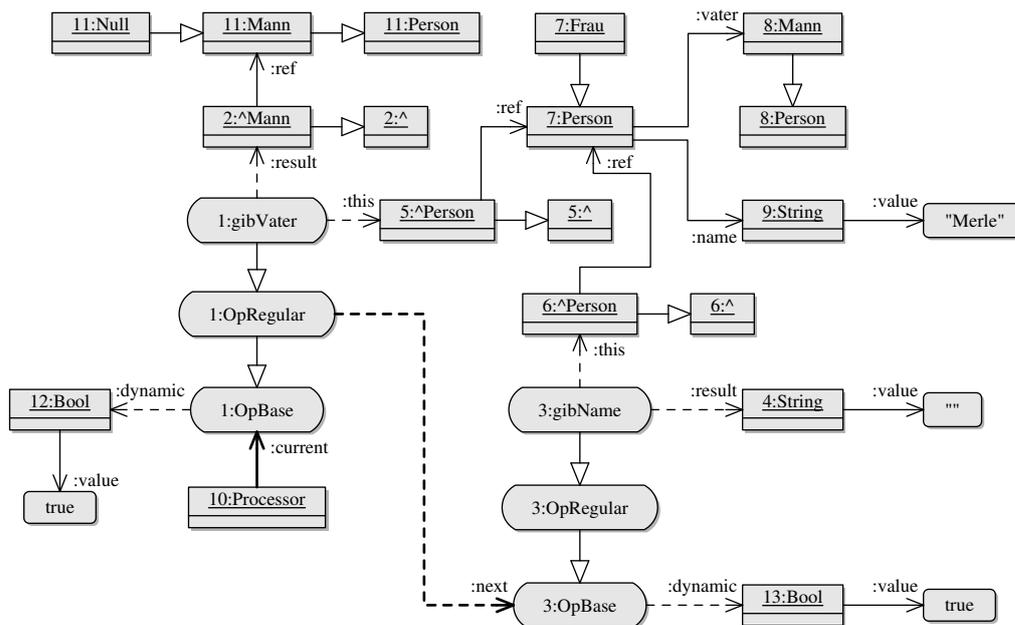


Abb. 5.14: Darstellung zweier dynamisch gebundener Nachrichten

Eine Nachricht, deren Operation einen *this*-Parameter enthält und somit einer bestimmten Klasse zugeordnet ist, kann *dynamisch* oder *statisch* gebunden werden. *Dynamisch* erfolgt eine Bindung, wenn die Klasse der aufzurufenden Methode nicht a priori bekannt ist oder bekannt sein

soll. Umgekehrt erfolgt eine Bindung *statisch*, wenn die Klasse der aufzurufenden Methode von vornherein bekannt ist. Die dynamische Bindung führt zur Laufzeit zu der Auswahl der Methode gemäß dem tatsächlichen Typ des Empfänger-Objekts und ermöglicht somit Polymorphie. Sie wird modelliert, indem der Nachricht über das *dynamic*-Argument der Boole'sche Wert *true* zugeordnet wird. Die in Abb. 5.14 dargestellten Nachrichten sind dynamisch gebunden. Im Folgenden werden dynamisch gebundene Nachrichten aus Gründen der Übersichtlichkeit ohne die erforderlichen, mit *true* belegten *dynamic*-Argumente dargestellt.

Beim statischen Binden einer Nachricht ist die aufzurufende Methode und deren Klasse bekannt.⁶⁵ Ein solcher Aufruf wird modelliert, indem zum einen ein Verweisobjekt ergänzt wird, das über die *ref*-Verknüpfung auf das Partikel zu derjenigen Klasse zeigt, welche die gewünschte Methode enthält, und das über die *bound*-Verknüpfung auf die statisch gebundene Nachricht verweist.⁶⁶ Zum anderen wird der Nachricht über das *dynamic*-Argument der Boole'sche Wert *false* zugeordnet. In Abb. 5.15 ist eine an den Typ *Person* statisch gebundene Nachricht dargestellt.

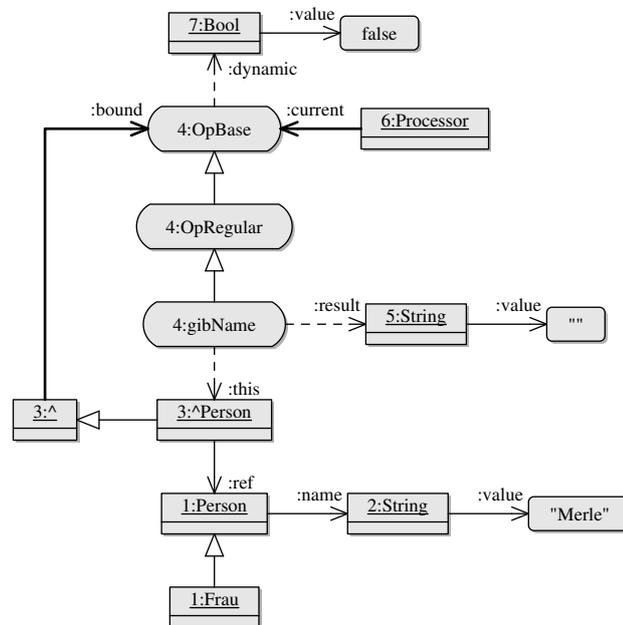


Abb. 5.15: Darstellung einer statisch gebundenen Nachricht

5.2.1.5 Methoden

Eine *Methode* implementiert eine Operation, indem sie die gewünschte Änderung des Datenbank- und Prozesszustands beschreibt. Diese Zustandsänderung wird über eine *Ersetzungsvorschrift*

⁶⁵ Beispiele statisch gebundener Nachrichten in Programmiersprachen umfassen den *super*-Aufruf in *Java* und den Aufruf einer Methode mit qualifiziertem Namen in *C++*.

⁶⁶ Entspricht der Typ des *this*-Parameters der Klasse der gewünschten Methode, kann der *this*-Parameter direkt für die statische Bindung wiederverwendet werden.

definiert: Eine Nachricht, die in der zur Methode gehörenden Operation typisiert ist, wird durch den Inhalt der Methode ersetzt, die wiederum aus einer Menge von untereinander verbundenen Nachrichten besteht.⁶⁷ Dabei wird vorausgesetzt, dass alle Daten-Argumente, die für die Ausführung der Methode benötigt werden, bereits ausgewertet sind. Durch diese Vorgehensweise wird gleichzeitig sowohl der Inhalt einer Methode als auch die Verarbeitung der Nachricht definiert, die zur Ausführung der Methode führt. Somit entspricht die Ersetzungsvorschrift einer Methode dem *Aufruf* dieser Methode. In Abb. 5.16 ist dargestellt, wie eine rekursive Funktion *fac* zur Ausrechnung der Fakultät einer natürlichen Zahl modelliert wird.⁶⁸ Aus Gründen der Übersichtlichkeit werden die *OpBase*- und *OpRegular*-Partikel sowie die Spezialisierungen jeweils in einem einzigen Knoten dargestellt.

5.2.1.6 Polymorphie

In objektorientierter Software ist es möglich, einer Operation mehr als eine Methode zuzuordnen, wobei zur Ausführungszeit je nach Typ des Empfänger-Objekts die jeweils passende Methode ausgewählt wird.⁶⁹ Das Modell ermöglicht dies, indem verschiedene Methoden, die Nachrichten zur selben Operation verarbeiten, unterschiedlich viel Kontext des Empfänger-Objekts spezifizieren. In Abb. 5.17 ist eine Operation *gibKlasse* dargestellt, die den Namen der Klasse des Objekts zurückliefert und somit für jede Klasse in der Vererbungshierarchie eine eigene Methode besitzt (in der Abbildung sind nur die beiden Methoden für die Klassen *Lebewesen* und *Mensch* dargestellt). Damit diese beiden Methoden auch korrekt verwendet werden, muss verhindert werden, dass die allgemeinere Methode ausgewählt wird, wenn eine speziellere Methode passt. Im Beispiel darf die Methode in der Klasse *Lebewesen* nicht zur Ausführung kommen, wenn die Nachricht zur Operation *gibKlasse* an ein Objekt der Klasse *Mensch* gesandt wird. Dies wird dadurch geregelt, dass auf den Methoden eine partielle Ordnung \leq definiert wird. Dabei gilt $m1 \leq m2$ für zwei Methoden $m1$ und $m2$, wenn die Typen der Argumente von $m1$ gemäß der Produkt-Ordnung spezieller als oder identisch zu den Typen der Argumente von $m2$ sind.⁷⁰ Soll eine Nachricht verarbeitet werden, wird die nach dieser Ordnung kleinste Methode gesucht. Existiert keine kleinste Methode, so liegt eine Mehrdeutigkeit vor (siehe hierzu Abschnitt 5.2.1.7).

5.2.1.7 Abarbeitung einer Nachricht

Die Verarbeitung einer aktiven Nachricht geschieht wie folgt:

⁶⁷ Genauer gesagt wird der Inhalt der Methode *hinter* der verarbeiteten Nachricht eingefügt. Dies erlaubt die mehrfache Verarbeitung einer Nachricht mit unterschiedlichen Argumenten, etwa in einer Schleife.

⁶⁸ Die Werte, die mit x und y bezeichnet werden, sind Platzhalter für tatsächliche Werte.

⁶⁹ Diese Beschränkung des Kontextes auf das Empfänger-Objekt wird *single dispatch* genannt. Einige, meist *LISP*-basierte Programmiersprachen wie *CLOS* (*Common LISP Object System*) oder *Dylan* unterstützen *multiple dispatch* und berücksichtigen die Typen *aller* Argumente.

⁷⁰ Dies ist der Fall, wenn gemäß der aus der Vererbungshierarchie resultierenden partiellen Ordnung auf Typen jeder Argument-Typ von $m1$ mindestens so speziell ist wie der entsprechende Argument-Typ von $m2$.

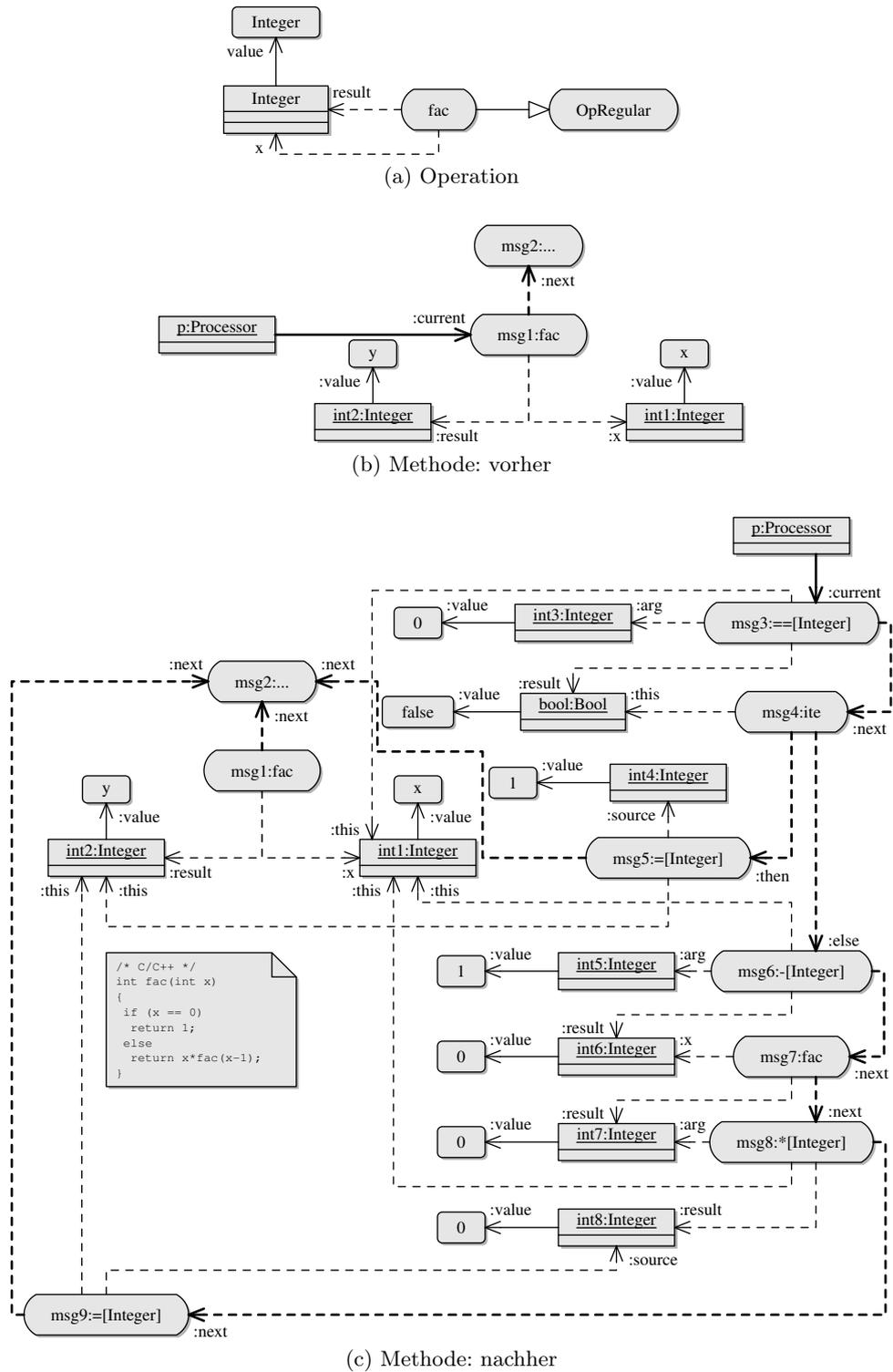
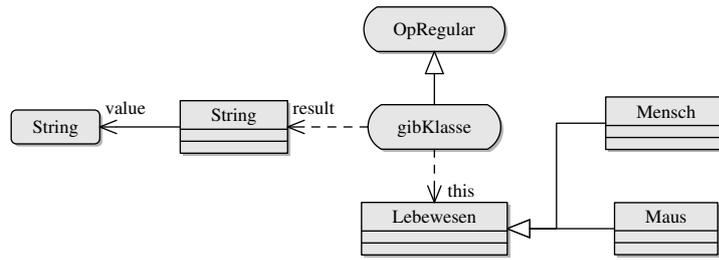
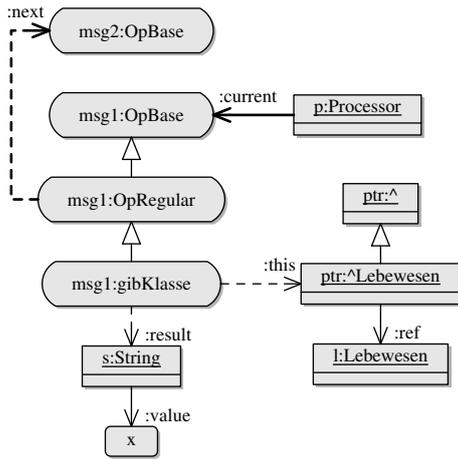


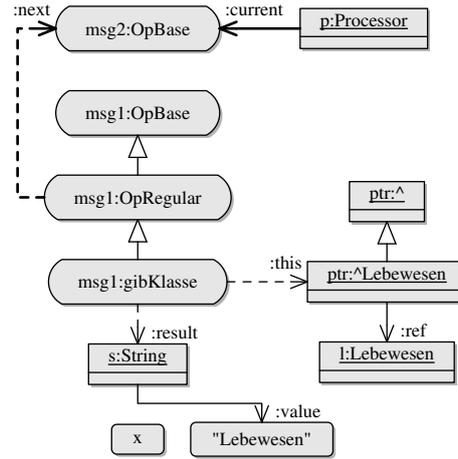
Abb. 5.16: Darstellung einer Methode am Beispiel der Fakultätsfunktion



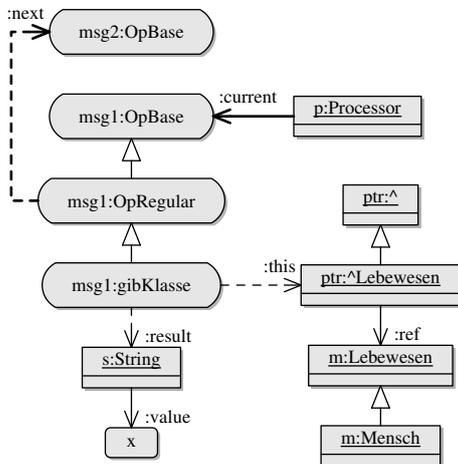
(a) Operation



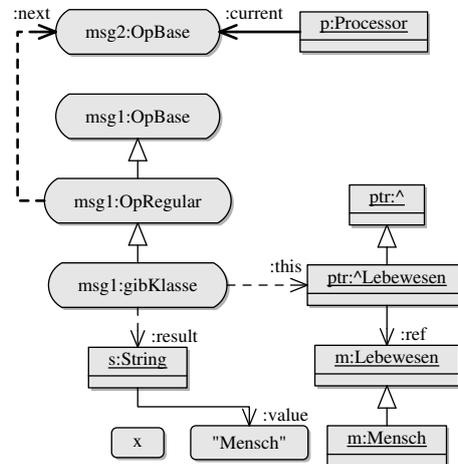
(b) Überschriebene Methode: vorher



(c) Überschriebene Methode: nachher



(d) Überschreibende Methode: vorher



(e) Überschreibende Methode: nachher

Abb. 5.17: Darstellung überschreibender und überschriebener Methoden

- (1) Hat die zugehörige Operation Daten-Parameter, werden zuerst die Daten-Argumente der Nachricht ausgewertet. In welcher Reihenfolge die Auswertung geschieht, wird vom Modell nicht vorgegeben und der Semantik der abgebildeten Programmiersprache entlehnt. Nachrichten-Argumente werden nicht ausgewertet und erlauben somit eine *verzögerte Auswertung*⁷¹ der Nachrichten, die vor allem für Fallunterscheidungen wesentlich ist.
- (2) Für jedes Argument, das als Wert übergeben werden soll, wird eine temporäre Variable erzeugt und mit dem zu übergebenden Objekt bzw. Wert initialisiert. Für Argumente, die als Referenz übergeben werden sollen, werden direkt die ursprünglichen Variablen verwendet, ohne temporäre Variablen zu erstellen.
- (3) Die Methode der von der Nachricht referenzierten Operation wird gesucht. Kommen mehrere Methoden in Frage, wird die kleinste Methode bezüglich der partiellen Ordnung auf den Methoden bestimmt. Gibt es keine passende Methode, oder existiert auf Grund mehrerer minimaler Methoden keine kleinste Methode⁷², liegt ein Programmfehler vor. In diesem Fall wird die Ausführung des Prozesses unterbrochen.
- (4) Die gefundene Methode wird in den Programmablauf integriert, indem der in der Methode durch miteinander interagierende Nachrichten beschriebene Ablauf in den aktuellen Prozess kopiert wird. Dabei wird der Kontext, der in der Methode spezifiziert wird, und der tatsächliche Kontext, der dem Prozess entstammt, identifiziert. Dadurch werden eine korrekte Argument-Übergabe sowie korrekte Rücksprünge zum Aufrufer gewährleistet.
- (5) Der Prozessor setzt die *current*-Verknüpfung auf diejenige Nachricht der Methoden-Kopie um, die als erstes bei der Ausführung der Methode verarbeitet werden soll, und macht diese Nachricht somit zur aktiven Nachricht.

Die Reihenfolge der einzelnen Schritte wird durch eine geeignete Verschaltung der Nachrichten sichergestellt. Beim Kodieren eines Methodenaufrufs muss insbesondere darauf geachtet werden, dass Nachrichten zum Auswerten der Daten-Argumente eingefügt werden, bevor die eigentliche Nachricht verarbeitet wird.

5.2.2 Beispiele

Das konzeptionelle Modell erlaubt die Darstellung der meisten Konstrukte typischer objektorientierter Programmiersprachen zum Zugriff auf und Manipulation von Daten, zur Steuerung des Kontrollflusses und zur Berechnung von Ergebnissen. In [76] werden die folgenden Konstrukte zum *Zugriff auf Daten* vorgestellt:

- Operationen zum Auslesen und Verändern von Variablen

⁷¹ engl. "lazy evaluation"

⁷² Dies kann bei Mehrfachvererbung auftreten, wenn die unterste Klasse einer Raute von ihren Oberklassen mehrere Methoden zu einer Operation erbt, diese Operation selbst aber nicht reimplementiert.

- Operationen zum Auslesen und Verändern von Objekten

Die Steuerung des Kontrollflusses umfasst die folgenden *Anweisungen*:

- Leere Anweisung
- Ende-Anweisung
- Fallunterscheidungen
- Schleifen
- Sprünge

Zur Berechnung von Ergebnissen werden folgende *Ausdrücke* unterstützt:

- Unäre Operatoren auf den eingebauten Datentypen (etwa Negation eines Boole'schen Wertes)
- Binäre Operatoren auf den eingebauten Datentypen (etwa Addition zweier Zahlen oder Konkatenation zweier Zeichenketten)
- Typanpassungen
- Konstruktoren zum Erstellen neuer Objekte

In diesem Abschnitt wird an den drei Beispielen "Zuweisung" (5.2.2.1), "Fallunterscheidung" (5.2.2.2) und "Ganzzahl-Addition" (5.2.2.3) erläutert, wie Anweisungen und Ausdrücke über Operationen und zugehörige Methoden modelliert werden. Für die detaillierte Darstellung sei auf [76, Kapitel 3] verwiesen.

5.2.2.1 Beispiel für Daten-Manipulation: Die Zuweisung

Der schreibende Zugriff auf eine Variable wird in einer eigenständigen Operation gekapselt. Diese Operation wird *Zuweisung* genannt und ist für jede Variablenklasse definiert. Sie besitzt zwei Daten-Parameter *this* und *source*, die beide vom selben Typ sind und die Variablenklasse der Ziel- und Quell-Variable bezeichnen (Abb. 5.18a und 5.19a). Die zugehörige Methode löscht die *ref*- bzw. *value*-Verknüpfung zum alten Objekt bzw. Wert und erstellt eine neue *ref*- bzw. *value*-Verknüpfung zu dem durch das *source*-Argument referenzierten Objekt (Abb. 5.18b und Abb. 5.18c) bzw. Wert (Abb. 5.19b und Abb. 5.19c).

Man beachte, dass eine Zuweisung an eine Variable lediglich das Ziel der in der Variable enthaltenen Verknüpfung verändert und das vor der Zuweisung referenzierte Objekt bzw. den vor der Zuweisung referenzierten Wert unverändert lässt. Eine Operation zum Ermitteln des von einer Variable referenzierten Objektes bzw. Wertes existiert nicht. Dies ist nicht notwendig, weil alle Methoden Eingaben in Form von Variablen erwarten. Somit kann das Ziel einer Variable (Objekt bzw. Wert) ohnehin nicht direkt in einem Programm verarbeitet werden, ohne vorher in einer Variable (erneut) gekapselt zu werden.

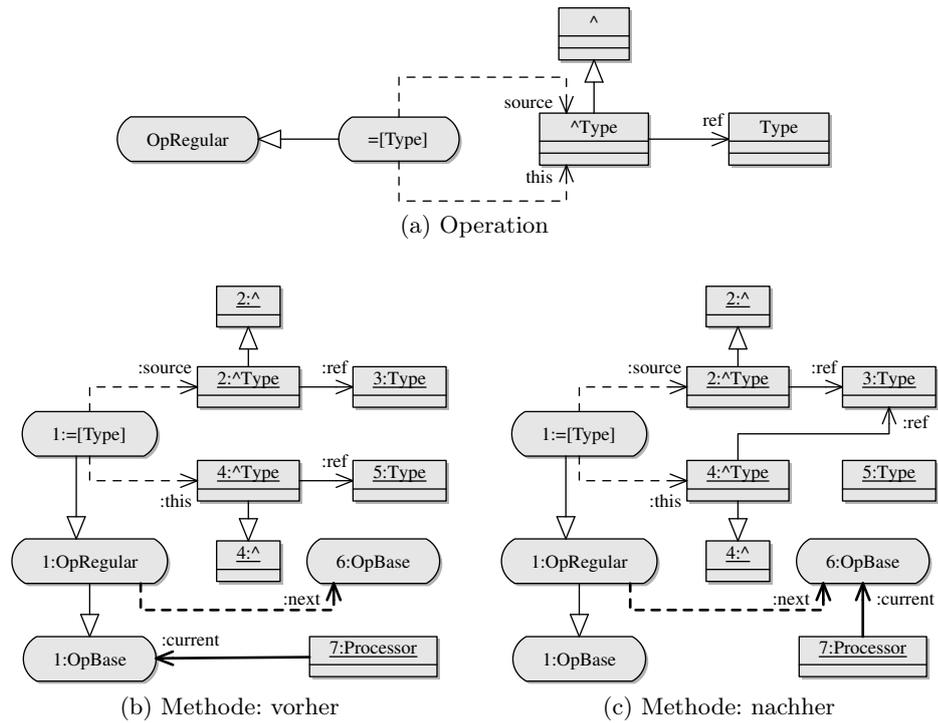


Abb. 5.18: Zuweisung eines Objekts an eine objektwertige Variable

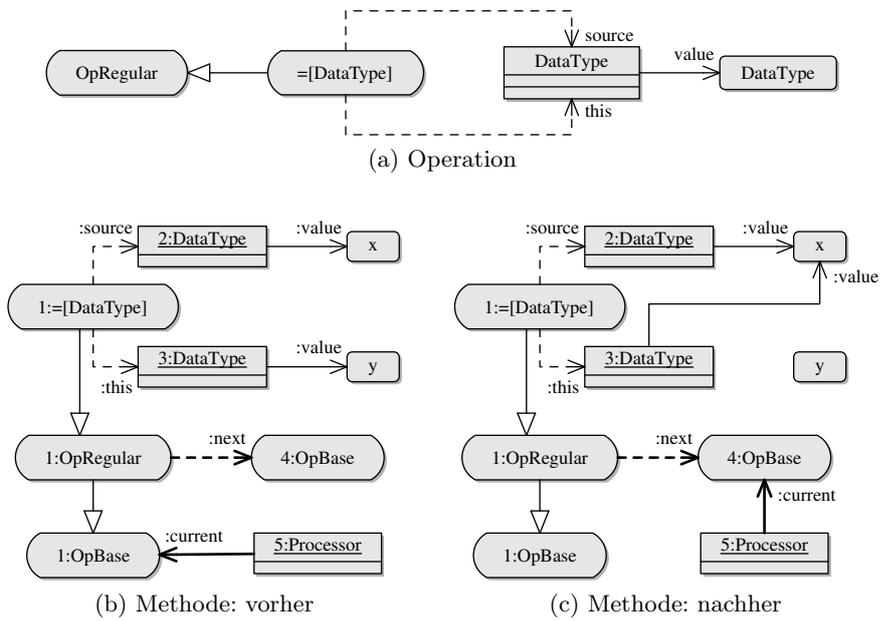


Abb. 5.19: Zuweisung eines Wertes an eine primitive Variable

5.2.2.2 Beispiel für Anweisungen: Die Fallunterscheidung

Anweisungen werden allgemein als spezielle Operationen aufgefasst. Somit wird das Ausführen von Anweisungen immer auf die Verarbeitung von Nachrichten zurückgeführt, die in Abschnitt 5.2.1.7 beschrieben ist. Charakteristisch für diese Operationen ist, dass sie keinen Rückgabetyt und somit keinen *result*-Parameter besitzen.⁷³

Eine einfache Fallunterscheidung wird als Operation *ite* (if-then-else) mit dem Daten-Parameter *this* vom Typ *Bool* und den beiden Operations-Parametern *then* und *else* definiert. Der *this*-Parameter beinhaltet dabei den Wahrheitswert, anhand dessen entschieden wird, ob die Verarbeitung bei der *then*- oder *else*-Nachricht fortgesetzt wird (Abb. 5.20a). Das Besondere an der *ite*-Operation ist neben den zusätzlichen Operations-Parametern, dass sie direkt von der Operation *OpBase* erbt und somit keinen *next*-Parameter besitzt. Der Grund hierfür ist einleuchtend: Die Nachfolger-Nachricht kann erst zur Laufzeit anhand des *this*-Arguments bestimmt werden.

Zu dieser Operation existieren zwei Methoden, die sich jeweils darin unterscheiden, dass die eine Methode den Wahrheitswert *true* und die andere den Wahrheitswert *false* im *this*-Argument erwartet. Die *true*-Methode führt zum Umsetzen der *current*-Verknüpfung auf die Nachricht im *then*-Argument (Abb. 5.20b und 5.20c), die *false*-Methode führt zum Umsetzen der *current*-Verknüpfung auf die Nachricht im *else*-Argument (Abb. 5.20d und 5.20e).

Fallunterscheidungen, die keinen *else*-Zweig besitzen, können abgebildet werden, indem das *else*-Argument mit einer leeren Anweisung belegt wird. Mehrfache Fallunterscheidungen⁷⁴ werden nicht direkt unterstützt, können jedoch durch mehrfache Anwendung einfacher Fallunterscheidungen und entsprechender “Verkabelung” der Nachrichten an den Übergängen abgebildet werden.

5.2.2.3 Beispiel für Ausdrücke: Die Ganzzahl-Addition

Ein Ausdruck wird allgemein als eine Folge von miteinander verketteten Operationen betrachtet. Folglich wird das Auswerten eines Ausdrucks allein auf die Verarbeitung der entsprechenden Nachrichten zurückgeführt. Die Reihenfolge, in der Teilausdrücke ausgewertet werden, wird vom Modell nicht vorgegeben und ist somit von der abzubildenden Programmiersprache zu definieren. Der Rückgriff auf das Verarbeiten von Nachrichten erfordert allerdings, dass jeder berechnete (Zwischen-)Wert in einer eigens dafür erstellten Variable zwischengespeichert wird, was in den meisten Programmiersprachen vor dem Programmierer verborgen wird.

Es folgt ein Beispiel des binären Operators “+”, der in Ausdrücken zur Addition zweier ganzen Zahlen verwendet wird. Die Ganzzahl-Addition wird durch eine Operation abgebildet, die drei Parameter besitzt. Die Parameter *this* und *arg* verweisen auf den ersten und zweiten Operanden des Additions-Operators. Der Parameter *result* nimmt das Ergebnis auf (Abb. 5.21).

⁷³ Dies ist in imperativen Programmiersprachen häufig der einzige Unterschied zwischen Anweisungen und Ausdrücken.

⁷⁴ etwa die *switch*-Anweisung in *Java*, *C* und *C++* oder die *case...of*-Anweisung in *Pascal*

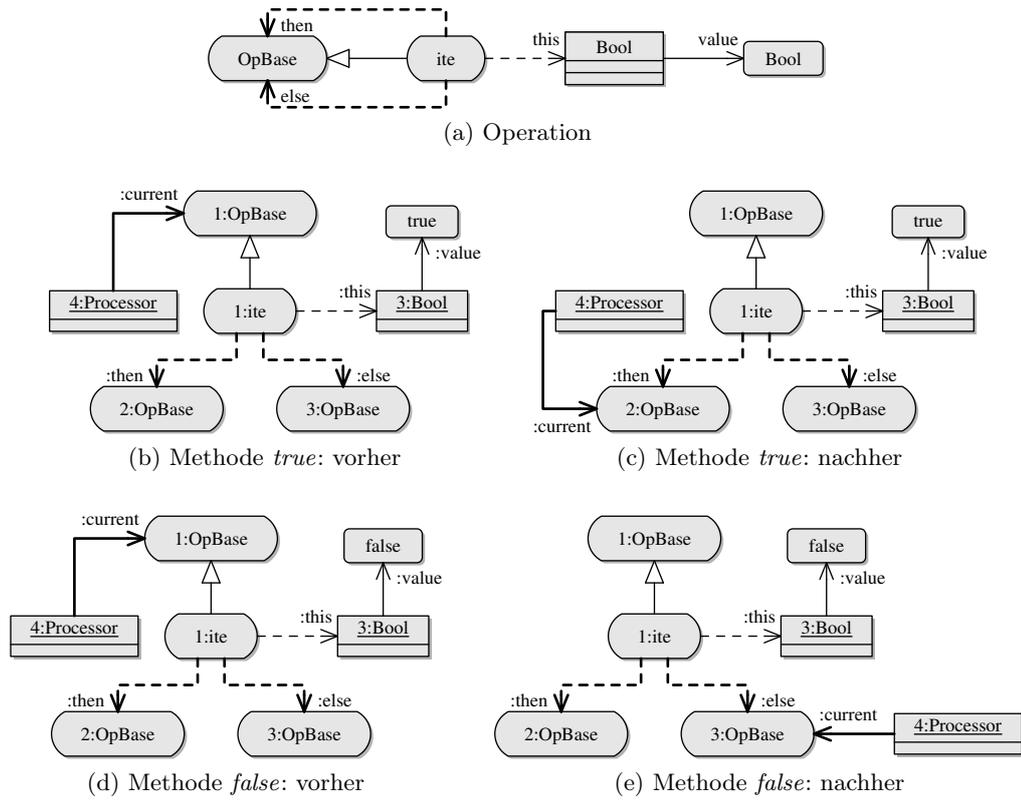


Abb. 5.20: Fallunterscheidung

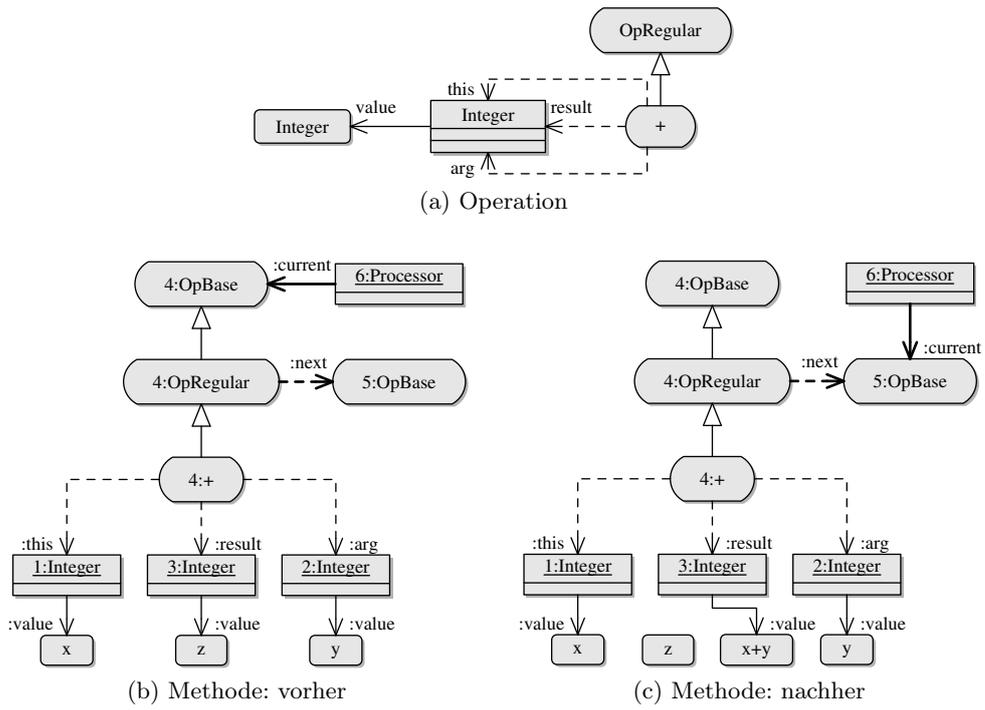


Abb. 5.21: Darstellung der Ganzzahl-Addition

Fallstudie: Konzeption

In diesem Kapitel wird die Fallstudie aus Kapitel 3 Schritt für Schritt ins konzeptionelle Modell übertragen.

6.1 Ausgangssituation

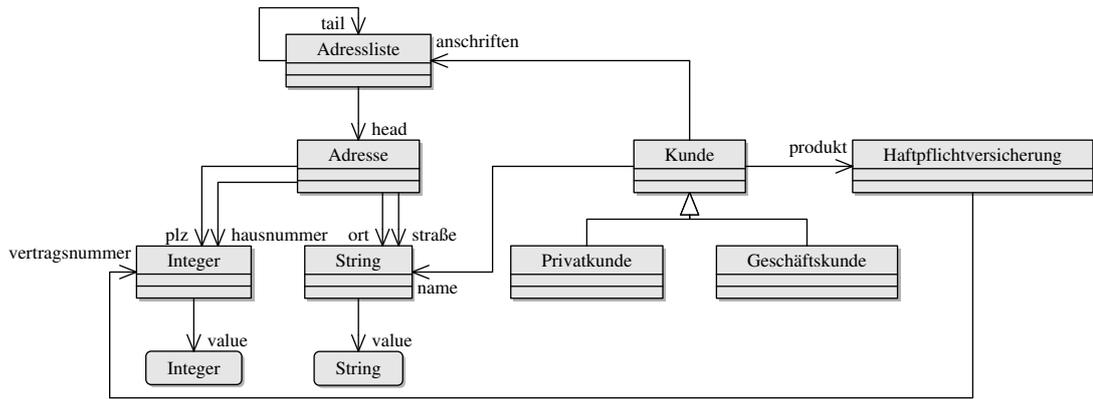
Eine Übertragung der Ausgangssituation der Fallstudie aus Abschnitt 3.1 ins konzeptionelle Modell zieht einige Änderungen nach sich. Zum einen muss die Partikel-Struktur der Objekte ausformuliert werden, so dass ein Objekt zu jedem seiner Typen ein entsprechendes Partikel erhält (5.1.1). Zum anderen sind Zugriffe auf Verknüpfungen und Attributwerte innerhalb von Methoden nur über die Inanspruchnahme der entsprechenden Lese-Operationen erlaubt [76, Abschnitt 3.2.2]. In Abb. 6.1 wird das angepasste Klassen- und Objektmodell dargestellt. Dabei sind die Objekte inklusive zugehöriger Attributwerte der besseren Übersichtlichkeit wegen eingerahmt. Nicht dargestellt ist das *Null*-Objekt sowie die drei von den *Adressliste*-Klasse ausgehenden und zum *Null*-Objekt weisenden *tail*-Verknüpfungen.

Weiter sind Lese-Operationen für die diversen Attribute und Assoziationen erforderlich. In Abb. 6.2 ist exemplarisch die Operation zum Lesen der Assoziation *head* der Klasse *Adressliste* samt zugehöriger Methode dargestellt.

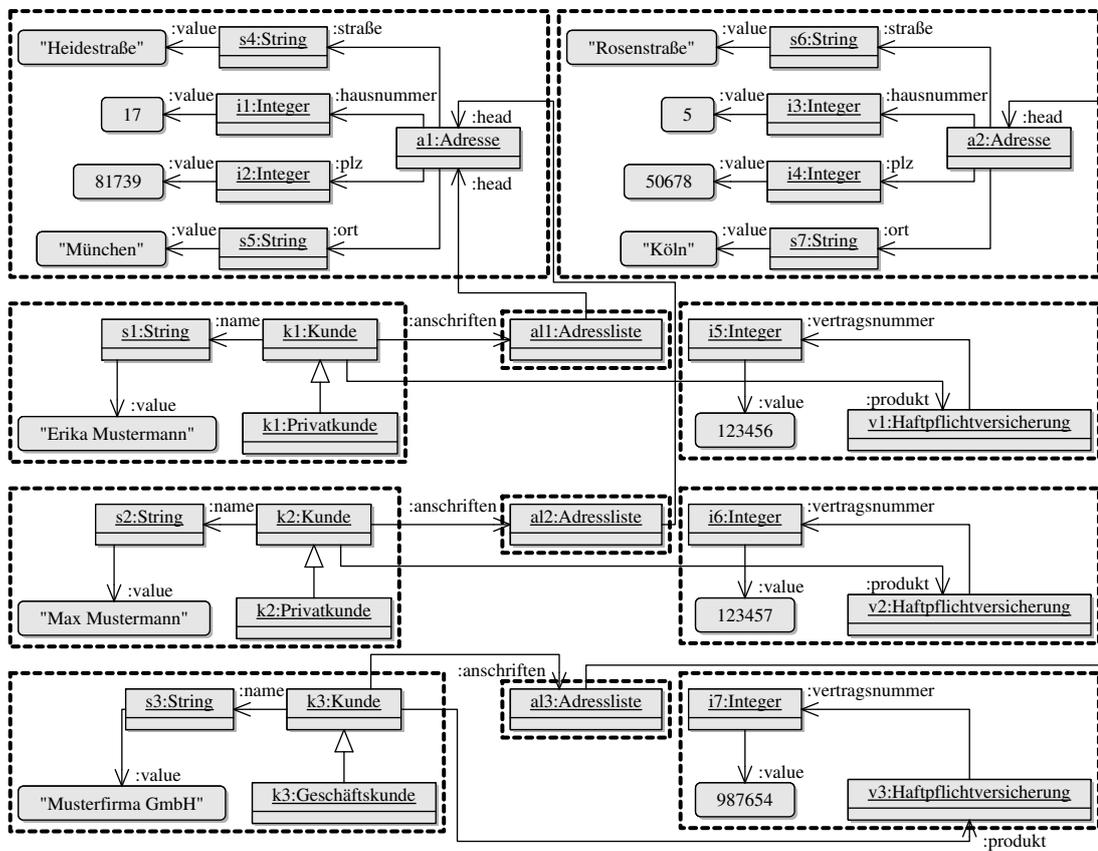
Schließlich wird die Operation *has* der Klasse *Adressliste* formuliert und die zugehörige Methode als Ersetzungsvorschrift implementiert. Dabei kann die zweite Fallunterscheidung entfallen, denn die Implementierung der Methode in der *Null*-Klasse liefert *false* zurück, da dies der Initialwert des primitiven Datentyps *Bool* ist (5.1.5).

6.2 Phase 1: Hinzunahme eines neuen Versicherungsprodukts

In dieser Phase werden Veränderungen am Versicherungsmodell vorgenommen. Die folgenden Änderungen werden im Schema genauso durchgeführt, wie es in Abschnitt 3.2 beschrieben wird:

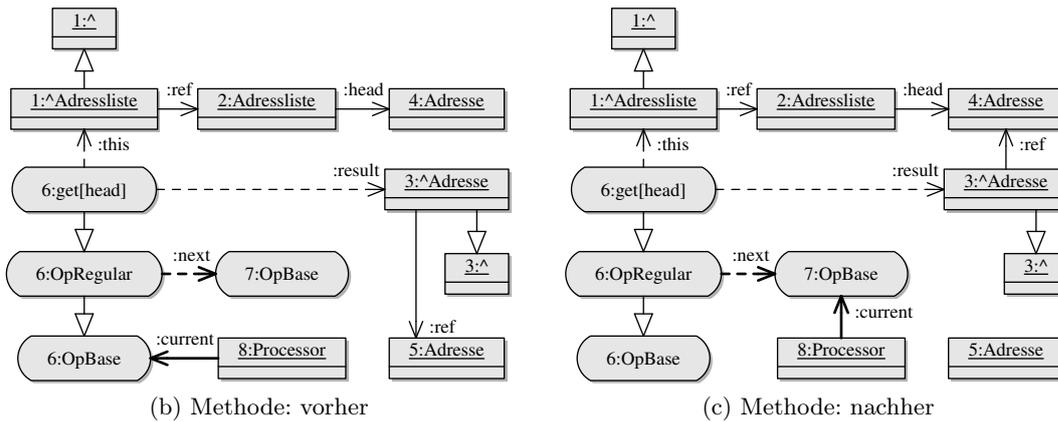
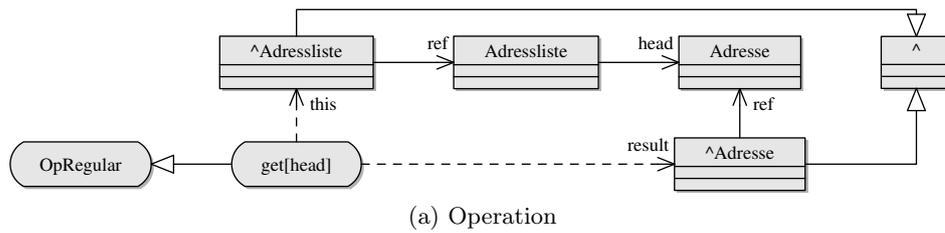


(a) Schema



(b) Ausprägung

Abb. 6.1: Ausgangssituation: Klassen und Objekte



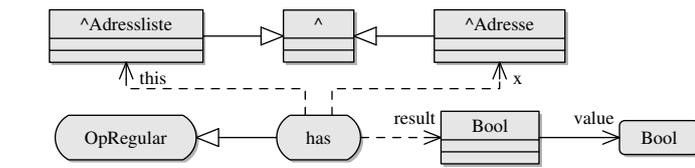
(c) Methode: nachher

Abb. 6.2: Ausgangssituation: Selektion über die *head*-Assoziation

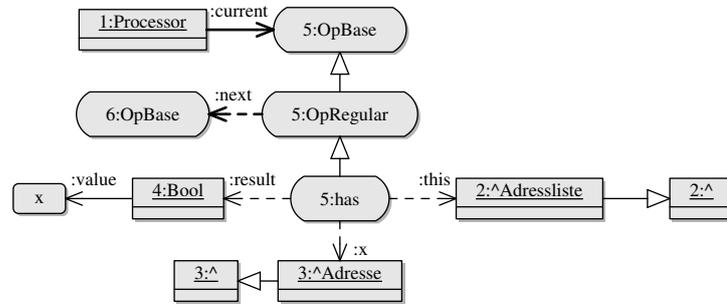
- Erstellung der Klasse *Versicherung* als Oberklasse der Klasse *Haftpflichtversicherung*
- Erstellung der Klasse *Rechtsschutzversicherung* als Unterklasse der Klasse *Versicherung*
- Verschieben des Attributs *vertragsnummer* in die Klasse *Versicherung*
- Verschieben des Ziels der von der Klasse *Kunde* ausgehenden Assoziation *produkt* zur Klasse *Versicherung*
- Verschieben der Quelle des von der Operation *get[vertragsnummer]* ausgehenden Parameters *this* zur Klasse *Versicherung*
- Verschieben des Ziels des von der Operation *get[produkt]* ausgehenden Parameters *result* zur Klasse *Versicherung*

Zusätzlich sind folgende Schema-Anpassungen notwendig:

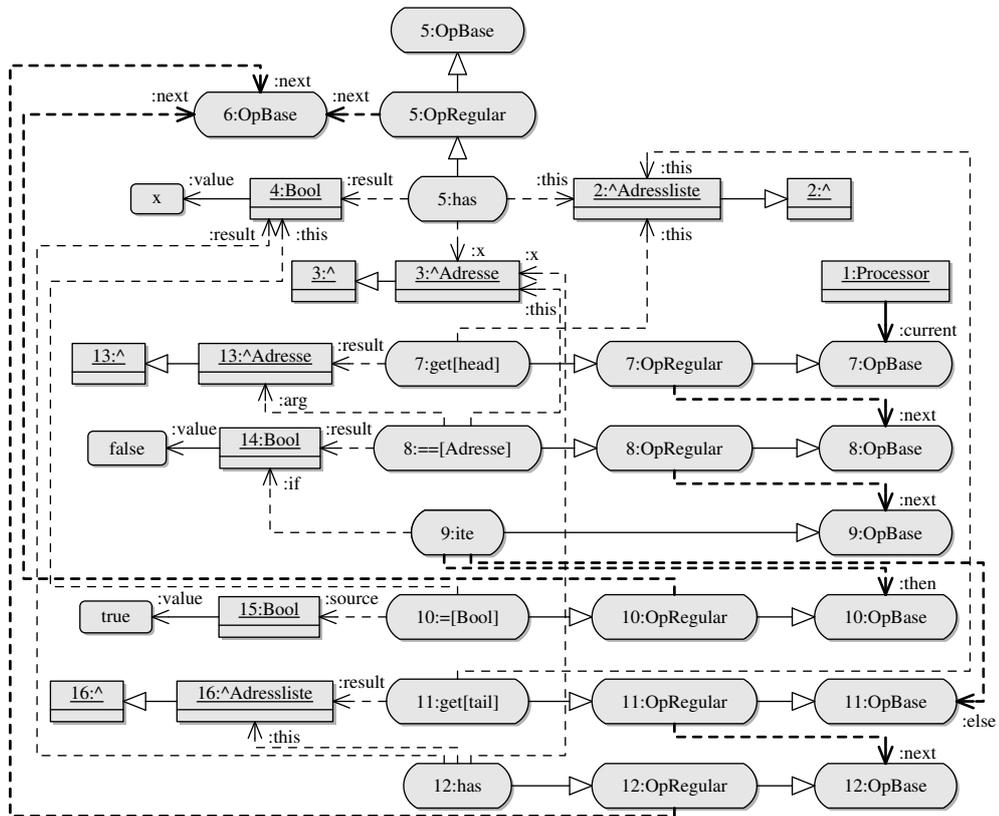
- Erstellung der Klasse \uparrow *Versicherung* und der Assoziation *ref* zur Klasse *Versicherung*
- Erstellung der Klasse \uparrow *Rechtsschutzversicherung* und der Assoziation *ref* zur Klasse *Rechtsschutzversicherung*
- Erstellung aller notwendigen Operationen für Typumwandlungen von und zu den Klassen *Versicherung* und *Rechtsschutzversicherung*
- Erstellung der Vergleichsoperationen für Objekte der Klassen *Versicherung* und *Rechtsschutzversicherung*



(a) Operation



(b) Methode: vorher



(c) Methode: nachher

Abb. 6.3: Ausgangssituation: *has*-Operation der Klasse *Adressliste* samt zugehöriger Methode

Auf der Ausprägungs-Ebene werden die Schema-Änderungen entsprechend nachgezogen:

- Ergänzung aller *Haftpflichtversicherung*-Objekte (einschließlich aller *Null*-Objekte) um ein Partikel zur Klasse *Versicherung*
- Ergänzung aller *Null*-Objekte um ein Partikel zur Klasse *Rechtsschutzversicherung*
- Verschieben aller im Attribut *vertragsnummer* typisierten Attributwerte zum Partikel der Klasse *Versicherung*
- Verschieben des Ziels aller in der Assoziation *produkt* der Klasse *Kunde* typisierten Verknüpfungen zum Partikel der Klasse *Versicherung*
- Verschieben der Quelle aller im Parameter *this* der Operation *get[vertragsnummer]* typisierten Argumente zum Partikel der Klasse *Versicherung*
- Verschieben des Ziels aller im Parameter *result* der Operation *get[produkt]* typisierten Argumente zum Partikel der Klasse *Versicherung*
- Entwicklung aller notwendigen Methoden für Typumwandlungen von und zu den Klassen *Versicherung* und *Rechtsschutzversicherung*
- Entwicklung der Vergleichsmethoden für Objekte der Klassen *Versicherung* und *Rechtsschutzversicherung*

Das angepasste Klassen- und Objektmodell ist in Abb. 6.4 dargestellt.

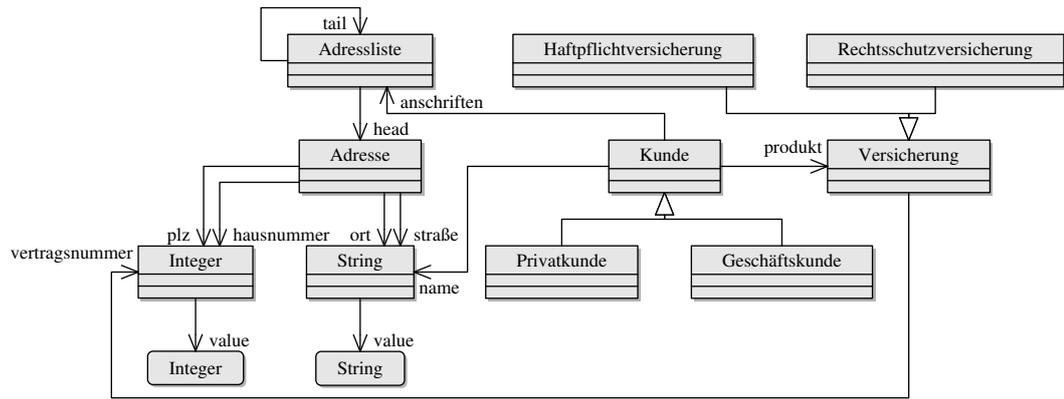
6.3 Phase 2: Hinzunahme ausländischer Kunden

In dieser Phase werden Anpassungen am Adress-Modell vorgenommen. Die folgenden Änderungen werden im Schema genauso durchgeführt, wie es in Abschnitt 3.3 beschrieben wird:

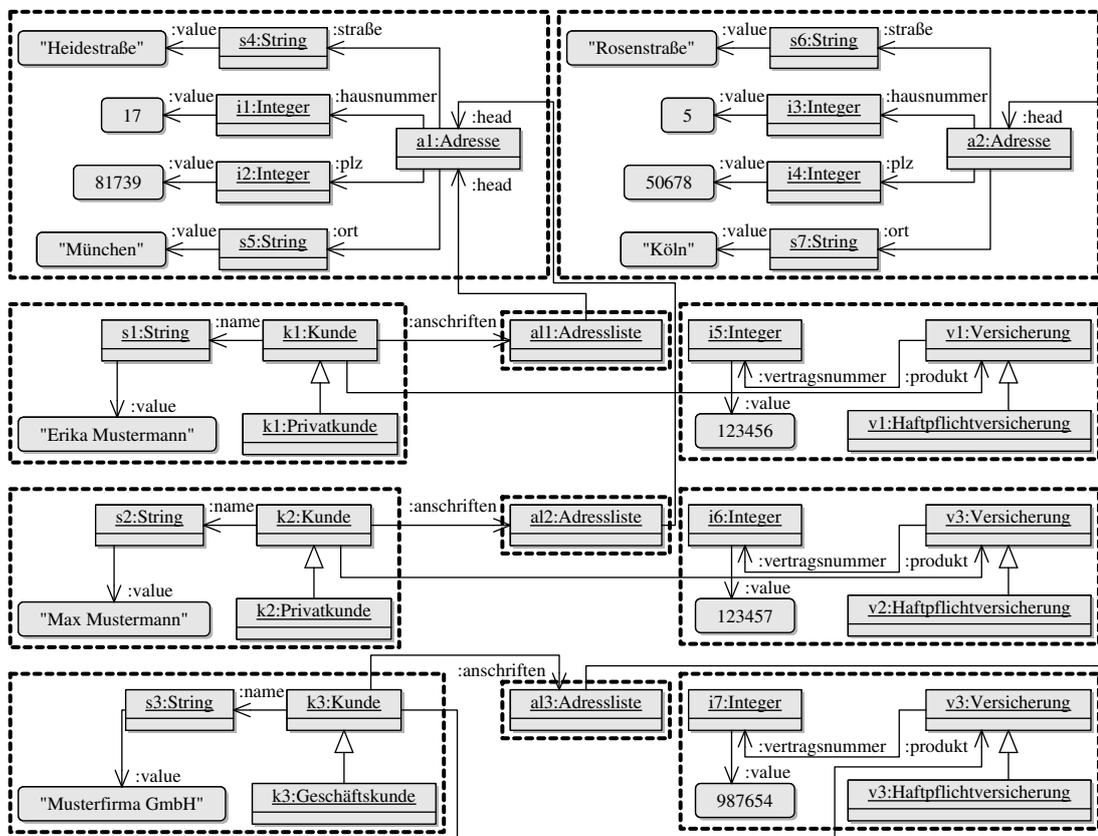
- Umbenennung der Klasse *Adresse* in *Inlandsadresse*
- Erstellung der Klasse *Adresse* als Oberklasse der Klasse *Inlandsadresse*
- Erstellung der Klasse *Auslandsadresse* als Unterklasse der Klasse *Adresse*
- Verschieben des Ziels der von der Klasse *Adressliste* ausgehenden Assoziation *head* zur Klasse *Adresse*
- Verschieben des Ziels des von der Operation *get[head]* ausgehenden Parameters *result* zur Klasse *Adresse*
- Verschieben des Ziels des von der Operation *has* ausgehenden Parameters *x* zur Klasse *Adresse*

Zusätzlich sind folgende Schema-Anpassungen notwendig:

- Erstellung der Klasse \uparrow *Adresse* und der Assoziation *ref* zur Klasse *Adresse*



(a) Schema



(b) Ausprägung

Abb. 6.4: Erweiterung des Versicherungs-Modells: Klassen und Objekte

- Erstellung der Klasse \uparrow *Auslandsadresse* und der Assoziation *ref* zur Klasse *Auslandsadresse*
- Erstellung aller notwendigen Operationen für Typumwandlungen von und zu den Klassen *Adresse* und *Auslandsadresse*
- Erstellung der Vergleichsoperationen für Objekte der Klassen *Adresse* und *Auslandsadresse*

Auf der Ausprägungs-Ebene werden die Schema-Änderungen entsprechend nachgezogen:

- Ergänzung aller *Inlandsadresse*-Objekte (einschließlich aller *Null*-Objekte) um ein Partikel zur Klasse *Adresse*
- Ergänzung aller *Null*-Objekte um ein Partikel zur Klasse *Auslandsadresse*
- Ergänzung aller *Null*-Objekte um einen Attributwert zum Attribut *freitext* der Klasse *Auslandsadresse*, initialisiert mit der leeren Zeichenkette.
- Verschieben des Ziels aller in der Assoziation *head* der Klasse *Adressliste* typisierten Verknüpfungen zum Partikel der Klasse *Adresse*
- Verschieben des Ziels aller im Parameter *result* der Operation *get/head* typisierten Argumente zum Partikel der Klasse *Adresse*
- Verschieben des Ziels aller im Parameter *x* der Operation *has* typisierten Argumente zum Partikel der Klasse *Adresse*
- Entwicklung aller notwendigen Methoden für Typumwandlungen von und zu den Klassen *Adresse* und *Auslandsadresse*
- Entwicklung der Vergleichsmethoden für Objekte der Klassen *Adresse* und *Auslandsadresse*

Das angepasste Klassen- und Objektmodell ist in Abb. 6.5 dargestellt. Die Migration der Operationen und Methoden führt auf Grund des Verschiebens der Ziele der Parameter zu keiner sichtbaren Veränderung in den Abbildungen 6.2 und 6.3.

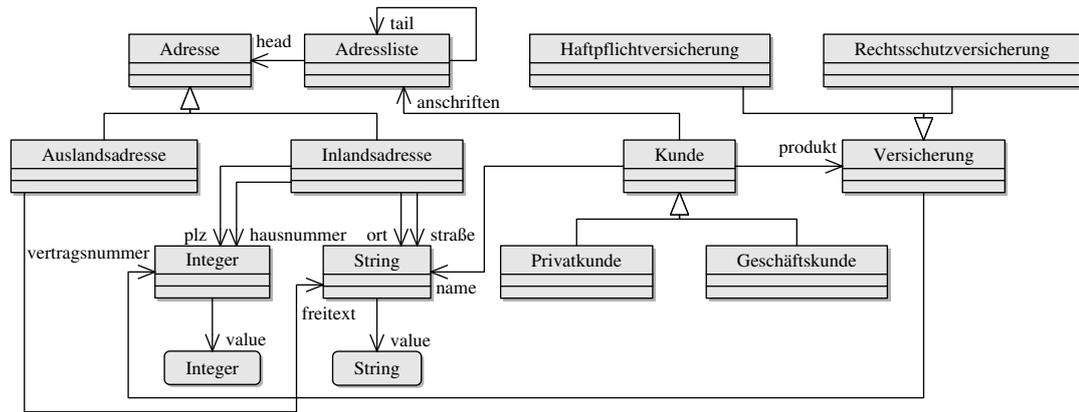
6.4 Phase 3: Konzentration auf Geschäftskunden

In dieser Phase werden Veränderungen am Kunden-Modell durchgeführt. Die folgenden Änderungen werden im Schema genauso durchgeführt, wie es in Abschnitt 3.4 beschrieben wird:

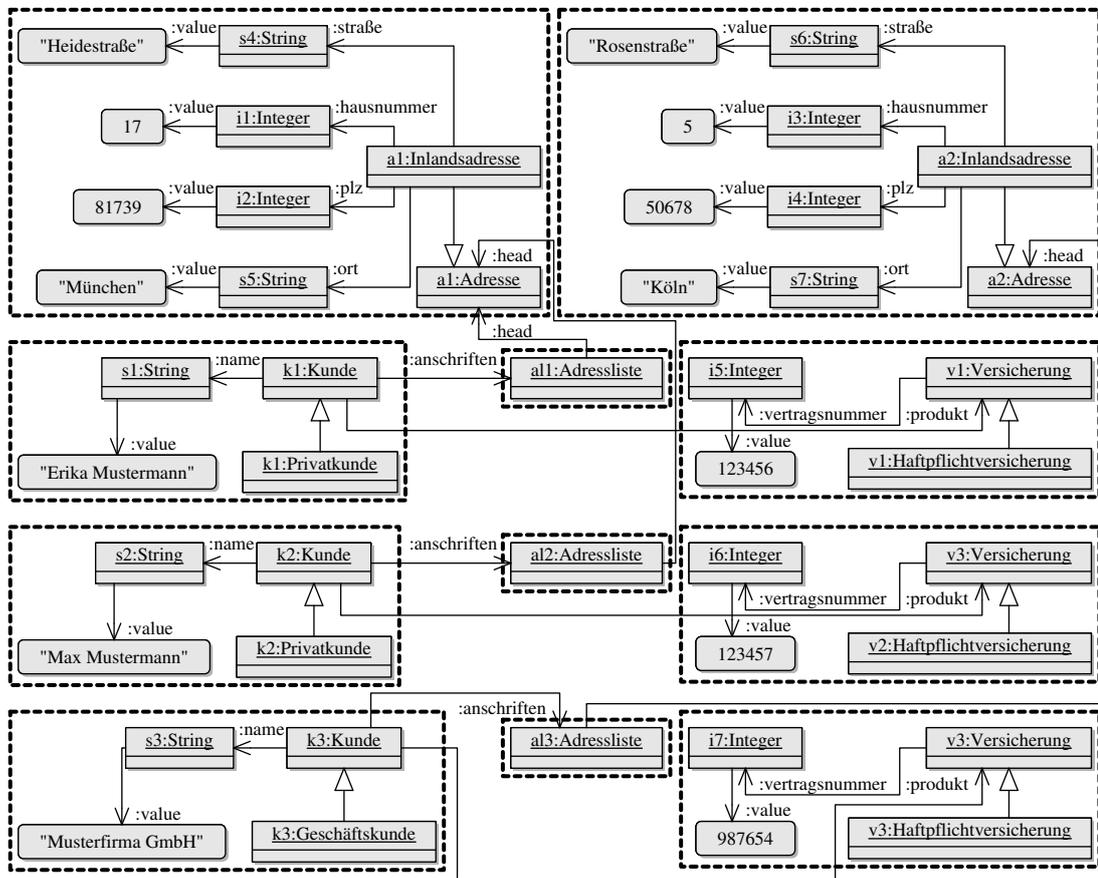
- Löschung der Klasse *Privatkunde*
- Verschmelzung der Klassen *Geschäftskunde* und *Kunde* zur Klasse *Geschäftskunde*

Zusätzlich sind folgende Schema-Anpassungen notwendig:

- Löschung der nicht mehr benötigten Klasse \uparrow *Privatkunde* samt ausgehender Assoziation *ref*



(a) Schema



(b) Ausprägung

Abb. 6.5: Erweiterung des Adress-Modells: Klassen und Objekte

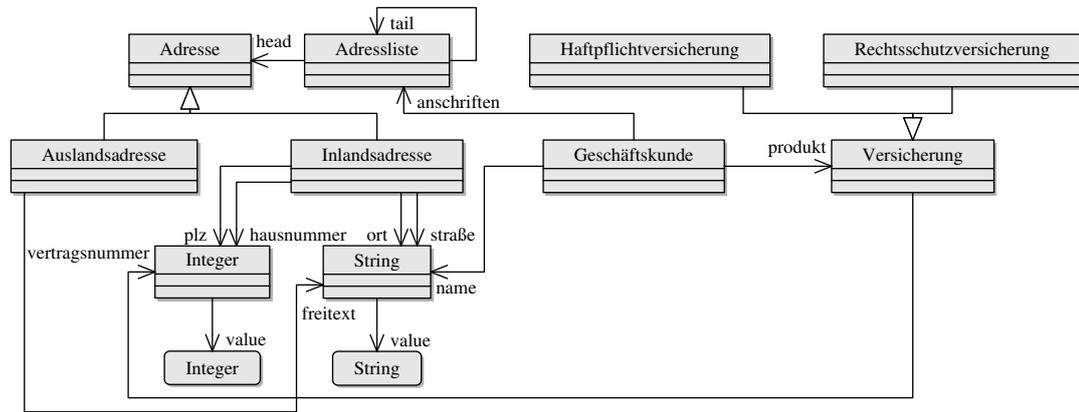
- Löschung aller Operationen für Typumwandlungen von und zu der Klasse *Privatkunde* sowie für Vergleiche von Objekten der Klasse *Privatkunde*
- Verschmelzung der Klassen \uparrow *Geschäftskunde* und \uparrow *Kunde* zur Klasse \uparrow *Geschäftskunde* sowie der zugehörigen *ref*-Assoziationen
- Verschmelzung der jeweiligen Operationen für Typumwandlungen von und zu den Klassen *Geschäftskunde* und *Kunde* mitsamt aller ausgehenden Parametern und eingehenden Assoziationen sowie der Operationen zum Vergleichen von Objekten dieser Klassen

Auf der Ausprägungs-Ebene werden die Schema-Änderungen entsprechend nachgezogen:

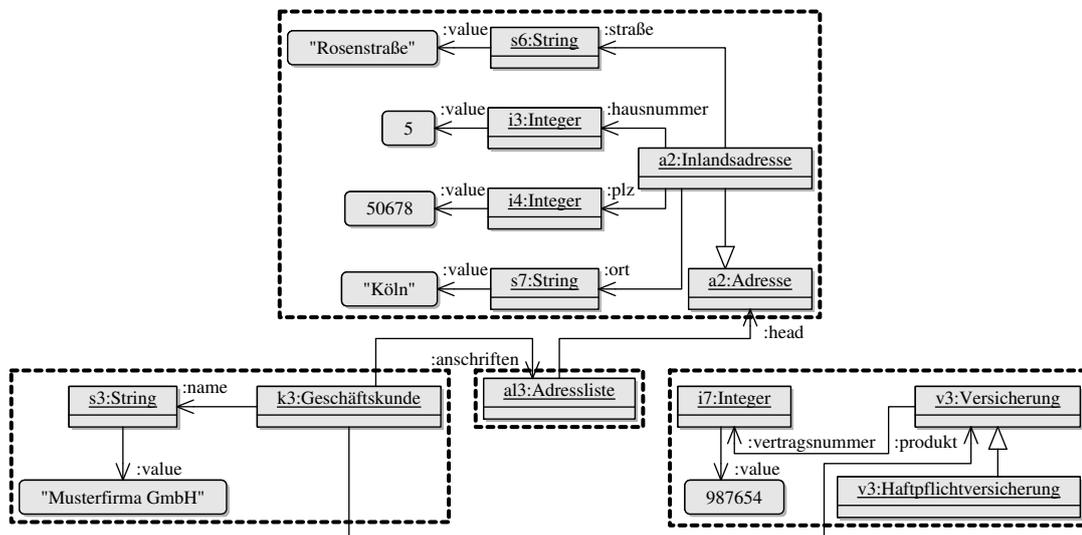
- Verschmelzung aller miteinander über Vererbungsverknüpfungen verbundenen Partikelpaare zu den Klassen *Geschäftskunde* und *Kunde*
- Löschung aller Partikel zu den Klassen *Privatkunde* und \uparrow *Privatkunde*
- Löschung aller Methoden und Nachrichten zu den Typumwandlungs-Operationen, die von oder zur Klasse *Privatkunde* umwandeln, sowie zu Vergleichsoperationen auf Objekten der Klasse *Privatkunde*

Hierbei muss beachtet werden, dass keine Programme und insbesondere Prozesse existieren, die auf die gelöschten Elemente verweisen. Ansonsten kann die Abarbeitung von Programmen nachhaltig gestört werden.

Das angepasste Klassen- und Objektmodell ist in Abb. 6.6 dargestellt. Dabei ist angenommen worden, dass die von den gelöschten Privatkunden referenzierten Adresslisten und die von diesen Listen referenzierte Inlandsadresse manuell gelöscht worden sind.



(a) Schema



(b) Ausprägung

Abb. 6.6: Reduktion des Kunden-Modells: Klassen und Objekte

Mathematisches Modell

Überblick

Dieser Teil beschreibt die Formalisierung des konzeptionellen Modells aus Kapitel 5 und stellt dar, wie in diesem Modell Refactorings definiert, angewandt und komponiert werden können. Die Intention dieses Kapitels ist es, einen mehr intuitiven Überblick über die Ergebnisse dieses Teils zu geben.

Das zentrale Konzept dieser Arbeit ist das gemeinsame *Schema*, in dem Daten, Programme und Prozesse typisiert sind. Ein Schema wird durch eine bestimmte *Graphstruktur* abgebildet. Diese Graphstruktur modelliert die Klassenstruktur inklusive Vererbung, die Assoziationen zwischen den Klassen und die Operationen samt ihren Parametern. Die formale Modellierung des Schemas wird in Kapitel 8 beschrieben.

Daten und *Prozesse* werden zusammen in einer weiteren Graphstruktur modelliert, die in dem Schema typisiert sind. Sowohl Objekte und Verknüpfungen als auch Nachrichten und Argumente reflektieren dabei wie im konzeptionellen Modell die Struktur von Klassen und Assoziationen bzw. von Operationen und Parametern des Schemas; dies gewährleistet, dass die Typisierung von Daten und Prozessen als ein Graphstruktur-Morphismus dargestellt werden kann. Zusätzlich existiert die Möglichkeit, über *Attribute* Werte zu primitiven Datentypen in die Graphstruktur aufzunehmen.

Das Besondere an der Darstellung von Daten und Prozessen ist, dass sie in *einer* Graphstruktur zusammengefasst werden. Dies hat zur Folge, dass Beziehungen zwischen Prozessen und den Daten, mit denen sie arbeiten, durch (interne) Kanten innerhalb der Graphstruktur hergestellt werden, und nicht durch (externe) Graphstruktur-Morphismen. Der Begriff der *Datenbank* umfasst im mathematischen Modell somit nicht nur die in einem Schema typisierten Daten, sondern auch die Prozesse, die mit ihnen arbeiten. Die Modellierung der Daten und Prozesse wird ebenfalls in Kapitel 8 dargestellt.

Kapitel 9 ist technischer Natur. Es definiert die im restlichen Teil verwendeten Kategorien und zeigt Beziehungen zwischen ihnen auf.

In Kapitel 10 wird die strukturelle und operationale Formalisierung von *Programmen* durchgeführt. Während Daten und Prozesse in einer einzigen gemeinsamen Graphstruktur zusammengefasst werden, werden Programme in ihre “Einzelteile”, *Methoden*, aufgespalten. Jede Methode entspricht dabei einer DPO-Regel⁷⁵ $L \xleftarrow{L} K \xrightarrow{r} R$ und beschreibt, wie sich die Datenbank und die Prozesse bei der Ausführung der Methode durch Entfernen (linke Seite der Regel) und Hinzufügen (rechte Seite der Regel) verändern. Die drei Graphstrukturen einer Methode formulieren den für die Methode relevanten Ausschnitt des Programmzustands, bestehend aus Objekten, Verknüpfungen, Nachrichten und Argumenten; die Graphstruktur-Morphismen beschreiben, ob diese Elemente unverändert beibehalten oder auf irgendeine Weise modifiziert werden. Ein Programm besteht somit aus einer Vielzahl von Regeln. Es ist auch nicht an eine bestimmte Datenbank gebunden, sondern kann mit jeder Datenbank arbeiten, sofern deren Schema mit dem Schema des Programms übereinstimmt. Die Abarbeitung eines Prozesses erfolgt, indem die Programm-Regeln auf die Datenbank und den Prozess angewandt werden. Verschiedene Elemente des Modells steuern dabei, in welcher Reihenfolge die Methoden zur Ausführung kommen.

Kapitel 11 geht der Frage nach, wie *Transformationen* mathematisch formuliert werden können. Das zentrale Ergebnis des Kapitels ist die formale Ableitung einer eindeutigen induzierten Migration aus einer gegebenen Schema-Transformation für Daten, Prozesse und Programme. Schließlich wird auch die horizontale Komposition von Transformationen im Detail untersucht.

Kapitel 12 geht im Detail auf die Umsetzung des konzeptionellen Modells ins mathematische Modell ein. Neben der Abbildung der Datenbank- und Software-Konzepte wird ein umfangreicher Katalog von Schema-Transformationen präsentiert, der auf den konzeptionellen Transformationen in [76, Kapitel 4] aufbaut und diese im vorgestellten mathematischen Modell umsetzt. Schließlich wird in Kapitel 13 die Fallstudie aus Kapitel 6 ins mathematische Modell übertragen.

⁷⁵ “DPO” steht für *Double-Pushout* bzw. *Doppel-Pushout* und steht für ein bestimmtes Modell zur algebraischen Beschreibung von Graphstruktur-Transformationen; mehr Informationen hierzu finden sich in Abschnitt B.2.

Schemata, Daten und Prozesse

Dieses Kapitel formalisiert in den Abschnitten 8.1 bis 8.6 die Elemente des konzeptionellen Modells aus Kapitel 5. Dabei wird Schritt für Schritt die Spezifikation M erarbeitet, die im Kern eine bestimmte Klasse von Graphen mit zusätzlichen Eigenschaften, welche die Abbildung objektorientierter Systeme unterstützen, beschreibt. In Abschnitt 8.7 wird die Spezifikation zusammenfassend dargestellt.

8.1 Klassen und Objekte

Für die Abbildung von (vorerst nur eigenständigen) Klassen und Objekten werden zwei Sorten *Class* und *Object* eingeführt. Die Typisierung von Objekten in Klassen wird durch eine Operation *typeO* realisiert:

$M_1 ::=$

sorts	
$C(lass)$	(Klassen)
$O(bject)$	(Objekte)
opns	
$typeO: O \rightarrow C$	(Typisierung von Objekten)

Dieses sehr einfache Modell berücksichtigt noch keine Vererbung zwischen den Klassen. Eine diesbezügliche Erweiterung erfolgt in Abschnitt 8.3.

Anmerkung 8.1 (Typisierung). Die Idee hinter *typeO* ist eine Operation, welche die Objektstruktur homomorph in die Klassenstruktur überträgt. Noch sind keine weiteren Operationssymbole und Prädikate in der Spezifikation enthalten, somit müssen keine Nebenbedingungen an die Operation *typeO* formuliert werden. Dies wird sich bei der Hinzunahme von Operationssymbolen und Prädikaten jedoch ändern. Die Forderung, dass *typeO* einen Homomorphismus darstellt, ist vorteilhaft, denn sie erlaubt es, die Schema- und die Ausprägungs-Ebenen kategoriell voneinander

zu trennen bzw. andersherum die obige Spezifikation als spezielle Komma-Kategorie aufzufassen. Mehr hierzu steht in Kapitel 9. \square

8.2 Assoziationen und Verknüpfungen

Zur Modellierung von binären gerichteten Assoziationen (auf Schema-Ebene) und entsprechenden Verknüpfungen (auf Ausprägungs-Ebene) bietet es sich an, analog zur Spezifikation von Graphen *gerichtete Kanten* zu verwenden, die eine Quelle mit einem Ziel verbinden. Dies führt zu der folgenden Erweiterung der Spezifikation M_1 :

$M_{2a} ::= M_1 +$

sorts

$A(ssociation)$ (Assoziationen)
 $L(ink)$ (Verknüpfungen)

opns

$sourceA: A \rightarrow C$ (Quelle einer Assoziation)
 $targetA: A \rightarrow C$ (Ziel einer Assoziation)
 $sourceL: L \rightarrow O$ (Quelle einer Verknüpfung)
 $targetL: L \rightarrow O$ (Ziel einer Verknüpfung)
 $typeL: L \rightarrow A$ (Typisierung von Verknüpfungen)

axms

Typisierung ist homomorph

$x \in L : typeO(sourceL(x)) = sourceA(typeL(x))$ (bzgl. *source*)

$x \in L : typeO(targetL(x)) = targetA(typeL(x))$ (bzgl. *target*)

Die Richtung von Assoziationen und Verknüpfungen ist hierbei durch die *sourceA*- und *targetA*- bzw. *sourceL*- und *targetL*-Operationen eindeutig bestimmt. Das Element, das über die *sourceA*- bzw. *sourceL*-Operation referenziert wird, bildet die Quelle der Assoziation bzw. Verknüpfung. Das Element, das über die *targetA*- bzw. *targetL*-Operation referenziert wird, bildet das Ziel der Assoziation bzw. Verknüpfung.

Die so abgebildeten Assoziationen sind in ihren Multiplizitäten an Quelle und Ziel noch nicht eingeschränkt und entsprechen somit $(0..*,0..*)$ -Assoziationen. Das konzeptionelle Modell fordert jedoch die $0..1$ -Multiplizität am Ziel einer Assoziation.⁷⁶ Für deren Einhaltung ist ein zusätzliches Axiom notwendig, das in der Spezifikation M_{2b} integriert wird:

⁷⁶ Eigentlich wird die 1-Multiplizität am Ziel von Assoziationen, die auf Klassen verweisen, gefordert. Dies kann jedoch mit positiven Horn-Formeln nicht ausgedrückt werden. Nähere Informationen zu diesem Problem finden sich in Abschnitt 14.1.3.

$M_{2b} ::= M_{2a} +$

axms

Verknüpfungen

$$\begin{aligned} x, y \in L : \text{type}L(x) = \text{type}L(y) \wedge \text{source}L(x) = \text{source}L(y) & \quad (\text{Zu-höchstens-eins-}) \\ \Rightarrow x = y & \quad (\text{Beziehungen}) \end{aligned}$$

Um später in Transformationen Assoziationen aus Klassen entfalten zu können, wird für jede Klasse eine spezielle Schleifen-Assoziation benötigt. Diese Assoziation wird über die Operation $\text{self}C$ für jede Klasse bestimmt. Analog wird jedem Partikel eine entsprechende Schleifen-Verknüpfung über die Operation $\text{self}O$ zugeordnet.⁷⁷ Dies resultiert in der folgenden Spezifikation:

$M_2 ::= M_{2b} +$

opns

$$\text{self}C : C \rightarrow A \quad (\text{Schleife einer Klasse})$$

$$\text{self}O : O \rightarrow L \quad (\text{Schleife eines Partikels})$$

axms

Typisierung ist homomorph

$$x \in O : \text{type}L(\text{self}O(x)) = \text{self}C(\text{type}O(x)) \quad (\text{bzgl. self})$$

self-Axiome

$$x \in C : \text{source}A(\text{self}C(x)) = x \quad (\text{self}C\text{-Anfang})$$

$$x \in C : \text{target}A(\text{self}C(x)) = x \quad (\text{self}C\text{-Ende})$$

$$x \in O : \text{source}L(\text{self}O(x)) = x \quad (\text{self}O\text{-Anfang})$$

$$x \in O : \text{target}L(\text{self}O(x)) = x \quad (\text{self}O\text{-Ende})$$

8.3 Vererbung

In einer Klassenhierarchie können Klassen miteinander über Vererbungsbeziehungen verbunden sein. Vererbungsbeziehungen sind *zweistellig* und *gerichtet*. Das bedeutet, dass bei jeder Vererbungsbeziehung genau zwei Klassen involviert sind, wobei jede der beiden Klassen entweder die Rolle der Oberklasse oder die Rolle der Unterklasse übernimmt. Eine solche Beziehung kann beispielsweise mit einer speziellen Assoziation modelliert werden, weil Assoziationen ebenfalls zweistellig und gerichtet sind.

⁷⁷ Diese Erweiterung entspricht der Spezifikation *Graph'* aus [43, Kapitel 2].

Wegen der Einzigartigkeit der Vererbungsbeziehung wird in dieser Arbeit jedoch ein anderer Ansatz gewählt. Dabei wird Vererbung durch ein spezielles *binäres Prädikat* abgebildet. Die grundlegende Idee ist hierbei, die Klassenhierarchie als *partielle Ordnung* zu begreifen. Die Relation \prec ist dabei wie folgt definiert: Für je zwei Klassen $C1$ und $C2$ gilt:

$$C1 \prec C2 \iff C1 \text{ ist direkte Unterklasse von } C2$$

Die Relation \leq ergibt sich aus dem reflexiven und transitiven Abschluss der Relation \prec . Bildet die zugrunde liegende Klassenstruktur eine Hierarchie, so ist die Relation \leq antisymmetrisch und somit eine partielle Ordnung. Diese partielle Ordnung wird nun auf der Schema-Ebene durch ein Prädikat *underC* abgebildet. Die Eigenschaften der Reflexivität, Antisymmetrie und Transitivität werden als entsprechende positive Horn-Formeln formuliert. Zusammen ergibt dies die folgende Spezifikation:

$M_{3a} ::= M_2 +$

prds

$underC: C C$ (Unterklasse von)

axms

Vererbung

$x \in C : underC(x, x)$ (Reflexivität)

$x, y \in C : underC(x, y) \wedge underC(y, x) \Rightarrow x = y$ (Antisymmetrie)

$x, y, z \in C : underC(x, y) \wedge underC(y, z) \Rightarrow underC(x, z)$ (Transitivität)

Diese Darstellung hat den großen Vorteil, dass Axiome formuliert werden können, die den transitiven Abschluss der Vererbungsbeziehung ausnutzen und somit über Vererbungspfade beliebiger Länge Aussagen treffen können. Das ist bei Assoziationen nicht ohne Weiteres möglich, hier kann eine Aussage über Assoziationspfade beliebiger Länge nur über eine unendliche Menge von Axiomen formuliert werden. Andersherum ist eine Darstellung von Assoziationen durch binäre Prädikate nicht vorteilhaft. Denn zwischen zwei Klassen können viele unterschiedliche Assoziationen existieren, die unabhängig voneinander ausgeprägt sein können. Somit wäre in dieser Darstellung für jede Assoziation ein eigenes Prädikat notwendig und die Anzahl der Prädikate nicht von vornherein abzusehen. Die Darstellung von Assoziationen als eigenständige Elemente einer Trägermenge vermeidet dieses Problem.

Die Hinzunahme von Vererbung ins Modell lässt auch die Frage entstehen, ob die Darstellung von Objekten angepasst werden muss. Weil eine Klasse nun im Schema durch mehrere Elemente dargestellt wird, die über Vererbungsbeziehungen miteinander verbunden sind, ist es sinnvoll, diese Struktur auf der Ausprägungs-Ebene zu reflektieren. Ein Element zur Sorte O stellt folglich kein *ganzes* Objekt, sondern lediglich ein *Partikel* dar. Der Unterschied in der Betrachtungsweise

ist genau dann zu erkennen, wenn ein Objekt zu einer Klasse mit mindestens einer Oberklasse dargestellt werden soll: Hier wird sowohl zu der Klasse als auch zu jeder direkten und indirekten Oberklasse im Schema ein Partikel ausgeprägt. Diese Partikel müssen ebenfalls über einen Mechanismus miteinander verbunden werden, damit klar ist, welche Partikel letztlich das Objekt ausmachen. Analog zur Darstellung der Schema-Vererbungsbeziehungen über ein binäres Prädikat wird für das Verbinden der Partikel ein weiteres binäres Prädikat *underO* gewählt, das die Ausprägungen der Vererbungsbeziehungen (Vererbungsverknüpfungen genannt) nachvollzieht. Dies führt zu der folgenden Erweiterung der Spezifikation:

$M_3 ::= M_{3a} +$

prds

$underO: O O$ (Unterobjekt von)

axms

Typisierung ist homomorph

$x, y \in O : underO(x, y) \Rightarrow underC(typeO(x), typeO(y))$ (bzgl. Unterobjekten)

Vererbung

$x \in O : underO(x, x)$ (Reflexivität)

$x, y \in O : underO(x, y) \wedge underO(y, x) \Rightarrow x = y$ (Antisymmetrie)

$x, y, z \in O : underO(x, y) \wedge underO(y, z) \Rightarrow underO(x, z)$ (Transitivität)

Die Eigenschaften der partiellen Ordnung sichern die Zyklensfreiheit in der Vererbungsstruktur. Ein Vererbungszyklus ist ein Kreis durch den Vererbungsgraphen entlang von Vererbungsbeziehungen, wobei die Anzahl der besuchten Vererbungsbeziehungen die Länge des Zyklus darstellt. In Abb. 8.1 sind beispielhaft zwei Schema-Vererbungszyklen der Länge eins⁷⁸ (8.1a) und der Länge drei (8.1b) gezeigt.

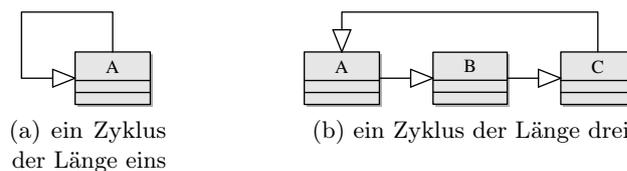


Abb. 8.1: Zyklen in der Schema-Vererbungsstruktur

Zyklen in der Vererbungsstruktur sind aus mehreren Gründen problematisch:

⁷⁸ Ein Zyklus der Länge eins wird auch *Vererbungsschlinge* genannt.

- Abbildung 8.2a verdeutlicht, dass bei Zyklen die Klasse eines Objekts nicht mehr eindeutig ist. Im abgebildeten Beispiel ist nicht klar, ob die Partikel ein *A*-, ein *B*- oder ein *C*-Objekt darstellen.
- Die Anzahl der möglichen Partikel in einem Objekt ist nicht mehr begrenzt. Abbildung 8.2b deutet an, dass durch die Abwicklung des Schema-Zyklus auf der Ausprägungsebene ein Objekt mit einer unbegrenzten Anzahl von Partikeln entsteht.

Die Vererbungsaxiome vermeiden diese Probleme.

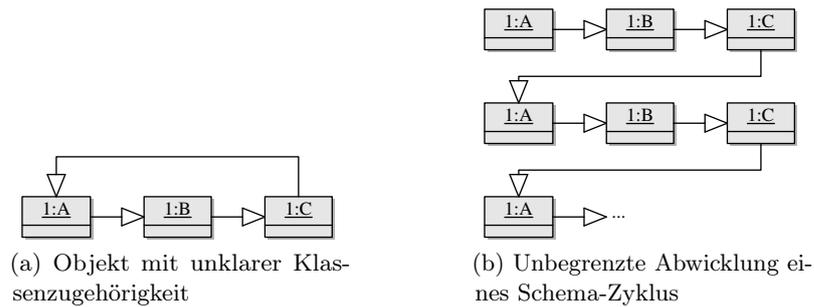


Abb. 8.2: Zyklen in der Ausprägungs-Vererbungsstruktur

Sind zwei Klassen in einer (direkten oder indirekten) Unter-/Oberklassen-Beziehung, so existiert ein gerichteter *Pfad* zwischen diesen Klassen entlang von Vererbungsbeziehungen. Analog existiert ein solcher Pfad zwischen zwei Partikeln entlang von Vererbungsverknüpfungen, wenn diese sich in einer (direkten oder indirekten) Unter-/Oberelement-Beziehung befinden. Diese Erkenntnis wird in der folgenden Definition formalisiert:

Definition 8.2 (Pfad). Eine Folge von Klassen oder Objekten $p = (x_1, x_2, \dots, x_n)$ in einem System A mit $n \in \mathbb{N}^+$ ist ein n -elementiger Pfad von x_1 nach x_n in A , wenn

$$(x_i, x_{i+1}) \in \text{under}^A$$

für alle $i < n$ gilt, wobei $\text{under} = \text{underC}$ für eine Folge von Klassen und $\text{under} = \text{underO}$ für eine Folge von Partikeln gilt. Ein leerer Pfad liegt vor bei $n = 1$, er hat genau ein Element. Die Notationen

$$x_i \in p$$

$$(x_i, x_{i+1}) \in p$$

drücken die Zugehörigkeit des Elements x_i bzw. der direkt benachbarten Elemente x_i und x_{i+1} zum Pfad p aus.

8.4 Komponenten

Nach dem konzeptionellen Modell darf ein Objekt zu jeder Klasse innerhalb seiner Klassenhierarchie höchstens ein Partikel besitzen. Dies entspricht einer *injektiven* Einbettung der Partikel eines Objekts in die Klassenstruktur. Um diese Eigenschaft zu erzwingen, lässt sich die Spezifikation um die folgenden Axiome erweitern, welche den transitiven Charakter der Vererbungsstruktur ausnutzen:

$$M_{4a} ::= M_3 +$$

axms

Vererbung

$$x, y, z \in O : \text{underO}(x, y) \wedge \text{underO}(x, z) \wedge \text{typeO}(y) = \text{typeO}(z) \quad (\text{Eindeutige Oberobjekte})$$

$$\Rightarrow y = z$$

$$x, y, z \in O : \text{underO}(y, x) \wedge \text{underO}(z, x) \wedge \text{typeO}(y) = \text{typeO}(z) \quad (\text{Eindeutige Unterobjekte})$$

$$\Rightarrow y = z$$

Das erste Axiom verhindert, dass ein Partikel zwei Oberobjekte vom selben Typ besitzt. Das zweite Axiom verhindert analog, dass ein Partikel zwei Unterobjekte vom selben Typ besitzt. Letzteres ist äquivalent zu der Forderung, dass zwei unterschiedliche Objekte zur selben Klasse sich keine Partikel teilen dürfen. Zusammen erzwingen die Axiome bei korrekten Objekten die gewünschte injektive Abbildung von zusammenhängenden Objektstrukturen auf Vererbungsstrukturen; dies ist auf Grund der Transitivität des *underO*-Prädikats auch bei Mehrfachvererbung gewährleistet. Ein Beispiel, das von den beschriebenen Axiomen ausgeschlossen wird, ist in Abb. 8.3 dargestellt.

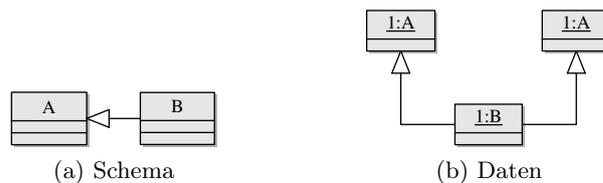
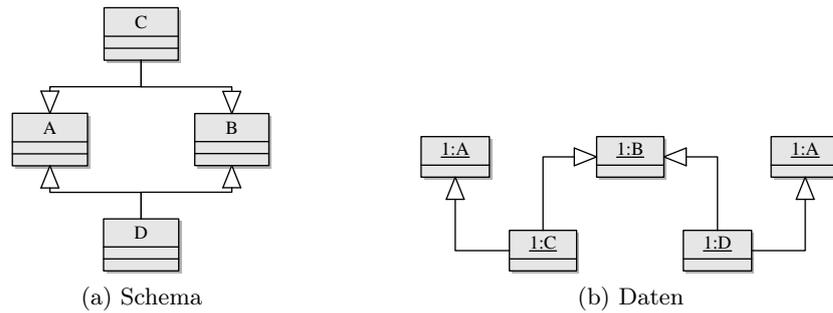


Abb. 8.3: Mehrfach klassifiziertes Objekt mit mehr als einem Partikel zur Klasse A (1)

Diese Axiome funktionieren jedoch nicht wie gewünscht, wenn mehrfach klassifizierte Objekte vorliegen. In Abb. 8.4 ist eine entsprechende Situation abgebildet: Hier besitzt das Objekt keinen eindeutigen speziellsten Typ. Es hat zwei Partikel zur Klasse A, was jedoch von den obigen Axiomen nicht verhindert wird. Somit ist eine injektive Abbildung der Objektstruktur in die Klassenstruktur nicht mehr gegeben.

Offensichtlich sind auch in diesem Beispiel die betreffenden Partikel zur Klasse A miteinander über Vererbungsverknüpfungen verbunden. Um solche Situationen auszuschließen, ist es sinnvoll, bei

Abb. 8.4: Mehrfach klassifiziertes Objekt mit mehr als einem Partikel zur Klasse A (2)

der Betrachtung von Objekten die Richtung der Vererbungsverknüpfungen zu ignorieren. Hierzu wird ein neues Konzept eingeführt: die *Komponente*. Eine Komponente ist ein Zusammenschluss von Elementen (Klassen oder Partikeln), die über Vererbungsbeziehungen miteinander verbunden sind, wobei die Richtung der Beziehungen ignoriert wird. Im obigen Beispiel gehören die vier Klassen A bis D zu einer Komponente auf der Schema-Ebene und die fünf Partikel zu einer (anderen) Komponente auf der Ausprägungs-Ebene. Unter Verwendung dieses Begriffs kann der Wunsch nach Typ-Eindeutigkeit von Partikeln folgendermaßen neu formuliert werden: Es wird gefordert, dass jede Komponente auf der Ausprägungsebene zu jeder Klasse höchstens ein Partikel besitzt. Damit wird die Objektstruktur in Abb. 8.4b ausgeschlossen.

Mathematisch lassen sich Komponenten wie folgt modellieren: Die partiellen Ordnungen, welche die Vererbungsstruktur modellieren und mit Hilfe der Prädikate *underC* auf der Schema- und *underO* auf der Ausprägungsebene ausgedrückt werden, werden symmetrisch und transitiv abgeschlossen und ergeben somit zwei Äquivalenzrelationen, die durch die Prädikate *relC* und *relO* abgebildet werden.⁷⁹ Die durch diese Äquivalenzen entstehenden Partitionen sind die gewünschten Komponenten. Das Axiom zur Typ-Eindeutigkeit von Partikeln kann nun leicht mit Hilfe des *relO*-Prädikats formuliert werden.

$$M_4 ::= M_3 +$$

prds

$$relC: C C \quad (\text{Verwandte Klassen})$$

$$relO: O O \quad (\text{Verwandte Partikel})$$

axms

Typisierung ist homomorph

$$x, y \in O : relO(x, y) \Rightarrow relC(typeO(x), typeO(y)) \quad (\text{bzgl. Komponenten})$$

Komponenten

$$x, y \in C : relC(x, y) \Rightarrow relC(y, x) \quad (\text{Symmetrie})$$

⁷⁹ Abkürzung für “related” (dt. “verwandt”)

$x, y, z \in C : relC(x, y) \wedge relC(y, z) \Rightarrow relC(x, z)$	(Transitivität)
$x, y \in O : relO(x, y) \Rightarrow relO(y, x)$	(Symmetrie)
$x, y, z \in O : relO(x, y) \wedge relO(y, z) \Rightarrow relO(x, z)$	(Transitivität)
$x, y \in C : underC(x, y) \Rightarrow relC(x, y)$	(Schema-Komponenten)
$x, y \in O : underO(x, y) \Rightarrow relO(x, y)$	(Ausprägungs-Komponenten)
$x, y \in O : relO(x, y) \wedge typeO(x) = typeO(y) \Rightarrow x = y$	(Eindeutige Komponenten)

Die Reflexivität der Relationen zu den Prädikaten $relC$ und $relO$ ergibt sich aus der Reflexivität der Relationen zu den Prädikaten $underC$ und $underO$ sowie den Axiomen “Schema-Komponenten” und “Ausprägungs-Komponenten”, so dass hierfür keine separaten Axiome notwendig sind.

Anmerkung 8.3 (Komponenten). Die Komponenten entsprechen in ihrer Semantik den Komponenten, die in den Papieren [51, 52] beschrieben werden. Dort werden Partikel bzw. Klassen, die ein Objekt bzw. eine Vererbungshierarchie ausmachen, ebenfalls zusammengefasst. Formal werden jedoch Komponenten in dieser Arbeit und Komponenten in [51, 52] ganz unterschiedlich modelliert. □

Anmerkung 8.4 (Komponenten und Vererbung). Die obige Spezifikation verhindert nicht, dass zwei Klassen oder Partikel sich in derselben Komponente befinden, die nicht über Vererbungsbeziehungen miteinander verbunden sind. Vielmehr wird lediglich verlangt, dass Komponenten die Vererbungsstrukturen umfassen. Es können somit weniger und größere Komponenten gebildet werden; im Extremfall sind alle Klassen und alle Partikel in jeweils einer Komponente. Dies ist jedoch in der Regel nur wenig sinnvoll. Auch wenn die weiteren Resultate in dieser Arbeit so formuliert sind, dass beliebige Komponentenbildungen erlaubt sind, sofern sie die partielle Ordnung respektieren, wird in den Beispielen implizit angenommen, dass die Prädikate $relC$ und $relO$ in jedem System die kleinsten Äquivalenzrelationen sind, welche die partiellen Ordnungen $underC$ und $underO$ in dem betrachteten System umfassen. □

Im letzten Abschnitt wurden Pfade zwischen Klassen bzw. zwischen Partikeln eingeführt. Zwei Elemente sind Teil eines Pfades, wenn sie durch eine (möglicherweise leere) ununterbrochene Kette von gleichgerichteten Vererbungsbeziehungen bzw. Vererbungsverknüpfungen miteinander verbunden sind. Gelegentlich ist es notwendig, Pfade um Paare von Elementen zu erweitern, die nicht über Vererbung miteinander in Beziehung stehen, sondern sich lediglich in derselben Komponente befinden. Dies führt zu der folgenden Definition:

Definition 8.5 (Erweiterter Pfad). *Eine Folge von Klassen oder Objekten $\hat{p} = (x_1, x_2, \dots, x_n)$ in einem System A mit $n \in \mathbb{N}^+$ ist ein n -elementiger erweiterter Pfad von x_1 nach x_n in A , wenn*

$$(x_i, x_{i+1}) \in rel^A$$

für alle $i < n$ gilt, wobei $rel = relC$ für eine Folge von Klassen und $rel = relO$ für eine Folge von Partikeln gilt. Ein leerer erweiterter Pfad liegt vor bei $n = 1$, er hat genau ein Element. Die Notationen

$$\begin{aligned} x_i &\in \hat{p} \\ (x_i, x_{i+1}) &\in \hat{p} \end{aligned}$$

drücken die Zugehörigkeit des Elements x_i bzw. der direkt benachbarten Elemente x_i und x_{i+1} zum erweiterten Pfad \hat{p} aus.

8.5 Datentypen und Werte

Um Werte zu primitiven Datentypen darzustellen, wird das Modell gemäß der Definition von attribuierten Graphstruktur-Signaturen (B.8) um eine Daten-Spezifikation erweitert, die zusätzliche Sorten und Operationssymbole zum Verarbeiten von Werten zu primitiven Datentypen anbietet. Diese Daten-Spezifikation, im Folgenden *Data* genannt, spezifiziert die primitiven Datentypen und die darauf existierenden Operationen und legt somit die Basis-Semantik fest. Folglich ist sie in hohem Maße von der abzubildenden Programmiersprache abhängig, denn nicht jede Programmiersprache besitzt denselben Satz an primitiven Datentypen. In dieser Arbeit wird die Existenz der Datentypen *BOOL* (Wahrheitswerte), *INTEGER* (Ganzzahlen), *CHARACTER* (Zeichen) und *STRING* (Zeichenketten) angenommen. Daraus ergibt sich die kombinierte Spezifikation *Data* zu:

$$Data ::= BOOL \cup INTEGER \cup CHARACTER \cup STRING$$

Jede dieser Spezifikationen beinhaltet mindestens eine Sorte für die Werte sowie Operationssymbole für Operationen, die auf dieser Sorte arbeiten.⁸⁰ Diese Operationen werden nicht formal definiert oder spezifiziert, sondern es wird angenommen, dass die Bedeutung der Operationen intuitiv erschlossen werden kann, etwa die Operationen *and* und *or* als Konjunktion und Disjunktion zweier Wahrheitswerte, die Operatoren $+$ und $-$ zum Addieren und Subtrahieren ganzer Zahlen oder die Operation *concat* zum Verketteten zweier Zeichenketten. Es wird vorausgesetzt, dass die Operationen die Semantik der zugrundeliegenden Programmiersprache vollständig abdecken können.

Die Daten-Spezifikation *Data* wird zweifach Teil des Modells, einmal für die Daten des Schemas und einmal für die Daten der Ausprägung. Somit beinhaltet das Modell für jeden primitiven Datentyp

⁸⁰ Dies schließt nicht aus, dass noch andere Sorten existieren oder dass Operationssymbole "zwischen" den Sorten der Spezifikationen vermitteln, etwa ein Operationssymbol $\leq: V_{INTEGER} V_{INTEGER} \rightarrow V_{BOOL}$ zum Vergleich zweier Ganzzahlen.

zwei Werte-Sorten, eine für das Schema und eine für die Ausprägung. Folglich kann die Spezifikation eines primitiven Datentyps für Schema und Ausprägung unterschiedlich implementiert werden. Was auf den ersten Blick seltsam anmutet, ist auf den zweiten Blick notwendig: Damit ein System, das als Ausprägung des Modells verstanden wird, mit Hilfe von Graph-Transformationen verändert werden kann, muss es als typisierte attributierte Graphstruktur aufgefasst werden können, und dazu muss die Daten-Spezifikation *Data* für das Schema durch das finale Modell implementiert werden. Das finale Modell als Implementierung der Daten-Spezifikation ist auf der Ebene der Ausprägung hingegen nur wenig sinnvoll.⁸¹ Somit werden von der Daten-Spezifikation *Data* zwei “Kopien” erstellt, die zu einer neuen Daten-Spezifikation *Data'* vereinigt werden:

$$Data' ::= Data \uplus Data$$

Zur besseren Unterscheidung der beiden Spezifikations-Kopien soll die folgende Namenskonvention gelten: Für jeden Datentyp t heißt die zugehörige Sorte für die Schema-Werte SV_t ⁸² und die zugehörige Sorte für die Ausprägungs-Werte IV_t ⁸³; ist in einer Spezifikation nur eine Kopie der Spezifikation *Data* enthalten, wird die Werte-Sorte V_t ⁸⁴ genannt. Die Namen der Operationen werden sowohl um den Namen des primitiven Datentyps, auf dem sie arbeiten, als auch um einen zusätzlichen Index S für Schema bzw. I für Ausprägung ergänzt, etwa $+INTEGER,S$ für die Addition von Ganzzahlen im Schema oder $+STRING,I$ für die Verkettung von Zeichenketten in der Ausprägung.⁸⁵

Genauso wie Objekte und Verknüpfungen der Ausprägung in Klassen und Assoziationen des Schemas typisiert sind, werden auch Werte der Ausprägung in Werten des Schemas durch geeignete Typ-Operationen typisiert. Weiter wird das Modell um zusätzliche Attribut-Sorten und Attribut-Operationssymbole erweitert, um die Beziehung zwischen Modell-Elementen (hierzu gehören Klassen und Objekte) und Werten herzustellen, genauso wie zwischen Objekten und Klassen durch Verknüpfungen und Assoziationen Beziehungen hergestellt werden.⁸⁶ Schließlich wird jeder Klasse für jeden primitiven Datentyp t ein von ihr ausgehendes Attribut *initCA t* zugeordnet, um später in Transformationen Hüllenklassen und somit Attribute aus Klassen entfalten zu können. Analog wird auf der Ausprägungsebene jedem Objekt für jeden primitiven Datentyp t ein Attribut *initOA t* zugeordnet.

Sei im Folgenden

⁸¹ Ein solches System hat für jeden primitiven Datentyp nur einen einzigen Wert zur Verfügung!

⁸² für *Schema Value* (dt. Schema-Wert)

⁸³ für *Instance Value* (dt. Ausprägungs-Wert)

⁸⁴ für *Value* (dt. Wert)

⁸⁵ Beide Indizes sind nicht zwingend notwendig, da durch den Namen sowie die Quell- und Zielsorten die Operationssymbole eindeutig bestimmt ist, erleichtern dem Leser jedoch die Zuordnung der Operation durch das Verringern des zum Verständnis erforderlichen Kontextes erheblich.

⁸⁶ Assoziationen und Verknüpfungen werden nicht attribuiert, weil dies für die Umsetzung des konzeptionellen Modells nicht benötigt wird.

$$(8.1) \quad TYPES ::= \{ BOOLEAN, INTEGER, CHARACTER, STRING \}$$

die Menge der unterstützten primitiven Datentypen. Dann werden dem Modell für jeden Typ $t \in TYPES$ folgende Sorten und Operationssymbole hinzugefügt:

- eine Sorte $C(lass)A(tribute)_t$ für Klassen-Attribute;
- eine Sorte $O(bject)A(tribute)_t$ für Objekt-Attribute;
- ein Operationssymbol $typeV_t: IV_t \rightarrow SV_t$, das Ausprägungs-Werte in Schema-Werten typisiert;
- ein Operationssymbol $typeOA_t: OA_t \rightarrow CA_t$, das Objekt-Attribute in Klassen-Attributen typisiert;
- zwei Operationssymbole $sourceCA_t: CA_t \rightarrow C$ und $targetCA_t: CA_t \rightarrow SV_t$, die eine Klasse über ein Klassen-Attribut mit einem Schema-Wert verbinden;
- zwei Operationssymbole $sourceOA_t: OA_t \rightarrow O$ und $targetOA_t: OA_t \rightarrow IV_t$, die ein Objekt über ein Objekt-Attribut mit einem Ausprägungs-Wert verbinden;
- ein Operationssymbol $initCA_t: C \rightarrow CA_t$, das eine Klasse über ein Klassen-Attribut mit einem Schema-Wert verbindet; und
- ein Operationssymbol $initOA_t: O \rightarrow OA_t$, das ein Objekt über ein Objekt-Attribut mit einem Schema-Wert verbindet.

Zusammen ergibt sich das folgende Modell, wobei t über die Menge der unterstützten primitiven Datentypen $TYPES$ aus (8.1) iteriert:

$$M_5 ::= M_4 \uplus Data' \uplus$$

sorts

$$\begin{array}{ll} CA_t & (t\text{-Klassen-Attribute}) \\ OA_t & (t\text{-Objekt-Attribute}) \end{array}$$

opns

$$\begin{array}{ll} typeV_t: IV_t \rightarrow SV_t & (\text{Typisierung von } t\text{-Werten}) \\ typeOA_t: OA_t \rightarrow CA_t & (\text{Typisierung von } t\text{-Attributen}) \\ sourceCA_t: CA_t \rightarrow C & (\text{Quelle eines } t\text{-Klassen-Attributs}) \\ targetCA_t: CA_t \rightarrow SV_t & (\text{Ziel eines } t\text{-Klassen-Attributs}) \\ initCA_t: C \rightarrow CA_t & (\text{Initiales } t\text{-Klassen-Attribut}) \\ sourceOA_t: OA_t \rightarrow O & (\text{Quelle eines } t\text{-Objekt-Attributs}) \\ targetOA_t: OA_t \rightarrow IV_t & (\text{Ziel eines } t\text{-Objekt-Attributs}) \\ initOA_t: O \rightarrow OA_t & (\text{Initiales } t\text{-Objekt-Attribut}) \end{array}$$

axms

Typisierung ist homomorph

$$x \in OA_t : typeO(sourceOA_t(x)) = sourceCA_t(typeOA_t(x)) \quad (\text{bzgl. } sourceA_t)$$

$$x \in OA_t : typeV_t(targetOA_t(x)) = targetCA_t(typeOA_t(x)) \quad (\text{bzgl. } targetA_t)$$

$$x \in O : typeOA_t(initOA_t(x)) = initCA_t(typeO(x)) \quad (\text{bzgl. } initA_t)$$

init-Axiome

$$x \in C : sourceCA_t(initCA_t(x)) = x \quad (initCA\text{-Anfang})$$

$$x \in O : sourceOA_t(initOA_t(x)) = x \quad (initOA\text{-Anfang})$$

8.6 Operationen und Nachrichten

Die Modellierung von Operationen und Nachrichten sowie von Parametern und Argumenten ähnelt der Modellierung von Klassen und Objekten bzw. Assoziationen und Verknüpfungen. Denn so wie Objekte Ausprägungen von Klassen und Verknüpfungen Ausprägungen von Assoziationen sind, können Nachrichten als Ausprägungen von Operationen und Argumente als Ausprägungen von Parametern aufgefasst werden. Weil Klassen, Assoziationen, Objekte und Verknüpfungen durch unterschiedliche Sorten repräsentiert werden, ergibt sich ein erster Modellierungsansatz durch die Hinzunahme weiterer Sorten und Operationssymbole wie folgt:

$M_{6a} ::= M_5 +$

sorts

$Op(eration)$ (Operationen)

$M(essage)$ (Nachrichten)

$Param(eter)$ (Parameter)

$Arg(ument)$ (Argumente)

opns

$sourceParam : Param \rightarrow Op$ (Quelle eines Parameters)

$targetParam : Param \rightarrow C$ (Ziel eines Parameters)

$sourceArg : Arg \rightarrow M$ (Quelle eines Arguments)

$targetArg : Arg \rightarrow O$ (Ziel eines Arguments)

$typeM : M \rightarrow Op$ (Typisierung von Nachrichten)

$typeArg : Arg \rightarrow Param$ (Typisierung von Argumenten)

axms

Typisierung ist homomorph

$$x \in Arg : typeM(sourceArg(x)) = sourceParam(typeArg(x)) \quad (\text{bzgl. } source)$$

$$x \in Arg : typeO(targetArg(x)) = targetParam(typeArg(x)) \quad (\text{bzgl. } target)$$

Dieser Ansatz ist jedoch problematisch, weil Parameter nicht auf Operationen und Argumente nicht auf Nachrichten verweisen können. Dies ist jedoch für die Fallunterscheidung wichtig (5.2.2.2). Dieses Problem kann auf zweierlei Weise gelöst werden: Entweder gibt es jeweils zwei unterschiedliche Sorten für Daten-Parameter und Operations-Parameter bzw. Daten-Argumente und Nachrichten-Argumente, oder Klassen und Operationen sowie Objekte und Nachrichten werden jeweils in einer Sorte zusammengefasst, wobei über Prädikate zwischen ihnen unterschieden wird.

In dieser Arbeit wird der zweite Ansatz gewählt, der auch weitere Vorteile mit sich bringt (s. u.). Somit werden Operationen und Nachrichten in den vorhandenen Sorten C (Operationen) und O (Nachrichten) als Knoten modelliert. Aus Symmetriegründen werden auch Parameter und Argumente durch Elemente zu den Sorten A und L als Kanten von einer Operation bzw. Nachricht zu einem anderen Element auf der Schema- bzw. Ausprägungs-Ebene dargestellt. Um Operationen und Nachrichten von Klassen und Objekten unterscheiden zu können, werden die Prädikate *operation* und *message* eingeführt, welche die Software (Prädikate sind erfüllt) von der Datenbank (Prädikate sind nicht erfüllt) abgrenzen. Parameter und Argumente werden über eine Konvention von Assoziationen und Verknüpfungen unterschieden: Geht die Kante von einem Knoten aus, für den das Prädikat *operation* bzw. *message* gültig ist, dann handelt es sich bei der Kante um einen Parameter bzw. ein Argument, andernfalls um eine Assoziation bzw. eine Verknüpfung.

Dies führt zu der folgenden Erweiterung der Spezifikation:

$M_6 ::= M_5 +$
prds
operation: C (Operationen)
message: O (Nachrichten)
axms

Typisierung ist homomorph

$x \in O : message(x) \Rightarrow operation(typeO(x))$ (bzgl. Nachrichten)

Neben der Symmetrie hat das Zusammenfassen von Assoziationen und Parametern bzw. Verknüpfungen und Argumenten einen weiteren Vorteil: Durch die Wiederverwendung des Zu-höchstens-eins-Axioms für Argumente und Parameter kann verhindert werden, dass ein Parameter mit mehr als einem Argument belegt wird.

Schließlich erfordert die Operationsstruktur im konzeptionellen Modell die Fähigkeit, Gemeinsamkeiten zu abstrahieren, etwa die *next*-Verbindung, deren Ausprägungen verwendet werden, um die Abarbeitungs-Reihenfolge der Nachrichten festzulegen, oder die *current*-Beziehung, die einem Ablauffaden die nächste zu verarbeitende Nachricht zuordnet. Um dies zu erreichen, wird die Vererbung, die schon zwischen Klassen und zwischen Objekten existiert, auf Operationen und Nachrichten ausgeweitet. Operationen spezialisieren folglich einander, wenn sie über die

underC-Relation miteinander in Beziehung gesetzt werden. Analog spezialisieren Nachrichten einander, wenn sie über die *underO*-Relation miteinander in Beziehung gesetzt werden. Die Möglichkeit der Wiederverwendung der *underC*- und *underO*-Prädikate für Operationen und Nachrichten ist somit ein weiterer Vorteil des Zusammenfassens von Klassen und Operationen sowie Objekten und Nachrichten in jeweils einer Sorte.⁸⁷

8.7 System-Spezifikation

Dieser Abschnitt fasst die Ergebnisse der letzten Abschnitte zusammen (t iteriert hier über die Menge der unterstützten primitiven Datentypen *TYPES* aus (8.1)):

Definition 8.6 (Spezifikation M). Die Spezifikation M ist wie folgt definiert:

$$M ::= M_6 = \text{Data}' \uplus$$

sorts

C	(Klassen)
A	(Assoziationen)
O	(Objekte)
L	(Verknüpfungen)
CA_t	(t -Klassen-Attribute)
OA_t	(t -Objekt-Attribute)

opns

$\text{type}O: O \rightarrow C$	(Typisierung von Objekten)
$\text{type}L: L \rightarrow A$	(Typisierung von Verknüpfungen)
$\text{type}V_t: IV_t \rightarrow SV_t$	(Typisierung von t -Werten)
$\text{type}OA_t: OA_t \rightarrow CA_t$	(Typisierung von t -Attributen)
$\text{source}A: A \rightarrow C$	(Quelle einer Assoziation)
$\text{target}A: A \rightarrow C$	(Ziel einer Assoziation)
$\text{source}L: L \rightarrow O$	(Quelle einer Verknüpfung)
$\text{target}L: L \rightarrow O$	(Ziel einer Verknüpfung)
$\text{source}CA_t: CA_t \rightarrow C$	(Quelle eines t -Klassen-Attributs)
$\text{target}CA_t: CA_t \rightarrow SV_t$	(Ziel eines t -Klassen-Attributs)
$\text{init}CA_t: C \rightarrow CA_t$	(Initiales t -Klassen-Attribut)
$\text{source}OA_t: OA_t \rightarrow O$	(Quelle eines t -Objekt-Attributs)

⁸⁷ Die vorgestellte Spezifikation hat jedoch den Nachteil, dass Operationen und Klassen bzw. Nachrichten und Objekte in ungewünschten Kombinationen auftreten, etwa dass eine Klasse spezieller als eine Operation ist oder umgekehrt. Ebenso können Klassen und Operationen bzw. Nachrichten und Objekte via *relC* bzw. *relO* zu einer Komponente zusammengeschlossen werden. Es bleibt zu untersuchen, wie weitere Axiome diese Fälle ausschließen können und ob solche Axiome sich mit den restlichen Ergebnissen dieser Arbeit vertragen.

$targetOA_t: OA_t \rightarrow IV_t$	(Ziel eines t -Objekt-Attributs)
$initOA_t: O \rightarrow OA_t$	(Initiales t -Objekt-Attribut)
$selfC: C \rightarrow A$	(Schlaufe einer Klasse)
$selfO: O \rightarrow L$	(Schlaufe eines Partikels)

prds

$operation: C$	(Operationen)
$message: O$	(Nachrichten)
$underC: C C$	(Unterklasse von)
$underO: O O$	(Unterobjekt von)
$relC: C C$	(Verwandte Klassen)
$relO: O O$	(Verwandte Objekte)

axms**Typisierung ist homomorph**

(M.1)	$x \in L : typeO(sourceL(x)) = sourceA(typeL(x))$	(bzgl. $source$)
(M.2)	$x \in L : typeO(targetL(x)) = targetA(typeL(x))$	(bzgl. $target$)
(M.3)	$x \in OA_t : typeO(sourceOA_t(x)) = sourceCA_t(typeOA_t(x))$	(bzgl. $sourceA_t$)
(M.4)	$x \in OA_t : typeV_t(targetOA_t(x)) = targetCA_t(typeOA_t(x))$	(bzgl. $targetA_t$)
(M.5)	$x \in O : typeOA_t(initOA_t(x)) = initCA_t(typeO(x))$	(bzgl. $initA_t$)
(M.6)	$x \in O : typeL(selfO(x)) = selfC(typeO(x))$	(bzgl. $self$)
(M.7)	$x \in O : message(x) \Rightarrow operation(typeO(x))$	(bzgl. Nachrichten)
(M.8)	$x, y \in O : underO(x, y) \Rightarrow underC(typeO(x), typeO(y))$	(bzgl. Unterobjekten)
(M.9)	$x, y \in O : relO(x, y) \Rightarrow relC(typeO(x), typeO(y))$	(bzgl. Komponenten)

self- und init-Kanten

(M.10)	$x \in C : sourceA(selfC(x)) = x$	($selfC$ -Anfang)
(M.11)	$x \in C : targetA(selfC(x)) = x$	($selfC$ -Ende)
(M.12)	$x \in C : sourceCA_t(initCA_t(x)) = x$	($initCA$ -Anfang)
(M.13)	$x \in O : sourceL(selfO(x)) = x$	($selfO$ -Anfang)
(M.14)	$x \in O : targetL(selfO(x)) = x$	($selfO$ -Ende)
(M.15)	$x \in O : sourceOA_t(initOA_t(x)) = x$	($initOA$ -Anfang)

Vererbung

(M.16)	$x \in C : underC(x, x)$	(Reflexivität)
--------	--------------------------	----------------

(M.17) $x, y \in C : \text{under}C(x, y) \wedge \text{under}C(y, x) \Rightarrow x = y$ (Antisymmetrie)

(M.18) $x, y, z \in C : \text{under}C(x, y) \wedge \text{under}C(y, z) \Rightarrow \text{under}C(x, z)$ (Transitivität)

(M.19) $x \in O : \text{under}O(x, x)$ (Reflexivität)

(M.20) $x, y \in O : \text{under}O(x, y) \wedge \text{under}O(y, x) \Rightarrow x = y$ (Antisymmetrie)

(M.21) $x, y, z \in O : \text{under}O(x, y) \wedge \text{under}O(y, z) \Rightarrow \text{under}O(x, z)$ (Transitivität)

Komponenten

(M.22) $x, y \in C : \text{rel}C(x, y) \Rightarrow \text{rel}C(y, x)$ (Symmetrie)

(M.23) $x, y, z \in C : \text{rel}C(x, y) \wedge \text{rel}C(y, z) \Rightarrow \text{rel}C(x, z)$ (Transitivität)

(M.24) $x, y \in O : \text{rel}O(x, y) \Rightarrow \text{rel}O(y, x)$ (Symmetrie)

(M.25) $x, y, z \in O : \text{rel}O(x, y) \wedge \text{rel}O(y, z) \Rightarrow \text{rel}O(x, z)$ (Transitivität)

(M.26) $x, y \in C : \text{under}C(x, y) \Rightarrow \text{rel}C(x, y)$ (Schema-Komponenten)

(M.27) $x, y \in O : \text{under}O(x, y) \Rightarrow \text{rel}O(x, y)$ (Ausprägungs-Komponenten)

(M.28) $x, y \in O : \text{rel}O(x, y) \wedge \text{type}O(x) = \text{type}O(y) \Rightarrow x = y$ (Eindeutige Komponenten)

Verknüpfungen

(M.29) $x, y \in L : \text{source}L(x) = \text{source}L(y) \wedge \text{type}L(x) = \text{type}L(y) \Rightarrow x = y$ (Zu-höchstens-eins-Beziehungen)

Kategorielle Strukturen, Übergänge und Eigenschaften

Dieses Kapitel definiert in Abschnitt 9.1 die für die weitere Betrachtung wesentlichen Spezifikationen MP und M' und zeigt die Isomorphie der daraus entstehenden Kategorien $\mathbf{Alg}(MP)^2$ und $\mathbf{Alg}(M')$.⁸⁸ Abschnitt 9.2 behandelt im Detail den Übergang von der Kategorie $\mathbf{Alg}(M')$ in die Kategorie $\mathbf{Alg}(M)$. Zum einen wird gezeigt, dass nur ein kleiner Teil der Axiome bei der Konstruktion der Epireflexion betrachtet werden muss. Zum anderen wird nachgewiesen, dass die Konstruktion das Schema unverändert lässt. Schließlich werden in Abschnitt 9.3 für die nachfolgenden Sätze benötigten Eigenschaften von Pullbacks und Pushouts nachgewiesen.

9.1 Die Kategorien $\mathbf{Alg}(MP)$ und $\mathbf{Alg}(M')$

Betrachtet man die Spezifikation M genauer, so fällt auf, dass viele Sorten, Operationssymbole, Prädikate und Axiome “doppelt” vorhanden sind, nämlich einmal zur Modellierung des Schemas und einmal zur Modellierung der Ausprägung. Dieser Abschnitt setzt an eben diesem Punkt an und versucht, diese gemeinsame Struktur von Schema und Ausprägung mathematisch zu erfassen. Dazu wird in einem ersten Schritt die Spezifikation M in die drei Aspekte Schema, Ausprägung und Typisierung aufgespalten. Die Objekte der sich ergebenden Kategorie $\mathbf{Alg}(MP)$ sind entweder Schemata oder Ausprägungen, zu den Morphismen gehören unter anderem alle Typisierungen und alle Schema- bzw. Ausprägungs-Transformationen. Der Vorteil dieser Kategorie liegt darin, den Schema- und den Ausprägungsanteil von Welt-Transformationen getrennt durch einzelne Morphismen darzustellen. Es ergibt sich somit:

Definition 9.1 (Spezifikation MP). *Die Spezifikation MP ist wie folgt definiert (t iteriert hier über die Menge der unterstützten primitiven Datentypen $TYPES$ aus (8.1)):*

$$MP = \text{Data} \uplus$$

sorts	
	(Knoten)
$N(\text{odes})$	

⁸⁸ $\mathbf{Alg}(MP)^2$ ist die Pfeilkategorie über $\mathbf{Alg}(MP)$.

$E(dges)$ (Kanten)
 $N(ode)A(tributes)_t$ (t -Knoten-Attribute)

opns

$source: E \rightarrow N$ (Quelle einer Kante)
 $target: E \rightarrow N$ (Ziel einer Kante)
 $sourceA: NA_t \rightarrow N$ (Quelle eines t -Knoten-Attributs)
 $targetA: NA_t \rightarrow V_t$ (Ziel eines t -Knoten-Attributs)
 $initA_t: N \rightarrow NA_t$ (Initiales t -Knoten-Attribut)
 $self: N \rightarrow E$ (Schleufe eines Knotens)

prds

$software: N$ (Software-Knoten)
 $under: N N$ (Unterknoten von)
 $rel: N N$ (Verwandte Knoten)

axms***self*- und *init*-Kanten**

- (MP.1) $x \in N : source(self(x)) = x$ (*self*-Anfang)
(MP.2) $x \in N : target(self(x)) = x$ (*self*-Ende)
(MP.3) $x \in N : sourceA_t(initA_t(x)) = x$ (*initA*-Anfang)

Vererbung

- (MP.4) $x \in N : under(x, x)$ (Reflexivität)
(MP.5) $x, y \in N : under(x, y) \wedge under(y, x) \Rightarrow x = y$ (Antisymmetrie)
(MP.6) $x, y, z \in N : under(x, y) \wedge under(y, z) \Rightarrow under(x, z)$ (Transitivität)

Komponenten

- (MP.7) $x, y \in N : rel(x, y) \Rightarrow rel(y, x)$ (Symmetrie)
(MP.8) $x, y, z \in N : rel(x, y) \wedge rel(y, z) \Rightarrow rel(x, z)$ (Transitivität)
(MP.9) $x, y \in N : under(x, y) \Rightarrow rel(x, y)$ (Komponenten)

Verglichen mit der Spezifikation M beschränkt sich die Spezifikation MP auf einen *Teil* von Systemen – entweder auf das Schema oder dessen Ausprägung. Somit hat sich die Anzahl der Sorten, Operationssymbole, Prädikate und Axiome annähernd halbiert. Weiterhin fehlen in MP konsequenterweise die Typisierungen sowie die Axiome, die sich auf Schema *und* Ausprägung beziehen; dies umfasst die Axiom-Gruppe *Typisierung ist homomorph* sowie die Axiome (M.28) und (M.29).

Allerdings sind die Typisierungen gerade so spezifiziert worden, dass sie als Homomorphismen zwischen Ausprägung und Schema angesehen werden können. Unter Ausblendung der Axiome (M.28) und (M.29) repräsentiert ein Morphismus in $\mathbf{Alg}(MP)$ (Typisierung) zusammen mit seiner Quelle (Ausprägung) und seinem Ziel (Schema) somit genau ein Objekt in $\mathbf{Alg}(M)$, welches ebenfalls aus Ausprägung, Schema und Typisierung besteht. Um dies zu formalisieren, wird in einem ersten Schritt eine Spezifikation M' definiert, die der Spezifikation M ohne die Axiome (M.28) und (M.29) entspricht. In einem zweiten Schritt wird gezeigt, dass die Kategorie von M' -Systemen und M' -Homomorphismen zu der Pfeilkategorie $\mathbf{Alg}(MP)^2$ isomorph ist.

Definition 9.2 (Spezifikation M'). Die Spezifikation M' ist definiert als die Spezifikation M aus Def. 8.6 ohne die Axiome (M.28) und (M.29).

Zuerst wird ein Paar von Zuordnungen $(\mathcal{D}_{\text{Ob}}, \mathcal{D}_{\text{Mor}})$ definiert, die einem $\mathbf{Alg}(MP)$ -Morphismus ein passendes M' -System und einem $\mathbf{Alg}(MP)^2$ -Morphismus einen passenden M' -Homomorphismus zuordnet:

Konstruktion 9.3 (Die Zuordnungen \mathcal{D}_{Ob} und \mathcal{D}_{Mor}). Seien $type_I: I \rightarrow S$ ein $\mathbf{Alg}(MP)^2$ -Objekt und $(m: I^1 \rightarrow I^2, r: S^1 \rightarrow S^2): type_{I^1} \rightarrow type_{I^2}$ ein $\mathbf{Alg}(MP)^2$ -Morphismus.⁸⁹ Dann wird eine Objekt-Zuordnung

$$\mathcal{D}_{\text{Ob}}: \text{Ob}^{\mathbf{Alg}(MP)^2} \rightarrow \text{Ob}^{\mathbf{Alg}(M')}$$

und eine Morphismen-Zuordnung

$$\mathcal{D}_{\text{Mor}}: \text{Mor}^{\mathbf{Alg}(MP)^2} \rightarrow \text{Mor}^{\mathbf{Alg}(M')}$$

folgendermaßen definiert (der Index t iteriert hier über die Menge der unterstützten primitiven Datentypen $TYPES$ aus (8.1)):

- $\mathcal{D}_{\text{Ob}}(type_I: I \rightarrow S) ::= D \in \text{Ob}^{\mathbf{Alg}(M')}$ mit:

$$D_C ::= S_N$$

$$D_A ::= S_E$$

$$D_O ::= I_N$$

$$D_L ::= I_E$$

$$D_{SV_t} ::= S_{V_t}$$

$$D_{IV_t} ::= I_{V_t}$$

$$D_{CA_t} ::= S_{NA_t}$$

$$D_{OA_t} ::= I_{NA_t}$$

$$operation^D ::= software^S$$

$$message^D ::= software^I$$

$$typeO^D: D_O \rightarrow D_C ::= type_{I,N}$$

$$typeL^D: D_L \rightarrow D_A ::= type_{I,E}$$

$$typeV_t^D: D_{IV_t} \rightarrow D_{SV_t} ::= type_{I,V_t}$$

$$typeOA_t^D: D_{OA_t} \rightarrow D_{CA_t} ::= type_{I,NA_t}$$

$$sourceA^D: D_A \rightarrow D_C ::= source^S$$

$$targetA^D: D_A \rightarrow D_C ::= target^S$$

$$sourceL^D: D_L \rightarrow D_O ::= source^I$$

$$targetL^D: D_L \rightarrow D_O ::= target^I$$

$$sourceCA_t^D: D_{CA_t} \rightarrow D_C ::= sourceA_t^S$$

$$targetCA_t^D: D_{CA_t} \rightarrow D_{SV_t} ::= targetA_t^S$$

⁸⁹ m = Migration, r = Restrukturierung

$$\begin{array}{ll}
\text{under}C^D ::= \text{under}^S & \text{source}OA_t^D : D_{OA_t} \rightarrow D_O ::= \text{source}A_t^I \\
\text{under}O^D ::= \text{under}^I & \text{target}OA_t^D : D_{OA_t} \rightarrow D_{IV_t} ::= \text{target}A_t^I \\
\text{rel}C^D ::= \text{rel}^S & \text{init}CA_t^D : D_C \rightarrow D_{CA_t} ::= \text{init}A_t^S \\
\text{rel}O^D ::= \text{rel}^I & \text{init}OA_t^D : D_O \rightarrow D_{OA_t} ::= \text{init}A_t^I \\
\text{self}C^D : D_C \rightarrow D_A ::= \text{self}^S & \\
\text{self}O^D : D_O \rightarrow D_L ::= \text{self}^I &
\end{array}$$

- $\mathcal{D}_{\text{Mor}}((m, r)) ::= m;r : D^1 \rightarrow D^2 \in \text{Mor}^{\mathbf{Alg}(M^i)}(D^1, D^2)$ mit:

$$m;r(x) ::= \begin{cases} m(x) & \text{wenn } x \in D_O^1 \uplus D_L^1 \uplus D_{IV_t}^1 \uplus D_{OA_t}^1 \\ r(x) & \text{wenn } x \in D_C^1 \uplus D_A^1 \uplus D_{SV_t}^1 \uplus D_{CA_t}^1 \end{cases}$$

D ist ein M^i -System:

- Die M^i -Axiome (M.1), (M.2), (M.3), (M.4), (M.5), (M.6), (M.7), (M.8) und (M.9) sind in D gültig, weil type_I als MP -Homomorphismus operationsverträglich ist und die Wahrheit von Prädikaten überträgt.
- Die Gültigkeit der M^i -Axiome (M.10), (M.11), (M.12), (M.16), (M.17), (M.18), (M.22), (M.23) und (M.26) folgt aus der Gültigkeit der äquivalenten MP -Axiome (MP.1), (MP.2), (MP.3), (MP.4), (MP.5), (MP.6), (MP.7), (MP.8) und (MP.9) im MP -System S .
- Die Gültigkeit der M^i -Axiome (M.13), (M.14), (M.15), (M.19), (M.20), (M.21), (M.24), (M.25) und (M.27) folgt aus der Gültigkeit der äquivalenten MP -Axiome (MP.1), (MP.2), (MP.3), (MP.4), (MP.5), (MP.6), (MP.7), (MP.8) und (MP.9) im MP -System I .

$m;r$ ist ein M^i -Homomorphismus. Dies folgt aus der Homomorphie-Eigenschaft der MP -Homomorphismen m und r sowie aus der Gültigkeit der Gleichung $r \circ \text{type}_{I_1} = \text{type}_{I_2} \circ m$ in $\mathbf{Alg}(MP)$, die aus den Konstruktion der Pfeilkategorie folgt. \square

Umgekehrt kann aus jedem M^i -System das Schema, die Ausprägung und der Typisierungs-Homomorphismus extrahiert werden. Diese Aufspaltung sowie eine verträgliche Morphismus-Abbildung führt das Paar von Zuordnungen $(\mathcal{P}_{\text{Ob}}, \mathcal{P}_{\text{Mor}})$ durch:

Konstruktion 9.4 (Die Zuordnungen \mathcal{P}_{Ob} und \mathcal{P}_{Mor}). Seien D^1, D^2 zwei $\mathbf{Alg}(M^i)$ -Objekte und $m;r : D^1 \rightarrow D^2$ ein $\mathbf{Alg}(M^i)$ -Morphismus. Dann wird eine Objekt-Zuordnung

$$\mathcal{P}_{\text{Ob}} : \text{Ob}^{\mathbf{Alg}(M^i)} \rightarrow \text{Ob}^{\mathbf{Alg}(MP)^2}$$

und eine Morphismen-Zuordnung

$$\mathcal{P}_{\text{Mor}} : \text{Mor}^{\mathbf{Alg}(M^i)} \rightarrow \text{Mor}^{\mathbf{Alg}(MP)^2}$$

folgendermaßen definiert (der Index t iteriert hier über die Menge der unterstützten primitiven Datentypen $TYPES$ aus (8.1)):

- $\mathcal{P}_{\text{Ob}}(D^1) ::= \text{type}_{I^1} : I^1 \rightarrow S^1 \in \text{Ob}^{\mathbf{Alg}(MP)^2}$ mit:

$$\begin{array}{ll}
 I_N^1 ::= D_O^1 & S_N^1 ::= D_C^1 \\
 I_E^1 ::= D_L^1 & S_E^1 ::= D_A^1 \\
 I_{V_t}^1 ::= D_{IV_t}^1 & S_{V_t}^1 ::= D_{SV_t}^1 \\
 I_{NA_t}^1 ::= D_{OA_t}^1 & S_{NA_t}^1 ::= D_{CA_t}^1 \\
 \text{source}^{I^1} : I_E^1 \rightarrow I_N^1 ::= \text{source}L^{D^1} & \text{source}^{S^1} : S_E^1 \rightarrow S_N^1 ::= \text{source}A^{D^1} \\
 \text{target}^{I^1} : I_E^1 \rightarrow I_N^1 ::= \text{target}L^{D^1} & \text{target}^{S^1} : S_E^1 \rightarrow S_N^1 ::= \text{target}A^{D^1} \\
 \text{source}A_t^{I^1} : I_{NA_t}^1 \rightarrow I_N^1 ::= \text{source}OA_t^{D^1} & \text{source}A_t^{S^1} : S_{NA_t}^1 \rightarrow S_N^1 ::= \text{source}CA_t^{D^1} \\
 \text{target}A_t^{I^1} : I_{NA_t}^1 \rightarrow I_{V_t}^1 ::= \text{target}OA_t^{D^1} & \text{target}A_t^{S^1} : S_{NA_t}^1 \rightarrow S_{V_t}^1 ::= \text{target}CA_t^{D^1} \\
 \text{init}A_t^{I^1} : I_N^1 \rightarrow I_{NA_t}^1 ::= \text{init}OA_t^{D^1} & \text{init}A_t^{S^1} : S_N^1 \rightarrow S_{NA_t}^1 ::= \text{init}CA_t^{D^1} \\
 \text{self}^{I^1} : I_N^1 \rightarrow I_E^1 ::= \text{self}C^{D^1} & \text{self}^{S^1} : S_N^1 \rightarrow S_E^1 ::= \text{self}O^{D^1} \\
 \text{software}^{I^1} ::= \text{message}^{D^1} & \text{software}^{S^1} ::= \text{operation}^{D^1} \\
 \text{under}^{I^1} ::= \text{under}O^{D^1} & \text{under}^{S^1} ::= \text{under}C^{D^1} \\
 \text{rel}^{I^1} ::= \text{rel}O^{D^1} & \text{rel}^{S^1} ::= \text{rel}C^{D^1} \\
 \\
 \text{type}_{I^1} ::= \text{type}O^{D^1} \cup \text{type}L^{D^1} \cup \text{type}V_t^{D^1} \cup \text{type}OA_t^{D^1} &
 \end{array}$$

- $\mathcal{P}_{\text{Ob}}(D^2) ::= \text{type}_{I^2} : I^2 \rightarrow S^2 \in \text{Ob}^{\mathbf{Alg}(MP)^2}$, wobei type_{I^2} , I^2 und S^2 analog zu type_{I^1} , I^1 und S^1 konstruiert werden.
- $\mathcal{P}_{\text{Mor}}(m;r) ::= (m : I^1 \rightarrow I^2, r : S^1 \rightarrow S^2) : \text{type}_{I^1} \rightarrow \text{type}_{I^2} \in \text{Mor}^{\mathbf{Alg}(MP)^2}(\text{type}_{I^1}, \text{type}_{I^2})$ mit:

$$\begin{array}{l}
 m ::= m;r \Big|_{D_O^1 \uplus D_L^1 \uplus D_{IV_t}^1 \uplus D_{OA_t}^1} \\
 r ::= m;r \Big|_{D_C^1 \uplus D_A^1 \uplus D_{SV_t}^1 \uplus D_{CA_t}^1}
 \end{array}$$

$I^1 \xrightarrow{\text{type}_{I^1}} S^1$ ist ein $\mathbf{Alg}(MP)^2$ -Objekt:

- I^1 ist ein MP -System. Denn die Gültigkeit der MP -Axiome (MP.1), (MP.2), (MP.3), (MP.4), (MP.5), (MP.6), (MP.7), (MP.8) und (MP.9) folgt aus der Gültigkeit der äquivalenten M' -Axiome (M.13), (M.14), (M.15), (M.19), (M.20), (M.21), (M.24), (M.25) und (M.27) im M' -System D^1 .
- S^1 ist ein MP -System. Denn die Gültigkeit der MP -Axiome (MP.1), (MP.2), (MP.3), (MP.4), (MP.5), (MP.6), (MP.7), (MP.8) und (MP.9) folgt aus der Gültigkeit der äquivalenten M' -Axiome (M.10), (M.11), (M.12), (M.16), (M.17), (M.18), (M.22), (M.23) und (M.26) im M' -System D^1 .

- $type_{I^1}$ ist ein MP -Homomorphismus. Dies folgt aus der Gültigkeit der Axiome (M.1), (M.2), (M.3), (M.4), (M.5), (M.6), (M.7), (M.8) und (M.9) im M' -System D^1 .

$I^2 \xrightarrow{type_{I^2}} S^2$ ein $\mathbf{Alg}(MP)^2$ -Objekt. Die Begründung verläuft analog zu $I^1 \xrightarrow{type_{I^1}} S^1$.

(m, r) ist $\mathbf{Alg}(MP)^2$ -Morphismus:

- m und r sind MP -Homomorphismen. Dies folgt aus der Konstruktion der Morphismen m und r als Einschränkungen von $m;r$ sowie aus der Homomorphie von $m;r$.
- Die Gültigkeit der Gleichung $r \circ type_{I^1} = type_{I^2} \circ m$ in $\mathbf{Alg}(MP)$ folgt aus der Gültigkeit der M' -Axiome (M.1), (M.2), (M.3), (M.4), (M.5), (M.6), (M.7), (M.8) und (M.9) in D^1 und D^2 sowie aus der Homomorphie von $m;r$.

□

Diese Paare von Zuordnungen ergeben Funktoren, weil sie Identitäten übertragen und bezüglich der Komposition von Morphismen verträglich sind. Dies wird in den nächsten beiden Lemmata nachgewiesen.

Lemma 9.5. \mathcal{D}_{Ob} und \mathcal{D}_{Mor} ergeben einen Funktor $\mathcal{D}: \mathbf{Alg}(MP)^2 \rightarrow \mathbf{Alg}(M')$.

Beweis. Es gilt

$$\begin{aligned} \mathcal{D}_{\text{Mor}}((m', r') \circ (m, r)) &= \mathcal{D}_{\text{Mor}}((m' \circ m, r' \circ r)) \\ &= (m' \circ m); (r' \circ r) \\ &= m'; r' \circ m; r \\ &= \mathcal{D}_{\text{Mor}}((m', r')) \circ \mathcal{D}_{\text{Mor}}((m, r)) \end{aligned}$$

für je zwei $\mathbf{Alg}(MP)^2$ -Morphismen $(m, r): type \rightarrow type'$ und $(m', r'): type' \rightarrow type''$ sowie

$$\begin{aligned} \mathcal{D}_{\text{Mor}}(id_{type}) &= \mathcal{D}_{\text{Mor}}((id_I, id_S)) \\ &= id_I; id_S \\ &= id_{\mathcal{D}} \\ &= id_{\mathcal{D}_{\text{Ob}}(type)} \end{aligned}$$

für alle $\mathbf{Alg}(MP)^2$ -Identitäten $id_{type}: type \rightarrow type$. □

Lemma 9.6. \mathcal{P}_{Ob} und \mathcal{P}_{Mor} ergeben einen Funktor $\mathcal{P}: \mathbf{Alg}(M') \rightarrow \mathbf{Alg}(MP)^2$.

Beweis. Es gilt

$$\begin{aligned}
 \mathcal{P}_{\text{Mor}}(m'; r' \circ m; r) &= \mathcal{P}_{\text{Mor}}((m' \circ m); (r' \circ r)) \\
 &= (m' \circ m, r' \circ r) \\
 &= (m', r') \circ (m, r) \\
 &= \mathcal{P}_{\text{Mor}}(m'; r') \circ \mathcal{P}_{\text{Mor}}(m; r)
 \end{aligned}$$

für je zwei $\mathbf{Alg}(M')$ -Morphismen $m; r: D \rightarrow D'$ und $m'; r': D' \rightarrow D''$ sowie

$$\begin{aligned}
 \mathcal{P}_{\text{Mor}}(id_D) &= \mathcal{P}_{\text{Mor}}(id_I; id_S) \\
 &= (id_I, id_S) \\
 &= id_{\text{type}} \\
 &= id_{\mathcal{P}_{\text{Ob}}(D)}
 \end{aligned}$$

für alle $\mathbf{Alg}(M')$ -Identitäten $id_D: D \rightarrow D$. □

Offensichtlich sind die Funktoren \mathcal{D} und \mathcal{P} so konstruiert, dass sie zueinander invers sind. Das bedeutet, dass die beiden betrachteten Kategorien $\mathbf{Alg}(M')$ und $\mathbf{Alg}(MP)^2$ isomorph sind, wie der folgende Satz zeigt.

Satz 9.7. $\mathbf{Alg}(MP)^2$ ist isomorph zu $\mathbf{Alg}(M')$.

Beweis. Nach Lemma 9.5 und Lemma 9.6 sind \mathcal{D} und \mathcal{P} Funktoren. Offensichtlich gilt $\mathcal{D} \circ \mathcal{P} = 1_{\mathbf{Alg}(M')}$ und $\mathcal{P} \circ \mathcal{D} = 1_{\mathbf{Alg}(MP)^2}$. □

Die Kategorie $\mathbf{Alg}(MP)^2$ enthält folglich dieselben Objekte und Morphismen wie $\mathbf{Alg}(M')$, nur in einer anderen Darstellung. Zur Vereinfachung werden beide Darstellungen im Rahmen dieser Arbeit gleichgestellt, ohne die Funktoren \mathcal{D} und \mathcal{P} explizit zu notieren, etwa wenn von einem M -System $I \xrightarrow{\text{type}_I} S$ (korrekt ist: $\mathcal{D}(I \xrightarrow{\text{type}_I} S)$) die Rede ist.

Genauso wie nun $\mathbf{Alg}(M')$ Systeme enthält, die nicht in Ordnung sind, weil sie die Axiome (M.28) und (M.29) nicht berücksichtigen, so besitzt auch $\mathbf{Alg}(MP)^2$ "zu viele" Objekte. Eine Einschränkung auf die "richtigen" Systeme ist aus Symmetriegründen sinnvoll:

Definition 9.8 (Kategorie Sys). Sei $\mathcal{P}|_{\mathbf{Alg}(M)}$ die Einschränkung von \mathcal{P} auf die Unterkategorie $\mathbf{Alg}(M)$ von $\mathbf{Alg}(M')$. Dann ist die Kategorie **Sys** definiert als die Unterkategorie von $\mathbf{Alg}(MP)^2$, die durch das Bild von $\mathcal{P}|_{\mathbf{Alg}(M)}$ gegeben ist.

Satz 9.9. **Sys** ist isomorph zu $\mathbf{Alg}(M)$.

Beweis. Der Funktor \mathcal{P} ist voll, weil er Isomorphismus ist [2, Bemerkung 3.28 (2)]. Das stellt nach [2, Bemerkung 4.2 (3)] sicher, dass **Sys** überhaupt eine Kategorie ist. Die Isomorphie der beiden Kategorien folgt direkt aus der Isomorphie von \mathcal{P} . □

Die Isomorphie der beiden Kategorien $\mathbf{Alg}(M)$ und \mathbf{Sys} stellt sicher, dass ein System als eine in einem Schema korrekt typisierte Ausprägung angesehen werden kann. Wie bei den Kategorien $\mathbf{Alg}(MP)^2$ und $\mathbf{Alg}(M')$ wird auch von $\mathbf{Alg}(M)$ -Objekten und -Morphismen zu \mathbf{Sys} -Objekten und -Morphismen und zurück übergegangen, ohne die Funktoren \mathcal{D} und \mathcal{P} explizit zu benutzen.

Die präsentierten Kategorien enthalten Systeme zu *allen* möglichen Schemata. Gelegentlich ist es nützlich, das Schema festzuhalten und nur Systeme zu einem bestimmten Schema zu erlauben. Die so entstehenden Komma-Kategorien erleichtern zum einen die mathematische Formulierung von Programmen zu einem festen Schema (Kapitel 10). Zum anderen können System-Transformationen als Funktoren zwischen entsprechenden Komma-Kategorien zum Quell- und Ziel-Schema verstanden werden (Kapitel 11). Dies motiviert die folgende Definition:

Definition 9.10 (Kategorie $\mathbf{Sys}(S)$). *Sei $S \in \text{Ob}^{\mathbf{Alg}(MP)}$ ein Schema. Dann bezeichnet $\mathbf{Sys}(S)$ diejenige volle Unterkategorie von $\mathbf{Alg}(MP) \downarrow S$, deren Objekte auch zur Kategorie \mathbf{Sys} gehören. Die Kategorie $\mathbf{Sys}(S)$ enthält also alle M -Systeme zu einem festen Schema S .*

9.2 Der Übergang von $\mathbf{Alg}(M')$ zu $\mathbf{Alg}(M)$

Wie in den späteren Kapiteln deutlich wird, werden Transformationen und induzierte Migrationen von Systemen durch Konstruktionen in der Kategorie $\mathbf{Alg}(MP)^2$ definiert. Objekte dieser Kategorie entsprechen Objekten der Kategorie $\mathbf{Alg}(M')$ und erfüllen somit nicht zwangsläufig alle M -Axiome. Nach Satz A.79 gibt es für jedes M -System A eine Epireflexion $(\mathcal{F}^M(A) \in \text{Ob}^{\mathbf{Alg}(M)}, u_A: A \rightarrow \mathcal{F}^M(A) \in \text{Mor}^{\mathbf{Alg}(M)})$, die entsprechend Konstruktion A.92 konstruiert werden kann. Jedoch ist die Konstruktion umständlich, denn sie berücksichtigt *alle* Axiome der Spezifikation – und das sind immerhin 29. Ziel dieses Abschnitts ist es, die Konstruktion für den betrachteten Fall dahingehend zu vereinfachen, dass nur ein geringer Teil der Axiome – nämlich lediglich vier Axiome – berücksichtigt werden muss. Ein weiteres Ziel ist es nachzuweisen, dass die Epireflexion das Schema unverändert lässt. Schließlich wird gezeigt, dass in einem wichtigen Spezialfall die Anzahl der zu berücksichtigenden Axiome sich nochmal um die Hälfte reduzieren lässt.

Liegt ein M -System vor, dann sind die M -Axiome (M.28) und (M.29) nicht unbedingt in dem System gültig. Eine Faktorisierung nach einer Kongruenzrelation, die sich aus diesen beiden Axiomen allein ergibt, führt nicht notwendigerweise zu einem M -System, weil Klassen von Systemen zu einer Spezifikation mit positiven Horn-Formeln nicht unter Quotientenbildung abgeschlossen sind (vgl. [50, Bsp. 5.4]). Es ist sinnvoll zu überlegen, welche Axiome durch die Faktorisierung verletzt werden, für die also Variablenbelegungen existieren, welche die Prämisse wahr machen, ohne dass die Konklusion gilt:

- In der Konklusion des Axioms (M.28) wird die Gleichheit zweier Objekte gefordert, wenn sie sich in einer Komponente befinden und gleich typisiert sind. Weil nur bzgl. *relO* äquivalente

Partikel zusammengelegt werden, bleibt die Äquivalenzrelation gültig. Die Axiome (M.24) und (M.25) werden somit nicht ungültig. Anders sieht es mit der Vererbung aus: Werden zwei Klassen zusammengelegt, kann dies zur Folge haben, dass die Transitivität der Vererbungsverknüpfungen nicht mehr gewährleistet ist. Eine solche Situation ist in Abb. 9.1 dargestellt, in der die beiden Klassen C und D zusammengelegt werden, wobei \equiv die aus dem Axiom (M.29) entstehende erzeugte Kongruenz ist: Hier fehlt in der Ausprägung die Vererbungsbeziehung zwischen dem Partikel $2:B$ und dem Partikel $5:E$.

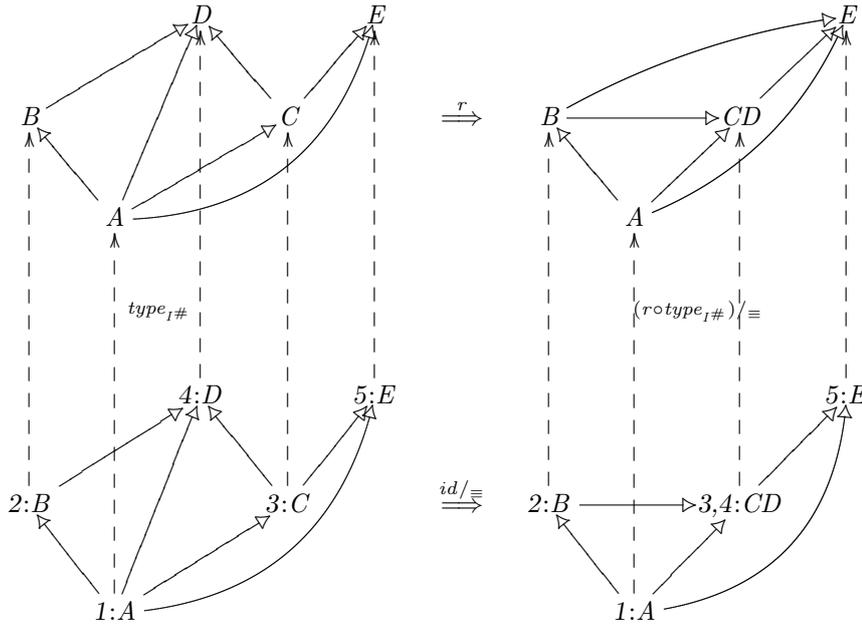


Abb. 9.1: Fehlender transitiver Abschluss nach Verschmelzung zweier Klassen

- Ähnlich gelagert ist der Fall bei Axiom (M.29): Durch das Zusammenlegen von zwei gleich typisierten Verknüpfungen mit gleicher Quelle werden auf Grund des Operationsabschlusses auch die Zielpartikel zusammengelegt. Diese Partikel können ursprünglich aber in unterschiedlichen Komponenten liegen und – auf der Ausprägungsebene – getrennten Vererbungsstrukturen angehören. Somit impliziert die Faktorisierung nach diesem Axiom eine Anpassung der Transitivität von sowohl der *relO*- als auch der *underO*-Relation.

Aus diesen Überlegungen heraus sollte es möglich sein, allein mit der Anpassung an die beiden Axiome (M.28) und (M.29), gefolgt von einem transitiven Abschluss der Relationen zu den Prädikaten *underO* und *relO* (Axiome (M.21) und (M.25)), auszukommen. Dazu wird die Spezifikation M'' definiert:

Definition 9.11 (Spezifikation M''). Sei $M = (\Sigma, H)$ die Spezifikation aus Def. 8.6. Sei $H' \subset H$ definiert als Menge von Axiomen, die lediglich die vier M -Axiome (M.21), (M.25),

(M.28) und (M.29) enthält. Dann ist die Spezifikation M'' definiert durch:

$$M'' ::= (\Sigma, H')$$

Sei Σ die Signatur, die den Spezifikationen M , M' und M'' zugrunde liegt. Es wird nun gezeigt, dass der Epirefektor $\mathcal{F}^{M''}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M'')$ aus Konstruktion A.92 die Gültigkeit der in M'' fehlenden M' -Axiome erhält und somit M' -Systeme auf M -Systeme abbildet:

Lemma 9.12. *Sei ein M' -System $D ::= I \xrightarrow{\text{type}_I} S$ gegeben, und sei $\mathcal{F}^{M''}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M'')$ der Epirefektor nach Konstruktion A.92. Dann sind in $\mathcal{F}_{\text{Ob}}^{M''}(D)$ die M' -Axiome (M.1) bis (M.9) gültig.*

Beweis. Siehe Abschnitt C.1.1. □

Lemma 9.13. *Sei ein M' -System $D ::= I \xrightarrow{\text{type}_I} S$ gegeben, und sei $\mathcal{F}^{M''}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M'')$ der Epirefektor nach Konstruktion A.92. Gelte $\mathcal{F}_{\text{Ob}}^{M''}(D) = I' \xrightarrow{\text{type}_{I'}} S'$. Dann gilt $S = S'$.*

Beweis. Siehe Abschnitt C.1.2. □

Lemma 9.14. *Sei ein M' -System $D ::= I \xrightarrow{\text{type}_I} S$ gegeben, und sei $\mathcal{F}^{M''}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M'')$ der Epirefektor nach Konstruktion A.92. Dann sind in $\mathcal{F}_{\text{Ob}}^{M''}(D)$ die M' -Axiome (M.10) bis (M.20), (M.22) bis (M.24) und (M.26) bis (M.27) gültig.*

Beweis. Siehe Abschnitt C.1.3. □

Satz 9.15. *Sei ein M' -System $D ::= I \xrightarrow{\text{type}_I} S$ gegeben, und sei $\mathcal{F}^{M''}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M'')$ der Epirefektor nach Konstruktion A.92. Dann ist $\mathcal{F}_{\text{Ob}}^{M''}(D)$ ein M -System.*

Beweis. Offensichtlich sind in $\mathcal{F}_{\text{Ob}}^{M''}(D)$ die M -Axiome (M.21), (M.25), (M.28) und (M.29) gültig, weil diese Axiome Teil der Spezifikation M'' sind. Die Gültigkeit der restlichen M -Axiome folgt aus Lemma 9.12 und Lemma 9.14. □

Eine weitere Vereinfachung der Konstruktion der Epireflexion ist möglich, wenn an den in einem M' -System enthaltenen Typisierung-Homomorphismus zusätzliche Bedingungen geknüpft werden. Die im Folgenden beschriebene Einschränkung wird in Abschnitt 11.3 benötigt, um dort die Transformation von Doppel-Pushout-Diagrammen entlang eines Refactorings zu ermöglichen.

Definition 9.16 (Spezifikation M^*). *Sei $M = (\Sigma, H)$ die Spezifikation aus Def. 8.6. Sei $H' \subset H$ definiert als Menge von Axiomen, die lediglich die zwei M -Axiome (M.21) und (M.28) enthält. Dann ist die Spezifikation M^* definiert durch:*

$$M^* ::= (\Sigma, H')$$

Satz 9.17. Sei ein M' -System $D ::= I^\# \xrightarrow{r^t \circ \text{type}_{I^\#}} S'$ gegeben, wobei $I^\# \xrightarrow{\text{type}_{I^\#}} S^\#$ ein M -System und $r^t: S^\# \rightarrow S'$ ein MP-Homomorphismus ist, der injektiv auf Nicht-self-Kanten ist, d. h. es gilt:

$$r_E^t(x) = r_E^t(y) \Rightarrow x = y \vee x, y \in \text{ran self}^{S^\#}$$

für alle $x, y \in S_E^\#$. Sei ferner $\mathcal{F}^{M^*}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M^*)$ der Epirefektor nach Konstruktion A.92. Dann gilt $\mathcal{F}_{\text{Ob}}^{M^*}(D) \cong \mathcal{F}_{\text{Ob}}^{M''}(D)$, und die Epireflexion kann folgendermaßen konstruiert werden:

- einmalige Faktorisierung nach dem M^* -Axiom (M.28) im ersten Schritt der Konstruktion, um die Eindeutigkeit der Elemente innerhalb einer Komponente zu gewährleisten;
- (unter Umständen) mehrmalige Faktorisierung nach dem M^* -Axiom (M.21) ab dem zweiten Schritt der Konstruktion, um den transitiven Abschluss der underO-Relation sicherzustellen.

Beweis. Siehe Abschnitt C.1.4. □

Schließlich kann der Epirefektor $\mathcal{F}^{M''}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M'')$ eingeschränkt auf die Kommakategorie $\mathbf{Alg}(MP) \downarrow S$ für ein Schema S betrachtet werden:

Satz 9.18. Es gibt einen Epirefektor $\mathcal{F}^S: \mathbf{Alg}(MP) \downarrow S \rightarrow \mathbf{Sys}(S)$.

Beweis. Der Funktor $\mathcal{F}^{M''}$ aus Satz 9.15 erfüllt die Anforderungen, weil das Schema nach Lemma 9.13 unverändert bleibt. □

Anmerkung 9.19 (Funktor-Vertauschbarkeit). Nach Satz 9.17 kann der Funktor \mathcal{F}^S aus Satz 9.18 für entsprechend strukturierte Systeme durch den Funktor \mathcal{F}^{M^*} ersetzt werden. □

9.3 Pullbacks, Pushouts und spezielle Homomorphismen

Dieser Abschnitt weist nach, dass Pullbacks und Pushouts in den betrachteten Kategorien spezielle Homomorphismen übertragen.

Lemma 9.20 (Pullbacks übertragen strikt volle Homomorphismen). Sei eine Signatur $\Sigma = (S, OP, P)$ gegeben und sei $\text{Spec} = (\Sigma, H)$ eine Spezifikation zu dieser Signatur. Sei $I \xleftarrow{l'} I' \xrightarrow{\text{type}_{I'}} S'$ Pullback von $I \xrightarrow{\text{type}_I} S \xleftarrow{l} S'$ in $\mathbf{Alg}(\text{Spec})$. Dann gilt: Wenn l strikt voll auf einem Prädikat $p \in P_w$ mit $w \in S^*$ ist, dann auch l' .

Beweis. Seien ein Prädikat $p \in P_w$ mit $w \in S^*$ sowie ein Tupel $x \in I'_w$ gegeben mit:

$$(9.1) \quad l'_w(x) \in p^I$$

Zu zeigen ist, dass gilt:

$$(9.2) \quad x \in p^{I'}$$

Aus (9.1) folgt:

$$(9.3) \quad \begin{array}{ll} \text{type}_{I,w}(l'_w(x)) \in p^S & (\text{type}_I \text{ ist Homomorphismus}) \\ \Rightarrow l_w(\text{type}_{I,w}(x)) \in p^S & (\text{Pullbacks kommutieren}) \end{array}$$

Nach Voraussetzung ist l strikt voll. Somit folgt aus (9.3):

$$(9.4) \quad \text{type}_{I,w}(x) \in p^{S'}$$

Nach Korollar A.87 folgt aus (9.1) und (9.4) die Wahrheit von (9.2). □

Lemma 9.21 (Pushouts übertragen strikt volle Homomorphismen). Sei $L \xrightarrow{m} G \xleftarrow{f} D$ Pushout von $L \xleftarrow{l} K \xrightarrow{k} D$ in $\mathbf{Alg}(MP_*)$. Dann gilt: Wenn l strikt voll auf einem Prädikat $p \in P_w$ mit $w \in S^*$ ist, dann auch f .

Beweis. Siehe Abschnitt C.1.5. □

Programme

Dieser Abschnitt formalisiert objektorientierte *Programme*. Dazu werden Programme in einzelne *Methoden* aufgeteilt. Jede Methode wiederum besteht aus einzelnen *Aktionen*, welche nicht mehr unterteilte Veränderung des Programm-Zustandes darstellen. Beispiele für Aktionen sind die Zuweisung eines Objektes oder Wertes zu einer Variable (5.2.2.1) oder das Versenden einer Nachricht zu einer Operation an ein Objekt (5.2.1.4).

Aktionen sind somit die elementaren Bauteile objektorientierter Programme. Werden Prozesse als spezielle Graphstrukturen aufgefasst, die den Zustand eines Programms darstellen, bietet es sich an, Programme und somit Aktionen als DPO-Regeln darzustellen. Leider können DPO-Regeln in den bisher betrachteten Kategorien nicht ohne Weiteres zu Graphstruktur-Transformationen fortgesetzt werden, denn dafür ist die Existenz und Eindeutigkeit des Pushout-Komplements bis auf Isomorphie eine wichtige Voraussetzung. Dies ist jedoch in Kategorien von Systemen und Homomorphismen zu einer Signatur mit Prädikaten im Allgemeinen nicht gegeben, wie das folgende Beispiel zeigt.

Beispiel 10.1 (Kein bis auf Isomorphie eindeutiges Pushout-Komplement). Sei eine Signatur mit einer Sorte S und einem einstelligen Prädikat p auf dieser Sorte gegeben. Sei eine Situation wie in Abb. 10.1a gegeben, wobei ausgefüllte Kreise das Prädikat p erfüllende Elemente bezeichnen. Dann gibt es zwei Möglichkeiten, ein Pushout-Komplement zu ergänzen (Abb. 10.1b und Abb. 10.1c), die nicht zueinander isomorph sind. Somit ist das Pushout-Komplement nicht bis auf Isomorphie eindeutig. \square

Anstatt nun eine geeignete Unterklasse von Monomorphismen \mathcal{M} zu suchen und nachzuweisen, dass die Kategorie $(\mathbf{Alg}(MP), \mathcal{M})$ eine adhäsive HLR-Kategorie darstellt, wird in dieser Arbeit ein anderer Weg gewählt: Die Spezifikation MP wird durch das Weglassen von Axiomen und Prädikaten so weit zu der Signatur MP_* vereinfacht, dass $\mathbf{Alg}(MP_*) \downarrow S$ eine adhäsive Kategorie ist und somit DPO-Graphstruktur-Transformationen in dieser Kategorie möglich sind. Dazu wird zuerst in Abschnitt 10.1 eine *Aktion* als DPO-Regel in der Kategorie $\mathbf{Alg}(MP_*)$ definiert. In Abschnitt 10.2 werden *konsistente* Aktionen eingeführt, welche Aktionen in zwei Punkten

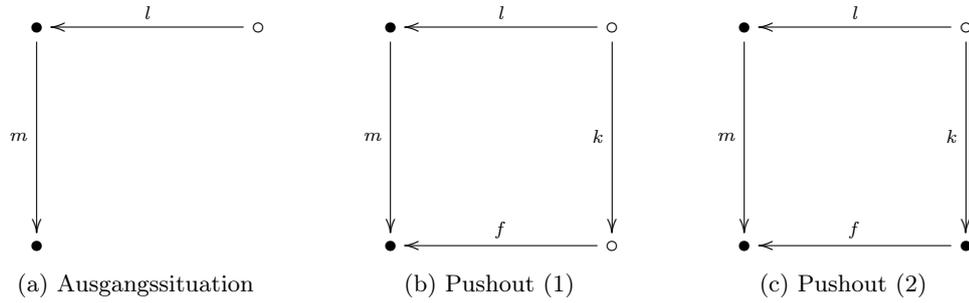


Abb. 10.1: Kein bis auf Isomorphie eindeutiges Pushout-Komplement

einschränken. Der Nachweis, dass die Anwendung von konsistenten Aktionen alle M -Axiome erhält und somit M -Modelle in M -Modelle transformiert, bildet das zentrale Ergebnis dieses Kapitels und rechtfertigt den Übergang von $\mathbf{Alg}(MP)$ zu $\mathbf{Alg}(MP_*)$. In Abschnitt 10.3 wird schließlich untersucht, welche Umstände die Wahl der korrekten Methode zu einer aktiven Nachricht bei der Programmausführung beeinflussen.

10.1 Aktionen und DPO-Transformationen

Um das Problem des nicht eindeutigen Pushout-Komplements zu lösen, wird die Spezifikation MP auf ihre algebraische Signatur beschränkt:

Definition 10.2 (Signatur MP_*). Sei Σ die Signatur der Spezifikation MP . Dann sei MP_* definiert als die Signatur (S_A, OP^A) der Spezifikation Σ^A aus Konstruktion A.94.

Anmerkung 10.3 (Signatur MP_).* Die Signatur MP_* ergibt sich aus der Spezifikation MP in drei Schritten:

- (1) Weglassen aller MP -Axiome.
- (2) Transformation der sich ergebenden Signatur in die Spezifikation Σ^A gemäß Konstruktion A.94, die jedes n -stellige Prädikat p über dem Sortenwort $w = s_1 s_2 \dots s_n$ in eine Sorte p , n Operationssymbole $\pi_p = (\pi_{p,i} : p \rightarrow s_i)_{1 \leq i \leq n}$ und eine positive Horn-Formel

$$x, y \in p : \pi_{p,1}(x) = \pi_{p,1}(y) \wedge \pi_{p,2}(x) = \pi_{p,2}(y) \wedge \dots \wedge \pi_{p,n}(x) = \pi_{p,n}(y) \Rightarrow x = y$$

umwandelt; jedes Operationssymbol ordnet dabei einem Prädikat-Element eines “seiner” Elemente zu.

- (3) Weglassen aller in Schritt 2 entstandenen positiven Horn-Formeln. □

MP_* ist eine Graphstruktur-Signatur, somit lässt sich das DPO-Modell zur Modellierung und Durchführung von Graphstruktur-Transformationen verwenden. Eine Aktion wird nun im mathematischen Modell als eine S -typisierte *Data*-attributierte DPO-Regel für ein festes Schema S dargestellt:

Definition 10.4 (Aktion). Sei S eine MP_* -Typ-Graphstruktur. Dann ist eine Aktion eine S -typisierte *Data*-attributierte DPO-Regel (Abb. 10.2), für die gilt, dass die S -typisierten *Data*-attribuierten Graphstrukturen $L \xrightarrow{type_L} S$, $K \xrightarrow{type_K} S$ und $R \xrightarrow{type_R} S$ M -Systeme sind.

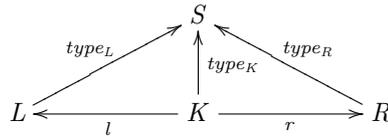


Abb. 10.2: Aktion als eine S -typisierte *Data*-attributierte DPO-Regel

Anmerkung 10.5 (Aktion). Obwohl in der Kategorie, in der eine Aktion formuliert ist, die M -Axiome “vergessen” sind, erfordert die obige Definition, dass diese Axiome gültig sind. Dies ist sinnvoll, weil eine Aktion M -Systeme in M -Systeme transformieren soll, dies aber sicherlich nur dann gewährleistet werden kann, wenn die Systeme innerhalb der Aktion selbst korrekte M -Systeme sind. □

Eine induzierte Graphstruktur-Transformation in der Kategorie $\mathbf{Alg}(MP_*) \downarrow S$ führt auf Grund der fehlenden Axiome nicht unbedingt zu MP - oder M -Systemen. Es gilt also zu untersuchen, welche Bedingungen Regel und Ansatz erfüllen müssen, damit die resultierende Graphstruktur-Transformation zu korrekt strukturierten typisierten Systemen führt. Es stellt sich heraus, dass weder das Bilden des Pushout-Komplements $D \xrightarrow{type_D} S$ auf der linken Seite der Regel noch das Bilden des Pushouts $H \xrightarrow{type_H} S$ auf der rechten Seite der Regel immer zu einem M -System führt, wie die folgenden Gegenbeispiele demonstrieren.

Beispiel 10.6 (Abschwächen der Wahrheit eines Prädikats). Sei das Schema wie in Abb. 10.3a gegeben. Die Regel in Abb. 10.3b entfernt eine abgeleitete, durch den transitiven Abschluss erzeugte Vererbungsverknüpfung durch Abschwächung der Wahrheit des Prädikats *underO*. Die Anwendung dieser Regel führt zu einem System, in dem das Axiom (M.21) nicht gilt (Abb. 10.3c). □

Beispiel 10.7 (Wahrmachen eines Prädikats). Sei das Schema wie in Abb. 10.4a gegeben. Die Regel in Abb. 10.4b verbindet zwei Partikel mit Hilfe einer Vererbungsverknüpfung. Die Anwendung auf eine Situation, in der die zwei Partikel bereits über eine Vererbungsverknüpfung miteinander in Beziehung stehen, führt zu einem System, in dem das Axiom (A.20) für das Prädikat *underO* nicht gilt, weil auf die beiden Partikel durch zwei Elemente zur Prädikat-Sorte *underO* verwiesen

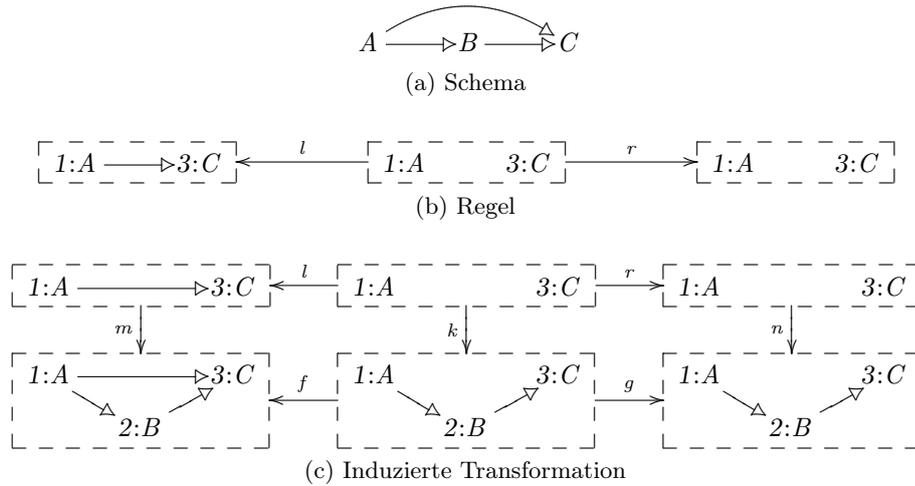


Abb. 10.3: Abschwächen der Wahrheit des Vererbungs-Prädikats

wird (Abb. 10.4c). Dies wird durch die Notation \Longrightarrow verdeutlicht. Eine analoge Situation entsteht beim Wahrmachen des Prädikats *message*. \square

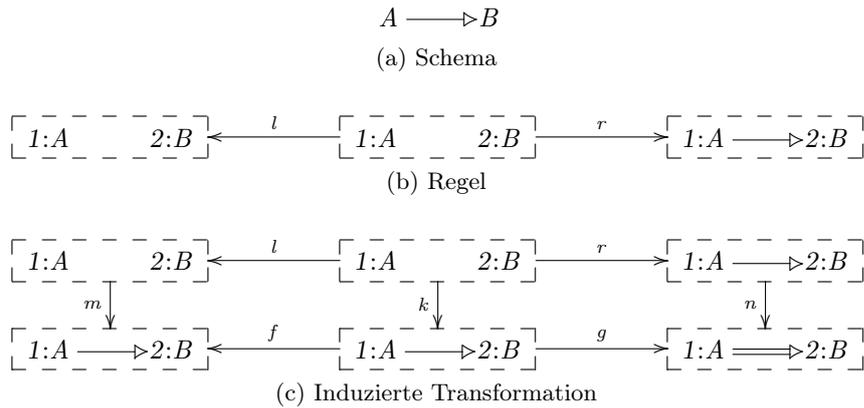


Abb. 10.4: Wahrmachen des Vererbungs-Prädikats

Beispiel 10.8 (Hinzufügen eines Oberobjekts). Sei das Schema wie in Abb. 10.5a gegeben. Die Regel in Abb. 10.5b fügt einem Partikel ein Oberobjekt zu. Die Anwendung auf eine Situation, in der das Partikel bereits ein Oberobjekt zum selben Typ besitzt, führt zu einem System, in dem das Axiom (M.28) nicht gilt (Abb. 10.5c). \square

Beispiel 10.9 (Hinzufügen eines Unterobjekts). Sei das Schema wie in Abb. 10.6a gegeben. Die Regel in Abb. 10.6b fügt einem Partikel ein Unterobjekt zu. Die Anwendung auf eine Situation, in der das Partikel bereits ein Unterobjekt zum selben Typ besitzt, führt zu einem System, in dem das Axiom (M.28) nicht gilt (Abb. 10.6c). \square

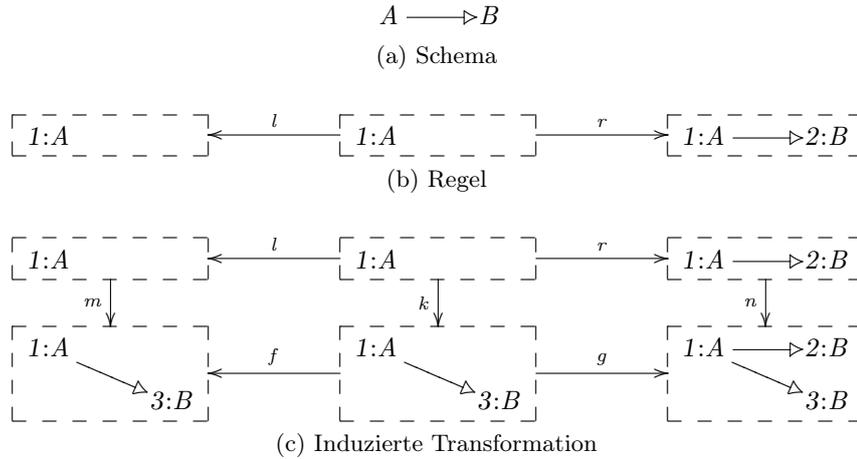


Abb. 10.5: Hinzufügen eines Oberobjekts

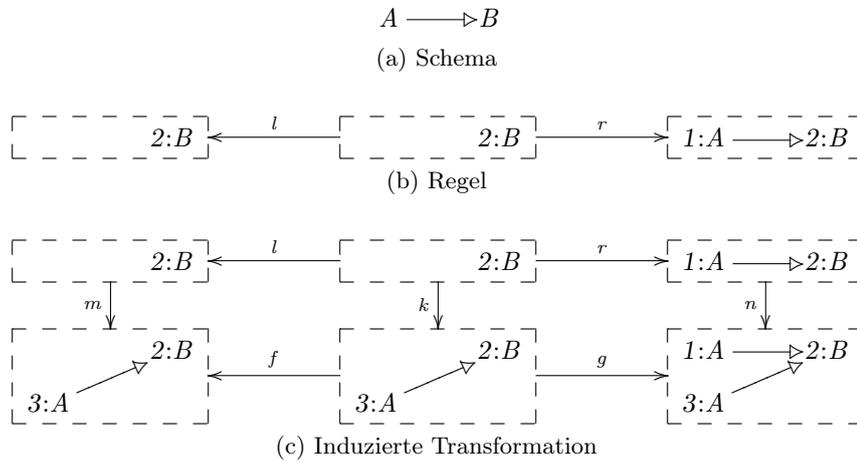


Abb. 10.6: Hinzufügen eines Unterobjekts

Beispiel 10.10 (Hinzufügen einer Verknüpfung). Sei das Schema wie in Abb. 10.7a gegeben. Die Regel in Abb. 10.7b fügt eine Verknüpfung hinzu. Angewandt auf einen Knoten, der bereits eine solche Verknüpfung besitzt, führt zu einem System, in dem das Axiom (M.29) nicht gilt (Abb. 10.7c). □

Das Gegenbeispiel für das Bilden des Pushout-Komplements auf der linken Seite der Regel nutzt die Abschwächung der Wahrheit von Prädikaten aus, um ein System zu konstruieren, das auf der Ausprägungs-Ebene die Axiome nicht erfüllt. Die Gegenbeispiele für die Pushout-Bildung auf der rechten Seite der Regel nutzen offenbar aus, dass durch die Pushout-Konstruktion in der resultierenden typisierten Graphstruktur $H \xrightarrow{\text{type}_H} S$ die Prämissen der Axiome (A.20), (M.28) und (M.29) wahr werden, ohne dass die Konklusion erfüllt ist. Beides kann durch eine stärkere Einschränkung von Aktionen verhindert werden. Dies wird im nächsten Abschnitt vorgestellt.

$$A \xrightarrow{x} B$$

(a) Schema

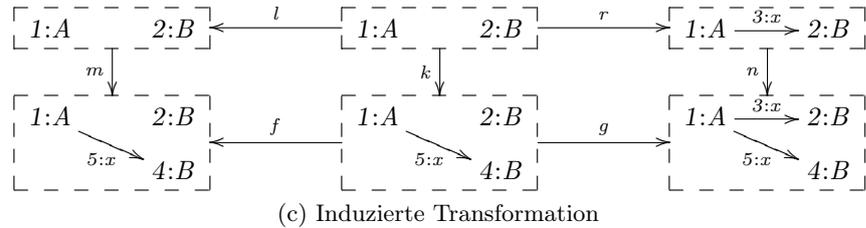
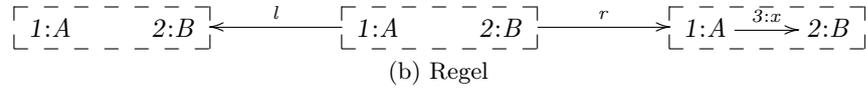


Abb. 10.7: Hinzufügen einer Verknüpfung

10.2 Konsistente Aktionen und Erhaltung der Axiome

In diesem Abschnitt werden im ersten Teil *vervollständigende Homomorphismen* (10.2.1) und *Kanten zurückziehende Spans* (10.2.2) eingeführt. Vervollständigende Homomorphismen besitzen die Fähigkeit, ein Prädikat erfüllende Elemente “zurückzuziehen”, auch wenn nur für einen Teil dieser Elemente Urbilder bekannt sind. Kanten zurückziehende Spans stellen sicher, dass Kanten, die auf der rechten Seite einer Aktion hinzugefügt werden, vorher auf der linken Seite gelöscht worden sind. Beides wird genutzt, um im zweiten Teil *konsistente Aktionen* zu formulieren (10.2.3). DPO-Graphstruktur-Transformationen entlang konsistenter Aktionen übertragen alle notwendigen Axiome (Theorem 10.33), was die Gegenbeispiele des letzten Abschnitts verbietet. Somit sind konsistente Aktionen zur Abbildung von Programm-Methoden bestens geeignet.

10.2.1 Vervollständigende Homomorphismen

Um die notwendigen Einschränkungen von Aktionen zu formulieren, werden zunächst spezielle Homomorphismen eingeführt, die so genannten *vervollständigenden Homomorphismen*. Diese Homomorphismen stellen eine stärkere Form von strikt vollen Homomorphismen dar, wie sie in Abschnitt A.3 dargestellt sind: Während strikt volle Homomorphismen Prädikate “zurückziehen”, wenn jeweils *alle* Elemente im Bild des Homomorphismus’ liegen, tun dies vervollständigende Homomorphismen auch, wenn nur *ein Teil* der Urbilder bekannt ist.

Ein Beispiel soll dies veranschaulichen. In Abb. 10.8a sind zwei Systeme und ein Homomorphismus f dazwischen abgebildet. Das linke System besteht aus einem Partikel x , das rechte aus zwei Partikeln x' und y' , wobei diese Partikel über das *under*-Prädikat miteinander in Beziehung stehen, was durch den Vererbungspfeil deutlich gemacht wird. Der Homomorphismus bildet das Partikel x des linken Systems auf das Partikel x' des rechten Systems ab.

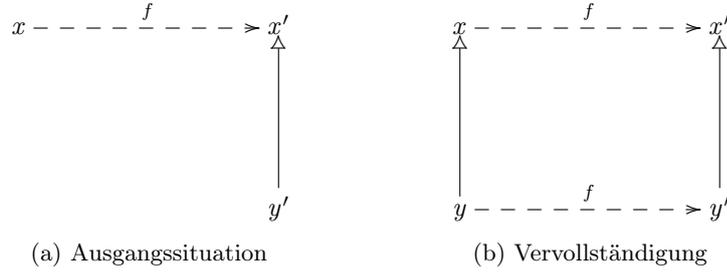


Abb. 10.8: Beispiel eines vervollständigenden Homomorphismus'

Es ist somit die Situation vorhanden, dass ein Element durch f auf ein anderes Element abgebildet wird, das – zusammen mit anderen Elementen – ein Prädikat erfüllt. Wenn f vervollständigend ist, dann folgt daraus die Existenz eines Urbilds y von y' , und zwar so, dass x und y ebenfalls zusammen das betrachtete Prädikat erfüllen. Es liegt somit die Situation wie in Abb. 10.8b vor. Vervollständigende Homomorphismen erlauben also in den betrachteten Kategorien, aus einer Vererbungs- oder Komponentenstruktur im Ziel auf eine gleiche Vererbungs- bzw. Komponentenstruktur in der Quelle zu schließen, sofern auch nur ein Element der betrachteten Struktur ein Urbild besitzt. Sie werden *vervollständigend* genannt, weil sie die bekannten Urbilder um die restliche “zurückgezogene” Struktur vervollständigen.

Es folgt die formale Definition vervollständigender Homomorphismen:

Definition 10.11 (Vervollständigender Homomorphismus). Sei $\Sigma = (S, OP, P)$ eine Signatur. Seien weiter $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B und $p \in P_w$ ein Prädikat zu einem Sortenwort $w \in S^*$. Dann ist h vervollständigend auf dem Prädikat p , wenn für jedes nichtleere Sortenwort w' , das durch Streichung beliebiger Sorten aus w entsteht, und je zwei Tupel $x \in B_w$ und $x' \in A_{w'}$ die Implikation

$$h_{w'}(x') = \langle x \rangle_{w'} \wedge x \in p^B \Rightarrow \exists y \in A_w : \langle y \rangle_{w'} = x' \wedge h_w(y) = x \wedge y \in p^A$$

gilt, wobei die Notation $\langle x \rangle_{w'}$ für die Projektion des Tupels x auf die Elemente zu den Sorten in dem Sortenwort w' steht. h ist vervollständigend, wenn h auf jedem Prädikat vervollständigend ist.

Anmerkung 10.12 (Vervollständigende Homomorphismen). Für null- und einstellige Prädikate gilt $w = w'$, weil keine Sorte gestrichen werden kann. Somit fallen auf null- und einstelligen Prädikaten vervollständigende und strikt volle Homomorphismen zusammen. \square

Vervollständigende Homomorphismen sind eng verwandt mit strikt vollen und mit injektiven Homomorphismen. Dies wird in den nächsten beiden Lemmata gezeigt.

Lemma 10.13 (Vervollständigende Homomorphismen (1)). *Seien $\Sigma = (S, OP, P)$ eine Signatur und $h: A \rightarrow B$ ein vervollständigender Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann ist h strikt voll.*

Beweis. Die Substitutionen $w' \mapsto w$, $x' \mapsto x$ und $x \mapsto h_w(x)$ führen zu der Implikation:

$$h_w(x) = h_w(x) \wedge h_w(x) \in p^B \Rightarrow \exists y \in A_w : y = x \wedge h_w(y) = h_w(x) \wedge y \in p^A$$

Dies ist offensichtlich äquivalent zu der Implikation (A.3). Somit ist h strikt voll. \square

Lemma 10.14 (Vervollständigende Homomorphismen (2)). *Seien $\Sigma = (S, OP, P)$ eine Signatur, $s \in S$ eine beliebige Sorte und $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann gilt: h ist vervollständigend auf $=_s$ genau dann, wenn h_s injektiv ist.*

Beweis. Offensichtlich gilt $w = s s$.

- \Rightarrow : Dies folgt aus Lemma 10.13 und Lemma A.24.
- \Leftarrow : Sei f_s injektiv, und sei ein Tupel $(x, x) \in B_w = B_s \times B_s$ gegeben. Es wird nach den möglichen Sortenwörtern w' unterschieden:

(1) $w' = w$. Sei ein Tupel $(y, y') \in A_w = A_s \times A_s$ gegeben mit:

$$(10.1) \quad f_s(y) = x$$

$$(10.2) \quad f_s(y') = x$$

Es ist zu zeigen, dass $y = y'$ gilt. Aus (10.1) und (10.2) folgt $f_s(y) = f_s(y')$. Aus der vorausgesetzten Injektivität von f_s folgt schließlich $y = y'$.

(2) $w' = s$.⁹⁰ Sei ein Element $y \in A_s$ gegeben mit:

$$f_s(y) = x$$

Es ist zu zeigen, dass ein Element $y' \in A_s$ existiert mit $f_s(y') = x$ und $y = y'$. Dies folgt jedoch unmittelbar für $y' ::= y$. \square

10.2.2 Kanten zurückziehende Spans

Ein Kanten zurückziehender Span stellt sicher, dass für jede Kante, die auf der rechten Seite einer Aktion hinzugefügt wird, auf der linken Seite eine Kante vom gleichen Typ und mit gleicher Quelle gelöscht wird. Anschaulich verhindern Kanten zurückziehende Spans somit, dass Kanten beliebig erzeugt werden. Diese Einschränkung ist wichtig für die Einhaltung von Axiom M.29.

⁹⁰ Es ist unerheblich, ob die erste oder die zweite Sorte gestrichen wird.

Definition 10.15 (Kanten zurückziehender Span). Sei S eine MP_* -Typ-Graphstruktur und $L \xleftarrow{l} K \xrightarrow{r} R$ ein in S typisierter MP_* -Span. Dann zieht $L \xleftarrow{l} K \xrightarrow{r} R$ Kanten zurück, wenn für jede Kante $e_R \in R_E$ ein Knoten $k \in K_N$ und eine Kante $e_L \in L_E$ existieren, so dass die Gleichungen

$$\begin{aligned} source^L(e_L) &= l_N(k) \\ source^R(e_R) &= r_N(k) \\ type_{L,E}(e_L) &= type_{R,E}(e_R) \end{aligned}$$

erfüllt sind.

10.2.3 Konsistente Aktionen

Mit vervollständigenden Homomorphismen und Kanten zurückziehenden Spans werden Aktionen jetzt so eingeschränkt, dass die in Abschnitt 10.1 vorgestellten Gegenbeispiele ausgeschlossen werden:

Definition 10.16 (Konsistente Aktion). Sei S eine MP_* -Typ-Graphstruktur. Dann ist jede Aktion $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, wenn

- (1) l und r vervollständigend sind, und
- (2) wenn $L \xleftarrow{l} K \xrightarrow{r} R$ Kanten zurückzieht.

Anmerkung 10.17 (Konsistente Aktion und strikt volle Homomorphismen). In einer konsistenten Aktion sind die Homomorphismen l und r strikt voll. Dies folgt aus der Eigenschaft der Vervollständigung nach Definition 10.16 und aus Lemma 10.13. \square

Anmerkung 10.18 (Konsistente Aktion). Punkt 1 verhindert auf der linken Seite einer Aktion, dass durch Abschwächung der Wahrheit von Prädikaten eine Situation wie in Beispiel 10.6 entsteht.⁹¹ Punkt 1 verhindert auf der rechten Seite einer Aktion zum einen mehrfach durch Prädikate ausgezeichnete Elemente wie in Beispiel 10.7. Zum anderen wird das Erweitern bestehender Vererbungs- und Komponentenstrukturen verboten, was zur Vermeidung von Problemen wie in den Beispielen 10.8 und 10.9 führt. Punkt 2 schließlich besagt anschaulich, dass eine Kante von einer Aktion nicht einfach hinzugefügt werden darf; vielmehr ist es notwendig, eine vorher bereits existierende Kante vom gleichen Typ und mit gleicher Quelle zu löschen. Dies verhindert die Anomalie in Beispiel 10.10. \square

⁹¹ Dafür würde es reichen, dass l strikt voll ist. Allerdings überträgt der Epirefektor nach Konstruktion A.92 keine strikt vollen Homomorphismen (vgl. Lemma 11.19), aber dafür vervollständigende Homomorphismen (vgl. Lemma 11.20). Für die Migration von Programmen ist es jedoch wichtig, dass der Epirefektor konsistente Aktionen überträgt (vgl. Abschnitt 11.3.1). Aus diesem Grund wird bereits jetzt ein vervollständigender Homomorphismus auf der linken Seite einer konsistenten Aktion gefordert.

Liegt eine konsistente Aktion vor, so kann nachgewiesen werden, dass die DPO-Transformation entlang einer solchen Aktion die M -Axiome des Ausgangssystems erhält. Dieses Theorem ist zentral für die Kodierung von Programmen durch konsistente Aktionen in der Kategorie $\mathbf{Alg}(MP_*) \downarrow S$ im nächsten Abschnitt und wird in mehreren Schritten bewiesen. Zuerst wird nachgewiesen, dass das Pushout-Komplement $D \xrightarrow{\text{type}_D} S$ die M -Axiome erfüllt:

Satz 10.19 (*M -Axiome gelten in $D \xrightarrow{\text{type}_D} S$*). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement. Dann gelten in $D \xrightarrow{\text{type}_D} S$ alle M -Axiome.

Beweis. Nach [16, Satz 10] ist das Pushout-Komplement D ein Untersystem⁹² von G in $\mathbf{Alg}(MP_*)$. Weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist, ist l vervollständigend und nach Lemma 10.13 strikt voll. Aus Lemma 9.21 folgt, dass f ebenfalls strikt voll ist. Nach Lemma A.24 ist D volles Untersystem von G . Weil nach Lemma A.32 der identische Homomorphismus auf dem Schema Isomorphismus und somit voll ist, folgt, dass $D \xrightarrow{\text{type}_D} S$ volles Untersystem von $G \xrightarrow{\text{type}_G} S$ ist. Nach Lemma A.75 ist jedes volle Untersystem eines *Spec*-Systems zu einer beliebigen Spezifikation *Spec* selbst ein *Spec*-System. Daraus folgt schließlich, dass $D \xrightarrow{\text{type}_D} S$ ein M -System ist. \square

Nun wird die Gültigkeit der einzelnen Axiome im Pushout-System $H \xrightarrow{\text{type}_H} S$ schrittweise nachgewiesen:

Lemma 10.20 (*MP-Axiom (MP.1) gilt in H*). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.1) in H gültig.

Beweis. Sei $x \in H_N$ gegeben. Nach Lemma B.26 existiert ein Urbild x' von x unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $x = g_N(x')$ mit $x' \in D_N$. Daraus folgt:

$$\begin{aligned} \text{source}^H(\text{self}^H(x)) &= \text{source}^H(\text{self}^H(g_N(x'))) && \text{(Definition von } x') \\ &= g_N(\text{source}^D(\text{self}^D(x'))) && (g \text{ ist Homomorphismus}) \\ &= g_N(x') && \text{(MP-Axiom (MP.1) gilt in } D) \\ &= x && \text{(Definition von } x') \end{aligned}$$

(2) Gelte $x = n_N(x')$ mit $x' \in D_N$. Dann folgt $\text{source}^H(\text{self}^H(x)) = x$ analog zum letzten Punkt, weil R ein *MP*-System ist. \square

⁹² Die Begriffe ‘‘Untersystem’’ und ‘‘Unteralgebra’’ sind wegen Satz A.96 austauschbar.

Lemma 10.21 (MP-Axiom (MP.2) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.2) in H gültig.

Beweis. Dies wird analog zum Beweis von Lemma 10.20 gezeigt. \square

Lemma 10.22 (MP-Axiom (MP.3) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.3) in H gültig.

Beweis. Dies wird analog zum Beweis von Lemma 10.20 gezeigt. \square

Lemma 10.23 (MP-Axiom (MP.4) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.4) in H gültig.

Beweis. Sei $z \in H_N$ gegeben. Nach Lemma B.26 existiert ein Urbild z' von z unter mindestens einem der beiden Homomorphismen g und n . Gelte $z = g_N(z')$. Weil D ein MP-System ist, gilt das Axiom (MP.4) in D . Daraus folgt $(z', z') \in \text{under}^D$. Da g ein MP-Homomorphismus ist und Homomorphismen die Wahrheit von Prädikaten übertragen, gilt $(g_N(z'), g_N(z')) \in \text{under}^H$ und somit $(z, z) \in \text{under}^H$.

Im Falle $z = n_N(z')$ verläuft der Beweis analog, da das Axiom (MP.4) in R ebenfalls gültig ist. \square

Lemma 10.24 (MP-Axiom (MP.5) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.5) in H gültig.

Beweis. Siehe Abschnitt C.2.1. \square

Lemma 10.25 (MP-Axiom (MP.6) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.6) in H gültig.

Beweis. Siehe Abschnitt C.2.2. \square

Lemma 10.26 (MP-Axiom (MP.7) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine Aktion, $G \xrightarrow{type_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.7) in H gültig.

Beweis. Siehe Abschnitt C.2.3. □

Lemma 10.27 (MP-Axiom (MP.8) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{type_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.8) in H gültig.

Beweis. Dies wird analog zu Lemma 10.25 nachgewiesen. □

Lemma 10.28 (MP-Axiom (MP.9) gilt in H). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine Aktion, $G \xrightarrow{type_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann ist das MP-Axiom (MP.9) in H gültig.

Beweis. Siehe Abschnitt C.2.4. □

Satz 10.29 (M^l -Axiome gelten in $H \xrightarrow{type_H} S$). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{type_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann gelten in $H \xrightarrow{type_H} S$ die M^l -Axiome (M.1) bis (M.27).

Beweis. Die M^l -Axiome können in drei Gruppen aufgeteilt werden:

- (1) Die Typisierungs-Axiome (M.1), (M.2), (M.3), (M.4), (M.5), (M.6), (M.7), (M.8) und (M.9) sind dann gültig, wenn H ein MP -System ist. Denn in diesem Fall ist $type_H$ MP_* -Homomorphismus zwischen zwei MP -Systemen; da $\mathbf{Alg}(MP)$ im Wesentlichen volle Unterkategorie von $\mathbf{Alg}(MP_*)$ ist⁹³, ist $type_H$ folglich MP -Homomorphismus und erfüllt die Typisierungs-Axiome auf Grund der Homomorphie-Eigenschaft.
- (2) Die M^l -Axiome (M.10), (M.11), (M.12), (M.16), (M.17), (M.18), (M.22), (M.23) und (M.26) entsprechen den MP -Axiomen (MP.1), (MP.2), (MP.3), (MP.4), (MP.5), (MP.6), (MP.7), (MP.8) und (MP.9) für das Schema S . Aus der Definition einer Aktion folgt, dass S ein MP -System ist und somit alle diese Axiome in S gültig sind. Daraus folgt die Gültigkeit der korrespondierenden M^l -Axiome in $H \xrightarrow{type_H} S$.

⁹³ Auf Grund der zwischengeschalteten Äquivalenz ist es keine gewöhnliche Unterkategorie.

- (3) Die M^1 -Axiome (M.13), (M.14), (M.15), (M.19), (M.20), (M.21), (M.24), (M.25) und (M.27) entsprechen den MP -Axiomen (MP.1), (MP.2), (MP.3), (MP.4), (MP.5), (MP.6), (MP.7), (MP.8) und (MP.9) für die Ausprägung H . Die Gültigkeit dieser Axiome folgt aus den Lemmata 10.20 bis 10.28. \square

Satz 10.30 (Prädikat-Axiome gelten in $H \xrightarrow{\text{type}_H} S$). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann gilt in $H \xrightarrow{\text{type}_H} S$ das Axiom (A.20) für jedes Prädikat $p \in P_w$ mit $w \in S^*$.

Beweis. Siehe Abschnitt C.2.5. \square

Satz 10.31 (M-Axiom (M.28) gilt in $H \xrightarrow{\text{type}_H} S$). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann gilt in $H \xrightarrow{\text{type}_H} S$ das M-Axiom (M.28).

Beweis. Siehe Abschnitt C.2.6. \square

Satz 10.32 (M-Axiom (M.29) gilt in $H \xrightarrow{\text{type}_H} S$). Sei $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ das bis auf Isomorphie eindeutige Pushout-Komplement und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$. Dann gilt in $H \xrightarrow{\text{type}_H} S$ das M-Axiom (M.29).

Beweis. Siehe Abschnitt C.2.7. \square

Schließlich kann auf Grundlage dieser Lemmata das Theorem formuliert und nachgewiesen werden:

Theorem 10.33 (DPO-Transformation erhält M-Axiome). Sei $p = L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion, $G \xrightarrow{\text{type}_G} S$ ein M -System und $m: L \rightarrow G$ ein S -typisierter Data-attributierter DPO-Ansatz, so dass (l, m) sowohl die Kontakt- als auch die Identifikationsbedingung erfüllen. Sei nun (p', k, n) die nach Satz B.59 eindeutig existierende (p, m) -induzierte S -typisierte Data-attributierte Graphstruktur-Transformation mit dem S -typisierten MP_* -Span $p' = G \xleftarrow{f} D \xrightarrow{g} H$ und den S -typisierten MP_* -Graphstruktur-Morphismen $k: K \rightarrow D$ und $n: R \rightarrow H$. Dann sind die M -Axiome in $D \xrightarrow{\text{type}_D} S$ und $H \xrightarrow{\text{type}_H} S$ gültig.

Beweis. $D \xrightarrow{\text{type}_D} S$ ist ein M -System nach Satz 10.19. $H \xrightarrow{\text{type}_H} S$ ist ein M -System nach Satz 10.29, Satz 10.30, Satz 10.31 und Satz 10.32. \square

Theorem 10.33 rechtfertigt die Kodierung von Programmen als DPO-Graphstruktur-Transformationen in der Kategorie $\mathbf{Alg}(MP_*) \downarrow S$. In dem folgenden Abschnitt wird dargestellt, wie sich Methoden als Aktionen auffassen lassen.

10.3 Finden eines Ansatzes

Dieser Abschnitt behandelt die Fragestellung, wann eine Methode während eines Programmlaufs abgearbeitet wird. Es wird untersucht, wann es zu Mehrdeutigkeiten bei der Auswahl der passenden Methode kommt und wie in solchen Fällen die korrekte Abarbeitung des Programms gewährleistet werden kann. Dabei werden zwei sich ergänzende Vorgehensweisen verfolgt: Partielles Ordnen von Methoden (10.3.1) und partiell injektive Regelansätze (10.3.2).

10.3.1 Partielle Ordnung von Methoden

Ist eine Operation gegeben, dann kann es passieren, dass in einem laufenden Prozess zu einem Zeitpunkt mehr als eine Methode zur Abarbeitung einer aktiven Nachricht in Frage kommt, nämlich wenn einer Operation innerhalb einer Klassenhierarchie mehr als eine Methode zugeordnet ist. Das objektorientierte Paradigma fordert in diesem Fall, dass die *speziellste* Methode, bezogen auf die Spezialisierungsordnung der Parameter-Typen der Methode, zur Ausführung kommt.

Im mathematischen Modell kann dies auf einfache Weise formuliert werden. Zuerst müssen mehrdeutige Methoden verboten werden. Eine Mehrdeutigkeit liegt vor, wenn zwei Methoden (bis auf Isomorphie) dieselbe linke, aber unterschiedliche rechte Seiten besitzen:

Annahme 10.34 (Verbot von mehrdeutigen Methoden). Seien $m = L \xleftarrow{l} K \xrightarrow{r} R$ und $m' = L' \xleftarrow{l'} K' \xrightarrow{r'} R'$ zwei Methoden, und gelte $L \cong L'$. Dann gilt $m \cong m'$, d. h. es gilt $K \cong K'$ und $R \cong R'$, und alle Isomorphismen sind mit den Morphismen der beiden Spans verträglich, so dass das entstehende Diagramm kommutiert. \square

Nun können die vorhandenen Methoden durch den Vergleich ihrer linken Seiten partiell geordnet werden:

Definition 10.35 (Partielle Ordnung von Methoden). Seien $m = L \xleftarrow{l} K \xrightarrow{r} R$ und $m' = L' \xleftarrow{l'} K' \xrightarrow{r'} R'$ zwei Methoden. Dann gilt $m \leq m'$, wenn eine Einbettung $i: L' \rightarrow L$ existiert.

Durch die Kodierung der Methoden wird sichergestellt, dass eine Methode mit spezielleren Parameter-Typen kleiner als eine Methode mit allgemeineren Parameter-Typen ist. Denn die Methode mit spezielleren Parameter-Typen besitzt mehr Kontext durch die Angabe weiterer speziellerer Partikel in den Parametern. Eine Einbettung der Methode mit allgemeineren Parameter-Typen ist somit immer möglich; umgekehrt können die zusätzlichen Partikel jedoch nicht injektiv abgebildet werden. Hier wird auch verständlich, warum in der Definition die geforderte Einbettung *kontravariant* zu der aufgestellten Ordnungsrelation ist: Dadurch wird sichergestellt, dass eine Methode mit spezielleren Parameter-Typen auch *kleiner* als eine Methode mit allgemeineren

Parameter-Typen ist, was der üblichen Typ-Ordnung in Spezialisierungshierarchien entspricht, obwohl sie von ihrer Graphstruktur her *größer* als die Methode mit allgemeineren Parameter-Typen ist.

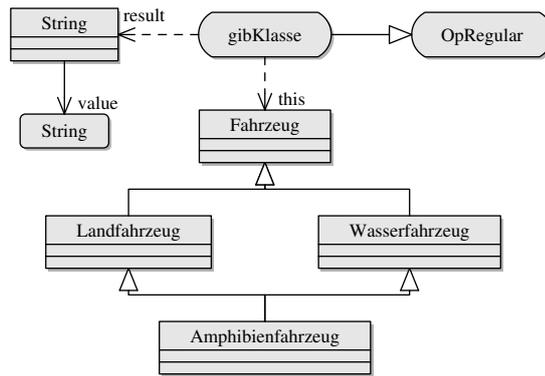
Sei nun $M = \{p_1, p_2, \dots, p_n\}$ mit $n > 1$ die Menge derjenigen Methoden, die zur Abarbeitung einer Nachricht als Kandidaten in Frage kommen. Seien m_1 bis m_n die zugehörigen Ansätze. Dann werden gemäß der oben definierten partiellen Ordnung die minimalen Methoden in M bestimmt. Existiert nur eine minimale Methode p_i mit $1 \leq i \leq n$, so ist die Wahl der Methode eindeutig, und p_i wird ausgeführt, indem sie die Daten und den Prozess unter dem Ansatz m_i gemäß ihrer Definition transformiert. Existieren mehrere minimale Methoden, liegt eine nicht aufzulösende Mehrdeutigkeit vor, und die Abarbeitung des Ablauffadens bricht ab. Eine solche Situation kann etwa bei Mehrfachvererbung auftauchen, wenn zwei Geschwister-Methoden nicht in der speziellsten Klasse redefiniert werden (Abb. 10.9; aus Gründen der Übersichtlichkeit werden die *OpBase*- und *OpRegular*-Partikel sowie die Spezialisierungen jeweils in einem einzigen Knoten dargestellt): Falls eine Nachricht zu der Operation *gibKlasse* an ein Objekt der Klasse *Amphibienfahrzeug* geschickt wird, ist unklar, welche Methode ausgeführt werden soll, weil beide Methoden gemäß der partiellen Ordnung minimal sind.

10.3.2 Partiiell injektive Ansätze

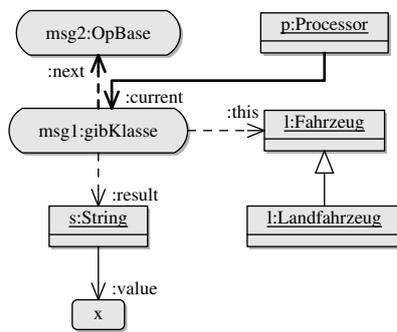
Für gewöhnlich ist die Ersetzungsvorschrift, welche die Semantik einer Methode beschreibt, so formuliert, dass unterschiedliche Elemente der Methode durchaus identischen Elementen im eigentlichen Prozess zugeordnet werden können. Ein Beispiel hierzu ist in Abb. 5.21b dargestellt: Werden zwei gleiche Zahlen addiert, dann werden die unterschiedlichen Terme x und y bei der Ausführung der Methode derselben Zahl zugeordnet. Somit kann sowohl die Addition gleicher als auch die Addition unterschiedlicher Zahlen mit derselben Methode abgebildet werden.

Es gibt jedoch Situationen, in denen dieses Zusammenlegen unterschiedlicher Methoden-Elemente vermieden werden muss. Eine solche Situation ist die Abbildung des Operators `==`, der zwei Objekte zu einem beliebigen Typ auf Gleichheit testet. In Abb. 10.10 ist der Gleichheits-Operator auf Objekten zu einem beliebigen Typ *Type* dargestellt. Zu dem Schema in Abb. 10.10a werden zwei Methoden formuliert, einmal die Methode *true* bei gleichen Objekten (Abb. 10.10b und 10.10c) und einmal die Methode *false* bei unterschiedlichen Objekten (Abb. 10.10d und 10.10e). Das Problem dabei ist jedoch, dass die Methode *false* auch zum Einsatz kommen kann, wenn ein Objekt mit sich selbst verglichen wird: In diesem Fall werden beide *Type*-Partikel in der Methode durch den Ansatz auf das eine *Type*-Partikel des betrachteten Objekts abgebildet. Das ist offensichtlich unerwünscht, führt dies doch zur Laufzeit zu einer Mehrdeutigkeit zwischen den beiden Methoden und somit zu einem Programm-Abbruch (vgl. 5.2.1.7).⁹⁴

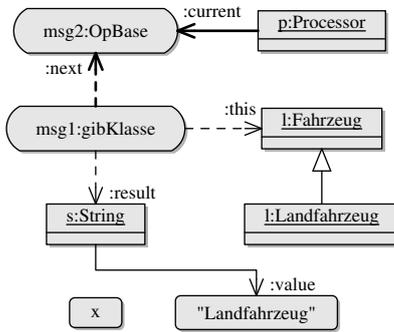
⁹⁴ Vgl. hierzu auch [18, Kapitel 5].



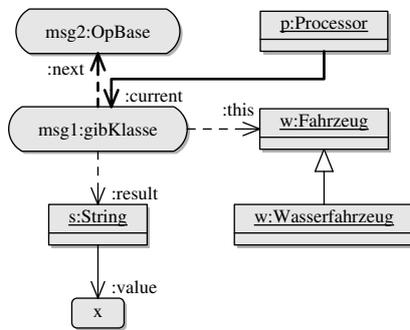
(a) Operation



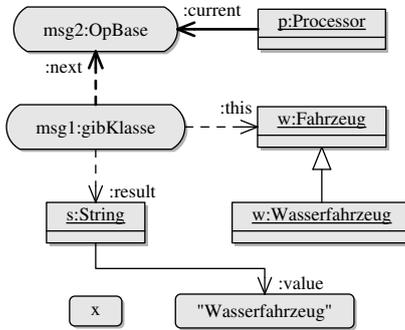
(b) Landfahrzeug-Methode: vorher



(c) Landfahrzeug-Methode: nachher

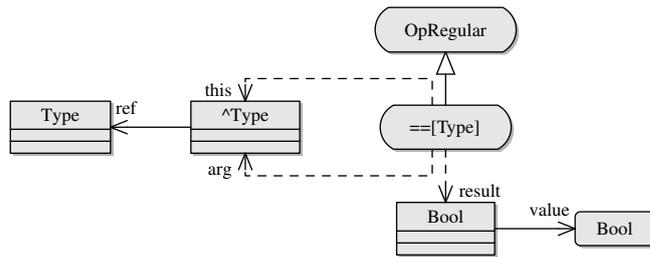


(d) Wasserfahrzeug-Methode: vorher

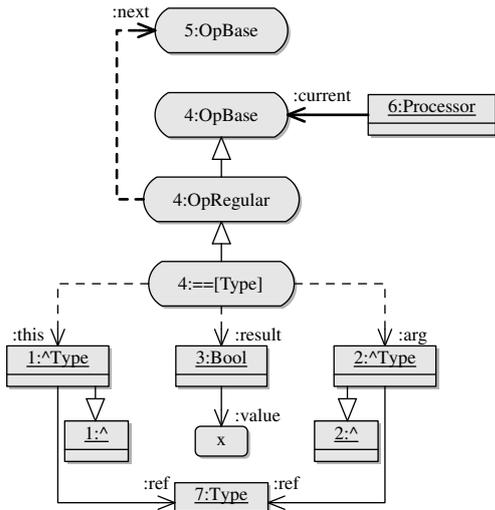


(e) Wasserfahrzeug-Methode: nachher

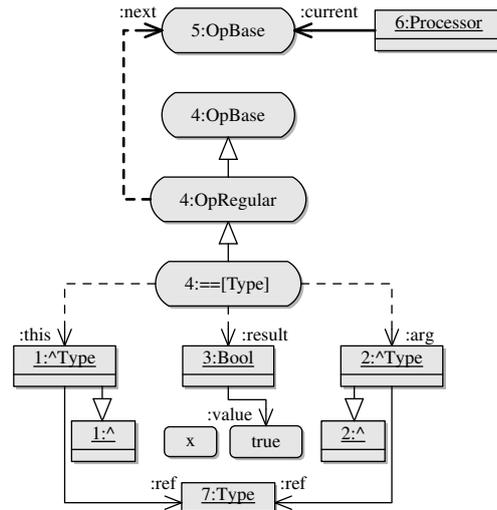
Abb. 10.9: Mehrdeutige Methoden



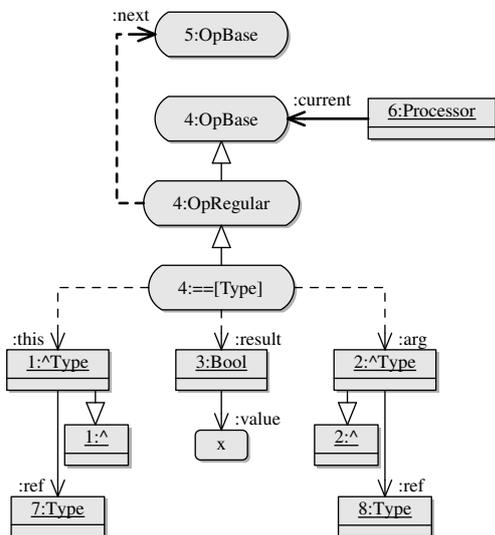
(a) Operation



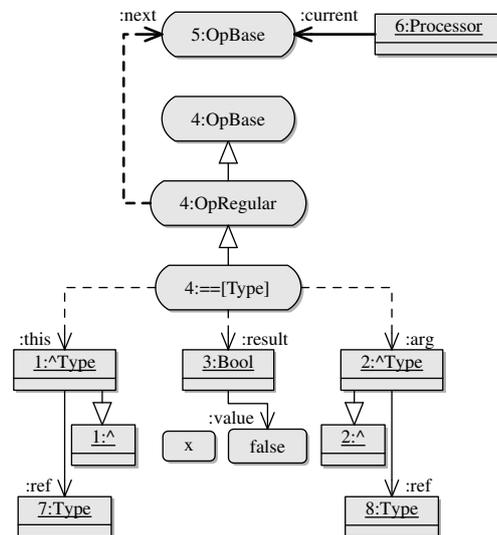
(b) Methode *true*: vorher



(c) Methode *true*: nachher



(d) Methode *false*: vorher



(e) Methode *false*: nachher

Abb. 10.10: Gleichheit auf *Type*-Objekten: Operation und Methoden

Somit ist es sinnvoll, in diesem und ähnlichen Fällen⁹⁵ zu verhindern, dass unterschiedliche Methoden-Elemente bei der Ausführung der Methode identischen Daten- und Prozess-Elementen zugeordnet werden. Dazu werden alle Methoden in zwei Mengen NIM ⁹⁶ und IM ⁹⁷ aufgeteilt. Methoden in der Menge NIM unterliegen keinen Einschränkungen bezüglich der Abbildung in die Daten- und Prozess-Welt. Methoden in der Menge IM hingegen dürfen nur so abgebildet werden, dass Partikel nicht zusammengelegt werden dürfen; dazu gehören insbesondere die diskutierten Gleichheits- und Ungleichheits-Operatoren auf Objekten.⁹⁸ Nun wird das Problem dahingehend gelöst, dass jedem Ansatz einer Regel, die einer Methode aus der Menge IM entspricht, das Zusammenlegen von Partikeln verboten wird. Der Ansatz ist in diesen Fällen somit partiell injektiv. Bei Methoden aus der Menge NIM besteht diese Beschränkung nicht, und jeder Ansatz ist erlaubt.⁹⁹

Definition 10.36 (Partiell injektiver Ansatz). *Sei $m: L \rightarrow G$ ein S -typisierter Data-attribuerter DPO-Ansatz. Dann ist m partiell injektiv, wenn m keine Partikel identifiziert, wenn also für alle $x, y \in L_N$ gilt:*

$$(10.3) \quad m_N(x) = m_N(y) \wedge x \notin \text{message}^L \wedge y \notin \text{message}^L \Rightarrow x = y$$

⁹⁵ Eine ähnliche Situation liegt beim Operator $!=$ vor, der auf Ungleichheit zweier Objekte prüft, wobei diese Prüfung auch indirekt über die Operatoren $==$ und $!$ (dem Verneinungsoperator auf Wahrheitswerten) kodiert werden kann.

⁹⁶ für “non-injective match” (dt. “nicht injektiver Ansatz”)

⁹⁷ für “injective match” (dt. “injektiver Ansatz”)

⁹⁸ Gleichheits- und Ungleichheits-Operatoren auf Werten können durch passende Operationen in der Daten-Algebra definiert werden, die “das Richtige” tun. Somit ist für diese Operatoren keine Spezialbehandlung notwendig.

⁹⁹ Ein anderer möglicher Ansatz ist, generell mit injektiven Ansätzen zu arbeiten und über Quotienten von Regeln mögliche Identifikationen von Elementen auf der linken Seite zu spezifizieren; vgl. hierzu auch [31, 32].

Transformationen

In diesem Kapitel werden *Transformationen* auf Schema- und Ausprägungs-Ebene formalisiert. Abschnitt 11.1 führt Transformationen und Refactorings im mathematischen Modell als spezielle *Spans* ein. Weiter wird untersucht, wie eine Schema-Transformation durch die Kombination einer kofreien mit einer freien Konstruktion eindeutig in eine Ausprägungs-Transformation überführt werden kann. Dabei behandelt Abschnitt 11.2 die Migration von Daten und Prozessen und Abschnitt 11.3 die Migration von Programmen. In Abschnitt 11.4 wird zum einen gezeigt, wie Transformationen komponiert werden können, um aus mehreren einzelnen Transformationen eine einzige Transformation zu erstellen. Zum anderen wird untersucht, welche formalen Voraussetzungen gegeben sein müssen, damit die Anwendung der einzelnen Transformationen und die Anwendung der komponierten Transformation zu identischen Resultaten führt.

11.1 Grundlagen

Allgemeine Schema-Transformationen sollen ein beliebiges Schema in ein beliebiges anderes Schema überführen und dabei spezifizieren, wie die Klassen und Assoziationen im ursprünglichen Schema mit den Klassen und Assoziationen im neuen Schema zusammenhängen. Seien zwei Schemata S und S' aus $\mathbf{Alg}(MP)$ gegeben. Dann ist der erste offensichtliche Ansatz ein Homomorphismus $r: S \rightarrow S'$. Dieser Ansatz erlaubt immerhin schon die beiden folgenden Aktionen:

- *Extension*: Ist r nicht surjektiv, wird das ursprüngliche Schema S in ein größeres Schema S' hinein abgebildet. Dieser Typ von Transformation erlaubt also das Erweitern des ursprünglichen Schemas S um Klassen und Assoziationen.
- *Faltung*: Ist r nicht injektiv, werden Teile des ursprünglichen Schemas S zusammengelegt. Dieser Typ von Transformation erlaubt also das Verringern des Abstraktionsgrades auf Schema-Ebene, indem im ursprünglichen Schema S unterschiedliche Elemente (Klassen oder Assoziationen) im neuen Schema S' nicht mehr unterschieden werden.

Natürlich lassen sich beide Aktionen in einem Homomorphismus kombinieren, wenn r weder injektiv noch surjektiv ist. Der letzte Fall — r ist injektiv und surjektiv — modelliert die identische Transformation.¹⁰⁰

Wird die Richtung der Homomorphismen umgedreht und ein Homomorphismus $l: S' \rightarrow S$ betrachtet, ergibt dies die folgenden komplementären Aktionen:

- *Reduktion*: Ist l nicht surjektiv, wird das neue Schema S' in ein größeres Schema S abgebildet. Somit werden Teile des ursprünglichen Schemas entfernt. Dieser Typ von Transformation erlaubt also das Löschen von Klassen und Assoziationen im ursprünglichen Schema S .
- *Entfaltung*: Ist l nicht injektiv, werden Teile des ursprünglichen Schemas S im neuen Schema S' "auseinandergezogen". Dieser Typ von Transformation erlaubt also das Erhöhen des Abstraktionsgrades auf Schema-Ebene, indem im ursprünglichen Schema S ein Element (Klasse oder Assoziation) im neuen Schema S' mehreren Klassen oder Assoziationen entspricht.

Es scheint sinnvoll zu sein, dass eine Schema-Transformation alle vier Aktionen unterstützt. Idealerweise sollten alle vier Aktionen in einer einzigen Schema-Transformation vereinigt werden können. Analog zum DPO-Ansatz bei algebraischen Graphstruktur-Transformationen (B.2.1) kann dieses Ziel erreicht werden, indem nicht ein einzelner Homomorphismus, sondern ein *Span* von Homomorphismen verwendet wird [51, 52] (Abb. 11.1).

$$S \xleftarrow{l} S^\# \xrightarrow{r} S'$$

Abb. 11.1: Ein Span als Modell einer Schema-Transformation in $\mathbf{Alg}(MP)$

Ein solcher Span kann als *Relation* zwischen S und S' aufgefasst werden. Ein Element aus S kann mit beliebig vielen Elementen in S' in Beziehung gesetzt werden; ebenso kann ein Element aus S' mit beliebig vielen Elementen in S verknüpft sein. Elemente in S , die mit keinem Element in S' in Relation stehen, werden gelöscht. Dual dazu werden Elemente in S' , die mit keinem Element in S verbunden sind, neu erzeugt. Alle in Beziehung stehenden Elemente sind im Schnittstellen-Schema $S^\#$ enthalten und werden gemäß l und r entweder reduziert oder vervielfältigt. Dies entspricht genau der Interpretation von DPO-Regeln.

Diese Erkenntnisse können problemlos auf die Ausprägungs-Ebene übertragen werden. Zusammengefasst ergibt sich:

Definition 11.1 (Transformation in $\mathbf{Alg}(MP)$). Eine Transformation $t: S \xrightarrow{S^\#} S'$ in $\mathbf{Alg}(MP)$ ist ein Span $S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$.

¹⁰⁰ Das ist nicht ganz korrekt, denn das gilt nur für *volle* injektive und surjektive Homomorphismen. Nicht volle Homomorphismen können Prädikate im Ziel-Schema wahr werden lassen, so dass kein Homomorphismus zurück mehr möglich ist. Für die Betrachtung der *strukturellen* Eigenschaften der Aktionen ist die Vereinfachung aber völlig ausreichend.

Refactorings sind nach Def. 4.10 spezielle Transformationen, die das Verhalten der Software und den Informationsgehalt der Datenbank erhalten. Von den vier oben beschriebenen Aktionen Extension, Faltung, Reduktion und Entfaltung ist sicherlich die Reduktion problematisch, denn sie kann zu Löschungen auf der Ausprägungsebene und somit zum Verlust von Daten führen. Dies motiviert die folgende Definition eines Refactorings im mathematischen Modell:¹⁰¹

Definition 11.2 (Refactoring in $\mathbf{Alg}(MP)$). Eine Transformation $t: S \xrightarrow{S^\#} S'$ ist genau dann ein Refactoring, wenn l^t surjektiv ist.

Anmerkung 11.3 (Nicht surjektive Homomorphismen auf der rechten Seite eines Refactorings). Die Definition eines Refactorings ist nicht symmetrisch, denn sie erlaubt das Verwenden nicht surjektiver Homomorphismen auf der rechten Seite einer Transformation. Der offensichtliche Zweck dieser Asymmetrie ist der Wunsch, Schema-Erweiterungen zu erlauben, die nach Abschnitt 4.5.3 als Refactorings begriffen werden können. Die tieferliegende Ursache ist eine grundlegende Asymmetrie zwischen Erzeugen und Löschen: Das Erzeugen neuer Elemente im Schema zieht nicht unbedingt das Erzeugen neuer Elemente in der Ausprägung nach sich, während das Löschen vorhandener Schema-Elemente immer auch das Löschen aller darin typisierten Elemente in der Ausprägung zur Folge hat. □

Jetzt bleibt nur noch die Frage, wie Schema-Transformationen auf bestehende Systeme angewandt und Daten, Prozesse und Programme zum alten Schema in das neue Schema migriert werden können. Dies wird in den nächsten zwei Abschnitten untersucht.

11.2 Migration von Daten und Prozessen

Daten und damit arbeitende Prozesse befinden sich in einem gemeinsamen System. Somit können sie zusammenfassend betrachtet werden. Seien ein M -System $I \xrightarrow{type_I} S$ und eine Schema-Transformation $t: S \xrightarrow{S^\#} S'$ in der Kategorie $\mathbf{Alg}(MP)$ gegeben (Abb. 11.2).

$$\begin{array}{c}
 S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S' \\
 \uparrow type_I \\
 I
 \end{array}$$

Abb. 11.2: System und Schema-Transformation in $\mathbf{Alg}(MP)$

Dann wird das M -System $I \xrightarrow{type_I} S$ folgendermaßen zum neuen Schema S' migriert:

¹⁰¹ In Abschnitt 14.2.6 wird ein alternativer Ansatz vorgestellt, in dem Refactorings als Schema-Transformationen definiert werden, welche die so genannte Informationskapazität nicht verringern, was über (einfache) Surjektivität auf der linken Seite der Schema-Transformation hinausgeht.

- (1) Auf der linken Seite wird der Pullback $I \xleftarrow{l^t} I\# \xrightarrow{\text{type}_{I\#}} S\#$ von $I \xrightarrow{\text{type}_I} S \xleftarrow{l^t} S\#$ konstruiert, um ein System $I\# \xrightarrow{\text{type}_{I\#}} S\#$ zu erhalten.
- (2) Auf der rechten Seite wird das System $I\# \xrightarrow{r^t \circ \text{type}_{I\#}} S'$ nach den M -Axiomen durchfaktoriert, was das System $I' \xrightarrow{\text{type}_{I'}} S'$ ergibt.

Der gesamte Vorgang ist in Abb. 11.3 dargestellt.

$$\begin{array}{ccccc}
 S & \xleftarrow{l^t} & S\# & \xrightarrow{r^t} & S' \\
 \uparrow \text{type}_{I\#} & & \uparrow \text{type}_{I\#} & \nearrow r^t \circ \text{type}_{I\#} & \uparrow \text{type}_{I'} \\
 \text{P.B.} & & & & \\
 I & \xleftarrow{l^t} & I\# & \xrightarrow{id_{I\#}} & I\# \xrightarrow{[\cong]} I' \\
 & & & & \uparrow \text{type}_{I'}
 \end{array}$$

Abb. 11.3: System-Migration in $\mathbf{Alg}(MP)$

Die beschriebene Konstruktion geht von zwei Annahmen aus:

- (1) Das System $I\# \xrightarrow{\text{type}_{I\#}} S\#$ ist ein M -System, das also insbesondere die Axiome (M.28) und (M.29) erfüllt. Dies wird in Lemma 11.9 nachgewiesen.
- (2) Die Anpassung des Systems $I\# \xrightarrow{r^t \circ \text{type}_{I\#}} S'$ an die M -Axiome lässt das Schema unverändert. Dies gilt nach Lemma 9.13.

Bevor nachgewiesen wird, dass die Pullback-Konstruktion M -Systeme in M -Systeme überträgt, ist es hilfreich, die Konstruktion in einem anderen Licht zu betrachten. Faktisch wird hier ein System zu einem Schema S "entlang" l^t auf ein System zu einem Schema $S\#$ abgebildet. Dies lässt sich als eine Objekt-Zuordnung $\mathcal{P}_{\text{Ob}}^{l^t}: \text{Ob}^{\mathbf{Alg}(MP)\downarrow S} \rightarrow \text{Ob}^{\mathbf{Alg}(MP)\downarrow S\#}$ interpretieren, wobei die Pullback-Konstruktion sich jeweils auf ein Objekt festlegt.¹⁰² Es wird nun gezeigt, dass diese Objekt-Zuordnung auf Morphismen fortgesetzt werden kann und dass sich daraus ein Funktor $\mathcal{P}^{l^t}: \mathbf{Alg}(MP)\downarrow S \rightarrow \mathbf{Alg}(MP)\downarrow S\#$ ergibt:

Konstruktion 11.4 (Pullback-Funktor). Sei ein $\mathbf{Alg}(MP)$ -Morphismus $f: S^B \rightarrow S^A$ gegeben. Dann sei eine Objekt-Zuordnung

$$\mathcal{P}_{\text{Ob}}^f: \text{Ob}^{\mathbf{Alg}(MP)\downarrow S^A} \rightarrow \text{Ob}^{\mathbf{Alg}(MP)\downarrow S^B}$$

und eine Morphismen-Zuordnung

$$\mathcal{P}_{\text{Mor}}^f: \text{Mor}^{\mathbf{Alg}(MP)\downarrow S^A} \rightarrow \text{Mor}^{\mathbf{Alg}(MP)\downarrow S^B}$$

folgendermaßen definiert:

¹⁰² Pullbacks sind nur "bis auf Isomorphie" eindeutig bestimmt. Weil eine Abbildung jedoch rechtseindeutig sein muss, ist es notwendig, sich bei jeder Pullback-Konstruktion für ein bestimmtes Pullback-Objekt aus der Klasse aller isomorphen Pullback-Objekte zu entscheiden.

- Sei ein Objekt $A \xrightarrow{\text{type}_A} S^A \in \text{Ob}^{\mathbf{Alg}(MP)\downarrow S^A}$ gegeben, und sei $A \xleftarrow{f^*} B \xrightarrow{\text{type}_B} S^B$ Pullback von $A \xrightarrow{\text{type}_A} S^A \xleftarrow{f} S^B$ in $\mathbf{Alg}(MP)$. Dann sei definiert:

$$\mathcal{P}_{\text{Ob}}^f(A \xrightarrow{\text{type}_A} S^A) ::= B \xrightarrow{\text{type}_B} S^B$$

- Sei ein Morphismus $m: (A^1 \xrightarrow{\text{type}_{A^1}} S^A) \rightarrow (A^2 \xrightarrow{\text{type}_{A^2}} S^A) \in \text{Mor}^{\mathbf{Alg}(MP)\downarrow S^A}$ gegeben, und seien $A^1 \xleftarrow{f_1^*} B^1 \xrightarrow{\text{type}_{B^1}} S^B$ und $A^2 \xleftarrow{f_2^*} B^2 \xrightarrow{\text{type}_{B^2}} S^B$ Pullbacks von $A^1 \xrightarrow{\text{type}_{A^1}} S^A \xleftarrow{f} S^B$ bzw. $A^2 \xrightarrow{\text{type}_{A^2}} S^A \xleftarrow{f} S^B$ in $\mathbf{Alg}(MP)$. Dann sei definiert:

$$\mathcal{P}_{\text{Mor}}^f(m) ::= m^*$$

wobei m^* den eindeutigen Mittler-Morphismus in das Pullback-Objekt $A^2 \xleftarrow{f_2^*} B^2 \xrightarrow{\text{type}_{B^2}} S^B$ darstellt (Abb. 11.4). \square

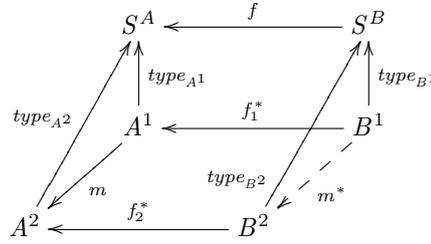


Abb. 11.4: Definition von $\mathcal{P}_{\text{Mor}}^f$

Lemma 11.5 (Pullback-Funktor). $\mathcal{P}^f = (\mathcal{P}_{\text{Ob}}^f, \mathcal{P}_{\text{Mor}}^f)$ aus Konstruktion 11.4 ist ein Funktor $\mathcal{P}^f: \mathbf{Alg}(MP)\downarrow S^A \rightarrow \mathbf{Alg}(MP)\downarrow S^B$, genannt Pullback-Funktor entlang f .

Beweis. \mathcal{P}^f ist verträglich bezüglich Komposition. Nach Definition gilt $\mathcal{P}^f(n \circ m) ::= (n \circ m)^*$ mit $\text{type}_{B^3} \circ (n \circ m)^* = \text{type}_{B^1}$ und $f_3^* \circ (n \circ m)^* = n \circ m \circ f_1^*$ (Abb. 11.5). Weiter gilt zum einen:

$$\begin{aligned} \text{type}_{B^3} \circ n^* \circ m^* &= \text{type}_{B^2} \circ m^* && (n^* \text{ ist Mittler-Morphismus}) \\ &= \text{type}_{B^1} && (m^* \text{ ist Mittler-Morphismus}) \end{aligned}$$

Zum anderen gilt:

$$\begin{aligned} f_3^* \circ n^* \circ m^* &= n \circ f_2^* \circ m^* && (n^* \text{ ist Mittler-Morphismus}) \\ &= n \circ m \circ f_1^* && (m^* \text{ ist Mittler-Morphismus}) \end{aligned}$$

Aus der Eindeutigkeit von $(n \circ m)^*$ folgt $(n \circ m)^* = n^* \circ m^*$ und somit $\mathcal{P}^f(n \circ m) = \mathcal{P}^f(n) \circ \mathcal{P}^f(m)$. \mathcal{P}^f überträgt Identitäten. Denn die Pullback-Konstruktion wählt auf Grund der geforderten Rechts-eindeutigkeit für jede Ausgangssituation dasselbe Pullback-Objekt. Auf Grund der Verträglichkeit

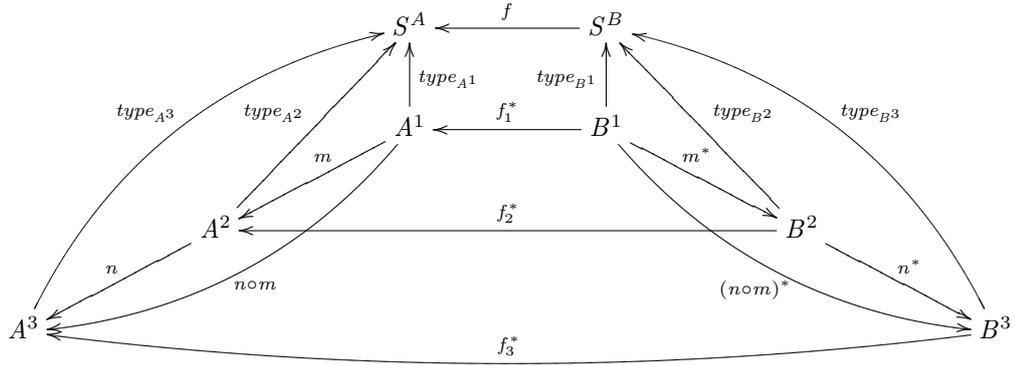


Abb. 11.5: \mathcal{P}^f ist verträglich bezüglich Komposition

mit den Vergleichsmorphismen gilt sowohl $f_1^* \circ id_{A^1}^* = id_{A^1} \circ f_1^*$ als auch $type_{A^2} \circ id_{A^1}^* = type_{A^2}$ (Abb. 11.6). Offensichtlich gilt auch $f_1^* \circ id_{A^2} = id_{A^1} \circ f_1^*$ sowie $type_{A^2} \circ id_{A^2} = type_{A^2}$. Aus der Eindeutigkeit von $id_{A^1}^*$ als Mittler-Morphismus folgt $id_{A^1}^* = id_{A^2}$ und somit $\mathcal{P}^f(id_{A^1}) = id_{A^2}$. \square

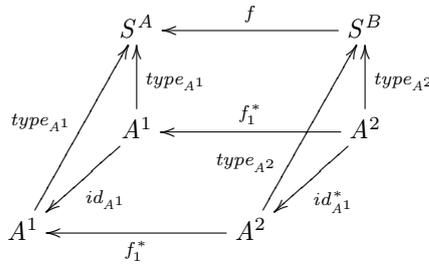


Abb. 11.6: \mathcal{P}^f überträgt Identitäten

Auf der rechten Seite ergibt sich eine ähnliche Situation. Die Komposition der Typisierung $type_{I\#}$ mit dem Refaktorisierungs-Morphismus r^t kann ebenfalls zu einem Funktor $\mathcal{F}^{r^t} : \mathbf{Alg}(MP)\downarrow S^\# \rightarrow \mathbf{Alg}(MP)\downarrow S'$ fortgesetzt werden:

Konstruktion 11.6 (Kompositions-Funktor). Sei ein $\mathbf{Alg}(MP)$ -Morphismus $f : S^A \rightarrow S^B$ gegeben. Dann sei eine Objekt-Zuordnung

$$\mathcal{F}_{\text{Ob}}^f : \text{Ob}^{\mathbf{Alg}(MP)\downarrow S^A} \rightarrow \text{Ob}^{\mathbf{Alg}(MP)\downarrow S^B}$$

und eine Morphismen-Zuordnung

$$\mathcal{F}_{\text{Mor}}^f : \text{Mor}^{\mathbf{Alg}(MP)\downarrow S^A} \rightarrow \text{Mor}^{\mathbf{Alg}(MP)\downarrow S^B}$$

folgendermaßen definiert:

- Für jedes Objekt $A \xrightarrow{type_A} S^A \in \text{Ob}^{\mathbf{Alg}(MP)\downarrow S^A}$ sei definiert:

$$\mathcal{F}_{\text{Ob}}^f(A \xrightarrow{\text{type}_A} S^A) ::= A \xrightarrow{f \circ \text{type}_A} S^B$$

- Für jeden Morphismus $m: (A^1 \xrightarrow{\text{type}_{A^1}} S^A) \rightarrow (A^2 \xrightarrow{\text{type}_{A^2}} S^A) \in \text{Mor}^{\mathbf{Alg}(MP)\downarrow S^A}$ sei definiert:

$$\mathcal{F}_{\text{Mor}}^f(m) ::= m$$

Dies ist wohldefiniert, denn für jeden $\mathbf{Alg}(MP)\downarrow S^A$ -Morphismus $m: (A^1 \xrightarrow{\text{type}_{A^1}} S^A) \rightarrow (A^2 \xrightarrow{\text{type}_{A^2}} S^A) \in \text{Mor}^{\mathbf{Alg}(MP)\downarrow S^A}$ gilt $\text{type}_{A^2} \circ m = \text{type}_{A^1}$ und somit auch $f \circ \text{type}_{A^2} \circ m = f \circ \text{type}_{A^1}$.

Lemma 11.7 (Kompositions-Funktor). $\mathcal{F}^f = (\mathcal{F}_{\text{Ob}}^f, \mathcal{F}_{\text{Mor}}^f)$ aus Konstruktion 11.6 ist ein Funktor $\mathcal{F}^f: \mathbf{Alg}(MP)\downarrow S^A \rightarrow \mathbf{Alg}(MP)\downarrow S^B$, genannt Kompositions-Funktor entlang f .

Beweis. Die Verträglichkeit bezüglich der Komposition und die Übertragung identischer Homomorphismen folgt unmittelbar aus der Definition von $\mathcal{F}_{\text{Mor}}^f$. \square

Beide Funktoren sind zueinander adjungiert, wie das folgende Lemma nachweist:

Lemma 11.8 (Pullback-Funktor und Kompositions-Funktor sind adjungiert). Für jeden $\mathbf{Alg}(MP)$ -Morphismus f gilt $\mathcal{F}^f \dashv \mathcal{P}^f$.

Beweis. Siehe [43, Theorem 11]. \square

Jetzt kann die Vermutung, dass die Pullback-Konstruktion M -Systeme auf M -Systeme abbildet, über Eigenschaften des Pullback-Funktors formuliert werden:

Lemma 11.9 (Pullback erhält M -Axiome). Sei ein $\mathbf{Alg}(MP)$ -Morphismus $f_S: S^B \rightarrow S^A$ gegeben. Dann lässt sich der Pullback-Funktor $\mathcal{P}^{f_S}: \mathbf{Alg}(MP)\downarrow S^A \rightarrow \mathbf{Alg}(MP)\downarrow S^B$ zu einem Funktor $\mathcal{P}^{f_S}: \mathbf{Sys}(S^A) \rightarrow \mathbf{Sys}(S^B)$ einschränken. Somit bildet \mathcal{P}^{f_S} M -Systeme auf M -Systeme und M -Homomorphismen auf M -Homomorphismen ab.

Beweis. Siehe Abschnitt C.3.1. \square

Auf der rechten Seite gelten die M -Axiome im System $I\# \xrightarrow{r^t \circ \text{type}_{I\#}} S'$ nicht in jedem Fall, wie das folgende Beispiel zeigt:

Beispiel 11.10 (Verschmelzen zweier Klassen in derselben Komponente). Das Beispiel in Abb. 11.7 verschmilzt im Schema zwei Klassen, die über eine Vererbungsbeziehung miteinander verbunden sind und sich nach Axiom (M.26) somit in derselben Komponente befinden. Das resultierende System erfüllt nach der Schema-Transformation das Axiom M.28 nicht mehr, weil ein ursprünglich zur Klasse A gehörendes Objekt nach der Migration nun aus zwei unterschiedlichen aber gleich typisierten Partikel innerhalb derselben Komponente besteht. \square

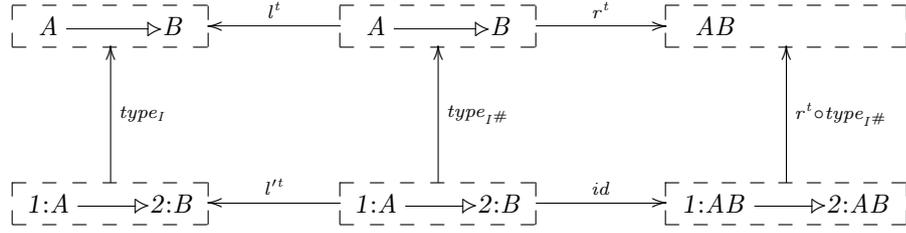


Abb. 11.7: Verschmelzen zweier Klassen in derselben Komponente

Aus diesem Grund ist eine zusätzliche explizite Anpassung an die M -Axiome notwendig, die in dem M -System $I' \xrightarrow{type_{I'}} S'$ resultiert. Diese Anpassung ist in Abschnitt 9.2 beschrieben und ist ein Funktor (vgl. Satz 9.18). Daraus ergibt sich schließlich:

Definition 11.11 (Migrations-Funktor). Sei eine Transformation $t: S \xrightarrow{S^\#} S'$ gegeben. Dann ist der Migrations-Funktor $\mathcal{M}^t: \mathbf{Sys}(S) \rightarrow \mathbf{Sys}(S')$ definiert als die Komposition des Pullback-Funktors, des Kompositions-Funktors und des Epirefektors in die Unterkategorie aller M -Systeme:

$$\mathcal{M}^t ::= \mathcal{F}^{S'} \circ \mathcal{F}^{r^t} \circ \mathcal{P}^{l^t}$$

Theorem 11.12 (Korrektheit der Migration von Daten und Prozessen). Sei $I \xrightarrow{type_I} S$ ein M -System, bestehend aus Daten und Prozessen I , die in einem Schema S durch $type_I$ typisiert sind. Sei eine Transformation $t: S \xrightarrow{S^\#} S'$ gegeben. Sei $I' \xrightarrow{type_{I'}} S' = \mathcal{M}^t(I \xrightarrow{type_I} S)$ das gemäß t migrierte System. Dann ist $I' \xrightarrow{type_{I'}} S'$ ein M -System.

Beweis. Dies folgt aus Lemma 11.9 und Satz 9.18. □

11.3 Migration von Programmen

Programme bestehen aus Systemen, die über Morphismen miteinander verbunden sind. Es existieren nun zwei Arten von Strukturen, die eine Transformation erhalten muss:

- (1) Noch abzuarbeitende Programme, also die Aktionen, die besagen, wie Daten und Prozesse während ihrer Laufzeit manipuliert werden.
- (2) Bereits abgearbeitete Programme, also in der Vergangenheit durch Aktionen durchgeführte Manipulationen an Prozessen und Daten.

Beide Punkte stellen sicher, dass Programme durch eine Transformation nicht zerstört werden. Punkt 1 ist offensichtlich: Er garantiert, dass existierende konsistente Aktionen durch eine Transformation in konsistente Aktionen übertragen werden. Punkt 2 ist subtiler, aber dennoch wichtig: Er stellt sicher, dass zu einem beliebigen Zeitpunkt alle bereits abgelaufenen Prozesse zum

aktuellen Programm “passen”. Das bedeutet, dass Prozesse und Programme zu jedem Zeitpunkt zueinander kompatibel sind. Prozesse und Programme werden somit nicht versioniert, es gibt immer genau eine Sicht auf das Gesamtsystem, nämlich diejenige zum aktuell betrachteten Schema. Dies vereinfacht das mathematische Modell, denn es müssen nicht mehrere, zu mehreren Schemata passende und zueinander inkompatible Informationen aufbewahrt werden.

Es gilt also zu zeigen, dass die Migration aus Abschnitt 11.2 die folgenden Eigenschaften besitzt:

- sie bildet konsistente Aktionen auf konsistente Aktionen ab;
- sie bildet partiell injektive Ansätze (aus der Menge IM) auf partiell injektive Ansätze ab;
- sie bildet Pushout-Diagramme auf Pushout-Diagramme ab.

Alle drei Eigenschaften stellen sicher, dass Migrationen weder die Ablauffähigkeit noch auszuführender Programme noch rückwirkend die Ablauffähigkeit bereits ausgeführter Programme verändern. Jedoch müssen Transformationen zusätzlich eingeschränkt werden, damit alle diese Eigenschaften bei einer Migration übertragen werden. Das nachfolgende Beispiel veranschaulicht die Problematik.

Beispiel 11.13 (Problem bei der Migration von Pushouts). Sei eine Transformation $t: S \overset{S^\#}{\rightsquigarrow} S'$ gegeben, die das Schema S in Abb. 11.8a in das Schema S' in Abb. 11.8c durch das Zusammenlegen der beiden Klassen B und C zu der Klasse BC durch r^t auf der rechten Seite der Schema-Transformation transformiert (die linke Seite l^t sei die Identität). Sei ein Pushout-Diagramm als die rechte Seite einer induzierten Graphstruktur-Transformation wie in Abb. 11.8b gegeben. Die Migration auf der rechten Seite via $\mathcal{F}^{S'} \circ \mathcal{F}^{r^t}$ führt zu dem Diagramm in Abb. 11.8d. Dabei werden die in dem Pushout-Objekt von dem A -Objekt ausgehenden Verknüpfungen und deren Ziele durch den Funktor $\mathcal{F}^{S'}$ zusammengelegt, um die Gültigkeit des Axioms (M.29) zu erzwingen. Das resultierende Diagramm ist in der Kategorie $\mathbf{Alg}(MP_*)$ jedoch offensichtlich kein Pushout-Diagramm mehr. Die induzierte Graphstruktur-Transformation wird somit nicht korrekt migriert.

Dieses Problem kann umgangen werden, indem das Zusammenlegen von Kanten in Schema-Transformationen verboten wird.¹⁰³ Dies führt zu der folgenden Definition einer *konsistenten* Transformation:

Definition 11.14 (Konsistente Transformation in $\mathbf{Alg}(MP)$). Eine Transformation $S \overset{l^t}{\leftarrow} S^\# \xrightarrow{r^t} S'$ in $\mathbf{Alg}(MP)$ ist konsistent, wenn r^t injektiv auf Nicht-self-Kanten ist, d. h. wenn gilt:

$$r_E^t(x) = r_E^t(y) \Rightarrow x = y \vee x, y \in \text{ran self}^{S^\#}$$

¹⁰³ Mit Kanten sind hier Elemente der Trägermenge zur Sorte E , also Assoziationen zu Klassen und Operationen sowie Parameter, gemeint. Assoziationen zu primitiven Datentypen sind nicht betroffen, da sie durch Attribute umgesetzt werden und Attribute durch keine Axiome identifiziert werden.

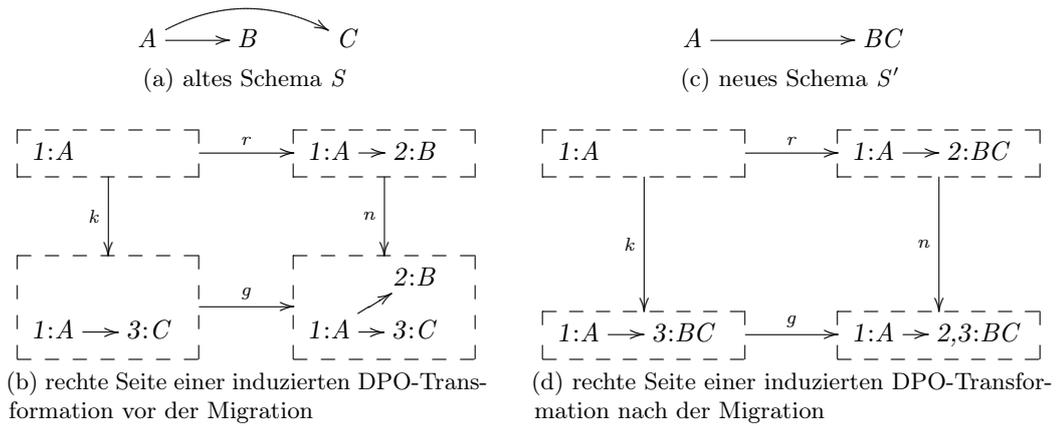


Abb. 11.8: Migration eines Pushouts ergibt nicht immer einen Pushout

für alle $x, y \in S_E^\#$.

Aufbauend auf dieser Definition, werden die oben genannten erwünschten Eigenschaften einer Migration in den folgenden Lemmata nachgewiesen. Zuvor wird jedoch in Lemma 11.15 eine Eigenschaft des Pullback-Funktors gezeigt, auf der die folgenden Beweise aufbauen.

Lemma 11.15 (Pullback-Lemma). Seien zwei $\mathbf{Sys}(S^1)$ -Objekte $A \xrightarrow{type_A} S^1$ und $B \xrightarrow{type_B} S^1$, ein $\mathbf{Sys}(S^1)$ -Morphismus $m: (A \xrightarrow{type_A} S^1) \rightarrow (B \xrightarrow{type_B} S^1)$ sowie eine Transformation $S^1 \xleftarrow{l} S^2 \xrightarrow{r} S^3$ gegeben. Seien $A' \xrightarrow{type_{A'}} S^2$, $B' \xrightarrow{type_{B'}} S^2$ und $m': (A' \xrightarrow{type_{A'}} S^2) \rightarrow (B' \xrightarrow{type_{B'}} S^2)$ die mit Hilfe des Pullback-Funktors migrierten Objekte bzw. der migrierte Morphismus (Abb. 11.9). Dann ist $A \xleftarrow{l_A} A' \xrightarrow{m'} B' \xrightarrow{l_B} B$ Pullback von $A \xleftarrow{m} B \xrightarrow{l_B} B'$.

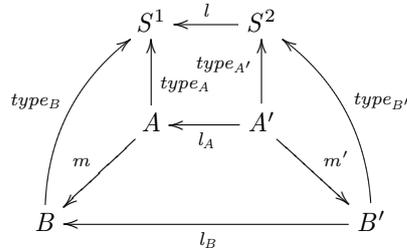


Abb. 11.9: Pullback-Lemma (1)

Beweis. Ein ‘‘Auffächern’’ des Diagramms in Abb. 11.9 ergibt das Diagramm in Abb. 11.10, in dem das äußere Rechteck und das rechte Quadrat nach Konstruktion Pullbacks sind. Aus [2, Satz 11.10] folgt, dass das linke Quadrat ebenfalls Pullback ist. \square

$$\begin{array}{ccccc}
A' & \xrightarrow{m'} & B' & \xrightarrow{\text{type}_{B'}} & S^2 \\
\downarrow l_A & & \downarrow l_B & & \downarrow l \\
A & \xrightarrow{m} & B & \xrightarrow{\text{type}_B} & S^1
\end{array}$$

Abb. 11.10: Pullback-Lemma (2)

11.3.1 Migration konsistenter Aktionen

Die Migration konsistenter Aktionen in konsistente Aktionen ist nach Definition 10.16 dann gegeben, wenn der Migrationsfunktor $\mathcal{F}^{S'} \circ \mathcal{F}^{r^t} \circ \mathcal{P}^{l^t}$

- vervollständigende Homomorphismen (Def. 10.11) sowie
- Kanten zurückziehende Spans (Def. 10.15)

überträgt. Dies wird in den folgenden Lemmata nachgewiesen. Zuerst wird der Pullback-Funktor auf der linken Seite einer Migration betrachtet.

Lemma 11.16 (Pullback-Funktor überträgt injektive Morphismen). *Seien eine Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein $\mathbf{Sys}(S)$ -Morphismus $f: (I^1 \xrightarrow{\text{type}_{I^1}} S) \rightarrow (I^2 \xrightarrow{\text{type}_{I^2}} S)$ gegeben. Sei f injektiv. Dann ist der $\mathbf{Sys}(S^\#)$ -Morphismus $\mathcal{P}^{l^t}(f)$ ebenfalls injektiv.*

Beweis. Nach Lemma A.80 sind Monomorphismen in $\mathbf{Alg}(MP)$ genau die injektiven Homomorphismen. Somit ist f $\mathbf{Alg}(MP)$ -Monomorphismus. Nach [43, Lemma 16] ist jeder $\mathbf{Alg}(MP)$ -Monomorphismus auch Monomorphismus in $\mathbf{Alg}(MP) \downarrow S$, somit ist f Monomorphismus in $\mathbf{Alg}(MP) \downarrow S$. Wiederum nach Lemma A.80 folgt, dass f Monomorphismus in $\mathbf{Sys}(S)$ ist. Nach Lemma 11.8 ist \mathcal{P}^{l^t} rechtsadjungiert und überträgt nach [2, Satz 18.9] Monomorphismen.¹⁰⁴ Somit ist $\mathcal{P}^{l^t}(f)$ Monomorphismus in der Kategorie $\mathbf{Sys}(S^\#)$. Durch eine entsprechende Argumentation folgt, dass $\mathcal{P}^{l^t}(f)$ injektiv ist. \square

Lemma 11.17 (Pullback-Funktor überträgt vervollständigende Morphismen). *Seien eine Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein $\mathbf{Sys}(S)$ -Morphismus $f: (I^1 \xrightarrow{\text{type}_{I^1}} S) \rightarrow (I^2 \xrightarrow{\text{type}_{I^2}} S)$ gegeben. Sei f vervollständigend. Dann ist der $\mathbf{Sys}(S^\#)$ -Morphismus $\mathcal{P}^{l^t}(f)$ ebenfalls vervollständigend.*

Beweis. Siehe Abschnitt C.3.2. \square

Lemma 11.18 (Pullback-Funktor überträgt das Zurückziehen von Kanten). *Seien eine Transformation $t: S \xrightarrow{S^\#} S'$ sowie der $\mathbf{Sys}(S)$ -Span $(I^1 \xrightarrow{\text{type}_{I^1}} S) \xleftarrow{l} (I^2 \xrightarrow{\text{type}_{I^2}} S) \xrightarrow{r}$*

¹⁰⁴ Genauer: Rechtsadjungierte Funktoren übertragen Limiten, jedoch lässt sich jeder Monomorphismus eindeutig als ein spezielles Pullback-Diagramm darstellen [2, Satz 11.16].

$(I^3 \xrightarrow{\text{type}_{I^3}} S)$, der Kanten zurückzieht (Def. 10.15), gegeben. Dann zieht der $\mathbf{Sys}(S^\#)$ -Span $\mathcal{P}^{I^t}(I^1 \xrightarrow{\text{type}_{I^1}} S) \xleftarrow{\mathcal{P}^{I^t}(l)} \mathcal{P}^{I^t}(I^2 \xrightarrow{\text{type}_{I^2}} S) \xrightarrow{\mathcal{P}^{I^t}(r)} \mathcal{P}^{I^t}(I^3 \xrightarrow{\text{type}_{I^3}} S)$ ebenfalls Kanten zurück.

Beweis. Siehe Abschnitt C.3.3. □

Der Kompositions-Funktor \mathcal{F}^{r^t} auf der rechten Seite einer Migration behält alle Morphismen unverändert bei, lediglich ein Schema-Wechsel wird durchgeführt. Durch diese Invarianz der Morphismen auf der Ausprägungsebene überträgt \mathcal{F}^{r^t} alle benötigten Eigenschaften – Injektivität, Vollheit, Vervollständigung und das Zurückziehen von Kanten entlang Spans. Für den Epirefektor $\mathcal{F}^{S'}$ hingegen muss die Übertragung dieser Eigenschaften nachgewiesen werden.

Lemma 11.19 (Epirefektor überträgt nicht alle strikt vollen Homomorphismen). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein Morphismus $f: (I^1 \xrightarrow{\text{type}_{I^1}} S') \rightarrow (I^2 \xrightarrow{\text{type}_{I^2}} S')$ in der Kategorie $\mathbf{Alg}(\mathbf{MP}) \downarrow S'$ gegeben. Sei f strikt voll. Dann ist der $\mathbf{Sys}(S')$ -Morphismus $\mathcal{F}^{S'}(f)$ nicht unbedingt strikt voll.*

Beweis. In Abb. 11.11 ist eine Situation dargestellt, in welcher der Epirefektor einen strikt vollen Homomorphismus nicht überträgt, wobei nur die Ausprägungsebene abgebildet ist. Der Homomorphismus $f: X \rightarrow Y$ auf der linken Seite der Abbildung ist strikt voll. Im Ziel von f ist der transitive Abschluss der Vererbungsrelation $\text{under}O^Y$ nicht gegeben, was durch den Epirefektor entsprechend korrigiert wird. Die Quelle hingegen verändert sich nicht, weil durch das fehlende Urbild von $2:B$ kein Anlass zur Anpassung der Vererbungsrelation $\text{under}O^X$ besteht. Daraus ergibt sich jedoch, dass der Homomorphismus $\mathcal{F}^{S'}(f)$ nicht strikt voll ist, weil $(\mathcal{F}^{S'}(f)_o(1:A), \mathcal{F}^{S'}(f)_o(3:C)) \in \text{under}O^{\mathcal{F}^{S'}(Y)}$ und $(1:A, 3:C) \notin \text{under}O^{\mathcal{F}^{S'}(X)}$ gilt. □

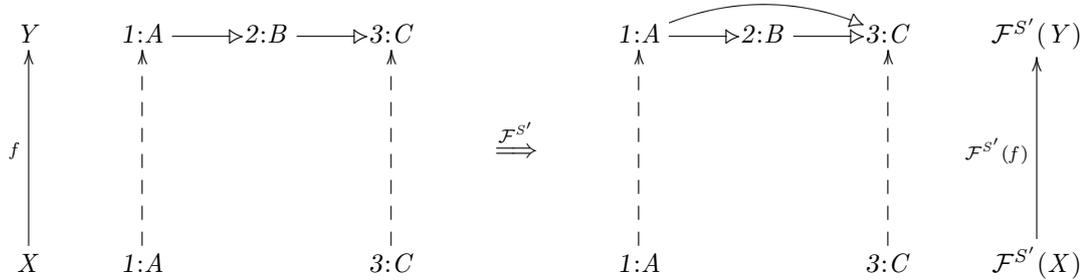


Abb. 11.11: $\mathcal{F}^{S'}(f)$ überträgt nicht alle strikt vollen Homomorphismen

Lemma 11.20 (Epirefektor überträgt vervollständigende Morphismen). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein Morphismus $f: (I^1 \xrightarrow{\text{type}_{I^1}} S') \rightarrow (I^2 \xrightarrow{\text{type}_{I^2}} S')$ in der Kategorie $\mathbf{Alg}(\mathbf{MP}) \downarrow S'$ gegeben. Sei f vervollständigend. Dann ist der $\mathbf{Sys}(S')$ -Morphismus $\mathcal{F}^{S'}(f)$ ebenfalls vervollständigend.*

Beweis. Siehe Abschnitt C.3.4. □

Lemma 11.21 (Epirefektor überträgt das Zurückziehen von Kanten). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein $\mathbf{Alg}(\mathbf{MP})\downarrow S'$ -Span $(I^1 \xrightarrow{\text{type}_{I^1}} S') \xleftarrow{l} (I^2 \xrightarrow{\text{type}_{I^2}} S') \xrightarrow{r} (I^3 \xrightarrow{\text{type}_{I^3}} S')$, der Kanten zurückzieht (Def. 10.15), gegeben. Dann zieht der $\mathbf{Sys}(S')$ -Span $\mathcal{F}^{S'}(I^1 \xrightarrow{\text{type}_{I^1}} S') \xleftarrow{\mathcal{F}^{S'}(l)} \mathcal{F}^{S'}(I^2 \xrightarrow{\text{type}_{I^2}} S') \xrightarrow{\mathcal{F}^{S'}(r)} \mathcal{F}^{S'}(I^3 \xrightarrow{\text{type}_{I^3}} S')$ ebenfalls Kanten zurück.*

Beweis. Siehe Abschnitt C.3.5. □

Zusammen ergibt sich der folgende Satz:

Satz 11.22 (Migration überträgt konsistente Aktionen). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ sowie eine konsistente Aktion $(I^1 \xrightarrow{\text{type}_{I^1}} S) \xleftarrow{l} (I^2 \xrightarrow{\text{type}_{I^2}} S) \xrightarrow{r} (I^3 \xrightarrow{\text{type}_{I^3}} S)$ in $\mathbf{Sys}(S)$ gegeben. Dann ist $\mathcal{M}^t(I^1 \xrightarrow{\text{type}_{I^1}} S) \xleftarrow{\mathcal{M}^t(l)} \mathcal{M}^t(I^2 \xrightarrow{\text{type}_{I^2}} S) \xrightarrow{\mathcal{M}^t(r)} \mathcal{M}^t(I^3 \xrightarrow{\text{type}_{I^3}} S)$ eine konsistente Aktion in $\mathbf{Sys}(S')$.*

Beweis. Dies folgt aus den Lemmata 11.16 bis 11.18 und 11.20 bis 11.21. □

11.3.2 Migration von Ansätzen

In diesem Abschnitt wird gezeigt, dass partiell injektive Ansätze durch eine Migration partiell injektiv bleiben.

Lemma 11.23 (Pullback-Funktor überträgt partiell injektive Ansätze). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein partiell injektiver Ansatz $m: (L \xrightarrow{\text{type}_L} S) \rightarrow (G \xrightarrow{\text{type}_G} S)$ in $\mathbf{Sys}(S)$ gegeben. Dann ist $\mathcal{P}^t(m)$ ein partiell injektiver Ansatz in $\mathbf{Sys}(S^\#)$.*

Beweis. Dies folgt direkt aus Lemma 11.16. □

Analog zur Übertragung konsistenter Aktionen überträgt \mathcal{F}^{r^t} partiell injektive Ansätze automatisch. Somit muss die Übertragung partiell injektiver Ansätze nur noch für den Epirefektor $\mathcal{F}^{S'}$ nachgewiesen werden.

Lemma 11.24 (Epirefektor überträgt partiell injektive Ansätze). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein partiell injektiver Ansatz $m: (L \xrightarrow{\text{type}_L} S) \rightarrow (G \xrightarrow{\text{type}_G} S)$ in $\mathbf{Alg}(\mathbf{MP})\downarrow S'$ gegeben. Dann ist $\mathcal{F}^{S'}(m)$ ein partiell injektiver Ansatz in $\mathbf{Sys}(S')$.*

Beweis. Dies folgt aus Lemma 11.20 und Lemma 10.14. □

Zusammen ergibt sich der folgende Satz:

Satz 11.25 (Migration überträgt partiell injektive Ansätze). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ sowie ein partiell injektiver Ansatz $m: (L \xrightarrow{\text{type}_L} S) \rightarrow (G \xrightarrow{\text{type}_G} S)$ in $\mathbf{Sys}(S)$ gegeben. Dann ist $\mathcal{M}^t(m)$ ein partiell injektiver Ansatz in $\mathbf{Sys}(S')$.*

Beweis. Dies folgt aus Lemma 11.23 und Lemma 11.24. □

11.3.3 Migration von Pushouts

In diesem Abschnitt wird nachgewiesen, dass die Migration aus Abschnitt 11.2 Pushouts überträgt. Dies wird für die linke und die rechte Seite der Migration separat betrachtet.

11.3.3.1 Pullback-Funktor

Auf der linken Seite der Migration wird der Pullback-Funktor benutzt, um Strukturen zu migrieren. Nach Lemma 11.8 ist dieser Funktor rechtsadjungiert, überträgt somit nach [2, Satz 18.9] Limiten. Da Pushouts jedoch spezielle Kolimiten sind, wird folgendermaßen vorgegangen:

- (1) Zuerst wird nachgewiesen, dass jeder Pushout, der durch Anwendung einer Aktion auf eine (Programm-)Graphstruktur entsteht, ein Pullback ist.
- (2) Dieses Pullback-Diagramm wird durch den Pullback-Funktor in ein Pullback-Diagramm der Zielkategorie übersetzt.
- (3) Schließlich wird gezeigt, dass das resultierende Pullback-Diagramm auch ein Pushout in der Zielkategorie ist.

Im Folgenden werden die Pullbacks und Pushouts nur in der Kategorie $\mathbf{Alg}(MP_*)$ betrachtet, weil ihre Pullback- bzw. Pushout-Eigenschaft in $\mathbf{Alg}(MP_*)\downarrow S$ bzw. $\mathbf{Alg}(MP_*)\downarrow S^\#$ aus [20, Fakt A.19(5) und Fakt A.23(4)] folgt.¹⁰⁵

Lemma 11.26 (Pushouts sind Pullbacks). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ und eine Aktion $(L \xrightarrow{type_L} S) \xleftarrow{l} (K \xrightarrow{type_K} S) \xrightarrow{r} (R \xrightarrow{type_R} S)$ gegeben. Sei ferner $L \xrightarrow{m} G \xleftarrow{f} D$ Pushout von $L \xleftarrow{l} K \xrightarrow{k} D$ in $\mathbf{Alg}(MP_*)$. Dann ist $L \xleftarrow{l} K \xrightarrow{k} D$ Pullback von $L \xrightarrow{m} G \xleftarrow{f} D$ in $\mathbf{Alg}(MP_*)$.*

Beweis. Nach Definition einer Aktion ist l injektiv. Somit folgt aus Lemma B.31 die zu beweisende Behauptung. \square

Somit liegt ein Pullback-Diagramm in $\mathbf{Alg}(MP_*)\downarrow S$ vor, das mit Hilfe des Pullback-Funktors \mathcal{P}^{l^t} in die Kategorie $\mathbf{Alg}(MP_*)\downarrow S^\#$ übertragen wird. Um nachzuweisen, dass das übertragene Pullback-Diagramm auch ein Pushout ist, wird die Erhaltung der folgenden drei Eigenschaften nachgewiesen:

- (1) \mathcal{P}^{l^t} überträgt \mathcal{M} -Morphismen nach Def. B.17;
- (2) \mathcal{P}^{l^t} überträgt gemeinsam surjektive Kospanns;

¹⁰⁵ Das Resultat dieses Abschnitts ergibt sich vermutlich ebenfalls aus der Adhäsionseigenschaft der Kategorie $\mathbf{Alg}(MP_*)^2$. Allerdings ist die Argumentation wegen der mehrfachen Anwendung des Pullback-Lemmas (Lemma 11.15) nicht unbedingt kürzer.

(3) \mathcal{P}^{l^t} überträgt partiell injektive Morphismenpaare nach Def. B.25.

Aus Lemma B.32 folgt dann die gewünschte Behauptung.

Lemma 11.27 (Pullback-Funktor überträgt \mathcal{M} -Morphismen). *Sei eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ gegeben. Sei f ein \mathcal{M} -Morphismus in der Kategorie $\mathbf{Alg}(MP_*) \downarrow S$. Dann ist $\mathcal{P}^{l^t}(f)$ ein \mathcal{M} -Morphismus in der Kategorie $\mathbf{Alg}(MP_*) \downarrow S^\#$.*

Beweis. Pullbacks in $\mathbf{Alg}(MP_*)$ können nach [20, Fakt A.23 (3)] komponentenweise konstruiert werden. Pullbacks in \mathbf{Set} übertragen injektive und bijektive Abbildungen [2, Sätze 7.36 und 11.18]. Daraus folgt die zu beweisende Behauptung. \square

Lemma 11.28 (Pullback-Funktor überträgt gemeinsam surjektive Morphismen). *Sei eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ gegeben. Sei $(I^1 \xrightarrow{\text{type}_{I^1}} S) \xrightarrow{l} (I^2 \xrightarrow{\text{type}_{I^2}} S) \xleftarrow{r} (I^3 \xrightarrow{\text{type}_{I^3}} S)$ ein gemeinsam surjektiver $\mathbf{Alg}(MP_*) \downarrow S$ -Kospan. Dann ist der Kospan*

$$\mathcal{P}^{l^t}(I^1 \xrightarrow{\text{type}_{I^1}} S) \xrightarrow{\mathcal{P}^{l^t}(l)} \mathcal{P}^{l^t}(I^2 \xrightarrow{\text{type}_{I^2}} S) \xleftarrow{\mathcal{P}^{l^t}(r)} \mathcal{P}^{l^t}(I^3 \xrightarrow{\text{type}_{I^3}} S)$$

gemeinsam surjektiv in $\mathbf{Alg}(MP_) \downarrow S^\#$.*

Beweis. Siehe Abschnitt C.3.6. \square

Lemma 11.29 (Pullback-Funktor überträgt partiell injektive Morphismenpaare). *Sei eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ gegeben. Seien zwei $\mathbf{Alg}(MP_*) \downarrow S$ -Morphismen $l: (K \xrightarrow{\text{type}_K} S) \rightarrow (L \xrightarrow{\text{type}_L} S)$ und $m: (L \xrightarrow{\text{type}_L} S) \rightarrow (G \xrightarrow{\text{type}_G} S)$ gegeben, wobei m injektiv bis auf l ist. Dann ist $\mathcal{P}^{l^t}(m)$ injektiv bis auf $\mathcal{P}^{l^t}(l)$ in der Kategorie $\mathbf{Alg}(MP_*) \downarrow S^\#$.*

Beweis. Siehe Abschnitt C.3.7. \square

Zusammen ergibt sich schließlich der folgende Satz:

Satz 11.30 (Pullback-Funktor überträgt Pushouts). *Es sei eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ gegeben. Sei*

$$(D \xrightarrow{\text{type}_D} S) \xrightarrow{g} (H \xrightarrow{\text{type}_H} S) \xleftarrow{n} (R \xrightarrow{\text{type}_R} S)$$

Pushout von

$$(D \xrightarrow{\text{type}_D} S) \xleftarrow{k} (K \xrightarrow{\text{type}_K} S) \xrightarrow{r} (R \xrightarrow{\text{type}_R} S)$$

in $\mathbf{Alg}(MP_) \downarrow S$. Dann ist*

$$\mathcal{P}^{l^t}(D \xrightarrow{\text{type}_D} S) \xrightarrow{\mathcal{P}^{l^t}(g)} \mathcal{P}^{l^t}(H \xrightarrow{\text{type}_H} S) \xleftarrow{\mathcal{P}^{l^t}(n)} \mathcal{P}^{l^t}(R \xrightarrow{\text{type}_R} S)$$

Pushout von

$$\mathcal{P}^{l^t}(D \xrightarrow{\text{type}_D} S) \xleftarrow{\mathcal{P}^{l^t}(k)} \mathcal{P}^{l^t}(K \xrightarrow{\text{type}_K} S) \xrightarrow{\mathcal{P}^{l^t}(r)} \mathcal{P}^{l^t}(R \xrightarrow{\text{type}_R} S)$$

in $\mathbf{Alg}(MP_*) \downarrow S^\#$.

Beweis. Dies folgt aus den Lemmata 11.26 bis 11.29. \square

11.3.3.2 Kompositions-Funktor

Der Kompositions-Funktor überträgt Pushout-Diagramme, weil er linksadjungiert zum Pullback-Funktor ist:

Satz 11.31 (Kompositions-Funktor überträgt Pushouts). *Sei eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ gegeben. Sei*

$$(D \xrightarrow{\text{type}_D} S^\#) \xrightarrow{g} (H \xrightarrow{\text{type}_H} S^\#) \xleftarrow{n} (R \xrightarrow{\text{type}_R} S^\#)$$

Pushout von

$$(D \xrightarrow{\text{type}_D} S^\#) \xleftarrow{k} (K \xrightarrow{\text{type}_K} S^\#) \xrightarrow{r} (R \xrightarrow{\text{type}_R} S^\#)$$

in $\mathbf{Alg}(MP_*) \downarrow S^\#$. Dann ist

$$\mathcal{F}^{r^t}(D \xrightarrow{\text{type}_D} S^\#) \xrightarrow{\mathcal{F}^{r^t}(g)} \mathcal{F}^{r^t}(H \xrightarrow{\text{type}_H} S^\#) \xleftarrow{\mathcal{F}^{r^t}(n)} \mathcal{F}^{r^t}(R \xrightarrow{\text{type}_R} S^\#)$$

Pushout von

$$\mathcal{F}^{r^t}(D \xrightarrow{\text{type}_D} S^\#) \xleftarrow{\mathcal{F}^{r^t}(k)} \mathcal{F}^{r^t}(K \xrightarrow{\text{type}_K} S^\#) \xrightarrow{\mathcal{F}^{r^t}(r)} \mathcal{F}^{r^t}(R \xrightarrow{\text{type}_R} S^\#)$$

in $\mathbf{Alg}(MP_*) \downarrow S'$.

Beweis. Nach Lemma 11.8 ist \mathcal{F}^{r^t} linksadjungiert und überträgt nach [2, Dualisierung von Satz 18.9] Kolimiten. \square

11.3.3.3 Epirefektor

Die Epireflexion in die Unterkategorie aller M -Systeme ist ein freier Funktor und überträgt somit Pushouts in die Unterkategorie. Allerdings ist dies nicht ausreichend, denn es muss sichergestellt werden, dass die Pushout-Diagramme nach der Anwendung des Funktors auch Pushouts in der *ursprünglichen* Kategorie $\mathbf{Alg}(MP_*)$ bzw. $\mathbf{Alg}(MP_*) \downarrow S'$ bleiben. Denn nur dann werden angewandte Graphstruktur-Transformationen auf angewandte Graphstruktur-Transformationen

abgebildet. Das ist jedoch nicht immer der Fall, wie in Beispiel 11.13 gezeigt wurde. In dem folgenden Lemma wird eine hinreichende Bedingung formuliert, die solche Fälle für beliebige algebraische Kategorien und entsprechende epirefektive Unterkategorien ausschließt. Die grundlegende Idee ist auszuschließen, dass die Anwendung des Epirefektors auf das Pushout-Objekt zwei Elemente zusammenlegt, deren Urbilder nicht aus demselben System stammen.

Lemma 11.32 (Epirefektor-Pushout-Lemma). *Seien $\Sigma = (S, OP, P)$ eine beliebige Signatur, $\text{Spec} = (\Sigma, H)$ eine Spezifikation zu dieser Signatur und $\mathcal{F}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(\text{Spec})$ ein Epirefektor in die Unterkategorie aller Spec -Systeme. Sei S ein Σ -System. Sei weiter*

$$(D \xrightarrow{\text{type}_D} S) \xrightarrow{g} (H \xrightarrow{\text{type}_H} S) \xleftarrow{n} (R \xrightarrow{\text{type}_R} S)$$

Pushout von

$$(D \xrightarrow{\text{type}_D} S) \xleftarrow{k} (K \xrightarrow{\text{type}_K} S) \xrightarrow{r} (R \xrightarrow{\text{type}_R} S)$$

in $\mathbf{Alg}(\Sigma) \downarrow S$. Dann ist

$$\mathcal{F}(D \xrightarrow{\text{type}_D} S) \xrightarrow{\mathcal{F}(g)} \mathcal{F}(H \xrightarrow{\text{type}_H} S) \xleftarrow{\mathcal{F}(n)} \mathcal{F}(R \xrightarrow{\text{type}_R} S)$$

Pushout von

$$\mathcal{F}(D \xrightarrow{\text{type}_D} S) \xleftarrow{\mathcal{F}(k)} \mathcal{F}(K \xrightarrow{\text{type}_K} S) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R \xrightarrow{\text{type}_R} S)$$

in $\mathbf{Alg}(\Sigma) \downarrow S$, wenn sich der Kern von u^H aus den Kernen der universellen Morphismen der Epireflexion u^R und u^D ergibt, wenn also gilt: Für jedes Prädikat $p \in P_w$ mit $w \in S^*$ und jedes Tupel $x \in (H \xrightarrow{\text{type}_H} S)_w$ mit $x \in \ker_p^{u^H}$ gibt es ein Tupel $x' \in (D \xrightarrow{\text{type}_D} S)_w$ mit:

$$\begin{aligned} g_w(x') &= x \\ x' &\in \ker_p^{u^D} \end{aligned}$$

oder ein Tupel $x' \in (R \xrightarrow{\text{type}_R} S)_w$ mit:

$$\begin{aligned} n_s(x') &= x \\ x' &\in \ker_p^{u^R} \end{aligned}$$

Beweis. Siehe Abschnitt C.3.8. □

Der nächste Satz zeigt, dass die in Lemma 11.32 geforderte Bedingung für konsistente Transformationen erfüllt ist:

Satz 11.33 (Epirefektor bewahrt Pushouts). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l} S^\# \xrightarrow{r} S'$ und eine konsistente Aktion $(L \xrightarrow{\text{type}_L} S') \xleftarrow{l} (K \xrightarrow{\text{type}_K} S') \xrightarrow{r} (R \xrightarrow{\text{type}_R} S')$ gegeben. Sei*

$$(D \xrightarrow{\text{type}_D} S') \xrightarrow{g} (H \xrightarrow{\text{type}_H} S') \xleftarrow{n} (R \xrightarrow{\text{type}_R} S')$$

Pushout von

$$(D \xrightarrow{\text{type}_D} S') \xleftarrow{k} (K \xrightarrow{\text{type}_K} S') \xrightarrow{r} (R \xrightarrow{\text{type}_R} S')$$

in $\mathbf{Alg}(MP_*) \downarrow S'$, wobei alle Systeme M' -Systeme sind. Dann ist

$$\mathcal{F}^{S'}(D \xrightarrow{\text{type}_D} S') \xrightarrow{\mathcal{F}^{S'}(g)} \mathcal{F}^{S'}(H \xrightarrow{\text{type}_H} S') \xleftarrow{\mathcal{F}^{S'}(n)} \mathcal{F}^{S'}(R \xrightarrow{\text{type}_R} S')$$

Pushout von

$$\mathcal{F}^{S'}(D \xrightarrow{\text{type}_D} S') \xleftarrow{\mathcal{F}^{S'}(k)} \mathcal{F}^{S'}(K \xrightarrow{\text{type}_K} S') \xrightarrow{\mathcal{F}^{S'}(r)} \mathcal{F}^{S'}(R \xrightarrow{\text{type}_R} S')$$

in $\mathbf{Alg}(MP_*) \downarrow S'$.

Beweis. Siehe Abschnitt C.3.9. □

11.3.3.4 Gesamte Migration

Nun kann bewiesen werden, dass die Migration Pushouts bewahrt und somit bereits abgelaufene Programme nach einer Schema-Transformation mit dem neuen Schema verträglich sind:

Satz 11.34 (Migration überträgt Pushouts). *Seien eine konsistente Transformation $t: S \xrightarrow{S^\#} S' = S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ und eine konsistente Aktion $(L \xrightarrow{\text{type}_L} S) \xleftarrow{l} (K \xrightarrow{\text{type}_K} S) \xrightarrow{r} (R \xrightarrow{\text{type}_R} S)$ gegeben. Sei*

$$(D \xrightarrow{\text{type}_D} S) \xrightarrow{g} (H \xrightarrow{\text{type}_H} S) \xleftarrow{n} (R \xrightarrow{\text{type}_R} S)$$

Pushout von

$$(D \xrightarrow{\text{type}_D} S) \xleftarrow{k} (K \xrightarrow{\text{type}_K} S) \xrightarrow{r} (R \xrightarrow{\text{type}_R} S)$$

in $\mathbf{Alg}(MP_*) \downarrow S$, wobei alle Systeme M -Systeme sind. Dann ist

$$\mathcal{M}^t(D \xrightarrow{\text{type}_D} S) \xrightarrow{\mathcal{M}^t(g)} \mathcal{M}^t(H \xrightarrow{\text{type}_H} S) \xleftarrow{\mathcal{M}^t(n)} \mathcal{M}^t(R \xrightarrow{\text{type}_R} S)$$

Pushout von

$$\mathcal{M}^t(D \xrightarrow{\text{type}_D} S) \xleftarrow{\mathcal{M}^t(k)} \mathcal{M}^t(K \xrightarrow{\text{type}_K} S) \xrightarrow{\mathcal{M}^t(r)} \mathcal{M}^t(R \xrightarrow{\text{type}_R} S)$$

in $\mathbf{Alg}(MP_*) \downarrow S'$.

Beweis. Dies folgt aus den Sätzen 11.30, 11.31 und 11.33. □

11.3.4 Zusammenfassung

Die Ergebnisse der letzten drei Abschnitte werden in dem folgenden Theorem zusammengefasst:

Theorem 11.35 (Korrektheit der Migration von Programmen). *Sei eine konsistente Transformation $t: S \xrightarrow{S^\#} S'$ gegeben. Dann überträgt der Migrations-Funktor \mathcal{M}^t konsistente Aktionen, partiell injektive Ansätze und angewandte Methoden von der Kategorie $\mathbf{Sys}(S)$ in die Kategorie $\mathbf{Sys}(S')$.*

Beweis. Aus Satz 11.22 und Satz 11.25 folgt die Übertragung von konsistenten Aktionen und partiell injektiven Ansätzen. Satz 11.34 stellt sicher, dass Doppel-Pushout-Diagramme von der Kategorie $\mathbf{Alg}(MP_*) \downarrow S$ in die Kategorie $\mathbf{Alg}(MP_*) \downarrow S'$ übertragen werden. Angewandte Methoden entsprechen DPO-Transformation entlang konsistenter Aktionen in der Kategorie $\mathbf{Alg}(MP_*) \downarrow S$ bzw. $\mathbf{Alg}(MP_*) \downarrow S'$. Zusammen mit Theorem 10.33 folgt somit, dass angewandte Methoden von der Kategorie $\mathbf{Sys}(S)$ in die Kategorie $\mathbf{Sys}(S')$ übertragen werden. \square

Anmerkung 11.36 (Erhaltung der Semantik). Die Betrachtungen in diesem Abschnitt stellen lediglich sicher, dass Programme nach einer induzierten Migration genauso ablaufen können wie vor der Migration. Es ist jedoch nicht untersucht worden, inwieweit die von den Programmen berechneten *Ergebnisse* erhalten bleiben und somit die Programme nach einer Migration verhaltensmäßig äquivalent zu den Programmen vor der Migration sind. Mehr hierzu steht in Abschnitt 14.2.4. \square

11.4 Komposition

Bei der Komposition zweier Transformationen geht es darum, zwei oder mehrere einzelne Transformationen zu einer einzigen Transformation zu verschmelzen. Diese Komposition von Transformationen hat mehrere Vorteile:

- *Abstraktion:* Bei beliebiger Komponierbarkeit von Transformationen ist es nicht mehr wichtig zu wissen, ob eine Transformation elementaren oder zusammenfassenden Charakter hat. Somit kann eine auf einer höheren Abstraktionsstufe stehende Schema-Änderung, die aus mehreren einzelnen Transformationen besteht, selbst als eine Transformation aufgefasst werden. Dies ermöglicht Top-down-¹⁰⁶ und Bottom-up-Entwurf¹⁰⁷ von komplexen Schema-Transformationen.
- *Performanz:* Die Umsetzung der in diesem Teil beschriebenen Migration in einem Software-Programm kann von komponierten Transformationen profitieren, weil kategorielle Konstruktionen wie das Bilden von Pullbacks oder die freie Konstruktion in die Unterkategorie aller M -Systeme nur einmal durchgeführt werden muss.

¹⁰⁶ vom Ganzen zum Elementaren aufteilend

¹⁰⁷ vom Elementaren zum Ganzen aufbauend

Jede Transformation ist ein spezieller Span. Spans werden in der Regel komponiert, indem der Pullback von dem rechten Morphismus des ersten Spans und von dem linken Morphismus des zweiten Spans gebildet wird [14]:

Definition 11.37 (Komposition von Transformationen). *Seien zwei beliebige Transformationen $t_1: S \overset{S^\#}{\rightsquigarrow} S' = S \xleftarrow{l^{t_1}} S^\# \xrightarrow{r^{t_1}} S'$ und $t_2: S' \overset{S^{\#\#}}{\rightsquigarrow} S'' = S' \xleftarrow{l^{t_2}} S^{\#\#} \xrightarrow{r^{t_2}} S''$ so gegeben, dass das Zielsystem der ersten Transformation und das Ausgangssystem der zweiten Transformation übereinstimmen. Dann ist die Komposition dieser Transformationen, $t_2 \circ t_1$, definiert als der Span $S \xleftarrow{l^{t_1} \circ \bar{l}^{t_2}} P^S \xrightarrow{r^{t_2} \circ \bar{r}^{t_1}} S''$, wobei $S^\# \xleftarrow{\bar{l}^{t_2}} P^S \xrightarrow{\bar{r}^{t_1}} S^{\#\#}$ Pullback von $S^\# \xrightarrow{r^{t_1}} S' \xleftarrow{l^{t_2}} S^{\#\#}$ in $\mathbf{Alg}(MP)$ ist (Abb. 11.12).*

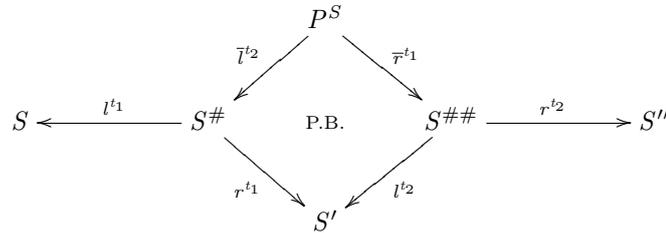


Abb. 11.12: Komposition von Transformationen

Das nachfolgende Lemma zeigt, dass konsistente Transformationen unter Komposition von Spans abgeschlossen sind:

Lemma 11.38 (Konsistente Transformationen sind abgeschlossen unter Komposition). *Seien zwei beliebige konsistente Transformationen $t_1: S \overset{S^\#}{\rightsquigarrow} S' = S \xleftarrow{l^{t_1}} S^\# \xrightarrow{r^{t_1}} S'$ und $t_2: S' \overset{S^{\#\#}}{\rightsquigarrow} S'' = S' \xleftarrow{l^{t_2}} S^{\#\#} \xrightarrow{r^{t_2}} S''$ gegeben, und sei $t_2 \circ t_1 = S \xleftarrow{l^{t_1} \circ \bar{l}^{t_2}} P^S \xrightarrow{r^{t_2} \circ \bar{r}^{t_1}} S''$ die komponierte Transformation, die aus der Komposition der beiden Spans in $\mathbf{Alg}(MP)$ entsteht. Dann ist $t_2 \circ t_1$ ebenfalls eine konsistente Transformation.*

Beweis. Es ist zu zeigen, dass $r_E^{t_2} \circ r_E^{t_1}$ injektiv ist. Nach Voraussetzung ist t_2 konsistent. Somit ist $r_E^{t_2}$ injektiv. Nach Voraussetzung ist t_1 ebenfalls konsistent, woraus die Injektivität von $r_E^{t_1}$ folgt. Weil Pullbacks in $\mathbf{Alg}(MP)$ komponentenweise über Pullbacks in \mathbf{Set} konstruiert werden können und Pullbacks in \mathbf{Set} injektive Abbildungen übertragen, folgt die Injektivität von $r_E^{t_1}$. Injektive Abbildungen sind unter Komposition abgeschlossen. Somit ist auch $r_E^{t_2} \circ r_E^{t_1}$ injektiv. \square

Diese “technische” Komponierbarkeit konsistenter Transformationen ist jedoch erst die eine Seite der Medaille. Natürlich soll auch die Migration von Daten, Programmen und Prozessen entlang der komponierten Transformation dasselbe Resultat liefern wie die komponierte Migration entlang den einzelnen Transformationen. Dass dies nicht immer der Fall ist, zeigt das nachfolgende Beispiel.

Beispiel 11.39 (Falsche Migration bei komponierter Transformation). Seien zwei beliebige konsistente Schema-Transformationen t_1 und t_2 wie in Abb. 11.13a und Abb. 11.13b gegeben. Die Komposition via Pullback ergibt die komponierte Schema-Transformation in Abb. 11.13c. Die Komposition der induzierten Migrationen einer beispielhaften Datenbank entlang den einzelnen Schema-Transformationen ist in Abb. 11.14a zu sehen, die induzierte Migration derselben Datenbank entlang der komponierten Schema-Transformation ist in Abb. 11.14b dargestellt. Offensichtlich unterscheiden sich die Ergebnisse der beiden Migrationen. \square

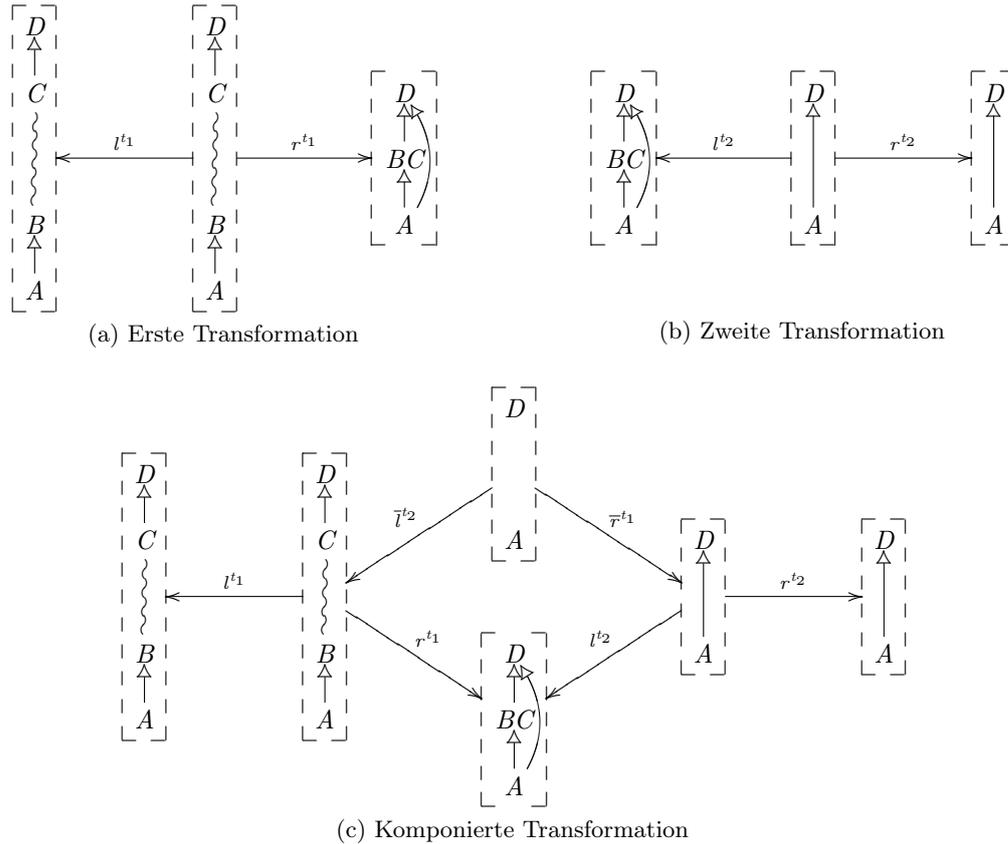
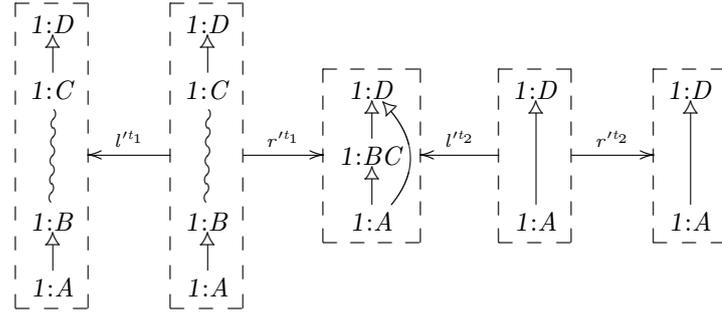


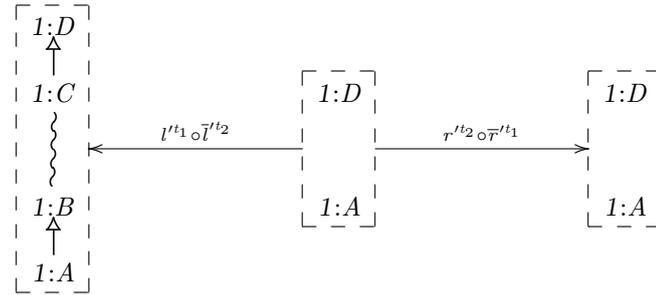
Abb. 11.13: Zwei einzelne Transformationen und deren Komposition

Das Beispiel zeigt, dass der Migrationsfunctor aus Def. 11.11 nicht für alle konsistente Transformationen verträglich mit der Komposition ist. Es gilt also, die möglichen Transformationen weiter einzuschränken, um Kompositionalität in allen Fällen zu gewährleisten. In [43] ist eine Bedingung formuliert, deren Erfüllung Kompositionalität sicherstellt:

Lemma 11.40 (Kompositionslemma). *Seien zwei beliebige konsistente Transformationen $t_1: S \xrightarrow{S^\#} S' = S \xleftarrow{l^1} S^\# \xrightarrow{r^1} S'$ und $t_2: S' \xrightarrow{S^{\#\#}} S'' = S' \xleftarrow{l^2} S^{\#\#} \xrightarrow{r^2} S''$ gegeben, und sei $t_2 \circ t_1 = S \xleftarrow{l^1 \circ \bar{l}^2} P^S \xrightarrow{r^2 \circ \bar{r}^1} S''$ die komponierte Transformation, die aus der Komposition*



(a) Komposition der induzierten Migrationen entlang den einzelnen Schema-Transformationen



(b) Induzierte Migration entlang der komponierten Schema-Transformation

Abb. 11.14: Aus den Schema-Transformationen abgeleitete Migrationen

der beiden Spans in $\mathbf{Alg}(MP)$ entsteht. Sei $I \xrightarrow{type_I} S$ ein zu migrierendes System und $I' \xrightarrow{type_{I'}} S'$ und $I'' \xrightarrow{type_{I''}} S''$ die Resultate der Migration nach der ersten bzw. zweiten Transformation (Abb. 11.15). Dann gilt: Sind der Pullback-Funktor \mathcal{P}^{l^2} und der Epirefektor \mathcal{F}^{M^*} vertauschbar, d. h. gilt

$$(11.1) \quad \mathcal{P}^{l^2} \circ \mathcal{F}^{M^*} \cong \mathcal{F}^{M^*} \circ \mathcal{P}^{l^2} \quad ,$$

dann folgt daraus:

$$\mathcal{M}^{t_2 \circ t_1} \cong \mathcal{M}^{t_2} \circ \mathcal{M}^{t_1}$$

Beweis. Siehe [43, Theorem 31]. □

Offenbar ist Bedingung (11.1) im obigen Beispiel verletzt. Und in der Tat scheint das Zusammenspiel der rechten Seite der ersten Transformation und der linken Seite der zweiten Transformation wesentlich für die Existenz einer mit Migrationen verträglichen Komposition zu sein. In dem präsentierten Beispiel wird auf der rechten Seite der ersten Transformation eine zusätzliche Vererbungsbeziehung $A \longrightarrow D$ hinzugefügt, um den transitiven Abschluss der Vererbungsrelation zu gewährleisten. Auf der linken Seite der zweiten Transformation hingegen wird die Notwendigkeit

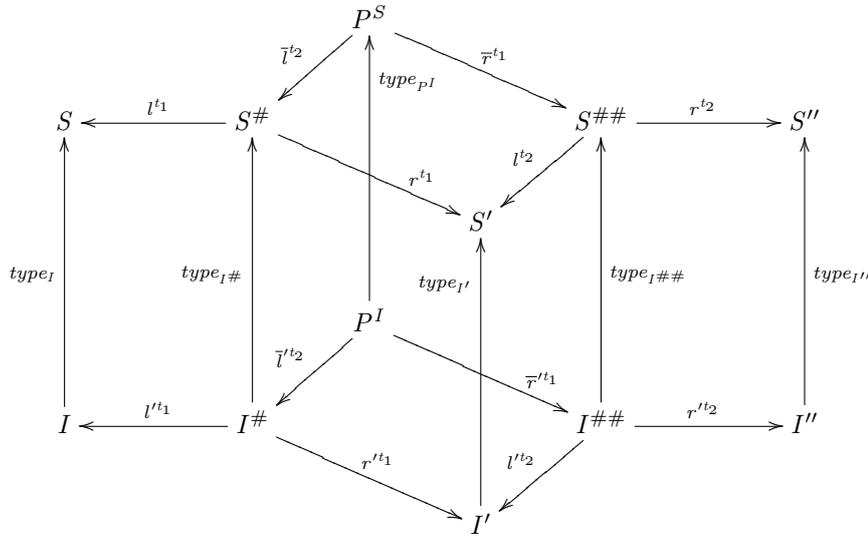


Abb. 11.15: Kompositionslemma

dieses transitiven Abschlusses eliminiert, indem die zwischen A und D liegende Klasse gelöscht wird. Um mit Migrationen verträgliche Kompositionalität zu erhalten, müssen somit entweder die rechten oder die linken Seiten von Transformationen geeignet eingeschränkt werden. In dieser Arbeit wird der Ansatz gewählt, die linke Seite einer Transformation mit Auflagen zu belegen.¹⁰⁸ Dazu fließt die Beobachtung, dass die Vererbungsbeziehung $A \longrightarrow D$ auf der linken Seite der zweiten Transformation ihren “Abkürzungscharakter” verliert, in die Definition spezieller Homomorphismen ein, die eben solche Abkürzungen reflektieren:

Definition 11.41 (Abkürzende Pfade zurückziehende Homomorphismen). Sei $f : A \rightarrow B$ ein MP-Homomorphismus. Dann zieht f abkürzende Pfade zurück, wenn es für je drei Elemente $x, z \in A_N$ und $y' \in B_N$ mit

$$\begin{aligned} (x, z) &\in \text{under}^A \\ (f_N(x), y') &\in \text{under}^B \\ (y', f_N(z)) &\in \text{under}^B \end{aligned}$$

ein Element $y \in A_N$ gibt mit:

$$\begin{aligned} f_N(y) &= y' \\ (x, y) &\in \text{under}^A \\ (y, z) &\in \text{under}^A \end{aligned}$$

¹⁰⁸ Das Einschränken der rechten Seite ist ein Ansatzpunkt für weitergehende Forschungsaktivitäten.

Diese Homomorphismen garantieren, dass jeder Quell-Pfad, der im Ziel eine Abkürzung um ein Element darstellt, in der Quelle eine Abkürzung um ein Urbild des Ziel-Elements sein muss. Ein Beispiel soll dies veranschaulichen. In Abb. 11.16a sind zwei Systeme und ein Homomorphismus f dazwischen abgebildet. Das linke System besteht aus zwei Partikeln x und z , wobei beide Partikel über das *under*-Prädikat miteinander in Beziehung stehen, was durch den Vererbungs-pfeil deutlich gemacht wird. Das rechte System besteht aus drei Partikeln x' , y' und z' , wobei x' und y' , y' und z' sowie x' und z' ebenfalls jeweils über das *under*-Prädikat miteinander in Beziehung stehen. Der Homomorphismus bildet die Partikel x und z des linken Systems auf die Partikel x' und z' des rechten Systems ab.

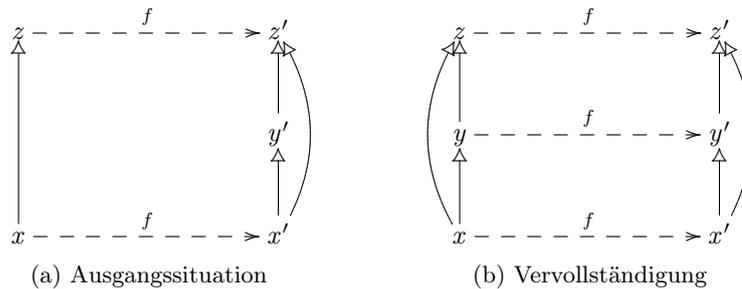


Abb. 11.16: Beispiel eines abkürzende Pfade zurückziehenden Homomorphismus'

Es ist somit die Situation vorhanden, dass die beiden Partikel x und z durch f so abgebildet werden, dass im Ziel ein zwischengeschaltetes Partikel y' auf dem Pfad zwischen x' und z' existiert. Wenn f abkürzende Pfade zurückzieht, dann folgt daraus die Existenz eines Urbilds y von y' , und zwar so, dass y ebenfalls ein zwischengeschaltetes Partikel auf dem Pfad zwischen x und z in der Quelle ist. Es liegt somit die Situation wie in Abb. 11.16b vor. Abkürzende Pfade zurückziehende Homomorphismen erlauben also in den betrachteten Kategorien, aus einem Vererbungs-pfad im Ziel auf einen mindestens genauso langen Vererbungs-pfad in der Quelle zu schließen, sofern Anfang und Ende des Pfades Urbilder besitzen.

Abkürzende Pfade zurückziehende Homomorphismen ziehen effektiv Pfade, die aus mehr als einem zwischengeschalteten Partikel bestehen, zurück, sofern die äußerste Abkürzung existiert. Dies zeigt das folgende Lemma.

Lemma 11.42 (Zurückziehen von kompletten Pfaden). Sei $f: A \rightarrow B$ ein abkürzende Pfade zurückziehender MP-Homomorphismus. Seien zwei Elemente $x, z \in A_N$ gegeben mit:

$$(11.2) \quad (x, z) \in \text{under}^A$$

Sei $(y_1 ::= f_N(x), y_2, \dots, y_n ::= f_N(z))$ ein n -elementiger Pfad in B mit $n \in \mathbb{N}^+$. Dann gibt es einen m -elementigen Pfad $(x_1 ::= x, x_2, \dots, x_m ::= z)$ in A mit:

$$(11.3) \quad \forall y_i \exists x_j : y_i = f_N(x_j)$$

Beweis. Siehe Abschnitt C.3.10. □

Die zwei folgenden Lemmata zeigen, dass abkürzende Pfade zurückziehende Homomorphismen unter Komposition und Pullback-Bildung abgeschlossen sind.

Lemma 11.43 (Abkürzende Pfade zurückziehende Homomorphismen sind abgeschlossen unter Komposition). *Seien $f: A \rightarrow B$ und $g: B \rightarrow C$ zwei MP-Homomorphismen, die abkürzende Pfade zurückziehen. Dann zieht $g \circ f$ ebenfalls abkürzende Pfade zurück.*

Beweis. Seien zwei Elemente $x, z \in A_N$ und ein Element $y' \in C_N$ gegeben mit:

$$(11.4) \quad (x, z) \in \text{under}^A$$

$$(11.5) \quad (g_N(f_N(x)), y') \in \text{under}^C$$

$$(11.6) \quad (y', g_N(f_N(z))) \in \text{under}^C$$

Aus (11.4) folgt, weil f Homomorphismus ist:

$$(11.7) \quad (f_N(x), f_N(z)) \in \text{under}^B$$

Weil g nach Voraussetzung abkürzende Pfade zurückzieht, folgt aus (11.7), (11.5) und (11.4), dass ein $\hat{y} \in B_N$ existiert mit:

$$(11.8) \quad g_N(\hat{y}) = y'$$

$$(11.9) \quad (f_N(x), \hat{y}) \in \text{under}^B$$

$$(11.10) \quad (\hat{y}, f_N(z)) \in \text{under}^B$$

Weil f nach Voraussetzung abkürzende Pfade zurückzieht, folgt aus (11.4), (11.9) und (11.10), dass ein $y \in A_N$ existiert mit:

$$(11.11) \quad f_N(y) = \hat{y}$$

$$(11.12) \quad (x, y) \in \text{under}^A$$

$$(11.13) \quad (y, z) \in \text{under}^A$$

Aus (11.8) und (11.11) folgt:

$$(11.14) \quad g_N(f_N(y)) = y'$$

Aus (11.14), (11.12) und (11.13) folgt, dass $g \circ f$ abkürzende Pfade zurückzieht. □

Lemma 11.44 (Pullbacks übertragen abkürzende Pfade zurückziehende Homomorphismen). Sei $S^\# \xleftarrow{\bar{l}^{t_2}} P^S \xrightarrow{\bar{r}^{t_1}} S^{\#\#}$ Pullback von $S^\# \xrightarrow{r^{t_1}} S' \xleftarrow{l^{t_2}} S^{\#\#}$ in $\mathbf{Alg}(MP)$ (Abb. 11.17). Ziehe l^{t_2} abkürzende Pfade zurück. Dann zieht auch \bar{l}^{t_2} abkürzende Pfade zurück.

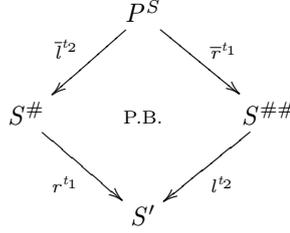


Abb. 11.17: Pullbacks übertragen abkürzende Pfade zurückziehende Homomorphismen

Beweis. Siehe Abschnitt C.3.11. □

Nun ist es möglich, *komponierbare* Transformationen auf der Grundlage von abkürzende Pfade zurückziehenden Homomorphismen zu definieren:

Definition 11.45 (Komponierbare Transformation). Eine konsistente Transformation $S \xleftarrow{l^t} S^\# \xrightarrow{r^t} S'$ in $\mathbf{Alg}(MP)$ ist komponierbar, wenn l^t abkürzende Pfade zurückzieht.

Bevor nachgewiesen wird, dass komponierbare Transformationen mit Migrationen verträglich sind, wird gezeigt, dass die Span-Komposition zweier komponierbarer Transformation wieder zu einer komponierbaren Transformation führt.

Lemma 11.46 (Komponierbare Transformationen sind abgeschlossen unter Komposition). Seien zwei beliebige komponierbare Transformationen $t_1: S \xrightarrow{S^\#} S' = S \xleftarrow{l^{t_1}} S^\# \xrightarrow{r^{t_1}} S'$ und $t_2: S' \xrightarrow{S^{\#\#}} S'' = S' \xleftarrow{l^{t_2}} S^{\#\#} \xrightarrow{r^{t_2}} S''$ gegeben, und sei $t_2 \circ t_1 = S \xleftarrow{l^{t_1} \circ \bar{l}^{t_2}} P^S \xrightarrow{r^{t_2} \circ \bar{r}^{t_1}} S''$ die komponierte Transformation, die aus der Komposition der beiden Spans in $\mathbf{Alg}(MP)$ entsteht. Dann ist $t_2 \circ t_1$ ebenfalls eine komponierbare Transformation.

Beweis. Nach Voraussetzung sind t_1 und t_2 konsistent, somit ziehen l^{t_1} und l^{t_2} abkürzende Pfade zurück. Nach Lemma 11.44 zieht somit auch \bar{l}^{t_2} abkürzende Pfade zurück. Nach Lemma 11.43 zieht schließlich $l^{t_1} \circ \bar{l}^{t_2}$ ebenfalls abkürzende Pfade zurück. Daraus folgt, dass $S \xleftarrow{l^{t_1} \circ \bar{l}^{t_2}} P^S \xrightarrow{r^{t_2} \circ \bar{r}^{t_1}} S''$ eine komponierbare Transformation ist. □

Damit komponierbare Transformationen mit Migrationen verträglich sind, ist nach Lemma 11.40 die Kompatibilität des Pullback-Funktors $\mathcal{P}^{l^{t_2}}$ und des Epirefektors \mathcal{F}^{M^*} nachzuweisen. Die Situation ist in Abb. 11.18 dargestellt. Das System $I^\# \xrightarrow{\text{type}_{I^\#}} S'$ ist ein Zwischenresultat der Migration *nach* der Anwendung des Kompositionsfunktors $\mathcal{F}^{r^{t_1}}$ aber *vor* der Anwendung des Epirefektors \mathcal{F}^{M^*} . Nun gibt es zwei Möglichkeiten:

- (1) Erst \mathcal{F}^{M^*} , dann $\mathcal{P}^{l'^2}$: Dies ergibt das System $I^{\#\#} \xrightarrow{\text{type}_{I^{\#\#}}} S^{\#\#}$.
- (2) Erst $\mathcal{P}^{l'^2}$, dann \mathcal{F}^{M^*} : Dies resultiert in dem System $\mathcal{F}^{M^*}(P) \xrightarrow{\mathcal{F}^{M^*}(t)} S^{\#\#}$.

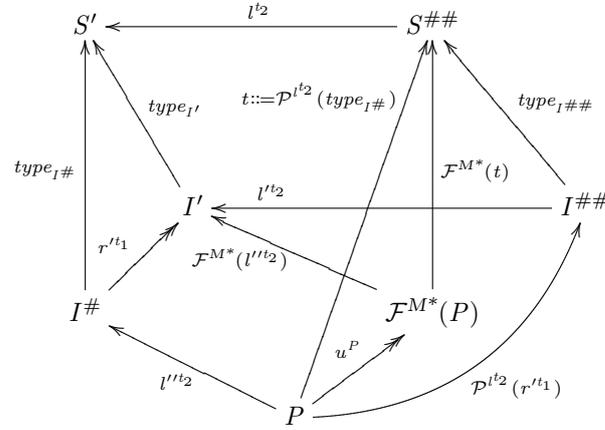


Abb. 11.18: Pullback-Epirefektor-Interaktion

Nach Lemma 11.40 ist die Komposition von Migrationen entlang einzelner Transformationen dann isomorph zur Migration entlang der Komposition einzelner Transformationen, wenn gilt:

$$(11.15) \quad I^{\#\#} \xrightarrow{\text{type}_{I^{\#\#}}} S^{\#\#} \cong \mathcal{F}^{M^*}(P) \xrightarrow{\mathcal{F}^{M^*}(t)} S^{\#\#}$$

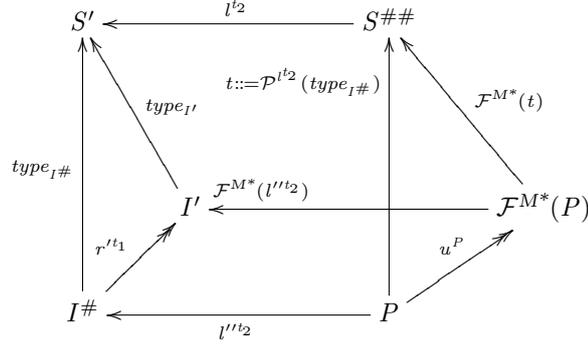
Dies wird in zwei Schritten bewiesen. Zuerst wird im Isomorphielemma 11.47 die Wahrheit von (11.15) unter der Voraussetzung bewiesen, dass $\mathcal{F}^{M^*}(t)$ und $\mathcal{F}^{M^*}(l'^2)$ gemeinsam strikt voll sind. In Lemma 11.48 wird gezeigt, dass diese Homomorphismen auf Grund von speziellen Eigenschaften des Epirefektors \mathcal{F}^{M^*} gemeinsam strikt voll sind.

Lemma 11.47 (Isomorphielemma). *Sei in $\mathbf{Alg}(MP)$ ein kommutierendes Diagramm wie in Abb. 11.18 gegeben, wobei*

- $I' \xleftarrow{l'^2} I^{\#\#} \xrightarrow{\text{type}_{I^{\#\#}}} S^{\#\#}$ Pullback von $I' \xrightarrow{\text{type}_{I'}} S' \xleftarrow{l'^2} S^{\#\#}$ und
- $I^{\#} \xleftarrow{l''^2} P \xrightarrow{t} S^{\#\#}$ Pullback von $I^{\#} \xrightarrow{\text{type}_{I^{\#}}} S' \xleftarrow{l'^2} S^{\#\#}$

in $\mathbf{Alg}(MP)$ ist. Seien $\mathcal{F}^{M^*}(l'^2)$ und $\mathcal{F}^{M^*}(t)$ gemeinsam strikt voll. Dann existiert ein Isomorphismus $i: (I^{\#\#} \xrightarrow{\text{type}_{I^{\#\#}}} S^{\#\#}) \rightarrow (\mathcal{F}^{M^*}(P) \xrightarrow{\mathcal{F}^{M^*}(t)} S^{\#\#})$ in $\mathbf{Alg}(MP) \downarrow S^{\#\#}$, so dass $I' \xleftarrow{\mathcal{F}^{M^*}(l'^2)} \mathcal{F}^{M^*}(P) \xrightarrow{\mathcal{F}^{M^*}(t)} S^{\#\#}$ Pullback von $I' \xrightarrow{\text{type}_{I'}} S' \xleftarrow{l'^2} S^{\#\#}$ in $\mathbf{Alg}(MP)$ ist.

Beweis. Siehe Abschnitt C.3.12. □

Abb. 11.19: Anwendung des Epirefektors \mathcal{F}^{M^*} auf ein Pullback-Diagramm

Lemma 11.48 (\mathcal{F}^{M^*} und gemeinsam strikt volle Morphismen). Sei in $\mathbf{Alg}(MP)$ ein kommutierendes Diagramm wie in Abb. 11.19 gegeben, wobei $I^{##} \xleftarrow{l''^{t_2}} P \xrightarrow{t} S^{##}$ Pullback von $I^{##} \xrightarrow{\text{type}_{I^{##}}} S' \xleftarrow{l^{t_2}} S^{##}$ in $\mathbf{Alg}(MP)$ ist. Dann sind $\mathcal{F}^{M^*}(l'^{t_2})$ und $\mathcal{F}^{M^*}(t)$ gemeinsam strikt voll.

Beweis. Siehe Abschnitt C.3.13. □

Daraus folgt:

Satz 11.49 (Epirefektor-Pullback-Vertauschbarkeit). Sei in $\mathbf{Alg}(MP)$ ein kommutierendes Diagramm wie in Abb. 11.18 gegeben, wobei

- $I' \xleftarrow{l'^{t_2}} I^{##} \xrightarrow{\text{type}_{I^{##}}} S^{##}$ Pullback von $I' \xrightarrow{\text{type}_{I'}} S' \xleftarrow{l^{t_2}} S^{##}$ und
- $I^{##} \xleftarrow{l''^{t_2}} P \xrightarrow{t} S^{##}$ Pullback von $I^{##} \xrightarrow{\text{type}_{I^{##}}} S' \xleftarrow{l^{t_2}} S^{##}$

in $\mathbf{Alg}(MP)$ ist. Dann folgt daraus die Existenz eines Isomorphismus' $i: (I^{##} \xrightarrow{\text{type}_{I^{##}}} S^{##}) \rightarrow (\mathcal{F}^{M^*}(P) \xrightarrow{\mathcal{F}^{M^*}(t)} S^{##})$ in $\mathbf{Alg}(MP) \downarrow S^{##}$, so dass $I' \xleftarrow{\mathcal{F}^{M^*}(l'^{t_2})} \mathcal{F}^{M^*}(P) \xrightarrow{\mathcal{F}^{M^*}(t)} S^{##}$ Pullback von $I' \xrightarrow{\text{type}_{I'}} S' \xleftarrow{l^{t_2}} S^{##}$ in $\mathbf{Alg}(MP)$ ist.

Beweis. Dies folgt unmittelbar aus Lemma 11.47 und Lemma 11.48. □

Auf diesem Korollar aufbauend ergibt sich schließlich das Kompositionstheorem:

Theorem 11.50 (Kompositionstheorem). Seien zwei beliebige komponierbare Transformationen $t_1: S \xrightarrow{S^{##}} S' = S \xleftarrow{l^{t_1}} S^{##} \xrightarrow{r^{t_1}} S'$ und $t_2: S' \xrightarrow{S^{##}} S'' = S' \xleftarrow{l^{t_2}} S^{##} \xrightarrow{r^{t_2}} S''$ gegeben. Dann gilt:

$$\mathcal{M}^{t_2 \circ t_1} \cong \mathcal{M}^{t_2} \circ \mathcal{M}^{t_1}$$

Beweis. Aus Satz 11.49 folgt:

$$\mathcal{P}^{l^{t_2}} \circ \mathcal{F}^{M^*} \cong \mathcal{F}^{M^*} \circ \mathcal{P}^{l^{t_2}}$$

Zusammen mit Lemma 11.40 folgt daraus die gewünschte Behauptung. □

Umsetzung des konzeptionellen Modells

In diesem Kapitel wird im Detail beschrieben, wie das konzeptionelle Modell ins mathematische Modell abgebildet wird. Dazu gehört die Untersuchung von semantischen Randbedingungen, die nicht über Axiome formalisiert bzw. formalisierbar sind. Zuerst wird in Abschnitt 12.1 die Umsetzung von Datenbanken beschrieben. In Abschnitt 12.2 wird die Umsetzung von Programmen und Prozessen erläutert. Schließlich enthält Abschnitt 12.3 einen Katalog von formalisierten Schema-Transformationen.

12.1 Datenbanken

In diesem Abschnitt werden die Umsetzung des Datenbank-Teils des konzeptionellen Modells beschrieben sowie zusätzliche semantische Bedingungen an Daten formuliert, die vom mathematischen Modell nicht sichergestellt werden können

12.1.1 Klassen und Objekte

Assoziationen und Verknüpfungen werden gemäß Abschnitt 8.1 im mathematischen Modell umgesetzt. Zusätzlich wird gefordert, dass jedes Objekt zu jeder seiner direkten und indirekten Oberklassen genau ein Partikel besitzt und somit vollständig ausgeprägt ist.¹⁰⁹

12.1.2 Assoziationen und Verknüpfungen

Assoziationen und Verknüpfungen werden gemäß Abschnitt 8.2 im mathematischen Modell umgesetzt. Zusätzlich wird gefordert, dass jede Assoziation zu einer benutzerdefinierten Klasse und jedes Attribut ausgeprägt wird und objektwertige Assoziationen und Attribute somit effektiv

¹⁰⁹ Das mathematische Modell fordert lediglich, dass jedes Objekt zu jeder seiner direkten und indirekten Oberklassen *höchstens* ein Partikel besitzt.

(0..*,1)-Assoziationen sind. Dies erlaubt Methoden davon auszugehen, dass beim Verändern von Variablen und Objekten immer eine alte Verknüpfung existiert, die gelöscht werden kann. Soll ein Objekt zu einer ausgehenden Assoziation keine Verknüpfung besitzen, muss eine Verknüpfung zu einem *Null*-Objekt erstellt werden.

Die Forderung nach der Ausprägung von Assoziationen existiert nicht für Assoziationen zu Operationen. Dies liegt daran, dass Verknüpfungen zu solchen Assoziationen nicht von Programmen manipuliert werden.¹¹⁰ Somit entfällt die Notwendigkeit nach einer *Null*-Operation sowie nach immer ausgeprägten Assoziationen zu Operationen.

12.2 Software

In diesem Abschnitt werden die Umsetzung des Software-Teils des konzeptionellen Modells beschrieben sowie zusätzliche semantische Bedingungen an Prozesse und Programme formuliert, die vom mathematischen Modell nicht sichergestellt werden können.

12.2.1 Prozesse

Operationen und Nachrichten werden gemäß Abschnitt 8.6 im mathematischen Modell umgesetzt. Zusätzlich wird gefordert, dass jede Nachricht zu jeder seiner direkten und indirekten Oberoperationen genau ein Partikel besitzt und somit vollständig ausgeprägt ist.¹¹¹

12.2.2 Programme

Dieser Abschnitt beschreibt die Abbildung der Methoden des konzeptionellen Modells in das mathematische Modell durch entsprechende Graphstruktur-Transformationen. In Abschnitt 5.2.1.5 werden Methoden bereits über eine Vorher/Nachher-Gegenüberstellung von Graphstrukturen dargestellt. Um daraus eine Aktion $L \xleftarrow{l} K \xrightarrow{r} R$ zu erhalten, wird wie folgt vorgegangen:

- (1) Die Graphstruktur L ist der Ausgangsgraph, der den Programm-Zustand *vor* der Anwendung der Methode darstellt.
- (2) Die Graphstruktur R ist der Zielgraph, der den Programm-Zustand *nach* der Anwendung der Methode darstellt.

¹¹⁰ Die einzige Ausnahme stellen Verknüpfungen zur Assoziation *current* von der Klasse *Processor* zur Operation *OpBase* dar, die von allen Methoden manipuliert wird. Eine Veränderung dieser Verknüpfungen ist jedoch inhärenter Teil der Nachrichtenverarbeitung und die Existenz dieser Verknüpfungen somit unabdingbare Voraussetzung für den Ablauf von Prozessen.

¹¹¹ Das mathematische Modell fordert lediglich, dass jede Nachricht zu jeder ihrer direkten und indirekten Oberoperationen *höchstens* ein Partikel besitzt.

- (3) Die Graphstruktur K ergibt sich aus dem *Schnitt* des Ausgangs- und Zielgraphen. Er enthält somit nur Elemente, die sowohl zu L als auch zu R gehören.
- (4) Die Graphstruktur-Morphismen l und r sind *Einbettungen* von K in L bzw. R .

Dabei sind folgende Rahmenbedingungen zu berücksichtigen:

- Es wird davon ausgegangen, dass auf der Ausprägungs-Ebene alle Elemente primitiver Datentypen existieren, so dass Terme entsprechend ausgewertet werden können.¹¹²
- Der Variablenvorrat einer Aktion berechnet sich aus der Vereinigung aller in dem Ausgangs- und Zielgraphen vorkommenden Variablen. Die in der Aktion benutzte Daten-Algebra ist die Term-Algebra über diesem Variablenvorrat. Beispielsweise enthält der Variablenvorrat zu der Methode *fac* in Abb. 5.16 die Variablen x und y zur Sorte $IV_{Integer}$.
- Elemente primitiver Datentypen, die innerhalb von Methoden verwendet werden, werden durch geeignete Terme zu der Daten-Signatur *Data* ausgedrückt. Dies ist notwendig, weil in Aktionen die Daten-Algebra der Term-Algebra zur Daten-Signatur (über einem passenden Variablenvorrat, s. o.) entspricht und somit nur Terme vorkommen können. Wird beispielsweise die operationale Struktur der natürlichen Zahlen durch eine Signatur mit einer Sorte N , einem Konstantensymbol $zero: \rightarrow N$ zur Selektion der Null und einem Operationssymbol $succ: N \rightarrow N$ zur Ermittlung des Nachfolgers einer Zahl repräsentiert, so wird die Konstante 2 durch den Term $succ(succ(zero))$ dargestellt. Eine Auswertung des Terms in der auf der Ausprägungs-Ebene tatsächlich verwendeten Daten-Algebra ergibt dann die gewünschte Konstante.
- Jeder im konzeptionellen Modell mehrfach verwendete Term besitzt im mathematischen Modell genau eine Repräsentation. Diese Anforderung folgt direkt aus der Eigenschaft von Term-Algebren, dass jeder Term genau einmal vorkommt. Beispielsweise werden bei der Umsetzung der Fakultätsfunktion aus Abb. 5.16c die vier Nullen alle auf denselben Term *zero* abgebildet.¹¹³
- Jeder unterstützte Operator, der auf Elementen primitiver Datentypen arbeitet, muss durch eine oder mehrere Operationen in der Daten-Algebra implementiert werden. Im einfachsten Fall existiert für jeden Operator genau ein zugehöriges Operationssymbol in der Signatur und eine semantisch dazu passende Operation in der Daten-Algebra.
- Verknüpfungen werden nie endgültig gelöscht, lediglich umgehängt. Soll eine Verknüpfung logisch gelöscht werden, muss auf der rechten Seite der Methode eine Verknüpfung zu einem *Null*-Objekt hinzugefügt werden. Dies stellt sicher, dass durch Anwendung von Methoden die Ausprägung aller Assoziationen zu benutzerdefinierten Klassen und aller Attribute erhalten bleibt (vgl. Abschnitt 12.1.2).

¹¹² Dies ist in der Praxis natürlich so nicht umsetzbar, weil es unendlich große Graphstrukturen zur Folge hat. Durch geeignete "lazy evaluation"-Techniken können die Elemente primitiver Datentypen jedoch auf jenen (endlichen) Teil beschränkt werden, der von dem jeweils vorhandenen Programmzustand benötigt wird. Vgl. hierzu auch [37, Abschnitt 3.2].

¹¹³ Vgl. hierzu auch [37, Abschnitt 3.2].

Schließlich muss der linke Teil einer Methode zusammenhängend sein, damit die Methode für eine Ausführung überhaupt in Frage kommt. Dies stellt sicher, dass das Löschen einer Operation die Methoden zu dieser Operation deaktiviert, weil die Löschung die linke Seite einer Methode effektiv in zwei Teile partitioniert: den Teil mit der Klasse *Processor* und der Operation *OpBase* sowie den Teil mit den Klassen und Operationen, auf welche die Parameter der gelöschten Operation verwiesen haben (vgl. die Beschreibung der Transformation “Löschen einer Operation” in Abschnitt 12.3.4.8).

12.3 Transformationen

Dieser Abschnitt präsentiert einen Katalog von Schema-Transformationen, aus denen die Transformation objektorientierter Systeme abgeleitet werden kann. Jede Schema-Transformation wird dabei als Span zwischen Ausgangs- und Ziel-Schema kodiert. Wie bei Graphstruktur-Transformationen besitzt das Zwischenschema zusammen mit den beiden ausgehenden Homomorphismen die Aufgabe, gleiche Elemente im Quell- und im Zielschema durch gleiche Urbilder zu identifizieren.

Dies soll anhand von zwei exemplarischen Schema-Transformationen demonstriert werden. Die erste Schema-Transformation dient der Erstellung einer Oberklasse und kann im mathematischen Modell wie in Abb. 12.1 formuliert werden (siehe Anmerkung 12.1 zur verwendeten Notation). Man beachte die Einführung von temporären Schema-Elementen (hier: Klassen B' und A'), die nur im Zwischensystem existieren. Diese Elemente sind notwendig, um die korrekte Identifikation der Elemente im Quell- und Zielschema zu gewährleisten. Insbesondere kann die neue Klasse C hier nicht einfach im Zielschema hinzugefügt werden. Denn alles, was auch Auswirkungen auf die Ausprägungsebene besitzen soll, muss bereits im Zwischenschema formuliert werden. Dazu gehört auch das Hinzufügen einer Oberklasse, weil die zugehörigen neuen Partikel in der Ausprägungsebene erzeugt werden müssen.

Die Aufgabe der zweiten Schema-Transformation ist das Verschieben des Anfangs einer Assoziation nach oben entlang einer Vererbungslinie. Sie kann wie in Abb. 12.2 formuliert werden. Man beachte auch hier die Einführung von temporären Schema-Elementen (hier: Klasse X) im Zwischensystem. Dies ist hier notwendig, damit die Abbildungen des Zwischenschemas ins Quell- und ins Zielsystem homomorph sind; ein einfaches Verschieben der Assoziation im Zwischen- oder Zielschema scheitert an den Homomorphie-Bedingungen.

Der Rest dieses Kapitels beschreibt, wie die konzeptionellen Transformationen aus [76, Kapitel 4] im mathematischen Modell formuliert werden. Die folgenden konzeptionellen Transformationen werden unterstützt:

- *Klassen und Vererbung*: Erzeugen einer eigenständigen Klasse, Erzeugen einer Oberklasse, Erzeugen einer Unterklasse, Verschmelzen zweier verwandter Klassen, Verschmelzen zweier nicht verwandter Klassen, Verschieben einer Oberklasse entlang einer Vererbungsbeziehung, Verschieben einer Unterklasse entlang einer Vererbungsbeziehung, Löschen einer Klasse.

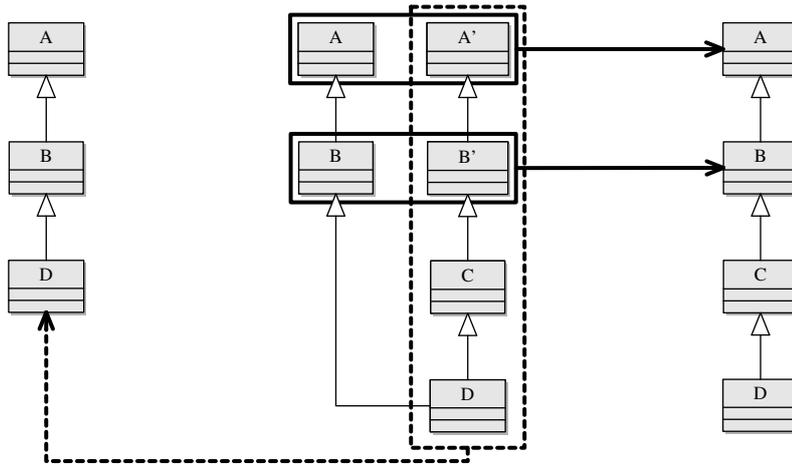


Abb. 12.1: Erzeugen einer Oberklasse

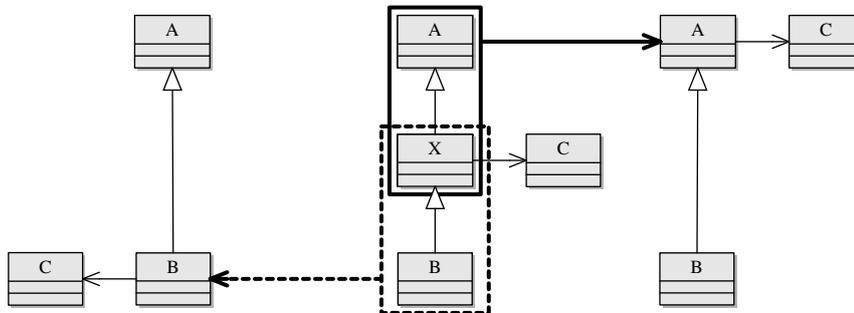


Abb. 12.2: Verschieben des Anfangs einer Assoziation zu einer Oberklasse

- *Assoziationen*: Erzeugen einer Assoziation, Verschieben des Anfangs einer Assoziation zu einer Oberklasse, Verschieben des Endes einer Assoziation zu einer Oberklasse, Löschen einer Assoziation.
- *Attribute*: Erzeugen eines Attributs, Verschieben eines Attributs zu einer Oberklasse, Löschen eines Attributs.
- *Operationen*: Erzeugen einer eigenständigen Operation, Erzeugen einer Oberoperation, Erzeugen einer Unteroperation, Verschmelzen zweier verwandter Operationen, Verschmelzen zweier nicht verwandter Operationen, Verschieben einer Oberoperation entlang einer Vererbungsbeziehung, Verschieben einer Unteroperation entlang einer Vererbungsbeziehung, Löschen einer Operation.
- *Parameter*: Erzeugen eines Parameters, Verschieben des Anfangs eines Parameters zu einer Oberoperation, Verschieben des Endes eines Parameters zu einer Oberoperation, Löschen eines Parameters.

Die konzeptionelle Transformation “Verschieben des Endes eines Parameters zu einer Oberklasse” wird nicht unterstützt. Mehr dazu steht in Abschnitt 14.2.2.

Während im konzeptionellen Modell die Beziehungen zwischen Elementen auf der linken und rechten Seite einer Transformation allein durch Namensgleichheit beschrieben wird, werden sie im mathematischen Modell in dem $\mathbf{Alg}(MP)$ -Span kodiert, der einer Transformation zugrunde liegt. Die zugehörige Migration der Ausprägungsebene wird nicht beschrieben, weil sich diese aus den Sätzen dieses Kapitels automatisch ergibt. Allerdings existieren in einigen wenigen Fällen Unterschiede zwischen den konzeptionellen und den mathematischen Transformationen, die dann ebenfalls in dem Kapitel behandelt werden.

Anmerkung 12.1 (Notation). Gleichnamige Elemente im Quell-, Zwischen- und Zielschema werden implizit aufeinander abgebildet, ohne dass dies durch Pfeile gesondert vermerkt wird. Kästen mit dicker gestrichelter Umrandung markieren Elemente im Zwischenschema, die aus einem einzigen Quellschema-Element entfaltet werden. Kästen mit dicker durchgezogener Umrandung markieren Elemente im Zwischenschema, die zu einem einzigen Zielschema-Element zusammengelegt werden.

□

12.3.1 Klassen und Vererbung

Dieser Abschnitt stellt Schema-Transformationen vor, die das Verändern der Klassen-Struktur zum Ziel haben. Neben dem Erzeugen neuer und Löschen alter Klassen werden Transformationen vorgestellt, die das Zusammenfassen von Klassen sowie die Veränderung der Vererbungsstruktur ermöglichen.

12.3.1.1 Erzeugen einer eigenständigen Klasse

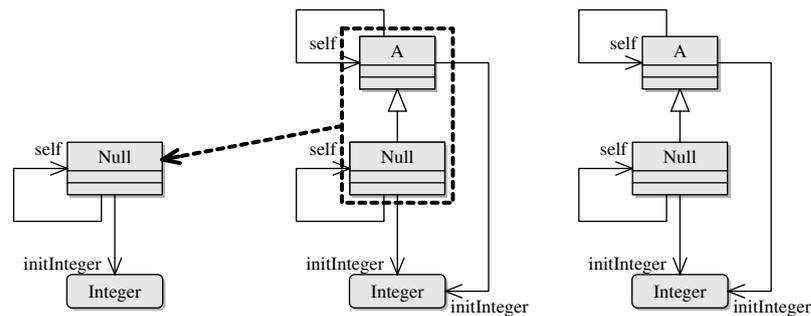


Abb. 12.3: Erzeugen einer eigenständigen Klasse

Beschreibung: Eine neue Klasse wird erzeugt (Abb. 12.3). Diese Klasse wird eine direkte Oberklasse der *Null*-Klasse. Weitere Vererbungsbeziehungen zu anderen Klassen werden nicht angelegt.

Typ: Refactoring

Motivation: Eine neue Klasse ermöglicht die Aufnahme neuer Informationen und neuen Verhaltens.

Voraussetzungen: —

Migration: Jedes *Null*-Objekt erhält ein neues Partikel zur neuen Klasse, das über eine Vererbungsverknüpfung mit dem jeweiligen *Null*-Partikel verbunden ist.

Umsetzung: Auf der linken Seite wird die Klasse *Null* in zwei Klassen aufgespalten, einmal die *Null*-Klasse selbst und einmal die neu zu erstellende Klasse. Die neue Klasse ist dabei eine direkte Oberklasse der *Null*-Klasse. Zusätzlich werden sowohl die *self*-Assoziation als auch die *initT*-Assoziationen für jeden primitiven Datentyp *T* dupliziert (in der Abbildung ist dies exemplarisch für den primitiven Datentype *Integer* dargestellt). Die rechte Seite stellt die Identität dar.

Anmerkung: Im Folgenden werden *self*- und *initT*-Assoziationen aus Gründen der Übersichtlichkeit nur dann explizit dargestellt, wenn aus ihnen andere Assoziationen entfaltet werden.

12.3.1.2 Erzeugen einer Oberklasse

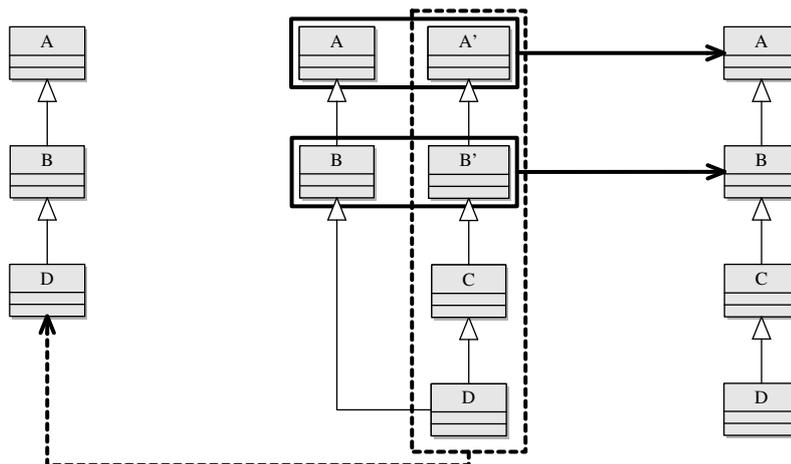


Abb. 12.4: Erzeugen einer Oberklasse

Beschreibung: Eine neue Klasse generalisiert eine nichtleere Menge bestehender Klassen (Abb. 12.4). Diese Klasse wird eine direkte Oberklasse der zu generalisierenden Klassen. Weitere Vererbungsbeziehungen zu anderen Klassen werden nicht angelegt.

Typ: Refactoring

Motivation: Die Generalisierung von Klassen erlaubt es, die Gemeinsamkeit in Hinblick auf Informationsgehalt und Verhalten einer Menge bestehender Klassen im Schema auszudrücken. In einem zweiten Schritt können Assoziationen und Operationen der spezielleren Klassen nach Bedarf zu der generelleren Klasse verschoben werden.

Voraussetzungen: —

Migration: Jedes Objekt zu einer der generalisierten Klassen erhält ein zusätzliches Partikel zur neuen Oberklasse, das über eine Vererbungsverknüpfung mit dem Partikel der generalisierten Klasse verbunden ist.

Umsetzung: Auf der linken Seite werden die neuen Oberklassen aus der Unterklasse entfaltet. Werden die neuen Oberklassen zwischen zwei bestehenden Klassen eingefügt, dann werden zusätzlich alle alten direkten und indirekten Oberklassen mit entfaltet. Auf der rechten Seite werden die bereits existierenden Oberklassen mit den duplizierten Oberklassen zusammengelegt, so dass nur die neuen Oberklassen als neue Schema-Elemente verbleiben.

Anmerkung 12.2 (Duplizierung der gesamten Klassenhierarchie). In der Transformation wird die komplette echte Oberklassenhierarchie der Unterklasse *D* durch Entfaltung dupliziert, anstatt nur die Oberklasse *B* zu kopieren. Der Grund ist, dass nur so gewährleistet werden kann, dass auch die migrierte Ausprägung zum Zwischen-Schema ein Objekt immer alle nötigen Partikel besitzt, falls dies in der ursprünglichen Ausprägung auch gilt. Diese Anforderung kann jedoch fallen gelassen werden, wenn festgelegt wird, dass diese Invariante nur für die ursprüngliche und die komplett migrierte Ausprägung gelten muss. In diesem Fall werden diese und andere Transformationen einfacher, weil weniger temporäre Klassen entfaltet werden müssen. □

12.3.1.3 Erzeugen einer Unterklasse

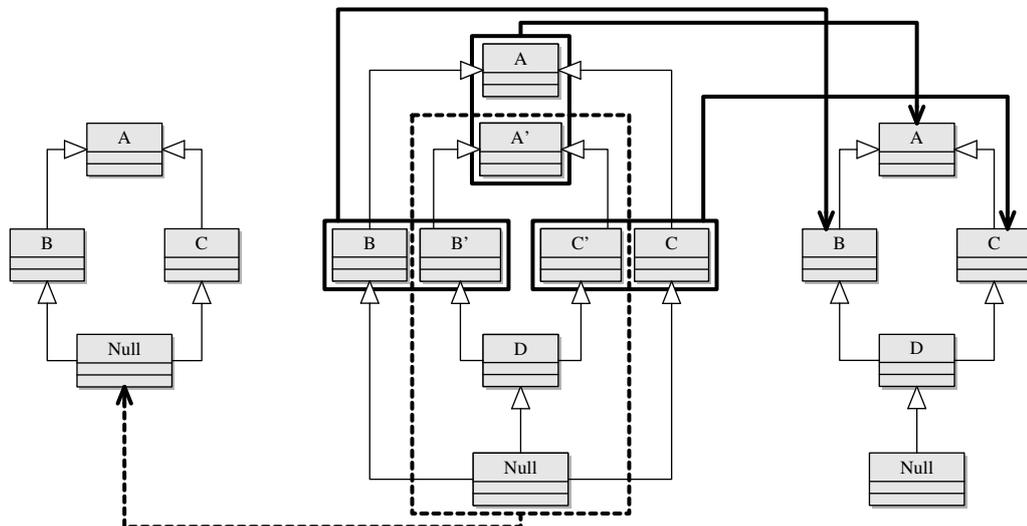


Abb. 12.5: Erzeugen einer Unterklasse

Beschreibung: Eine neue Klasse spezialisiert eine nichtleere Menge bestehender Klassen (Abb. 12.5). Diese Klasse wird eine direkte Oberklasse der *Null*-Klasse und direkte Unterklasse der

zu spezialisierenden Klassen. Weitere Vererbungsbeziehungen zu anderen Klassen werden nicht angelegt.

Typ: Refactoring

Motivation: Die Spezialisierung von Klassen erlaubt es, zusätzliche Informationen und zusätzliches Verhalten für neue Objekte zu diesen Typen hinzuzufügen.

Voraussetzungen: —

Migration: Jedes *Null*-Objekt erhält ein neues Partikel zur neuen Klasse, das über Vererbungsverknüpfungen mit dem jeweiligen *Null*-Partikel und den jeweiligen Partikeln zu den spezialisierten Klassen verbunden ist. Objekte zu anderen Klassen werden nicht verändert.

Umsetzung: Auf der linken Seite wird die neue Unterklasse zusammen mit all ihren Oberklassen aus der *Null*-Klasse entfaltet. Auf der rechten Seite werden die bereits existierenden Oberklassen mit den erneut herausgezogenen Oberklassen zusammengelegt, so dass nur die Unterklasse als neues Schema-Element verbleibt.

12.3.1.4 Verschmelzen zweier verwandter Klassen

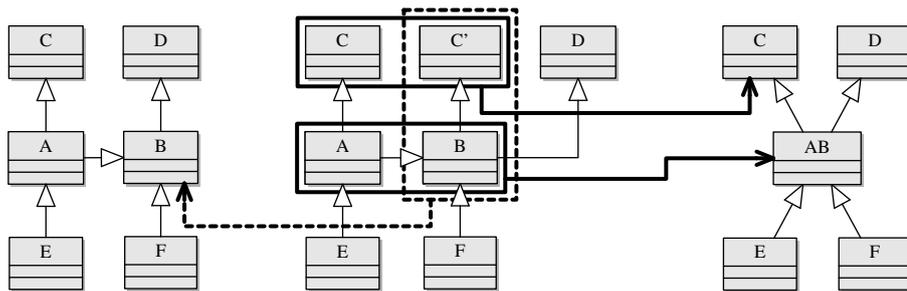


Abb. 12.6: Verschmelzen zweier verwandter Klassen

Beschreibung: Zwei Klassen $A \longrightarrow B$, die miteinander über eine (direkte) Vererbungsbeziehung verbunden sind, werden zu einer Klasse AB zusammengelegt (Abb. 12.6). Die Oberklassen von AB sind die Oberklassen von A (B ausgenommen) und die Oberklassen von B . Die Unterklassen von AB sind die Unterklassen von A und die Unterklassen von B (A ausgenommen). Alle Assoziationen, die mit den Klassen A oder B verbunden sind, werden mit der Klasse AB verbunden; entsprechendes gilt für Parameter von Operationen.

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, Abstraktionen, die sich als zu fein herausgestellt haben, wieder zu vereinen.

Voraussetzungen: Alle Methoden der zu verschmelzenden Klassen gehören unterschiedlichen Operationen an.

Migration: Miteinander über eine Vererbungsverknüpfung verbundene Partikel zu den beiden Klassen A und B werden zusammengelegt. Andere A - und B -Partikel werden zu AB -Partikeln umtypisiert. Objekte zum Typ B , die nicht auch Objekte zum Typ A sind, erhalten für jede Klasse in der echten Oberklassenhierarchie von A (die Oberklassenhierarchie von B ausgenommen) ein entsprechendes neues Partikel.

Umsetzung: Auf der linken Seite werden alle Oberklassen der Unterklasse aus der Oberklasse entfaltet. Auf der rechten Seite werden zum einen die Unter- und Oberklasse und zum anderen die nun doppelt vorhandenen Oberklassen zusammengelegt.

12.3.1.5 Verschmelzen zweier nicht verwandter Klassen

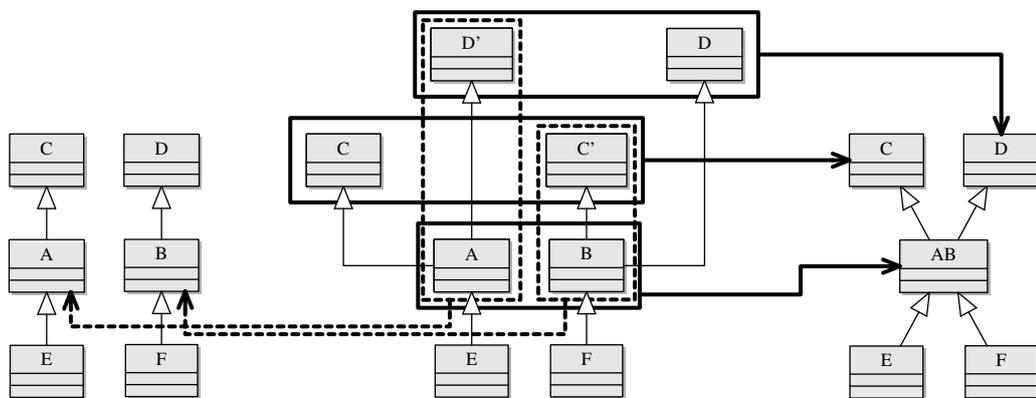


Abb. 12.7: Verschmelzen zweier nicht verwandter Klassen

Beschreibung: Zwei Klassen A und B , die nicht miteinander über eine Vererbungsbeziehung verbunden sind, werden zu einer Klasse AB zusammengelegt (Abb. 12.7). Die Oberklassen von AB sind die Oberklassen von A und die Oberklassen von B . Die Unterklassen von AB sind die Unterklassen von A und die Unterklassen von B . Alle Assoziationen, die mit den Klassen A oder B verbunden sind, werden mit der Klasse AB verbunden; entsprechendes gilt für Parameter von Operationen.

Typ: Refactoring

Motivation: Diese Transformation ist nützlich, wenn Schemata unterschiedlicher Herkunft aber mit gleicher Semantik vereinheitlicht werden sollen.

Voraussetzungen: Alle Methoden der zu verschmelzenden Klassen gehören unterschiedlichen Operationen an.

Migration: A - und B -Partikel werden zu AB -Partikeln umtypisiert. Objekte zum Typ B , die nicht auch Objekte zum Typ A sind, erhalten für jede Klasse in der echten Oberklassenhierarchie von A ein entsprechendes neues Partikel. Objekte zum Typ A , die nicht auch Objekte zum Typ B

sind, erhalten für jede Klasse in der echten Oberklassenhierarchie von B ein entsprechendes neues Partikel.

Umsetzung: Auf der linken Seite werden alle Oberklassen der zusammenzulegenden Klassen aus der jeweils gegensätzlichen Klasse entfaltet, so dass in der Mitte beide Klassen eine identische Oberklassenhierarchie besitzen. Auf der rechten Seite werden zum einen die beiden zusammenzulegenden Klassen und zum anderen die nun doppelt vorhandenen Oberklassen zusammengelegt.

12.3.1.6 Verschieben einer Oberklasse entlang einer Vererbungsbeziehung

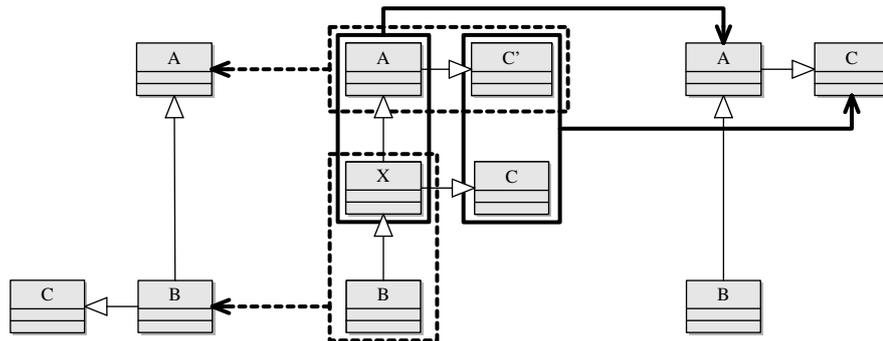


Abb. 12.8: Verschieben einer Oberklasse entlang einer Vererbungsbeziehung

Beschreibung: Eine direkte Oberklasse V einer bestehenden Klasse Q wird indirekte Oberklasse, indem sie entlang einer (direkten) Vererbungsbeziehung $Q \longrightarrow Z$ verschoben wird (Abb. 12.8).¹¹⁴

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, vorhandene Generalisierungen zu verallgemeinern und somit Abstraktionen in einer Klassen-Hierarchie hinzuzufügen.

Voraussetzungen: —

Migration: In jedem Q -Objekt wird das V -Partikel vom Q -Partikel gelöst und mit dem Z -Partikel verbunden. Jedes Z -Objekt erhält ein neues Partikel zu jeder Klasse in der Oberklassenhierarchie von V .

Umsetzung: Auf der linken Seite wird aus der Quellklasse eine temporäre Oberklasse entfaltet, so dass diese sich zwischen Quell- und Zielklasse befindet. Dabei wird die zu verschiebenden Oberklasse Oberklasse der temporären Klasse. Auf der rechten Seite wird die temporäre Klasse mit der Zielklasse zusammengelegt.

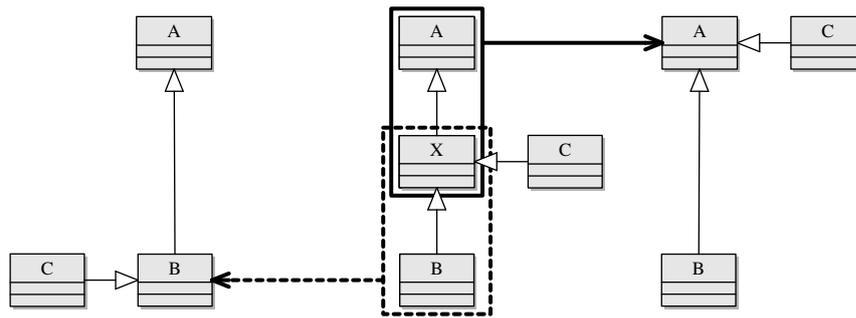


Abb. 12.9: Verschieben einer Unterklasse entlang einer Vererbungsbeziehung

12.3.1.7 Verschieben einer Unterklasse entlang einer Vererbungsbeziehung

Beschreibung: Eine indirekte Unterklasse V einer bestehenden Klasse Z wird direkte Unterklasse, indem sie entlang einer (direkten) Vererbungsbeziehung $Q \longrightarrow Z$ verschoben wird (Abb. 12.9).

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, vorhandene Spezialisierungen zu verallgemeinern und somit Abstraktionen aus einer Klassen-Hierarchie zu entfernen.

Voraussetzungen: —

Migration: In jedem V -Objekt wird das V -Partikel vom Q -Partikel gelöst und mit dem Z -Partikel verbunden.

Umsetzung: Auf der linken Seite wird aus der Quellklasse eine temporäre Oberklasse entfaltet, so dass diese sich zwischen Quell- und Zielklasse befindet. Dabei wird die zu verschiebende Unterklasse Unterklasse der temporären Klasse. Auf der rechten Seite wird die temporäre Klasse mit der Zielklasse zusammengelegt.

Anmerkung: Jedes V -Objekt besitzt nach der Transformation weiterhin das entsprechende Q -Partikel und ist somit mehrfach klassifiziert, sofern kein Partikel zu einer gemeinsamen Unterklasse von V und Q existiert.

12.3.1.8 Löschen einer Klasse

Beschreibung: Eine bestehende Klasse wird samt allen ein- und ausgehenden Vererbungsbeziehungen, Assoziationen und Parametern von Operationen gelöscht (Abb. 12.10).

Typ: Entwicklung

Motivation: Das Löschen einer Klasse ermöglicht das Entfernen nicht benötigter Informationen.

Voraussetzungen: —

¹¹⁴ Q steht für Quelle, Z für Ziel, V für die zu verschiebende Klasse.

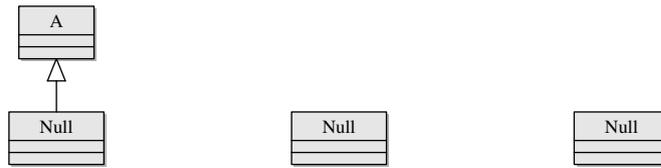


Abb. 12.10: Löschen einer Wurzelklasse

Migration: Alle in der gelöschten Klasse und den gelöschten Assoziationen und Parametern typisierten Partikel, Verknüpfungen und Argumente werden gelöscht. Dabei können Objekte in mehrere Objekte aufgespalten werden, wenn die zu löschende Klasse über Vererbungsbeziehungen als Verbindungsglied zwischen mehreren Klassen fungiert. Auch mehrfach klassifizierte Objekte können entstehen, wenn die nach unten abschließende Klasse einer Raute in der Vererbungshierarchie gelöscht wird.

Umsetzung: Auf der linken Seite wird die Klasse samt allen ein- und ausgehenden Vererbungsbeziehungen, Assoziationen und Parametern von Operationen gelöscht. Die rechte Seite stellt die Identität dar.

Anmerkung: Wird eine Zwischenklasse gelöscht, die also sowohl Unterklasse als auch Oberklasse ist, und werden nicht gleichzeitig *alle* Vererbungsbeziehungen unterhalb und oberhalb der zu löschenden Klasse mitgelöscht (Abb. 12.11), dann ist die Transformation nicht komponierbar (vgl. Abschnitt 11.4). Weil die Klasse *Null* Unterklasse aller anderen Klassen ist, folgt daraus, dass nur die Löschung von Wurzelklassen (wie in Abb. 12.10 dargestellt) mit anderen Transformationen komponierbar ist.

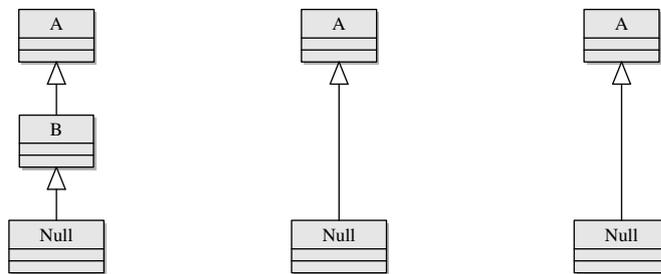


Abb. 12.11: Löschen einer Zwischenklasse (nicht komponierbar)

12.3.2 Assoziationen

In diesem Abschnitt sind Schema-Transformationen zu finden, die sich auf Assoziationen beziehen.¹¹⁵ Neben dem Erzeugen und Löschen von Assoziationen werden auch Transformationen vorgestellt, die Quellen und/oder Ziele von Assoziationen entlang einer Vererbungslinie verschieben.

¹¹⁵ Dies schließt Attribute nicht ein; entsprechende Transformationen werden in Abschnitt 12.3.3 vorgestellt.

12.3.2.1 Erzeugen einer Assoziation zu einer Klasse

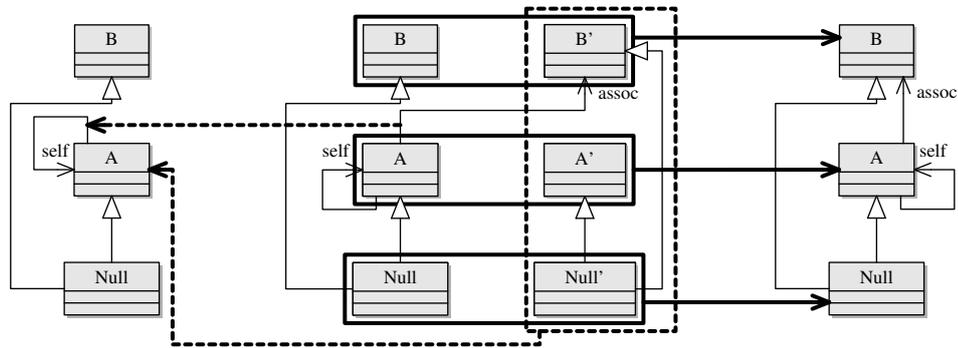


Abb. 12.12: Erzeugen einer Assoziation zu einer Klasse

Beschreibung: Eine neue Assoziation zu einer Klasse wird erzeugt (Abb. 12.12).

Typ: Refactoring

Motivation: Eine neue Assoziation zu einer Klasse erlaubt es, Verbindungen zwischen Objekten aufzubauen und somit Daten zu strukturieren und Aufgaben zu verteilen.

Voraussetzungen: —

Migration: Jedes Partikel, dessen Klasse die Quellklasse der neuen Assoziation darstellt, erhält eine Verknüpfung, die in der neuen Assoziation typisiert ist und auf ein *Null*-Objekt verweist.

Umsetzung: Auf der linken Seite wird aus der Quellklasse der zu erzeugenden Assoziation die komplette Klassenhierarchie als Kopie entfaltet sowie eine neue Assoziation zwischen ihrer Quellklasse und dem Duplikat ihrer Zielklasse aus der *self*-Assoziation der Quellklasse entfaltet. Auf der rechten Seite werden die nun doppelt vorhandenen Klassen zusammengelegt.

12.3.2.2 Erzeugen einer Assoziation zu einer Operation

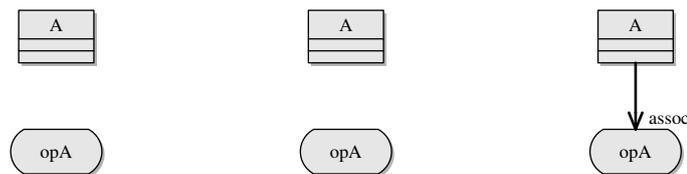


Abb. 12.13: Erzeugen einer Assoziation zu einer Operation

Beschreibung: Eine neue Assoziation zu einer Operation wird erzeugt (Abb. 12.13).

Typ: Refactoring

Motivation: Eine neue Assoziation zu einer Operation erlaubt es, Verbindungen zwischen Objekten und Nachrichten aufzubauen.

Voraussetzungen: —

Migration: Es ist keine Anpassung notwendig.

Umsetzung: Die linke Seite stellt die Identität dar. Auf der rechten Seite wird die neue Assoziation hinzugefügt.

12.3.2.3 Verschieben des Anfangs einer Assoziation zu einer Oberklasse

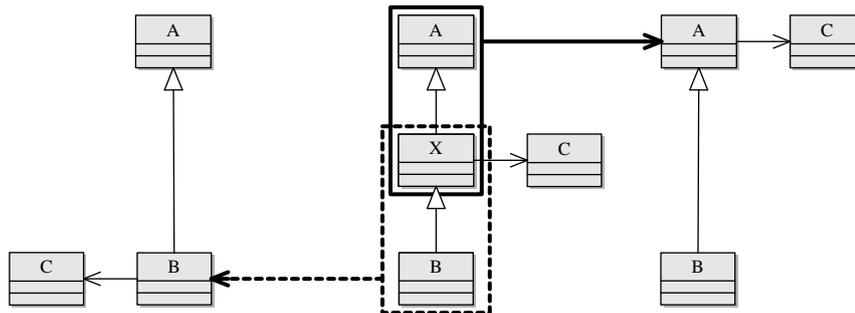


Abb. 12.14: Verschieben des Anfangs einer Assoziation zu einer Oberklasse

Beschreibung: Der Anfang einer Assoziation $Q \longrightarrow V$ wird entlang einer (direkten) Vererbungsbeziehung $Q \longrightarrow Z$ verschoben (Abb. 12.14).

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, Informationen einer Klasse eine Abstraktionsstufe höher anzusiedeln. Dadurch werden die über die Assoziation erreichbaren Informationen und Dienste für Objekte anderer (Geschwister-)Klassen nutzbar gemacht.

Voraussetzungen: —

Migration: Der Anfang jeder Verknüpfung zu der geänderten Assoziation wird entlang der Vererbungsverknüpfung zu dem Partikel der ausgewählten Oberklasse verschoben.

Umsetzung: Auf der linken Seite wird aus der Unterklasse eine temporäre Oberklasse entfaltet, so dass diese sich zwischen den beiden ursprünglichen Klassen befindet. Dabei wird der Anfang der Assoziation in der Mitte bereits auf diese temporäre Klasse gesetzt. Auf der rechten Seite wird die temporäre Klasse mit der ursprünglichen Oberklasse zusammengelegt.

12.3.2.4 Verschieben des Endes einer Assoziation zu einer Oberklasse

Beschreibung: Das Ende einer Assoziation $V \longrightarrow Q$ wird entlang einer (direkten) Vererbungsbeziehung $Q \longrightarrow Z$ verschoben (Abb. 12.15).

Typ: Entwicklung

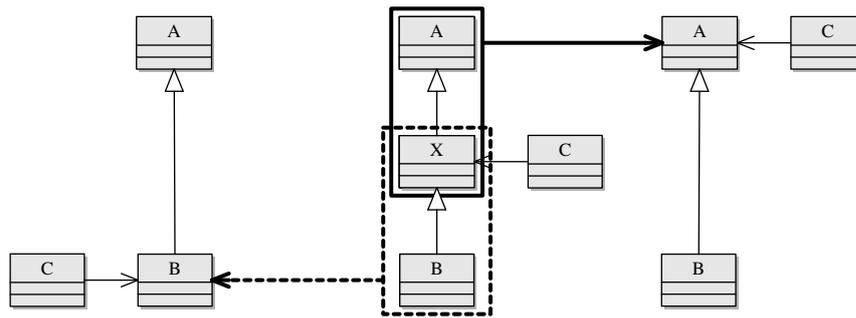


Abb. 12.15: Verschieben des Endes einer Assoziation zu einer Oberklasse

Motivation: Diese Transformation erlaubt es einem Objekt, durch die Wahl einer generelleren Zielklasse der Assoziation mit mehr Objekten in Beziehung zu treten. Dies geschieht auf Kosten der verfügbaren Informationen und Dienste, die über diese Beziehung ermittelt, verändert bzw. in Anspruch genommen werden können.

Voraussetzungen: —

Migration: Das Ende jeder Verknüpfung zu der geänderten Assoziation wird entlang der Vererbungsverknüpfung zu dem Partikel der ausgewählten Oberklasse verschoben.

Umsetzung: Auf der linken Seite wird aus der Unterklasse eine temporäre Oberklasse entfaltet, so dass diese sich zwischen den beiden ursprünglichen Klassen befindet. Dabei wird das Ende der Assoziation in der Mitte bereits auf diese temporäre Klasse gesetzt. Auf der rechten Seite wird die temporäre Klasse mit der ursprünglichen Oberklasse zusammengesetzt.

12.3.2.5 Löschen einer Assoziation

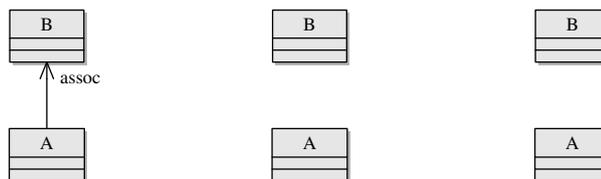


Abb. 12.16: Löschen einer Assoziation

Beschreibung: Eine bestehende Assoziation wird gelöscht (Abb. 12.16).

Typ: Entwicklung

Motivation: Das Löschen einer Assoziation ermöglicht das Entfernen nicht benötigter Informationen.

Voraussetzungen: —

Migration: Alle in der gelöschten Assoziation typisierten Verknüpfungen werden gelöscht.

Umsetzung: Auf der linken Seite wird die betroffene Assoziation gelöscht. Die rechte Seite stellt die Identität dar.

12.3.3 Attribute

In diesem Abschnitt sind Schema-Transformationen zu finden, die sich auf Attribute beziehen. Wie bei Assoziationen werden Transformationen zum Erzeugen und Löschen von Attributen sowie zum Verschieben eines Attributs entlang einer Vererbungslinie vorgestellt.

12.3.3.1 Erzeugen eines Attributs

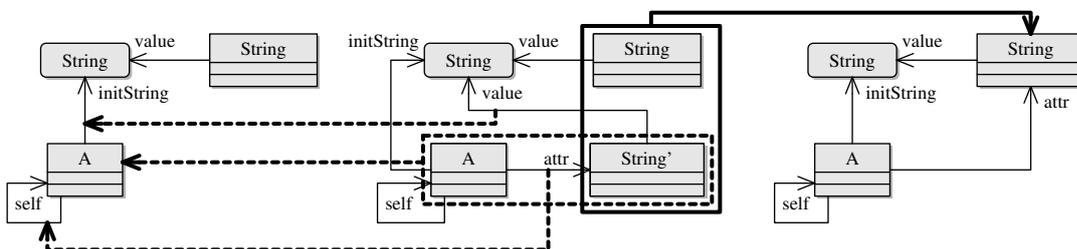


Abb. 12.17: Erzeugen eines Attributs

Beschreibung: Ein neues Attribut wird erzeugt (Abb. 12.17).

Typ: Refactoring

Motivation: Ein neues Attribut erlaubt es, Objekte mit zusätzlichen Daten anzureichern.

Voraussetzungen: —

Migration: Jedes Partikel, dessen Klasse die Quellklasse des neuen Attributs darstellt, erhält einen Attributwert, der in dem neuen Attribut typisiert ist und der mit einem Initialwert gemäß Abschnitt 5.1.5 belegt ist.

Umsetzung: Auf der linken Seite werden aus der Quellklasse des zu erzeugenden Attributs die Ziel-Hüllenklasse, aus der *self*-Assoziation der Quellklasse das zu erzeugende Attribut sowie aus der *initT*-Assoziation der Quellklasse die *value*-Assoziation zwischen der neuen Hüllenklasse und dem primitiven Datentyp *T* entfaltet. Auf der rechten Seite werden die alte und die neue Hüllenklasse sowie die *value*-Assoziationen zusammengelegt.¹¹⁶

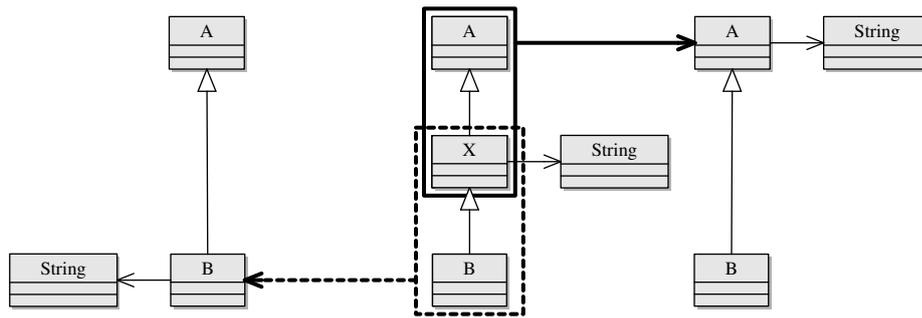


Abb. 12.18: Verschieben eines Attributs zu einer Oberklasse

12.3.3.2 Verschieben eines Attributs zu einer Oberklasse

Beschreibung: Ein Attribut $Q \longrightarrow Type$ wird entlang einer (direkten) Vererbungsbeziehung $Q \longrightarrow Z$ verschoben (Abb. 12.18). Diese Transformation ist strukturgleich zu der Transformation “Verschieben des Anfangs einer Assoziation zu einer Oberklasse” (12.3.2.3).

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, Informationen einer Klasse eine Abstraktionsstufe höher anzusiedeln. Dadurch werden die über das Attribut erreichbaren Informationen und Dienste für Objekte anderer (Geschwister-)Klassen nutzbar gemacht.

Voraussetzungen: —

Migration: Jeder Attributwert zu dem geänderten Attribut wird entlang der Vererbungsverknüpfung zu dem Partikel der ausgewählten Oberklasse verschoben.

Umsetzung: Auf der linken Seite wird aus der Unterklasse eine temporäre Oberklasse entfaltet, so dass diese sich zwischen den beiden ursprünglichen Klassen befindet. Dabei wird das Attribut in der Mitte bereits dieser temporären Klasse zugeordnet. Auf der rechten Seite wird die temporäre Klasse mit der ursprünglichen Oberklasse zusammengelegt.

12.3.3.3 Löschen eines Attributs

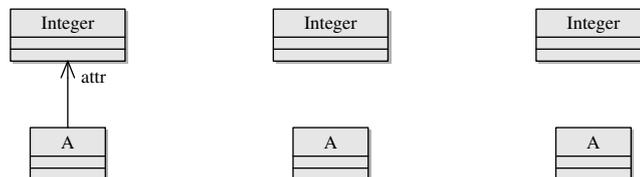


Abb. 12.19: Löschen eines Attributs

¹¹⁶ Dies verletzt die Definition einer konsistenten Transformation (Def. 11.14) nicht, weil die *value*- und *value'*-Assoziationen eigentlich Attribut-Kanten sind und somit zusammengelegt werden dürfen.

Beschreibung: Ein bestehendes Attribut wird gelöscht (Abb. 12.19). Diese Transformation ist strukturgleich zu der Transformation “Löschen einer Assoziation” (12.3.2.5).

Typ: Entwicklung

Motivation: Das Löschen eines Attributs ermöglicht das Entfernen nicht benötigter Informationen.

Voraussetzungen: —

Migration: Alle in dem gelöschten Attribut typisierten Attributwerte werden gelöscht.

Umsetzung: Auf der linken Seite wird das betroffene Attribut gelöscht. Die rechte Seite stellt die Identität dar.

12.3.4 Operationen

Dieser Abschnitt beschreibt Schema-Transformationen, die auf Operationen arbeiten. Neben dem Erzeugen neuer und Löschen alter Operationen werden Transformationen vorgestellt, die das Zusammenfassen von Operationen sowie die Veränderung der Operationsstruktur ermöglichen.

12.3.4.1 Erzeugen einer eigenständigen Operation



Abb. 12.20: Erzeugen einer eigenständigen Operation

Beschreibung: Eine neue Operation wird erzeugt (Abb. 12.20).

Typ: Entwicklung

Motivation: Neue Operationen erlauben das Anbieten neuer Dienste und somit die Erweiterung der Software um neue Funktionalität.

Voraussetzungen: —

Migration: Es ist keine unmittelbare Anpassung notwendig. Sobald der Dienst genutzt werden soll, muss jedoch eine entsprechende Methode zu dieser Operation implementiert werden.

Umsetzung: Die linke Seite stellt die Identität dar. Auf der rechten Seite wird die neue Operation hinzugefügt.

Anmerkung: Diese Transformation ist lediglich der Vollständigkeit und Symmetrie halber aufgeführt. Wegen der Anforderung des konzeptionellen Modells, dass alle Operationen direkte oder indirekte Unteroperationen von *OpBase* sein müssen, sind neue Operationen nie eigenständig und werden immer über die Transformationen “Erzeugen einer Oberoperation” (12.3.4.2) und “Erzeugen einer Unteroperation” (12.3.4.3) eingeführt.

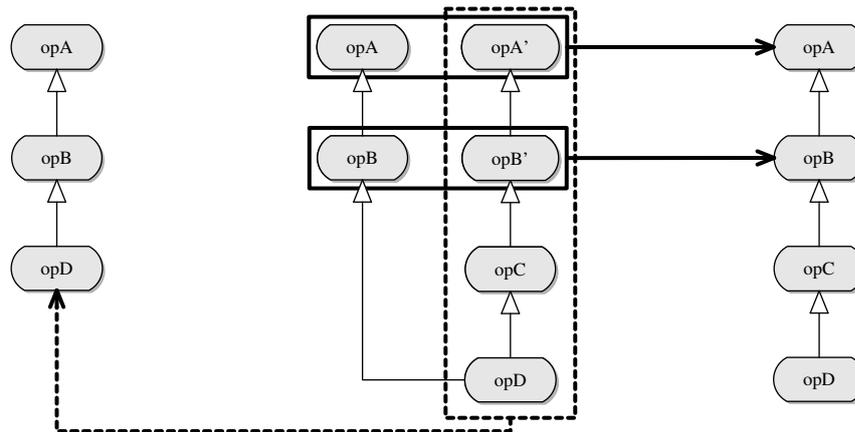


Abb. 12.21: Erzeugen einer Oberoperation

12.3.4.2 Erzeugen einer Oberoperation

Beschreibung: Eine neue Operation generalisiert eine nichtleere Menge bestehender Operationen (Abb. 12.21). Diese Operation wird eine direkte Oberoperation der zu generalisierenden Operationen. Weitere Vererbungsbeziehungen zu anderen Operationen werden nicht angelegt. Diese Transformation ist strukturgleich zu der Transformation “Erzeugen einer Oberklasse” (12.3.1.2).

Typ: Refactoring

Motivation: Die Generalisierung von Operationen erlaubt es, die Gemeinsamkeit in Hinblick auf Informationsgehalt und Verhalten einer Menge bestehender Operationen im Schema auszudrücken. In einem zweiten Schritt können Parameter und Operationen der Unteroperationen nach Bedarf zu der generelleren Operationen verschoben werden.

Voraussetzungen: —

Migration: Jede Nachricht zu einer der generalisierten Operationen erhält ein zusätzliches Partikel zur neuen Oberoperation, das über eine Vererbungsverknüpfung mit dem Partikel der generalisierten Operation verbunden ist.

Umsetzung: Auf der linken Seite werden die Oberoperationen aus der Unteroperation entfaltet. Werden die neuen Oberoperationen zwischen zwei bestehenden Operationen eingefügt, dann werden zusätzlich alle alten direkten und indirekten Oberoperationen entfaltet. Auf der rechten Seite werden die bereits existierenden Oberoperationen mit den duplizierten Oberoperationen zusammengelegt, so dass nur die neuen Oberoperationen als neue Schema-Elemente verbleiben.

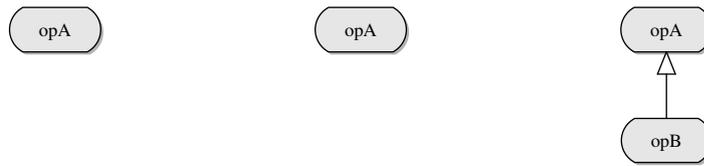


Abb. 12.22: Erzeugen einer Unteroperation

12.3.4.3 Erzeugen einer Unteroperation

Beschreibung: Eine neue Operation spezialisiert eine nichtleere Menge bestehender Operationen (Abb. 12.22). Diese Operation wird eine direkte Unteroperation der zu spezialisierenden Operationen. Weitere Vererbungsbeziehungen zu anderen Operationen werden nicht angelegt.

Typ: Refactoring

Motivation: Die Spezialisierung von Operationen erlaubt es, zusätzliche Informationen und zusätzliches Verhalten für neue Nachrichten zu diesen Operationen hinzuzufügen.

Voraussetzungen: —

Migration: Es ist keine unmittelbare Anpassung notwendig. Sobald der Dienst genutzt werden soll, muss jedoch eine entsprechende Methode zu dieser Operation implementiert werden.

Umsetzung: Die linke Seite stellt die Identität dar. Auf der rechten Seite wird die Unteroperation mitsamt den Vererbungsbeziehungen hinzugefügt.

12.3.4.4 Verschmelzen zweier verwandter Operationen

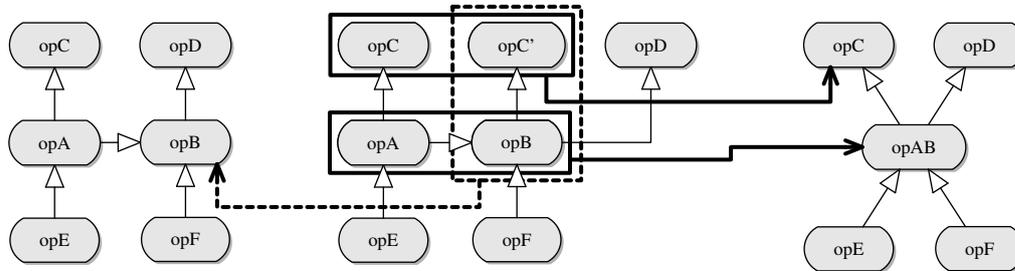


Abb. 12.23: Verschmelzen zweier verwandter Operationen

Beschreibung: Zwei Operationen $opA \longrightarrow opB$, die miteinander über eine (direkte) Vererbungsbeziehung verbunden sind, werden zu einer Operation $opAB$ zusammengelegt (Abb. 12.23). Die Oberoperationen von $opAB$ sind die Oberoperationen von opA (opB ausgenommen) und die Oberoperationen von opB . Die Unteroperationen von $opAB$ sind die Unteroperationen von opA und die Unteroperationen von opB (opA ausgenommen). Alle Assoziationen und Parameter, die mit den Operationen opA oder opB verbunden sind, werden mit der Operation $opAB$ verbunden.

Diese Transformation ist strukturgleich zu der Transformation “Verschmelzen zweier verwandter Klassen” (12.3.1.4).

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, Abstraktionen, die sich als zu fein herausgestellt haben, wieder zu vereinen.

Voraussetzungen: —

Migration: Miteinander über eine Vererbungsverknüpfung verbundene Partikel zu den beiden Operationen opA und opB werden zusammengelegt. Andere opA - und opB -Partikel werden zu $opAB$ -Partikeln umtypisiert. Nachrichten zur Operation opB , die nicht auch Nachrichten zur Operation opA sind, erhalten für jede Operation in der echten Oberoperationshierarchie von opA (die Oberoperationshierarchie von opB ausgenommen) ein entsprechendes neues Partikel.

Umsetzung: Auf der linken Seite werden alle Oberoperationen der Unteroperation aus der Oberoperation entfalteter. Auf der rechten Seite werden zum einen die Unter- und Oberoperation und zum anderen die nun doppelt vorhandenen Oberoperationen zusammengelegt.

12.3.4.5 Verschmelzen zweier nicht verwandter Operationen

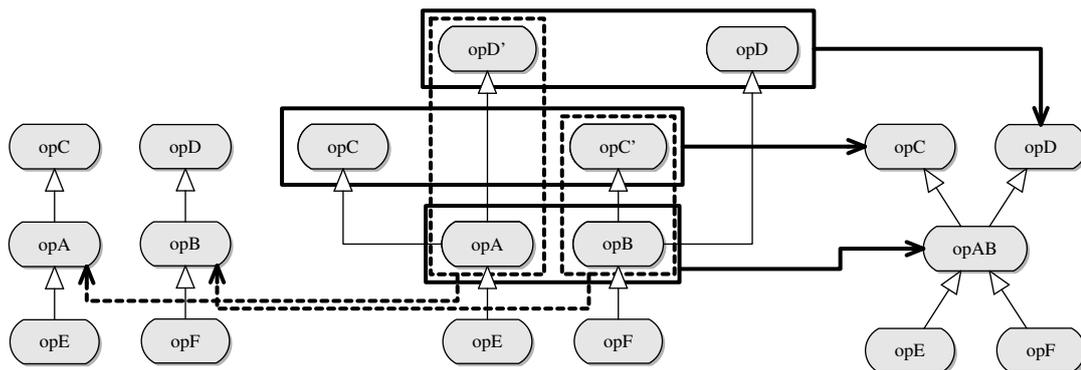


Abb. 12.24: Verschmelzen zweier nicht verwandter Operationen

Beschreibung: Zwei Operationen opA und opB , die nicht miteinander über eine Vererbungsbeziehung verbunden sind, werden zu einer Operation $opAB$ zusammengelegt (Abb. 12.24). Die Oberoperationen von $opAB$ sind die Oberoperationen von opA und die Oberoperationen von opB . Die Unteroperationen von $opAB$ sind die Unteroperationen von opA und die Unteroperationen von opB . Alle Assoziationen und Parameter, die mit den Operationen opA oder opB verbunden sind, werden mit der Operation $opAB$ verbunden. Diese Transformation ist strukturgleich zu der Transformation “Verschmelzen zweier nicht verwandter Klassen” (12.3.1.5).

Typ: Refactoring

Motivation: Diese Transformation ist nützlich, wenn Schemata unterschiedlicher Herkunft aber mit gleicher Semantik vereinheitlicht werden sollen.

Voraussetzungen: —

Migration: opA - und opB -Partikel werden zu $opAB$ -Partikeln umtypisiert. Nachrichten zur Operation opB , die nicht auch Nachrichten zur Operation opA sind, erhalten für jede Operation in der echten Oberoperationshierarchie von opA ein entsprechendes neues Partikel. Nachrichten zur Operation opA , die nicht auch Nachrichten zur Operation opB sind, erhalten für jede Operation in der echten Oberoperationshierarchie von opB ein entsprechendes neues Partikel.

Umsetzung: Auf der linken Seite werden alle Oberoperationen der zusammenzulegenden Operationen aus der jeweils gegensätzlichen Operation entfaltet, so dass in der Mitte beide Operationen eine identische Oberoperationshierarchie besitzen. Auf der rechten Seite werden zum einen die beiden zusammenzulegenden Operationen und zum anderen die nun doppelt vorhandenen Oberoperationen zusammengelegt.

12.3.4.6 Verschieben einer Oberoperation entlang einer Vererbungsbeziehung

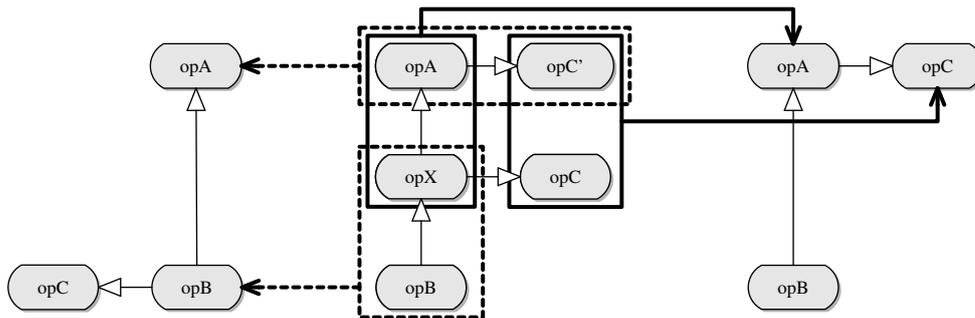


Abb. 12.25: Verschieben einer Oberoperation entlang einer Vererbungsbeziehung

Beschreibung: Eine direkte Oberoperation opV einer bestehenden Operation opQ wird indirekte Oberoperation, indem sie entlang einer (direkten) Vererbungsbeziehung $opQ \longrightarrow opZ$ verschoben wird (Abb. 12.25). Diese Transformation ist strukturgleich zu der Transformation “Verschieben einer Oberklasse entlang einer Vererbungsbeziehung” (12.3.1.6).

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, vorhandene Generalisierungen zu verallgemeinern und somit Abstraktionen in einer Operationshierarchie hinzuzufügen.

Voraussetzungen: —

Migration: In jedem opQ -Objekt wird das opV -Partikel vom opQ -Partikel gelöst und mit dem opZ -Partikel verbunden. Jedes opZ -Objekt erhält ein neues Partikel zu jeder Operation in der Oberoperationshierarchie von opV .

Umsetzung: Auf der linken Seite wird aus der Quelloperation eine temporäre Operoperation entfaltet, so dass diese sich zwischen Quell- und Zieloperation befindet. Dabei wird die zu verschiebenden Operoperation Operoperation der temporären Operation. Auf der rechten Seite wird die temporäre Operation mit der Zieloperation zusammengelegt.

12.3.4.7 Verschieben einer Unteroperation entlang einer Vererbungsbeziehung

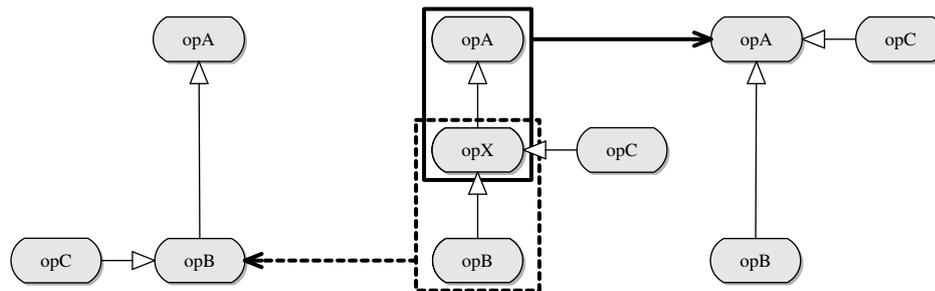


Abb. 12.26: Verschieben einer Unteroperation entlang einer Vererbungsbeziehung

Beschreibung: Eine indirekte Unteroperation opV einer bestehenden Operation opZ wird direkte Unteroperation, indem sie entlang einer (direkten) Vererbungsbeziehung $opQ \longrightarrow opZ$ verschoben wird (Abb. 12.26). Diese Transformation ist strukturgleich zu der Transformation “Verschieben einer Unterklasse entlang einer Vererbungsbeziehung” (12.3.1.7).

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, vorhandene Spezialisierungen zu verallgemeinern und somit Abstraktionen aus einer Operationshierarchie zu entfernen.

Voraussetzungen: —

Migration: In jeder opV -Nachricht wird das opV -Partikel vom opQ -Partikel gelöst und mit dem opZ -Partikel verbunden.

Umsetzung: Auf der linken Seite wird aus der Quelloperation eine temporäre Oberoperation entfaltet, so dass diese sich zwischen Quell- und Zieloperation befindet. Dabei wird die zu verschiebende Unteroperation Unteroperation der temporären Operation. Auf der rechten Seite wird die temporäre Operation mit der Zieloperation zusammengelegt.

Anmerkung: Jede opV -Nachricht besitzt nach der Transformation weiterhin das entsprechende opQ -Partikel und ist somit mehrfach klassifiziert, sofern kein Partikel zu einer gemeinsamen Unteroperation von opV und opQ existiert.

12.3.4.8 Löschen einer Operation

Beschreibung: Eine bestehende Operation wird einschließlich aller Parameter gelöscht (Abb. 12.27). Diese Transformation ist strukturgleich zu der Transformation “Löschen einer Klasse” (12.3.1.8).

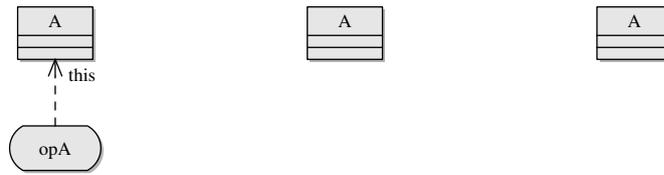


Abb. 12.27: Löschen einer Operation

Typ: Entwicklung

Motivation: Das Löschen bestehender Operationen erlaubt das Entfernen nicht mehr benötigter Dienste aus der Software.

Voraussetzungen: —

Migration: Alle Nachrichten und Methoden zu der Operation werden gelöscht.

Umsetzung: Auf der linken Seite wird die betroffene Operation gelöscht. Die rechte Seite stellt die Identität dar.

Anmerkung: Das Löschen einer Operation löscht nicht die implementierenden Methoden. Das liegt daran, dass Methoden als Aktionen umgesetzt sind, die aus ganzen Systemen bestehen, welche durch die Migration nicht komplett entfernt werden können (vgl. Kapitel 10). Die in Abschnitt 12.2.2 formulierte Forderung, dass eine Methode nur dann zur Anwendung kommen kann, wenn ihre linke Seite einen zusammenhängenden Graphen bildet, stellt jedoch sicher, dass das Löschen einer Operation alle zugehörigen Methoden außer Kraft setzt.¹¹⁷

12.3.5 Parameter

In diesem Abschnitt werden Schema-Transformationen beschrieben, die auf Parametern arbeiten. Dies beinhaltet das Erzeugen, Verschieben und Löschen von Parametern.

12.3.5.1 Erzeugen eines Parameters

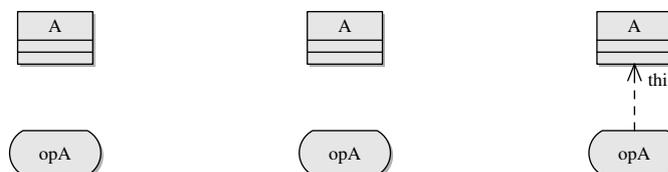


Abb. 12.28: Erzeugen eines Parameters

¹¹⁷ Diese einzige Ausnahme sind Operationen ohne jegliche Parameter. Solche Operationen sind jedoch nicht nützlich, denn sie können den Programmzustand lediglich durch eine Umsetzung des *next*-Parameters verändern.

Beschreibung: Ein neuer Parameter zu einer Operation wird erzeugt, indem die Operation mit einer Klasse oder einer anderen Operation verbunden wird (Abb. 12.28 zeigt dies beispielhaft für einen Daten-Parameter). Diese Transformation ist strukturgleich zu der Transformation “Erzeugen einer Assoziation zu einer Operation” (12.3.2.2).

Typ: Entwicklung

Motivation: Das Hinzufügen eines neuen Parameters erlaubt es, dem durch die Operation spezifizierten Dienst weitere Informationen zur Verrichtung seiner Aufgabe zukommen zu lassen.

Voraussetzungen: —

Migration: Es ist keine unmittelbare Anpassung notwendig. Sobald der Parameter in einer Methode zu der Operation genutzt werden soll, müssen jedoch alle Nachrichten entsprechend angepasst werden. Diese Anpassung kann im vorgestellten Modell zur Zeit nur manuell ohne Unterstützung durch eine induzierte Migration erfolgen.

Umsetzung: Die linke Seite stellt die Identität dar. Auf der rechten Seite wird der neue Parameter hinzugefügt.

12.3.5.2 Verschieben des Anfangs eines Parameters zu einer Oberoperation

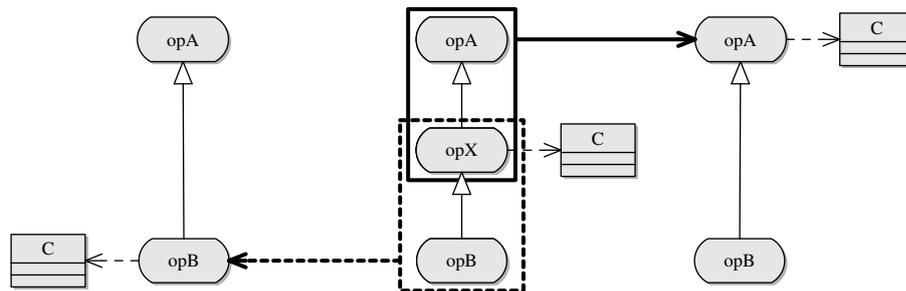


Abb. 12.29: Verschieben des Anfangs eines Daten-Parameters zu einer Oberoperation

Beschreibung: Der Anfang eines (Daten- oder Operations-)Parameters wird entlang einer (direkten) Vererbungsbeziehung zu einer Oberoperation der Quelloperation des Parameters verschoben (Abb. 12.29 zeigt dies beispielhaft für einen Daten-Parameter). Diese Transformation ist strukturgleich zu der Transformation “Verschieben des Anfangs einer Assoziation zu einer Oberklasse” (12.3.2.3).

Typ: Refactoring

Motivation: Diese Transformation erlaubt es, Parameter eine Abstraktionsstufe höher anzusiedeln. Dies ist zur Vermeidung von Redundanz nützlich, wenn innerhalb der Operationshierarchie ein Parameter von mehreren Operationen geteilt werden soll.

Voraussetzungen: —

Migration: Der Anfang jedes Arguments zu dem geänderten Parameter wird entlang der Vererbungsverknüpfung zu dem Partikel der ausgewählten Oberoperation verschoben.

Umsetzung: Auf der linken Seite wird aus der Ausgangsoperation eine temporäre Oberoperation entfaltet, so dass diese sich zwischen den beiden ursprünglichen Operation befindet. Dabei wird der Anfang des Parameters in der Mitte bereits auf diese temporäre Operation gesetzt. Auf der rechten Seite wird die temporäre Operation mit der ursprünglichen Oberoperation zusammengelegt.

12.3.5.3 Verschieben des Endes eines Parameters zu einer Oberoperation

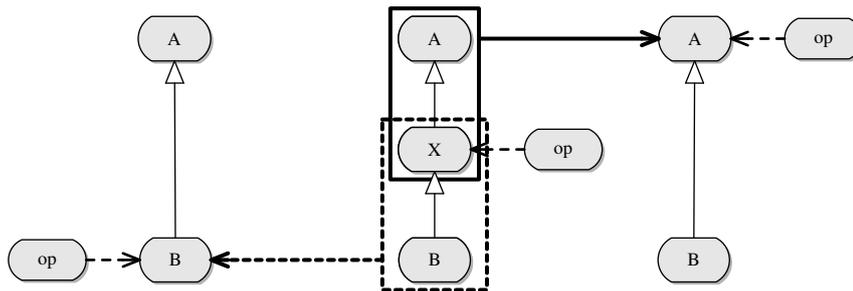


Abb. 12.30: Verschieben des Endes eines Parameters zu einer Oberoperation

Beschreibung: Das Ende eines Parameters wird entlang einer (direkten) Vererbungsbeziehung zu einer Oberoperation der Zieleroperation des Parameters verschoben (Abb. 12.30). Diese Transformation ist strukturgleich zu der Transformation “Verschieben des Endes einer Assoziation zu einer Oberklasse” (12.3.2.4).

Typ: Entwicklung

Motivation: Diese Transformation erlaubt es, Parameter einer Operation zu verallgemeinern und somit die Erweiterung der Funktionalität bestehender Methoden vorzubereiten, indem deren Argumentbereich vergrößert wird. Dies geschieht auf Kosten der verfügbaren Informationen und Dienste, die über diese Parameter ermittelt, verändert bzw. in Anspruch genommen werden können.

Voraussetzungen: —

Migration: Das Ende jedes Arguments zu dem geänderten Parameter wird entlang der Vererbungsverknüpfung zu dem Partikel der ausgewählten Oberoperation verschoben.

Umsetzung: Auf der linken Seite wird aus der Unteroperation eine temporäre Oberoperation entfaltet, so dass diese sich zwischen den beiden ursprünglichen Operationen befindet. Dabei wird das Ende des Parameters in der Mitte bereits auf diese temporäre Operation gesetzt. Auf der rechten Seite wird die temporäre Operation mit der ursprünglichen Oberoperation zusammengelegt.

12.3.5.4 Löschen eines Parameters

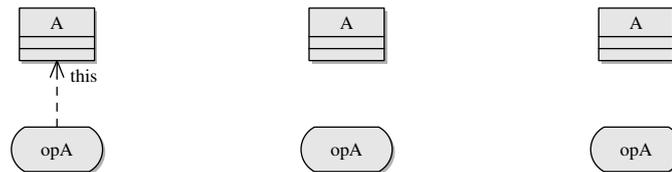


Abb. 12.31: Löschen eines Parameters

Beschreibung: Eine bestehende Verbindung zwischen einer Operation und einer Klasse oder einer anderen Operation wird gelöscht (Abb. 12.31 zeigt dies beispielhaft für einen Daten-Parameter). Diese Transformation ist strukturgleich zu der Transformation “Löschen einer Assoziation” (12.3.2.5).

Typ: Entwicklung

Motivation: Das Löschen eines bestehenden Parameters erlaubt es, für den durch die Operation spezifizierten Dienst überflüssig gewordene Informationen zu entfernen.

Voraussetzungen: —

Migration: Jedes Argument, das in dem entfernten Parameter typisiert ist, wird gelöscht.

Umsetzung: Auf der linken Seite wird der betroffene Parameter gelöscht. Die rechte Seite stellt die Identität dar.

Fallstudie: Formalisierung

In diesem Kapitel wird die Fallstudie aus Kapitel 6 Schritt für Schritt ins mathematische Modell übertragen. Die Demonstration der Auswirkungen induzierter Migrationen konzentriert sich auf Platzgründen dabei auf typisierte Daten und Programme und lässt Prozesse außer Acht. Weil Programme jedoch genau wie Prozesse mit Hilfe von Objekten, Verknüpfungen, Nachrichten und Argumenten kodiert werden, zeigen die Beispiele der Migration von Programmen auch immer die Veränderungen entsprechender Prozesse.

13.1 Ausgangssituation

Die Ausgangssituation der Fallstudie aus Abschnitt 6.1 kann fast ohne Veränderungen ins mathematische Modell übertragen werden. Jede Klasse und jedes Objekt wird durch ein Element der Trägermenge zur Sorte C bzw. O dargestellt. Jede Assoziation und jede Verknüpfung, die nicht auf einen primitiven Datentyp bzw. Wert verweist, wird durch ein Element der Trägermenge zur Sorte A bzw. L dargestellt, wobei Quelle und Ziel der Assoziation bzw. Verknüpfung über die Operationen $sourceA$ und $targetA$ bzw. $sourceL$ und $targetL$ festgelegt werden. Jede Assoziation und jede Verknüpfung zu einem primitiven Datentyp t bzw. einem entsprechend typisierten Wert wird über ein Element der Trägermenge zur Sorte CA_t bzw. OA_t dargestellt, wobei Quelle und Ziel über die Operationen $sourceCA_t$ und $targetCA_t$ bzw. $sourceOA_t$ und $targetOA_t$ definiert werden. Jeder primitive Datentyp t entspricht einem eindeutigen Element der Trägermenge zur Sorte SV_t .¹¹⁸ Jeder Wert zu einem primitiven Datentyp t entspricht einem Element der Trägermenge zur Sorte IV_t . Operationen und Nachrichten entsprechen Elementen der Trägermenge zur Sorte C bzw. O , die das Prädikat *operation* bzw. *message* erfüllen.

Die Benennung von Objekt- und Nachrichten-Partikeln entspricht dem Schema $ID:Obj[Name]$, wobei ID ein eindeutiger Identifikator des Partikels, Obj der Name des zum Partikel gehörenden

¹¹⁸ Der Wert ist eindeutig, weil die Algebra, die im Schema zur Spezifikation des primitiven Datentyps t verwendet wird, die finale Algebra ist (vgl. Def. B.37).

Objekts bzw. der zum Partikel gehörenden Nachricht und *Name* der Name der zugehörigen Klasse bzw. der zugehörigen Operation ist. Die Benennung von Verknüpfungen und Argumenten hat den analogen Aufbau *ID[Name]*, wobei *Name* der Name der zugehörigen Assoziation bzw. des zugehörigen Parameters und *ID* der Identifikator der Verknüpfung bzw. des Arguments ist. Diese Benennung macht somit zum einen deutlich, wie die Typisierung der Elemente der Ausprägungsebene im Schema erfolgt. Zum anderen wird über die Objekt-Namen die *underO*-Relation verdeutlicht, denn alle Partikel, die über *underO* miteinander in Beziehung stehen, gehören demselben Objekt an.

In Abb. 13.1 und Abb. 13.2 werden das übertragene Klassenmodell sowie die Objekte ohne Attributwerte dargestellt; die Attributwerte werden exemplarisch für das Objekt *a1[Adresse]* in Abb. 13.3 gezeigt. Dabei werden Elemente der Trägermengen als Knoten und Operationen zwischen den Trägermengen als Kanten zwischen den entsprechenden Elementen dargestellt. Operationen und Nachrichten werden (wie im konzeptionellen Modell) als Knoten mit abgerundeten Ecken dargestellt. Aus Gründen der Übersichtlichkeit werden von den Relationen *underC* und *underO* nur die Erzeugendenrelationen dargestellt;¹¹⁹ die Notation entspricht derjenigen des konzeptionellen Modells. Weiter wird die Relation *relC* nicht abgebildet, weil sie sich in dem Beispiel vollständig aus der Relation *underC* ergibt. Schließlich werden die *selfC*- und *selfO*- sowie die *initCA_t*- und *initOA_t*-Kanten für alle primitiven Datentypen $t \in TYPES$ nicht dargestellt; dies wird in Abb. 13.4 exemplarisch für die Klasse *Adresse* und in Abb. 13.5 für das Objekt *a1[Adresse]* gezeigt.

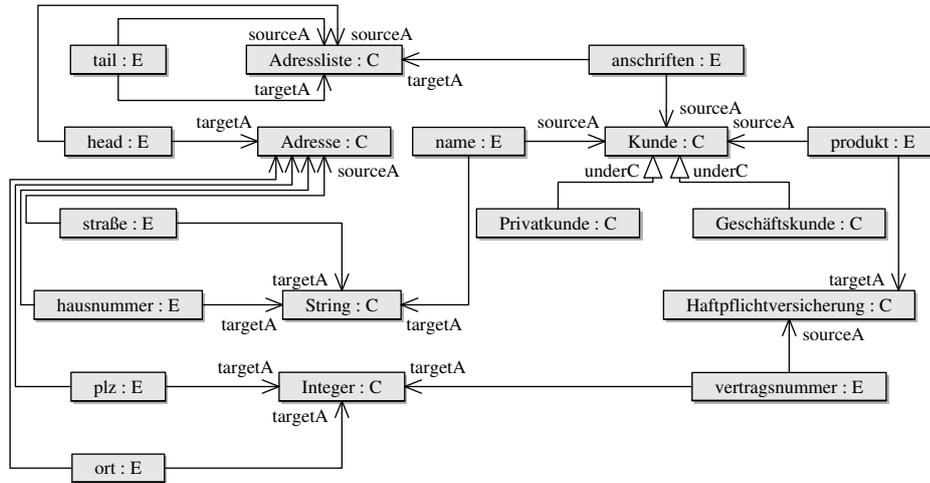


Abb. 13.1: Ausgangssituation: Schema

In Abb. 13.6 ist die Übertragung der Operation zur Selektion der *head*-Assoziation der Klasse *Adressliste* dargestellt; die zugehörige Methode ist in den Abbildungen 13.7 bis 13.9 gezeigt, wobei der Klebgraph (Abb. 13.8) sich aus dem Schnitt der linken und der rechten Seite der Methode

¹¹⁹ Es fehlen somit die reflexiven und transitiven Paare.

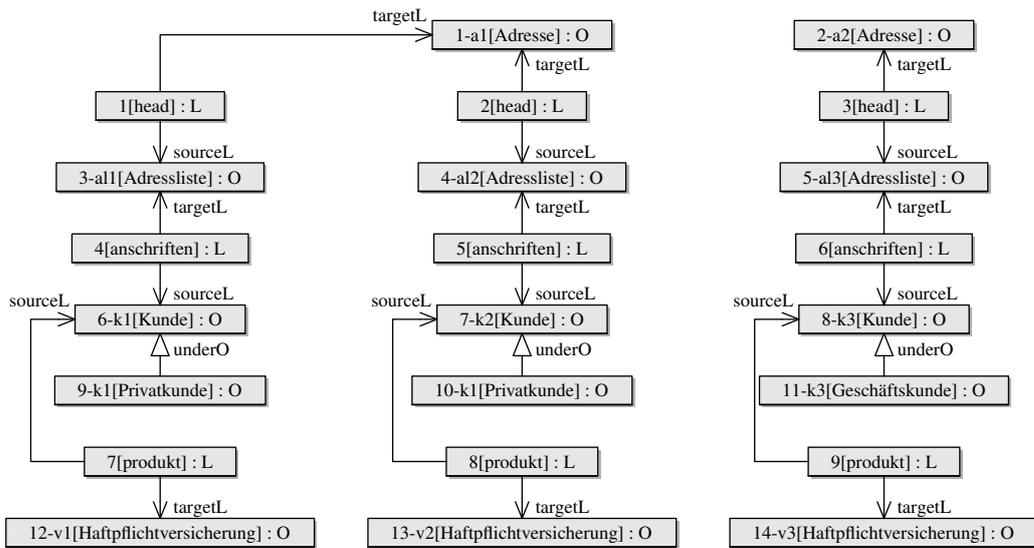


Abb. 13.2: Ausgangssituation: Objekte ohne Attributwerte

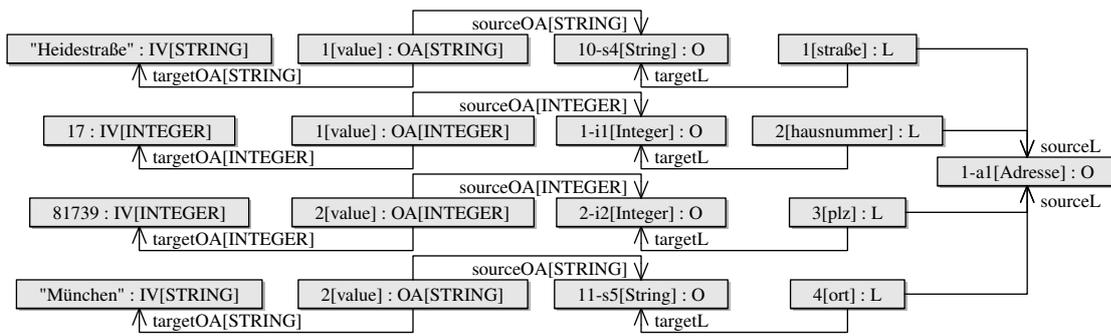


Abb. 13.3: Ausgangssituation: Attributwerte des Objekts a1[Adresse]

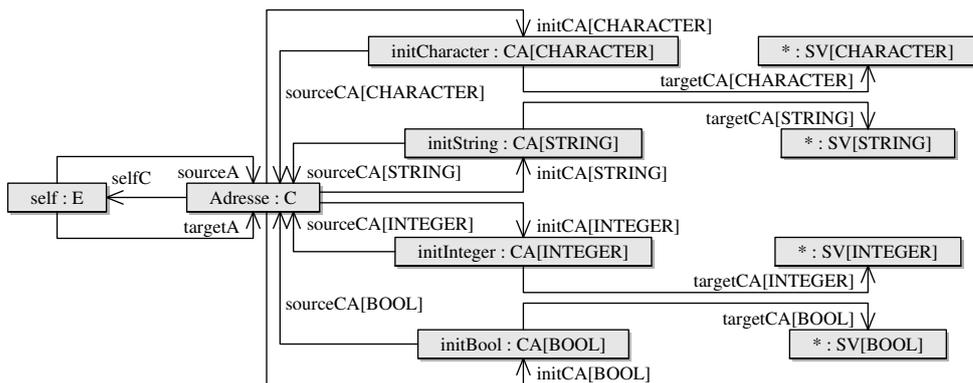


Abb. 13.4: Ausgangssituation: Zusätzliche Struktur von Klassen

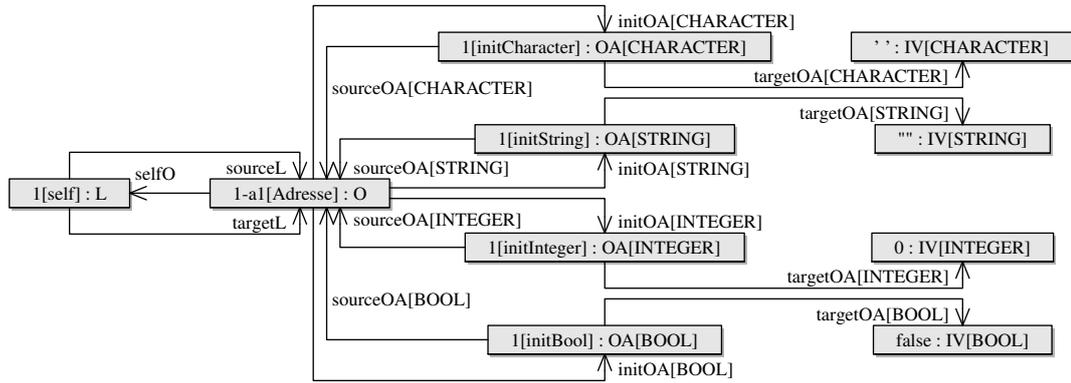


Abb. 13.5: Ausgangssituation: Zusätzliche Struktur von Objekten

ergibt. Die Morphismen der DPO-Regel sind die Einbettungen des mittleren Graphen in die linke bzw. rechte Seite und ergeben sich eindeutig aus der Benennung der Elemente.

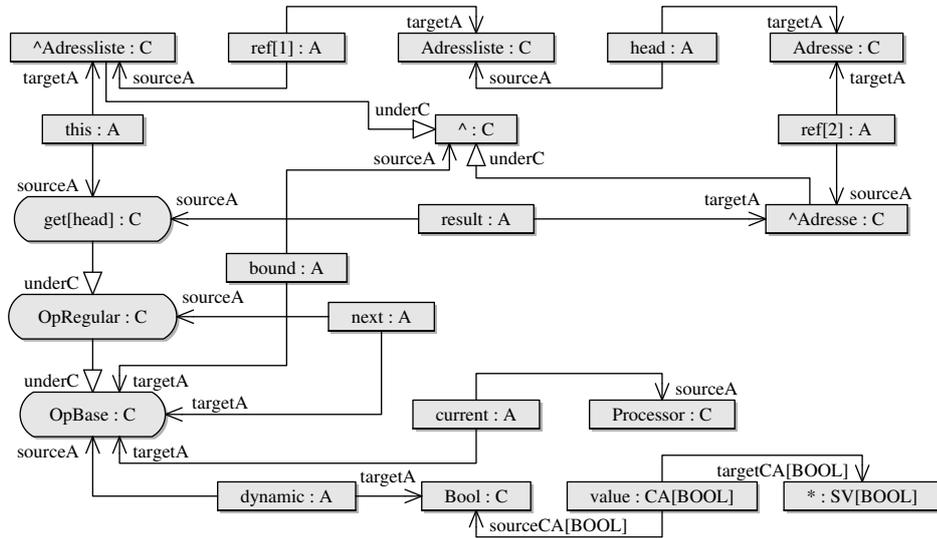


Abb. 13.6: Ausgangssituation: *get/head*-Operation

Schließlich ist in Abb. 13.10 die Übertragung der *has*-Operation der Klasse *Adressliste* zu sehen. Die zugehörige Methode ist in den Abbildungen 13.11 bis 13.13 dargestellt.¹²⁰

¹²⁰ Die bereits in Abb. 13.6 dargestellten Klassen, Operationen, Assoziationen und Parameter *Processor*, *OpBase*, *current*, *bound*, *dynamic* und *next* sind aus Gründen der Übersichtlichkeit in Abb. 13.10 weggelassen worden.

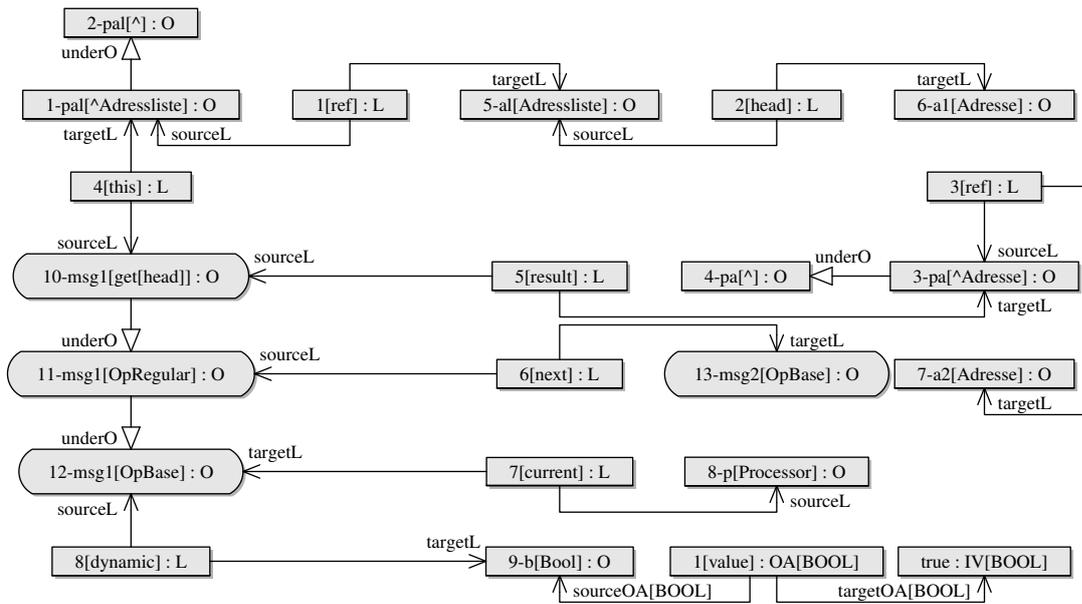


Abb. 13.7: Ausgangssituation: *get/head*-Methode (linke Seite)

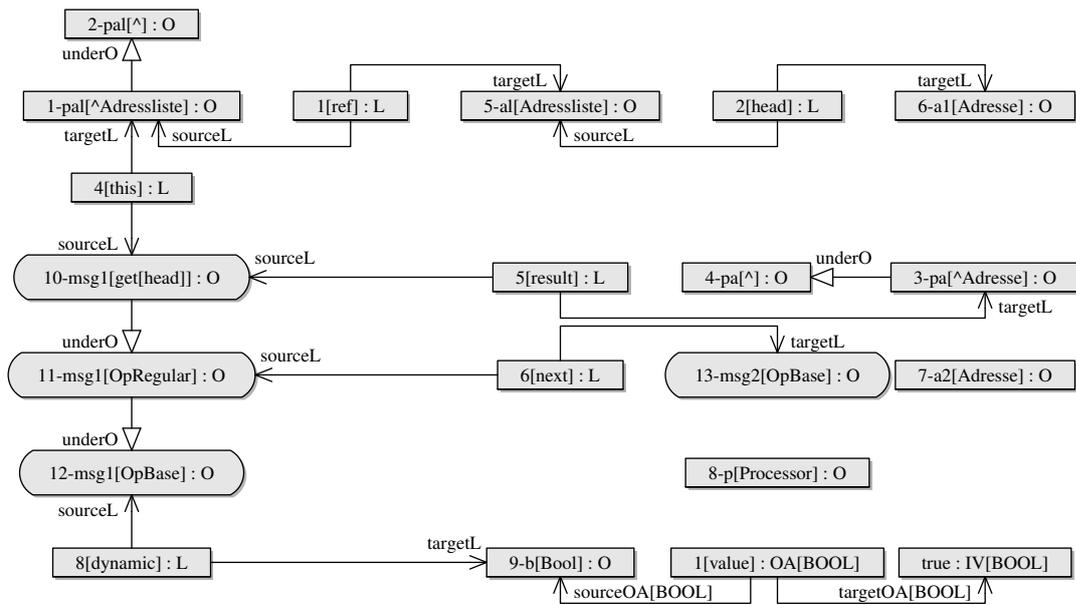


Abb. 13.8: Ausgangssituation: *get/head*-Methode (Schnittstelle)

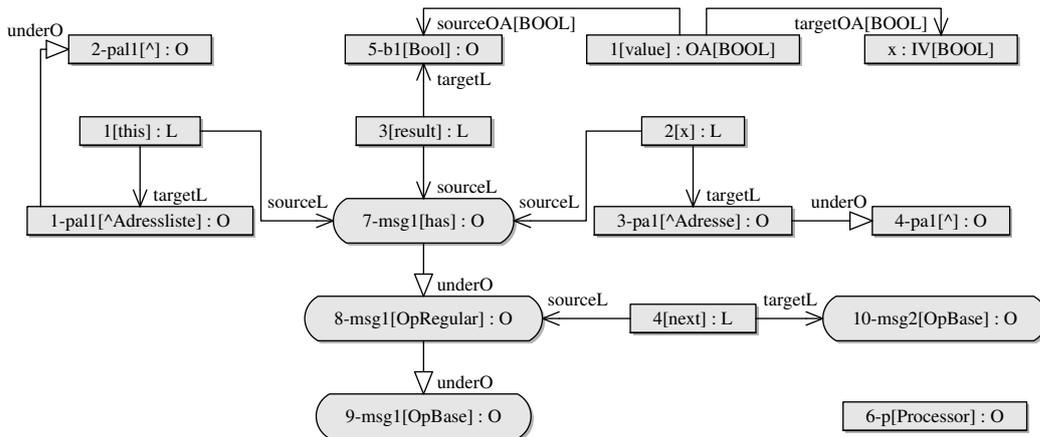


Abb. 13.12: Ausgangssituation: *has*-Methode (Schnittstelle)

13.2 Phase 1: Hinzunahme eines neuen Versicherungsprodukts

In dieser Phase werden Veränderungen am Versicherungsmodell vorgenommen. Die folgenden Schema-Änderungen aus Abschnitt 6.2 können über entsprechende Schema-Transformationen aus Abschnitt 12.3 formuliert werden:

- Erstellung der Klasse *Versicherung* als Oberklasse der Klasse *Haftpflichtversicherung* durch die Transformation “Erzeugen einer Oberklasse” (12.3.1.2)
- Erstellung der Klasse *Rechtsschutzversicherung* als Unterklasse der Klasse *Versicherung* durch die Transformation “Erzeugen einer Unterklasse” (12.3.1.3)
- Verschieben des Attributs *vertragsnummer* in die Klasse *Versicherung* durch die Transformation “Verschieben eines Attributs zu einer Oberklasse” (12.3.3.2)
- Verschieben des Ziels der von der Klasse *Kunde* ausgehenden Assoziation *produkt* zur Klasse *Versicherung* durch die Transformation “Verschieben des Endes einer Assoziation zu einer Oberklasse” (12.3.2.4)

Im Folgenden wird der letzte Punkt im Detail betrachtet. Die ersten drei Punkte führen zu dem Schema in Abb. 13.14 und der migrierten Ausprägung in Abb. 13.15. Die Anwendung der Schema-Transformation “Verschieben des Endes einer Assoziation zu einer Oberklasse” führt zuerst zu dem Schnittstellen-Schema in Abb. 13.16. Dabei werden eine neue Unterklasse *X* aus der Klasse *Haftpflichtversicherung* entfaltet sowie das Ziel der Assoziation *produkt* zur Klasse *X* verschoben. Die Übertragung der linken Seite der Schema-Transformation mit Hilfe des Pullback-Funktors führt zu der migrierten Ausprägung in Abb. 13.17. Dabei werden entsprechend dem veränderten Schema aus jedem Partikel zur Klasse *Haftpflichtversicherung* ein neues Unterpartikel zur Klasse *X* entfaltet sowie die Ziele der *produkt*-Verknüpfungen zu den neuen *X*-Partikeln verschoben.

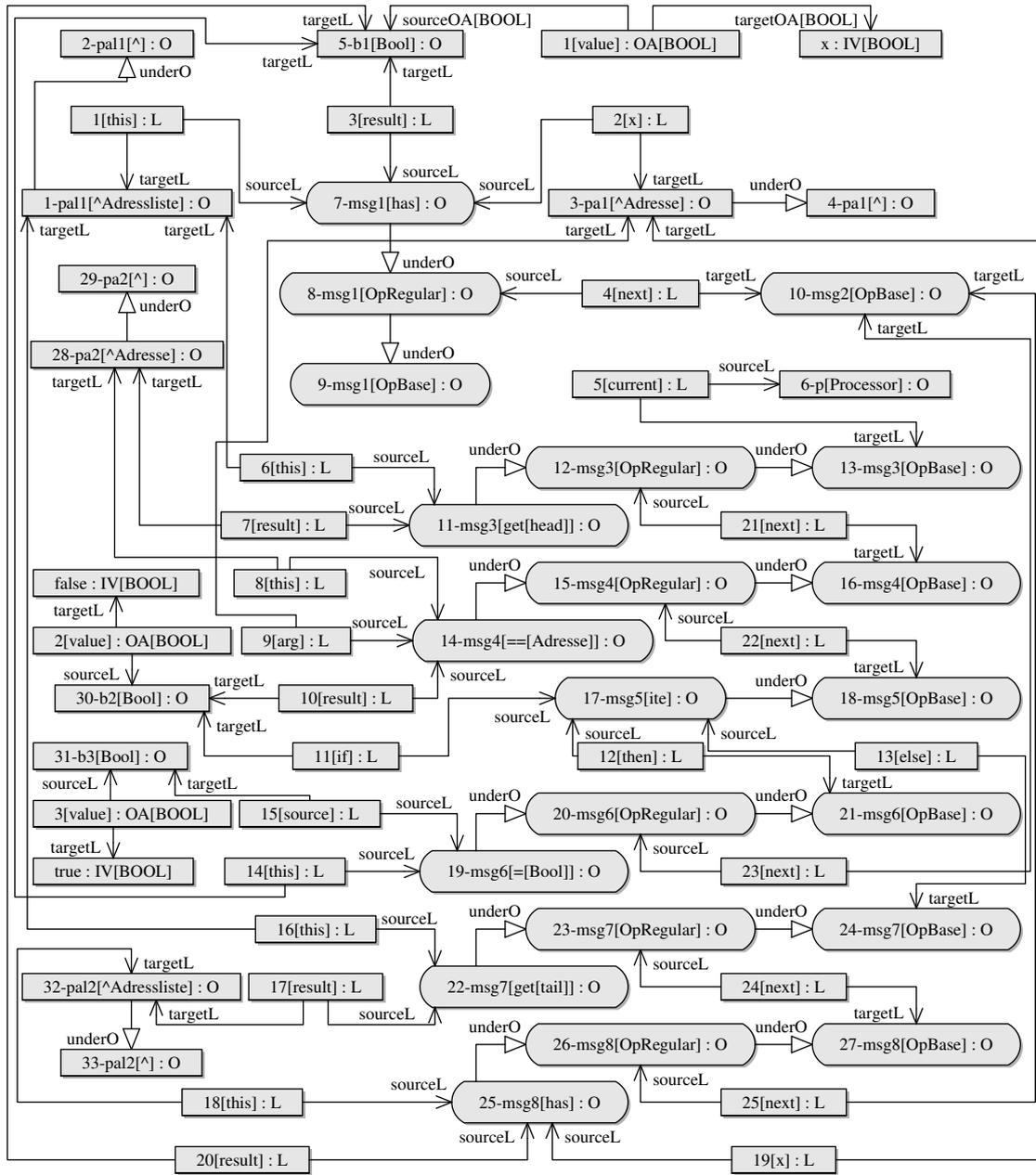


Abb. 13.13: Ausgangssituation: *has*-Methode (rechte Seite)

Das Zielschema der Schematransformation ist in Abb. 13.18 zu sehen. Hier wird die neu eingeführte Klasse X mit der bestehenden Klasse *Versicherung* zusammengesetzt. Die Migration entlang des Kompositionsfunktors ergibt die Ausprägung in Abb. 13.19. Die einzige Veränderung ist die Umtypisierung der X -Partikel auf Grund der zusammengesetzten Klassen. An diesem Beispiel ist zu erkennen, dass die Ausprägung nach Anwendung des Kompositionsfunktors nicht unbedingt alle M -Axiome erfüllt, weil zwei Partikel zu derselben Klasse in demselben Objekt existieren. Die weitere Migration dieser Ausprägung mit Hilfe des Epirefektors führt zu der migrierten und korrekten Ausprägung in Abb. 13.20. Dabei werden die Partikel zu den Klassen X und *Versicherung* zusammengesetzt. Dies geschieht auf Grund von Axiom (M.28), da die zusammenzulegenden Partikel im Zielschema in derselben Klasse *Versicherung* typisiert sind. Dies zeigt, dass gemäß Satz 9.17 die Faktorisierung nach den Axiomen (M.21) und (M.28) zur Konstruktion der Epireflexion ausreichend ist, wobei in dem Beispiel ein zusätzlicher transitiver Abschluss der Vererbungsverknüpfungen nicht erforderlich ist. Das Ergebnis der Migration ist eine zur Spezifikation M konforme Ausprägung und demonstriert die Korrektheit der Migration von Daten und Prozessen gemäß Theorem 11.12.

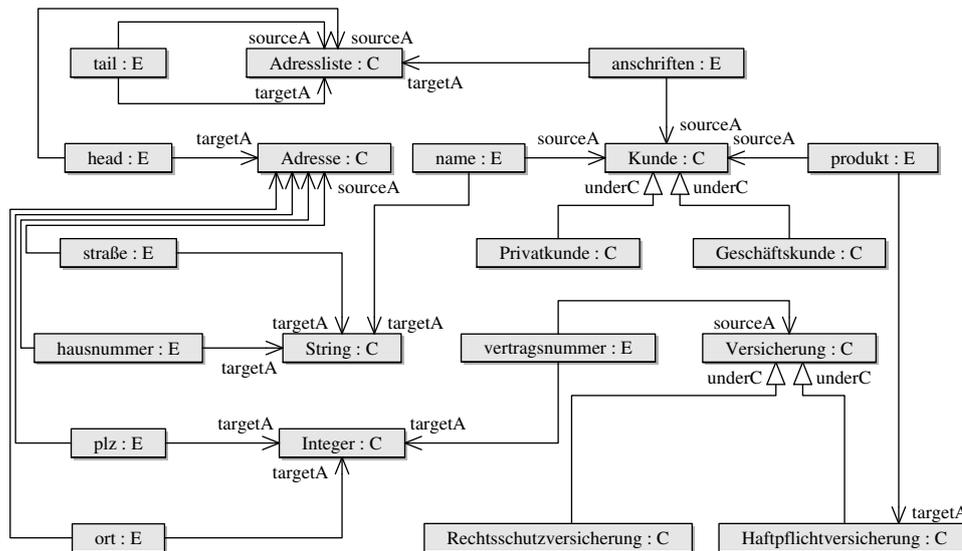


Abb. 13.14: Erweiterung des Versicherungsmodells: Ausgangsschema

Die folgenden Schema-Änderungen können nicht umgesetzt werden:

- Verschieben der Quelle des von der Operation $get[vertragsnummer]$ ausgehenden Parameters $this$ zur Klasse *Versicherung*
- Verschieben des Ziels des von der Operation $get[produkt]$ ausgehenden Parameters $result$ zur Klasse *Versicherung*

Dies hat zur Folge, dass die jeweiligen Operationen und Methoden weiterhin nur für Objekte der Klasse *Haftpflichtversicherung* definiert sind bzw. nur Objekte der Klasse *Haftpflichtversicherung*

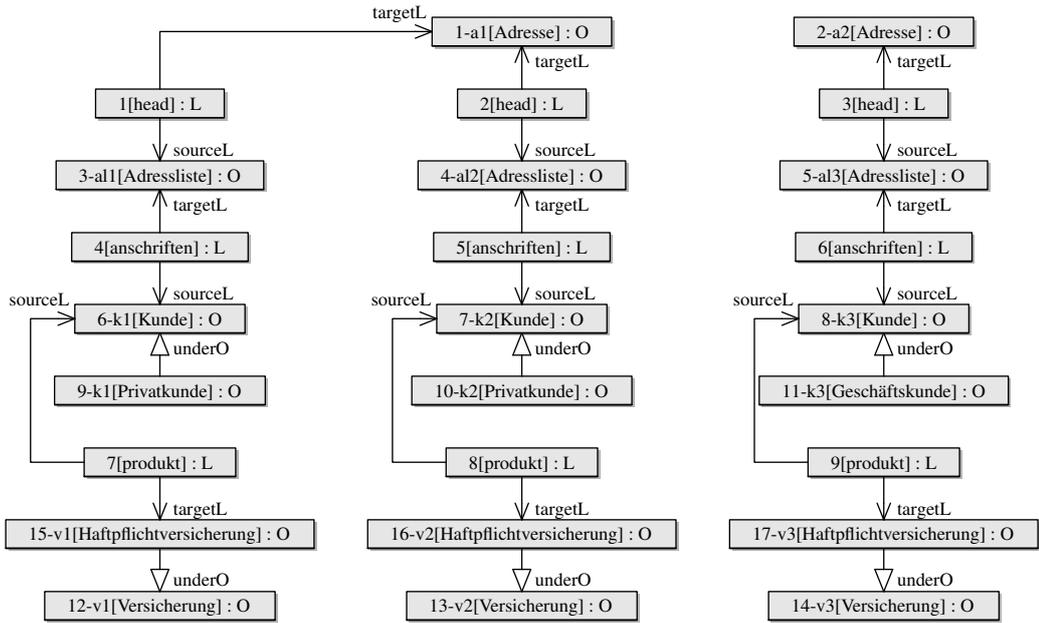


Abb. 13.15: Erweiterung des Versicherungsmodells: Ausgangsausprägung

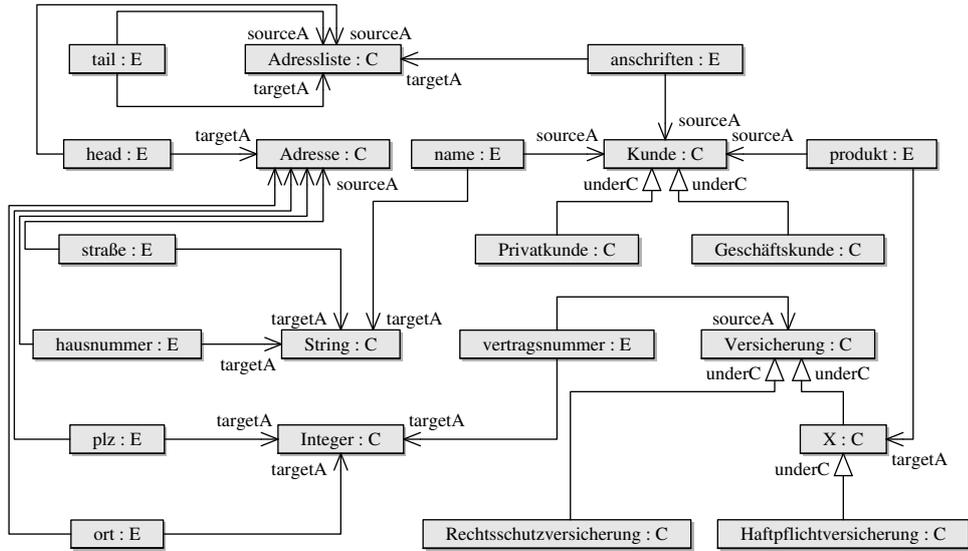


Abb. 13.16: Erweiterung des Versicherungsmodells: Schnittstellen-Schema

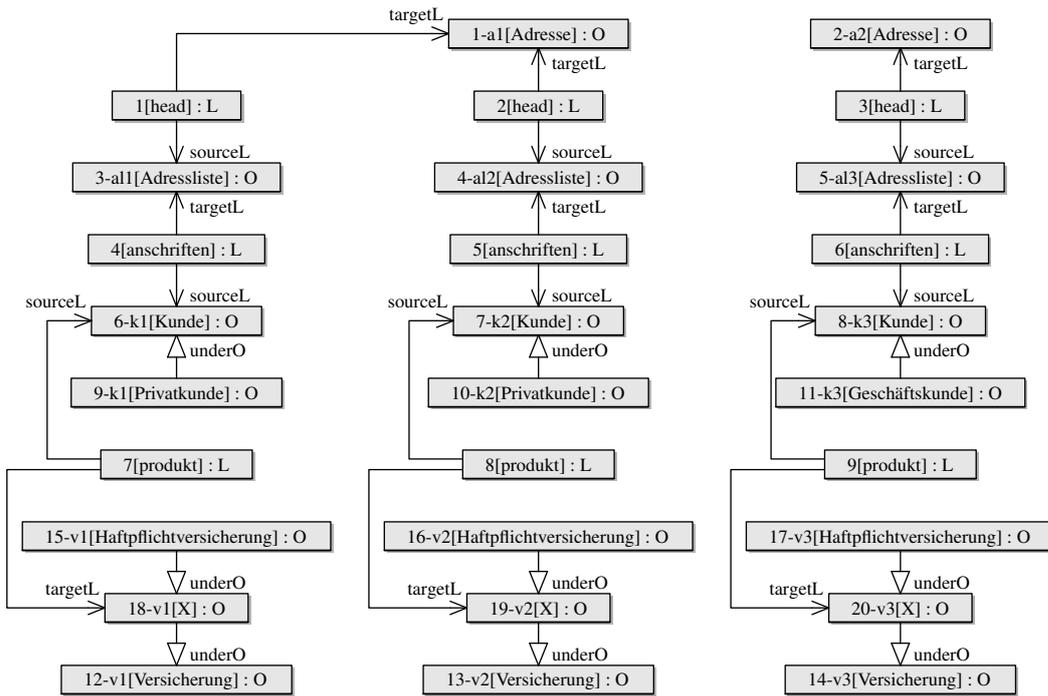


Abb. 13.17: Erweiterung des Versicherungsmodells: Ausprägung nach Anwendung des Pullback-Funktors

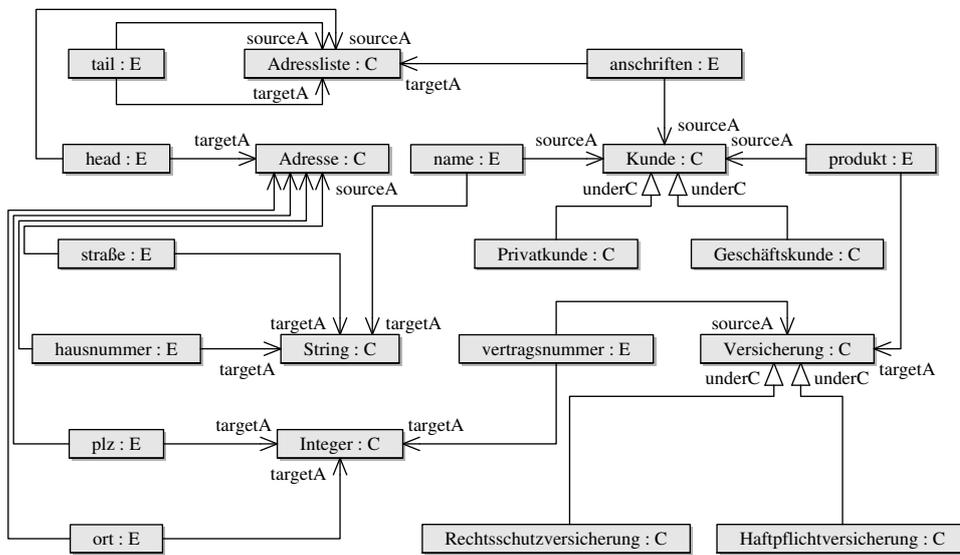


Abb. 13.18: Erweiterung des Versicherungsmodells: Zielschema

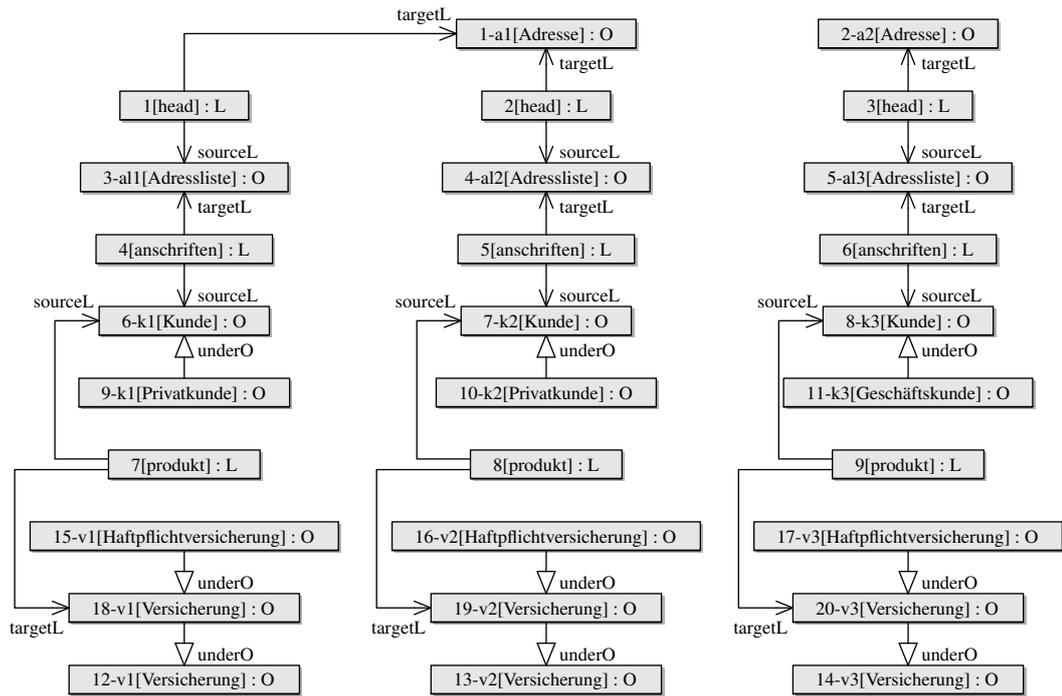


Abb. 13.19: Erweiterung des Versicherungsmodells: Ausprägung nach Anwendung des Kompositionsfunktors

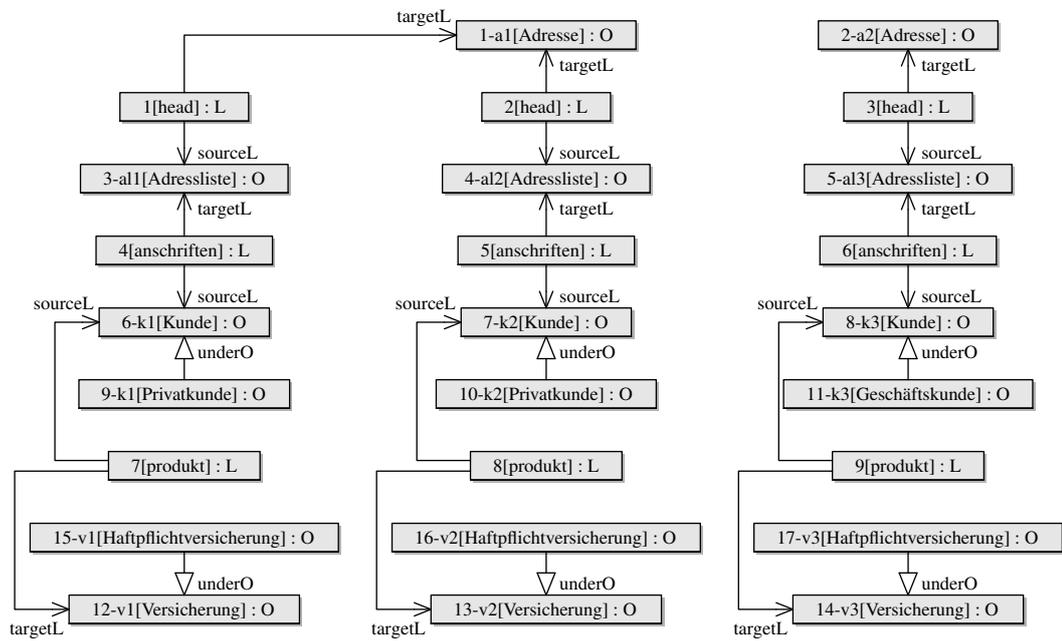


Abb. 13.20: Erweiterung des Versicherungsmodells: Ausprägung nach Anwendung des Epirefektors

liefern können. Dies stellt für laufende Prozesse kein Problem dar, weil dort keine Objekte zur neuen Klasse *Rechtsschutzversicherung* auftreten. Im Zuge der weiteren Programmentwicklung müssen die fehlenden Operationen und Methoden jedoch nachgereicht bzw. die bestehenden Operationen und Methoden geeignet verallgemeinert werden.

13.3 Phase 2: Hinzunahme ausländischer Kunden

In dieser Phase werden Anpassungen am Adress-Modell vorgenommen. Die folgenden Schema-Änderungen aus Abschnitt 6.3 können über entsprechende Schema-Transformationen aus Abschnitt 12.3 formuliert werden:

- Umbenennung der Klasse *Adresse* in *Inlandsadresse* (dies erfordert keinerlei Veränderungen des Modells, weil Namen von Elementen nicht Teil des Modells sind)
- Erstellung der Klasse *Adresse* als Oberklasse der Klasse *Inlandsadresse* durch die Transformation “Erzeugen einer Oberklasse” (12.3.1.2)
- Erstellung der Klasse *Auslandsadresse* als Unterklasse der Klasse *Adresse* durch die Transformation “Erzeugen einer Unterklasse” (12.3.1.3)
- Verschieben des Ziels der von der Klasse *Adressliste* ausgehenden Assoziation *head* zur Klasse *Adresse* durch die Transformation “Verschieben des Endes einer Assoziation zu einer Oberklasse” (12.3.2.4)

Die folgenden Schema-Änderungen können nicht umgesetzt werden:

- Verschieben des Ziels des von der Operation *get/head* ausgehenden Parameters *result* zur Klasse *Adresse*
- Verschieben des Ziels des von der Operation *has* ausgehenden Parameters *x* zur Klasse *Adresse*

Die Migration der Daten und der Software geschieht analog zur letzten Phase. Das resultierende Schema und die migrierte Ausprägung sind in Abb. 13.21 und Abb. 13.22 dargestellt.

Ein wichtiger Unterschied zum konzeptionellen Modell in dieser Phase ist, dass bedingt durch die aufgezählten nicht unterstützten Schema-Änderungen die *get/head*-Operation der Klasse *Adressliste* und die zugehörige Methode nicht so angepasst werden können, dass sie allgemeine Adressen zurückliefern. Somit resultiert die Migration in einer *get/head*-Operation, deren *result*-Parameter weiterhin ein Objekt der Klasse *Inlandsadresse* zurückgibt. Analog ist die *get/head*-Methode so formuliert, dass das *result*-Argument mit einem Verweis auf ein Partikel der Klasse *Inlandsadresse* belegt wird. Dies ist in den Abbildungen 13.23 bis 13.26 dargestellt. Dasselbe Problem tritt bei der *has*-Operation und -Methode der Klasse *Adressliste* in Bezug auf den Typ des Parameters bzw. des Arguments *x* auf. Das Ergebnis der Migration dieser Operation und

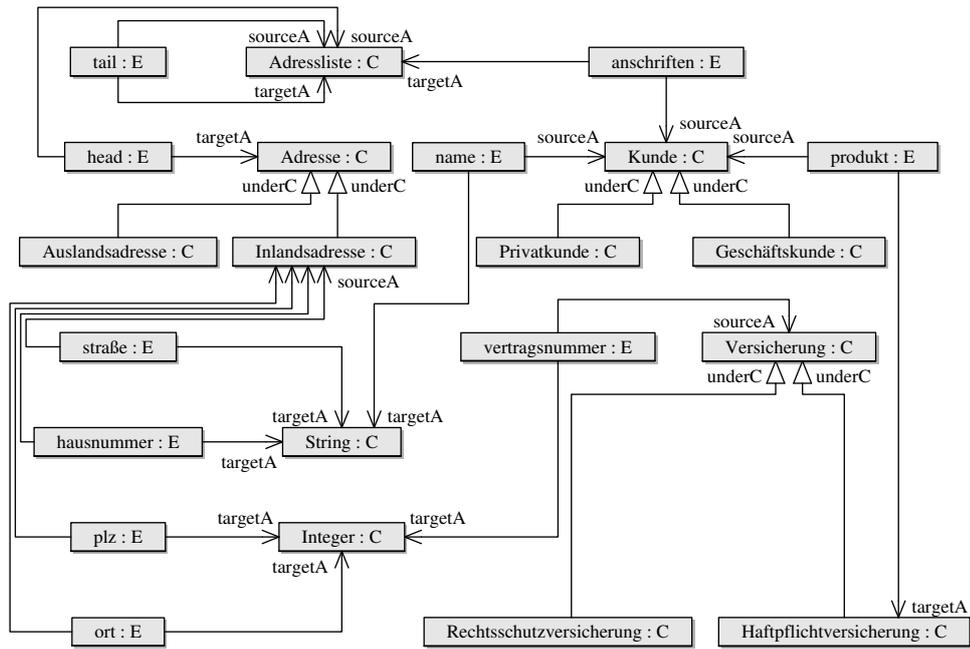


Abb. 13.21: Erweiterung des Adressmodells: Zielschema

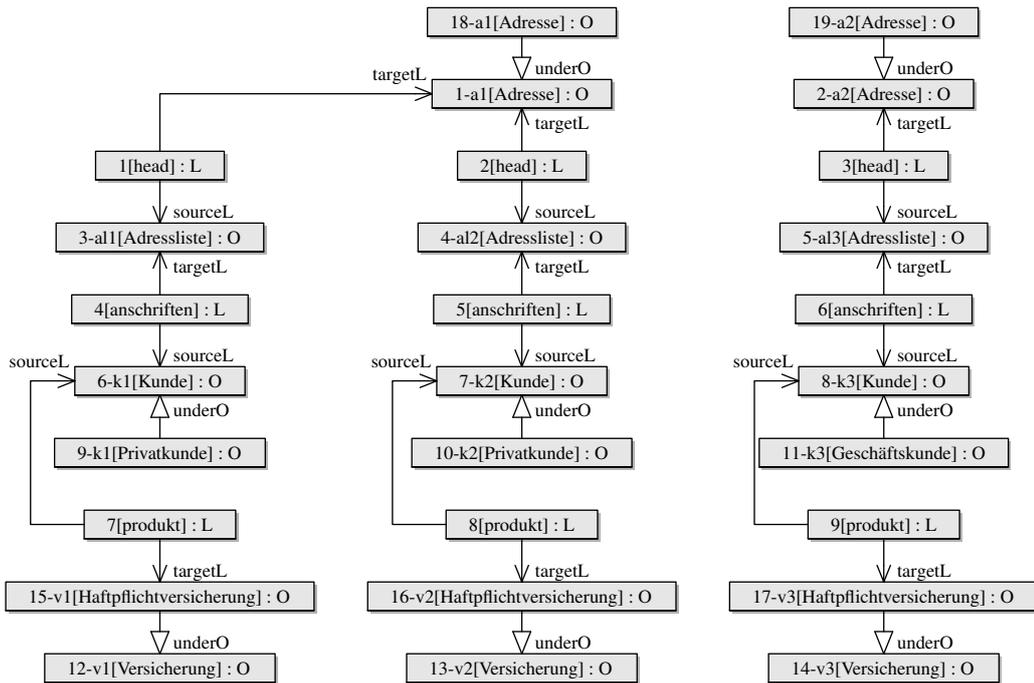


Abb. 13.22: Erweiterung des Adressmodells: Migrierte Ausprägung

zugehörigen Methode ist in den Abbildungen 13.27 bis 13.30 zu sehen. Wie zu sehen ist, erhält die Migration der Methoden konsistente Aktionen: Zum einen sind die Einbettungen der Schnittstelle in den linken und rechten Teil vervollständigend, zum anderen ziehen die Methoden Kanten zurück. Dies demonstriert die Korrektheit der Migration von Programmen gemäß Theorem 11.35.¹²¹

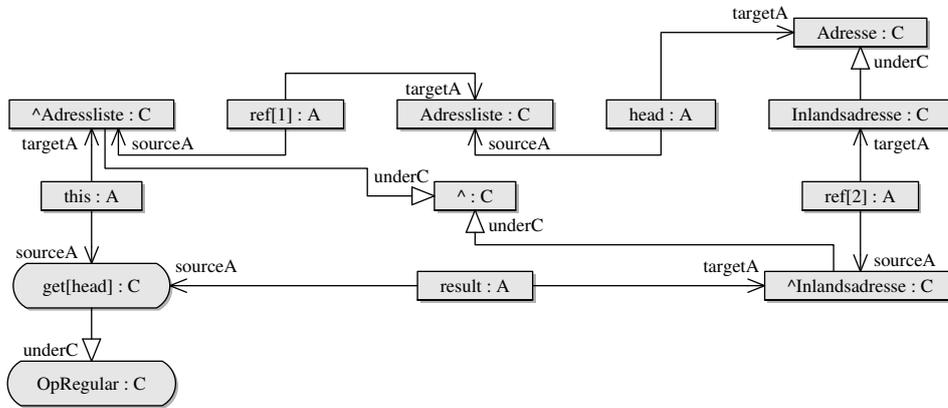


Abb. 13.23: Erweiterung des Adressmodells: *get/head*-Operation

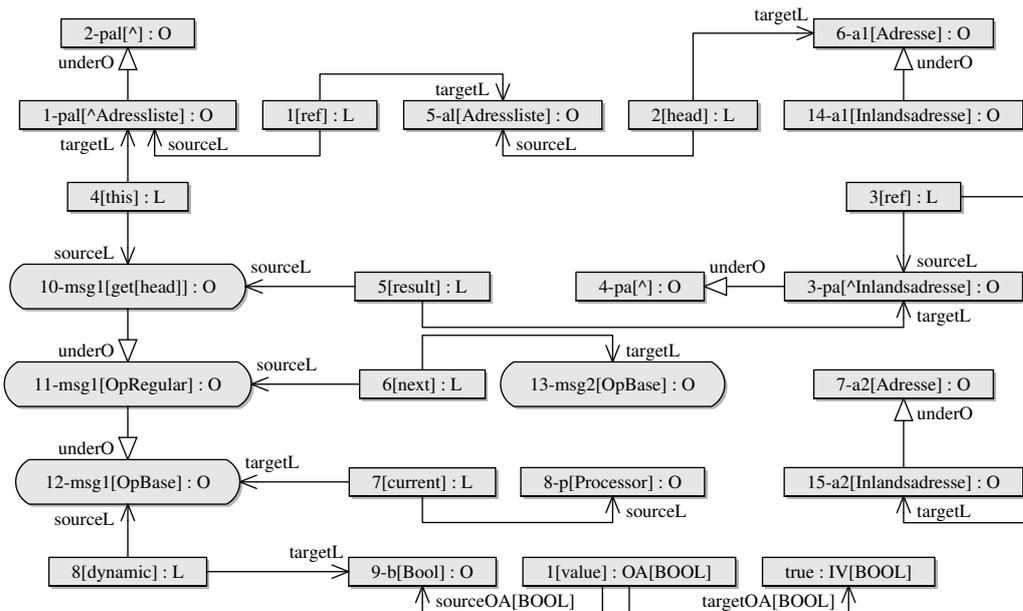


Abb. 13.24: Erweiterung des Adressmodells: *get/head*-Methode (linke Seite)

Zu beachten ist, dass das beschriebene Problem keine Auswirkung auf laufende Prozesse hat, weil dort nur Objekte zur Klasse *Inlandsadresse* vorkommen und somit keine Änderungen im

¹²¹ Die Übertragung partiell injektiver Ansätze und von Doppel-Pushout-Diagrammen kann aus Platzgründen im Rahmen dieser kleinen Fallstudie nicht demonstriert werden.

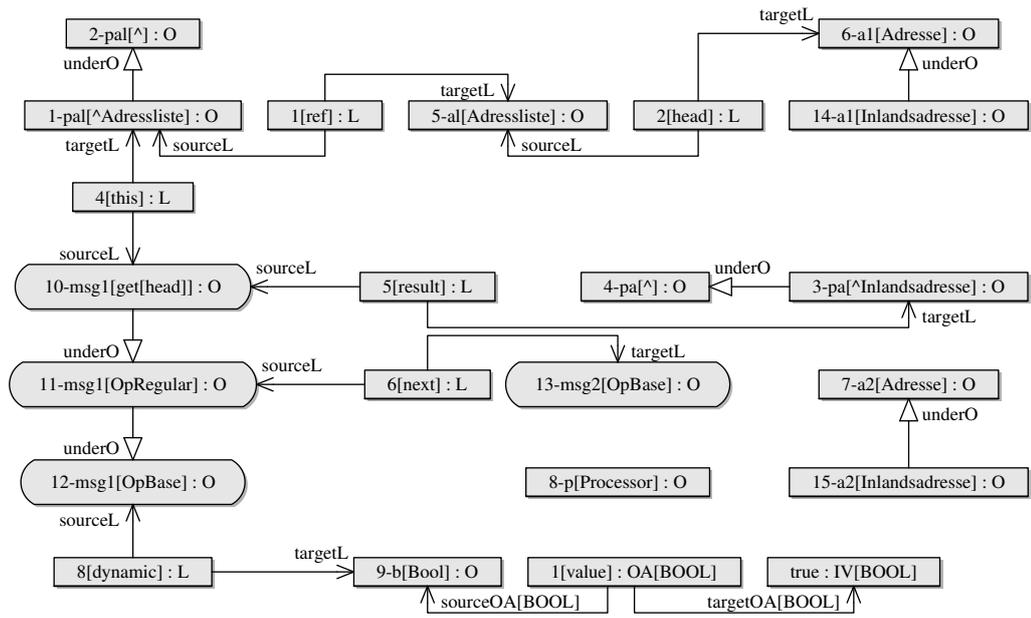


Abb. 13.25: Erweiterung des Adressmodells: `get/head`-Methode (Schnittstelle)

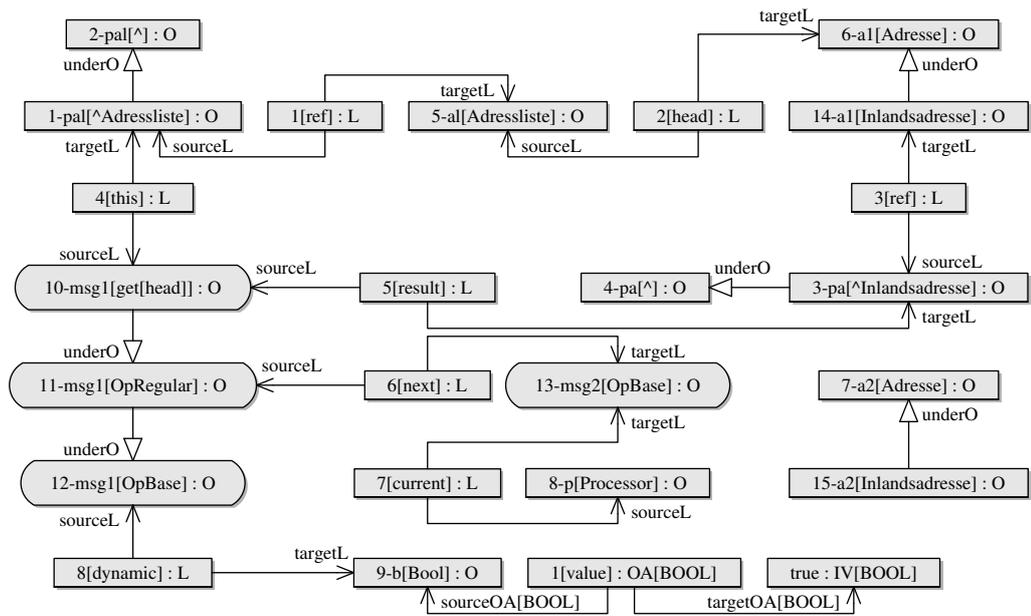


Abb. 13.26: Erweiterung des Adressmodells: `get/head`-Methode (rechte Seite)

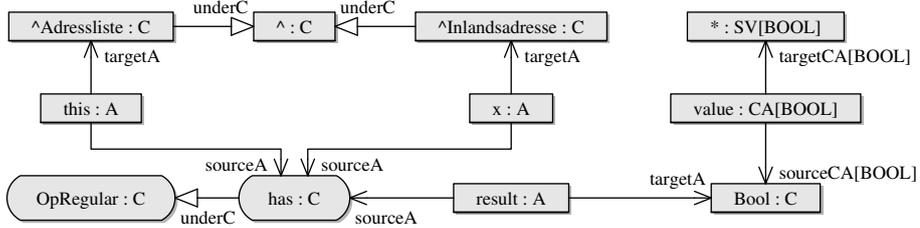


Abb. 13.27: Erweiterung des Adressmodells: *has*-Operation

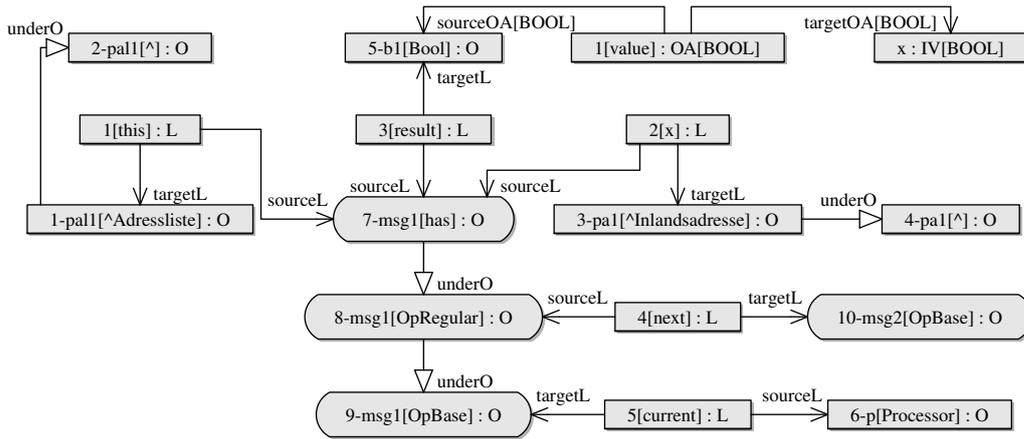


Abb. 13.28: Erweiterung des Adressmodells: *has*-Methode (linke Seite)

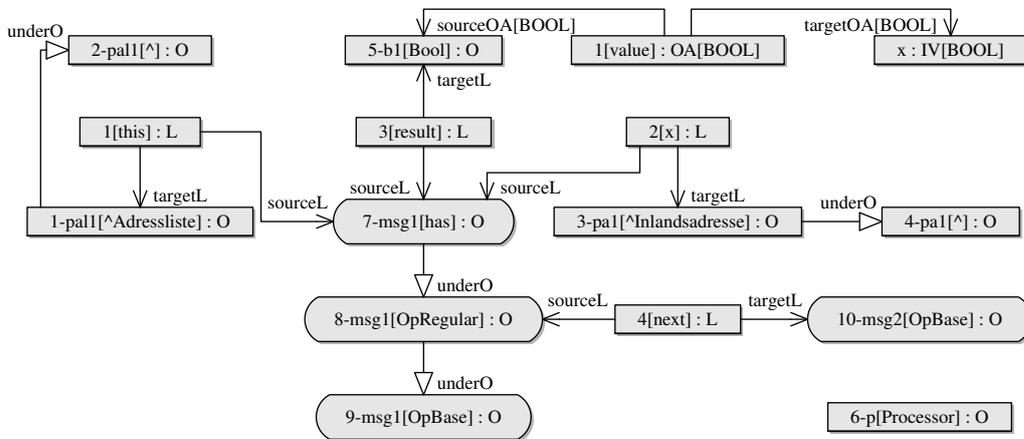


Abb. 13.29: Erweiterung des Adressmodells: *has*-Methode (Schnittstelle)

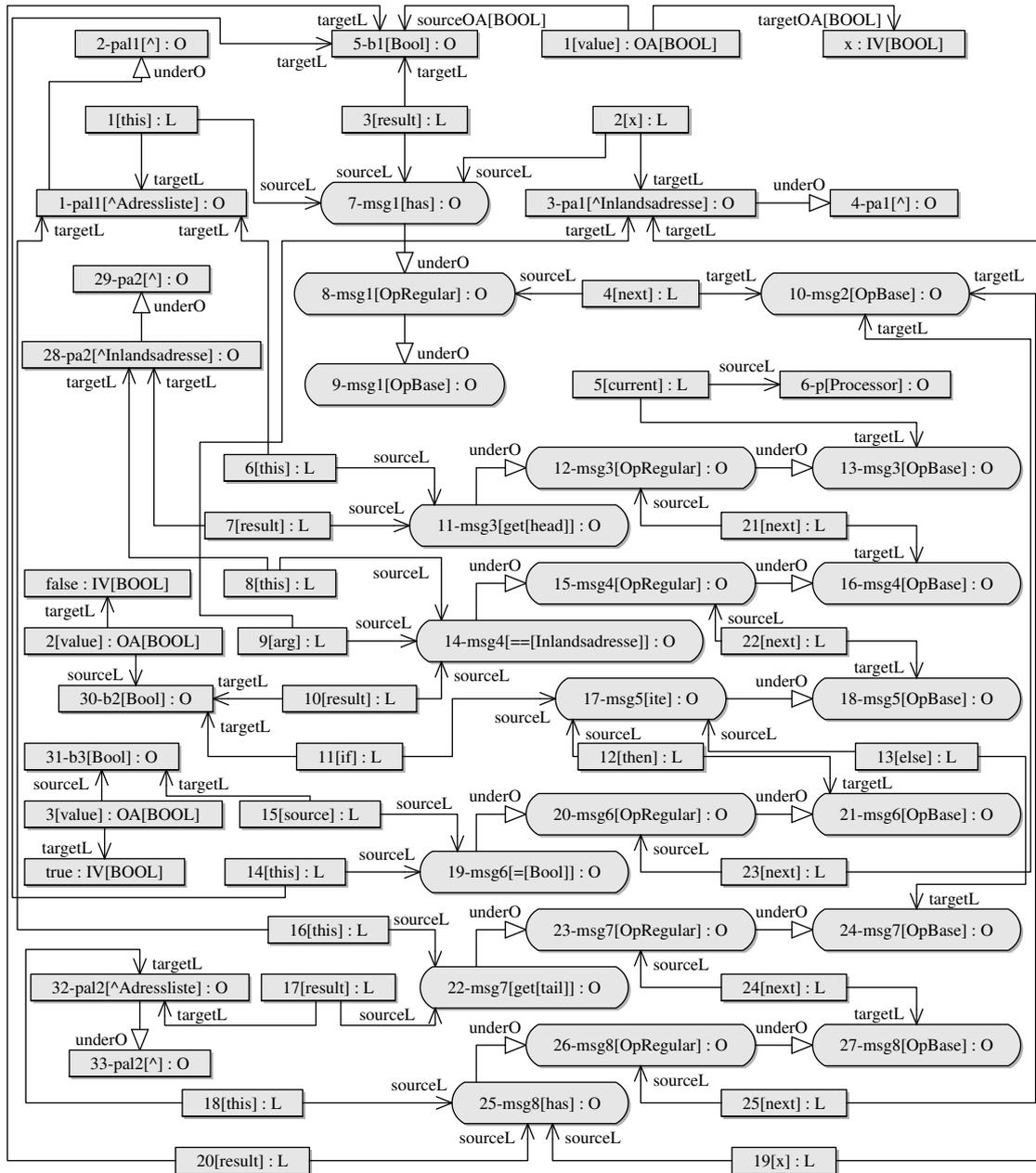


Abb. 13.30: Erweiterung des Adressmodells: *has*-Methode (rechte Seite)

Verhalten zu erwarten sind. Für zukünftige Programmentwicklungen ist es jedoch notwendig, die Verallgemeinerung der *has*-Operation und -Methode manuell vorzunehmen.

13.4 Phase 3: Konzentration auf Geschäftskunden

In dieser Phase werden Veränderungen am Kundenmodell vorgenommen. Alle Schema-Änderungen aus Abschnitt 6.4 können über entsprechende Schema-Transformationen aus Abschnitt 12.3 formuliert werden:

- Löschung der Klasse *Privatkunde* durch die Transformation “Löschen einer Klasse” (12.3.1.8)
- Verschmelzung der Klassen *Geschäftskunde* und *Kunde* zur Klasse *Geschäftskunde* durch die Transformation “Verschmelzen zweier verwandter Klassen” (12.3.1.4)

Der zweite Teil der folgenden zusätzlichen Schema-Änderung kann nicht umgesetzt werden, weil die *ref*-Assoziationen nicht verschmolzen werden können (vgl. auch Abschnitt 14.2.1):

- Verschmelzung der Klassen \uparrow *Geschäftskunde* und \uparrow *Kunde* zur Klasse \uparrow *Geschäftskunde* sowie der zugehörigen *ref*-Assoziationen

Im Folgenden werden die beiden Punkte im Detail betrachtet. Im ersten Schritt führt die Anwendung der Schema-Transformation “Löschen einer Klasse” zu dem Schnittstellen-Schema in Abb. 13.31. Die Übertragung der linken Seite der Schema-Transformation mit Hilfe des Pullback-Funktors führt zu der migrierten Ausprägung in Abb. 13.32. Dabei werden entsprechend dem veränderten Schema alle Partikel zu der Klasse *Privatkunde* gelöscht. Das Schema auf der rechten Seite verändert sich nicht, weil kein Erweitern um Elemente oder Zusammenlegen von Elementen stattfindet. Somit stellt die migrierte Ausprägung in Abb. 13.32 das Ergebnis der gesamten Migration zur ersten Transformation dar.

Im zweiten Schritt müssen zur Anwendung der Transformation “Verschmelzen zweier verwandter Klassen” im Schnittstellen-Schema Oberklassen der Klasse *Kunde* aus der alten Klasse *Geschäftskunde* entfaltet werden. Weil die Klasse *Kunde* keine Oberklassen besitzt, bleibt das Schnittstellen-Schema somit unverändert und entspricht dem Schema in Abb. 13.31. Die Anwendung des Pullback-Funktors entlang eines Isomorphismus’ verändert die Ausprägung nicht. Folglich ist auf der linken Seite die migrierte Ausprägung identisch zu der Ausprägung in Abb. 13.32. Auf der rechten Seite werden die Klassen *Kunde* und *Geschäftskunde* zur neuen Klasse *Geschäftskunde* zusammengefasst (Abb. 13.33). Die Migration entlang des Kompositionsfunktors ergibt die Ausprägung in Abb. 13.34. Die einzige Veränderung ist die Umtypisierung der *Geschäftskunde*-Partikel auf Grund der zusammengelegten Klassen. Die weitere Migration dieser Ausprägung mit Hilfe des Epirefektors führt zu der migrierten und korrekten Ausprägung in Abb. 13.35. Dabei werden die nun doppelt vorhandenen Partikel zu der Klasse *Geschäftskunde* zusammengelegt.

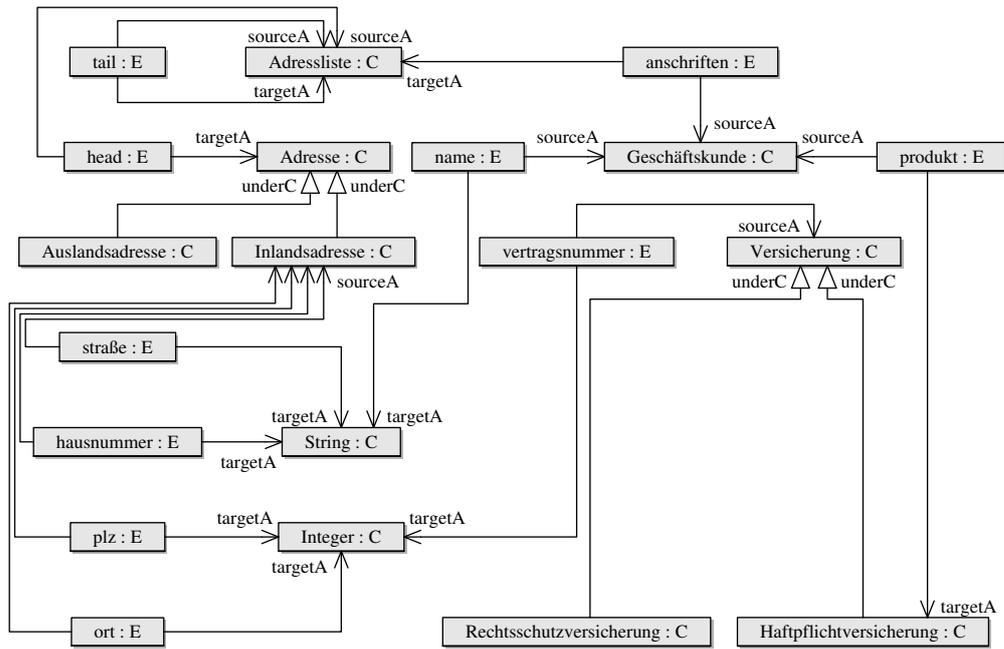


Abb. 13.33: Reduktion des Kundenmodells (2): Zielschema

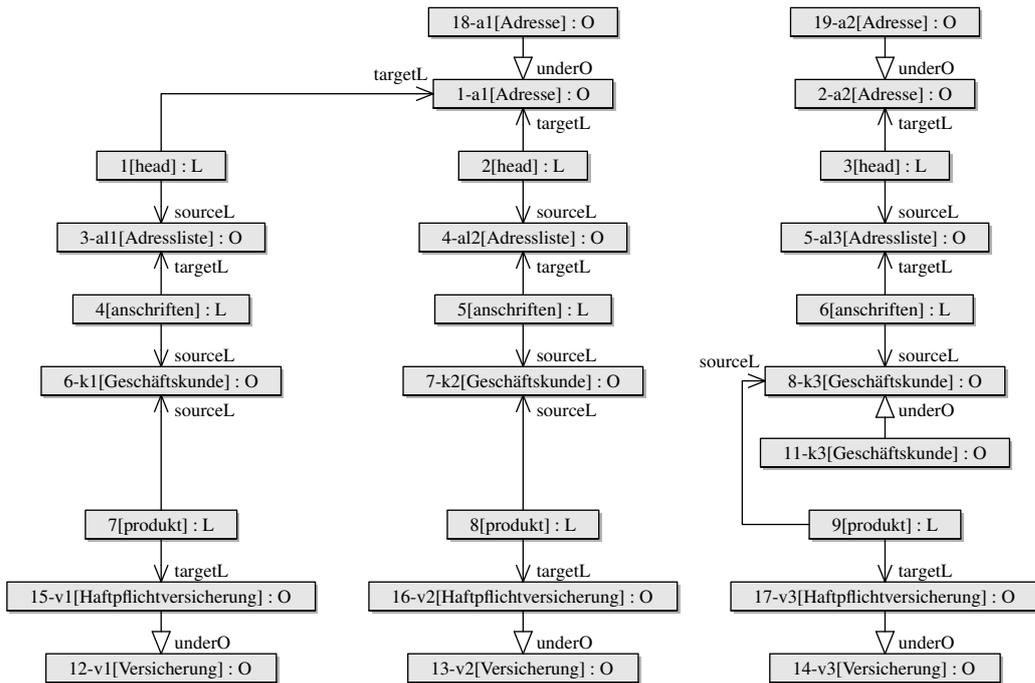


Abb. 13.34: Reduktion des Kundenmodells (2): Ausprägung nach Anwendung des Kompositions-funktors

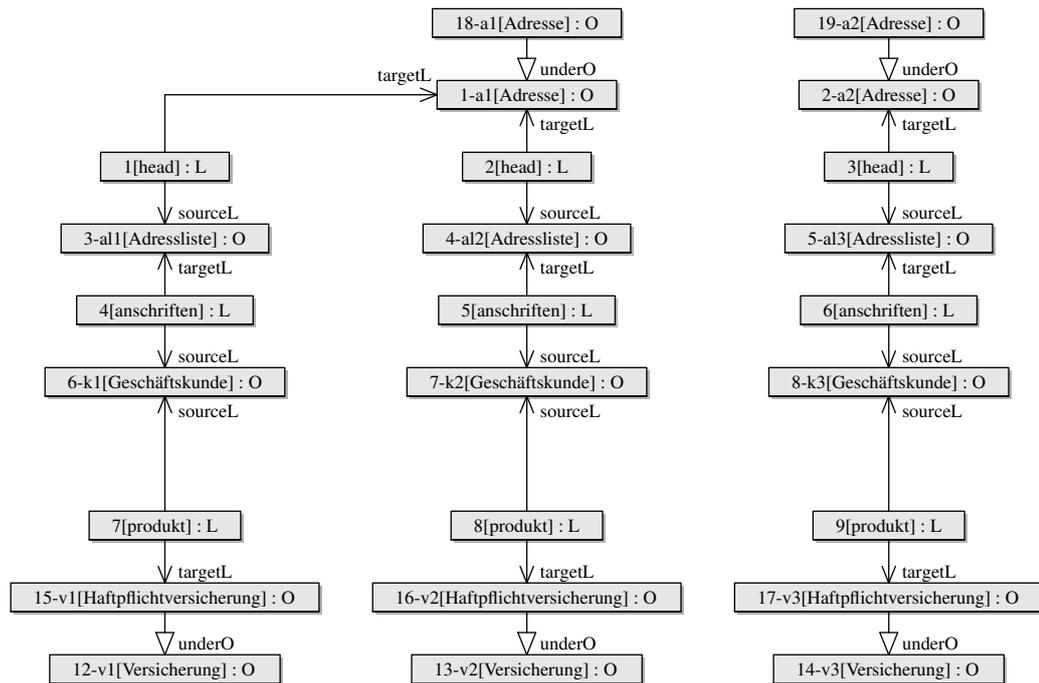


Abb. 13.35: Reduktion des Kundenmodells (2): Ausprägung nach Anwendung des Epirefektors

Abbildung 13.35 macht deutlich, dass wenn noch Privatkunden im Kundenbestand vor der Schema-Transformation vorhanden sind, die Privatkunden während der Migration zu Geschäftskunden werden. Dies könnte im Rahmen einer Modellerweiterung vermieden werden, indem die Klasse *Kunde* eine abstrakte Klasse wird, denn dann müssen die allein stehenden Partikel *6-k1* und *7-k2*, die in der Migration nach dem Löschen der Klasse *Privatkunde* auftauchen (Abb. 13.32), gelöscht werden. Mehr zu diesem Problem steht in Abschnitt 14.1.1.

13.5 Komposition von Transformationen

In diesem Abschnitt soll am Beispiel aufgezeigt werden, wie Schema-Transformationen komponiert und auf der Ausprägung als Migration fortgesetzt werden. Als Grundlage dient das Ausgangsschema (Abb. 13.1) sowie die zugehörige Ausprägung (Abb. 13.2). Es sollen die Schema-Transformationen zur Erstellung der Klasse *Versicherung* als Oberklasse der Klasse *Haftpflichtversicherung* sowie die Schema-Transformation zum Verschieben des Ziels der von der Klasse *Kunde* ausgehenden Assoziation *produkt* zur Klasse *Versicherung* komponiert und die induzierte Migration dieser komponierten Schema-Transformation berechnet werden. Die einzelnen Schema-Transformationen “Erzeugen einer Oberklasse” und “Verschieben des Endes einer Assoziation zu einer Oberklasse” sind in Abb. 13.36 bezogen auf das gegebene Schema dargestellt, wobei nur der

relevante Kontext gezeigt wird. Aus Platzgründen werden die Klassen *Kunde*, *Versicherung* und *Haftpflichtversicherung* in den Diagrammen im Folgenden durch *K*, *V* und *HPV* abgekürzt.

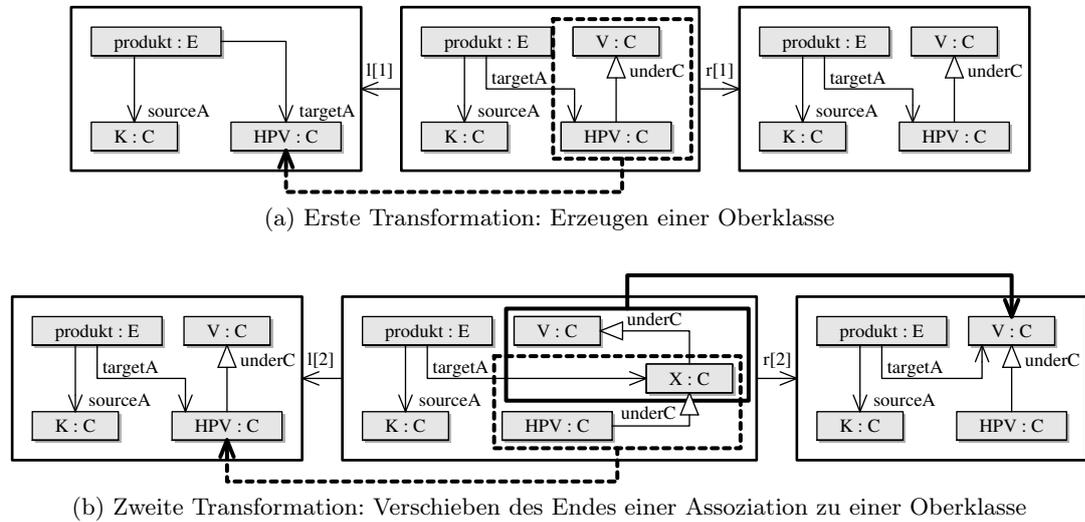


Abb. 13.36: Komposition: Zwei komponierbare Schema-Transformationen

Beide Schema-Transformationen sind nach Def. 11.45 komponierbar, und die linke Seite der zweiten Schema-Transformation entspricht der rechten Seite der ersten Schema-Transformation. Somit lässt sich eine einzige komponierte Schema-Transformation aus den beiden einzelnen Schema-Transformationen durch das Bilden einer Pullback-Graphstruktur bilden. Die resultierende Schema-Transformation ist in Abb. 13.37 dargestellt. Die Konstruktion der komponierten Schema-Transformation ist in Abb. 13.38 verdeutlicht. Es lässt sich leicht überblicken, dass die Komposition der einzelnen induzierten Migrationen dieselbe Ausprägung ergibt wie die induzierte Migration zu der komponierten Schema-Transformation und dass somit die Komposition gemäß Theorem 11.50 korrekt ist.

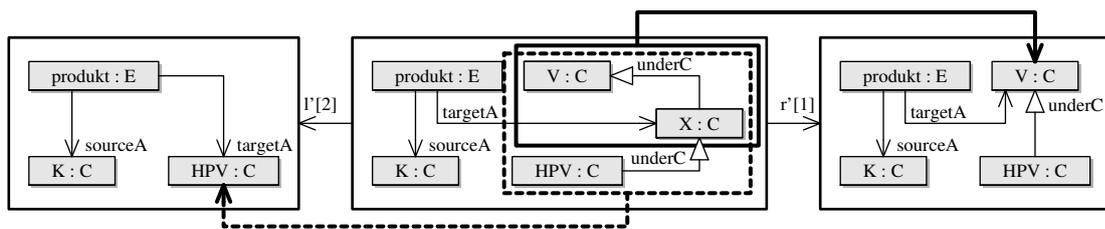


Abb. 13.37: Komposition: Komponierte Schema-Transformation

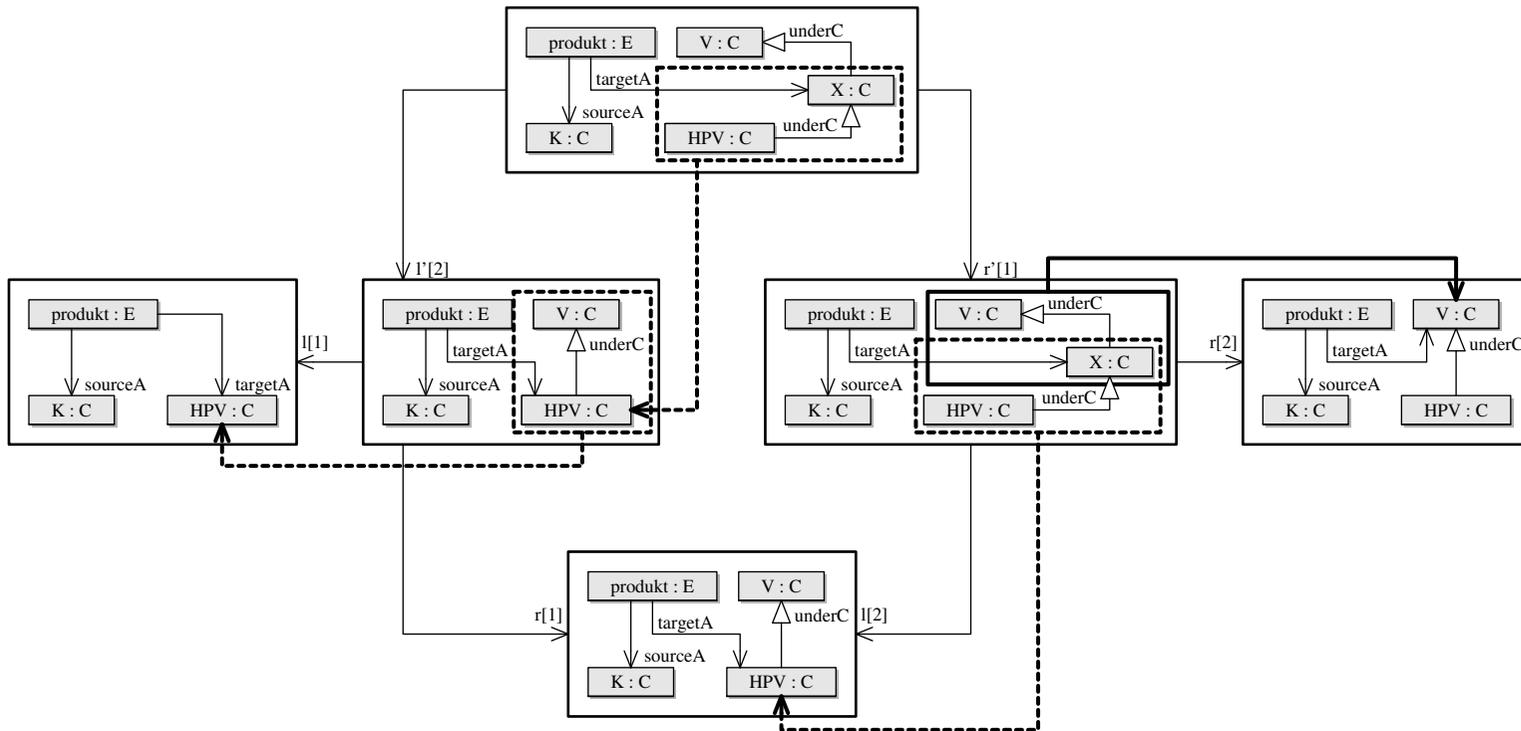


Abb. 13.38: Komposition: Konstruktion der komponierten Schema-Transformation durch Pullback-Bildung

Teil IV

Ausblick

Einschränkungen und Erweiterungen

Das in Teil III beschriebene mathematische Modell ist in der Lage, viele wichtige Eigenschaften objektorientierter Systeme abzubilden. Neben den zentralen Schema-Konzepten Klasse, Assoziation und Attribut sowie den jeweiligen Entsprechungen Objekt, Verknüpfung und Wert auf der Ausprägungsebene können Veränderungen der Schema- und Ausprägungs-Anteile über Spans von Homomorphismen formuliert werden. Somit bietet das Modell eine gute Grundlage für das Refactoring objektorientierter Systeme.

Es besitzt jedoch auch einige Einschränkungen, die in den folgenden Abschnitten näher beschrieben werden. Einige Einschränkungen beziehen sich auf Schwächen des Modells, Schemata und Ausprägungen “ordentlich” zu spezifizieren:

- *Abstrakte Klassen:* Das Modell bietet keine Unterstützung für die Abbildung der Semantik abstrakter Klassen (14.1.1).
- *Komplexe Assoziationen:* Das Modell unterstützt nur einfache Assoziationen. Weder die Semantik komplexerer Beziehungen wie Kompositionen (14.1.2) noch Mindestens-eins-Beschränkungen an Quelle und Ziel von Assoziationen (14.1.3) können vom Modell erzwungen werden.
- *Vollständige Objekte:* Das Modell stellt nicht sicher, dass Objekte alle nötigen Partikel besitzen (14.1.4).
- *Zusammenhängende Objekte:* Das Modell stellt nicht sicher, dass Objekte zusammenhängend sind (14.1.5).
- *Direkte vs. indirekte Vererbung:* Im Modell wird nicht zwischen direkter und indirekter Vererbung unterschieden (14.1.6).
- *Typisierung von Daten in Software:* Das Modell erlaubt eine Typisierung von Daten in Software (14.1.7).
- *Adhäsive HLR-Kategorien:* Das Modell nutzt bei der Formulierung von Programmen als DPO-Regeln nicht den theoretischen Unterbau adhäsiver HLR-Kategorien (14.1.8).

Andere Einschränkungen beziehen sich auf Schwächen des Modells im Bereich der Schema- und Ausprägungs-Transformationen:

- *Symmetrie von Transformationen:* Assoziationen können entfaltet, aber nicht gefaltet werden, sofern Programme und Prozesse existieren (14.2.1).
- *Verallgemeinerung von Daten-Parametern:* Daten-Parameter können nicht generalisiert werden. Somit können Operationen auch nicht verallgemeinert werden (14.2.2).
- *Mehrdeutigkeit nach Prozess-Transformation:* Unter bestimmten Umständen können transformierte Prozesse zum neuen Schema unter ansonsten gleichen Bedingungen zu anderen Ergebnissen führen als die ursprünglichen Prozesse zum alten Schema (14.2.3).
- *Erhaltung der Semantik von Programmen:* Das Modell stellt formal nicht sicher, dass ein Programm nach einer von einem Refactoring induzierten Migration verhaltensmäßig äquivalent zu dem Programm vor der Migration ist (14.2.4).
- *Daten-getriebene Schema-Transformationen:* Schema-Transformationen dürfen keine Abhängigkeiten zur Ausprägungs-Ebene besitzen, was einige interessante Migrationen ausschließt (14.2.5).
- *Formalisierung von Refactorings:* Die mathematische Beschreibung von Refactorings als Schema-Transformationen, deren linke Seite surjektiv ist, ist unzulänglich, wenn verlangt wird, dass ein migriertes System ein eindeutiges Urbild unter der Migration besitzt (14.2.6).

In einigen Fällen werden mögliche Lösungsansätze zur Überwindung der aufgezeigten Einschränkungen angerissen. Allen Lösungsansätzen ist jedoch gemein, dass ihre Integration in das bestehende Modell wegen seiner Komplexität eine nicht triviale Angelegenheit ist und weiterer Untersuchungen bedarf.

14.1 Datenbank- und Software-Einschränkungen

In diesem Abschnitt werden Einschränkungen beschrieben, die sich auf den Datenbank- und Software-Teil des Modells beziehen.

14.1.1 Abstrakte Klassen

Das mathematische Modell unterstützt bis jetzt nur *konkrete* Klassen: Ein Objekt darf zu *jeder* Klasse existieren. In der Software-Entwicklung werden jedoch häufig *abstrakte* Klassen benötigt, zu denen keine Objekte existieren dürfen. Typischerweise werden Schnittstellen zwischen Software-Komponenten mit Hilfe abstrakter Klassen umgesetzt.

Eine mögliche Implementierung der Semantik abstrakter Klassen im mathematischen Modell soll im Folgenden vorgestellt werden. Generell stehen nur (Epi-)Reflexionen und (Mono-)Koreflexionen

zur Verfügung, will man im kategoriellen Kontext eine eindeutige Konstruktion erhalten. Eine Epireflexion erscheint dem Autor an dieser Stelle nicht zweckmäßig, weil sie über das Bilden eines kategoriellen Quotientenobjekts entsteht. Hat man jedoch eine Ausprägung, in der sich ein Objekt zu einer abstrakten Klasse befindet, so gibt es keine intuitive und eindeutige “Zusammenlegung” dieses Partikels mit anderen Partikeln, so dass die resultierende Ausprägung keine abstrakten Objekte mehr besitzt.

Eine Monokoreflexion hingegen ist vielversprechender: Unterscheidet man abstrakte und konkrete Partikel, dann lässt sich durch Weglassen derjenigen abstrakten Partikel, die keine konkreten Unterpartikel besitzen, ein kategorielles Unterobjekt bilden, das nur noch konkrete Objekte besitzt. Ist dieses Weglassen in allen Fällen eindeutig, so erhält man eine Monokoreflexion in die Unterkategorie aller Systeme ohne abstrakte Objekte. Eine Formulierung dieses “Weglassens” lässt sich allerdings nicht allein mit positiven Horn-Formeln bewerkstelligen. Stattdessen wird eine *Existenzaussage* eingeführt, die in Abhängigkeit von einem Prädikat $concreteO$ steht: Ist ein Partikel *nicht konkret* (also abstrakt), dann muss ein Unterpartikel existieren. In der Konstruktion der Koreflexion werden nun alle Partikel, welche die Prämisse, aber nicht die Konklusion erfüllen – also genau diejenigen abstrakten Partikel, die keine Unterpartikel besitzen – aus der Ausprägung entfernt.¹²² Die folgende Spezifikation führt die Prädikate $concreteC$ und $concreteO$ sowie die entsprechenden Axiome ein, wobei das Prädikat $concreteC$ abstrakte *Klassen* im Schema markiert.

$MAbs ::= M +$

prds

$concreteC : C$ (Konkrete Klasse)

$concreteO : O$ (Konkretes Partikel)

axms

Typisierung ist homomorph

(MAbs.1) $x \in O : concreteO(x) \Rightarrow concreteC(typeO(x))$ (bzgl. konkreter Partikel)

Erzwungene Unterobjekte

(MAbs.2) $x \in O : \neg concreteO(x) \Rightarrow \exists y \in O : (y, x) \in underO$ (erzwingt Unterpartikel für abstrakte Partikel)

Dieser Ansatz, so reizvoll er auch erscheinen mag, bedarf weitergehender Analyse. Zum einen muss nachgewiesen werden, dass die beschriebene Konstruktion des Weglassens unerwünschter Partikel auch wirklich eine (Mono-)Koreflexion zur Folge hat. Zum anderen muss die Konstruktion in den Migrationsprozess an der “richtigen” Stelle integriert werden. Experimente haben gezeigt, dass es sinnvoll ist, die Koreflexion nach Anwendung des Pullback-, aber vor Anwendung des Kompositionsfunktors einzubauen. Dies hat den Vorteil, dass auf der linken Seite der Transformation

¹²² Die Prüfung auf *konkrete* Unterpartikel kann auf Grund der rekursiven Anwendung des Axioms entfallen.

die Pullback-Konstruktion erhalten bleibt, weil kofreie Funktoren Limiten und somit Pullbacks übertragen. Allerdings haben weitere Untersuchungen gezeigt, dass die Kompositionalität von Transformationen nicht mehr unter den in dieser Arbeit gezeigten Bedingungen gegeben ist. Denn der kofreie Funktor überträgt nicht immer surjektive Homomorphismen. Dadurch verliert aber das Isomorphielemma (Lemma 11.47) seine Gültigkeit, weil aus der Surjektivität von r'^{t_1} nicht mehr die Surjektivität von i gefolgert werden kann. Hier sind also weitere Untersuchungen notwendig, um die Konstruktion in das bisherige mathematische Modell so zu integrieren, dass seine Eigenschaften wie Kompositionalität von Transformationen möglichst erhalten bleiben.

14.1.2 Kompositionen

Im letzten Abschnitt wurde gezeigt, wie die Semantik abstrakter Klassen über eine kofreie Konstruktion abgebildet werden kann. Dieselbe Idee kann auch für die Spezifikation von Kompositionsstrukturen angewandt werden. Dazu werden die Prädikate $wholeC$ und $wholeO$ definiert, die jeweils Klassen und Partikel markieren, die für sich alleine – d. h. ohne Eigentümer – existieren können. Klassen und Objekte, denen dieses Prädikat fehlt, müssen hingegen jemanden haben, der auf sie zeigt und sie somit “besitzt”. Eine entsprechende Spezifikation $MComp_1$ kann folgendermaßen aussehen:

$MComp_1 ::= M +$

prds

$wholeC : C$ (Eigenständige Klasse)

$wholeO : O$ (Eigenständiges Partikel)

axms

Typisierung ist homomorph

(MComp1.1) $x \in O : wholeO(x) \Rightarrow wholeC(typeO(x))$ (bzgl. eigenständiger Partikel)

Erzwungene Eigentümer

(MComp1.2) $x \in O : \neg wholeO(x) \Rightarrow \exists y \in L : targetL(y) = x$ (erzwingt Eigentümer für Teil-Partikel)

Die Forderung, dass nicht beliebige Assoziationen bzw. Verknüpfungen die Rolle des Besitzens übernehmen können, sondern nur spezielle Kompositionen auf Schema- und Instanzebene, kann durch eine Erweiterung dieser Spezifikation umgesetzt werden. Dazu werden weitere Prädikate, etwa $compA$ und $compL$, benötigt, um gewöhnliche Assoziationen und Verknüpfungen von Kompositionen und Kompositions-Verknüpfungen unterscheiden zu können. Weiter muss das Axiom (MComp1.2) so erweitert werden, dass für die Verknüpfung y das Prädikat $compL$ benötigt wird. Schließlich muss noch ein geeignetes Typisierungs-Axiom analog zu (MComp1.1) formuliert werden. Daraus ergibt sich die Spezifikation $MComp_2$:

$MComp_2 ::= M +$

prds

$wholeC: C$ (Eigenständige Klasse)

$wholeO: O$ (Eigenständiges Partikel)

$compA: A$ (Komposition)

$compL: L$ (Kompositionsverknüpfung)

axms

Typisierung ist homomorph

(MComp2.1) $x \in O : wholeO(x) \Rightarrow wholeC(typeO(x))$ (bzgl. eigenständiger Partikel)

(MComp2.2) $x \in L : compL(x) \Rightarrow compA(typeL(x))$ (bzgl. Komposition)

Erzwungene Eigentümer

(MComp2.3) $x \in O : \neg wholeO(x)$ (erzwingt Eigentümer
 $\Rightarrow \exists y \in L : targetL(y) = x \wedge compL(y)$ für Teil-Partikel)

Auch hier muss analog zum letzten Abschnitt untersucht werden, ob die Weglass-Konstruktion eindeutig ist und wie die Konstruktion in das restliche mathematische Modell integriert werden kann.

14.1.3 Mindestens-eins-Beschränkungen

Im vorgestellten Modell ist es nicht möglich, Mindestens-eins-Beschränkungen zu formulieren. In objektorientierten Modellen werden solche Beschränkungen häufig bei den Multiplizitäten an Assoziationsenden verwendet, um bereits im Schema gewisse Konsistenzbedingungen festzuhalten.¹²³ Eine andere Anwendung solcher Beschränkungen ist die Forderung nach standardisierten Ausprägungen von Klassen (sog. *Default-Objekte*); dadurch kann sichergestellt werden, dass für Mindestens-eins-Verknüpfungen immer ein Zielobjekt existiert.

Um solche Beschränkungen zu erlauben, kann dieselbe Grundidee wie bei abstrakten Klassen und Kompositionen wiederverwendet werden. Zwei Prädikate *optionalSourceA* und *optionalTargetA* markieren Assoziationen, deren Quelle bzw. Ziel optional ist, was im bisherigen Modell der einzige unterstützte Fall ist. Fehlt das eine oder andere Prädikat bei einer Assoziation, ist die jeweilige Quelle oder das jeweilige Ziel der Assoziation nicht optional und muss auf der Ausprägungsebene somit existieren. Dies wird durch entsprechende Existenzaussagen analog zu den letzten beiden Abschnitten erreicht. Die Prädikate *optionalSourceL* und *optionalTargetL* reflektieren die Prädikate *optionalSourceA* und *optionalTargetA* der Schema-Ebene, um Systeme in Schema und Ausprägung analog zur Spezifikation *MP* aus Abschnitt 9.1 aufspalten zu können.

¹²³ Ein Beispiel ist "Eine Gabel besitzt mindestens zwei Zinken".

$MOpt ::= M +$

prds

$optionalSourceA: A$	(Assoziation mit optionaler Quelle)
$optionalTargetA: A$	(Assoziation mit optionalem Ziel)
$optionalSourceL: L$	(Verknüpfung mit optionaler Quelle)
$optionalTargetL: L$	(Verknüpfung mit optionalem Ziel)

axms

Typisierung ist homomorph

(MOpt.1) $x \in L : optionalSourceL(x) \Rightarrow optionalSourceA(typeL(x))$ (bzgl. Mindestens-von-eins-Verknüpfungen)

(MOpt.2) $x \in L : optionalTargetL(x) \Rightarrow optionalTargetA(typeL(x))$ (bzgl. Mindestens-zu-eins-Verknüpfungen)

Mindestens-eins-Axiome

(MOpt.3) $x \in O, y \in A : typeO(x) = targetA(y) \wedge \neg optionalSourceA(y) \Rightarrow \exists y' \in L : typeL(y') = y \wedge targetL(y') = x$ (erzwingt Mindestens-von-eins-Assoziationen)

(MOpt.4) $x \in O, y \in A : typeO(x) = sourceA(y) \wedge \neg optionalTargetA(y) \Rightarrow \exists y' \in L : typeL(y') = y \wedge sourceL(y') = x$ (erzwingt Mindestens-zu-eins-Assoziationen)

Verglichen mit der Umsetzung von abstrakten Klassen und Kompositionen sind die Axiome komplexer, weil sie die Typisierungen in Anspruch nehmen. Somit wird die Integration der Konstruktion in das bisherige mathematische Modell erheblich aufwändiger sein, weil Schema und Ausprägung in der angenommenen kofreien Konstruktion in ihrer Gesamtheit betrachtet werden müssen. Auch muss dafür Sorge getragen werden, dass das Schema durch die Konstruktion nicht verändert wird, weil ansonsten die Weglass-Konstruktion nicht mehr eindeutig ist. Im obigen Beispiel kann bei einer Anpassung ansonsten ebensogut die jeweilige Assoziation im Schema statt des Objekts in der Ausprägung gelöscht werden, um die Axiome (MOpt.3) und (MOpt.4) zu erfüllen.

14.1.4 Vollständige Objekte

Eine Variante von Mindestens-eins-Einschränkungen liegt vor bei der Ausprägung von Vererbungsstrukturen. Während das präsentierte Modell verbietet, dass mehr als ein Partikel zu einer

Klasse innerhalb desselben Objekts existiert, ist es nicht möglich, die Existenz eines Partikels zu erzwingen. Somit kann es unvollständige Objekte geben, denen Werte und Verknüpfungen auf Grund nicht existierender Partikel fehlen.

Über eine entsprechende Weglass-Konstruktion können solche unvollständigen Objekte eliminiert werden:

$MINh ::= M +$
axms

Vererbungs-Axiome

(MINh.1) $x \in O, y \in C : \text{under}C(\text{type}O(x), y)$ (erzwingt komplette
 $\Rightarrow \exists y' \in O : \text{type}O(y') = y \wedge \text{under}O(x, y')$ Objekte)

Allerdings muss hier ebenfalls sichergestellt werden, dass das Schema durch die Konstruktion nicht verändert wird. Ansonsten kann bei einer Anpassung die Vererbungsrelation im Schema statt des Unterobjekts in der Ausprägung gelöscht werden, um das Axiom (MINh.1) zu erfüllen.

14.1.5 Zusammenhängende Objekte

Im Modell ist es möglich, dass die *relC*- und *relO*-Relationen größer sind als der transitive Abschluss der zugehörigen *underC*- bzw. *underO*-Relationen. Damit wird eine Partitionierung von Objekten in mehrere, voneinander getrennte Teile ermöglicht, so dass die Objekte nicht mehr zusammenhängend sind. Eine Beschränkung der *rel*-Relationen auf genau jene Relationen, die durch den transitiven Abschluss der *under*-Relationen entstehen, ist jedoch mit positiven Horn-Formeln vermutlich nicht möglich. Alternative Formen der Spezifikation und deren Interaktion mit dem restlichen mathematischen Modell sind vom Autor bisher nicht betrachtet worden, bieten jedoch einen Ansatzpunkt für künftige Forschungsaktivitäten.

14.1.6 Direkte vs. indirekte Vererbung

Das vorliegende Modell “vergisst” die erzeugenden Relationen der Vererbungsstrukturen auf Klassen und Objekten, lediglich der transitive Abschluss der Vererbungsstruktur wird behalten.¹²⁴ Dies unterscheidet sich von objektorientierten Programmiersprachen, in denen lediglich die direkten Vererbungsbeziehungen im Quelltext spezifiziert werden.¹²⁵ Dadurch ergibt sich der Nachteil, dass beim Löschen von Vererbungsbeziehungen im konzeptionellen Modell unter Umständen viel mehr

¹²⁴ Für Operationen und Nachrichten gilt dies ebenfalls.

¹²⁵ Zumindest ist dem Autor keine einzige objektorientierte Programmiersprache bekannt, in der nicht nur die direkten, sondern auch alle indirekten Oberklassen einer Klasse im Quelltext angegeben werden müssen.

getan werden muss, weil nicht nur die direkten, sondern auch alle indirekten Vererbungsbeziehungen berücksichtigt werden müssen.

Eine Möglichkeit, die direkte Vererbungsstruktur in Modell zu “behalten”, liegt in der Verwendung weiterer Prädikate (etwa $subC$ und $subO$). Die bestehenden Prädikate $underC$ und $underO$ können dann als transitiver Abschluss der sub -Prädikate definiert werden. Dies verkompliziert jedoch das Modell nicht unerheblich, ohne einen nennenswerten Gewinn mit sich zu bringen, weil nicht wenige Beweise um die Behandlung der neuen Prädikate erweitert werden müssen.

14.1.7 Typisierung von Daten in Software

Im präsentierten Modell ist es möglich, Objekte und Verknüpfungen in Operationen und Parametern zu typisieren. Das ist offensichtlich nicht besonders sinnvoll. Ein möglicher Ansatz, dies zu verbieten, liegt in der Forderung, dass Typisierungen auf dem Prädikat $message$ voll sind, was mit der folgenden Implikation formuliert werden kann:

$MType ::= M +$

axms

(MType.1) $x \in O : operation(typeO(x)) \Rightarrow message(x)$ (Typisierung ist voll auf Software)

Eine solche Implikation stellt jedoch beim Übergang von der Kategorie $\mathbf{Alg}(MType)$ zur Kategorie $\mathbf{Alg}(MP)^2$ ein Problem dar. Denn das Axiom (MType.1) spezifiziert eine Homomorphisms-Eigenschaft, die in der Spezifikation MP nicht formuliert werden kann. Daraus resultiert die Notwendigkeit, bei den in der Migration durchgeführten Konstruktionen den Nachweis zu erbringen, dass das Axiom (MType.1) nicht verletzt wird. Während dies für den Pullback-Funktor relativ einfach ist, weil Pullbacks volle Homomorphismen übertragen, ist dies für den Kompositionsfunktor und den Epirefektor nur unter Zuhilfenahme geeigneter Zusatzbedingungen möglich.

14.1.8 Adhäsive HLR-Kategorien

Programme werden im mathematischen Modell durch DPO-Regeln in einer geeigneten adhäsiven Kategorie modelliert, wobei diese Kategorie durch Vereinfachung der eigentlichen Modell-Kategorie entsteht. Dadurch ist es notwendig, zwischen den beiden Kategorien geeignet zu “übersetzen”, was einen nicht unerheblichen Beweisaufwand darstellt. Eine andere Möglichkeit ist, eine geeignete Unterklasse \mathcal{M} von Monomorphismen zu finden, für die $(\mathbf{Alg}(M), \mathcal{M})$ eine adhäsive HLR-Kategorie bildet.

Wie die Einführung zu Kapitel 10 demonstriert, sind Kategorien zu Signaturen mit Prädikaten nicht adhäsiv. Ähnliche Überlegungen führen zu der Einsicht, dass auch beliebige Axiome die

Adhäsions-Eigenschaften zerstören können. Eine zu klein gewählte Klasse von Monomorphismen kann wiederum die Menge der erlaubten DPO-Regeln zu sehr einschränken. Beispielsweise führt eine Beschränkung auf *reguläre* Monomorphismen¹²⁶ dazu, dass keine Verknüpfungen hinzugefügt werden können: Die Egalisator-Konstruktion führt zu dem Schluss, dass die Verknüpfung z durch zwei ausgehende Homomorphismen auf zwei unterschiedliche Verknüpfungen abgebildet werden müsste, damit sie im Quellsystem (= Egalisator-Objekt) weggelassen werden kann; dies ist jedoch auf Grund von Axiom (M.29) nicht möglich (Abb. 14.1).

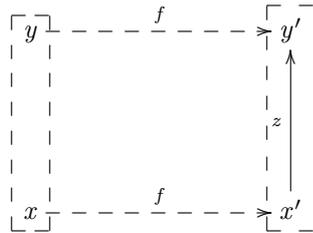


Abb. 14.1: Ein nicht regulärer Monomorphismus in $\mathbf{Alg}(MP)$

Eine Beschränkung auf volle Monomorphismen in implikativ spezifizierten Kategorien führt im Allgemeinen ebenfalls nicht immer zum Erfolg. Das folgende Gegenbeispiel zeigt, dass volle Monomorphismen nicht immer unter Pushouts stabil sind. Sei die folgende Spezifikation mit Gleichungen gegeben:

```

Bad ::=
  sorts
    S
    T
  axms
    x, y ∈ S, z ∈ T : x = y
    
```

Weil die Spezifikation *Bad* ohne Prädikate auskommt, ist jeder Monomorphismus voll. Das Diagramm in Abb. 14.2 zeigt ein kommutierendes Quadrat in $\mathbf{Alg}(Bad)$, wobei ausgefüllte Kreise Elemente zur Sorte *S* und nicht ausgefüllte Kreise Elemente zur Sorte *T* darstellen. Offensichtlich ist das Diagramm ein Pushout in $\mathbf{Alg}(Bad)$. Da f nicht injektiv ist, überträgt der Pushout jedoch die Injektivität von l nicht. Somit ist die Klasse \mathcal{M} aller vollen Monomorphismen in $\mathbf{Alg}(Bad)$ nicht unter Bildung von Pushouts abgeschlossen.

Eine Beschränkung auf volle Monomorphismen in der Kategorie $\mathbf{Alg}(M)$ sieht hingegen vielversprechend aus. Dazu müssen die folgenden Eigenschaften bewiesen werden:

- Es muss nachgewiesen werden, dass volle Monomorphismen in $\mathbf{Alg}(M)$ unter Pushouts stabil sind.

¹²⁶ Ein Monomorphismus ist *regulär*, wenn er Egalisator-Morphismus von zwei beliebigen Morphismen ist.

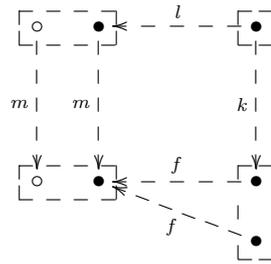


Abb. 14.2: Pushout in $\mathbf{Alg}(Bad)$ überträgt keine injektive Homomorphismen

- Es muss nachgewiesen werden, dass Pushouts entlang voller Monomorphismen Van-Kampen-Quadrate sind.

Beide Nachweise erfordern eine detaillierte Betrachtung der einzelnen M -Axiome und der Eigenschaften des Epirefektors $\mathcal{F}^{M''} : \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M'')$ aus Satz 9.15. Auch ist es erforderlich, die Konstruktion von Pushouts in $\mathbf{Alg}(M)$ entlang voller M -Homomorphismen auszuformulieren, weil die kanonische Konstruktion über Koprodukt und Koegalisateur schwierig zu handhaben ist, da die Konstruktion der Pushout-Trägermengen nicht komponentenweise erfolgen kann.

14.2 Transformations-Einschränkungen

In diesem Abschnitt werden Einschränkungen von Transformationen im mathematischen Modell beschrieben.

14.2.1 Symmetrie von Transformationen

Um Anomalien bei der Migration von Programmen wie in Beispiel 11.13 auszuschließen, ist es auf der rechten Seite einer Transformation verboten, Assoziationen zu identifizieren. Dadurch sind Transformationen aber nicht mehr symmetrisch: Es ist nicht mehr möglich, das Vervielfältigen von Assoziationen durch kompensierende Transformationen rückgängig zu machen. Doch ist dies häufig dann erforderlich, wenn vorhandene Abstraktionen wieder verworfen werden sollen. Ein Beispiel ist in Abb. 14.3 zu sehen. Dabei wird die Klassenhierarchie (Abb. 14.3a) in eine einzige Klasse transformiert. Idealerweise sollten die nun äquivalenten Assoziationen namens *kinder* identifiziert werden (Abb. 14.3b). Das ist jedoch nicht möglich, so dass die Assoziationen getrennt verbleiben müssen (Abb. 14.3c).

14.2.2 Verallgemeinerung von Daten-Parametern

Das mathematische Modell unterstützt nicht die Verallgemeinerung von Daten-Parametern. Die zugehörige konzeptionelle Transformation ist in Abb. 14.4 dargestellt. Der Grund für die

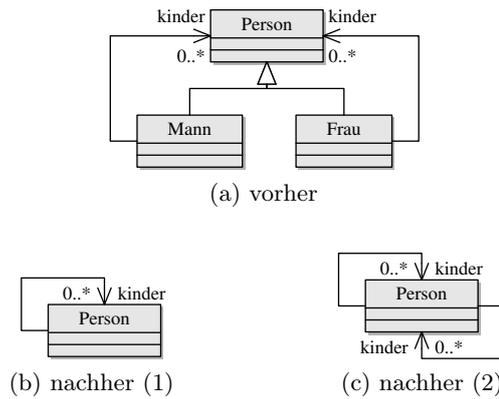


Abb. 14.3: Reduktion von Abstraktion

Unverträglichkeit mit dem mathematischen Modell ist, dass die Migration der Ausprägungs-Ebene alte Verweisobjekte löschen und neue Verweisobjekte erstellen muss, ohne dass im Schema die zugehörigen Klassen verändert werden. Im präsentierten mathematischen Modell können neue Objekte auf der Ausprägungs-Ebene jedoch nur entstehen, wenn sie aus existierenden Objekten entfaltet werden. Im vorliegenden Fall müsste das Verweisobjekt zur Unterklasse auf Schema-Ebene dupliziert werden, damit entsprechende Verweisobjekte auf der Ausprägungs-Ebene entstehen. Dies würde jedoch zu einer Vervielfachung *aller* entsprechend typisierter Verweisobjekte führen und nicht nur jener, auf die über Argumente von Nachrichten zur betrachteten Operation verwiesen wird.

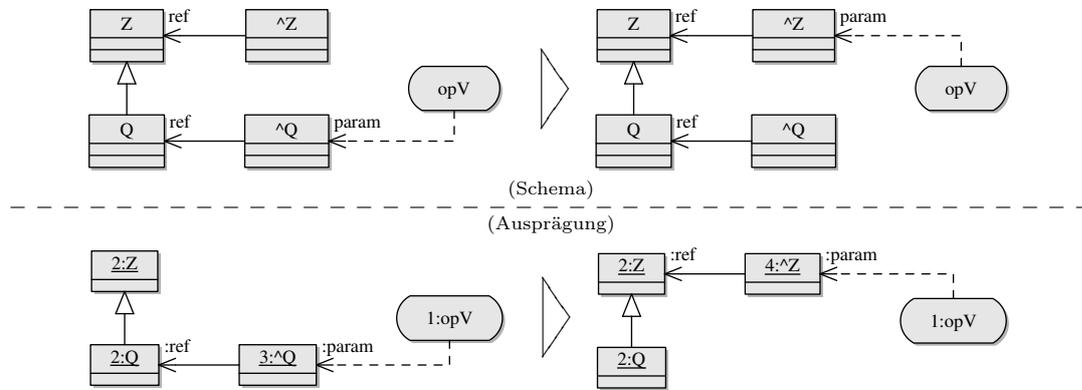


Abb. 14.4: Verschieben des Endes eines Parameters zu einer Oberklasse

Es ist somit eine Kontext-sensitive Ersetzung von Teilen der Ausprägungs-Ebene nötig, wobei der Kontext nicht allein über Schema-Restrukturierungen, sondern durch bestimmte "Muster" auf der Ausprägungs-Ebene spezifiziert wird. Eine Möglichkeit, dieses Problem zu lösen, ist die Anwendung von DPO-Transformationen, wie sie auch in der Spezifikation von Programmen im mathematischen Modell verwendet wird. Die Vorgehensweise sieht wie folgt aus:

- (1) Erstellung eines neuen Parameters $param'$ von der Operation opV zur Verweisklasse $\uparrow Z$ durch Anwendung der Schema-Transformation “Erzeugen eines Parameters” (12.3.5.1).
- (2) Anwendung einer DPO-Regel auf die Ausprägungs-Ebene, die für jedes $\uparrow Q$ -Objekt, das Ziel eines $param$ -Arguments ist, ein $\uparrow Z$ -Objekt samt ausgehender ref -Verknüpfung sowie ein $param'$ -Argument mit derselben Quelle wie das $param$ -Argument erzeugt. Weiter wird das $\uparrow Q$ -Objekt in derselben DPO-Regel samt ausgehender ref -Verknüpfung gelöscht. Die DPO-Regel ergibt sich direkt aus der Ausprägungs-Migration in Abb. 14.4, mit der Änderung, dass das neue Argument auf der rechten Seite in dem Parameter $param'$ typisiert ist. Der Klebgraph der DPO-Regel entsteht aus dem Schnitt der linken und rechten Seite.
- (3) Löschung des alten Parameters $param$ durch Anwendung der Schema-Transformation “Löschen eines Parameters” (12.3.5.4).

Es bleibt zu untersuchen, inwiefern die Anwendung der DPO-Regel die Axiome bewahrt. Ein Problem mit dem beschriebenen Ansatz ist ferner, dass beim DPO-Modell das gelöschte $\uparrow Q$ -Objekt keine weiteren eingehenden Verknüpfungen und Argumente besitzen darf, weil ansonsten die DPO-Regel auf Grund der nicht erfüllten Klebebedingung nicht angewandt werden kann. Hier sind andere Modelle wie der SPO-Ansatz¹²⁷ zu evaluieren. Alternativ ist es möglich, das $\uparrow Q$ -Objekt gar nicht zu löschen; in diesem Fall ist man auch in jenem Fall auf der sicheren Seite, wenn das Verweisobjekt noch in einem anderen Kontext benutzt wird.

14.2.3 Mehrdeutigkeit nach Prozess-Transformation

Satz 11.34 sagt aus, dass Programme zum alten Schema nach einer Schema-Transformation gültige Programme zum neuen Schema sind. Dies bedeutet jedoch nicht, dass eine erneute Ausführung eines *bereits abgelaufenen Prozesses* zu einem transformierten Programm in einem Prozess resultiert, der semantisch äquivalent zum alten Prozess ist, der im alten Schema typisiert ist. Dies liegt an der Annahme 10.34, die durch Programm-Transformationen nicht notwendigerweise erhalten wird. Deutlich wird dies, wenn eine Programm-Transformationen zwei Klassen zusammenlegt, die beide dieselbe Operation implementieren: Nach der Zusammenlegung existieren zwei Methoden zu derselben Operation *in derselben Klasse*. Somit ist die Annahme 10.34 nach der Transformation verletzt, was zu einer Mehrdeutigkeit bei der Auswahl einer Methode zu einer Nachricht führen kann (5.2.1.7). Aus diesem Grund ist das Zusammenlegen von Klassen nur dann ein Refactoring, wenn alle Methoden der Klassen zu unterschiedlichen Operationen gehören (vgl. die Voraussetzungen der Transformationen, die in den Abschnitten 12.3.1.4 und 12.3.1.5 beschrieben werden).

¹²⁷ SPO steht für *Single-Pushout* und beschreibt einen Ansatz, der Löschungen in Graphen auch in unbekanntem Kontext erlaubt; vgl [71, Kapitel 4].

14.2.4 Erhaltung der Semantik von Programmen

Theorem 11.35 stellt sicher, dass eine Methode, die vor der Migration eines Programms anwendbar ist, auch nach der Migration anwendbar ist. Dies garantiert aber nicht, dass die *Ergebnisse* des Programms erhalten bleiben. Dies ist bei allgemeinen Transformationen auch nicht der Fall. Ein Beispiel ist die Schema-Transformation, die das gesamte Schema und somit auch alle Daten, Prozesse und Programme löscht. Offensichtlich sind Methoden, die vor der Migration anwendbar waren, nach der Migration weiterhin anwendbar, denn die leere Methode ist über den leeren Ansatz in dem leeren Prozess anwendbar; allerdings sind die Resultate völlig unterschiedlich.

Es bleibt somit zu untersuchen, für welche Schema-Transformation eine Semantikerhaltung gegeben ist. Dazu ist im ersten Schritt eine geeignete formale Definition der (operationalen) Semantik von Programmen in dem präsentierten Modell zu verankern. Im zweiten Schritt ist die Klasse der Refactorings auf Semantikerhaltung zu untersuchen, da nach Definition genau diese Schema-Transformationen die Semantik von Programmen erhalten. Im mathematischen Modell entsprechen Refactorings nicht-löschenden Schema-Transformationen, deren linke Seite somit surjektiv ist. Es ist somit zu prüfen, ob die Surjektivität auf der linken Seite für die Definition von Refactorings ausreichend ist.

14.2.5 Daten-getriebene Schema-Transformationen

Das präsentierte Modell unterstützt die automatische Migration der Ausprägungs-Ebene auf Grundlage der gewünschten Schema-Transformation. Bisher wurde der Prozess nur in einer Richtung betrachtet, nämlich vom Schema zur Ausprägung aus. Es wurde also zuerst eine gewünschte Schema-Transformation formuliert und dann untersucht, wie diese Veränderung auf der Ausprägungsebene fortgesetzt werden kann. Diese Sichtweise wird im Rahmen dieser Arbeit durchgängig angewandt.

In der Praxis ist es jedoch häufig notwendig, den umgekehrten Weg einzuschlagen: Die Ausprägungsebene soll nach bestimmten Regeln umgebaut werden, ohne dass die benötigte Schema-Transformation von vornherein bekannt ist. Die Aufgabe lautet hier also, zu einer gewünschten Migration die passende Schema-Transformation zu finden, so dass die Migration im Rahmen des mathematischen Modells beschrieben werden kann. Für viele solcher Umbau-Maßnahmen existiert im präsentierten Modell sicherlich keine geeignete Schema-Transformation, insbesondere dann nicht, wenn die gewünschte Migration wenig strukturiert ist. Es ist jedoch interessant festzustellen, dass es gewisse Migrationen gibt, die sehr strukturiert sind, aber dennoch nicht im mathematischen Modell als induzierte Migration einer bestimmten Schema-Transformation beschrieben werden können. Ein solcher Fall wird im folgenden Beispiel erläutert.

Beispiel 14.1. Es sei eine Klasse *Person* wie in Abb. 14.5a gegeben, bei der das Geschlecht einer Person durch ein Boole'sches Attribut *istMaennlich* bestimmt wird. Ziel ist es, dieses Attribut durch

zwei Unterklassen *Mann* und *Frau* zu ersetzen (Abb. 14.5b).¹²⁸ Diese Migration ist strukturiert, und für jedes Objekt kann problemlos bestimmt werden, wie es umzuformen ist, indem einfach das passende Unterobjekt hinzugefügt und der Attributwert zum Attribut *istMännlich* gelöscht wird. Es lässt sich im präsentierten Modell jedoch keine Schema-Transformation formulieren, aus der diese Migration abgeleitet werden kann. Denn die konkrete Migration eines Objekts hängt von der Belegung des Attributs *istMaennlich* in diesem Objekt ab. Diese Daten-Abhängigkeit lässt sich jedoch nicht in der Beschreibung einer Schema-Transformation nutzen. \square



Abb. 14.5: Umwandlung eines Attributs in Unterklassen

Dieses Beispiel verleitet dazu, über eine Erweiterung der Beschreibung von Schema-Transformationen im konzeptionellen und mathematischen Modell nachzudenken, welche die Formulierung von Daten-Abhängigkeiten in Schema-Transformationen ermöglicht. [34] untersucht dieses Problem unter dem Namen “Modell-spezifische Migration” ebenfalls und schlägt zwei Lösungswege vor, von denen der erste auf Interaktion mit dem Benutzer abzielt und der zweite auf der Ableitung impliziter Informationen aus dem Modell beruht. Eine mögliche Formalisierung dieser Ansätze in dem mathematischen Modell dieser Arbeit steht noch aus.

14.2.6 Formalisierung von Refactorings

Im Rahmen dieser Arbeit wurde die charakteristische Eigenschaft von Refactorings, nämlich das Erhalten von Information und Verhalten, nur intuitiv definiert. Für die Erhaltung von Information wurde in Abschnitt 4.5.2 ein Ansatz zur Formalisierung kurz angerissen, der hier etwas ausführlicher vorgestellt werden soll. In [57–59] wird das Beibehalten des Informationsgehalts einer Datenbank durch Schema-Transformationen über die *Informationskapazität* der Schemata formalisiert. Die Informationskapazität eines Schemas S entspricht dabei der Menge $I(S)$ aller möglichen Ausprägungen zu diesem Schema und repräsentiert somit den Informationsgehalt aller potentiellen Datenbanken zu diesem Schema. Weiterhin werden Abbildungen zwischen verschiedenen Schemata samt den induzierten Abbildungen auf den Schema-Ausprägungen auf die Erhaltung, Erweiterung und Reduzierung der Informationskapazität untersucht. In der Arbeit wird argumentiert, dass eine Schema-Transformation genau dann die Informationskapazität erhält, wenn die induzierte Migration total und injektiv ist, d. h. wenn zu jeder transformierten

¹²⁸ vgl. das Refactoring “Typenschlüssel durch Unterklassen ersetzen” in [27, S. 227]

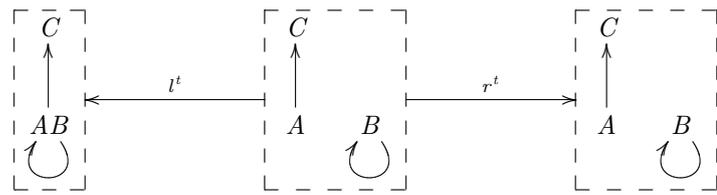
Ausprägung zum Ziel-Schema eine eindeutige Ausprägung zum Quell-Schema existiert. Dies garantiert, dass jede Abfrage auf einer transformierten Datenbank in eine Abfrage auf einer eindeutig bestimmten ursprünglichen Datenbank umgewandelt werden kann [57, Abschnitt 3].

Diese Konzepte lassen sich relativ einfach auf das mathematische Modell dieser Arbeit anwenden. Offensichtlich entspricht die intuitive Definition eines Refactorings der Beibehaltung der Informationskapazität des Ausgangsschemas. Eine Schema-Transformation t ist demzufolge genau dann ein Refactoring, wenn der Migrationsfunktorkomplex \mathcal{M}^t aus Def. 11.11 injektiv ist. Diese Definition stimmt jedoch mit der Definition 11.2 eines Refactorings nicht überein, denn es gibt Schema-Transformationen, deren linke Seite surjektiv ist und die dennoch zu induzierten Migrationen führen, die unterschiedliche Ausgangssysteme identifizieren. Das folgende Gegenbeispiel ist [43, Beispiel 3] entnommen.

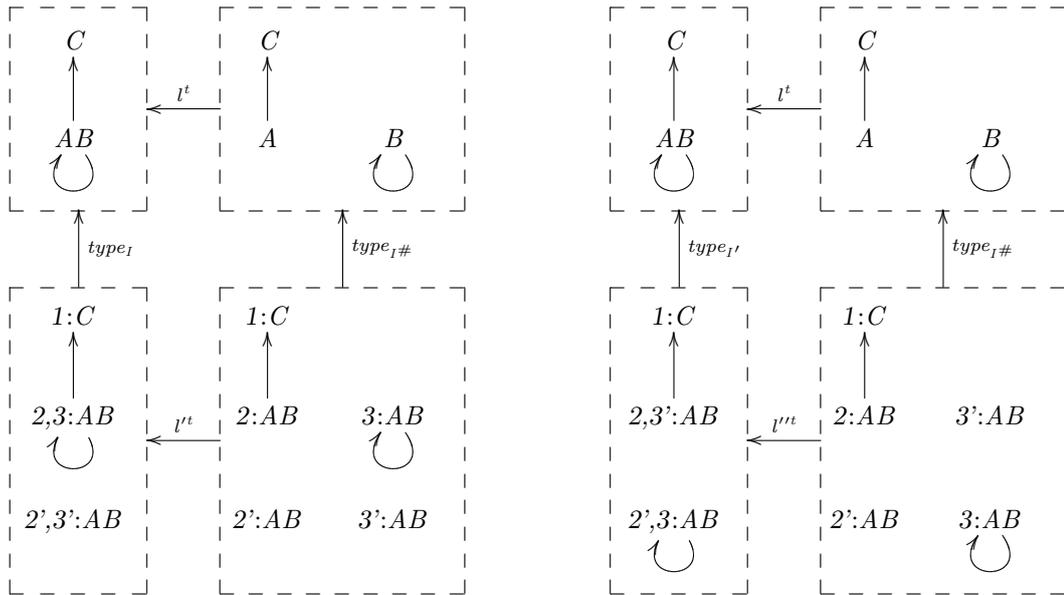
Beispiel 14.2 (Nicht injektiver Migrationsfunktorkomplex). Das folgende Beispiel demonstriert, dass die Migration zweier nicht isomorpher Ausgangssysteme dasselbe System ergeben kann, auch wenn die linke Seite einer Schema-Transformation surjektiv ist. In Abb. 14.6a ist die Transformation dargestellt, die eine Klasse in zwei Teile spaltet und zusätzlich zwei mit der ursprünglichen Klasse verbundene Assoziationen auf die beiden Teile verteilt. In Abb. 14.6b und Abb. 14.6c ist dargestellt, dass die Anwendung des Pullback-Funktorkomplexes auf zwei nicht isomorphe Systeme I und I' in demselben System $I^\#$ resultiert. \square

Surjektivität auf der linken Seite einer Schema-Transformation garantiert somit nicht die Erhaltung der Informationskapazität nach [57–59]. Für eine gegebene Schema-Transformation $t: S \xrightarrow{S^\#} S'$ müssen somit folgende Punkte untersucht werden:

- Unter welchen Bedingungen ist der Pullback-Funktorkomplex \mathcal{P}^{l^t} injektiv? Der Autor vermutet, dass es ausreichend ist, neben der Surjektivität von l^t zu fordern, dass alle Elemente, die durch die linke Seite eines Refactorings identifiziert werden, in irgendeiner Weise miteinander verbunden sind (etwa über die *relC*- oder *underC*-Prädikate).
- Unter welchen Bedingungen ist der gesamte Migrationsfunktorkomplex \mathcal{M}^t injektiv? Eine Untersuchung der einzelnen Funktorkomplexe auf Injektivität ist nicht ausreichend, weil der Epirefektorkomplex $\mathcal{F}^{S'}$ in der Regel nicht injektiv ist. Hier muss die Interaktion zwischen dem Pullback-Funktorkomplex \mathcal{P}^{l^t} auf der linken Seite und dem Kompositionsfunktorkomplex \mathcal{F}^{r^t} und dem Epirefektorkomplex $\mathcal{F}^{S'}$ auf der rechten Seite berücksichtigt werden. Der Autor vermutet, dass der Epirefektorkomplex $\mathcal{F}^{S'}$ zwei Ausprägungs-Elemente genau dann identifizieren darf, wenn mindestens eines dieser Elemente durch “Auseinanderziehen” auf der linken Seite der Migration entstanden ist.



(a) Transformation



(b) Erstes Pullback-Diagramm

(c) Zweites Pullback-Diagramm

Abb. 14.6: Unterschiedliche Systeme werden in dasselbe System migriert

Fazit

Die vorliegende Arbeit stellt einen Ansatz vor, objektorientierte Systeme – bestehend aus Daten, Programmen und Prozessen zu einem einheitlichen Schema – ganzheitlich zu beschreiben und Transformationen solcher Systeme zu formalisieren, wobei die Migration der Ausprägung aus der Transformation des Schemas eindeutig abgeleitet wird. Die zentralen objektorientierten Konzepte werden unterstützt. Das konzeptionelle Modell ist wenig einschränkend, so dass die Hoffnung besteht, die meisten semantischen Konstrukte aktueller Programmiersprachen abbilden zu können. Die Fundierung des Ansatzes in universeller Algebra und Kategorientheorie erlaubt zum einen formale Korrektheitsbeweise. Zum anderen bietet sie vielfältige Anknüpfungspunkte, die zur Lösung des einen oder anderen im Anhang aufgezeigten Problems aus Kapitel 14 hilfreich sein können.

In der Theorie hat sich der Ansatz bislang bewährt. Die Validierung des Ansatzes in der Praxis unter realistischen Bedingungen steht jedoch noch aus. Hier stehen drei größere Arbeiten an. Zum einen müssen die vorgestellten Konstruktionen effizient implementiert werden, so dass die Arbeit mit objektorientierten Systemen praxistauglich ist. Dies ist um so wichtiger, als gerade große Systeme mit einer enormen Datenmenge, vielen damit arbeitenden Programmen und vielen laufenden Prozessen von Refactorings profitieren, die den ordnungsgemäßen Ablauf des Systems nicht stören.

Zum zweiten muss ein geeignetes Werkzeug entwickelt werden, das es erlaubt, Schema-Transformationen im Allgemeinen und Schema-Refactorings im Besonderen durchzuführen. Hierzu ist zum einen eine geeignete Visualisierung der Systeme zu überlegen, weil die Systeme in der Praxis sehr groß sind. Zum anderen ist zu untersuchen, zu welchem Zeitpunkt die Migration durchgeführt wird. Die vorgestellten Konstruktionen legen nahe, die Migration der gesamten Ausprägung zum Zeitpunkt der Schema-Transformation vollständig durchzuführen. Es bleibt zu analysieren, ob und wie die in der Einleitung angeschnittene On-Demand-Migration in dem vorgestellten Modell umgesetzt werden kann.

Schließlich muss über eine Integration in bestehende Systeme nachgedacht werden. Hierbei ist der erste Schritt die Abbildung einer konkreten und verbreiteten Programmiersprache wie Java

oder C++ oder einer entsprechenden Teilmenge hiervon ins konzeptionelle Modell. Im zweiten Schritt muss untersucht werden, wie eine Integration bestehender Datenbestände, Programme und Prozesse in die Strukturen des Modells umgesetzt werden kann, wobei in allen Fällen eine Zwei-Wege-Synchronisation notwendig sein wird, so lange die Daten und Programme außerhalb des Systems genutzt und gewartet werden. Bei Programmen kommt noch die Schwierigkeit hinzu, den Abgleich zwischen der Graph-Repräsentation von Programmen im vorgestellten Modell und der textuellen Repräsentation der Programme in der jeweiligen Programmiersprache durchzuführen. Denn eine Entwicklung von Programmen in der Graph-Repräsentation wird auch mit entsprechender Werkzeug-Unterstützung voraussichtlich weniger produktiv sein als herkömmliche, auf Quelltext beruhende Entwicklungsprozesse. Hier sind geeignete Werkzeuge zu entwickeln, die idealerweise in die verwendeten Entwicklungsumgebungen eingebettet sind. Auch bei Prozessen ist ein Wechsel der Repräsentation notwendig, etwa bei der Fehlersuche, so dass auch hier entsprechende Werkzeuge bereitgestellt werden müssen.

Auch auf der theoretischen Ebene bietet das Modell neben den im letzten Teil dargestellten Einschränkungen und Erweiterungen Ansätze für künftige Forschungsaktivitäten. Wie in [52] erwähnt wäre es von Vorteil, atomare Transformations-Bausteine zu bestimmen, aus denen alle anderen Transformationen durch Komposition gebildet werden können. Diese Bausteine hätten Schema-Charakter und könnten beispielsweise durch Graph-Transformationen auf das jeweilige konkrete Schema angewandt werden. Durch diese Indirektion wäre man in der Lage, Schema-Transformationen in Regel und Ansatz zu trennen, was in dem vorgestellten Modell noch nicht möglich ist, da die Schema-Transformationen über konkrete Spans auf dem jeweils betrachteten Schema formuliert werden müssen. Ein weiteres Problem ist die Unterstützung von On-Demand-Migration von Daten, Programmen und Prozessen. Hier muss untersucht werden, ob die beschriebenen Konstruktionen auch partiell durchgeführt werden können.

Der Autor hat die Hoffnung, dass die vorliegende Arbeit ihre Nützlichkeit in der Praxis beweisen wird und dass die beschriebenen Schwachpunkte in der Zukunft behoben werden können.

Teil V

Anhang

Mehrsortige algebraische Systeme

In diesem Kapitel sind alle Definitionen und Sätze aus der Theorie mehrsortiger algebraischer Systeme zu finden, die in dieser Arbeit benötigt werden. Es orientiert sich im Großen und Ganzen an [77]. Beweise und erläuternde Bemerkungen sind jedoch aus Platzgründen weggelassen worden.

A.1 Signaturen und Systeme

Definition A.1 (Signatur, algebraische Signatur). Eine Signatur Σ ist ein Tripel $(S, OP = (OP_{w,s})_{w \in S^*, s \in S}, P = (P_w)_{w \in S^*})$ mit

- einer Menge S von Sorten,
- einer Familie $(OP_{w,s})_{w \in S^*, s \in S}$ von Operationssymbolen und
- einer Familie $(P_w)_{w \in S^*}$ von Relationssymbolen oder Prädikaten,

wobei $=_s \in P_{s,s}$ für jede Sorte $s \in S$. Dabei bezeichnet S^* die Menge aller Wörter über den Sorten in S , einschließlich des leeren Wortes ε . Operationssymbole $c \in OP_{\varepsilon,s}$ für jede Sorte $s \in S$ werden auch Konstantensymbole genannt.

Besteht eine Signatur nur aus Sorten und Operationssymbolen, ist die Familie von Prädikaten P also leer, wird die Signatur algebraisch genannt.

Definition A.2 (System, Algebra). Sei Σ eine Signatur. Dann ist

$$A = ((A_s)_{s \in S}, (op^A: A_w \rightarrow A_s)_{w \in S^*, s \in S, op \in OP_{w,s}}, (p^A \subseteq A_w)_{w \in S^*, p \in P_w})$$

ein System zur Signatur Σ oder kurz Σ -System, wenn

- A_s eine Menge zu jeder Sorte $s \in S$, genannt Trägermenge,
- $op^A: A_w \rightarrow A_s$ eine Abbildung zu jedem Operationssymbol $op \in OP_{w,s}$, genannt Operation,

- $p^A \subseteq A_w$ eine Menge zu jedem Prädikat $p \in P_w$, genannt Relation, und
- $=_s^A$ die identische Relation $Id_{A_s} = \{(x, x) \mid x \in A_s\}$ für jede Sorte $s \in S$ ist.

Dabei ist die Notation A_w für ein Sorten-Wort $w = s_1 s_2 \dots s_n$, $n > 0$ eine abkürzende Schreibweise für das Produkt $A_{s_1} \times A_{s_2} \times \dots \times A_{s_n}$; falls w das leere Wort ε darstellt, gilt $A_\varepsilon = \{*\}$, so dass eine Abbildung $f: A_\varepsilon \rightarrow A_s$ genau einen Wert im Bildbereich auswählt.

Ein System zu einer algebraischen Signatur wird Algebra genannt.

Definition A.3 (Homomorphismus). Sei Σ eine Signatur und seien A, B zwei Σ -Systeme. Dann ist eine Sorten-indizierte Familie von Abbildungen $h: A \rightarrow B = (h_s: A_s \rightarrow B_s)_{s \in S}$ ein Σ -Homomorphismus, wenn

$$h_s \circ op^A = op^B \circ h_w$$

für alle $op \in OP_{w,s}$, $w \in S^*$ und $s \in S$ sowie

$$h_w(p^A) \subseteq p^B$$

für alle $p \in P_w$ und $w \in S^*$ gilt. Dabei bezeichnet die Notation h_w das Produkt $h_{s_1} \times h_{s_2} \times \dots \times h_{s_n}$ für $w = s_1 s_2 \dots s_n$, $n > 0$ und die identische Abbildung $\{*\} \rightarrow \{*\}$ für $w = \varepsilon$.

Definition A.4 (Kern). Sei Σ eine Signatur und sei $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann ist der Kern von h , $\ker h = (\ker_p^h \subseteq A_w)_{w \in S^*, p \in P_w}$ eine Familie von Relationen auf den Trägermengen von A , definiert durch:

$$x \in \ker_p^h \iff h_w(x) \in p^B$$

für alle $w \in S^*$, $p \in P_w$ und $x \in A_w$. Aus der Definition eines Homomorphismus' folgt:

$$p^A \subseteq \ker_p^h$$

Definition A.5 (Komposition von Homomorphismen). Sei Σ eine Signatur und seien $f: A \rightarrow B$ und $g: B \rightarrow C$ zwei Σ -Homomorphismen. Dann wird die Sorten-indizierte Familie von Abbildungen $((g \circ f)_s: A_s \rightarrow C_s)_{s \in S}$ definiert durch

$$(g \circ f)_s(x) = g_s(f_s(x))$$

für alle $x \in A_s$ und alle $s \in S$.

Lemma A.6 (Komposition von Homomorphismen). Die Sorten-indizierte Familie von Abbildungen $((g \circ f)_s: A_s \rightarrow C_s)_{s \in S}$ aus Def. A.5 ist ein Σ -Homomorphismus $g \circ f: A \rightarrow C$. \square

Definition A.7 (Identischer Homomorphismus). Sei $\Sigma = (S, OP, P)$ eine Signatur. Dann wird für jedes Σ -System A die Sorten-indizierte Familie von Abbildungen $(id_s^A: A_s \rightarrow A_s)_{s \in S}$ durch $id_s^A(x) = x$ für alle Sorten $s \in S$ und alle Elemente $x \in A_s$ definiert.

Lemma A.8 (Identischer Homomorphismus). Die Sorten-indizierte Familie von Abbildungen $(id_s^A: A_s \rightarrow A_s)_{s \in S}$ aus Def. A.7 ist ein Σ -Homomorphismus $id^A: A \rightarrow A$, genannt identischer Homomorphismus auf A . \square

Definition A.9 (Kategorie von Systemen). Sei Σ eine Signatur. Dann bezeichnet $\mathbf{Alg}(\Sigma)$ die folgende Kategorie:

- $\mathbf{Ob}^{\mathbf{Alg}(\Sigma)}$ besteht aus allen Σ -Systemen;
- $\mathbf{Mor}^{\mathbf{Alg}(\Sigma)} = (\mathbf{Mor}^{\mathbf{Alg}(\Sigma)}(A, B))_{A, B \in \mathbf{Ob}^{\mathbf{Alg}(\Sigma)}}$ besteht aus der Menge aller Σ -Homomorphismen zwischen je zwei Σ -Systemen A und B ;
- die Komposition $\circ^{\mathbf{Alg}(\Sigma)}$ ist die Komposition von Homomorphismen gemäß Def. A.5; und
- die Identitäten $id^{\mathbf{Alg}(\Sigma)} = (id_A^{\mathbf{Alg}(\Sigma)} \in \mathbf{Mor}^{\mathbf{Alg}(\Sigma)}(A, A))_{A \in \mathbf{Ob}^{\mathbf{Alg}(\Sigma)}}$ sind die identischen Homomorphismen gemäß Def. A.7.

A.2 Terme und Auswertung

Definition A.10 (Termssystem). Sei Σ eine Signatur. Dann wird T^Σ , das Termssystem zur Signatur Σ , folgendermaßen definiert:

- T_s^Σ ist die Menge aller Grundterme zur Sorte s und wird wie folgt induktiv definiert:
 - $c \in T_s^\Sigma$ für alle $c \in OP_{\varepsilon, s}$
 - $f(t) \in T_s^\Sigma$ für alle $f \in OP_{w, s}$, $w \in S^* \neq \varepsilon$ und $t \in T_w^\Sigma$
- $c^{T^\Sigma}(*) ::= c$ für alle $c \in OP_{\varepsilon, s}$
- $f^{T^\Sigma}(t) ::= f(t)$ für alle $f \in OP_{w, s}$, $w \in S^* \neq \varepsilon$ und $t \in T_w^\Sigma$
- $p^{T^\Sigma} ::= \begin{cases} Id_{T_s^\Sigma} & \text{falls } p = =_s \text{ für ein } s \in S \\ \emptyset & \text{sonst} \end{cases}$ für alle $p \in P_w$ und $w \in S^*$

Definition A.11 (Auswertung von Grundtermen). Sei Σ eine Signatur und A ein Σ -System. Dann ist die Sorten-indizierte Familie von Abbildungen $(eval_s^A: T_s^\Sigma \rightarrow A_s)_{s \in S}$, definiert für alle Sorten $s \in S$ durch

$$\begin{aligned} eval_s^A(c) &::= c^A(*) && \text{für } c \in OP_{\varepsilon, s} \\ eval_s^A(f(t)) &::= f^A(eval_w^A(t)) && \text{für } f \in OP_{w, s}, w \in S^* \neq \varepsilon \text{ und } t \in T_w^\Sigma \end{aligned}$$

die Auswertung der Grundterme zur Signatur Σ in A .

Lemma A.12 (Auswertung von Grundtermen). Die Sorten-indizierte Familie von Abbildungen $(eval_s^A: T_s^\Sigma \rightarrow A_s)_{s \in S}$ aus Def. A.11 ist ein Σ -Homomorphismus $eval^A: T^\Sigma \rightarrow A$. \square

Lemma A.13 (Eindeutige Auswertung von Grundtermen). Seien Σ eine Signatur und A ein Σ -System. Dann gibt es genau einen Σ -Homomorphismus $eval^A: T^\Sigma \rightarrow A$. \square

Definition A.14 (Variablenvorrat). Sei Σ eine Signatur. Dann ist jede Sorten-indizierte Familie von Variablenmengen $X = (X_s)_{s \in S}$ ein Σ -Variablenvorrat.

Definition A.15 (Variablenbelegung). Seien Σ eine Signatur, A ein Σ -System und X ein Σ -Variablenvorrat. Dann ist jede Sorten-indizierte Familie von Abbildungen $asg: X \rightarrow A = (asg_s: X_s \rightarrow A_s)_{s \in S}$ eine Variablenbelegung in A .

Definition A.16 (Termssystem mit Variablen). Seien Σ eine Signatur und X ein Σ -Variablenvorrat. Dann wird $T^\Sigma(X)$, das Termssystem zur Signatur Σ mit Variablen X , folgendermaßen definiert:

- $T^\Sigma(X)_s$ ist die Menge aller Terme mit Variablen X zur Sorte s und wird wie folgt induktiv definiert:
 - $x \in T^\Sigma(X)_s$ für alle $x \in X_s$
 - $c \in T^\Sigma(X)_s$ für alle $c \in OP_{\varepsilon, s}$
 - $f(t) \in T^\Sigma(X)_s$ für alle $f \in OP_{w, s}$, $w \in S^* \neq \varepsilon$ und $t \in T^\Sigma(X)_w$
- $c^{T^\Sigma(X)}(*) ::= c$ für alle $c \in OP_{\varepsilon, s}$
- $f^{T^\Sigma(X)}(t) ::= f(t)$ für alle $f \in OP_{w, s}$, $w \in S^* \neq \varepsilon$ und $t \in T^\Sigma(X)_w$
- $p^{T^\Sigma(X)} ::= \begin{cases} Id_{T^\Sigma(X)_s} & \text{falls } p = =_s \text{ für ein } s \in S \\ \emptyset & \text{sonst} \end{cases}$ für alle $p \in P_w$ und $w \in S^*$

Definition A.17 (Auswertung von Termen mit Variablen). Seien Σ eine Signatur, X ein Σ -Variablenvorrat, A ein Σ -System und $asg: X \rightarrow A$ eine Variablenbelegung in A . Dann ist die Sorten-indizierte Familie von Abbildungen $(eval(asg)_s: T^\Sigma(X)_s \rightarrow A_s)_{s \in S}$, definiert für alle Sorten $s \in S$ durch

$$\begin{aligned} eval(asg)_s(x) &::= asg_s(x) && \text{für } x \in X_s \\ eval(asg)_s(c) &::= c^A(*) && \text{für } c \in OP_{\varepsilon, s} \\ eval(asg)_s(f(t)) &::= f^A(eval(asg)_w(t)) && \text{für } f \in OP_{w, s}, w \in S^* \neq \varepsilon \text{ und } t \in T^\Sigma(X)_w \end{aligned}$$

die Auswertung der Terme mit Variablen X zur Signatur Σ in A .

Lemma A.18 (Auswertung von Termen mit Variablen). Die Sorten-indizierte Familie von Abbildungen $(eval(asg)_s: T^\Sigma(X)_s \rightarrow A_s)_{s \in S}$ aus Def. A.17 ist ein Σ -Homomorphismus $eval(asg): T^\Sigma(X) \rightarrow A$. \square

Lemma A.19 (Eindeutige Auswertung von Termen mit Variablen). *Seien Σ eine Signatur, X ein Σ -Variablenvorrat, A ein Σ -System und $f, g: T^\Sigma(X) \rightarrow A$ zwei Σ -Homomorphismen, die auf allen Variablen übereinstimmen, d. h. es gilt*

$$(A.1) \quad f_s(x) = g_s(x)$$

für alle $x \in X_s$ und alle $s \in S$. Dann gilt $f = g$. □

A.3 Spezielle Homomorphismen und Systeme

Definition A.20 (Injektive, surjektive und bijektive Homomorphismen). *Sei eine Signatur $\Sigma = (S, OP, P)$ gegeben, und sei $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann ist h injektiv (surjektiv, bijektiv), wenn für jede Sorte $s \in S$ die Abbildung $h_s: A_s \rightarrow B_s$ injektiv (surjektiv, bijektiv) ist.*

Definition A.21 (Voller Homomorphismus). *Sei $\Sigma = (S, OP, P)$ eine Signatur, sei $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B und sei $p \in P_w$ ein Prädikat zu einem Sortenwort $w \in S^*$. Dann ist h voll auf dem Prädikat p , wenn*

$$(A.2) \quad h_w(x) \in p^B \Rightarrow \exists y \in A_w : h_w(x) = h_w(y) \wedge y \in p^A$$

für alle $x \in A_w$ gilt. h ist voll, wenn h voll auf jedem Prädikat ist.

Definition A.22 (Strikt voller Homomorphismus). *Sei $\Sigma = (S, OP, P)$ eine Signatur, sei $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B und sei $p \in P_w$ ein Prädikat zu einem Sortenwort $w \in S^*$. Dann ist h strikt voll auf dem Prädikat p , wenn*

$$(A.3) \quad h_w(x) \in p^B \Rightarrow x \in p^A$$

für alle $x \in A_w$ gilt. h ist strikt voll, wenn h strikt voll auf jedem Prädikat ist.

Lemma A.23 (Strikt volle Homomorphismen). *Seien $\Sigma = (S, OP, P)$ eine Signatur und $h: A \rightarrow B$ ein strikt voller Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann ist h voll. □*

Lemma A.24 (Volle injektive Homomorphismen). *Seien $\Sigma = (S, OP, P)$ eine Signatur und $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann gilt: h ist genau dann strikt voll, wenn h voll und injektiv ist. □*

Definition A.25 (Gemeinsam strikt volle Homomorphismen). *Sei $\Sigma = (S, OP, P)$ eine Signatur, sei $(f^i: A \rightarrow B^i)_{i \in I}$ eine Familie von Σ -Homomorphismen mit gleicher Quelle und sei*

$p \in P_w$ ein Prädikat zu einem Sortenwort $w \in S^*$. Dann sind $(f^i)_{i \in I}$ gemeinsam strikt voll auf dem Prädikat p , wenn

$$(A.4) \quad (\forall i \in I : f_w^i(x) \in p^{B^i}) \Rightarrow x \in p^A$$

für alle $x \in A_w$ gilt. $(f^i)_{i \in I}$ sind gemeinsam strikt voll, wenn $(f^i)_{i \in I}$ gemeinsam strikt voll auf jedem Prädikat sind.

Lemma A.26 (Komposition strikt voller Homomorphismen). Sei $\Sigma = (S, OP, P)$ eine Signatur und sei $p \in P_w$ ein Prädikat zu einem Sortenwort $w \in S^*$. Sei weiter $(f^i : A \rightarrow B^i)_{i \in I}$ eine Familie von Σ -Homomorphismen mit gleicher Quelle, die gemeinsam strikt voll sind, und sei $g : X \rightarrow A$ ein weiterer strikt voller Σ -Homomorphismus. Dann gilt, dass die Familie $(f^i \circ g : X \rightarrow B^i)_{i \in I}$ von Σ -Homomorphismen ebenfalls gemeinsam strikt voll ist. \square

Definition A.27 (Monomorphismus). Sei $\Sigma = (S, OP, P)$ eine Signatur. Ein Σ -Homomorphismus $f : A \rightarrow B$ heißt Monomorphismus in $\mathbf{Alg}(\Sigma)$, wenn

$$f \circ g = f \circ h \Rightarrow g = h$$

für alle Σ -Homomorphismen $g : C \rightarrow A$ und $h : C \rightarrow A$ sowie alle Σ -Systeme C gilt.

Lemma A.28 (Charakterisierung von Monomorphismen in Kategorien von Systemen). Sei $\Sigma = (S, OP, P)$ eine Signatur. Sei $h : A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann ist h genau dann ein Monomorphismus, wenn h injektiv ist. \square

Definition A.29 (Epimorphismus). Sei $\Sigma = (S, OP, P)$ eine Signatur. Ein Σ -Homomorphismus $f : A \rightarrow B$ heißt Epimorphismus in $\mathbf{Alg}(\Sigma)$, wenn

$$g \circ f = h \circ f \Rightarrow g = h$$

für alle Σ -Homomorphismen $g : B \rightarrow C$ und $h : B \rightarrow C$ sowie alle Σ -Systeme C gilt.

Lemma A.30 (Charakterisierung von Epimorphismen in Kategorien von Systemen). Sei $\Sigma = (S, OP, P)$ eine Signatur. Sei $h : A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann ist h genau dann ein Epimorphismus, wenn h surjektiv ist. \square

Definition A.31 (Isomorphismus). Sei $\Sigma = (S, OP, P)$ eine Signatur. Ein Σ -Homomorphismus $f : A \rightarrow B$ heißt Isomorphismus in $\mathbf{Alg}(\Sigma)$, wenn es einen Σ -Homomorphismus $f^{-1} : B \rightarrow A$ gibt, so dass gilt:

$$\begin{aligned} f^{-1} \circ f &= id^A \\ f \circ f^{-1} &= id^B \end{aligned}$$

Lemma A.32 (Charakterisierung von Isomorphismen in Kategorien von Systemen).

Sei $\Sigma = (S, OP, P)$ eine Signatur. Sei $h: A \rightarrow B$ ein Σ -Homomorphismus zwischen zwei Σ -Systemen A und B . Dann ist h genau dann ein Isomorphismus, wenn h bijektiv und voll ist. \square

Definition A.33 (Extremaler Monomorphismus). Sei $\Sigma = (S, OP, P)$ eine Signatur. Ein Σ -Monomorphismus $m: A \rightarrow B$ heißt extremal in $\mathbf{Alg}(\Sigma)$, wenn für jede Faktorisierung $m = f \circ e$ mit den Σ -Homomorphismen $e: A \rightarrow C$ und $f: C \rightarrow B$ sowie einem Σ -System C die folgende Implikation gilt:

$$e \text{ ist Epimorphismus} \Rightarrow e \text{ ist Isomorphismus}$$

Lemma A.34 (Charakterisierung von extremalen Monomorphismen in Kategorien von Systemen). Sei $\Sigma = (S, OP, P)$ eine Signatur. Sei $h: A \rightarrow B$ ein Σ -Monomorphismus zwischen zwei Σ -Systemen A und B . Dann ist h genau dann extremal, wenn h voll ist. \square

Definition A.35 (Extremaler Epimorphismus). Sei $\Sigma = (S, OP, P)$ eine Signatur. Ein Σ -Epimorphismus $e: A \rightarrow B$ heißt extremal in $\mathbf{Alg}(\Sigma)$, wenn für jede Faktorisierung $e = m \circ f$ mit den Σ -Homomorphismen $f: A \rightarrow C$ und $m: C \rightarrow B$ sowie einem Σ -System C die folgende Implikation gilt:

$$m \text{ ist Monomorphismus} \Rightarrow m \text{ ist Isomorphismus}$$

Lemma A.36 (Charakterisierung von extremalen Epimorphismen in Kategorien von Systemen). Sei $\Sigma = (S, OP, P)$ eine Signatur. Sei $h: A \rightarrow B$ ein Σ -Epimorphismus zwischen zwei Σ -Systemen A und B . Dann ist h genau dann extremal, wenn h voll ist. \square

Satz A.37 ((Epi, ExtrMono)-Faktorisierung). Sei $\Sigma = (S, OP, P)$ eine Signatur. Dann hat $\mathbf{Alg}(\Sigma)$ (Epi, ExtrMono)-Faktorisierungen, d. h. jeder Σ -Homomorphismus $h: A \rightarrow B$ lässt sich eindeutig in einen Σ -Epimorphismus $e: A \rightarrow C$ und einen extremalen Σ -Monomorphismus $m: C \rightarrow B$ für ein Σ -System C zerlegen, so dass $h = m \circ e$ gilt. \square

Definition A.38 (Untersystem, volles Untersystem). Sei Σ eine Signatur und sei A ein Σ -System. Dann ist $(B, m: B \rightarrow A)$ ein Untersystem von A , wenn m injektiver Σ -Homomorphismus ist. Ist m voll, spricht man von einem vollen Untersystem.

Definition A.39 (Quotientensystem, volles Quotientensystem). Sei Σ eine Signatur und sei A ein Σ -System. Dann ist $(B, e: A \rightarrow B)$ ein Quotientensystem von A , wenn e surjektiver Σ -Homomorphismus ist. Ist e voll, spricht man von einem vollen Quotientensystem.

Definition A.40 (Kongruenzrelation). Sei $\Sigma = (S, OP, P)$ eine Signatur und sei A ein Σ -System. Dann ist eine Prädikat-indizierte Familie von Relationen auf den Trägermengen von A $\equiv = (\equiv_p \subseteq A_w)_{w \in S^*, p \in P_w}$ eine Kongruenzrelation auf A , wenn

- $p^A \subseteq \equiv_p$ und
- die Relationen $\equiv_{=s}$ für alle $s \in S$ Äquivalenzrelationen sind, die zusätzlich operationsverträglich sind, d. h. wenn für alle Operationssymbole $op \in OP_{w,s}$ mit $w = s_1 s_2 \dots s_n \in S^*$ und $n > 0$, $s \in S$ und $(x_1, x_2, \dots, x_n), (y_1, y_2, \dots, y_n) \in A_w$ gilt:

$$x_1 \equiv_{=s_1} y_1 \wedge x_2 \equiv_{=s_2} y_2 \wedge \dots \wedge x_n \equiv_{=s_n} y_n \Rightarrow op^A(x_1, x_2, \dots, x_n) \equiv_{=s} op^A(y_1, y_2, \dots, y_n)$$

Konstruktion A.41 (Von Kongruenzrelation erzeugtes Quotientensystem, natürlicher Homomorphismus). Seien Σ eine Signatur, A ein Σ -System und \equiv eine Kongruenzrelation auf A . Dann ergibt die folgende Konstruktion ebenfalls ein Σ -System, das von \equiv erzeugte Quotientensystem A/\equiv :

- $(A/\equiv)_s ::= A_s/\equiv_{=s}$ für alle Sorten $s \in S$;
- $op^{A/\equiv}([x]_{\equiv_w}) ::= [op^A(x)]_{\equiv_s}$ für alle Operationssymbole $op \in OP_{w,s}$ und alle Tupel $x \in A_w$ mit $w \in S^*$ und $s \in S$, wobei $[\]_{\equiv_s} : A_s \rightarrow A_s/\equiv_{=s}$ für jede Sorte $s \in S$ die Abbildung der Elemente der Trägermenge A_s in die jeweiligen Äquivalenzklassen darstellt und $[\]_{\equiv_w}$ das Produkt $[\]_{\equiv_{s_1}} \times [\]_{\equiv_{s_2}} \times \dots \times [\]_{\equiv_{s_n}}$ für $w = s_1 s_2 \dots s_n$, $n > 0$ und die identische Abbildung $\{*\} \rightarrow \{*\}$ für $w = \varepsilon$ ist;
- $p^{A/\equiv} ::= \{ [x]_{\equiv_w} \mid \exists y \in A_w : [x]_{\equiv_w} = [y]_{\equiv_w} \wedge y \in \equiv_p \}$ für alle Prädikate $p \in P_w$ mit $w \in S^*$.

Die Operationen sind wohldefiniert. Für ein beliebiges Operationssymbol $op \in OP_{w,s}$ mit $w \in S^*$ und $s \in S$ seien zwei Tupel $x, y \in A_w$ gegeben mit $[x]_{\equiv_w} = [y]_{\equiv_w}$. Daraus folgt unmittelbar aus der Definition von $[\]_{\equiv_w}$:

$$(A.5) \quad x \equiv_{=w} y$$

Weil \equiv nach Voraussetzung Kongruenzrelation ist, folgt aus (A.5):

$$(A.6) \quad op^A(x) \equiv_{=s} op^A(y)$$

Somit ergibt sich:

$$\begin{aligned} op^{A/\equiv}([x]_{\equiv_w}) &= [op^A(x)]_{\equiv_s} && \text{(Definition von } op^{A/\equiv}\text{)} \\ &= [op^A(y)]_{\equiv_s} && \text{(Definition von } [\]_{\equiv_s} \text{ und (A.6))} \\ &= op^{A/\equiv}([y]_{\equiv_w}) && \text{(Definition von } op^{A/\equiv}\text{)} \end{aligned}$$

□

Lemma A.42 (Surjektiver natürlicher Homomorphismus). Die Sorten-indizierte Familie von Abbildungen $([\]_{\equiv_s} : A_s \rightarrow A_s/\equiv_{=s})_{s \in S}$ aus Konstruktion A.41 ist ein surjektiver Σ -Homo-

morphismus $[\]_{\equiv}: A \rightarrow A/\equiv$, genannt natürlicher Homomorphismus in das Quotientensystem A/\equiv . \square

Lemma A.43 (Kern eines Homomorphismus' ist Kongruenzrelation). Sei Σ eine Signatur, und sei $f: A \rightarrow B$ ein Σ -Homomorphismus. Dann ist $\ker f$ eine Kongruenzrelation. \square

Satz A.44 (Homomorphismussatz). Sei $\Sigma = (S, OP, P)$ eine Signatur. Seien drei Σ -Systeme A , B und C sowie ein Σ -Homomorphismus $f: A \rightarrow B$ und ein surjektiver Σ -Homomorphismus $g: A \rightarrow C$ gegeben. Gelte $\ker g \subseteq \ker f$. Dann existiert ein eindeutiger Σ -Homomorphismus $g^*: C \rightarrow B$ mit $g^* \circ g = f$ (Abb. A.1). \square

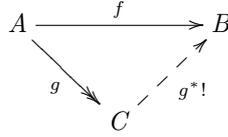


Abb. A.1: Homomorphismussatz

A.4 Limiten

Konstruktion A.45 (Produkt). Sei $(A^i)_{i \in I}$ eine über I indizierte Familie von Σ -Systemen. Dann werden das Σ -System $\times A$, genannt *Produktsystem*, sowie die Familie $(\pi^i: \times A \rightarrow A^i)_{i \in I}$ von Σ -Homomorphismen, genannt *Projektionen*, folgendermaßen konstruiert:

- $\times A_s ::= \prod_{i \in I} A_s^i$ für alle Sorten $s \in S$;
- $op^{\times A}((x_1^i)_{i \in I}, (x_2^i)_{i \in I}, \dots, (x_n^i)_{i \in I}) ::= (op^{A^i}(x_1^i, x_2^i, \dots, x_n^i))_{i \in I}$ für alle Operationssymbole $op \in OP_{w,s}$ mit $w = s_1 s_2 \dots s_n \in S^*$, $n \in \mathbb{N}_0$ und $s \in S$ sowie für alle Element-Tupel $((x_1^j)_{j \in I}, (x_2^j)_{j \in I}, \dots, (x_n^j)_{j \in I}) \in \times A_w$;
- $p^{\times A} ::= \{ ((x_1^i)_{i \in I}, (x_2^i)_{i \in I}, \dots, (x_n^i)_{i \in I}) \mid (x_1^j, x_2^j, \dots, x_n^j) \in p^{A^j} \text{ für alle } j \in I \}$ für alle Prädikate $p \in P_w$ mit $w = s_1 s_2 \dots s_n \in S^*$ und $n \in \mathbb{N}_0$;
- $\pi_s^i((x_1, x_2, \dots, x_n)) ::= x_i$ für alle Sorten $s \in S$ und alle $(x_1, x_2, \dots, x_n) \in \times A_s$.

$\times A$ ist wohldefiniert. Die Wohldefiniertheit der Operationen in $\times A$ folgt direkt aus der Wohldefiniertheit der Operationen in A^i für alle $i \in I$. Für die Wohldefiniertheit der Relationen ist zu zeigen, dass $=_s^{\times A} = Id_{\times A_s}$ für alle $s \in S$. Sei $x = ((x_1^i)_{i \in I}, (x_2^i)_{i \in I}) \in =_s^{\times A}$. Dann gilt nach Definition $(x_1^j, x_2^j) \in =_s^{A^j}$ für alle $j \in I$ und somit $(x_1^i)_{i \in I} = (x_2^i)_{i \in I}$.

Schließlich ist zu zeigen, dass π^i für alle $i \in I$ ein Σ -Homomorphismus ist:

- Für jedes Operationssymbol $op \in OP_{w,s}$ mit $w = s_1 s_2 \dots s_n \in S^*$, $n \in \mathbb{N}_0$ und $s \in S$ sowie für alle $x = ((x_1^j)_{j \in I}, (x_2^j)_{j \in I}, \dots, (x_n^j)_{j \in I}) \in \times A_w$ gilt:

$$\begin{aligned} & \pi_s^i(op^{\times A}((x_1^j)_{j \in I}, (x_2^j)_{j \in I}, \dots, (x_n^j)_{j \in I})) \\ &= \pi_s^i((op^{A^j}(x_1^j, x_2^j, \dots, x_n^j))_{j \in I}) && \text{(Definition von } op^{\times A}\text{)} \\ &= op^{A^i}(x_1^i, x_2^i, \dots, x_n^i) && \text{(Definition von } \pi_s^i\text{)} \\ &= op^{A^i}(\pi_w^i((x_1^j)_{j \in I}, (x_2^j)_{j \in I}, \dots, (x_n^j)_{j \in I})) && \text{(Definition von } \pi_w^i\text{)} \end{aligned}$$

- Für jedes Prädikat $p \in P_w$ mit $w = s_1 s_2 \dots s_n \in S^*$, $n \in \mathbb{N}_0$ sowie für alle $x = ((x_1^j)_{j \in I}, (x_2^j)_{j \in I}, \dots, (x_n^j)_{j \in I}) \in A_w$ gilt:

$$\begin{aligned} & ((x_1^j)_{j \in I}, (x_2^j)_{j \in I}, \dots, (x_n^j)_{j \in I}) \in p^{\times A} \\ & \Rightarrow (x_1^i, x_2^i, \dots, x_n^i) \in p^{A^i} && \text{(Definition von } p^{\times A}\text{)} \\ & \Rightarrow \pi_w^i((x_1^j)_{j \in I}, (x_2^j)_{j \in I}, \dots, (x_n^j)_{j \in I}) \in p^{A^i} && \text{(Definition von } \pi_w^i\text{)} \end{aligned}$$

□

Lemma A.46 (Universelle Eigenschaft des Produkts). Sei $(A^i)_{i \in I}$ eine über I indizierte Familie von Σ -Systemen. Dann ist $(\times A, (\pi^i: \times A \rightarrow A^i)_{i \in I})$ aus Konstruktion A.45 kategoriell das Produkt von $(A^i)_{i \in I}$ in $\mathbf{Alg}(\Sigma)$. □

Konstruktion A.47 (Egalisator). Seien zwei Σ -Homomorphismen $f: A \rightarrow B$ und $g: A \rightarrow B$ gegeben. Dann werden das Σ -System E , genannt *Egalisator-System*, sowie der Σ -Homomorphismus $e: E \rightarrow A = (e_s: E_s \rightarrow A_s)_{s \in S}$ folgendermaßen konstruiert:

- $E_s ::= \{ x \in A_s \mid f_s(x) = g_s(x) \}$ für alle Sorten $s \in S$;
- $op^E ::= op^A|_{E_w}$ für alle Operationssymbole $op \in OP_{w,s}$ mit $w \in S^*$ und $s \in S$;
- $p^E ::= p^A \cap E_w$ für alle Prädikate $p \in P_w$ mit $w \in S^*$;
- $e_s(x) = x$ für alle Sorten $s \in S$ und alle $x \in E_s$.

E ist wohldefiniert. Denn seien ein Operationssymbol $op \in OP_{w,s}$ mit $w \in S^*$ und $s \in S$ sowie ein $x \in E_w$ gegeben. Dann ergibt sich:

$$\begin{aligned} & f_w(x) = g_w(x) && \text{(Konstruktion von } E_w\text{)} \\ & \Rightarrow op^B(f_w(x)) = op^B(g_w(x)) && \text{(} op^B \text{ ist rechtseindeutig)} \\ & \Rightarrow f_s(op^A(x)) = g_s(op^A(x)) && \text{(} f \text{ und } g \text{ sind Homomorphismen)} \\ & \Rightarrow op^A(x) \in E_s && \text{(Konstruktion von } E_s\text{)} \end{aligned}$$

Für die Wohldefiniertheit der Relationen ist zu zeigen, dass $=_s^E = Id_{E_s}$ für alle $s \in S$. Sei $(x_1, x_2) \in =_s^E$. Dann folgt aus der Definition $(x_1, x_2) \in =_s^A$ und somit $x_1 = x_2$, weil A ein Σ -System ist. Umgekehrt folgt für jedes reflexive Paar $(y, y) \in Id_{E_s}$, dass $(y, y) \in Id_{A_s}$ und $(y, y) \in E_{s_s}$ und somit $(y, y) \in =_s^E$.

Schließlich ist zu zeigen, dass e ein Σ -Homomorphismus ist:

- Für jedes Operationssymbol $op \in OP_{w,s}$ mit $w \in S^*$ und $s \in S$ sowie für alle $x \in E_w$ gilt:

$$\begin{aligned} e_s(op^E(x)) &= op^E(x) && \text{(Definition von } e_s) \\ &= op^A(x) && \text{(Definition von } op^E) \\ &= op^A(e_w(x)) && \text{(Definition von } e_w) \end{aligned}$$

- Für jedes Prädikat $p \in P_w$ mit $w \in S^*$ sowie für alle $x \in E_w$ gilt:

$$\begin{aligned} x \in p^E &\Rightarrow x \in p^A && \text{(Definition von } p^E) \\ &\Rightarrow e_s(x) \in p^A && \text{(Definition von } e_s) \end{aligned}$$

□

Lemma A.48 (Universelle Eigenschaft des Egalisators). *Seien zwei Σ -Homomorphismen $f: A \rightarrow B$ und $g: A \rightarrow B$ gegeben. Dann ist (E, e) aus Konstruktion A.47 kategoriell der Egalisator von f und g in $\mathbf{Alg}(\Sigma)$.* □

Satz A.49 ($\mathbf{Alg}(\Sigma)$ hat Limiten). *Für jedes kleine Diagramm in $\mathbf{Alg}(\Sigma)$ existiert ein Limes.* □

Lemma A.50 (Projektionen sind gemeinsam strikt voll). *Sei $(\times A, (\pi^i)_{i \in I})$ Produkt von $(A^i)_{i \in I}$ in $\mathbf{Alg}(\Sigma)$. Dann sind die Projektionen $(\pi^i)_{i \in I}$ gemeinsam strikt voll.* □

Lemma A.51 (Egalisator ist strikt voll). *Sei (E, e) Egalisator zweier Σ -Homomorphismen $f: A \rightarrow B$ und $g: A \rightarrow B$ in $\mathbf{Alg}(\Sigma)$. Dann ist e strikt voll.* □

Lemma A.52 (Pullbacks und eindeutige Urbilder (1)). *Sei der Σ -Span $A \xleftarrow{g^*} D \xrightarrow{f^*} B$ Pullback des Σ -Kospans $A \xrightarrow{f} C \xleftarrow{g} B$. Seien zwei Elemente $x \in A_s$ und $y \in B_s$ zu einer beliebigen Sorte $s \in S$ gegeben mit $f_s(x) = g_s(y)$. Dann existiert ein eindeutiges Element $z \in D_s$ mit $g_s^*(z) = x$ und $f_s^*(z) = y$.* □

Lemma A.53 (Pullback-Morphismen sind gemeinsam strikt voll (1)). *Sei der Σ -Span $A \xleftarrow{g^*} D \xrightarrow{f^*} B$ Pullback des Σ -Kospans $A \xrightarrow{f} C \xleftarrow{g} B$. Dann sind g^* und f^* gemeinsam strikt voll.* □

Lemma A.54 (Pullbacks übertragen surjektive Homomorphismen (1)). Sei der Σ -Span $A \xleftarrow{g^*} D \xrightarrow{f^*} B$ Pullback des Σ -Kospans $A \xrightarrow{f} C \xleftarrow{g} B$. Sei f surjektiv. Dann ist f^* ebenfalls surjektiv. \square

A.5 Kolimiten

Konstruktion A.55 (Koprodukt, Summe). Sei $(A^i)_{i \in I}$ eine über I indizierte Familie von Σ -Systemen. Sei X ein Variablenvorrat, definiert für alle Sorten $s \in S$ durch:

$$X_s ::= \uplus_{i \in I} A_s^i$$

Dann wird das Σ -System $+A$, genannt *Summensystem*, definiert durch

$$+A ::= T^\Sigma(X) / \equiv$$

wobei $\equiv ::= \text{cong}(\sim)$ die kleinste Kongruenzrelation ist, die folgende Prädikat-indizierte Familie von Relationen $\sim = (\sim_p)_{p \in P}$ umfasst:

- Sei $p = =_s \in P_{s\ s}$ für eine beliebige Sorte $s \in S$. Dann gilt

$$(op^{T^\Sigma(X)}(x), op^{A^i}(x)) \in \sim_{=s}$$

für alle $op \in OP_{w,s}$ mit $w \in S^*$ und alle $x \in A_w^i$ für ein $i \in I$.

- Sei $p \in P_w$ mit $w \in S^*$ und $p \neq =_s$ für alle $s \in S$. Dann gilt

$$x \in \sim_p$$

falls $x \in p^{A^i}$ für ein $i \in I$.

- Keine anderen Elemente sind in \sim .

Die Familie $(\iota^i: A^i \rightarrow +A)_{i \in I}$ von Σ -Homomorphismen, genannt *Inklusionen*, wird für alle $i \in I$, alle $s \in S$ und alle $x \in A_s^i$ definiert durch:

$$\iota_s^i(x) ::= [x]_{\equiv_s}$$

$+A$ ist wohldefiniert nach Konstruktion. Die Prädikat-indizierte Familie von Relationen $\sim = (\sim_p)_{p \in P}$ ist für alle $p = =_s$ mit $s \in S$ wohldefiniert, denn nach Definition von X ist für alle $op \in OP_{w,s}$ mit $w \in S^*$ jedes Element $x \in A_w^i$ auch Element der Trägermenge $T^\Sigma(X)_w$, und jedes Element $op^{A^i}(x) \in A_s^i$ ist auch Element der Trägermenge $T^\Sigma(X)_s$. Somit sind die Elemente,

die durch die Terme $op^{T^\Sigma(X)}(x)$ und $op^{A^i}(x)$ bezeichnet werden, Elemente derselben Trägermenge $T^\Sigma(X)_s$.

Es bleibt zu zeigen, dass i^i für alle $i \in I$ ein Σ -Homomorphismus ist:

- Für jedes Operationssymbol $op \in OP_{w,s}$ mit $w \in S^*$ und $s \in S$ sowie für alle $x \in A_w^i$ gilt:

$$\begin{aligned} i_s^i(op^{A^i}(x)) &= [op^{A^i}(x)]_{\equiv_s} && \text{(Definition von } i_s^i) \\ &= [op^{T^\Sigma(X)}(x)]_{\equiv_s} && \text{(Definition von } \equiv_s) \\ &= op^{+A}([x]_{\equiv_s}) && ([]_{\equiv} \text{ ist Homomorphismus)} \\ &= op^{+A}(i_w^i(x)) && \text{(Definition von } i_w^i) \end{aligned}$$

- Für jedes Prädikat $p \in P_w$ mit $w \in S^*$ und $p \neq =_s$ für alle $s \in S$ sowie für alle $x \in A_w^i$ gilt:

$$\begin{aligned} x \in p^{A^i} &\Rightarrow x \in \sim_p && \text{(Definition von } \sim_p) \\ &\Rightarrow x \in \equiv_p && (\sim \subseteq \equiv) \\ &\Rightarrow [x]_{\equiv_w} \in p^{+A} && \text{(Definition eines Quotientensystems)} \\ &\Rightarrow i_w^i(x) \in p^{+A} && \text{(Definition von } i_w^i) \end{aligned}$$

- Für jedes Prädikat $p = =_s \in P_{s,s}$ mit $s \in S$ sowie für alle $x \in A_{s,s}^i$ folgt $x \in p^{A^i} \Rightarrow i_w^i(x) \in p^{+A}$ aus der Rechtseindeutigkeit von i_w^i . \square

Lemma A.56 (Universelle Eigenschaft des Koproducts). Sei $(A^i)_{i \in I}$ eine über I indizierte Familie von Σ -Systemen. Dann ist $(+A, (i^i: A^i \rightarrow +A)_{i \in I})$ aus Konstruktion A.55 kategoriell das Koproduct von $(A^i)_{i \in I}$ in $\mathbf{Alg}(\Sigma)$. \square

Konstruktion A.57 (Koegalisor). Seien zwei Σ -Homomorphismen $f: A \rightarrow B$ und $g: A \rightarrow B$ gegeben. Dann werden das Σ -System C , genannt *Koegalisor-System*, definiert durch

$$C ::= B / \equiv$$

wobei $\equiv ::= \text{cong}(\sim)$ die kleinste Kongruenzrelation ist, die folgende Prädikat-indizierte Familie von Relationen $\sim = (\sim_p)_{p \in P}$ umfasst:

- Sei $p = =_s \in P_{s,s}$ für eine beliebige Sorte $s \in S$. Dann gilt

$$(f_s(x), g_s(x)) \in \sim_{=s}$$

für alle $x \in A_s$.

- Sei $p \in P_w$ mit $w \in S^*$ und $p \neq =_s$ für alle $s \in S$. Dann gilt

$$x \in \sim_p$$

falls $x \in p^B$.

- Keine anderen Elemente sind in \sim .

Der Σ -Homomorphismus $c: B \rightarrow C = (c_s: B_s \rightarrow C_s)_{s \in S}$ wird für alle $s \in S$ und alle $x \in B_s$ definiert durch:

$$c_s(x) ::= [x]_{\equiv_s}$$

C ist wohldefiniert nach Konstruktion. Die Homomorphismus-Eigenschaft von c folgt aus der Homomorphismus-Eigenschaft des natürlichen Homomorphismus' $[\]_{\equiv}: B \rightarrow C$. \square

Lemma A.58 (Universelle Eigenschaft des Koegalisors). *Seien zwei Σ -Homomorphismen $f: A \rightarrow B$ und $g: A \rightarrow B$ gegeben. Dann ist (C, c) aus Konstruktion A.57 kategoriell der Koegalisor von f und g in $\mathbf{Alg}(\Sigma)$.* \square

Satz A.59 ($\mathbf{Alg}(\Sigma)$ hat Kolimiten). *Für jedes kleine Diagramm in $\mathbf{Alg}(\Sigma)$ existiert ein Kolimes.* \square

Konstruktion A.60 (Gerichteter Limes). Sei (I, \leq) eine gerichtete partielle Ordnung.¹ Sei eine Familie von Σ -Homomorphismen $(h^{i,j}: A^i \rightarrow A^j)_{i,j \in I, i \leq j}$ gegeben, für die gilt:

$$(A.7) \quad h^{i,i}: A^i \rightarrow A^i = id^{A^i} \quad \text{für alle } i \in I$$

$$(A.8) \quad h^{j,k} \circ h^{i,j} = h^{i,k} \quad \text{für alle } i, j, k \in I \text{ mit } i \leq j \leq k$$

Dann wird das Σ -System A^∞ sowie für alle $i \in I$ die Sorten-indizierte Familie $(\eta_s^i: A_s^i \rightarrow A_s^\infty)_{s \in S}$ von Abbildungen folgendermaßen konstruiert:

- Für jede Sorte $s \in S$ gelte:

$$A_s^\infty ::= (\bigsqcup_{i \in I} A_s^i) / \equiv_s$$

wobei die Relation \equiv_s wie folgt definiert ist: Für $x \in A_s^i$ und $y \in A_s^j$ mit $i, j \in I$ gilt $x \equiv_s y$ genau dann, wenn ein $k \in I$ existiert mit $\max(i, j) \leq k$ und $h_s^{i,k}(x) = h_s^{j,k}(y)$;

- Für jedes Operationssymbol $op: w \rightarrow s \in OP_{w,s}$ mit $w = s_1 s_2 \dots s_n \in S^*$ und $s \in S$ sowie den Elementen $x_i \in A_{s_i}^{k_i}$ mit $k_i \in I$ für alle $1 \leq i \leq n$ gelte:

$$op^{A^\infty}([x_1]_{\equiv_{s_1}}, [x_2]_{\equiv_{s_2}}, \dots, [x_n]_{\equiv_{s_n}}) ::= [op^{A^m}(h_{s_1}^{k_1, m}(x_1), h_{s_2}^{k_2, m}(x_2), \dots, h_{s_n}^{k_n, m}(x_n))]_{\equiv_s}$$

wobei $m \geq \max(k_1, k_2, \dots, k_n)$ ist;

¹ Eine partielle Ordnung (I, \leq) ist *gerichtet*, falls für jede endliche Menge $M \subseteq I$ eine obere Schranke existiert.

- Für jedes Prädikat $p \in P_w$ mit $w = s_1 s_2 \dots s_n \in S^*$ sowie den Elementen $x_i \in A_{s_i}^{k_i}$ mit $k_i \in I$ für alle $1 \leq i \leq n$ gelte

$$([x_1]_{\equiv_{s_1}}, [x_2]_{\equiv_{s_2}}, \dots, [x_n]_{\equiv_{s_n}}) \in p^{A^\infty}$$

wenn $m \geq \max(k_1, k_2, \dots, k_n)$ existiert, so dass gilt:

$$(h_{s_1}^{k_1, m}(x_1), h_{s_2}^{k_2, m}(x_2), \dots, h_{s_n}^{k_n, m}(x_n)) \in p^{A^m}$$

- Die Sorten-indizierte Familie $(i_s^i : A_s^i \rightarrow A_s^\infty)_{s \in S}$ ist als Einschränkung der Sorten-indizierten Familie von natürlichen Abbildungen in das System A^∞ durch

$$i_s^i : A_s^i \rightarrow A_s^\infty ::= []_{\equiv_s} \Big|_{A_s^i}$$

für alle $i \in I$ und alle $s \in S$ definiert.

A^∞ ist wohldefiniert. Dazu ist zu zeigen, dass $\equiv = (\equiv_s)_{s \in S}$ eine Familie von Äquivalenzrelationen, die Operationssymbole repräsentantenunabhängig und die Gleichheits-Relationen Identitäten sind:

- \equiv ist Familie von Äquivalenzen. Es wird gezeigt, dass \equiv_s für alle $s \in S$ Äquivalenzrelation ist und somit die Faktorisierung \equiv wohldefiniert ist. Für eine beliebige Sorte $s \in S$ gilt:
 - \equiv_s ist reflexiv: Sei $x \in \cup_{i \in I} A_s^i$. Somit gibt es ein $i \in I$ mit $x \in A_s^i$. Daraus folgt

$$\begin{aligned} h_s^{i, i}(x) &= id_s^{A_s^i}(x) && \text{(A.7)} \\ &= x && \text{(Eigenschaft des identischen Homomorphismus')} \end{aligned}$$

und somit $x \equiv_s x$.

- \equiv_s ist symmetrisch: Gelte $x \equiv_s y$ für $x \in A_s^i$ und $y \in A_s^j$ mit $i, j \in I$. Nach Definition gibt es ein $k \in I$ mit $h_s^{i, k}(x) = h_s^{j, k}(y)$. Offensichtlich gilt $h_s^{j, k}(y) = h_s^{i, k}(x)$ und somit $y \equiv_s x$.
- \equiv_s ist transitiv: Gelte $x \equiv_s y$ und $y \equiv_s z$ für $x \in A_s^i$, $y \in A_s^j$ und $z \in A_s^k$ mit $i, j, k \in I$. Nach Definition gibt es $m, n \in I$ mit $\max(i, j) \leq m$, $\max(j, k) \leq n$ und:

$$(A.9) \quad h_s^{i, m}(x) = h_s^{j, m}(y)$$

$$(A.10) \quad h_s^{j, n}(y) = h_s^{k, n}(z)$$

Nach Definition gibt es $p \in I$ mit $\max(m, n) \leq p$. Daraus folgt:

$$h_s^{i, p}(x) = h_s^{m, p}(h_s^{i, m}(x)) \quad (A.8)$$

$$= h_s^{m, p}(h_s^{j, m}(y)) \quad (A.9)$$

$$= h_s^{j, p}(y) \quad (A.8)$$

$$= h_s^{n,p}(h_s^{j,n}(y)) \quad (\text{A.8})$$

$$= h_s^{n,p}(h_s^{k,n}(z)) \quad (\text{A.10})$$

$$= h_s^{k,p}(z) \quad (\text{A.8})$$

Daraus folgt $x \equiv_s z$.

- Die Operationen in A^∞ sind wohldefiniert. Sei $op: w \rightarrow s \in OP_{w,s}$ mit $w = s_1 s_2 \dots s_n \in S^*$ und $s \in S$ ein beliebiges Operationssymbol. Seien für alle $1 \leq i \leq n$ die Elemente $x_i \in A_{s_i}^{k_i}$ und $y_i \in A_{s_i}^{l_i}$ gegeben mit $k_i, l_i \in I$, so dass gilt:

$$(A.11) \quad [x_i]_{\equiv_{s_i}} = [y_i]_{\equiv_{s_i}}$$

Zu zeigen ist, dass gilt:

$$(A.12) \quad op^{A^\infty}([x_1]_{\equiv_{s_1}}, [x_2]_{\equiv_{s_2}}, \dots, [x_n]_{\equiv_{s_n}}) = op^{A^\infty}([y_1]_{\equiv_{s_1}}, [y_2]_{\equiv_{s_2}}, \dots, [y_n]_{\equiv_{s_n}})$$

Aus der Definition der Operationen in A^∞ folgt, dass statt (A.12) äquivalent gezeigt werden kann, dass

$$[op^{A^v}(h_{s_1}^{k_1,v}(x_1), h_{s_2}^{k_2,v}(x_2), \dots, h_{s_n}^{k_n,v}(x_n))]_{\equiv_s} = [op^{A^w}(h_{s_1}^{l_1,w}(y_1), h_{s_2}^{l_2,w}(y_2), \dots, h_{s_n}^{l_n,w}(y_n))]_{\equiv_s}$$

gilt für $v, w \in I$ mit $\max(k_1, k_2, \dots, k_n) \leq v$ und $\max(l_1, l_2, \dots, l_n) \leq w$.

Aus (A.11) und der Definition von \equiv_s folgt, dass für alle $1 \leq i \leq n$ ein $m_i \in I$ existiert mit $\max(k_i, l_i) \leq m_i$ und:

$$(A.13) \quad h_s^{k_i, m_i}(x_i) = h_s^{l_i, m_i}(y_i)$$

Sei t eine beliebige obere Schranke von $\{m_1, m_2, \dots, m_n, v, w\}$. Es ergibt sich:

$$\begin{aligned} & h_s^{v,t}(op^{A^v}(h_{s_1}^{k_1,v}(x_1), h_{s_2}^{k_2,v}(x_2), \dots, h_{s_n}^{k_n,v}(x_n))) \\ &= op^{A^t}(h_{s_1}^{v,t}(h_{s_1}^{k_1,v}(x_1)), h_{s_2}^{v,t}(h_{s_2}^{k_2,v}(x_2)), \dots, h_{s_n}^{v,t}(h_{s_n}^{k_n,v}(x_n))) \quad (h^{v,t} \text{ ist Homomorph.}) \end{aligned}$$

$$= op^{A^t}(h_{s_1}^{k_1,t}(x_1), h_{s_2}^{k_2,t}(x_2), \dots, h_{s_n}^{k_n,t}(x_n)) \quad (\text{A.8})$$

$$= op^{A^t}(h_{s_1}^{m_1,t}(h_{s_1}^{k_1,m_1}(x_1)), h_{s_2}^{m_2,t}(h_{s_2}^{k_2,m_2}(x_2)), \dots, h_{s_n}^{m_n,t}(h_{s_n}^{k_n,m_n}(x_n))) \quad (\text{A.8})$$

$$= op^{A^t}(h_{s_1}^{m_1,t}(h_{s_1}^{l_1,m_1}(y_1)), h_{s_2}^{m_2,t}(h_{s_2}^{l_2,m_2}(y_2)), \dots, h_{s_n}^{m_n,t}(h_{s_n}^{l_n,m_n}(y_n))) \quad (\text{A.13})$$

$$= op^{A^t}(h_{s_1}^{l_1,t}(y_1), h_{s_2}^{l_2,t}(y_2), \dots, h_{s_n}^{l_n,t}(y_n)) \quad (\text{A.8})$$

$$= op^{A^t}(h_{s_1}^{w,t}(h_{s_1}^{l_1,w}(y_1)), h_{s_2}^{w,t}(h_{s_2}^{l_2,w}(y_2)), \dots, h_{s_n}^{w,t}(h_{s_n}^{l_n,w}(y_n))) \quad (\text{A.8})$$

$$= h_s^{w,t}(op^{A^w}(h_{s_1}^{l_1,w}(y_1), h_{s_2}^{l_2,w}(y_2), \dots, h_{s_n}^{l_n,w}(y_n))) \quad (h^{w,t} \text{ ist Homomorph.})$$

- Die Relationen in A^∞ sind wohldefiniert. Es ist zu zeigen, dass für jedes Prädikat $p = =_s \in P_{s,s}$ mit $s \in S$ sowie den Elementen $x_1 \in A_s^{k_1}$ und $x_2 \in A_s^{k_2}$ mit $k_1, k_2 \in I$ gilt:

$$(A.14) \quad ([x_1]_{\equiv_s}, [x_2]_{\equiv_s}) \in =_s^{A^\infty} \Rightarrow [x_1]_{\equiv_s} = [x_2]_{\equiv_s}$$

Gelte die Prämisse von (A.14). Aus der Definition der Relationen folgt, dass ein $m \in I$ existiert mit $\max(k_1, k_2) \leq m$ und:

$$(A.15) \quad (h_s^{k_1, m}(x_1), h_s^{k_2, m}(x_2)) \in =_s^{A^m}$$

Weil A^m ein Σ -System ist, ist $=_s^{A^m}$ die identische Relation auf A_s^m . Somit folgt aus (A.15):

$$\begin{aligned} h_s^{k_1, m}(x_1) &= h_s^{k_2, m}(x_2) \\ \Rightarrow [x_1]_{\equiv_s} &= [x_2]_{\equiv_s} \end{aligned} \quad (\text{Definition von } \equiv_s)$$

□

Lemma A.61 (Einbettungen sind Homomorphismen). Die Sorten-indizierte Familie von Abbildungen $(i_s^i: A_s^i \rightarrow A_s^\infty)_{s \in S}$ aus Konstruktion A.60 ist für alle $i \in I$ ein Σ -Homomorphismus $i^i: A^i \rightarrow A^\infty$, der mit der Familie von Homomorphismen $(h^{i,j}: A^i \rightarrow A^j)_{i,j \in I, i \leq j}$ verträglich ist, d. h. es gilt:

$$(A.16) \quad i^j \circ h^{i,j} = i^i$$

für alle $i, j \in I$ mit $i \leq j$.

□

Lemma A.62 (Gerichteter Limes und Kolimes). Sei (I, \leq) eine gerichtete partielle Ordnung. Sei eine Familie von Σ -Homomorphismen $(h^{i,j}: A^i \rightarrow A^j)_{i,j \in I, i \leq j}$ gegeben, für die gilt:

$$\begin{aligned} h^{i,i}: A^i &\rightarrow A^i = id^{A^i} && \text{für alle } i \in I \\ h^{j,k} \circ h^{i,j} &= h^{i,k} && \text{für alle } i, j, k \in I \text{ mit } i \leq j \leq k \end{aligned}$$

Dann ist der gerichtete Limes $(A^\infty, (i^i: A^i \rightarrow A^\infty)_{i \in I})$ aus Konstruktion A.60 Kolimes des Diagramms $(h^{i,j}: A^i \rightarrow A^j)_{i,j \in I, i \leq j}$ in $\mathbf{Alg}(\Sigma)$.

□

A.6 Implikative Spezifikationen und Modelle

Definition A.63 (Atomare Formel). Seien Σ eine Signatur und X ein Σ -Variablenvorrat. Dann ist die Menge $\text{Atom}_\Sigma(X)$ der atomaren Formeln über dem Variablenvorrat X folgendermaßen definiert:

$$t \in T^\Sigma(X)_w \wedge p \in P_w \Rightarrow p(t) \in \text{Atom}_\Sigma(X)$$

für alle $w \in S^*$.

Definition A.64 (Lösung einer atomaren Formel). Seien Σ eine Signatur, X ein Σ -Variablenvorrat, $p(t) \in \text{Atom}_\Sigma(X)$ eine atomare Formel zu einem Prädikat $p \in P_w$ und A ein Σ -System. Dann löst eine Variablenbelegung $\text{asg}: X \rightarrow A$ die atomare Formel $p(t)$ in A , wenn gilt:

$$\text{eval}(\text{asg})_w(t) \in p^A$$

Die Variablenbelegung asg ist dann Lösung der atomaren Formel $p(t)$ in A .

Definition A.65 (Gültigkeit einer atomaren Formel). Seien Σ eine Signatur, X ein Σ -Variablenvorrat, $p(t) \in \text{Atom}_\Sigma(X)$ eine atomare Formel zu einem Prädikat $p \in P_w$ und A ein Σ -System. Dann ist die atomare Formel gültig in A , wenn jede Variablenbelegung $\text{asg}: X \rightarrow A$ die atomare Formel in A löst.

Definition A.66 (Implikation, endliche Implikation). Seien Σ eine Signatur, X ein Σ -Variablenvorrat und I eine Menge. Dann ist

$$X : (\text{pre}_i)_{i \in I} \Rightarrow \text{con}$$

mit $\text{pre}_i \in \text{Atom}_\Sigma(X)$ für alle $i \in I$ und $\text{con} \in \text{Atom}_\Sigma(X)$ eine Implikation über X . Die atomaren Formeln pre_i werden Prämissen und die atomare Formel con Konklusion genannt.

Ist X endlich,² dann handelt es sich um eine endliche Implikation. Ist die Indexmenge I leer, handelt es sich um eine Gleichung, und es wird lediglich die Konklusion notiert.

Definition A.67 (Lösung einer Implikation). Seien Σ eine Signatur, X ein Σ -Variablenvorrat, $h = ((\text{pre}_i)_{i \in I} \Rightarrow \text{con})$ eine Implikation über X und A ein Σ -System. Dann löst eine Variablenbelegung $\text{asg}: X \rightarrow A$ die Implikation h in A , wenn gilt: Falls asg alle Prämissen $(\text{pre}_i)_{i \in I}$ in A löst, dann auch die Konklusion con . Die Variablenbelegung asg ist dann Lösung der Implikation h in A .

Definition A.68 (Gültigkeit einer Implikation). Seien Σ eine Signatur, X ein Σ -Variablenvorrat, $h = ((\text{pre}_i)_{i \in I} \Rightarrow \text{con})$ eine Implikation über X und A ein Σ -System. Dann ist h gültig in A , wenn jede Variablenbelegung $\text{asg}: X \rightarrow A$ die Implikation h in A löst.

Definition A.69 (Axiome, implikative Spezifikation). Seien eine Signatur Σ sowie eine Menge I von Implikationen über einem Σ -Variablenvorrat X , Axiome genannt, gegeben. Dann ist $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Ist Σ algebraisch, dann wird Spec algebraische

² Eine Familie von Mengen $X = (X_i)_{i \in I}$ über eine endliche Indexmenge I ist endlich, wenn $\uplus_{i \in I} X_i$ endlich ist.

Spezifikation genannt. Enthält I nur endliche Implikationen, wird Spec endlich implikative Spezifikation genannt.

Definition A.70 (Modell einer Spezifikation). Seien $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation und A ein Σ -System. Dann ist A genau dann ein Modell der Spezifikation Spec , wenn jedes Axiom aus I in A gültig ist. In diesem Fall heißt A auch Spec -System und erfüllt die Spezifikation Spec .

Definition A.71 (Kategorie von Modellen). Sei Spec eine implikative Spezifikation. Dann bezeichnet $\mathbf{Alg}(\text{Spec})$ die folgende Kategorie:

- $\text{Ob}^{\mathbf{Alg}(\text{Spec})}$ besteht aus allen Spec -Modellen;
- $\text{Mor}^{\mathbf{Alg}(\text{Spec})} = (\text{Mor}^{\mathbf{Alg}(\text{Spec})}(A, B))_{A, B \in \text{Ob}^{\mathbf{Alg}(\text{Spec})}}$ besteht aus der Menge aller Σ -Homomorphismen zwischen je zwei Spec -Modellen A und B ;
- die Komposition $\circ^{\mathbf{Alg}(\text{Spec})}$ ist die Komposition von Homomorphismen gemäß Def. A.5; und
- die Identitäten $\text{id}^{\mathbf{Alg}(\text{Spec})} = (\text{id}_A^{\mathbf{Alg}(\text{Spec})} \in \text{Mor}^{\mathbf{Alg}(\text{Spec})}(A, A))_{A \in \text{Ob}^{\mathbf{Alg}(\text{Spec})}}$ sind die identischen Homomorphismen gemäß Def. A.7.

Lemma A.72 (Homomorphismen übertragen Lösungen atomarer Formeln). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation, sei $f: A \rightarrow B$ ein Spec -Homomorphismus, und sei $\text{asg}: X \rightarrow A$ eine Variablenbelegung in A , die eine atomare Formel $p(t) \in \text{Atom}_\Sigma(X)$ mit $t \in T^\Sigma(X)_w$ und $p \in P_w$, $w \in S^*$ löst. Dann löst die Variablenbelegung $f \circ \text{asg}: X \rightarrow B$ die atomare Formel $p(t)$ in B . \square

Lemma A.73 (Strikt volle Homomorphismen reflektieren Lösungen atomarer Formeln). Sei $f: A \rightarrow B$ ein strikt voller Spec -Homomorphismus, und sei $\text{asg}: X \rightarrow A$ eine Variablenbelegung in A . Dann gilt: Wenn die Variablenbelegung $f \circ \text{asg}: X \rightarrow B$ eine atomare Formel $p(t) \in \text{Atom}_\Sigma(X)$ mit $t \in T^\Sigma(X)_w$ und $p \in P_w$, $w \in S^*$ in B löst, dann löst die Variablenbelegung asg diese atomare Formel in A . \square

Lemma A.74 (Volle Unterkategorie). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Dann ist $\mathbf{Alg}(\text{Spec})$ volle Unterkategorie von $\mathbf{Alg}(\Sigma)$. \square

Lemma A.75 (Abgeschlossenheit unter Bildung voller Untersysteme). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Dann ist $\mathbf{Alg}(\text{Spec})$ unter Bildung voller Untersysteme in $\mathbf{Alg}(\Sigma)$ abgeschlossen.³ \square

³ Das bedeutet, dass jedes volle $\mathbf{Alg}(\Sigma)$ -Untersystem eines $\mathbf{Alg}(\text{Spec})$ -Systems ein $\mathbf{Alg}(\text{Spec})$ -System ist.

Lemma A.76 (Abgeschlossenheit unter Produktbildung). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Dann ist $\mathbf{Alg}(\text{Spec})$ unter Bildung von Produkten in $\mathbf{Alg}(\Sigma)$ abgeschlossen, d. h. für eine Menge von Spec-Modellen $(A^i)_{i \in I}$ ist das $\mathbf{Alg}(\Sigma)$ -Produkt $(\times A, (\pi^i)_{i \in I})$ ein Spec-Modell. \square

Lemma A.77 (Abgeschlossenheit unter Isomorphie). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Dann ist $\mathbf{Alg}(\text{Spec})$ unter Isomorphie in $\mathbf{Alg}(\Sigma)$ abgeschlossen, d. h. für jedes Spec-Modell A und jeden $\mathbf{Alg}(\Sigma)$ -Isomorphismus $i: B \rightarrow A$ gilt, dass B ein Spec-Modell ist. \square

Satz A.78 (Implikative Unterkategorie ist epirefektiv (1)). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Dann ist $\mathbf{Alg}(\text{Spec})$ volle epirefektive Unterkategorie von $\mathbf{Alg}(\Sigma)$. \square

Satz A.79 (Implikative Unterkategorie ist epirefektiv (2)). Sei $\text{Spec}' = (\Sigma, I')$ eine implikative Spezifikation und sei $\text{Spec} = (\Sigma, I)$ mit $I \subseteq I'$ eine implikative Unterspezifikation zur selben Signatur. Dann ist $\mathbf{Alg}(\text{Spec}')$ volle epirefektive Unterkategorie von $\mathbf{Alg}(\text{Spec})$. \square

A.7 Morphismen, Limiten und Kolimiten in implikativ spezifizierten Unterkategorien

Lemma A.80 (Charakterisierung von Monomorphismen in Kategorien von Modellen). Seien Σ eine Signatur, $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation zu dieser Signatur und $h: A \rightarrow B$ ein Spec-Homomorphismus zwischen zwei Spec-Modellen A und B . Dann ist h genau dann ein Monomorphismus, wenn h injektiv ist. \square

Lemma A.81 (Epimorphismen in Kategorien von Modellen). Sei Σ eine Signatur, sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation zu dieser Signatur und sei $h: A \rightarrow B$ ein Spec-Homomorphismus zwischen zwei Spec-Modellen A und B . Dann ist h ein Epimorphismus, wenn h surjektiv ist. \square

Lemma A.82 (Kategorien von Modellen sind nicht immer balanciert). Sei eine Signatur Σ mit einer Sorte S und einem Prädikat p auf dieser Sorte gegeben. Dann ist die Kategorie $\mathbf{Alg}(\Sigma)$ aller Systeme zu dieser Signatur und aller existierenden Homomorphismen zwischen ihnen nicht balanciert, d. h. nicht jeder Morphismus, der gleichzeitig Mono- und Epimorphismus ist, ist automatisch ein Isomorphismus. \square

Satz A.83 ($\mathbf{Alg}(\text{Spec})$ hat Limiten). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Dann existiert für jedes kleine Diagramm in $\mathbf{Alg}(\text{Spec})$ ein Limes, und dieser Limes ist identisch zu dem Limes desselben Diagramms in $\mathbf{Alg}(\Sigma)$. \square

Korollar A.84 (Projektionen sind gemeinsam strikt voll). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Sei $(\times A, (\pi^i)_{i \in I})$ Produkt von $(A^i)_{i \in I}$ in $\mathbf{Alg}(\text{Spec})$. Dann sind die Projektionen $(\pi^i)_{i \in I}$ gemeinsam strikt voll. \square

Korollar A.85 (Egalisator ist strikt voll). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Sei (E, e) Egalisator zweier Σ -Homomorphismen $f: A \rightarrow B$ und $g: A \rightarrow B$ in $\mathbf{Alg}(\text{Spec})$. Dann ist e strikt voll. \square

Korollar A.86 (Pullbacks und eindeutige Urbilder (2)). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Sei der Spec-Span $A \xleftarrow{g^*} D \xrightarrow{f^*} B$ Pullback des Spec-Kospans $A \xrightarrow{f} C \xleftarrow{g} B$. Seien zwei Elemente $x \in A_s$ und $y \in B_s$ zu einer beliebigen Sorte $s \in S$ gegeben mit $f_s(x) = g_s(y)$. Dann existiert ein eindeutiges Element $z \in D_s$ mit $g_s^*(z) = x$ und $f_s^*(z) = y$. \square

Korollar A.87 (Pullback-Morphismen sind gemeinsam strikt voll (2)). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Sei der Spec-Span $A \xleftarrow{g^*} D \xrightarrow{f^*} B$ Pullback des Spec-Kospans $A \xrightarrow{f} C \xleftarrow{g} B$. Dann sind g^* und f^* gemeinsam strikt voll. \square

Korollar A.88 (Pullbacks übertragen surjektive Homomorphismen (2)). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Sei der Σ -Span $A \xleftarrow{g^*} D \xrightarrow{f^*} B$ Pullback des Σ -Kospans $A \xrightarrow{f} C \xleftarrow{g} B$. Sei f surjektiv. Dann ist f^* ebenfalls surjektiv. \square

Satz A.89 ($\mathbf{Alg}(\text{Spec})$ hat Kolimiten). Sei $\text{Spec} = (\Sigma, I)$ eine implikative Spezifikation. Dann existiert für jedes kleine Diagramm in $\mathbf{Alg}(\text{Spec})$ ein Kolimes. \square

A.8 Horn-Formel-spezifizierte Unterkategorien und Konstruktion der Epireflexion

Definition A.90 (Positive Horn-Formel). Seien Σ eine Signatur, X ein endlicher Σ -Variablenvorrat und I eine endliche Menge. Dann ist die Implikation

$$X : (\text{pre}_i)_{i \in I} \Rightarrow \text{con}$$

mit $\text{pre}_i \in \text{Atom}_\Sigma(X)$ für alle $i \in I$ und $\text{con} \in \text{Atom}_\Sigma(X)$ eine positive Horn-Formel über X .⁴ Insbesondere ist jede Gleichung über einem endlichen Variablenvorrat eine positive Horn-Formel.

Definition A.91 (Spezifikation mit positiven Horn-Formeln). Seien eine Signatur Σ sowie eine Menge H von positiven Horn-Formeln über einem endlichen Σ -Variablenvorrat X gegeben. Dann ist $\text{Spec} = (\Sigma, H)$ eine Spezifikation mit positiven Horn-Formeln oder kurz Horn-Formel-Spezifikation.

⁴ nach dem amerikanischen Mathematiker Alfred Horn, vgl. auch [35]

Konstruktion A.92 (Epireflexion). Sei $\Sigma = (S, OP, P)$ eine beliebige Signatur und sei $Spec = (\Sigma, H)$ eine zugehörige Spezifikation mit positiven Horn-Formeln H über einem Variablenvorrat X . Sei ferner ein beliebiges Σ -System A gegeben. Dann werden die Systeme $(A^i)_{i \in \mathbb{N}_0}$ wie folgt induktiv konstruiert:

- (1) $A^0 ::= A$
- (2) $A^{i+1} ::= A^i / \equiv^i$

Für jedes $i \in \mathbb{N}_0$ werden die Relation Δp^{A^i} und die Kongruenzrelation \equiv^i folgendermaßen konstruiert:

- Δp^{A^i} ist für jedes Prädikat $p \in P_w$ mit $w \in S^*$ die Menge all jener Tupel $x \in A_w^i$, für die gilt: Es gibt eine positive Horn-Formel $X : p_1(t_1) \wedge p_2(t_2) \wedge \dots \wedge p_n(t_n) \Rightarrow p(t)$ in H mit den Term-Tupeln $t_j \in T^\Sigma(X)_{w_j}$, $1 \leq j \leq n$ und $t \in T^\Sigma(X)_w$ für $w_1, w_2, \dots, w_n \in S^*$ sowie eine Variablenbelegung $asg : X \rightarrow A^i$, so dass $eval(asg)_w(t) = x$ und $eval(asg)_{w_j}(t_j) \in p_j^{A^i}$ für alle $1 \leq j \leq n$ gilt;
- $\equiv^i ::= \text{cong}((p^{A^i} \cup \Delta p^{A^i})_{p \in P})$ für alle Prädikate $p \in P_w$ mit $w \in S^*$, wobei $\text{cong}(R)$ die kleinste Kongruenzrelation darstellt, welche die Prädikat-indizierte Familie von Relationen $R = (R_p)_{p \in P}$ umfasst;
- $[\]_{\equiv^i} : A^i \rightarrow A^{i+1}$ ist der natürliche Homomorphismus in das jeweilige Quotientensystem.

Im Folgenden sei für alle $i, j \in \mathbb{N}_0$ und $j > i + 1$ definiert:

$$(A.17) \quad h^{i,i} : A^i \rightarrow A^i ::= id^{A^i}$$

$$(A.18) \quad h^{i,i+1} : A^i \rightarrow A^{i+1} ::= [\]_{\equiv^i}$$

$$(A.19) \quad h^{i,j} : A^i \rightarrow A^j ::= h^{i+1,j} \circ h^{i,i+1}$$

Sei $(A^\infty, (i^i : A^i \rightarrow A^\infty)_{i \in \mathbb{N}_0})$ der gerichtete Limes von $(h^{i,j} : A^i \rightarrow A^j)_{i, j \in \mathbb{N}_0, i \leq j}$ (Abb. A.2). Dann ist (A^∞, i^0) die zu konstruierende *Spec-Epireflexion* von A . \square

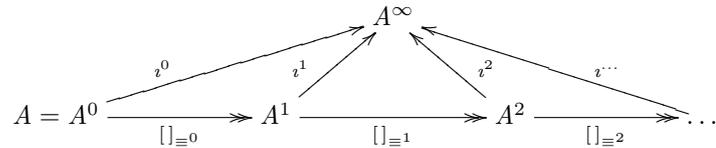


Abb. A.2: Konstruktion der Epireflexion

Satz A.93 (Epireflexion). Sei $\Sigma = (S, OP, P)$ eine beliebige Signatur und sei $Spec = (\Sigma, H)$ eine zugehörige Spezifikation mit einer Menge von positiven Horn-Formeln H über einem Variablenvorrat X . Dann ist (A^∞, i^0) aus Konstruktion A.92 eine **Alg(Spec)**-Reflexion für jedes **Alg(Σ)**-Objekt A , wobei i^0 **Alg(Σ)**-Epimorphismus ist. \square

A.9 Systeme und Algebren

Konstruktion A.94 (Algebraische Spezifikation Σ^A). Sei $\Sigma = (S, OP, P)$ eine Signatur. Sei P' die Menge von Prädikaten ohne die Gleichheitsprädikate, d. h. es gelte:

$$P' ::= P \setminus (=)_s \text{ für alle } s \in S$$

Dann wird die algebraische Spezifikation $\Sigma^A = (S_A, OP^A, A)$ folgendermaßen konstruiert:⁵

- $S_A ::= S \uplus (\bigsqcup_{w \in S^*} P'_w)$, die Sorten in S_A sind also die Sorten und Prädikate in Σ ohne die Gleichheitsprädikate;
- $OP^A ::= OP \uplus (\pi_p)_{p \in P'_w, w = s_1 s_2 \dots s_n \in S^*}$ mit $\pi_p = (\pi_{p,i} : p \rightarrow s_i)_{1 \leq i \leq n}$ und $n \geq 0$; zu jedem Prädikat $p \in P'_w$ wird folglich für jede Sorte s_i in w ein Operationssymbol $\pi_{p,i}$ hinzugefügt, das von der Prädikatensorte p in die Sorte s_i abbildet;
- für jedes Prädikat $p \in P'_w$ mit $w \in S$ ist das Axiom

$$(A.20) \quad x, y \in p : \pi_{p,w}(x) = \pi_{p,w}(y) \Rightarrow x = y$$

Element der Axiom-Menge A . Dabei ist $\pi_{p,w}(x) = \pi_{p,w}(y)$ eine Kurzschreibweise für $\pi_{p,1}(x) = \pi_{p,1}(y) \wedge \pi_{p,2}(x) = \pi_{p,2}(y) \wedge \dots \wedge \pi_{p,n}(x) = \pi_{p,n}(y)$ mit $w = s_1 s_2 \dots s_n \in S^*$, $n > 0$ und für die leere Prämisse mit $w = \varepsilon$.⁶ □

Konstruktion A.95 (Funktorkonstruktion \mathcal{F}_Σ). Sei $\Sigma = (S, OP, P)$ eine Signatur und Σ^A die zugehörige algebraische Spezifikation aus Konstruktion A.94. Seien A, B zwei Σ^A -Systeme und $f : A \rightarrow B$ ein Σ^A -Homomorphismus. Dann werden die Σ -Systeme $\mathcal{F}_\Sigma A$ und $\mathcal{F}_\Sigma B$ sowie der Σ -Homomorphismus $\mathcal{F}_\Sigma f : \mathcal{F}_\Sigma A \rightarrow \mathcal{F}_\Sigma B$ wie folgt konstruiert:

- $\mathcal{F}_\Sigma A$ wird definiert durch:
 - $\mathcal{F}_\Sigma A_s ::= A_s$ für alle $s \in S$
 - $op^{\mathcal{F}_\Sigma A} ::= op^A$ für alle $op \in OP_{w,s}$ mit $w \in S^*$ und $s \in S$
 - $p^{\mathcal{F}_\Sigma A} ::= \{ \pi_{p,w}^A(x) \mid x \in A_p \}$ für alle $p \in P'_w$, $w \in S^*$
 - $=_s^{\mathcal{F}_\Sigma A} ::= Id_{\mathcal{F}_\Sigma A_s}$ für alle $s \in S$
- $\mathcal{F}_\Sigma B$ wird analog zu $\mathcal{F}_\Sigma A$ definiert.
- $\mathcal{F}_\Sigma f_s ::= f_s$ für alle $s \in S$

⁵ Das ‘‘A’’ in Σ^A steht für ‘‘algebraisch’’ (s. Def. A.1).

⁶ Für $w = \varepsilon$ ist das Axiom somit identisch zu der Gleichung $x, y \in p : x = y$.

Dabei ist $\pi_{p,w}^A$ die Kurzschreibweise für das Produkt $\pi_{p,1}^A \times \pi_{p,2}^A \times \cdots \times \pi_{p,n}^A$ für $w = s_1 s_2 \dots s_n \in S^*$, $n > 0$ und für die Identität $\{*\} \rightarrow \{*\}$ für $w = \varepsilon$. Dies gilt analog für $\pi_{p,w}^B$.

Die Konstruktion der Σ -Systeme ist wohldefiniert. Die Wohldefiniertheit der Operationen folgt aus der Wohldefiniertheit der Operationen im Σ^A -System und der Eins-zu-eins-Übernahme der Trägermengen. Die Relationen sind Teilmengen der entsprechenden Trägermengen, da $\text{Im } \pi_{p,w}^A \subseteq A_w$ nach Definition des π -Operationssymbols für jedes Prädikat $p \in P'_w$ mit $w \in S^*$ gilt. Schließlich sind die Gleichheitsrelationen nach Definition allesamt Identitäten.

Es bleibt zu zeigen, dass $\mathcal{F}_\Sigma f$ Σ -Homomorphismus ist. Offensichtlich gilt $\mathcal{F}_\Sigma f_s \circ op^A = op^B \circ \mathcal{F}_\Sigma f_w$ für alle $s \in S$ und $w \in S^*$, da f Σ^A -Homomorphismus ist und nach Definition die Homomorphismus-Abbildungen und die Operationen auf den Trägermengen eins-zu-eins übernommen werden. Weiter gilt

$$\begin{aligned}
\mathcal{F}_\Sigma f_s (=_{s}^{\mathcal{F}_\Sigma A}) &= \mathcal{F}_\Sigma f_s s (Id_{\mathcal{F}_\Sigma A_s}) && \text{(Definition von } =_{s}^{\mathcal{F}_\Sigma A} \text{)} \\
&= f_s s (Id_{\mathcal{F}_\Sigma A_s}) && \text{(Definition von } \mathcal{F}_\Sigma f \text{)} \\
&= f_s s (Id_{A_s}) && \text{(Definition von } \mathcal{F}_\Sigma A_s \text{)} \\
&\subseteq Id_{B_s} && (f_s(A_s) \subseteq B_s) \\
&= Id_{\mathcal{F}_\Sigma B_s} && \text{(Definition von } \mathcal{F}_\Sigma B_s \text{)} \\
&= =_{s}^{\mathcal{F}_\Sigma B} && \text{(Definition von } =_{s}^{\mathcal{F}_\Sigma B} \text{)}
\end{aligned}$$

für alle $=_s \in P_{s s}$ und $s \in S$. Es bleibt zu zeigen, dass $\mathcal{F}_\Sigma f_w(p^{\mathcal{F}_\Sigma A}) \subseteq p^{\mathcal{F}_\Sigma B}$ für alle $p \in P'_w$ und $w \in S^*$. Sei $x \in p^{\mathcal{F}_\Sigma A}$ beliebig. Dann gilt nach Konstruktion, dass ein $y \in A_p$ existiert mit $\pi_{p,w}^A(y) = x$. Somit ergibt sich:

$$\begin{aligned}
\mathcal{F}_\Sigma f_w(x) &= f_w(x) && \text{(nach Definition von } \mathcal{F}_\Sigma f \text{)} \\
&= f_w(\pi_{p,w}^A(y)) && \text{(nach Voraussetzung)} \\
&= \pi_{p,w}^B(f_p(y)) && (f \text{ ist Homomorphismus)}
\end{aligned}$$

Aus der Definition von $p^{\mathcal{F}_\Sigma B}$ folgt unmittelbar $\mathcal{F}_\Sigma f_w(x) \in p^{\mathcal{F}_\Sigma B}$. □

Satz A.96 (Äquivalenz von $\text{Alg}(\Sigma)$ und $\text{Alg}(\Sigma^A)$). Sei $\Sigma = (S, OP, P)$ eine Signatur und Σ^A die zugehörige algebraische Spezifikation aus Konstruktion A.94. Dann sind die Kategorien $\text{Alg}(\Sigma)$ und $\text{Alg}(\Sigma^A)$ äquivalent. □

B

Algebraische Graph-Strukturen

In diesem Kapitel sind alle Definitionen und Sätze aus dem Bereich “algebraische Graphstrukturen” zu finden, die in dieser Arbeit benötigt werden. Dies schließt Ergebnisse zur Transformationen dieser Strukturen mittels des DPO-(Doppel-Pushout-)Ansatzes ein. Es orientiert sich im Großen und Ganzen an [75]. Beweise und erläuternde Bemerkungen sind jedoch aus Platzgründen weggelassen worden.

B.1 Graphstrukturen

Definition B.1 (Gerichteter Graph). *Ein gerichteter Graph ist eine Algebra zu der folgenden Signatur Graph:*

Graph =

sorts

$N(ode)$ (Knoten)

$E(dge)$ (Kanten)

opns

$s(ource): E \rightarrow N$ (Quelle einer Kante)

$t(arget): E \rightarrow N$ (Ziel einer Kante)

Definition B.2 (Graph-Morphismus). *Ein Graph-Morphismus $h: A \rightarrow B$ zwischen zwei gerichteten Graphen A und B ist eine strukturverträgliche Abbildung von Knoten und Kanten des Graphen A auf Knoten und Kanten des Graphen B . Die Abbildung ist strukturverträglich, wenn sie Quellen und Ziele von Kanten erhält. Genauer: Ist ein Knoten n Quelle bzw. Ziel einer Kante e im Graphen A , so ist der Knoten $h(n)$ Quelle bzw. Ziel der Kante $h(e)$.*

Diese Strukturverträglichkeit ist leichter zu formalisieren, wenn die algebraische Definition von gerichteten Graphen zugrunde gelegt wird. Ein Graph-Homomorphismus $h: A \rightarrow B$ zwischen

zwei Graph-Algebren A und B besteht aus einer Knoten-Abbildung $h_N: A_N \rightarrow B_N$ und einer Kanten-Abbildung $h_E: A_E \rightarrow B_E$, so dass die Gleichungen

$$e \in E : h_N(s^A(e)) = s^B(h_E(e))$$

$$e \in E : h_N(t^A(e)) = t^B(h_E(e))$$

gelten.

Definition B.3 (Kategorie Graph). Alle gerichteten Graphen und Graph-Morphismen bilden zusammen die Kategorie **Graph**, die isomorph zu der Kategorie **Alg**(Graph) aller Graph-Algebren und aller Graph-Homomorphismen ist.

B.1.1 Einfache Graphstrukturen

Definition B.4 (Graphstruktur-Signatur). Eine algebraische Signatur $\Sigma = (S, OP)$ ist eine Graphstruktur-Signatur, wenn alle ihre Operationssymbole einstellig⁷ sind.

Definition B.5 (Graphstruktur). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur. Dann ist jede Σ -Algebra eine Σ -Graphstruktur.

Definition B.6 (Graphstruktur-Morphismus). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur. Dann ist jeder Σ -Homomorphismus $f: A \rightarrow B$ zwischen zwei Σ -Graphstrukturen A und B ein Σ -Graphstruktur-Morphismus.

Definition B.7 (Graphstruktur-Kategorie). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur. Dann bildet **GS**(Σ) die Kategorie aller Σ -Graphstrukturen und aller Σ -Graphstruktur-Morphismen.

B.1.2 Attributierte Graphstrukturen

B.1.2.1 Grundlegendes

Definition B.8 (Σ^D -attributierte Graphstruktur-Signatur). Sei $\Sigma^G = (S_G, OP^G)$ eine Graphstruktur-Signatur und $\Sigma^D = (S_D, OP^D)$ eine beliebige algebraische Signatur, genannt Daten-Signatur. Dann ist $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur, wenn gilt:

$$(1) S_D \subseteq S_G$$

⁷ $OP_{\varepsilon, s} = OP_{s_1 s_2 w, s} = \emptyset$ für alle $s, s_1, s_2 \in S$ und $w \in S^*$

$$(2) OP^D \cap OP^G = \emptyset$$

Für jede Σ^D -attributierte Graphstruktur-Signatur $\Sigma = (\Sigma^G, \Sigma^D)$ sei $\Sigma^* = (S_G, OP^G \cup OP^D)$ die zugehörige kombinierte Signatur.

Anmerkung B.9 (Σ^D -attributierte Graphstruktur-Signatur). Definition B.8 ist strenger als die in der Literatur verbreitete, welche versteckte Sorten in S_D erlaubt, die nicht in S_G enthalten sein müssen [20, Def. 11.1]. In dieser Arbeit werden solcherart versteckte Sorten nicht benötigt. Somit sind alle Daten-Sorten Teil der Graphstruktur-Signatur, was die formale Beschreibung von attribuierten Graphstruktur-Signaturen etwas vereinfacht. \square

Anmerkung B.10 (Spezialfall Graphstruktur-Signatur (1)). Jede (gewöhnliche) Graphstruktur-Signatur $\Sigma = (S, OP)$ kann als Σ^D -attributierte Graphstruktur-Signatur $\Sigma' = (\Sigma, \Sigma^D)$ mit der leeren Daten-Signatur $\Sigma^D = (\emptyset, \emptyset)$ angesehen werden. \square

Definition B.11 (Wohlstrukturierte Σ^D -attributierte Graphstruktur-Signatur). Eine Σ^D -attributierte Graphstruktur-Signatur $\Sigma = (\Sigma^G, \Sigma^D)$ heißt wohlstrukturiert, wenn $s \notin S_D$ für jedes Operationssymbol $op: s \rightarrow s'$ in OP^G gilt.

Anmerkung B.12 (Spezialfall Graphstruktur-Signatur (2)). Jede (gewöhnliche) Graphstruktur-Signatur $\Sigma = (S, OP)$ ist offensichtlich wohlstrukturiert. \square

Sei im Folgenden $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur.

Definition B.13 (Σ^D -attributierte Graphstruktur). Jede Σ^* -Algebra A ist eine Σ^D -attributierte Σ^G -Graphstruktur oder kurz Σ -Graphstruktur.

Definition B.14 (Σ^D -attribuierter Graphstruktur-Morphismus). Jeder Σ^* -Homomorphismus $f: A \rightarrow B$ zwischen zwei Σ -Graphstrukturen A und B ist ein Σ^D -attribuierter Σ^G -Graphstruktur-Morphismus oder kurz Σ -Graphstruktur-Morphismus.

Definition B.15 (Kategorie $\mathbf{AGS}(\Sigma)$). Alle Σ -Graphstrukturen und alle Σ -Graphstruktur-Morphismen bilden die Kategorie $\mathbf{AGS}(\Sigma)$.

Anmerkung B.16 (Spezialfall Graphstruktur-Signatur (3)). Falls Σ^D weder Sorten noch Operationssymbole besitzt, folgt $\mathbf{AGS}(\Sigma) = \mathbf{GS}(\Sigma^G)$. \square

B.1.2.2 Konstruktion von Pushouts

Definition B.17 (Klasse \mathcal{M} von Morphismen in $\mathbf{AGS}(\Sigma)$). Seien A und B zwei Σ -Graphstrukturen. Dann ist ein Σ -Graphstruktur-Morphismus $f: A \rightarrow B$ ein \mathcal{M} -Morphismus, wenn

- f_s injektiv ist für alle $s \in S_G \setminus S_D$ und
- f_s bijektiv ist für alle $s \in S_D$.

Konstruktion B.18 (Sorten-indizierte Familie von Äquivalenzrelationen \equiv). Seien zwei Σ -Graphstruktur-Morphismen $f: D \rightarrow A$ und $g: D \rightarrow B$ gegeben, wobei f ein \mathcal{M} -Morphismus ist. Sei $\sim = (\sim_s)_{s \in S_G}$ mit $\sim_s \subseteq (A_s \uplus B_s)^2$ diejenige Sorten-indizierte Familie von Relationen, welche durch

$$\sim_s ::= \{ (f_s(x), g_s(x)) \mid x \in D_s \}$$

für alle Sorten $s \in S_G$ definiert wird. Dann sei $\equiv = (\equiv_s)_{s \in S_G}$ definiert als die kleinste Sorten-indizierte Familie von Äquivalenzrelationen, die \sim umfasst. \square

Lemma B.19 (\equiv und Urbilder (1)). Sei ein Σ -Span $A \xleftarrow{f} D \xrightarrow{g} B$ gegeben, wobei f injektiv ist. Sei \equiv die Sorten-indizierte Familie von Äquivalenzrelationen aus Konstruktion B.18. Dann gilt: Aus $[x]_{\equiv_s} = [y]_{\equiv_s}$ für eine beliebige Sorte $s \in S_G$ und zwei Elemente $x, y \in A_s \uplus B_s$ folgt $x = y$ oder $x, y \in \text{Im } f_s \uplus \text{Im } g_s$. \square

Lemma B.20 (\equiv und Urbilder (2)). Sei ein Σ -Span $A \xleftarrow{f} D \xrightarrow{g} B$ gegeben, wobei f injektiv ist. Sei \equiv die Sorten-indizierte Familie von Äquivalenzrelationen aus Konstruktion B.18. Dann gilt: Für jede Sorte $s \in S_G$ und je zwei Elemente $x, y \in A_s$ mit $x \neq y$ und $[x]_{\equiv_s} = [y]_{\equiv_s}$ existieren zwei eindeutige Elemente $x', y' \in D_s$ mit den folgenden Eigenschaften:

$$\begin{aligned} \text{(B.1)} \quad & f_s(x') = x \\ \text{(B.2)} \quad & f_s(y') = y \\ \text{(B.3)} \quad & g_s(x') = g_s(y') \end{aligned}$$

\square

Lemma B.21 (\equiv und Urbilder (3)). Sei ein Σ -Span $A \xleftarrow{f} D \xrightarrow{g} B$ gegeben, wobei f injektiv ist. Sei \equiv die Sorten-indizierte Familie von Äquivalenzrelationen aus Konstruktion B.18. Dann gilt: Für jede Sorte $s \in S_G$ und je zwei Elemente $x, y \in B_s$ mit $[x]_{\equiv_s} = [y]_{\equiv_s}$ gilt $x = y$. \square

Lemma B.22 (\equiv und Urbilder (4)). Sei ein Σ -Span $A \xleftarrow{f} D \xrightarrow{g} B$ gegeben, wobei f injektiv ist. Sei \equiv die Sorten-indizierte Familie von Äquivalenzrelationen aus Konstruktion B.18. Dann gilt: Für jede Sorte $s \in S_G$ und je zwei Elemente $x \in A_s$ und $y \in B_s$ existiert ein eindeutiges Element $z \in D_s$ mit $f_s(z) = x$ und $g_s(z) = y$. \square

Konstruktion B.23 (Pushouts in $\text{AGS}(\Sigma)$). Seien $f: D \rightarrow A$ und $g: D \rightarrow B$ zwei Σ -Graphstruktur-Morphismen, wobei f ein \mathcal{M} -Morphismus ist. Sei $\equiv = (\equiv_s)_{s \in S_G}$ die Sorten-indizierte Familie von Äquivalenzrelationen aus Konstruktion B.18. Dann werden die Σ -Graphstruktur C sowie die Σ -Graphstruktur-Morphismen $f^*: B \rightarrow C$ und $g^*: A \rightarrow C$ folgendermaßen definiert:

- $C_s = (A_s \uplus B_s) / \equiv_s$ für alle $s \in S_G$
- $op^C([x]_{\equiv_s}) = \begin{cases} [op^A(x)]_{\equiv_{s'}}, & \text{falls } x \in A_s \\ [op^B(x)]_{\equiv_{s'}}, & \text{falls } x \in B_s \end{cases}$
für alle Operationssymbole $op \in OP_{s,s'}^G$ mit $s, s' \in S_G$
- $op^C([x]_{\equiv_w}) = [op^B(y)]_{\equiv_s}$ mit $y \in B_w$ und $[x]_{\equiv_w} = [y]_{\equiv_w}$
für alle Operationssymbole $op \in OP_{w,s}^D$ mit $w \in S_D^*$ und $s \in S_D^8$
- $f_s^*(x) = [x]_{\equiv_s}$ für alle Sorten $s \in S_G$ und alle $x \in B_s$
- $g_s^*(x) = [x]_{\equiv_s}$ für alle Sorten $s \in S_G$ und alle $x \in A_s$ □

Lemma B.24 (Pushouts in $\text{AGS}(\Sigma)$). Sei ein Σ -Span $A \xleftarrow{f} D \xrightarrow{g} B$ gegeben, wobei f ein \mathcal{M} -Morphismus ist. Dann ist der Σ -Kospan $A \xrightarrow{g^*} C \xleftarrow{f^*} B$ aus Konstruktion B.23 Pushout von $A \xleftarrow{f} D \xrightarrow{g} B$. □

B.1.2.3 Eigenschaften von Pushouts und Pullbacks

Definition B.25 (Partielle Injektivität). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur, und seien $l: K \rightarrow L$ und $m: L \rightarrow G$ zwei Σ -Graphstruktur-Morphismen. Dann ist m injektiv bis auf l , wenn für jede Sorte $s \in S_G$ und je zwei Elemente $x, y \in L_s$ gilt:

$$m_s(x) = m_s(y) \wedge x \neq y \Rightarrow x, y \in \text{Im } l_s$$

Lemma B.26 (Pushout-Morphismen sind gemeinsam surjektiv in $\text{AGS}(\Sigma)$). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur. Weiter sei $A \xrightarrow{g^*} C \xleftarrow{f^*} B$ Pushout von $A \xleftarrow{f} D \xrightarrow{g} B$. Dann sind f^* und g^* gemeinsam surjektiv, d. h. für jede Sorte $s \in S_G$ und jedes Element $x \in C_s$ existiert ein Urbild $x_1 \in B_s$ unter f_s^* oder ein Urbild $x_2 \in A_s$ unter g_s^* . □

Lemma B.27 (Pushouts und partielle Injektivität in $\text{AGS}(\Sigma)$). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur. Weiter sei $A \xrightarrow{g^*} C \xleftarrow{f^*} B$ Pushout von $A \xleftarrow{f} D \xrightarrow{g} B$, wobei f injektiv ist. Dann ist g^* injektiv bis auf f . □

Lemma B.28 (Pushouts übertragen injektive Homomorphismen in $\text{AGS}(\Sigma)$). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur. Weiter sei $A \xrightarrow{g^*} C \xleftarrow{f^*} B$ Pushout von $A \xleftarrow{f} D \xrightarrow{g} B$. Dann gilt: Wenn f injektiv ist, dann auch f^* . □

Lemma B.29 (Pushouts und Urbilder in $\text{AGS}(\Sigma)$). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur. Weiter sei $A \xrightarrow{g^*} C \xleftarrow{f^*} B$ Pushout von $A \xleftarrow{f} D \xrightarrow{g} B$, wobei f injektiv ist. Dann gilt: Für je zwei Elemente $x \in A_s$ und $y \in B_s$ mit $s \in S_G$ und $g_s^*(x) = f_s^*(y)$ gibt es ein eindeutiges Element $z \in D_s$ mit $f_s(z) = x$ und $g_s(z) = y$. □

⁸ Sei $w = \{s_1, s_2, \dots, s_n\}$ und $x = (x_1, x_2, \dots, x_n)$ mit $x_i \in (A_{s_i} \uplus B_{s_i})$ und $1 \leq i \leq n$. Dann gilt $[x]_{\equiv_w} = ([x_1]_{\equiv_{s_1}}, [x_2]_{\equiv_{s_2}}, \dots, [x_n]_{\equiv_{s_n}})$.

Lemma B.30 (Pullbacks und Urbilder in $\mathbf{AGS}(\Sigma)$). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur. Weiter sei $A \xleftarrow{f} D \xrightarrow{g} B$ Pullback von $A \xrightarrow{g^*} C \xleftarrow{f^*} B$. Dann gilt: Für je zwei Elemente $x \in A_s$ und $y \in B_s$ mit $s \in S_G$ und $g_s^*(x) = f_s^*(y)$ gibt es ein eindeutiges Element $z \in D_s$ mit $f_s(z) = x$ und $g_s(z) = y$. \square

Lemma B.31 (Pushouts und Pullbacks in $\mathbf{AGS}(\Sigma)$). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur. Weiter sei $A \xrightarrow{g^*} C \xleftarrow{f^*} B$ Pushout von $A \xleftarrow{f} D \xrightarrow{g} B$. Dann gilt: $A \xleftarrow{f} D \xrightarrow{g} B$ ist Pullback von $A \xrightarrow{g^*} C \xleftarrow{f^*} B$, wenn f injektiv ist. \square

Lemma B.32 (Pullbacks und Pushouts in $\mathbf{AGS}(\Sigma)$). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur. Weiter sei $A \xleftarrow{f} D \xrightarrow{g} B$ Pullback von $A \xrightarrow{g^*} C \xleftarrow{f^*} B$. Dann gilt: $A \xrightarrow{g^*} C \xleftarrow{f^*} B$ ist Pushout von $A \xleftarrow{f} D \xrightarrow{g} B$,

(1) wenn f und f^* \mathcal{M} -Morphismen sind, und

(2) wenn f^* und g^* gemeinsam surjektiv sind, und

(3) wenn g^* injektiv bis auf f ist. \square

B.1.3 Typisierte Graphstrukturen

Sei im Folgenden $\Sigma = (S, OP)$ eine beliebige Graphstruktur-Signatur.

Definition B.33 (Typ-Graphstruktur). Eine Σ -Typ-Graphstruktur ist eine beliebige Σ -Graphstruktur.

Definition B.34 (Typisierte Graphstruktur). Sei T eine Σ -Typ-Graphstruktur. Dann ist $A = (A^\#, type_A)$ eine T -typisierte Σ -Graphstruktur, wenn

- $A^\#$ eine Σ -Graphstruktur und
- $type_A: A^\# \rightarrow T$ ein Σ -Graphstruktur-Morphismus ist.

Definition B.35 (Typisierter Graphstruktur-Morphismus). Sei T eine Σ -Typ-Graphstruktur, und seien $A = (A^\#, type_A)$, $B = (B^\#, type_B)$ zwei T -typisierte Σ -Graphstrukturen. Dann ist $f: A \rightarrow B$ ein T -typisierter Σ -Graphstruktur-Morphismus, wenn

- $f: A^\# \rightarrow B^\#$ ein Σ -Graphstruktur-Morphismus ist und
- $type_B \circ f = type_A$ in der Kategorie $\mathbf{GS}(\Sigma)$ gilt.

Definition B.36 (Kategorie $\mathbf{GS}(\Sigma) \downarrow T$). Sei T eine Σ -Typ-Graphstruktur. Dann bilden alle T -typisierten Σ -Graphstrukturen und T -typisierten Σ -Graphstruktur-Morphismen die Kategorie $\mathbf{GS}(\Sigma) \downarrow T$.

B.1.4 Typisierte attributierte Graphstrukturen

Sei im Folgenden $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur mit einer Graphstruktur-Signatur Σ^G und einer Daten-Signatur Σ^D .

Definition B.37 (Σ^D -attrib. Typ-Graphstruktur). Eine Σ -Graphstruktur $T = (T^G, T^D)$ ist eine Σ^D -attributierte Σ^G -Typ-Graphstruktur oder kurz Σ -Typ-Graphstruktur, wenn T^D die finale Σ^D -Algebra darstellt.

Definition B.38 (Typisierte Σ^D -attrib. Graphstruktur). Sei T eine Σ -Typ-Graphstruktur. Dann ist $A = (A^\#, type_A)$ eine T -typisierte Σ -Graphstruktur, wenn

- $A^\#$ eine Σ -Graphstruktur und
- $type_A: A^\# \rightarrow T$ ein Σ -Graphstruktur-Morphismus ist.

Definition B.39 (Typisierter Σ^D -attrib. Graphstruktur-Morphismus). Seien T eine Σ -Typ-Graphstruktur und $A = (A^\#, type_A)$, $B = (B^\#, type_B)$ zwei T -typisierte Σ -Graphstrukturen. Dann ist ein Σ -Graphstruktur-Morphismus $f: A^\# \rightarrow B^\#$ ein T -typisierter Σ -Graphstruktur-Morphismus $f: A \rightarrow B$, wenn er verträglich mit den Typisierungen ist, d. h. wenn

$$type_B \circ f = type_A$$

in $\mathbf{AGS}(\Sigma)$ gilt.

Definition B.40 (Kategorie $\mathbf{AGS}(\Sigma)\downarrow T$). Sei T eine Σ -Typ-Graphstruktur. Dann bilden alle T -typisierten Σ -Graphstrukturen und T -typisierten Σ -Graphstruktur-Morphismen die Kategorie $\mathbf{AGS}(\Sigma)\downarrow T$.

B.2 Graphstruktur-Transformationen

B.2.1 Das grundlegende DPO-Modell

Definition B.41 (Pushout-Komplement). Sei \mathbf{C} eine Kategorie mit Pushouts, und seien drei \mathbf{C} -Objekte A , B und C sowie zwei \mathbf{C} -Morphismen $f: A \rightarrow B$ und $g: B \rightarrow C$ gegeben. Dann ist $A \xrightarrow{g^*} D \xrightarrow{f^*} C$ ein Pushout-Komplement von $A \xrightarrow{f} B \xrightarrow{g} C$, wenn

- D ein \mathbf{C} -Objekt ist,
- f^* und g^* zwei \mathbf{C} -Morphismen sind und
- $B \xrightarrow{g} C \xleftarrow{f^*} D$ Pushout von $B \xleftarrow{f} A \xrightarrow{g^*} D$ in \mathbf{C} ist.

Definition B.42 (Eindeutigkeit des Pushout-Komplements bis auf Isomorphie). Sei \mathbf{C} eine Kategorie mit Pushouts, und sei $A \xrightarrow{g^*} D \xrightarrow{f^*} C$ ein Pushout-Komplement von $A \xrightarrow{f} B \xrightarrow{g} C$. Dann ist $A \xrightarrow{g^*} D \xrightarrow{f^*} C$ eindeutig bis auf Isomorphie, wenn es für jedes andere Pushout-Komplement $A \xrightarrow{g'} D' \xrightarrow{f'} C$ einen eindeutigen \mathbf{C} -Isomorphismus $i: D \rightarrow D'$ gibt, so dass die Gleichungen

$$\begin{aligned} f' \circ i &= f^* \\ i \circ g^* &= g' \end{aligned}$$

gelten.⁹

Definition B.43 (Kontaktbedingung für Graphstrukturen). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, und seien $K \xrightarrow{l} L \xrightarrow{m} G$ zwei Σ -Graphstruktur-Morphismen. Dann erfüllen (l, m) die Kontaktbedingung, wenn für je zwei Sorten $s_1, s_2 \in S$, ein Operationssymbol $op \in OP_{s_2, s_1}$ und je zwei Elemente $x \in L_{s_1}, y \in G_{s_2}$ gilt:

$$m_{s_1}(x) = op^G(y) \wedge y \notin \text{Im } m_{s_2} \Rightarrow x \in \text{Im } l_{s_1}$$

Definition B.44 (Identifikationsbedingung für Graphstrukturen). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, und seien $K \xrightarrow{l} L \xrightarrow{m} G$ zwei Σ -Graphstruktur-Morphismen. Dann erfüllen (l, m) die Identifikationsbedingung, wenn m injektiv bis auf l ist.¹⁰

Lemma B.45 (Existenz des Pushout-Komplements). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, und seien $K \xrightarrow{l} L \xrightarrow{m} G$ zwei Σ -Graphstruktur-Morphismen. Dann existiert ein Pushout-Komplement $K \xrightarrow{k} D \xrightarrow{f} G$ genau dann, wenn (l, m) sowohl die Kontakt- als auch die Identifikationsbedingung erfüllen. \square

Lemma B.46 (Eindeutigkeit des Pushout-Komplements). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, und seien $K \xrightarrow{l} L \xrightarrow{m} G$ zwei Σ -Graphstruktur-Morphismen. Sei ferner $K \xrightarrow{k} D \xrightarrow{f} G$ ein Pushout-Komplement von $K \xrightarrow{l} L \xrightarrow{m} G$. Dann ist das Pushout-Komplement eindeutig bis auf Isomorphie, wenn l injektiv ist. \square

Definition B.47 (DPO-Regel). Sei $\Sigma = (S, OP)$ eine Graphstruktur-Signatur. Dann ist eine DPO-Regel ein Σ -Span $L \xleftarrow{l} K \xrightarrow{r} R$, wobei K die so genannte Klebe- oder Schnittstellen-Graphstruktur darstellt und l und r injektiv sind. \square

Definition B.48 (DPO-Ansatz). Seien $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, G eine Σ -Graphstruktur und $L \xleftarrow{l} K \xrightarrow{r} R$ eine DPO-Regel. Dann ist jeder Σ -Graphstruktur-Morphismus $m: L \rightarrow G$ ein DPO-Ansatz dieser Regel in G . \square

⁹ vgl. auch [49, Kapitel 3]

¹⁰ vgl. Def. B.25

Satz B.49 (DPO-Graphstruktur-Transformation). Seien $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, $p = L \xleftarrow{l} K \xrightarrow{r} R$ eine DPO-Regel und $m: L \rightarrow G$ ein zugehöriger DPO-Ansatz in einer Σ -Graphstruktur G , so dass (l, m) sowohl die Kontakt- als auch die Identifikationsbedingung erfüllen. Dann gibt es eine (p, m) -induzierte Graphstruktur-Transformation (p', k, n) , wobei

- p' ein Σ -Span $G \xleftarrow{f} D \xrightarrow{g} H$ mit den Σ -Graphstrukturen D und H ist und
- $k: K \rightarrow D$ und $n: R \rightarrow H$ Σ -Graphstruktur-Morphismen sind,

so dass $L \xrightarrow{m} G \xleftarrow{f} D$ Pushout von $L \xleftarrow{l} K \xrightarrow{k} D$ und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$ ist (Abb. B.1).

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ m \downarrow & & \downarrow k & & \downarrow n \\ G & \xleftarrow{f} & D & \xrightarrow{g} & H \end{array}$$

Abb. B.1: Eine durch Regelanwendung entstandene Transformation im DPO-Modell

Diese (p, m) -induzierte Graphstruktur-Transformation ist eindeutig in dem Sinne, dass für jede andere (p, m) -induzierte Graphstruktur-Transformation $(G \xleftarrow{f'} D' \xrightarrow{g'} H', k': K \rightarrow D', n': R \rightarrow H')$ zwei $\mathbf{Alg}(\Sigma)$ -Isomorphismen $i^D: D \rightarrow D'$ und $i^H: H \rightarrow H'$ existieren, so dass das gesamte Diagramm kommutiert. \square

B.2.2 DPO-Transformationen von attributierten Graphstrukturen

Definition B.50 (Attributierte DPO-Regel). Seien $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur, L, K und R Σ -Graphstrukturen und X ein Σ^D -Variablenvorrat. Dann ist ein Σ -Span $L \xleftarrow{l} K \xrightarrow{r} R$ eine Σ^D -attributierte DPO-Regel, wenn

- die Σ^D -Redukte von L, K und R die Σ^D -Termalgebra $T^{\Sigma^D}(X)$ sind,
- l und r injektiv sind und
- die Σ^D -Redukte von l und r Identitäten sind.

Definition B.51 (Attributierter DPO-Ansatz). Seien $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur, G eine Σ -Graphstruktur und $L \xleftarrow{l} K \xrightarrow{r} R$ eine Σ^D -attributierte DPO-Regel. Dann ist jeder Σ -Graphstruktur-Morphismus $m: L \rightarrow G$ ein Σ^D -attributierter DPO-Ansatz dieser Regel in G .

Lemma B.52 (Grenzen-Vervollständigung). Sei $\Sigma = (\Sigma^G, \Sigma^D)$ eine wohlstrukturierte Σ^D -attributierte Graphstruktur-Signatur. Sei ein Diagramm in $\mathbf{AGS}(\Sigma)$ wie in Abb. B.2 gegeben, wobei (B, b) als die Grenze von m gemäß Konstruktion in [20, Lemma 11.17] konstruiert sind, d. h.:

- B ist die kleinste Unteralgebra von L , die auf dem Daten-Anteil mit L identisch ist und welche auf dem Graphstruktur-Anteil die Sorten-indizierte Familie von Mengen $(K_s \cup I_s)_{s \in S_G \setminus S_D}$ umfasst, wobei K_s und I_s definiert sind durch:

$$K_s ::= \{ x \in L_s \mid \exists op: s' \rightarrow s, \exists x' \in G_{s'} \setminus m_{s'}(L_{s'}) : m_s(x) = op^G(x') \}$$

$$I_s ::= \{ x \in L_s \mid \exists x' \in L_s, x \neq x' : m_s(x) = m_s(x') \}$$

- b ist die Einbettung der Unteralgebra B in L .

Dann existiert unter der Bedingung, dass (l, m) die Kontakt- und Identifikationsbedingung erfüllen, eine Vervollständigung $b^*: B \rightarrow K$ mit:

$$l \circ b^* = b$$

□

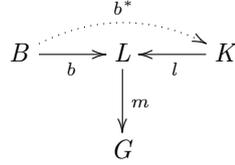


Abb. B.2: Grenzen-Vervollständigung

Satz B.53 (Attributierte DPO-Graphstruktur-Transformation). Seien $\Sigma = (\Sigma^G, \Sigma^D)$ eine wohlstrukturierte Σ^D -attributierte Graphstruktur-Signatur, $p = L \xleftarrow{l} K \xrightarrow{r} R$ eine Σ^D -attributierte DPO-Regel und $m: L \rightarrow G$ ein zugehöriger Σ^D -attribuierter DPO-Ansatz in einer Σ -Graphstruktur G , so dass (l, m) sowohl die Kontakt- als auch die Identifikationsbedingung erfüllen. Dann gibt es eine (p, m) -induzierte Σ^D -attributierte Graphstruktur-Transformation (p', k, n) , wobei

- p' ein Σ -Span $G \xleftarrow{f} D \xrightarrow{g} H$ mit den Σ -Graphstrukturen D und H ist und
- $k: K \rightarrow D$ und $n: R \rightarrow H$ Σ -Graphstruktur-Morphismen sind,

so dass $L \xrightarrow{m} G \xleftarrow{f} D$ Pushout von $L \xleftarrow{l} K \xrightarrow{k} D$ und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{k} K \xrightarrow{r} R$ ist (Abb. B.1).

Diese (p, m) -induzierte Σ^D -attributierte Graphstruktur-Transformation ist eindeutig in dem Sinne, dass für jede andere (p, m) -induzierte Σ^D -attributierte Graphstruktur-Transformation $(G \xleftarrow{f'} D' \xrightarrow{g'} H', k': K \rightarrow D', n': R \rightarrow H')$ zwei $\mathbf{Alg}(\Sigma)$ -Isomorphismen $i^D: D \rightarrow D'$ und $i^H: H \rightarrow H'$ existieren, so dass das gesamte Diagramm kommutiert. □

B.2.3 DPO-Transformationen von typisierten Graphstrukturen

Definition B.54 (Typisierte DPO-Regel). Seien $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, T eine Σ -Typ-Graphstruktur und $L = (L^\#, type_L)$, $K = (K^\#, type_K)$ und $R = (R^\#, type_R)$ T -typisierte Σ -Graphstrukturen. Dann ist ein T -typisierter Σ -Span $L \xleftarrow{l} K \xrightarrow{r} R$ eine T -typisierte DPO-Regel, wenn $L^\# \xleftarrow{l} K^\# \xrightarrow{r} R^\#$ eine DPO-Regel ist.

Definition B.55 (Typisierter DPO-Ansatz). Seien $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, T eine Σ -Typ-Graphstruktur, $G = (G^\#, type_G)$, $L = (L^\#, type_L)$, $K = (K^\#, type_K)$ und $R = (R^\#, type_R)$ T -typisierte Σ -Graphstrukturen und $L \xleftarrow{l} K \xrightarrow{r} R$ eine T -typisierte DPO-Regel. Dann ist ein T -typisierter Σ -Graphstruktur-Morphismus $m: L \rightarrow G$ ein T -typisierter DPO-Ansatz der Regel in G , wenn m ein DPO-Ansatz der DPO-Regel $L^\# \xleftarrow{l} K^\# \xrightarrow{r} R^\#$ in $G^\#$ ist.

Satz B.56 (Typisierte DPO-Graphstruktur-Transformation). Seien $\Sigma = (S, OP)$ eine Graphstruktur-Signatur, T eine Σ -Typ-Graphstruktur, $G = (G^\#, type_G)$, $L = (L^\#, type_L)$, $K = (K^\#, type_K)$ und $R = (R^\#, type_R)$ T -typisierte Σ -Graphstrukturen, $p = L \xleftarrow{l} K \xrightarrow{r} R$ eine T -typisierte DPO-Regel und $m: L \rightarrow G$ ein zugehöriger T -typisierter DPO-Ansatz in G , so dass (l, m) sowohl die Kontakt- als auch die Identifikationsbedingung erfüllen. Dann gibt es eine (p, m) -induzierte T -typisierte Graphstruktur-Transformation (p', k, n) , wobei

- p' ein T -typisierter Σ -Span $G \xleftarrow{f} D \xrightarrow{g} H$ mit den T -typisierten Σ -Graphstrukturen $D = (D^\#, type_D)$ und $H = (H^\#, type_H)$ ist und
- $k: K \rightarrow D$ und $n: R \rightarrow H$ T -typisierte Σ -Graphstruktur-Morphismen sind,

so dass $L \xrightarrow{m} G \xleftarrow{f} D$ Pushout von $L \xleftarrow{l} K \xrightarrow{k} D$ und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{l} K \xrightarrow{r} R$ ist.

Diese (p, m) -induzierte T -typisierte Graphstruktur-Transformation ist eindeutig in dem Sinne, dass für jede andere (p, m) -induzierte T -typisierte Graphstruktur-Transformation $(G \xleftarrow{f'} D' \xrightarrow{g'} H', k': K \rightarrow D', n': R \rightarrow H')$ zwei $\mathbf{Alg}(\Sigma)$ -Isomorphismen $i^D: D \rightarrow D'$ und $i^H: H \rightarrow H'$ existieren, so dass das gesamte Diagramm kommutiert. \square

Definition B.57 (Typisierte attributierte DPO-Regel). Seien $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur, T eine Σ -Typ-Graphstruktur und $L = (L^\#, type_L)$, $K = (K^\#, type_K)$ und $R = (R^\#, type_R)$ T -typisierte Σ -Graphstrukturen. Dann ist ein T -typisierter Σ -Span $L \xleftarrow{l} K \xrightarrow{r} R$ eine T -typisierte Σ^D -attributierte DPO-Regel, wenn $L^\# \xleftarrow{l} K^\# \xrightarrow{r} R^\#$ eine Σ^D -attributierte DPO-Regel ist.

Definition B.58 (Typisierter attributierter DPO-Ansatz). Seien $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur, T eine Σ -Typ-Graphstruktur, $G = (G^\#, type_G)$,

$L = (L^\#, \text{type}_L)$, $K = (K^\#, \text{type}_K)$ und $R = (R^\#, \text{type}_R)$ T -typisierte Σ -Graphstrukturen und $L \xleftarrow{l} K \xrightarrow{r} R$ eine T -typisierte Σ^D -attributierte DPO-Regel. Dann ist ein T -typisierter Σ -Graphstruktur-Morphismus $m: L \rightarrow G$ ein T -typisierter Σ^D -attributierter DPO-Ansatz dieser Regel in G , wenn m ein Σ^D -attributierter DPO-Ansatz der Σ^D -attributierten DPO-Regel $L^\# \xleftarrow{l} K^\# \xrightarrow{r} R^\#$ in $G^\#$ ist.

Satz B.59 (Typisierte attributierte DPO-Graphstruktur-Transformation). Seien $\Sigma = (\Sigma^G, \Sigma^D)$ eine Σ^D -attributierte Graphstruktur-Signatur, T eine Σ -Typ-Graphstruktur, $G = (G^\#, \text{type}_G)$, $L = (L^\#, \text{type}_L)$, $K = (K^\#, \text{type}_K)$ und $R = (R^\#, \text{type}_R)$ T -typisierte Σ -Graphstrukturen, $p = L \xleftarrow{l} K \xrightarrow{r} R$ eine T -typisierte Σ^D -attributierte DPO-Regel und $m: L \rightarrow G$ ein zugehöriger T -typisierter Σ^D -attributierter DPO-Ansatz in G , so dass (l, m) sowohl die Kontakt- als auch die Identifikationsbedingung erfüllen. Dann gibt es eine (p, m) -induzierte T -typisierte Σ^D -attributierte Graphstruktur-Transformation (p', k, n) , wobei

- p' ein T -typisierter Σ -Span $G \xleftarrow{f} D \xrightarrow{g} H$ mit den T -typisierten Σ -Graphstrukturen $D = (D^\#, \text{type}_D)$ und $H = (H^\#, \text{type}_H)$ ist und
- $k: K \rightarrow D$ und $n: R \rightarrow H$ T -typisierte Σ -Graphstruktur-Morphismen sind,

so dass $L \xrightarrow{m} G \xleftarrow{f} D$ Pushout von $L \xleftarrow{l} K \xrightarrow{k} D$ und $D \xrightarrow{g} H \xleftarrow{n} R$ Pushout von $D \xleftarrow{l} K \xrightarrow{r} R$ ist.

Diese (p, m) -induzierte T -typisierte Σ^D -attributierte Graphstruktur-Transformation ist eindeutig in dem Sinne, dass für jede andere (p, m) -induzierte T -typisierte Σ^D -attributierte Graphstruktur-Transformation $(G \xleftarrow{f'} D' \xrightarrow{g'} H', k': K \rightarrow D', n': R \rightarrow H')$ zwei $\mathbf{Alg}(\Sigma)$ -Isomorphismen $i^D: D \rightarrow D'$ und $i^H: H \rightarrow H'$ existieren, so dass das gesamte Diagramm kommutiert. \square

Beweise

In diesem Kapitel sind alle längeren Beweise enthalten. Das Kapitel gliedert sich in drei Abschnitte, die den Kapiteln 9 bis 11 entsprechen.

C.1 Beweise aus Kapitel 9

C.1.1 Beweis von Lemma 9.12

Seien im Folgenden die M' -Axiome (M.1) bis (M.9) Typisierungs-Axiome genannt. Nach Konstruktion A.92 ergibt sich das resultierende M'' -System durch schrittweise Annäherung des Ausgangssystems D an die (eventuell) nicht gültigen M'' -Axiome. Es wird nun mittels Induktion über i gezeigt, dass die Konstruktion der Epireflexion in jedem Schritt die Gültigkeit aller Typisierungs-Axiome erhält.

Induktionsbeginn $i = 0$: Es gilt $D^0 = D$. Die Typisierungs-Axiome sind in D gültig, weil sie Teil der Spezifikation M' sind und D nach Voraussetzung ein M' -System ist.

Induktionsvoraussetzung: Die Typisierungs-Axiome gelten in D^i .

Induktionsschritt $i \mapsto i + 1$: Hier wird nach den einzelnen Axiomen unterschieden:

- (1) *Axiome (M.1) bis (M.6).* Diese Axiome stellen Gleichungen dar. Die Gültigkeit von Gleichungen wird durch surjektive Homomorphismen übertragen [25, Fakt 3.1 (5)]. Weil der Homomorphismus $[\]_{\equiv i} : D^i \rightarrow D^{i+1}$ nach Konstruktion A.92 surjektiv ist, gelten die Gleichungen in D^{i+1} .
- (2) *Axiom (M.7).* Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$[x]_{\equiv i} \in \text{message}^{D^{i+1}} \Rightarrow \text{typeO}^{D^{i+1}}([x]_{\equiv i}) \in \text{operation}^{D^{i+1}}$$

Sei also ein Element $[x]_{\equiv i} \in D^{i+1}$ gegeben, und gelte die Prämisse $[x]_{\equiv i} \in \text{message}^{D^{i+1}}$. Nach Konstruktion A.41 existiert ein $x' \in D^i$ mit:

$$(C.1) \quad [x]_{\equiv_o^i} = [x']_{\equiv_o^i}$$

$$(C.2) \quad x' \in \text{message}^{D^i} \cup \Delta \text{message}^{D^i}$$

Weil kein M'' -Axiom in der Konklusion das Prädikat *message* wahr macht, folgt $\Delta \text{message}^{D^i} = \emptyset$. Aus (C.2) folgt somit:

$$(C.3) \quad x' \in \text{message}^{D^i}$$

Aus (C.3) folgt, dass die Variablenbelegung $x \mapsto x'$ die Prämisse des Axioms (M.7) in D^i erfüllt. Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.4) \quad \text{type}O^{D^i}(x') \in \text{operation}^{D^i}$$

Daraus folgt schließlich:

$$\begin{aligned} \text{type}O^{D^i}(x') \in \text{operation}^{D^i} & & (C.4) \\ \Rightarrow [\text{type}O^{D^i}(x')]_{\equiv_c^i} \in \text{operation}^{D^{i+1}} & & ([\]_{\equiv_c^i} \text{ ist Homomorphismus}) \\ \Rightarrow \text{type}O^{D^{i+1}}([x']_{\equiv_o^i}) \in \text{operation}^{D^{i+1}} & & ([\]_{\equiv_c^i} \text{ ist Homomorphismus}) \\ \Rightarrow \text{type}O^{D^{i+1}}([x]_{\equiv_o^i}) \in \text{operation}^{D^{i+1}} & & (C.1) \end{aligned}$$

(3) *Axiom* (M.8). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$([x]_{\equiv_o^i}, [y]_{\equiv_o^i}) \in \text{under}O^{D^{i+1}} \Rightarrow (\text{type}O^{D^{i+1}}([x]_{\equiv_o^i}), \text{type}O^{D^{i+1}}([y]_{\equiv_o^i})) \in \text{under}C^{D^{i+1}}$$

Seien also zwei Elemente $[x]_{\equiv_o^i} \in D_O^{i+1}$ und $[y]_{\equiv_o^i} \in D_O^{i+1}$ gegeben, und gelte die Prämisse $([x]_{\equiv_o^i}, [y]_{\equiv_o^i}) \in \text{under}O^{D^{i+1}}$. Nach Konstruktion A.41 existieren zwei Elemente $x', y' \in D_O^i$ mit:

$$(C.5) \quad [x]_{\equiv_o^i} = [x']_{\equiv_o^i}$$

$$(C.6) \quad [y]_{\equiv_o^i} = [y']_{\equiv_o^i}$$

$$(C.7) \quad (x', y') \in \text{under}O^{D^i} \cup \Delta \text{under}O^{D^i}$$

Nach (C.7) sind nun zwei Fälle zu unterscheiden:

- (a) $(x', y') \in \text{under}O^{D^i}$. In diesem Fall wird die Wahrheit der Konklusion analog zum Nachweis der Gültigkeit des Axioms (M.7) in Punkt 2 nachgewiesen.
- (b) $(x', y') \in \Delta \text{under}O^{D^i}$. Nun entstehen alle Elemente in $\Delta \text{under}O^{D^i}$ durch eine passende Variablenbelegung der Variablen in Axiom (M.21), weil dieses Axiom als einziges

$M^?$ -Axiom das Prädikat $underO$ in der Konklusion wahr macht. Somit existiert nach Konstruktion A.92 ein Element $z \in D^i$ mit:

$$(C.8) \quad (x', z) \in underO^{D^i}$$

$$(C.9) \quad (z, y') \in underO^{D^i}$$

Nach (C.8) und (C.9) erfüllen die Variablenbelegungen

$$x \mapsto x'$$

$$y \mapsto z$$

und

$$x \mapsto z$$

$$y \mapsto y'$$

jeweils die Prämisse des Axioms (M.8) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusionen:

$$(C.10) \quad (typeO^{D^i}(x'), typeO^{D^i}(z)) \in underC^{D^i}$$

$$(C.11) \quad (typeO^{D^i}(z), typeO^{D^i}(y')) \in underC^{D^i}$$

Nach (C.10) und (C.11) erfüllt die Variablenbelegung

$$x \mapsto typeO^{D^i}(x')$$

$$y \mapsto typeO^{D^i}(z)$$

$$z \mapsto typeO^{D^i}(y')$$

die Prämisse des Axioms (M.18) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.12) \quad (typeO^{D^i}(x'), typeO^{D^i}(y')) \in underC^{D^i}$$

Daraus folgt schließlich:

$$(typeO^{D^i}(x'), typeO^{D^i}(y')) \in underC^{D^i} \quad (C.12)$$

$$\Rightarrow ([typeO^{D^i}(x')]_{\equiv_c}, [typeO^{D^i}(y')]_{\equiv_c}) \in underC^{D^{i+1}} \quad ([]_{\equiv_c} \text{ ist Homomorphismus})$$

$$\begin{aligned} &\Rightarrow (\text{type}O^{D^{i+1}}([x']_{\equiv_i}), \text{type}O^{D^{i+1}}([y']_{\equiv_i})) \in \text{under}C^{D^{i+1}} \quad ([\]_{\equiv_i} \text{ ist Homomorphismus}) \\ &\Rightarrow (\text{type}O^{D^{i+1}}([x]_{\equiv_i}), \text{type}O^{D^{i+1}}([y]_{\equiv_i})) \in \text{under}C^{D^{i+1}} \quad (\text{C.5) und (C.6)} \end{aligned}$$

(4) *Axiom* (M.9). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$([x]_{\equiv_i}, [y]_{\equiv_i}) \in \text{rel}O^{D^{i+1}} \Rightarrow (\text{type}O^{D^{i+1}}([x]_{\equiv_i}), \text{type}O^{D^{i+1}}([y]_{\equiv_i})) \in \text{rel}C^{D^{i+1}}$$

Seien also zwei Elemente $[x]_{\equiv_i} \in D_O^{i+1}$ und $[y]_{\equiv_i} \in D_O^{i+1}$ gegeben, und gelte die Prämisse $([x]_{\equiv_i}, [y]_{\equiv_i}) \in \text{rel}O^{D^{i+1}}$. Nach Konstruktion A.41 existieren zwei Elemente $x', y' \in D_O^i$ mit:

$$(C.13) \quad [x]_{\equiv_i} = [x']_{\equiv_i}$$

$$(C.14) \quad [y]_{\equiv_i} = [y']_{\equiv_i}$$

$$(C.15) \quad (x', y') \in \text{rel}O^{D^i} \cup \Delta\text{rel}O^{D^i}$$

Nach (C.15) sind nun zwei Fälle zu unterscheiden:

- (a) $(x', y') \in \text{rel}O^{D^i}$. In diesem Fall wird die Wahrheit der Konklusion analog zum Nachweis der Gültigkeit des Axioms (M.7) in Punkt 2 nachgewiesen.
- (b) $(x', y') \in \Delta\text{rel}O^{D^i}$. Nun entstehen alle Elemente in $\Delta\text{rel}O^{D^i}$ durch eine passende Variablenbelegung der Variablen in Axiom (M.25), weil dieses Axiom als einziges M'' -Axiom das Prädikat $\text{rel}O$ in der Konklusion wahr macht. Somit existiert nach Konstruktion A.92 ein Element $z \in D^i$ mit:

$$(C.16) \quad (x', z) \in \text{rel}O^{D^i}$$

$$(C.17) \quad (z, y') \in \text{rel}O^{D^i}$$

Nach (C.16) und (C.17) erfüllen die Variablenbelegungen

$$x \mapsto x'$$

$$y \mapsto z$$

und

$$x \mapsto z$$

$$y \mapsto y'$$

jeweils die Prämisse des Axioms (M.9) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusionen:

$$(C.18) \quad (typeO^{D^i}(x'), typeO^{D^i}(z)) \in relC^{D^i}$$

$$(C.19) \quad (typeO^{D^i}(z), typeO^{D^i}(y')) \in relC^{D^i}$$

Nach (C.18) und (C.19) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto typeO^{D^i}(x') \\ y &\mapsto typeO^{D^i}(z) \\ z &\mapsto typeO^{D^i}(y') \end{aligned}$$

die Prämisse des Axioms (M.23) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.20) \quad (typeO^{D^i}(x'), typeO^{D^i}(y')) \in relC^{D^i}$$

Daraus folgt schließlich:

$$\begin{aligned} (typeO^{D^i}(x'), typeO^{D^i}(y')) &\in relC^{D^i} && (C.20) \\ \Rightarrow ([typeO^{D^i}(x')]_{\equiv_c}, [typeO^{D^i}(y')]_{\equiv_c}) &\in relC^{D^{i+1}} && ([]_{\equiv_c} \text{ ist Homomorphismus}) \\ \Rightarrow (typeO^{D^{i+1}}([x']_{\equiv_c}), typeO^{D^{i+1}}([y']_{\equiv_c})) &\in relC^{D^{i+1}} && ([]_{\equiv_c} \text{ ist Homomorphismus}) \\ \Rightarrow (typeO^{D^{i+1}}([x]_{\equiv_o}), typeO^{D^{i+1}}([y]_{\equiv_o})) &\in relC^{D^{i+1}} && (C.13) \text{ und } (C.14) \end{aligned}$$

□

C.1.2 Beweis von Lemma 9.13

Es wird für alle $i \in \mathbb{N}_0$ gezeigt, dass

$$(C.21) \quad p^{D^{i+1}} = p^{D^i}$$

für $p \in \{=C, =A, =CA_t, =SV_t, operation, underC, relC\}$ und alle $t \in TYPES$ gilt. Aus Konstruktion A.92 ergibt sich dann die Gültigkeit der zu beweisenden Behauptung.

Zunächst gilt $\Delta_s^{D^i} = \emptyset$ für $s \in \{C, A, CA_t, SV_t\}$ und alle $i \in \mathbb{N}_0$, weil kein M'' -Axiom in der Konklusion Elemente aus Trägermengen zu den Schema-Sorten zusammenlegt. Es bleibt somit der Kongruenzabschluss der Relationenfamilie $(=s^{D^i} \cup \Delta_s^{D^i})_{s \in S}$ zu untersuchen. Dies wird nur für die beiden Axiome (M.28) und (M.29) durchgeführt, weil diese Axiome die einzigen M'' -Axiome sind, die in der Konklusion Elemente gleichsetzen und sich somit auf die Kongruenzrelation auswirken. Es wird gezeigt, dass kein Paar in den Relationen $\Delta_O^{D^i}$ und $\Delta_L^{D^i}$ sich auf die Gleichheitsrelationen zu den Schema-Sorten auswirkt.

- (1) $(x, y) \in \Delta =_{\mathcal{O}}^{D^i}$. Dieses Paar kann nur durch die Anwendung des Axioms (M.28) hinzugefügt worden sein. Somit erfüllen die Elemente x und y die Prämisse des Axioms (M.28), es gilt somit insbesondere:

$$(C.22) \quad typeO^{D^i}(x) = typeO^{D^i}(y)$$

Aus (C.22) folgt:

$$(C.23) \quad selfC^{D^i}(typeO^{D^i}(x)) = selfC^{D^i}(typeO^{D^i}(y))$$

$$(C.24) \quad initCA_t^{D^i}(typeO^{D^i}(x)) = initCA_t^{D^i}(typeO^{D^i}(y))$$

$$(C.25) \quad targetCA_t^{D^i}(initCA_t^{D^i}(typeO^{D^i}(x))) = targetCA_t^{D^i}(initCA_t^{D^i}(typeO^{D^i}(y)))$$

Für den Kongruenzabschluss sind vier Wege von den Elementen x und y in das Schema zu untersuchen: $O \xrightarrow{typeO} C$, $O \xrightarrow{selfO} L \xrightarrow{typeL} A$, $O \xrightarrow{initOA_t} OA_t \xrightarrow{typeOA_t} CA_t$ und $O \xrightarrow{initOA_t} OA_t \xrightarrow{targetOA_t} IV_t \xrightarrow{typeV_t} SV_t$. Die Präfixe $O \xrightarrow{selfO} L \xrightarrow{sourceL} O$, $O \xrightarrow{selfO} L \xrightarrow{targetL} O$ und $O \xrightarrow{initOA_t} OA_t \xrightarrow{sourceOA_t} O$, die den genannten Wegen beliebig oft vorangestellt werden können, dürfen ignoriert werden, weil die M -Axiome (M.13), (M.14) und (M.15) Gleichungen darstellen, deren Gültigkeit durch die surjektiven natürlichen Homomorphismen gemäß [25, Fakt 3.1 (5)] übertragen wird, und somit in D^i gelten.

- (a) $O \xrightarrow{typeO} C$. Nach Voraussetzung gilt $(x, y) \in \Delta =_{\mathcal{O}}^{D^i}$. Auf Grund des Operationsabschlusses folgt daraus:

$$(typeO^{D^i}(x), typeO^{D^i}(y)) \in \equiv_C^i$$

Nach (C.22) ist $(typeO^{D^i}(x), typeO^{D^i}(y))$ jedoch bereits Element der Relation $=_{\mathcal{C}}^{D^i}$.

- (b) $O \xrightarrow{selfO} L \xrightarrow{typeL} A$. Nach Voraussetzung gilt $(x, y) \in \Delta =_{\mathcal{O}}^{D^i}$. Auf Grund des Operationsabschlusses folgt daraus:

$$(C.26) \quad (typeL^{D^i}(selfO^{D^i}(x)), typeL^{D^i}(selfO^{D^i}(y))) \in \equiv_A^i$$

Nach Lemma 9.12 gilt Axiom (M.6) in D^i . Somit kann (C.26) ersetzt werden durch:

$$(selfC^{D^i}(typeO^{D^i}(x)), selfC^{D^i}(typeO^{D^i}(y))) \in \equiv_A^i$$

Nach (C.23) ist $(selfC^{D^i}(typeO^{D^i}(x)), selfC^{D^i}(typeO^{D^i}(y)))$ jedoch bereits Element der Relation $=_A^{D^i}$.

- (c) $O \xrightarrow{initOA_t} OA_t \xrightarrow{typeOA_t} CA_t$. Nach Voraussetzung gilt $(x, y) \in \Delta =_{\mathcal{O}}^{D^i}$. Auf Grund des Operationsabschlusses folgt daraus:

$$(C.27) \quad (typeOA_t^{D^i}(initOA_t^{D^i}(x)), typeOA_t^{D^i}(initOA_t^{D^i}(y))) \in \equiv_{CA_t}^i$$

Nach Lemma 9.12 gilt Axiom (M.5) in D^i . Somit kann (C.27) ersetzt werden durch:

$$(initCA_t^{D^i}(typeO^{D^i}(x)), initCA_t^{D^i}(typeO^{D^i}(y))) \in \equiv_{CA_t}^i$$

Nach (C.24) ist $(initCA_t^{D^i}(typeO^{D^i}(x)), initCA_t^{D^i}(typeO^{D^i}(y)))$ jedoch bereits Element der Relation $=_{CA_t}^{D^i}$.

- (d) $O \xrightarrow{initOA_t} OA_t \xrightarrow{targetOA_t} IV_t \xrightarrow{typeV_t} SV_t$. Nach Voraussetzung gilt $(x, y) \in \Delta =_O^{D^i}$. Auf Grund des Operationsabschlusses folgt daraus:

(C.28)

$$(typeV_t^{D^i}(targetOA_t^{D^i}(initOA_t^{D^i}(x))), typeV_t^{D^i}(targetOA_t^{D^i}(initOA_t^{D^i}(y)))) \in \equiv_{SV_t}^i$$

Nach Lemma 9.12 gelten Axiom (M.4) und Axiom (M.5) in D^i . Somit kann (C.28) ersetzt werden durch:

$$(targetCA_t^{D^i}(initCA_t^{D^i}(typeO^{D^i}(x))), targetCA_t^{D^i}(initCA_t^{D^i}(typeO^{D^i}(y)))) \in \equiv_{SV_t}^i$$

Das Paar $(targetCA_t^{D^i}(initCA_t^{D^i}(typeO^{D^i}(x))), targetCA_t^{D^i}(initCA_t^{D^i}(typeO^{D^i}(y))))$ ist nach (C.25) jedoch bereits Element der Relation $=_{SV_t}^{D^i}$.

- (2) $(x, y) \in \Delta =_L^{D^i}$. Dieses Paar kann nur durch die Anwendung des Axioms (M.29) hinzugefügt worden sein. Somit erfüllen die Elemente x und y die Prämisse des Axioms (M.29), es gilt somit:

$$(C.29) \quad sourceL^{D^i}(x) = sourceL^{D^i}(y)$$

$$(C.30) \quad typeL^{D^i}(x) = typeL^{D^i}(y)$$

Aus (C.30) folgt:

$$(C.31) \quad targetA^{D^i}(typeL^{D^i}(x)) = targetA^{D^i}(typeL^{D^i}(y))$$

Nach Lemma 9.12 gilt Axiom (M.2) in D^i . Somit folgt aus (C.31):

$$(C.32) \quad typeO^{D^i}(targetL^{D^i}(x)) = typeO^{D^i}(targetL^{D^i}(y))$$

Es gibt drei Operationen, die auf der Trägermenge zur Sorte L definiert sind:

- (a) *Operationssymbol* $typeL$. Auf Grund des Operationsabschlusses gilt:

$$(x, y) \in \Delta =_L^{D^i} \Rightarrow (typeL^{D^i}(x), typeL^{D^i}(y)) \in \equiv_A^i$$

Nach (C.30) ist $(typeL^{D^i}(x), typeL^{D^i}(y))$ jedoch bereits Element der Relation $=_A^{D^i}$.

(b) *Operationssymbol sourceL*. Auf Grund des Operationsabschlusses gilt:

$$(x, y) \in \Delta =_L^{D^i} \Rightarrow (\text{source}L^{D^i}(x), \text{source}L^{D^i}(y)) \in \equiv_O^i$$

Nach (C.29) ist $(\text{source}L^{D^i}(x), \text{source}L^{D^i}(y))$ jedoch bereits Element der Relation $=_{SV_t}^{D^i}$.

(c) *Operationssymbol targetL*. Auf Grund des Operationsabschlusses gilt:

$$(C.33) \quad (x, y) \in \Delta =_L^{D^i} \Rightarrow (\text{target}L^{D^i}(x), \text{target}L^{D^i}(y)) \in \equiv_O^i$$

Analog zu Punkt (1) sind die vier Wege $O \xrightarrow{\text{type}O} C$, $O \xrightarrow{\text{self}O} L \xrightarrow{\text{type}L} A$, $O \xrightarrow{\text{init}OA_t} OA_t \xrightarrow{\text{type}OA_t} CA_t$ und $O \xrightarrow{\text{init}OA_t} OA_t \xrightarrow{\text{target}OA_t} IV_t \xrightarrow{\text{type}V_t} SV_t$ von den Elementen $\text{target}L^{D^i}(x)$ und $\text{target}L^{D^i}(y)$ in das Schema zu untersuchen. Dies wird genauso wie in Punkt (1) unter Zuhilfenahme von (C.32) bewiesen.

Somit folgt $\equiv_p^i = p^{D^i}$ für $p \in \{=C, =A, =CA_t\}$ und alle $t \in TYPES$. Weiter gilt $\equiv_{=SV_t}^i = =_{SV_t}^{D^i}$, da $\Delta =_{SV_t}^{D^i} = \emptyset$ und weil der Kongruenzabschluss dieser Relation keine neuen Paare hinzufügt. Schließlich gilt $\Delta p^{D^i} = \emptyset$ für $p \in \{\text{operation}, \text{under}C, \text{rel}C\}$, weil kein M' -Axiom diese Prädikate in der Konklusion wahr macht. Daraus folgt $\equiv_p^i = p^{D^i}$ und somit $p^{D^{i+1}} = p^{D^i}$ für $p \in \{=C, =A, =CA_t, =SV_t, \text{operation}, \text{under}C, \text{rel}C\}$ und alle $t \in TYPES$, w. z. z. w. \square

C.1.3 Beweis von Lemma 9.14

Seien im Folgenden die M' -Axiome (M.10) bis (M.15) Kanten-Axiome, (M.16) bis (M.20) Vererbungs-Axiome und die M' -Axiome (M.22) bis (M.24) und (M.26) bis (M.27) Komponenten-Axiome genannt. Nach Konstruktion A.92 ergibt sich das resultierende M'' -System durch schrittweise Annäherung des Ausgangssystems D an die (eventuell) nicht gültigen M'' -Axiome. Es wird nun mittels Induktion über i gezeigt, dass die Konstruktion der Epireflexion in jedem Schritt die Gültigkeit aller Kanten-, Vererbungs- und Komponenten-Axiome erhält.

Induktionsbeginn $i = 0$: Es gilt $D^0 = D$. Die Kanten-, Vererbungs- und Komponenten-Axiome sind in D gültig, weil sie Teil der Spezifikation M' sind und D nach Voraussetzung ein M' -System ist.

Induktionsvoraussetzung: Die Kanten-, Vererbungs- und Komponenten-Axiome gelten in D^i .

Induktionsschritt $i \mapsto i + 1$: Hier wird nach den einzelnen Axiomen unterschieden:

- (1) *Axiome (M.10) bis (M.15)*. Diese Axiome stellen Gleichungen dar. Sie gelten in D^{i+1} , weil der Homomorphismus $[\]_{\equiv^i} : D^i \rightarrow D^{i+1}$ nach Konstruktion A.92 surjektiv ist und die Gültigkeit von Gleichungen durch surjektive Homomorphismen übertragen wird [25, Fakt 3.1 (5)].

(2) *Axiom* (M.16). Zu zeigen ist

$$(C.34) \quad ([x]_{\equiv_C^i}, [x]_{\equiv_C^i}) \in \text{under}C^{D^{i+1}}$$

für alle $[x]_{\equiv_C^i} \in D_C^{i+1}$. Sei also ein Element $[x]_{\equiv_C^i} \in D_C^{i+1}$ gegeben. Offensichtlich gilt $x \in D_C^i$. Nach der Induktionsvoraussetzung gilt Axiom (M.16) in D^i . Daraus folgt:

$$(C.35) \quad (x, x) \in \text{under}C^{D^i}$$

Weil $[\]_{\equiv_C^i}$ als Homomorphismus Relationen überträgt, folgt aus (C.35) die Wahrheit von (C.34).

(3) *Axiom* (M.17). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.36) \quad ([x]_{\equiv_C^i}, [y]_{\equiv_C^i}) \in \text{under}C^{D^{i+1}} \wedge ([y]_{\equiv_C^i}, [x]_{\equiv_C^i}) \in \text{under}C^{D^{i+1}} \Rightarrow [x]_{\equiv_C^i} = [y]_{\equiv_C^i}$$

für alle $[x]_{\equiv_C^i}, [y]_{\equiv_C^i} \in D_C^{i+1}$. Gelte also die Prämisse von (C.36). Nach Lemma 9.13 stimmen die Schemata von D^i und D^{i+1} überein. Somit gilt $x, y \in D_C^i$ mit:

$$(C.37) \quad (x, y) \in \text{under}C^{D^i}$$

$$(C.38) \quad (y, x) \in \text{under}C^{D^i}$$

Nach (C.37) und (C.38) erfüllt die Variablenbelegung

$$x \mapsto x$$

$$y \mapsto y$$

die Prämisse des Axioms (M.17) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.39) \quad x = y$$

Weil $[\]_{\equiv_C^i}$ als Homomorphismus Relationen überträgt, folgt aus (C.39) die Wahrheit der Konklusion von (C.36).

(4) *Axiom* (M.18). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.40) \quad ([x]_{\equiv_C^i}, [y]_{\equiv_C^i}) \in \text{under}C^{D^{i+1}} \wedge ([y]_{\equiv_C^i}, [z]_{\equiv_C^i}) \in \text{under}C^{D^{i+1}} \\ \Rightarrow ([x]_{\equiv_C^i}, [z]_{\equiv_C^i}) \in \text{under}C^{D^{i+1}}$$

für alle $[x]_{\equiv_C^i}, [y]_{\equiv_C^i}, [z]_{\equiv_C^i} \in D_C^{i+1}$. Gelte also die Prämisse von (C.40). Nach Lemma 9.13 stimmen die Schemata von D^i und D^{i+1} überein. Somit gilt $x, y, z \in D_C^i$ mit:

$$(C.41) \quad (x, y) \in \text{under}C^{D^i}$$

$$(C.42) \quad (y, z) \in \text{under}C^{D^i}$$

Nach (C.41) und (C.42) erfüllt die Variablenbelegung

$$x \mapsto x$$

$$y \mapsto y$$

$$z \mapsto z$$

die Prämisse des Axioms (M.18) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.43) \quad (x, z) \in \text{under}C^{D^i}$$

Weil $[\]_{\equiv_i}$ als Homomorphismus Relationen überträgt, folgt aus (C.43) die Wahrheit der Konklusion von (C.40).

(5) *Axiom* (M.19). Zu zeigen ist

$$(C.44) \quad ([x]_{\equiv_o^i}, [x]_{\equiv_o^i}) \in \text{under}O^{D^{i+1}}$$

für alle $[x]_{\equiv_o^i} \in D_O^{i+1}$. Sei also ein Element $[x]_{\equiv_o^i} \in D_O^{i+1}$ gegeben. Offensichtlich gilt $x \in D_O^i$. Nach der Induktionsvoraussetzung gilt Axiom (M.19) in D^i . Daraus folgt:

$$(C.45) \quad (x, x) \in \text{under}O^{D^i}$$

Weil $[\]_{\equiv_i}$ als Homomorphismus Relationen überträgt, folgt aus (C.45) die Wahrheit von (C.44).

(6) *Axiom* (M.20). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.46) \quad ([x]_{\equiv_o^i}, [y]_{\equiv_o^i}) \in \text{under}O^{D^{i+1}} \wedge ([y]_{\equiv_o^i}, [x]_{\equiv_o^i}) \in \text{under}O^{D^{i+1}} \Rightarrow [x]_{\equiv_o^i} = [y]_{\equiv_o^i}$$

für alle $[x]_{\equiv_o^i}, [y]_{\equiv_o^i} \in D_O^{i+1}$. Gelte also die Prämisse von (C.46). Daraus folgt, dass die Variablenbelegungen

$$x \mapsto [x]_{\equiv_o^i}$$

$$y \mapsto [y]_{\equiv_o^i}$$

und

$$\begin{aligned} x &\mapsto [y]_{\equiv_o^i} \\ y &\mapsto [x]_{\equiv_o^i} \end{aligned}$$

jeweils die Prämisse des Axioms (M.8) in D^{i+1} erfüllen. Weil das Axiom nach Lemma 9.12 in D^{i+1} gilt, folgt die Wahrheit der Konklusionen:

$$(C.47) \quad (typeO^{D^{i+1}}([x]_{\equiv_o^i}), typeO^{D^{i+1}}([y]_{\equiv_o^i})) \in \text{under}C^{D^{i+1}}$$

$$(C.48) \quad (typeO^{D^{i+1}}([y]_{\equiv_o^i}), typeO^{D^{i+1}}([x]_{\equiv_o^i})) \in \text{under}C^{D^{i+1}}$$

Nach (C.47) und (C.48) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto typeO^{D^{i+1}}([x]_{\equiv_o^i}) \\ y &\mapsto typeO^{D^{i+1}}([y]_{\equiv_o^i}) \end{aligned}$$

die Prämisse des Axioms (M.17) in D^{i+1} . Weil das Axiom nach Punkt 3 in D^{i+1} gilt, folgt die Wahrheit der Konklusion:

$$(C.49) \quad typeO^{D^{i+1}}([x]_{\equiv_o^i}) = typeO^{D^{i+1}}([y]_{\equiv_o^i})$$

Weiter gilt nach Konstruktion A.41, dass vier Elemente x', y', x'' und $y'' \in D_o^i$ existieren mit:

$$(C.50) \quad [x]_{\equiv_o^i} = [x']_{\equiv_o^i} = [x'']_{\equiv_o^i}$$

$$(C.51) \quad [y]_{\equiv_o^i} = [y']_{\equiv_o^i} = [y'']_{\equiv_o^i}$$

$$(C.52) \quad (x', y') \in \text{under}O^{D^i} \cup \Delta\text{under}O^{D^i}$$

$$(C.53) \quad (y'', x'') \in \text{under}O^{D^i} \cup \Delta\text{under}O^{D^i}$$

Aus (C.50), (C.51) und (C.49) folgt:

$$(C.54) \quad typeO^{D^{i+1}}([x']_{\equiv_o^i}) = typeO^{D^{i+1}}([y']_{\equiv_o^i})$$

$$(C.55) \quad typeO^{D^{i+1}}([x'']_{\equiv_o^i}) = typeO^{D^{i+1}}([y'']_{\equiv_o^i})$$

Nach Lemma 9.13 stimmen die Schemata von D^i und D^{i+1} überein. Aus (C.54) und (C.55) folgt somit:

$$(C.56) \quad typeO^{D^i}(x') = typeO^{D^i}(y')$$

$$(C.57) \quad typeO^{D^i}(x'') = typeO^{D^i}(y'')$$

Nach (C.52) gibt es zwei Fälle zu unterscheiden:

(a) $(x', y') \in \text{under}O^{D^i}$. Dann erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto y' \end{aligned}$$

die Prämisse des Axioms (M.27) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.58) \quad (x', y') \in \text{rel}O^{D^i}$$

Nach (C.58) und (C.56) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto y' \end{aligned}$$

die Prämisse des Axioms (M.28) in D^i . Nach der Konstruktion von $\Delta = \Delta_O^{D^i}$ in Konstruktion A.92 folgt daraus:

$$(C.59) \quad \begin{aligned} &(x', y') \in \Delta = \Delta_O^{D^i} \\ \Rightarrow &[x']_{\equiv_o^i} = [y']_{\equiv_o^i} \end{aligned}$$

Aus (C.59), (C.50) und (C.51) folgt schließlich

$$[x]_{\equiv_o^i} = [y]_{\equiv_o^i}$$

und somit die Wahrheit der Konklusion von (C.46).

(b) $(x', y') \in \Delta \text{under}O^{D^i}$. Nun entstehen alle Elemente in $\Delta \text{under}O^{D^i}$ durch eine passende Variablenbelegung der Variablen in Axiom (M.21), weil dieses Axiom als einziges M^i -Axiom das Prädikat $\text{under}O$ in der Konklusion wahr macht. Somit existiert nach Konstruktion A.92 ein Element $z \in D^i$ mit:

$$(C.60) \quad (x', z) \in \text{under}O^{D^i}$$

$$(C.61) \quad (z, y') \in \text{under}O^{D^i}$$

Aus (C.60) und (C.61) folgt, dass die Variablenbelegungen

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto z \end{aligned}$$

und

$$x \mapsto z$$

$$y \mapsto y'$$

jeweils die Prämisse des Axioms (M.8) in D^i erfüllen. Weil das Axiom nach Lemma 9.12 in D^i gilt, folgt die Wahrheit der Konklusionen:

$$(C.62) \quad (typeO^{D^i}(x'), typeO^{D^i}(z)) \in underC^{D^i}$$

$$(C.63) \quad (typeO^{D^i}(z), typeO^{D^i}(y')) \in underC^{D^i}$$

Aus (C.62), (C.63) und (C.56) folgt, dass die Variablenbelegung

$$x \mapsto typeO^{D^i}(x')$$

$$y \mapsto typeO^{D^i}(z)$$

die Prämisse des Axioms (M.17) in D^i erfüllt. Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.64) \quad typeO^{D^i}(x') = typeO^{D^i}(z)$$

Zusammen mit (C.56) ergibt sich:

$$(C.65) \quad typeO^{D^i}(z) = typeO^{D^i}(y')$$

Weiter erfüllen nach (C.60) und (C.61) die Variablenbelegungen

$$x \mapsto x'$$

$$y \mapsto z$$

und

$$x \mapsto z$$

$$y \mapsto y'$$

jeweils die Prämisse des Axioms (M.27) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusionen:

$$(C.66) \quad (x', z) \in relO^{D^i}$$

$$(C.67) \quad (z, y') \in relO^{D^i}$$

Nach (C.66) und (C.64) sowie (C.67) und (C.65) erfüllen die Variablenbelegungen

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto z \end{aligned}$$

und

$$\begin{aligned} x &\mapsto z \\ y &\mapsto y' \end{aligned}$$

jeweils die Prämisse des Axioms (M.28) in D^i . Nach der Konstruktion von $\Delta = \frac{D^i}{O}$ in Konstruktion A.92 folgt daraus:

$$\begin{aligned} (x', z) &\in \Delta = \frac{D^i}{O} \\ (z, y') &\in \Delta = \frac{D^i}{O} \end{aligned}$$

Weil $\equiv^i = \text{cong}((=^s_{D^i} \cup \Delta =^s_{D^i})_{s \in S})$, ist dies äquivalent zu:

$$(C.68) \quad [x']_{\equiv^i_o} = [z]_{\equiv^i_o} = [y']_{\equiv^i_o}$$

Aus (C.68), (C.50) und (C.51) folgt schließlich

$$[x]_{\equiv^i_o} = [y]_{\equiv^i_o}$$

und somit die Wahrheit der Konklusion von (C.46).

(7) *Axiom* (M.22). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.69) \quad ([x]_{\equiv^i_c}, [y]_{\equiv^i_c}) \in \text{rel}C^{D^{i+1}} \Rightarrow ([y]_{\equiv^i_c}, [x]_{\equiv^i_c}) \in \text{rel}C^{D^{i+1}}$$

für alle $[x]_{\equiv^i_c}, [y]_{\equiv^i_c} \in D^{i+1}_C$. Gelte also die Prämisse von (C.69). Nach Lemma 9.13 stimmen die Schemata von D^i und D^{i+1} überein. Somit gilt $x, y \in D^i_C$ mit:

$$(C.70) \quad (x, y) \in \text{rel}C^{D^i}$$

Nach (C.70) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x \\ y &\mapsto y \end{aligned}$$

die Prämisse des Axioms (M.22) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.71) \quad (y, x) \in \text{rel}C^{D^i}$$

Weil $[\]_{\equiv i}$ als Homomorphismus Relationen überträgt, folgt aus (C.71) die Wahrheit der Konklusion von (C.69).

(8) *Axiom* (M.23). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.72) \quad ([x]_{\equiv_c^i}, [y]_{\equiv_c^i}) \in \text{rel}C^{D^{i+1}} \wedge ([y]_{\equiv_c^i}, [z]_{\equiv_c^i}) \in \text{rel}C^{D^{i+1}} \\ \Rightarrow ([x]_{\equiv_c^i}, [z]_{\equiv_c^i}) \in \text{rel}C^{D^{i+1}}$$

für alle $[x]_{\equiv_c^i}, [y]_{\equiv_c^i}, [z]_{\equiv_c^i} \in D_C^{i+1}$. Gelte also die Prämisse von (C.72). Nach Lemma 9.13 stimmen die Schemata von D^i und D^{i+1} überein. Somit gilt $x, y, z \in D_C^i$ mit:

$$(C.73) \quad (x, y) \in \text{rel}C^{D^i}$$

$$(C.74) \quad (y, z) \in \text{rel}C^{D^i}$$

Nach (C.73) und (C.74) erfüllt die Variablenbelegung

$$x \mapsto x$$

$$y \mapsto y$$

$$z \mapsto z$$

die Prämisse des Axioms (M.23) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.75) \quad (x, z) \in \text{rel}C^{D^i}$$

Weil $[\]_{\equiv i}$ als Homomorphismus Relationen überträgt, folgt aus (C.75) die Wahrheit der Konklusion von (C.72).

(9) *Axiom* (M.24). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.76) \quad ([x]_{\equiv_o^i}, [y]_{\equiv_o^i}) \in \text{rel}O^{D^{i+1}} \Rightarrow ([y]_{\equiv_o^i}, [x]_{\equiv_o^i}) \in \text{rel}O^{D^{i+1}}$$

für alle $[x]_{\equiv_o^i}, [y]_{\equiv_o^i} \in D_O^{i+1}$. Gelte also die Prämisse von (C.76). Nach Konstruktion A.41 existieren zwei Elemente $x', y' \in D_O^i$ mit:

$$(C.77) \quad [x]_{\equiv_o^i} = [x']_{\equiv_o^i}$$

$$(C.78) \quad [y]_{\equiv_o^i} = [y']_{\equiv_o^i}$$

$$(C.79) \quad (x', y') \in \text{rel}O^{D^i} \cup \Delta\text{rel}O^{D^i}$$

Nach (C.79) gibt es zwei Fälle zu unterscheiden:

(a) $(x', y') \in \text{rel}O^{D^i}$. Dann erfüllt die Variablenbelegung

$$x \mapsto x'$$

$$y \mapsto y'$$

die Prämisse des Axioms (M.24) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.80) \quad (y', x') \in \text{rel}O^{D^i}$$

Weil $[\]_{\equiv_o^i}$ als Homomorphismus Relationen überträgt, folgt aus (C.80) die Wahrheit der Konklusion von (C.76).

(b) $(x', y') \in \Delta\text{rel}O^{D^i}$. Nun entstehen alle Elemente in $\Delta\text{rel}O^{D^i}$ durch eine passende Variablenbelegung der Variablen in Axiom (M.25), weil dieses Axiom als einziges M'' -Axiom das Prädikat $\text{rel}O$ in der Konklusion wahr macht. Somit existiert nach Konstruktion A.92 ein Element $z \in D^i$ mit:

$$(C.81) \quad (x', z) \in \text{rel}O^{D^i}$$

$$(C.82) \quad (z, y') \in \text{rel}O^{D^i}$$

Nach (C.81) und (C.82) erfüllen die Variablenbelegungen

$$x \mapsto x'$$

$$y \mapsto z$$

und

$$x \mapsto z$$

$$y \mapsto y'$$

jeweils die Prämisse des Axioms (M.24) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusionen:

$$(C.83) \quad (z, x') \in relO^{D^i}$$

$$(C.84) \quad (y', z) \in relO^{D^i}$$

Nach (C.83) und (C.84) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto y' \\ y &\mapsto z \\ z &\mapsto x' \end{aligned}$$

die Prämisse des M'' -Axioms (M.25) in D^i . Nach der Konstruktion von $\Delta relO^{D^i}$ in Konstruktion A.92 folgt daraus:

$$(C.85) \quad (y', x') \in \Delta relO^{D^i}$$

Nach Konstruktion A.92 gilt offensichtlich $[x']_{\equiv i} = [x]_{\equiv i}$ und $[y']_{\equiv i} = [y]_{\equiv i}$. Weil $[\]_{\equiv i}$ als Homomorphismus Relationen überträgt, folgt aus (C.85) die Wahrheit der Konklusion von (C.76).

(10) *Axiom* (M.26). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.86) \quad ([x]_{\equiv_c}, [y]_{\equiv_c}) \in underC^{D^{i+1}} \Rightarrow ([x]_{\equiv_c}, [y]_{\equiv_c}) \in relC^{D^{i+1}}$$

für alle $[x]_{\equiv_c}, [y]_{\equiv_c} \in D_C^{i+1}$. Gelte also die Prämisse von (C.86). Nach Lemma 9.13 stimmen die Schemata von D^i und D^{i+1} überein. Somit gilt $x, y \in D_C^i$ mit:

$$(C.87) \quad (x, y) \in underC^{D^i}$$

Nach (C.87) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x \\ y &\mapsto y \end{aligned}$$

die Prämisse des Axioms (M.26) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.88) \quad (x, y) \in relC^{D^i}$$

Weil $[\]_{\equiv i}$ als Homomorphismus Relationen überträgt, folgt aus C.88 die Wahrheit der Konklusion von (C.86).

(11) *Axiom* (M.27). Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} :

$$(C.89) \quad ([x]_{\equiv_o}, [y]_{\equiv_o}) \in \text{under}O^{D^{i+1}} \Rightarrow ([x]_{\equiv_o}, [y]_{\equiv_o}) \in \text{rel}O^{D^{i+1}}$$

für alle $[x]_{\equiv_o}, [y]_{\equiv_o} \in D_C^{i+1}$. Gelte also die Prämisse von (C.89). Nach Konstruktion A.41 existieren zwei Elemente $x', y' \in D_O^i$ mit:

$$(C.90) \quad [x]_{\equiv_o} = [x']_{\equiv_o}$$

$$(C.91) \quad [y]_{\equiv_o} = [y']_{\equiv_o}$$

$$(C.92) \quad (x', y') \in \text{under}O^{D^i} \cup \Delta\text{under}O^{D^i}$$

Nach (C.92) gibt es zwei Fälle zu unterscheiden:

(a) $(x', y') \in \text{under}O^{D^i}$. Dann erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto y' \end{aligned}$$

die Prämisse des Axioms (M.27) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.93) \quad (x', y') \in \text{rel}O^{D^i}$$

Weil $[\]_{\equiv_o}$ als Homomorphismus Relationen überträgt, folgt aus (C.93) die Wahrheit der Konklusion von (C.89).

(b) $(x', y') \in \Delta\text{under}O^{D^i}$. Nun entstehen alle Elemente in $\Delta\text{under}O^{D^i}$ durch eine passende Variablenbelegung der Variablen in Axiom (M.21), weil dieses Axiom als einziges M' -Axiom das Prädikat $\text{under}O$ in der Konklusion wahr macht. Somit existiert nach Konstruktion A.92 ein Element $z \in D^i$ mit:

$$(C.94) \quad (x', z) \in \text{under}O^{D^i}$$

$$(C.95) \quad (z, y') \in \text{under}O^{D^i}$$

Nach (C.94) und (C.95) erfüllen die Variablenbelegungen

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto z \end{aligned}$$

und

$$x \mapsto z$$

$$y \mapsto y'$$

jeweils die Prämisse des Axioms (M.27) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusionen:

$$(C.96) \quad (x', z) \in \text{rel}O^{D^i}$$

$$(C.97) \quad (z, y') \in \text{rel}O^{D^i}$$

Nach (C.96) und (C.97) erfüllt die Variablenbelegung

$$x \mapsto x'$$

$$y \mapsto z$$

$$z \mapsto y'$$

die Prämisse des M'' -Axioms (M.25) in D^i . Nach der Konstruktion von $\Delta \text{rel}O^{D^i}$ in Konstruktion A.92 folgt daraus:

$$(C.98) \quad (x', y') \in \Delta \text{rel}O^{D^i}$$

Nach Konstruktion A.92 gilt offensichtlich $[x']_{\equiv^i} = [x']_{\equiv^i}$ und $[y']_{\equiv^i} = [y']_{\equiv^i}$. Weil $[\]_{\equiv^i}$ als Homomorphismus Relationen überträgt, folgt aus (C.98) die Wahrheit der Konklusion von (C.89). \square

C.1.4 Beweis von Satz 9.17

Nach Konstruktion A.92 ergibt sich das resultierende M^* -System durch schrittweise Annäherung des Ausgangssystems D an die (eventuell) nicht gültigen M^* -Axiome. Die Gültigkeit aller M -Axiome in $\mathcal{F}_{\text{Ob}}^{M^*}(D)$ bis auf die Axiome (M.25) und (M.29) wird analog zu Satz 9.15 nachgewiesen. Es wird nun mittels Induktion gezeigt, dass unter der vorausgesetzten Injektivität von r_E^t auf Nicht-*self*-Kanten die Konstruktion der Epireflexion in jedem Schritt auch die Gültigkeit der Axiome (M.25) und (M.29) erhält.

Zuerst wird nachgewiesen, dass das Axiom (M.28) in allen Systemen D^i mit $i > 0$ gilt. Daraus folgt nach der Konstruktion von $\Delta =_s^{D^i}$ in Konstruktion A.92, dass $\Delta =_s^{D^i} \subseteq =_s^{D^i}$ für alle $s \in S$ und somit $D_s^{i+1} = D_s^i$ für alle $i > 0$ gilt.

- *Induktionsbeginn* $i = 1$: Zu zeigen ist die Gültigkeit der folgenden Implikation in D^1 :

$$(C.99) \quad ([x]_{\equiv_0}, [y]_{\equiv_0}) \in \text{rel}O^{D^1} \wedge \text{type}O^{D^1}([x]_{\equiv_0}) = \text{type}O^{D^1}([y]_{\equiv_0}) \Rightarrow [x]_{\equiv_0} = [y]_{\equiv_0}$$

für alle $[x]_{\equiv_0}, [y]_{\equiv_0} \in D_O^1$. Gelte also die Prämisse von (C.99). Nach Konstruktion A.41 existieren zwei Elemente $x', y' \in D_O^0$ mit:

$$(C.100) \quad [x]_{\equiv_0} = [x']_{\equiv_0}$$

$$(C.101) \quad [y]_{\equiv_0} = [y']_{\equiv_0}$$

$$(C.102) \quad (x', y') \in relO^{D^0} \cup \Delta relO^{D^0}$$

Weil kein M^* -Axiom in der Konklusion das Prädikat $relO$ wahr macht, gilt $\Delta relO^{D^i} = \emptyset$. Aus (C.102) folgt somit:

$$(C.103) \quad (x', y') \in relO^{D^i}$$

Weiter gilt:

$$\begin{aligned} [typeO^{D^0}(x')]_{\equiv_c} &= typeO^{D^1}([x']_{\equiv_0}) && ([\]_{\equiv_0} \text{ ist Homomorphismus}) \\ &= typeO^{D^1}([x]_{\equiv_0}) && (C.100) \\ &= typeO^{D^1}([y]_{\equiv_0}) && (\text{Prämisse von (C.99)}) \\ &= typeO^{D^1}([y']_{\equiv_0}) && (C.101) \\ &= [typeO^{D^0}(y')]_{\equiv_c} && ([\]_{\equiv_0} \text{ ist Homomorphismus}) \\ (C.104) \quad \Rightarrow typeO^{D^0}(x') &= typeO^{D^0}(y') && ([\]_{\equiv_c} \text{ ist injektiv nach Lemma 9.13}) \end{aligned}$$

Nach (C.103) und (C.104) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto y' \end{aligned}$$

die Prämisse des Axioms (M.28) in D^0 . Nach der Konstruktion von $\Delta =_{D^0}^{D^0}$ in Konstruktion A.92 folgt daraus:

$$(C.105) \quad \begin{aligned} (x', y') &\in \Delta =_{D^0}^{D^0} \\ \Rightarrow [x']_{\equiv_0} &= [y']_{\equiv_0} \end{aligned}$$

Aus (C.105), (C.100) und (C.101) folgt schließlich die Wahrheit der Konklusion von (C.99).

- *Induktionsvoraussetzung:* Das Axiom (M.28) gilt in D^i mit $i > 0$.
- *Induktionsschritt $i \mapsto i + 1$:* Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} mit $i > 0$:

$$(C.106) \quad ([x]_{\equiv_o^i}, [y]_{\equiv_o^i}) \in relO^{D^{i+1}} \wedge typeO^{D^{i+1}}([x]_{\equiv_o^i}) = typeO^{D^{i+1}}([y]_{\equiv_o^i}) \Rightarrow [x]_{\equiv_o^i} = [y]_{\equiv_o^i}$$

für alle $[x]_{\equiv_o^i}, [y]_{\equiv_o^i} \in D_o^{i+1}$. Gelte also die Prämisse von (C.106). Nach der Induktionsvoraussetzung gilt das Axiom (M.28) in D^i . Weil dieses Axiom das einzige M^* -Axiom ist, das in der Konklusion Elemente identifiziert, folgt aus Konstruktion A.92, dass $=_s^{D^i} = \equiv_s^i$ für alle $s \in S$. Daraus folgt, dass $[\]_{\equiv^i}$ auch injektiv und somit bijektiv auf allen Trägermengen ist. Daraus ergibt sich:

$$(C.107) \quad [x]_{\equiv_o^i} = x$$

$$(C.108) \quad [y]_{\equiv_o^i} = y$$

Weiter existieren nach Konstruktion A.41 zwei Elemente $x', y' \in D_o^i$ mit:

$$(C.109) \quad [x]_{\equiv_o^i} = [x']_{\equiv_o^i}$$

$$(C.110) \quad [y]_{\equiv_o^i} = [y']_{\equiv_o^i}$$

$$(C.111) \quad (x', y') \in relO^{D^i} \cup \Delta relO^{D^i}$$

Aus der Injektivität von $[\]_{\equiv^i}$ folgt aus (C.109) und (C.110) $x = x'$ und $y = y'$. Zusammen mit (C.111) ergibt sich:

$$(C.112) \quad (x, y) \in relO^{D^i} \cup \Delta relO^{D^i}$$

Weil kein M^* -Axiom in der Konklusion das Prädikat $relO$ wahr macht, gilt $\Delta relO^{D^i} = \emptyset$. Aus (C.112) folgt somit:

$$(C.113) \quad (x, y) \in relO^{D^i}$$

Weiter gilt:

$$(C.114) \quad \begin{aligned} [typeO^{D^i}(x)]_{\equiv_c^i} &= typeO^{D^{i+1}}([x]_{\equiv_o^i}) && ([\]_{\equiv^i} \text{ ist Homomorphismus}) \\ &= typeO^{D^{i+1}}([y]_{\equiv_o^i}) && (\text{Prämisse von (C.106)}) \\ &= [typeO^{D^i}(y)]_{\equiv_c^i} && ([\]_{\equiv^i} \text{ ist Homomorphismus}) \\ \Rightarrow typeO^{D^i}(x) &= typeO^{D^i}(y) && ([\]_{\equiv_c^i} \text{ ist injektiv}) \end{aligned}$$

Nach (C.113) und (C.114) erfüllt die Variablenbelegung

$$x \mapsto x$$

$$y \mapsto y$$

die Prämisse des Axioms (M.28) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.115) \quad x = y$$

Aus (C.115), (C.107) und (C.108) folgt schließlich die Wahrheit der Konklusion von (C.106).

Nun wird mittels Induktion gezeigt, dass das Axiom (M.29) in allen Systemen D^i mit $i \in \mathbb{N}_0$ gilt. Nach Voraussetzung lassen sich die Operationen $typeO^D$, $typeL^D$, $typeV_t^D$ und $typeOA_t^D$ für alle $t \in TYPES$ in die beiden MP-Homomorphismen $type_{I\#,E}$ und r^t aufteilen. Insbesondere gilt:

$$(C.116) \quad typeL^D = r_E^t \circ type_{I\#,E}$$

- *Induktionsbeginn $i = 0$* : Es gilt $D^0 = D$. Es wird die Gültigkeit des Axioms (M.29) in D nachgewiesen. Seien also zwei Elemente $x, y \in D_L$ gegeben mit:

$$(C.117) \quad sourceL^D(x) = sourceL^D(y)$$

$$(C.118) \quad typeL^D(x) = typeL^D(y)$$

Es ist zu zeigen, dass die Konklusion $x = y$ folgt.

Nun ergibt sich:

$$r_E^t(type_{I\#,E}(x)) = typeL^D(x) \quad (C.116)$$

$$= typeL^D(y) \quad (C.118)$$

$$(C.119) \quad = r_E^t(type_{I\#,E}(y)) \quad (C.116)$$

Nach Voraussetzung sind zwei Fälle zu unterscheiden:

- (1) $type_{I\#,E}(x)$ und $type_{I\#,E}(y)$ sind beides *self*-Kanten. Somit gibt es $p, q \in S_N^\#$ mit:

$$(C.120) \quad self^{S^\#}(p) = type_{I\#,E}(x)$$

$$(C.121) \quad self^{S^\#}(q) = type_{I\#,E}(y)$$

Daraus folgt:

$$p = source^{S^\#}(self^{S^\#}(p)) \quad (\text{Axiom (MP.1) gilt in } S^\#)$$

$$= source^{S^\#}(type_{I\#,E}(x)) \quad (C.120)$$

$$= type_{I\#,N}(source^{I^\#}(x)) \quad (type_{I\#,E} \text{ ist Homomorphismus})$$

$$= type_{I\#,N}(source^{I^\#}(y)) \quad (C.117)$$

$$\begin{aligned}
&= source^{S^\#}(type_{I^\#,E}(y)) && (type_{I^\#} \text{ ist Homomorphismus}) \\
&= source^{S^\#}(self^{S^\#}(q)) && (C.121) \\
(C.122) \quad &= q && (\text{Axiom (MP.1) gilt in } S^\#)
\end{aligned}$$

Dies führt zu:

$$\begin{aligned}
&p = q && (C.122) \\
&\Rightarrow self^{S^\#}(p) = self^{S^\#}(q) && (self^{S^\#} \text{ ist Abbildung}) \\
(C.123) \quad &\Rightarrow type_{I^\#,E}(x) = type_{I^\#,E}(y) && (C.120) \text{ und } (C.121)
\end{aligned}$$

(2) Entweder $type_{I^\#,E}(x)$ oder $type_{I^\#,E}(y)$ ist keine *self*-Kante. Nach Voraussetzung ist r_E^t injektiv auf Nicht-*self*-Kanten. Somit folgt aus (C.119) sofort die Wahrheit von (C.123).

Nach (C.117) und (C.123) erfüllt die Variablenbelegung

$$\begin{aligned}
x &\mapsto x \\
y &\mapsto y
\end{aligned}$$

die Prämisse des Axioms (M.29) in $I^\# \xrightarrow{type_{I^\#}} S^\#$. Weil $I^\# \xrightarrow{type_{I^\#}} S^\#$ nach Voraussetzung ein *M*-System ist, ist die Konklusion $x = y$ wahr in $I^\# \xrightarrow{type_{I^\#}} S^\#$ und somit auch wahr in *D*.

- *Induktionsbeginn* $i = 1$: Es wird die Gültigkeit des Axioms (M.29) in D^1 nachgewiesen. Seien also zwei Elemente $[x]_{\equiv_L^0}, [y]_{\equiv_L^0} \in D_L^1$ gegeben mit:

$$(C.124) \quad sourceL^{D^1}([x]_{\equiv_L^0}) = sourceL^{D^1}([y]_{\equiv_L^0})$$

$$(C.125) \quad typeL^{D^1}([x]_{\equiv_L^0}) = typeL^{D^1}([y]_{\equiv_L^0})$$

Es ist zu zeigen, dass die Konklusion $[x]_{\equiv_L^0} = [y]_{\equiv_L^0}$ folgt.

Es gilt:

$$\begin{aligned}
&[typeL^{D^0}(x)]_{\equiv_A^0} = typeL^{D^1}([x]_{\equiv_L^0}) && ([]_{\equiv^0} \text{ ist Homomorphismus}) \\
&= typeL^{D^1}([y]_{\equiv_L^0}) && (C.125) \\
&= [typeL^{D^0}(y)]_{\equiv_A^0} && ([]_{\equiv^0} \text{ ist Homomorphismus}) \\
(C.126) \quad &\Rightarrow typeL^{D^0}(x) = typeL^{D^0}(y) && ([]_{\equiv_A^0} \text{ ist injektiv nach Lemma 9.13})
\end{aligned}$$

Weiter ergibt sich:

$$\begin{aligned}
& [sourceL^{D^0}(x)]_{\equiv^0} = sourceL^{D^1}([x]_{\equiv^0}) && ([]_{\equiv^0} \text{ ist Homomorphismus}) \\
& & = sourceL^{D^1}([y]_{\equiv^0}) && \text{(C.124)} \\
\text{(C.127)} \quad & = [sourceL^{D^0}(y)]_{\equiv^0} && ([]_{\equiv^0} \text{ ist Homomorphismus})
\end{aligned}$$

Nach (C.127) gibt es nun zwei Fälle zu unterscheiden:

- (1) $(sourceL^{D^0}(x), sourceL^{D^0}(y)) \in =_{D^0}^O$. Dies entspricht der Gleichung $sourceL^{D^0}(x) = sourceL^{D^0}(y)$. Zusammen mit (C.126) folgt, dass die Variablenbelegung

$$\begin{aligned}
x &\mapsto x \\
y &\mapsto y
\end{aligned}$$

die Prämisse des Axioms (M.29) in D^0 erfüllt. Oben wurde bewiesen, dass dieses Axiom in D^0 gültig ist. Somit folgt die Wahrheit der Konklusion:

$$\text{(C.128)} \quad x = y$$

Aus (C.128) folgt unmittelbar $[x]_{\equiv^0} = [y]_{\equiv^0}$ und somit die Gültigkeit der nachzuweisenden Implikation.

- (2) $(sourceL^{D^0}(x), sourceL^{D^0}(y)) \in \Delta_{=_{D^0}^O}$. Nun ist Axiom (M.28) das einzige M^* -Axiom, das in der Konklusion Elemente identifiziert. Nach Konstruktion A.92 gilt somit die Prämisse des Axioms (M.28) in D^0 :

$$\begin{aligned}
\text{(C.129)} \quad & (sourceL^{D^0}(x), sourceL^{D^0}(y)) \in relO^{D^0} \\
\text{(C.130)} \quad & typeO^{D^0}(sourceL^{D^0}(x)) = typeO^{D^0}(sourceL^{D^0}(y))
\end{aligned}$$

Weiter folgt:

$$\begin{aligned}
& r_E^t(type_{I\#,E}(x)) = typeL^{D^0}(x) && \text{(C.116)} \\
& & = typeL^{D^0}(y) && \text{(C.126)} \\
\text{(C.131)} \quad & = r_E^t(type_{I\#,E}(y)) && \text{(C.116)}
\end{aligned}$$

Nach Voraussetzung sind zwei Fälle zu unterscheiden:

- (a) $type_{I\#,E}(x)$ und $type_{I\#,E}(y)$ sind beides *self*-Kanten. Somit gibt es $p, q \in S_N^\#$ mit:

$$\text{(C.132)} \quad self^{S^\#}(p) = type_{I\#,E}(x)$$

$$\text{(C.133)} \quad self^{S^\#}(q) = type_{I\#,E}(y)$$

Nun ist x ebenfalls *self*-Kante. Weil $I^\#$ nach Voraussetzung *MP*-System ist, gilt Axiom (MP.1) in $I^\#$. Daraus folgt:

$$(C.134) \quad source^{I^\#}(self^{I^\#}(source^{I^\#}(x))) = source^{I^\#}(x)$$

Weiter gilt:

$$\begin{aligned} & type_{I^\#,E}(self^{I^\#}(source^{I^\#}(x))) \\ &= self^{S^\#}(source^{S^\#}(type_{I^\#,E}(x))) \quad (type_{I^\#} \text{ ist Homomorphismus}) \\ &= self^{S^\#}(source^{S^\#}(self^{S^\#}(p))) \quad (C.132) \\ &= self^{S^\#}(p) \quad (\text{Axiom (MP.1) gilt in } S^\#) \\ (C.135) \quad &= type_{I^\#,E}(x) \quad (C.132) \end{aligned}$$

Nach (C.134) und (C.135) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x \\ y &\mapsto self^{I^\#}(source^{I^\#}(x)) \end{aligned}$$

die Prämisse des Axioms (M.29) in $I^\# \xrightarrow{type_{I^\#}} S^\#$. Weil $I^\# \xrightarrow{type_{I^\#}} S^\#$ nach Voraussetzung ein *M*-System ist, folgt die Wahrheit der Konklusion in $I^\# \xrightarrow{type_{I^\#}} S^\#$ und auch in D^0 , weil die Träermengen zur Sorte O in beiden Systemen übereinstimmen:

$$(C.136) \quad x = self^{I^\#}(source^{I^\#}(x))$$

Analog folgt unter Ausnutzung von (C.133):

$$(C.137) \quad y = self^{I^\#}(source^{I^\#}(y))$$

Aus der Voraussetzung $(sourceL^{D^0}(x), sourceL^{D^0}(y)) \in \Delta_{=O}^{D^0}$ folgt durch den Kongruenzabschluss:

$$(C.138) \quad (selfO^{D^0}(sourceL^{D^0}(x)), selfO^{D^0}(sourceL^{D^0}(y))) \in \equiv_{=L}^{D^0}$$

Aus (C.136), (C.137) und (C.138) folgt schließlich $[x]_{\equiv_L^0} = [y]_{\equiv_L^0}$ und somit die Gültigkeit der nachzuweisenden Implikation.

- (b) Entweder $type_{I^\#,E}(x)$ oder $type_{I^\#,E}(y)$ ist keine *self*-Kante. Nach Voraussetzung ist r_E^t injektiv auf Nicht-*self*-Kanten. Somit folgt aus (C.131):

$$(C.139) \quad type_{I\#,E}(x) = type_{I\#,E}(y)$$

Daraus ergibt sich:

$$\begin{aligned} type_{I\#,N}(source^{I\#}(x)) &= source^{S\#}(type_{I\#,E}(x)) \quad (type_{I\#} \text{ ist Homomorphismus}) \\ &= source^{S\#}(type_{I\#,E}(y)) \quad (C.139) \\ (C.140) \quad &= type_{I\#,N}(source^{I\#}(y)) \quad (type_{I\#} \text{ ist Homomorphismus}) \end{aligned}$$

Nach (C.129) und (C.140) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto sourceL^{D^0}(x) \\ y &\mapsto sourceL^{D^0}(y) \end{aligned}$$

die Prämisse des Axioms (M.28) in $I\# \xrightarrow{type_{I\#}} S\#$. Weil $I\# \xrightarrow{type_{I\#}} S\#$ nach Voraussetzung ein M -System ist, folgt die Wahrheit der Konklusion in $I\# \xrightarrow{type_{I\#}} S\#$ und auch in D^0 , weil die Trägermengen zur Sorte O in beiden Systemen übereinstimmen:

$$(C.141) \quad sourceL^{D^0}(x) = sourceL^{D^0}(y)$$

Nach (C.141) und (C.126) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x \\ y &\mapsto y \end{aligned}$$

die Prämisse des Axioms (M.29) in D^0 . Oben wurde bewiesen, dass dieses Axiom in D^0 gültig ist. Somit folgt die Wahrheit der Konklusion:

$$(C.142) \quad x = y$$

Aus (C.142) folgt unmittelbar $[x]_{\equiv_L^0} = [y]_{\equiv_L^0}$ und somit die Gültigkeit der nachzuweisenden Implikation.

- *Induktionsvoraussetzung:* Das Axiom (M.29) gilt in D^i mit $i > 0$.
- *Induktionsschritt $i \mapsto i + 1$:* Zu zeigen ist die Gültigkeit der folgenden Implikation in D^{i+1} mit $i > 0$:

$$\begin{aligned} (C.143) \quad &sourceL^{D^{i+1}}([x]_{\equiv_L^i}) = sourceL^{D^{i+1}}([y]_{\equiv_L^i}) \\ &\wedge typeL^{D^{i+1}}([x]_{\equiv_L^i}) = typeL^{D^{i+1}}([y]_{\equiv_L^i}) \\ &\Rightarrow [x]_{\equiv_L^i} = [y]_{\equiv_L^i} \end{aligned}$$

für alle $[x]_{\equiv_L^i}, [y]_{\equiv_L^i} \in D_L^{i+1}$. Gelte also die Prämisse von (C.143). Oben wurde gezeigt, dass $[\]_{\equiv^i}$ für $i > 0$ bijektiv und somit injektiv auf allen Trägermengen ist. Daraus folgt:

$$\begin{aligned}
 & [sourceL^{D^i}(x)]_{\equiv_o^i} = sourceL^{D^{i+1}}([x]_{\equiv_L^i}) && ([\]_{\equiv^i} \text{ ist Homomorphismus}) \\
 & = sourceL^{D^{i+1}}([y]_{\equiv_L^i}) && (\text{Prämisse von (C.143)}) \\
 & = [sourceL^{D^i}(y)]_{\equiv_o^i} && ([\]_{\equiv^i} \text{ ist Homomorphismus}) \\
 \text{(C.144)} \quad & \Rightarrow sourceL^{D^i}(x) = sourceL^{D^i}(y) && ([\]_{\equiv_o^i} \text{ ist injektiv})
 \end{aligned}$$

Weiter gilt:

$$\begin{aligned}
 & [typeL^{D^i}(x)]_{\equiv_A^i} = typeL^{D^{i+1}}([x]_{\equiv_L^i}) && ([\]_{\equiv^i} \text{ ist Homomorphismus}) \\
 & = typeL^{D^{i+1}}([y]_{\equiv_L^i}) && (\text{Prämisse von (C.143)}) \\
 & = [typeL^{D^i}(y)]_{\equiv_A^i} && ([\]_{\equiv^i} \text{ ist Homomorphismus}) \\
 \text{(C.145)} \quad & \Rightarrow typeL^{D^i}(x) = typeL^{D^i}(y) && ([\]_{\equiv_A^i} \text{ ist injektiv})
 \end{aligned}$$

Nach (C.144) und (C.145) erfüllt die Variablenbelegung

$$\begin{aligned}
 x & \mapsto x \\
 y & \mapsto y
 \end{aligned}$$

die Prämisse des Axioms (M.29) in D^i . Nach der Induktionsvoraussetzung ist das Axiom in D^i gültig. Somit folgt die Wahrheit der Konklusion:

$$\text{(C.146)} \quad x = y$$

Aus (C.146) folgt unmittelbar die Wahrheit der Konklusion von (C.143).

Schließlich wird mittels Induktion nachgewiesen, dass das Axiom (M.25) in allen Systemen D^i mit $i \in \mathbb{N}_0$ gilt.

- *Induktionsbeginn* $i = 0$: Es gilt $D^0 = D$. Das Axiom (M.25) ist in D gültig, weil es Teil der Spezifikation M' ist und D nach Voraussetzung ein M' -System ist.
- *Induktionsvoraussetzung*: Das Axiom (M.25) gilt in D^i .
- *Induktionsschritt* $i \mapsto i + 1$: Gelte die Prämisse von Axiom (M.25) in D^{i+1} mit $[x]_{\equiv_o^i}, [y]_{\equiv_o^i}, [z]_{\equiv_o^i} \in D_o^{i+1}$:

$$\text{(C.147)} \quad ([x]_{\equiv_o^i}, [y]_{\equiv_o^i}) \in relO^{D^{i+1}}$$

$$\text{(C.148)} \quad ([y]_{\equiv_o^i}, [z]_{\equiv_o^i}) \in relO^{D^{i+1}}$$

Zu zeigen ist die Gültigkeit der Konklusion:

$$(C.149) \quad ([x]_{\equiv_o^i}, [z]_{\equiv_o^i}) \in relO^{D^{i+1}}$$

Nach Konstruktion A.41 existieren zwei Elemente $x', y' \in D_O^0$ mit:

$$(C.150) \quad [x]_{\equiv_o^i} = [x']_{\equiv_o^i}$$

$$(C.151) \quad [y]_{\equiv_o^i} = [y']_{\equiv_o^i}$$

$$(C.152) \quad (x', y') \in relO^{D^i} \cup \Delta relO^{D^i}$$

Weiter existieren nach Konstruktion A.41 zwei Elemente $y'', z' \in D_O^0$ mit:

$$(C.153) \quad [y]_{\equiv_o^i} = [y'']_{\equiv_o^i}$$

$$(C.154) \quad [z]_{\equiv_o^i} = [z']_{\equiv_o^i}$$

$$(C.155) \quad (y'', z') \in relO^{D^i} \cup \Delta relO^{D^i}$$

Weil kein M^* -Axiom in der Konklusion das Prädikat $relO$ wahr macht, gilt $\Delta relO^{D^i} = \emptyset$. Aus (C.152) und (C.155) folgt somit:

$$(C.156) \quad (x', y') \in relO^{D^i}$$

$$(C.157) \quad (y'', z') \in relO^{D^i}$$

Weil Axiom (M.28) das einzige M^* -Axiom ist, das in der Konklusion Elemente identifiziert, gibt es nach Konstruktion A.92 zwei Fälle zu unterscheiden:

- (1) $(y', y'') \in =_O^{D^i}$. Dies ist äquivalent zu $y' = y''$. Zusammen mit (C.156) und (C.157) folgt daraus, dass die Variablenbelegung

$$x \mapsto x'$$

$$y \mapsto y'$$

$$z \mapsto z'$$

die Prämisse des Axioms (M.25) in D^i erfüllt. Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.158) \quad (x', z') \in relO^{D^i}$$

Aus (C.158) folgt, weil $[\]_{\equiv_o^i}$ als Homomorphismus Relationen überträgt:

$$(C.159) \quad ([x']_{\equiv_i}, [z']_{\equiv_i}) \in relO^{D^{i+1}}$$

Aus (C.159), (C.150) und (C.154) folgt schließlich die Wahrheit von (C.149).

- (2) $(y', y'') \in \Delta =_O^{D^i}$. Nun ist Axiom (M.28) das einzige M^* -Axiom, das in der Konklusion Elemente identifiziert. Nach Konstruktion A.92 gilt somit die Prämisse des Axioms (M.28) in D^i :

$$(C.160) \quad (y', y'') \in relO^{D^i}$$

$$(C.161) \quad typeO^{D^i}(y') = typeO^{D^i}(y'')$$

Nun wird Axiom (M.25) zweimal angewandt. Nach (C.156) und (C.160) erfüllt die Variablenbelegung

$$x \mapsto x'$$

$$y \mapsto y'$$

$$z \mapsto y''$$

die Prämisse des Axioms (M.25) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.162) \quad (x', y'') \in relO^{D^i}$$

Nach (C.162) und (C.157) erfüllt die Variablenbelegung

$$x \mapsto x'$$

$$y \mapsto y''$$

$$z \mapsto z'$$

die Prämisse des Axioms (M.25) in D^i . Weil das Axiom nach der Induktionsvoraussetzung in D^i gilt, folgt die Wahrheit der Konklusion:

$$(C.163) \quad (x', z') \in relO^{D^i}$$

Aus (C.163) folgt, weil $[\]_{\equiv_i}$ als Homomorphismus Relationen überträgt:

$$(C.164) \quad ([x']_{\equiv_i}, [z']_{\equiv_i}) \in relO^{D^{i+1}}$$

Aus (C.164), (C.150) und (C.154) folgt schließlich die Wahrheit von (C.149). \square

C.1.5 Beweis von Lemma 9.21

Gelte die Prämisse der Implikation (A.3), es existiere also ein Element $x \in G_p$ sowie ein Tupel $y \in D_w$ mit:

$$(C.165) \quad \pi_{p,w}^G(x) = f_w(y)$$

Zu zeigen ist die Existenz eines Elements $x' \in D_p$ mit:

$$(C.166) \quad \pi_{p,w}^D(x') = y$$

Nach Lemma B.26 existiert ein Urbild \hat{x} von x unter mindestens einem der beiden Homomorphismen f und m :

(1) Gelte $\hat{x} \in D_p$ mit:

$$(C.167) \quad x = f_p(\hat{x})$$

Nach Voraussetzung ist l strikt voll. Daraus folgt nach Lemma A.24, dass l injektiv ist. Nach Lemma (B.28) übertragen Pushouts in $\mathbf{Alg}(MP_*)$ injektive Homomorphismen, somit ist f ebenfalls injektiv. Daraus folgt:

$$\begin{aligned} f_w(\pi_{p,w}^D(\hat{x})) &= \pi_{p,w}^G(f_p(\hat{x})) && (f \text{ ist Homomorphismus}) \\ &= \pi_{p,w}^G(x) && (C.167) \\ &= f_w(y) && (C.165) \end{aligned}$$

$$(C.168) \quad \Rightarrow \pi_{p,w}^D(\hat{x}) = y \quad (f \text{ ist injektiv})$$

Aus (C.168) folgt, dass die Variablenbelegung

$$x' \mapsto \hat{x}$$

die Gleichung (C.166) erfüllt.

(2) Gelte $\hat{x} \in L_p$ mit:

$$(C.169) \quad x = m_p(\hat{x})$$

Daraus folgt:

$$\begin{aligned}
m_w(\pi_{p,w}^L(\hat{x})) &= \pi_{p,w}^G(m_p(\hat{x})) && (m \text{ ist Homomorphismus}) \\
&= \pi_{p,w}^G(x) && \text{(C.169)} \\
\text{(C.170)} \quad &= f_w(y) && \text{(C.165)}
\end{aligned}$$

Aus (C.170) und Lemma B.29 folgt die Existenz eines eindeutigen Tupels $z \in K_w$ mit:

$$\text{(C.171)} \quad \pi_{p,w}^L(\hat{x}) = l_w(z)$$

$$\text{(C.172)} \quad y = k_w(z)$$

Weil l nach Voraussetzung strikt voll auf dem Prädikat p ist, folgt aus (C.171) die Existenz eines Elements $\hat{x}' \in K_p$ mit:

$$\text{(C.173)} \quad \pi_{p,w}^K(\hat{x}') = z$$

Daraus folgt:

$$\begin{aligned}
\pi_{p,w}^D(k_p(\hat{x}')) &= k_w(\pi_{p,w}^K(\hat{x}')) && (k \text{ ist Homomorphismus}) \\
&= k_w(z) && \text{(C.173)}
\end{aligned}$$

$$\text{(C.174)} \quad = y \quad \text{(C.172)}$$

Aus (C.174) folgt, dass die Variablenbelegung

$$x' \mapsto k_p(\hat{x}')$$

die Gleichung (C.166) erfüllt. □

C.2 Beweise aus Kapitel 10

C.2.1 Beweis von Lemma 10.24

Seien $x, y \in H_N$ gegeben mit $(x, y) \in \text{under}^H$ und $(y, x) \in \text{under}^H$. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz zweier Elemente $u_{xy}, u_{yx} \in H_{\text{under}}$ mit:

$$\text{(C.175)} \quad \pi_{\text{under},1}^H(u_{xy}) = x$$

$$\text{(C.176)} \quad \pi_{\text{under},2}^H(u_{xy}) = y$$

$$\text{(C.177)} \quad \pi_{\text{under},1}^H(u_{yx}) = y$$

$$\text{(C.178)} \quad \pi_{\text{under},2}^H(u_{yx}) = x$$

Zu zeigen ist die Wahrheit der Gleichung $x = y$.

Nach Lemma B.26 existiert ein Urbild u'_{xy} von u_{xy} und ein Urbild u'_{yx} von u_{yx} unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $u'_{xy}, u'_{yx} \in D_{\text{under}}$ mit:

$$(C.179) \quad u_{xy} = g_{\text{under}}(u'_{xy})$$

$$(C.180) \quad u_{yx} = g_{\text{under}}(u'_{yx})$$

Daraus folgt:

$$\begin{aligned}
 g_N(\pi_{\text{under},1}^D(u'_{xy})) &= \pi_{\text{under},1}^H(g_{\text{under}}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
 &= \pi_{\text{under},1}^H(u_{xy}) && (C.179) \\
 &= x && (C.175) \\
 &= \pi_{\text{under},2}^H(u_{yx}) && (C.178) \\
 &= \pi_{\text{under},2}^H(g_{\text{under}}(u'_{yx})) && (C.180) \\
 &= g_N(\pi_{\text{under},2}^D(u'_{yx})) && (g \text{ ist Homomorphismus}) \\
 (C.181) \quad &\Rightarrow \pi_{\text{under},1}^D(u'_{xy}) = \pi_{\text{under},2}^D(u'_{yx}) && (g \text{ ist injektiv nach Lemma B.28})
 \end{aligned}$$

Ferner gilt:

$$\begin{aligned}
 g_N(\pi_{\text{under},2}^D(u'_{xy})) &= \pi_{\text{under},2}^H(g_{\text{under}}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
 &= \pi_{\text{under},2}^H(u_{xy}) && (C.179) \\
 &= y && (C.176) \\
 &= \pi_{\text{under},1}^H(u_{yx}) && (C.177) \\
 &= \pi_{\text{under},1}^H(g_{\text{under}}(u'_{yx})) && (C.180) \\
 &= g_N(\pi_{\text{under},1}^D(u'_{yx})) && (g \text{ ist Homomorphismus}) \\
 (C.182) \quad &\Rightarrow \pi_{\text{under},2}^D(u'_{xy}) = \pi_{\text{under},1}^D(u'_{yx}) && (g \text{ ist injektiv nach Lemma B.28})
 \end{aligned}$$

Nach (C.181) und (C.182) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto \pi_{\text{under},1}^D(u'_{xy}) \\
 y &\mapsto \pi_{\text{under},2}^D(u'_{yx})
 \end{aligned}$$

die Prämisse des Axioms (MP.5) in D . Weil D ein MP-System ist, folgt die Wahrheit der Konklusion:

$$(C.183) \quad \pi_{\text{under},1}^D(u'_{xy}) = \pi_{\text{under},2}^D(u'_{yx})$$

Daraus folgt:

$$x = \pi_{under,1}^H(u_{xy}) \quad (\text{C.175})$$

$$= \pi_{under,1}^H(g_{under}(u'_{xy})) \quad (\text{C.179})$$

$$= g_N(\pi_{under,1}^D(u'_{xy})) \quad (g \text{ ist Homomorphismus})$$

$$= g_N(\pi_{under,2}^D(u'_{xy})) \quad (\text{C.183})$$

$$= \pi_{under,2}^H(g_{under}(u'_{xy})) \quad (g \text{ ist Homomorphismus})$$

$$= \pi_{under,2}^H(u_{xy}) \quad (\text{C.179})$$

$$= y \quad (\text{C.176})$$

(2) Gelte $u'_{xy}, u'_{yx} \in R_{under}$ mit:

$$(C.184) \quad u_{xy} = n_{under}(u'_{xy})$$

$$(C.185) \quad u_{yx} = n_{under}(u'_{yx})$$

Daraus folgt:

$$n_N(\pi_{under,1}^R(u'_{xy})) = \pi_{under,1}^H(n_{under}(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{under,1}^H(u_{xy}) \quad (\text{C.184})$$

$$= x \quad (\text{C.175})$$

$$= \pi_{under,2}^H(u_{yx}) \quad (\text{C.178})$$

$$= \pi_{under,2}^H(n_{under}(u'_{yx})) \quad (\text{C.185})$$

$$(C.186) \quad = n_N(\pi_{under,2}^R(u'_{yx})) \quad (n \text{ ist Homomorphismus})$$

Ebenso gilt:

$$n_N(\pi_{under,2}^R(u'_{xy})) = \pi_{under,2}^H(n_{under}(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{under,2}^H(u_{xy}) \quad (\text{C.184})$$

$$= y \quad (\text{C.176})$$

$$= \pi_{under,1}^H(u_{yx}) \quad (\text{C.177})$$

$$= \pi_{under,1}^H(n_{under}(u'_{yx})) \quad (\text{C.185})$$

$$(C.187) \quad = n_N(\pi_{under,1}^R(u'_{yx})) \quad (n \text{ ist Homomorphismus})$$

Nun gibt es drei Fälle zu unterscheiden:

(a) Es gelte $\pi_{under,1}^R(u'_{xy}) = \pi_{under,2}^R(u'_{yx})$ und $\pi_{under,2}^R(u'_{xy}) = \pi_{under,1}^R(u'_{yx})$. Dann erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto \pi_{\text{under},1}^R(u'_{xy}) \\ y &\mapsto \pi_{\text{under},2}^R(u'_{xy}) \end{aligned}$$

die Prämisse des Axioms (MP.5) in R . Weil R ein MP -System ist, folgt die Wahrheit der Konklusion:

$$(C.188) \quad \pi_{\text{under},1}^R(u'_{xy}) = \pi_{\text{under},2}^R(u'_{xy})$$

Daraus folgt:

$$x = \pi_{\text{under},1}^H(u_{xy}) \quad (C.175)$$

$$= \pi_{\text{under},1}^H(n_{\text{under}}(u'_{xy})) \quad (C.184)$$

$$= n_N(\pi_{\text{under},1}^R(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= n_N(\pi_{\text{under},2}^R(u'_{xy})) \quad (C.188)$$

$$= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{\text{under},2}^H(u_{xy}) \quad (C.184)$$

$$= y \quad (C.176)$$

(b) Es gelte:

$$(C.189) \quad \pi_{\text{under},1}^R(u'_{xy}) \neq \pi_{\text{under},2}^R(u'_{yx})$$

Aus (C.186) und (C.189) folgt nach Lemma B.27, dass $\pi_{\text{under},1}^R(u'_{xy})$ und $\pi_{\text{under},2}^R(u'_{yx})$ Urbilder unter r besitzen. Somit existieren $z_1, z_2 \in K_N$ mit:

$$(C.190) \quad \pi_{\text{under},1}^R(u'_{xy}) = r_N(z_1)$$

$$(C.191) \quad \pi_{\text{under},2}^R(u'_{yx}) = r_N(z_2)$$

Ferner vervollständigt r Vererbung, weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist. Aus (C.190) und (C.191) folgt somit die Existenz zweier Elemente $u''_{xy}, u''_{yx} \in K_{\text{under}}$ mit:

$$(C.192) \quad \pi_{\text{under},1}^K(u''_{xy}) = z_1$$

$$(C.193) \quad r_N(\pi_{\text{under},2}^K(u''_{xy})) = \pi_{\text{under},2}^R(u'_{xy})$$

$$(C.194) \quad \pi_{\text{under},2}^K(u''_{yx}) = z_2$$

$$(C.195) \quad r_N(\pi_{\text{under},1}^K(u''_{yx})) = \pi_{\text{under},1}^R(u'_{yx})$$

Daraus folgt:

$$\begin{aligned}
r_N(\pi_{\text{under},1}^K(u''_{xy})) &= r_N(z_1) & (C.192) \\
(C.196) \quad &= \pi_{\text{under},1}^R(u'_{xy}) & (C.190)
\end{aligned}$$

Analog gilt:

$$\begin{aligned}
r_N(\pi_{\text{under},2}^K(u''_{yx})) &= r_N(z_2) & (C.194) \\
(C.197) \quad &= \pi_{\text{under},2}^R(u'_{yx}) & (C.191)
\end{aligned}$$

Daraus folgt zum einen:

$$\begin{aligned}
g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))) &= g_N(k_N(\pi_{\text{under},1}^K(u''_{xy}))) & (k \text{ ist Homomorphismus}) \\
&= n_N(r_N(\pi_{\text{under},1}^K(u''_{xy}))) & (\text{Pushouts kommutieren}) \\
&= n_N(\pi_{\text{under},1}^R(u'_{xy})) & (C.196) \\
&= n_N(\pi_{\text{under},2}^R(u'_{yx})) & (C.186) \\
&= n_N(r_N(\pi_{\text{under},2}^K(u''_{yx}))) & (C.197) \\
&= g_N(k_N(\pi_{\text{under},2}^K(u''_{yx}))) & (\text{Pushouts kommutieren}) \\
&= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{yx}))) & (k \text{ ist Homomorphismus}) \\
(C.198) \quad & \Rightarrow \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yx})) & (g \text{ ist injektiv nach} \\
& & \text{Lemma B.28})
\end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))) &= g_N(k_N(\pi_{\text{under},2}^K(u''_{xy}))) & (k \text{ ist Homomorphismus}) \\
&= n_N(r_N(\pi_{\text{under},2}^K(u''_{xy}))) & (\text{Pushouts kommutieren}) \\
&= n_N(\pi_{\text{under},2}^R(u'_{xy})) & (C.193) \\
&= n_N(\pi_{\text{under},1}^R(u'_{yx})) & (C.187) \\
&= n_N(r_N(\pi_{\text{under},1}^K(u''_{yx}))) & (C.195) \\
&= g_N(k_N(\pi_{\text{under},1}^K(u''_{yx}))) & (\text{Pushouts kommutieren}) \\
&= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx}))) & (k \text{ ist Homomorphismus}) \\
(C.199) \quad & \Rightarrow \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx})) & (g \text{ ist injektiv nach} \\
& & \text{Lemma B.28})
\end{aligned}$$

Nach (C.198) und (C.199) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) \\ y &\mapsto \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy})) \end{aligned}$$

die Prämisse des Axioms (MP.5) in D . Weil D ein MP-System ist, folgt die Wahrheit der Konklusion:

$$(C.200) \quad \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))$$

Daraus folgt:

$$x = \pi_{\text{under},1}^H(u_{xy}) \quad (C.175)$$

$$= \pi_{\text{under},1}^H(n_{\text{under}}(u'_{xy})) \quad (C.184)$$

$$= n_N(\pi_{\text{under},1}^R(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},1}^K(u''_{xy}))) \quad (C.196)$$

$$= g_N(k_N(\pi_{\text{under},1}^K(u''_{xy}))) \quad (\text{Pushouts kommutieren})$$

$$= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))) \quad (k \text{ ist Homomorphismus})$$

$$= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))) \quad (C.200)$$

$$= g_N(k_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (k \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (\text{Pushouts kommutieren})$$

$$= n_N(\pi_{\text{under},2}^R(u'_{xy})) \quad (C.193)$$

$$= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{\text{under},2}^H(u_{xy}) \quad (C.184)$$

$$= y \quad (C.176)$$

(c) Es gelte:

$$(C.201) \quad \pi_{\text{under},2}^R(u'_{xy}) \neq \pi_{\text{under},1}^R(u'_{yx})$$

Aus (C.187) und (C.201) folgt nach Lemma B.27, dass $\pi_{\text{under},2}^R(u'_{xy})$ und $\pi_{\text{under},1}^R(u'_{yx})$ Urbilder unter r besitzen. Somit existieren $z_1, z_2 \in K_N$ mit:

$$(C.202) \quad \pi_{\text{under},2}^R(u'_{xy}) = r_N(z_1)$$

$$(C.203) \quad \pi_{\text{under},1}^R(u'_{yx}) = r_N(z_2)$$

Ferner vervollständigt r Vererbung, weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist. Aus (C.202) und (C.203) folgt somit die Existenz zweier Elemente $u''_{xy}, u''_{yx} \in K_{\text{under}}$ mit:

$$(C.204) \quad \pi_{\text{under},2}^K(u''_{xy}) = z_1$$

$$(C.205) \quad r_N(\pi_{\text{under},1}^K(u''_{xy})) = \pi_{\text{under},1}^R(u'_{xy})$$

$$(C.206) \quad \pi_{\text{under},1}^K(u''_{yx}) = z_2$$

$$(C.207) \quad r_N(\pi_{\text{under},2}^K(u''_{yx})) = \pi_{\text{under},2}^R(u'_{yx})$$

Daraus folgt:

$$r_N(\pi_{\text{under},2}^K(u''_{xy})) = r_N(z_1) \quad (C.204)$$

$$(C.208) \quad = \pi_{\text{under},2}^R(u'_{xy}) \quad (C.202)$$

Analog gilt:

$$r_N(\pi_{\text{under},1}^K(u''_{yx})) = r_N(z_2) \quad (C.206)$$

$$(C.209) \quad = \pi_{\text{under},1}^R(u'_{yx}) \quad (C.203)$$

Daraus folgt zum einen:

$$g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))) = g_N(k_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (k \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (\text{Pushouts kommutieren})$$

$$= n_N(\pi_{\text{under},2}^R(u'_{xy})) \quad (C.208)$$

$$= n_N(\pi_{\text{under},1}^R(u'_{yx})) \quad (C.187)$$

$$= n_N(r_N(\pi_{\text{under},1}^K(u''_{yx}))) \quad (C.209)$$

$$= g_N(k_N(\pi_{\text{under},1}^K(u''_{yx}))) \quad (\text{Pushouts kommutieren})$$

$$= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx}))) \quad (k \text{ ist Homomorphismus})$$

(C.210)

$$\Rightarrow \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx})) \quad (g \text{ ist injektiv nach}$$

Lemma B.28)

Zum anderen gilt:

$$g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))) = g_N(k_N(\pi_{\text{under},1}^K(u''_{xy}))) \quad (k \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},1}^K(u''_{xy}))) \quad (\text{Pushouts kommutieren})$$

$$= n_N(\pi_{\text{under},1}^R(u'_{xy})) \quad (C.205)$$

$$= n_N(\pi_{\text{under},2}^R(u'_{yx})) \quad (C.186)$$

$$= n_N(r_N(\pi_{\text{under},2}^K(u''_{yx}))) \quad (C.207)$$

$$= g_N(k_N(\pi_{\text{under},2}^K(u''_{yx}))) \quad (\text{Pushouts kommutieren})$$

$$\begin{aligned}
&= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{yx}))) \quad (k \text{ ist Homomorphismus}) \\
\text{(C.211)} \quad &\Rightarrow \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yx})) \quad (g \text{ ist injektiv nach} \\
&\quad \text{Lemma B.28})
\end{aligned}$$

Nach (C.210) und (C.211) erfüllt die Variablenbelegung

$$\begin{aligned}
x &\mapsto \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) \\
y &\mapsto \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))
\end{aligned}$$

die Prämisse des Axioms (MP.5) in D . Weil D ein MP-System ist, folgt die Wahrheit der Konklusion:

$$\text{(C.212)} \quad \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))$$

Daraus folgt:

$$\begin{aligned}
x &= \pi_{\text{under},1}^H(u_{xy}) && \text{(C.175)} \\
&= \pi_{\text{under},1}^H(n_{\text{under}}(u'_{xy})) && \text{(C.184)} \\
&= n_N(\pi_{\text{under},1}^R(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= n_N(r_N(\pi_{\text{under},1}^K(u''_{xy}))) && \text{(C.205)} \\
&= g_N(k_N(\pi_{\text{under},1}^K(u''_{xy}))) && \text{(Pushouts kommutieren)} \\
&= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))) && (k \text{ ist Homomorphismus}) \\
&= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))) && \text{(C.212)} \\
&= g_N(k_N(\pi_{\text{under},2}^K(u''_{xy}))) && (k \text{ ist Homomorphismus}) \\
&= n_N(r_N(\pi_{\text{under},2}^K(u''_{xy}))) && \text{(Pushouts kommutieren)} \\
&= n_N(\pi_{\text{under},2}^R(u'_{xy})) && \text{(C.208)} \\
&= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{\text{under},2}^H(u_{xy}) && \text{(C.184)} \\
&= y && \text{(C.176)}
\end{aligned}$$

(3) Gelte $u'_{xy} \in R_{\text{under}}$ und $u'_{yx} \in D_{\text{under}}$ mit:

$$\text{(C.213)} \quad u_{xy} = n_{\text{under}}(u'_{xy})$$

$$\text{(C.214)} \quad u_{yx} = g_{\text{under}}(u'_{yx})$$

Daraus folgt:

$$\begin{aligned}
n_N(\pi_{\text{under},1}^R(u'_{xy})) &= \pi_{\text{under},1}^H(n_{\text{under}}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{\text{under},1}^H(u_{xy}) && \text{(C.213)} \\
&= x && \text{(C.175)} \\
&= \pi_{\text{under},2}^H(u_{yx}) && \text{(C.178)} \\
&= \pi_{\text{under},2}^H(g_{\text{under}}(u'_{yx})) && \text{(C.214)} \\
\text{(C.215)} \quad &= g_N(\pi_{\text{under},2}^D(u'_{yx})) && (g \text{ ist Homomorphismus})
\end{aligned}$$

Aus (C.215) und Lemma B.29 folgt die Existenz eines eindeutigen Elements $z \in K_N$ mit:

$$\text{(C.216)} \quad \pi_{\text{under},1}^R(u'_{xy}) = r_N(z)$$

$$\text{(C.217)} \quad \pi_{\text{under},2}^D(u'_{yx}) = k_N(z)$$

Ferner vervollständigt r Vererbung, weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist. Aus (C.216) folgt somit die Existenz eines Elements $u''_{xy} \in K_{\text{under}}$ mit:

$$\text{(C.218)} \quad \pi_{\text{under},1}^K(u''_{xy}) = z$$

$$\text{(C.219)} \quad r_N(\pi_{\text{under},2}^K(u''_{xy})) = \pi_{\text{under},2}^R(u'_{xy})$$

Daraus folgt:

$$r_N(\pi_{\text{under},1}^K(u''_{xy})) = r_N(z) \quad \text{(C.218)}$$

$$\text{(C.220)} \quad = \pi_{\text{under},1}^R(u'_{xy}) \quad \text{(C.216)}$$

Nun gilt zum einen:

$$\begin{aligned}
\pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) &= k_N(\pi_{\text{under},1}^K(u''_{xy})) && (k \text{ ist Homomorphismus}) \\
&= k_N(z) && \text{(C.218)}
\end{aligned}$$

$$\text{(C.221)} \quad = \pi_{\text{under},2}^D(u'_{yx}) \quad \text{(C.217)}$$

Zum anderen gilt:

$$\begin{aligned}
g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))) &= g_N(k_N(\pi_{\text{under},2}^K(u''_{xy}))) && (k \text{ ist Homomorphismus}) \\
&= n_N(r_N(\pi_{\text{under},2}^K(u''_{xy}))) && \text{(Pushouts kommutieren)} \\
&= n_N(\pi_{\text{under},2}^R(u'_{yx})) && \text{(C.219)} \\
&= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{yx})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{\text{under},2}^H(u_{yx}) && \text{(C.213)} \\
&= y && \text{(C.176)}
\end{aligned}$$

$$= \pi_{\text{under},1}^H(u_{yx}) \quad (\text{C.177})$$

$$= \pi_{\text{under},1}^H(g_{\text{under}}(u'_{yx})) \quad (\text{C.214})$$

$$= g_N(\pi_{\text{under},1}^D(u'_{yx})) \quad (g \text{ ist Homomorphismus})$$

$$(C.222) \quad \Rightarrow \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},1}^D(u'_{yx}) \quad (g \text{ ist injektiv nach Lemma B.28})$$

Nach (C.221) und (C.222) erfüllt die Variablenbelegung

$$x \mapsto \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))$$

$$y \mapsto \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))$$

die Prämisse des Axioms (MP.5) in D . Weil D ein MP -System ist, folgt die Wahrheit der Konklusion:

$$(C.223) \quad \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))$$

Daraus folgt:

$$x = \pi_{\text{under},1}^H(u_{xy}) \quad (\text{C.175})$$

$$= \pi_{\text{under},1}^H(n_{\text{under}}(u'_{xy})) \quad (\text{C.213})$$

$$= n_N(\pi_{\text{under},1}^R(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},1}^K(u''_{xy}))) \quad (\text{C.220})$$

$$= g_N(k_N(\pi_{\text{under},1}^K(u''_{xy}))) \quad (\text{Pushouts kommutieren})$$

$$= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))) \quad (k \text{ ist Homomorphismus})$$

$$= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))) \quad (\text{C.223})$$

$$= g_N(k_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (k \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (\text{Pushouts kommutieren})$$

$$= n_N(\pi_{\text{under},2}^R(u'_{xy})) \quad (\text{C.219})$$

$$= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{\text{under},2}^H(u_{xy}) \quad (\text{C.213})$$

$$= y \quad (\text{C.176})$$

(4) Gelte $u'_{xy} \in D_{\text{under}}$ und $u'_{yx} \in R_{\text{under}}$ mit:

$$(C.224) \quad u_{xy} = g_{\text{under}}(u'_{xy})$$

$$(C.225) \quad u_{yx} = n_{\text{under}}(u'_{yx})$$

Daraus folgt:

$$\begin{aligned}
g_N(\pi_{under,1}^D(u'_{xy})) &= \pi_{under,1}^H(g_{under}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{under,1}^H(u_{xy}) && (C.224) \\
&= x && (C.175) \\
&= \pi_{under,2}^H(u_{yx}) && (C.178) \\
&= \pi_{under,2}^H(n_{under}(u'_{yx})) && (C.225) \\
(C.226) \quad &= n_N(\pi_{under,2}^R(u'_{yx})) && (n \text{ ist Homomorphismus})
\end{aligned}$$

Aus (C.226) und Lemma B.29 folgt die Existenz eines eindeutigen Elements $z \in K_N$ mit:

$$(C.227) \quad \pi_{under,1}^D(u'_{xy}) = k_N(z)$$

$$(C.228) \quad \pi_{under,2}^R(u'_{yx}) = r_N(z)$$

Ferner vervollständigt r Vererbung, weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist. Aus (C.228) folgt somit die Existenz eines Elements $u''_{yx} \in K_{under}$ mit:

$$(C.229) \quad \pi_{under,2}^K(u''_{yx}) = z$$

$$(C.230) \quad r_N(\pi_{under,1}^K(u''_{yx})) = \pi_{under,1}^R(u'_{yx})$$

Daraus folgt:

$$(C.231) \quad r_N(\pi_{under,2}^K(u''_{yx})) = r_N(z) \quad (C.229)$$

$$= \pi_{under,2}^R(u'_{yx}) \quad (C.228)$$

Nun gilt zum einen:

$$\pi_{under,2}^D(k_{under}(u''_{yx})) = k_N(\pi_{under,2}^K(u''_{yx})) \quad (k \text{ ist Homomorphismus})$$

$$= k_N(z) \quad (C.229)$$

$$(C.232) \quad = \pi_{under,1}^D(u'_{xy}) \quad (C.227)$$

Zum anderen gilt:

$$g_N(\pi_{under,1}^D(k_{under}(u''_{yx}))) = g_N(k_N(\pi_{under,1}^K(u''_{yx}))) \quad (k \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{under,1}^K(u''_{yx}))) \quad (\text{Pushouts kommutieren})$$

$$= n_N(\pi_{under,1}^R(u'_{yx})) \quad (C.230)$$

$$= \pi_{under,1}^H(n_{under}(u'_{yx})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{\text{under},1}^H(u_{yx}) \quad (\text{C.225})$$

$$= y \quad (\text{C.177})$$

$$= \pi_{\text{under},2}^H(u_{xy}) \quad (\text{C.176})$$

$$= \pi_{\text{under},2}^H(g_{\text{under}}(u'_{xy})) \quad (\text{C.224})$$

$$= g_N(\pi_{\text{under},2}^D(u'_{xy})) \quad (g \text{ ist Homomorphismus})$$

$$(C.233) \quad \Rightarrow \pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx})) = \pi_{\text{under},2}^D(u'_{xy}) \quad (g \text{ ist injektiv nach Lemma B.28})$$

Nach (C.232) und (C.233) erfüllt die Variablenbelegung

$$x \mapsto \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yx}))$$

$$y \mapsto \pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx}))$$

die Prämisse des Axioms (MP.5) in D . Weil D ein MP-System ist, folgt die Wahrheit der Konklusion:

$$(C.234) \quad \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yx})) = \pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx}))$$

Daraus folgt:

$$x = \pi_{\text{under},2}^H(u_{yx}) \quad (\text{C.178})$$

$$= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{yx})) \quad (\text{C.225})$$

$$= n_N(\pi_{\text{under},2}^R(u'_{yx})) \quad (n \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},2}^K(u''_{yx}))) \quad (\text{C.231})$$

$$= g_N(k_N(\pi_{\text{under},2}^K(u''_{yx}))) \quad (\text{Pushouts kommutieren})$$

$$= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{yx}))) \quad (k \text{ ist Homomorphismus})$$

$$= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{yx}))) \quad (\text{C.234})$$

$$= g_N(k_N(\pi_{\text{under},1}^K(u''_{yx}))) \quad (k \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},1}^K(u''_{yx}))) \quad (\text{Pushouts kommutieren})$$

$$= n_N(\pi_{\text{under},1}^R(u'_{yx})) \quad (\text{C.230})$$

$$= \pi_{\text{under},1}^H(n_{\text{under}}(u'_{yx})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{\text{under},1}^H(u_{yx}) \quad (\text{C.225})$$

$$= y \quad (\text{C.177})$$

□

C.2.2 Beweis von Lemma 10.25

Seien $x, y, z \in H_N$ gegeben mit $(x, y) \in \text{under}^H$ und $(y, z) \in \text{under}^H$. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz zweier Elemente $u_{xy}, u_{yz} \in H_{\text{under}}$ mit:

$$(C.235) \quad \pi_{\text{under},1}^H(u_{xy}) = x$$

$$(C.236) \quad \pi_{\text{under},2}^H(u_{xy}) = y$$

$$(C.237) \quad \pi_{\text{under},1}^H(u_{yz}) = y$$

$$(C.238) \quad \pi_{\text{under},2}^H(u_{yz}) = z$$

Zu zeigen ist, dass $(x, z) \in \text{under}^H$ gilt. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz eines Elements $u_{xz} \in H_{\text{under}}$, das die Gleichungen $\pi_{\text{under},1}^H(u_{xz}) = x$ und $\pi_{\text{under},2}^H(u_{xz}) = z$ erfüllt.

Nach Lemma B.26 existiert ein Urbild u'_{xy} von u_{xy} und ein Urbild u'_{yz} von u_{yz} unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $u'_{xy}, u'_{yz} \in D_{\text{under}}$ mit:

$$(C.239) \quad u_{xy} = g_{\text{under}}(u'_{xy})$$

$$(C.240) \quad u_{yz} = g_{\text{under}}(u'_{yz})$$

Daraus folgt:

$$\begin{aligned}
 g_N(\pi_{\text{under},2}^D(u'_{xy})) &= \pi_{\text{under},2}^H(g_{\text{under}}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
 &= \pi_{\text{under},2}^H(u_{xy}) && (C.239) \\
 &= y && (C.236) \\
 &= \pi_{\text{under},1}^H(u_{yz}) && (C.237) \\
 &= \pi_{\text{under},1}^H(g_{\text{under}}(u'_{yz})) && (C.240) \\
 &= g_N(\pi_{\text{under},1}^D(u'_{yz})) && (g \text{ ist Homomorphismus}) \\
 (C.241) \quad &\Rightarrow \pi_{\text{under},2}^D(u'_{xy}) = \pi_{\text{under},1}^D(u'_{yz}) && (g \text{ ist injektiv nach Lemma B.28})
 \end{aligned}$$

Nach (C.241) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto \pi_{\text{under},1}^D(u'_{xy}) \\
 y &\mapsto \pi_{\text{under},2}^D(u'_{xy}) \\
 z &\mapsto \pi_{\text{under},2}^D(u'_{yz})
 \end{aligned}$$

die Prämisse des Axioms (MP.6) in D . Weil D ein MP -System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $u'_{xz} \in D_{\text{under}}$ mit:

$$(C.242) \quad \pi_{\text{under},1}^D(u'_{xz}) = \pi_{\text{under},1}^D(u'_{xy})$$

$$(C.243) \quad \pi_{\text{under},2}^D(u'_{xz}) = \pi_{\text{under},2}^D(u'_{yz})$$

Daraus folgt zum einen:

$$\begin{aligned} \pi_{\text{under},1}^H(g_{\text{under}}(u'_{xz})) &= g_N(\pi_{\text{under},1}^D(u'_{xz})) && (g \text{ ist Homomorphismus}) \\ &= g_N(\pi_{\text{under},1}^D(u'_{xy})) && (C.242) \\ &= \pi_{\text{under},1}^H(g_{\text{under}}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\ &= \pi_{\text{under},1}^H(u_{xy}) && (C.239) \\ (C.244) \quad &= x && (C.235) \end{aligned}$$

Zum anderen gilt:

$$\begin{aligned} \pi_{\text{under},2}^H(g_{\text{under}}(u'_{xz})) &= g_N(\pi_{\text{under},2}^D(u'_{xz})) && (g \text{ ist Homomorphismus}) \\ &= g_N(\pi_{\text{under},2}^D(u'_{yz})) && (C.243) \\ &= \pi_{\text{under},2}^H(g_{\text{under}}(u'_{yz})) && (g \text{ ist Homomorphismus}) \\ &= \pi_{\text{under},2}^H(u_{yz}) && (C.240) \\ (C.245) \quad &= z && (C.238) \end{aligned}$$

Aus den Gleichungen (C.244) und (C.245) folgt $u_{xz} ::= g_{\text{under}}(u'_{xz})$.

(2) Gelte $u'_{xy}, u'_{yz} \in R_{\text{under}}$ mit:

$$(C.246) \quad u_{xy} = n_{\text{under}}(u'_{xy})$$

$$(C.247) \quad u_{yz} = n_{\text{under}}(u'_{yz})$$

Nun gibt es zwei Fälle zu unterscheiden:

(a) Es gelte $\pi_{\text{under},2}^R(u'_{xy}) = \pi_{\text{under},1}^R(u'_{yz})$. Dann erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto \pi_{\text{under},1}^R(u'_{xy}) \\ y &\mapsto \pi_{\text{under},2}^R(u'_{xy}) \\ z &\mapsto \pi_{\text{under},2}^R(u'_{yz}) \end{aligned}$$

die Prämisse des Axioms (MP.6) in R . Weil R ein MP-System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $u'_{xz} \in R_{\text{under}}$ mit:

$$(C.248) \quad \pi_{\text{under},1}^R(u'_{xz}) = \pi_{\text{under},1}^R(u'_{xy})$$

$$(C.249) \quad \pi_{\text{under},2}^R(u'_{xz}) = \pi_{\text{under},2}^R(u'_{yz})$$

Daraus folgt zum einen:

$$\begin{aligned}
 \pi_{under,1}^H(n_{under}(u'_{xz})) &= n_N(\pi_{under,1}^R(u'_{xz})) && (n \text{ ist Homomorphismus}) \\
 &= n_N(\pi_{under,1}^R(u'_{xy})) && \text{(C.248)} \\
 &= \pi_{under,1}^H(n_{under}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
 &= \pi_{under,1}^H(u_{xy}) && \text{(C.246)} \\
 \text{(C.250)} \quad &= x && \text{(C.235)}
 \end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
 \pi_{under,2}^H(n_{under}(u'_{xz})) &= n_N(\pi_{under,2}^R(u'_{xz})) && (n \text{ ist Homomorphismus}) \\
 &= n_N(\pi_{under,2}^R(u'_{yz})) && \text{(C.249)} \\
 &= \pi_{under,2}^H(n_{under}(u'_{yz})) && (n \text{ ist Homomorphismus}) \\
 &= \pi_{under,2}^H(u_{yz}) && \text{(C.247)} \\
 \text{(C.251)} \quad &= z && \text{(C.238)}
 \end{aligned}$$

Aus den Gleichungen (C.250) und (C.251) folgt $u_{xz} ::= n_{under}(u'_{xz})$.

(b) Es gelte:

$$\text{(C.252)} \quad \pi_{under,2}^R(u'_{xy}) \neq \pi_{under,1}^R(u'_{yz})$$

Dann folgt zunächst:

$$\begin{aligned}
 n_N(\pi_{under,2}^R(u'_{xy})) &= \pi_{under,2}^H(n_{under}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
 &= \pi_{under,2}^H(u_{xy}) && \text{(C.246)} \\
 &= y && \text{(C.236)} \\
 &= \pi_{under,1}^H(u_{yz}) && \text{(C.237)} \\
 &= \pi_{under,1}^H(n_{under}(u'_{yz})) && \text{(C.247)} \\
 \text{(C.253)} \quad &= n_N(\pi_{under,1}^R(u'_{yz})) && (n \text{ ist Homomorphismus})
 \end{aligned}$$

Aus (C.252) und (C.253) folgt nach Lemma B.27, dass $\pi_{under,2}^R(u'_{xy})$ und $\pi_{under,1}^R(u'_{yz})$ Urbilder unter r besitzen. Somit existieren $z_1, z_2 \in K_N$ mit:

$$\text{(C.254)} \quad \pi_{under,2}^R(u'_{xy}) = r_N(z_1)$$

$$\text{(C.255)} \quad \pi_{under,1}^R(u'_{yz}) = r_N(z_2)$$

Ferner vervollständigt r Vererbung, weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist. Aus (C.254) und (C.255) folgt somit die Existenz zweier Elemente $u''_{xy}, u''_{yz} \in K_{\text{under}}$ mit:

$$(C.256) \quad \pi_{\text{under},2}^K(u''_{xy}) = z_1$$

$$(C.257) \quad r_N(\pi_{\text{under},1}^K(u''_{xy})) = \pi_{\text{under},1}^R(u'_{xy})$$

$$(C.258) \quad \pi_{\text{under},1}^K(u''_{yz}) = z_2$$

$$(C.259) \quad r_N(\pi_{\text{under},2}^K(u''_{yz})) = \pi_{\text{under},2}^R(u'_{yz})$$

Daraus folgt:

$$r_N(\pi_{\text{under},2}^K(u''_{xy})) = r_N(z_1) \quad (C.256)$$

$$(C.260) \quad = \pi_{\text{under},2}^R(u'_{xy}) \quad (C.254)$$

Analog gilt:

$$r_N(\pi_{\text{under},1}^K(u''_{yz})) = r_N(z_2) \quad (C.258)$$

$$(C.261) \quad = \pi_{\text{under},1}^R(u'_{yz}) \quad (C.255)$$

Ferner gilt:

$$g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))) = g_N(k_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (k \text{ ist Homomorphismus})$$

$$= n_N(r_N(\pi_{\text{under},2}^K(u''_{xy}))) \quad (\text{Pushouts kommutieren})$$

$$= n_N(\pi_{\text{under},2}^R(u'_{xy})) \quad (C.260)$$

$$= n_N(\pi_{\text{under},1}^R(u'_{yz})) \quad (C.253)$$

$$= n_N(r_N(\pi_{\text{under},1}^K(u''_{yz}))) \quad (C.261)$$

$$= g_N(k_N(\pi_{\text{under},1}^K(u''_{yz}))) \quad (\text{Pushouts kommutieren})$$

$$= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{yz}))) \quad (k \text{ ist Homomorphismus})$$

(C.262)

$$\Rightarrow \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy})) = \pi_{\text{under},1}^D(k_{\text{under}}(u''_{yz})) \quad (g \text{ ist injektiv nach Lemma B.28})$$

Nach (C.262) erfüllt die Variablenbelegung

$$x \mapsto \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))$$

$$y \mapsto \pi_{\text{under},2}^D(k_{\text{under}}(u''_{xy}))$$

$$z \mapsto \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yz}))$$

die Prämisse des Axioms (MP.6) in D . Weil D ein MP -System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $u'_{xz} \in D_{\text{under}}$ mit:

$$(C.263) \quad \pi_{\text{under},1}^D(u'_{xz}) = \pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))$$

$$(C.264) \quad \pi_{\text{under},2}^D(u'_{xz}) = \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yz}))$$

Daraus folgt zum einen:

$$\begin{aligned} \pi_{\text{under},1}^H(g_{\text{under}}(u'_{xz})) &= g_N(\pi_{\text{under},1}^D(u'_{xz})) && (g \text{ ist Homomorphismus}) \\ &= g_N(\pi_{\text{under},1}^D(k_{\text{under}}(u''_{xy}))) && (C.263) \\ &= g_N(k_N(\pi_{\text{under},1}^K(u''_{xy}))) && (k \text{ ist Homomorphismus}) \\ &= n_N(r_N(\pi_{\text{under},1}^K(u''_{xy}))) && (\text{Pushouts kommutieren}) \\ &= n_N(\pi_{\text{under},1}^R(u'_{xy})) && (C.257) \\ &= \pi_{\text{under},1}^H(n_{\text{under}}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\ &= \pi_{\text{under},1}^H(u_{xy}) && (C.246) \\ (C.265) \quad &= x && (C.235) \end{aligned}$$

Zum anderen gilt:

$$\begin{aligned} \pi_{\text{under},2}^H(g_{\text{under}}(u'_{xz})) &= g_N(\pi_{\text{under},2}^D(u'_{xz})) && (g \text{ ist Homomorphismus}) \\ &= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{yz}))) && (C.264) \\ &= g_N(k_N(\pi_{\text{under},2}^K(u''_{yz}))) && (k \text{ ist Homomorphismus}) \\ &= n_N(r_N(\pi_{\text{under},2}^K(u''_{yz}))) && (\text{Pushouts kommutieren}) \\ &= n_N(\pi_{\text{under},2}^R(u'_{yz})) && (C.259) \\ &= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{yz})) && (n \text{ ist Homomorphismus}) \\ &= \pi_{\text{under},2}^H(u_{yz}) && (C.247) \\ (C.266) \quad &= z && (C.238) \end{aligned}$$

Aus den Gleichungen (C.265) und (C.266) folgt $u_{xz} ::= g_{\text{under}}(u'_{xz})$.

(3) Gelte $u'_{xy} \in R_{\text{under}}$ und $u'_{yz} \in D_{\text{under}}$ mit:

$$(C.267) \quad u_{xy} = n_{\text{under}}(u'_{xy})$$

$$(C.268) \quad u_{yz} = g_{\text{under}}(u'_{yz})$$

Daraus folgt:

$$\begin{aligned}
n_N(\pi_{under,2}^R(u'_{xy})) &= \pi_{under,2}^H(n_{under}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{under,2}^H(u_{xy}) && (C.267) \\
&= y && (C.236) \\
&= \pi_{under,1}^H(u_{yz}) && (C.237) \\
&= \pi_{under,1}^H(g_{under}(u'_{yz})) && (C.268) \\
(C.269) \quad &= g_N(\pi_{under,1}^D(u'_{yz})) && (g \text{ ist Homomorphismus})
\end{aligned}$$

Aus (C.269) und Lemma B.29 folgt die Existenz eines eindeutigen Elements $z \in K_N$ mit:

$$(C.270) \quad \pi_{under,2}^R(u'_{xy}) = r_N(z)$$

$$(C.271) \quad \pi_{under,1}^D(u'_{yz}) = k_N(z)$$

Ferner vervollständigt r Vererbung, weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist. Aus (C.270) folgt somit die Existenz eines Elements $u''_{xy} \in K_{under}$ mit:

$$(C.272) \quad \pi_{under,2}^K(u''_{xy}) = z$$

$$(C.273) \quad r_N(\pi_{under,1}^K(u''_{xy})) = \pi_{under,1}^R(u'_{xy})$$

Daraus folgt:

$$\begin{aligned}
\pi_{under,2}^D(k_{under}(u''_{xy})) &= k_N(\pi_{under,2}^K(u''_{xy})) && (k \text{ ist Homomorphismus}) \\
&= k_N(z) && (C.272)
\end{aligned}$$

$$(C.274) \quad = \pi_{under,1}^D(u'_{yz}) \quad (C.271)$$

Nach (C.274) erfüllt die Variablenbelegung

$$\begin{aligned}
x &\mapsto \pi_{under,1}^D(k_{under}(u''_{xy})) \\
y &\mapsto \pi_{under,2}^D(k_{under}(u''_{xy})) \\
z &\mapsto \pi_{under,2}^D(u'_{yz})
\end{aligned}$$

die Prämisse des Axioms (MP.6) in D . Weil D ein MP -System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $u'_{xz} \in D_{under}$ mit:

$$(C.275) \quad \pi_{under,1}^D(u'_{xz}) = \pi_{under,1}^D(k_{under}(u''_{xy}))$$

$$(C.276) \quad \pi_{under,2}^D(u'_{xz}) = \pi_{under,2}^D(u'_{yz})$$

Daraus folgt zum einen:

$$\begin{aligned}
\pi_{under,1}^H(g_{under}(u'_{xz})) &= g_N(\pi_{under,1}^D(u'_{xz})) && (g \text{ ist Homomorphismus}) \\
&= g_N(\pi_{under,1}^D(k_{under}(u''_{xy}))) && \text{(C.275)} \\
&= g_N(k_N(\pi_{under,1}^K(u''_{xy}))) && (k \text{ ist Homomorphismus}) \\
&= n_N(r_N(\pi_{under,1}^K(u''_{xy}))) && (\text{Pushouts kommutieren}) \\
&= n_N(\pi_{under,1}^R(u'_{xy})) && \text{(C.273)} \\
&= \pi_{under,1}^H(n_{under}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{under,1}^H(u_{xy}) && \text{(C.267)} \\
\text{(C.277)} \quad &= x && \text{(C.235)}
\end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
\pi_{under,2}^H(g_{under}(u'_{xz})) &= g_N(\pi_{under,2}^D(u'_{xz})) && (g \text{ ist Homomorphismus}) \\
&= g_N(\pi_{under,2}^D(u'_{yz})) && \text{(C.276)} \\
&= \pi_{under,2}^H(g_{under}(u'_{yz})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{under,2}^H(u_{yz}) && \text{(C.268)} \\
\text{(C.278)} \quad &= z && \text{(C.238)}
\end{aligned}$$

Aus den Gleichungen (C.277) und (C.278) folgt $u_{xz} ::= g_{under}(u'_{xz})$.

(4) Gelte $u'_{xy} \in D_{under}$ und $u'_{yz} \in R_{under}$ mit:

$$\text{(C.279)} \quad u_{xy} = g_{under}(u'_{xy})$$

$$\text{(C.280)} \quad u_{yz} = n_{under}(u'_{yz})$$

Daraus folgt:

$$\begin{aligned}
g_N(\pi_{under,2}^D(u'_{xy})) &= \pi_{under,2}^H(g_{under}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{under,2}^H(u_{xy}) && \text{(C.279)} \\
&= y && \text{(C.236)} \\
&= \pi_{under,1}^H(u_{yz}) && \text{(C.237)} \\
&= \pi_{under,1}^H(n_{under}(u'_{yz})) && \text{(C.280)} \\
\text{(C.281)} \quad &= n_N(\pi_{under,1}^R(u'_{yz})) && (g \text{ ist Homomorphismus})
\end{aligned}$$

Aus (C.281) und Lemma B.29 folgt die Existenz eines eindeutigen Elements $z \in K_N$ mit:

$$\text{(C.282)} \quad \pi_{under,2}^D(u'_{xy}) = k_N(z)$$

$$\text{(C.283)} \quad \pi_{under,1}^R(u'_{yz}) = r_N(z)$$

Ferner vervollständigt r Vererbung, weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist. Aus (C.283) folgt somit die Existenz eines Elements $u''_{yz} \in K_{\text{under}}$ mit:

$$(C.284) \quad \pi_{\text{under},1}^K(u''_{yz}) = z$$

$$(C.285) \quad r_N(\pi_{\text{under},2}^K(u''_{yz})) = \pi_{\text{under},2}^R(u'_{yz})$$

Daraus folgt:

$$\pi_{\text{under},1}^D(k_{\text{under}}(u''_{yz})) = k_N(\pi_{\text{under},1}^K(u''_{yz})) \quad (k \text{ ist Homomorphismus})$$

$$= k_N(z) \quad (C.284)$$

$$(C.286) \quad = \pi_{\text{under},2}^D(u'_{xy}) \quad (C.282)$$

Nach (C.286) erfüllt die Variablenbelegung

$$x \mapsto \pi_{\text{under},1}^D(u'_{xy})$$

$$y \mapsto \pi_{\text{under},1}^D(k_{\text{under}}(u''_{yz}))$$

$$z \mapsto \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yz}))$$

die Prämisse des Axioms (MP.6) in D . Weil D ein MP-System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $u'_{xz} \in D_{\text{under}}$ mit:

$$(C.287) \quad \pi_{\text{under},1}^D(u'_{xz}) = \pi_{\text{under},1}^D(u'_{xy})$$

$$(C.288) \quad \pi_{\text{under},2}^D(u'_{xz}) = \pi_{\text{under},2}^D(k_{\text{under}}(u''_{yz}))$$

Daraus folgt zum einen:

$$\pi_{\text{under},1}^H(g_{\text{under}}(u'_{xz})) = g_N(\pi_{\text{under},1}^D(u'_{xz})) \quad (g \text{ ist Homomorphismus})$$

$$= g_N(\pi_{\text{under},1}^D(u'_{xy})) \quad (C.287)$$

$$= \pi_{\text{under},1}^H(g_{\text{under}}(u'_{xy})) \quad (g \text{ ist Homomorphismus})$$

$$= \pi_{\text{under},1}^H(u_{xy}) \quad (C.279)$$

$$(C.289) \quad = x \quad (C.235)$$

Zum anderen gilt:

$$\pi_{\text{under},2}^H(g_{\text{under}}(u'_{xz})) = g_N(\pi_{\text{under},2}^D(u'_{xz})) \quad (g \text{ ist Homomorphismus})$$

$$= g_N(\pi_{\text{under},2}^D(k_{\text{under}}(u''_{yz}))) \quad (C.288)$$

$$= g_N(k_N(\pi_{\text{under},2}^K(u''_{yz}))) \quad (k \text{ ist Homomorphismus})$$

$$\begin{aligned}
&= n_N(r_N(\pi_{\text{under},2}^K(u''_{yz}))) && \text{(Pushouts kommutieren)} \\
&= n_N(\pi_{\text{under},2}^R(u'_{yz})) && \text{(C.285)} \\
&= \pi_{\text{under},2}^H(n_{\text{under}}(u'_{yz})) && (n \text{ ist Homomorphismus)} \\
&= \pi_{\text{under},2}^H(u_{yz}) && \text{(C.280)} \\
\text{(C.290)} \quad &= z && \text{(C.238)}
\end{aligned}$$

Aus den Gleichungen (C.289) und (C.290) folgt $u_{xz} ::= g_{\text{under}}(u'_{xz})$. \square

C.2.3 Beweis von Lemma 10.26

Seien $x, y \in H_N$ gegeben mit $(x, y) \in \text{rel}^H$. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz eines Elements $u_{xy} \in H_{\text{rel}}$ mit:

$$\text{(C.291)} \quad \pi_{\text{rel},1}^H(u_{xy}) = x$$

$$\text{(C.292)} \quad \pi_{\text{rel},2}^H(u_{xy}) = y$$

Zu zeigen ist, dass $(y, x) \in \text{rel}^H$ gilt. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz eines Elements $u_{yx} \in H_{\text{rel}}$, das die Gleichungen $\pi_{\text{rel},1}^H(u_{yx}) = y$ und $\pi_{\text{rel},2}^H(u_{yx}) = x$ erfüllt.

Nach Lemma B.26 existiert ein Urbild u'_{xy} von u_{xy} unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $u'_{xy} \in D_{\text{rel}}$ mit:

$$\text{(C.293)} \quad u_{xy} = g_{\text{rel}}(u'_{xy})$$

Nach (C.293) erfüllt die Variablenbelegung

$$x \mapsto \pi_{\text{rel},1}^D(u'_{xy})$$

$$y \mapsto \pi_{\text{rel},2}^D(u'_{xy})$$

die Prämisse des Axioms (MP.7) in D . Weil D ein MP-System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $u'_{yx} \in D_{\text{rel}}$ mit:

$$\text{(C.294)} \quad \pi_{\text{rel},1}^D(u'_{yx}) = \pi_{\text{rel},2}^D(u'_{xy})$$

$$\text{(C.295)} \quad \pi_{\text{rel},2}^D(u'_{yx}) = \pi_{\text{rel},1}^D(u'_{xy})$$

Daraus folgt zum einen:

$$\begin{aligned}
\pi_{rel,1}^H(g_{rel}(u'_{yx})) &= g_N(\pi_{rel,1}^D(u'_{yx})) && (g \text{ ist Homomorphismus}) \\
&= g_N(\pi_{rel,2}^D(u'_{xy})) && (C.294) \\
&= \pi_{rel,2}^H(g_{rel}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{rel,2}^H(u_{xy}) && (C.293) \\
(C.296) \qquad &= y && (C.292)
\end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
\pi_{rel,2}^H(g_{rel}(u'_{yx})) &= g_N(\pi_{rel,2}^D(u'_{yx})) && (g \text{ ist Homomorphismus}) \\
&= g_N(\pi_{rel,1}^D(u'_{xy})) && (C.295) \\
&= \pi_{rel,1}^H(g_{rel}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{rel,1}^H(u_{xy}) && (C.293) \\
(C.297) \qquad &= x && (C.291)
\end{aligned}$$

Aus den Gleichungen (C.296) und (C.297) folgt $u_{yx} ::= g_{rel}(u'_{yx})$.

(2) Gelte $u'_{xy} \in R_{rel}$ mit:

$$(C.298) \qquad u_{xy} = n_{rel}(u'_{xy})$$

Nach (C.298) erfüllt die Variablenbelegung

$$\begin{aligned}
x &\mapsto \pi_{rel,1}^R(u'_{xy}) \\
y &\mapsto \pi_{rel,2}^R(u'_{xy})
\end{aligned}$$

die Prämisse des Axioms (MP.7) in R . Weil R ein MP -System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $u'_{yx} \in R_{rel}$ mit:

$$(C.299) \qquad \pi_{rel,1}^R(u'_{yx}) = \pi_{rel,2}^R(u'_{xy})$$

$$(C.300) \qquad \pi_{rel,2}^R(u'_{yx}) = \pi_{rel,1}^R(u'_{xy})$$

Daraus folgt zum einen:

$$\begin{aligned}
\pi_{rel,1}^H(n_{rel}(u'_{yx})) &= n_N(\pi_{rel,1}^R(u'_{yx})) && (n \text{ ist Homomorphismus}) \\
&= n_N(\pi_{rel,2}^R(u'_{xy})) && (C.299) \\
&= \pi_{rel,2}^H(n_{rel}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{rel,2}^H(u_{xy}) && (C.298) \\
(C.301) \qquad &= y && (C.292)
\end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
\pi_{rel,2}^H(n_{rel}(u'_{yx})) &= n_N(\pi_{rel,2}^R(u'_{yx})) && (n \text{ ist Homomorphismus}) \\
&= n_N(\pi_{rel,1}^R(u'_{xy})) && \text{(C.300)} \\
&= \pi_{rel,1}^H(n_{rel}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{rel,1}^H(u_{xy}) && \text{(C.298)} \\
\text{(C.302)} \quad &= x && \text{(C.291)}
\end{aligned}$$

Aus den Gleichungen (C.301) und (C.302) folgt $u_{yx} ::= n_{rel}(u'_{yx})$. \square

C.2.4 Beweis von Lemma 10.28

Seien $x, y \in H_N$ gegeben mit $(x, y) \in \text{under}^H$. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz eines Elements $u_{xy} \in H_{\text{under}}$ mit:

$$\text{(C.303)} \quad \pi_{\text{under},1}^H(u_{xy}) = x$$

$$\text{(C.304)} \quad \pi_{\text{under},2}^H(u_{xy}) = y$$

Zu zeigen ist, dass $(x, y) \in \text{rel}^H$ gilt. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz eines Elements $v_{xy} \in H_{\text{rel}}$, welches die Gleichungen $\pi_{rel,1}^H(v_{xy}) = x$ und $\pi_{rel,2}^H(v_{xy}) = y$ erfüllt.

Nach Lemma B.26 existiert ein Urbild u'_{xy} von u_{xy} unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $u'_{xy} \in D_{\text{under}}$ mit:

$$\text{(C.305)} \quad u_{xy} = g_{\text{under}}(u'_{xy})$$

Nach (C.305) erfüllt die Variablenbelegung

$$x \mapsto \pi_{\text{under},1}^D(u'_{xy})$$

$$y \mapsto \pi_{\text{under},2}^D(u'_{xy})$$

die Prämisse des Axioms (MP.9) in D . Weil D ein MP-System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $v'_{xy} \in D_{\text{rel}}$ mit:

$$\text{(C.306)} \quad \pi_{rel,1}^D(v'_{xy}) = \pi_{\text{under},1}^D(u'_{xy})$$

$$\text{(C.307)} \quad \pi_{rel,2}^D(v'_{xy}) = \pi_{\text{under},2}^D(u'_{xy})$$

Daraus folgt zum einen:

$$\begin{aligned}
\pi_{rel,1}^H(g_{rel}(v'_{xy})) &= g_N(\pi_{rel,1}^D(v'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= g_N(\pi_{under,1}^D(u'_{xy})) && \text{(C.306)} \\
&= \pi_{under,1}^H(g_{under}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{under,1}^H(u_{xy}) && \text{(C.305)} \\
\text{(C.308)} \quad &= x && \text{(C.303)}
\end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
\pi_{rel,2}^H(g_{rel}(v'_{xy})) &= g_N(\pi_{rel,2}^D(v'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= g_N(\pi_{under,2}^D(u'_{xy})) && \text{(C.307)} \\
&= \pi_{under,2}^H(g_{under}(u'_{xy})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{under,2}^H(u_{xy}) && \text{(C.305)} \\
\text{(C.309)} \quad &= y && \text{(C.304)}
\end{aligned}$$

Aus den Gleichungen (C.308) und (C.309) folgt $v_{xy} ::= g_{rel}(v'_{xy})$.

(2) Gelte $u'_{xy} \in R_{under}$ mit:

$$\text{(C.310)} \quad u_{xy} = n_{under}(u'_{xy})$$

Nach (C.310) erfüllt die Variablenbelegung

$$\begin{aligned}
x &\mapsto \pi_{under,1}^R(u'_{xy}) \\
y &\mapsto \pi_{under,2}^R(u'_{xy})
\end{aligned}$$

die Prämisse des Axioms (MP.9) in R . Weil R ein MP -System ist, folgt die Wahrheit der Konklusion und somit die Existenz eines Elements $v'_{xy} \in R_{rel}$ mit:

$$\text{(C.311)} \quad \pi_{rel,1}^R(v'_{xy}) = \pi_{under,1}^R(u'_{xy})$$

$$\text{(C.312)} \quad \pi_{rel,2}^R(v'_{xy}) = \pi_{under,2}^R(u'_{xy})$$

Daraus folgt zum einen:

$$\begin{aligned}
\pi_{rel,1}^H(n_{rel}(v'_{xy})) &= n_N(\pi_{rel,1}^R(v'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= n_N(\pi_{under,1}^R(u'_{xy})) && \text{(C.311)} \\
&= \pi_{under,1}^H(n_{under}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{under,1}^H(u_{xy}) && \text{(C.310)} \\
\text{(C.313)} \quad &= x && \text{(C.303)}
\end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
\pi_{rel,2}^H(n_{rel}(v'_{xy})) &= n_N(\pi_{rel,2}^R(v'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= n_N(\pi_{under,2}^R(u'_{xy})) && \text{(C.312)} \\
&= \pi_{under,2}^H(n_{under}(u'_{xy})) && (n \text{ ist Homomorphismus}) \\
&= \pi_{under,2}^H(u_{xy}) && \text{(C.310)} \\
\text{(C.314)} \quad &= y && \text{(C.304)}
\end{aligned}$$

Aus den Gleichungen (C.313) und (C.314) folgt $v_{xy} ::= n_{rel}(v'_{xy})$. □

C.2.5 Beweis von Satz 10.30

Es reicht aus zu zeigen, dass die Prädikat-Axiome in H gelten, weil das Schema S nach Voraussetzung ein MP -System ist und somit diese Axiome in S gültig sind. Seien folglich $x, y \in H_p$ für ein beliebiges Prädikat $p \in P_w$, $w = s_1 s_2 \dots s_n \in S^*$, $n \geq 0$ gegeben, so dass

$$\text{(C.315)} \quad \pi_{p,w}^H(x) = \pi_{p,w}^H(y)$$

gilt. Zu zeigen ist die Wahrheit der Gleichung $x = y$.

Nach Lemma B.26 existiert ein Urbild x' von x und ein Urbild y' von y unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $x', y' \in D_p$ mit:

$$\text{(C.316)} \quad x = g_p(x')$$

$$\text{(C.317)} \quad y = g_p(y')$$

Daraus folgt:

$$\begin{aligned}
g_w(\pi_{p,w}^D(x')) &= \pi_{p,w}^H(g_p(x')) && (g \text{ ist Homomorphismus}) \\
&= \pi_{p,w}^H(x) && \text{(C.316)} \\
&= \pi_{p,w}^H(y) && \text{(C.315)} \\
&= \pi_{p,w}^H(g_p(y')) && \text{(C.317)} \\
&= g_w(\pi_{p,w}^D(y')) && (g \text{ ist Homomorphismus}) \\
\text{(C.318)} \quad &\Rightarrow \pi_{p,w}^D(x') = \pi_{p,w}^D(y') && (g \text{ ist injektiv nach Lemma B.28})
\end{aligned}$$

Nach (C.318) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto y' \end{aligned}$$

die Prämisse des Axioms (A.20) in $D \xrightarrow{\text{type}_D} S$. Da $D \xrightarrow{\text{type}_D} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$(C.319) \quad x' = y'$$

Daraus folgt:

$$x = g_p(x') \quad (C.316)$$

$$= g_p(y') \quad (C.319)$$

$$= y \quad (C.317)$$

(2) Gelte $x', y' \in R_p$ mit:

$$(C.320) \quad x = n_p(x')$$

$$(C.321) \quad y = n_p(y')$$

Nun gibt es zwei Fälle zu unterscheiden:

(a) Es gelte $\pi_{p,w}^R(x') = \pi_{p,w}^R(y')$. Dann erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto y' \end{aligned}$$

die Prämisse des Axioms (A.20) in $R \xrightarrow{\text{type}_R} S$. Da $R \xrightarrow{\text{type}_R} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$(C.322) \quad x' = y'$$

Daraus folgt:

$$x = n_p(x') \quad (C.320)$$

$$= n_p(y') \quad (C.322)$$

$$= y \quad (C.321)$$

(b) Es gelte:

$$(C.323) \quad \pi_{p,w}^R(x') \neq \pi_{p,w}^R(y')$$

Dann gilt zunächst:

$$\begin{aligned} n_w(\pi_{p,w}^R(x')) &= \pi_{p,w}^H(n_p(x')) && (n \text{ ist Homomorphismus}) \\ &= \pi_{p,w}^H(x) && (C.320) \end{aligned}$$

$$= \pi_{p,w}^H(y) \quad (C.315)$$

$$= \pi_{p,w}^H(n_p(y')) \quad (C.321)$$

$$(C.324) \quad = n_w(\pi_{p,w}^R(y')) \quad (n \text{ ist Homomorphismus})$$

Aus (C.323) und (C.324) folgt nach Lemma B.27, dass $\pi_{p,w}^R(x')$ und $\pi_{p,w}^R(y')$ Urbilder unter r besitzen. Somit existieren $\hat{x}, \hat{y} \in K_w$ mit:

$$(C.325) \quad \pi_{p,w}^R(x') = r_w(\hat{x})$$

$$(C.326) \quad \pi_{p,w}^R(y') = r_w(\hat{y})$$

Weil r nach Voraussetzung strikt voll ist, folgt aus (C.325) und (C.326) die Existenz zweier Elemente $x'', y'' \in K_p$ mit:

$$(C.327) \quad \pi_{p,w}^K(x'') = \hat{x}$$

$$(C.328) \quad \pi_{p,w}^K(y'') = \hat{y}$$

Daraus folgt:

$$\begin{aligned} \pi_{p,w}^R(r_p(x'')) &= r_w(\pi_{p,w}^K(x'')) && (r \text{ ist Homomorphismus}) \\ &= r_w(\hat{x}) && (C.327) \end{aligned}$$

$$(C.329) \quad = \pi_{p,w}^R(x') \quad (C.325)$$

Analog gilt:

$$\begin{aligned} \pi_{p,w}^R(r_p(y'')) &= r_w(\pi_{p,w}^K(y'')) && (r \text{ ist Homomorphismus}) \\ &= r_w(\hat{y}) && (C.328) \end{aligned}$$

$$(C.330) \quad = \pi_{p,w}^R(y') \quad (C.326)$$

Nach (C.329) und (C.330) erfüllen die Variablenbelegungen

$$\begin{aligned} x &\mapsto r_p(x'') \\ y &\mapsto x' \end{aligned}$$

und

$$\begin{aligned} x &\mapsto r_p(y'') \\ y &\mapsto y' \end{aligned}$$

jeweils die Prämisse des Axioms (A.20) in $R \xrightarrow{\text{type}_R} S$. Da $R \xrightarrow{\text{type}_R} S$ ein M -System ist, folgt die Wahrheit der Konklusionen:

$$(C.331) \quad r_p(x'') = x'$$

$$(C.332) \quad r_p(y'') = y'$$

Ferner gilt:

$$\begin{aligned} g_w(k_w(\hat{x})) &= n_w(r_w(\hat{x})) && \text{(Pushouts kommutieren)} \\ &= n_w(\pi_{p,w}^R(x')) && (C.325) \\ &= n_w(\pi_{p,w}^R(y')) && (C.324) \\ &= n_w(r_w(\hat{y})) && (C.326) \\ &= g_w(k_w(\hat{y})) && \text{(Pushouts kommutieren)} \\ (C.333) \quad &\Rightarrow k_w(\hat{x}) = k_w(\hat{y}) && (g \text{ ist injektiv nach} \\ &&& \text{Lemma B.28}) \end{aligned}$$

Daraus folgt:

$$\begin{aligned} \pi_{p,w}^D(k_p(x'')) &= k_w(\pi_{p,w}^K(x'')) && (k \text{ ist Homomorphismus)} \\ &= k_w(\hat{x}) && (C.327) \\ &= k_w(\hat{y}) && (C.333) \\ &= k_w(\pi_{p,w}^K(y'')) && (C.328) \\ (C.334) \quad &= \pi_{p,w}^D(k_p(y'')) && (k \text{ ist Homomorphismus}) \end{aligned}$$

Nach (C.334) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto k_p(x'') \\ y &\mapsto k_p(y'') \end{aligned}$$

die Prämisse des Axioms (A.20) in $D \xrightarrow{\text{type}_D} S$. Da $D \xrightarrow{\text{type}_D} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$(C.335) \quad k_p(x'') = k_p(y'')$$

Daraus folgt:

$$x = n_p(x') \quad (C.320)$$

$$= n_p(r_p(x'')) \quad (C.331)$$

$$= g_p(k_p(x'')) \quad (\text{Pushouts kommutieren})$$

$$= g_p(k_p(y'')) \quad (C.335)$$

$$= n_p(r_p(y'')) \quad (\text{Pushouts kommutieren})$$

$$= n_p(y') \quad (C.332)$$

$$= y \quad (C.321)$$

(3) Gelte $x' \in D_p$ und $y' \in R_p$ mit:

$$(C.336) \quad x = g_p(x')$$

$$(C.337) \quad y = n_p(y')$$

Daraus folgt:

$$g_w(\pi_{p,w}^D(x')) = \pi_{p,w}^H(g_p(x')) \quad (g \text{ ist Homomorphismus})$$

$$= \pi_{p,w}^H(x) \quad (C.336)$$

$$= \pi_{p,w}^H(y) \quad (C.315)$$

$$= \pi_{p,w}^H(n_p(y')) \quad (C.337)$$

$$(C.338) \quad = n_w(\pi_{p,w}^R(y')) \quad (n \text{ ist Homomorphismus})$$

Aus (C.338) und Lemma B.29 folgt die Existenz eines eindeutigen Elements $z \in K_w$ mit:

$$(C.339) \quad \pi_{p,w}^D(x') = k_w(z)$$

$$(C.340) \quad \pi_{p,w}^R(y') = r_w(z)$$

Weil r nach Voraussetzung strikt voll ist, folgt aus (C.340) die Existenz eines Elements $y'' \in K_p$ mit:

$$(C.341) \quad \pi_{p,w}^K(y'') = z$$

Daraus folgt:

$$\begin{aligned}
 \pi_{p,w}^R(r_p(y'')) &= r_w(\pi_{p,w}^K(y'')) && (r \text{ ist Homomorphismus}) \\
 &= r_w(z) && \text{(C.341)} \\
 \text{(C.342)} \quad &= \pi_{p,w}^R(y') && \text{(C.340)}
 \end{aligned}$$

Nach (C.342) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto r_p(y'') \\
 y &\mapsto y'
 \end{aligned}$$

die Prämisse des Axioms (A.20) in $R \xrightarrow{\text{type}_R} S$. Da $R \xrightarrow{\text{type}_R} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$\text{(C.343)} \quad r_p(y'') = y'$$

Ferner gilt:

$$\begin{aligned}
 g_w(\pi_{p,w}^D(x')) &= g_w(k_w(z)) && \text{(C.339)} \\
 &= n_w(r_w(z)) && \text{(Pushouts kommutieren)} \\
 &= n_w(r_w(\pi_{p,w}^K(y''))) && \text{(C.341)} \\
 &= g_w(k_w(\pi_{p,w}^K(y''))) && \text{(Pushouts kommutieren)} \\
 &= g_w(\pi_{p,w}^D(k_p(y''))) && (k \text{ ist Homomorphismus}) \\
 \text{(C.344)} \quad &\Rightarrow \pi_{p,w}^D(x') = \pi_{p,w}^D(k_p(y'')) && (g \text{ ist injektiv nach Lemma B.28})
 \end{aligned}$$

Nach (C.344) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto x' \\
 y &\mapsto k_p(y'')
 \end{aligned}$$

die Prämisse des Axioms (A.20) in $D \xrightarrow{\text{type}_D} S$. Da $D \xrightarrow{\text{type}_D} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$\text{(C.345)} \quad x' = k_p(y'')$$

Daraus folgt:

$$x = g_p(x') \quad (\text{C.336})$$

$$= g_p(k_p(y'')) \quad (\text{C.345})$$

$$= n_p(r_p(y'')) \quad (\text{Pushouts kommutieren})$$

$$= n_p(y') \quad (\text{C.343})$$

$$= y \quad (\text{C.337})$$

(4) Gelte $x' \in R_p$ und $y' \in D_p$ mit:

$$(\text{C.346}) \quad x = n_p(x')$$

$$(\text{C.347}) \quad y = g_p(y')$$

Dann folgt $x = y$ analog zum letzten Punkt durch Vertauschen von x und y sowie x' und y' . \square

C.2.6 Beweis von Satz 10.31

Seien $x, y \in H_N$ gegeben mit

$$(\text{C.348}) \quad \text{type}_{H,N}(x) = \text{type}_{H,N}(y)$$

sowie $(x, y) \in \text{rel}^H$. Nach Konstruktion A.94 ist dies äquivalent zu der Existenz eines Elements $u_{xy} \in H_{\text{rel}}$ mit:

$$(\text{C.349}) \quad \pi_{\text{rel},1}^H(u_{xy}) = x$$

$$(\text{C.350}) \quad \pi_{\text{rel},2}^H(u_{xy}) = y$$

Zu zeigen ist die Wahrheit der Gleichung $x = y$.

Nach Lemma B.26 existiert ein Urbild u'_{xy} von u_{xy} unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $u'_{xy} \in D_{\text{rel}}$ mit:

$$(\text{C.351}) \quad u_{xy} = g_{\text{rel}}(u'_{xy})$$

Daraus folgt:

$$\begin{aligned} \text{type}_{D,N}(\pi_{\text{rel},1}^D(u'_{xy})) &= \text{type}_{H,N}(g_N(\pi_{\text{rel},1}^D(u'_{xy}))) && (\text{type}_D = \text{type}_H \circ g) \\ &= \text{type}_{H,N}(\pi_{\text{rel},1}^H(g_{\text{rel}}(u'_{xy}))) && (g \text{ ist Homomorphismus}) \\ &= \text{type}_{H,N}(\pi_{\text{rel},1}^H(u_{xy})) && (\text{C.351}) \end{aligned}$$

$$= \text{type}_{H,N}(x) \quad (\text{C.349})$$

$$= \text{type}_{H,N}(y) \quad (\text{C.348})$$

$$= \text{type}_{H,N}(\pi_{rel,2}^H(u_{xy})) \quad (\text{C.350})$$

$$= \text{type}_{H,N}(\pi_{rel,2}^H(g_{rel}(u'_{xy}))) \quad (\text{C.351})$$

$$= \text{type}_{H,N}(g_N(\pi_{rel,2}^D(u'_{xy}))) \quad (g \text{ ist Homomorphismus})$$

$$(C.352) \quad = \text{type}_{D,N}(\pi_{rel,2}^D(u'_{xy})) \quad (\text{type}_D = \text{type}_H \circ g)$$

Nach (C.352) erfüllt die Variablenbelegung

$$x \mapsto \pi_{rel,1}^D(u'_{xy})$$

$$y \mapsto \pi_{rel,2}^D(u'_{xy})$$

die Prämisse des Axioms (M.28) in $D \xrightarrow{\text{type}_D} S$. Da $D \xrightarrow{\text{type}_D} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$(C.353) \quad \pi_{rel,1}^D(u'_{xy}) = \pi_{rel,2}^D(u'_{xy})$$

Daraus folgt:

$$x = \pi_{rel,1}^H(u_{xy}) \quad (\text{C.349})$$

$$= \pi_{rel,1}^H(g_{rel}(u'_{xy})) \quad (\text{C.351})$$

$$= g_N(\pi_{rel,1}^D(u'_{xy})) \quad (g \text{ ist Homomorphismus})$$

$$= g_N(\pi_{rel,2}^D(u'_{xy})) \quad (\text{C.353})$$

$$= \pi_{rel,2}^H(g_{rel}(u'_{xy})) \quad (g \text{ ist Homomorphismus})$$

$$= \pi_{rel,2}^H(u_{xy}) \quad (\text{C.351})$$

$$= y \quad (\text{C.350})$$

(2) Gelte $u'_{xy} \in R_{rel}$ mit:

$$(C.354) \quad u_{xy} = n_{rel}(u'_{xy})$$

Daraus folgt:

$$\text{type}_{R,N}(\pi_{rel,1}^R(u'_{xy})) = \text{type}_{H,N}(n_N(\pi_{rel,1}^R(u'_{xy}))) \quad (\text{type}_R = \text{type}_H \circ n)$$

$$= \text{type}_{H,N}(\pi_{rel,1}^H(n_{rel}(u'_{xy}))) \quad (n \text{ ist Homomorphismus})$$

$$= \text{type}_{H,N}(\pi_{rel,1}^H(u_{xy})) \quad (\text{C.354})$$

$$= \text{type}_{H,N}(x) \quad (\text{C.349})$$

$$= \text{type}_{H,N}(y) \quad (\text{C.348})$$

$$= \text{type}_{H,N}(\pi_{rel,2}^H(u_{xy})) \quad (\text{C.350})$$

$$= \text{type}_{H,N}(\pi_{rel,2}^H(n_{rel}(u'_{xy}))) \quad (\text{C.354})$$

$$= \text{type}_{H,N}(n_N(\pi_{rel,2}^R(u'_{xy}))) \quad (n \text{ ist Homomorphismus})$$

$$(C.355) \quad = \text{type}_{R,N}(\pi_{rel,2}^R(u'_{xy})) \quad (\text{type}_R = \text{type}_H \circ n)$$

Nach (C.355) erfüllt die Variablenbelegung

$$x \mapsto \pi_{rel,1}^R(u'_{xy})$$

$$y \mapsto \pi_{rel,2}^R(u'_{xy})$$

die Prämisse des Axioms (M.28) in $R \xrightarrow{\text{type}_R} S$. Da $R \xrightarrow{\text{type}_R} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$(C.356) \quad \pi_{rel,1}^R(u'_{xy}) = \pi_{rel,2}^R(u'_{xy})$$

Daraus folgt:

$$x = \pi_{rel,1}^H(u_{xy}) \quad (\text{C.349})$$

$$= \pi_{rel,1}^H(n_{rel}(u'_{xy})) \quad (\text{C.354})$$

$$= n_N(\pi_{rel,1}^R(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= n_N(\pi_{rel,2}^R(u'_{xy})) \quad (\text{C.356})$$

$$= \pi_{rel,2}^H(n_{rel}(u'_{xy})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{rel,2}^H(u_{xy}) \quad (\text{C.354})$$

$$= y \quad (\text{C.350})$$

□

C.2.7 Beweis von Satz 10.32

Seien $e_1, e_2 \in H_E$ gegeben mit:

$$(C.357) \quad \text{type}_{H,E}(e_1) = \text{type}_{H,E}(e_2)$$

$$(C.358) \quad \text{source}^H(e_1) = \text{source}^H(e_2)$$

Zu zeigen ist die Wahrheit der Gleichung $e_1 = e_2$.

Nach Lemma B.26 existiert ein Urbild e'_1 von e_1 und ein Urbild e'_2 von e_2 unter mindestens einem der beiden Homomorphismen g und n :

(1) Gelte $e'_1, e'_2 \in D_E$ mit:

$$(C.359) \quad e_1 = g_E(e'_1)$$

$$(C.360) \quad e_2 = g_E(e'_2)$$

Daraus folgt:

$$\begin{aligned}
 g_N(\text{source}^D(e'_1)) &= \text{source}^H(g_E(e'_1)) && (g \text{ ist Homomorphismus}) \\
 &= \text{source}^H(e_1) && (C.359) \\
 &= \text{source}^H(e_2) && (C.358) \\
 &= \text{source}^H(g_E(e'_2)) && (C.360) \\
 &= g_N(\text{source}^D(e'_2)) && (g \text{ ist Homomorphismus}) \\
 (C.361) \quad &\Rightarrow \text{source}^D(e'_1) = \text{source}^D(e'_2) && (g \text{ ist injektiv nach Lemma B.28})
 \end{aligned}$$

Ferner gilt:

$$\begin{aligned}
 \text{type}_{D,E}(e'_1) &= \text{type}_{H,E}(g_E(e'_1)) && (\text{type}_D = \text{type}_H \circ g) \\
 &= \text{type}_{H,E}(e_1) && (C.359) \\
 &= \text{type}_{H,E}(e_2) && (C.357) \\
 &= \text{type}_{H,E}(g_E(e'_2)) && (C.360) \\
 (C.362) \quad &= \text{type}_{D,E}(e'_2) && (\text{type}_D = \text{type}_H \circ g)
 \end{aligned}$$

Nach (C.361) und (C.362) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto e'_1 \\
 y &\mapsto e'_2
 \end{aligned}$$

die Prämisse des Axioms (M.29) in $D \xrightarrow{\text{type}_D} S$. Da $D \xrightarrow{\text{type}_D} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$(C.363) \quad e'_1 = e'_2$$

Daraus folgt:

$$e_1 = g_E(e'_1) \quad (C.359)$$

$$= g_E(e'_2) \quad (C.363)$$

$$= e_2 \quad (C.360)$$

(2) Gelte $e'_1, e'_2 \in D_E$ mit:

$$(C.364) \quad e_1 = n_E(e'_1)$$

$$(C.365) \quad e_2 = n_E(e'_2)$$

Daraus folgt:

$$\begin{aligned} type_{R,E}(e'_1) &= type_{H,E}(n_E(e'_1)) && (type_R = type_H \circ n) \\ &= type_{H,E}(e_1) && (C.364) \\ &= type_{H,E}(e_2) && (C.357) \\ &= type_{H,E}(n_E(e'_2)) && (C.365) \\ (C.366) \quad &= type_{R,E}(e'_2) && (type_R = type_H \circ n) \end{aligned}$$

Nun sind zwei Fälle zu unterscheiden:

(a) Es gelte:

$$(C.367) \quad source^R(e'_1) = source^R(e'_2)$$

Dann erfüllt nach (C.366) und (C.367) die Variablenbelegung

$$\begin{aligned} x &\mapsto e'_1 \\ y &\mapsto e'_2 \end{aligned}$$

die Prämisse des Axioms (M.29) in $R \xrightarrow{type_R} S$. Da $R \xrightarrow{type_R} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$(C.368) \quad e'_1 = e'_2$$

Daraus folgt:

$$e_1 = n_E(e'_1) \quad (C.364)$$

$$= n_E(e'_2) \quad (C.368)$$

$$= e_2 \quad (C.365)$$

(b) Es gelte:

$$(C.369) \quad source^R(e'_1) \neq source^R(e'_2)$$

Weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist, existieren zwei Knoten $k_1, k_2 \in K_N$ und zwei Kanten $\hat{e}_1, \hat{e}_2 \in L_E$ mit:

$$(C.370) \quad r_N(k_1) = source^R(e'_1)$$

$$(C.371) \quad l_N(k_1) = source^L(\hat{e}_1)$$

$$(C.372) \quad type_{L,E}(\hat{e}_1) = type_{R,E}(e'_1)$$

$$(C.373) \quad r_N(k_2) = source^R(e'_2)$$

$$(C.374) \quad l_N(k_2) = source^L(\hat{e}_2)$$

$$(C.375) \quad type_{L,E}(\hat{e}_2) = type_{R,E}(e'_2)$$

Nun gilt:

$$\begin{aligned}
 g_N(k_N(k_1)) &= n_N(r_N(k_1)) && \text{(Pushouts kommutieren)} \\
 &= n_N(source^R(e'_1)) && (C.370) \\
 &= source^H(n_E(e'_1)) && (n \text{ ist Homomorphismus}) \\
 &= source^H(e_1) && (C.364) \\
 &= source^H(e_2) && (C.358) \\
 &= source^H(n_E(e'_2)) && (C.365) \\
 &= n_N(source^R(e'_2)) && (n \text{ ist Homomorphismus}) \\
 &= n_N(r_N(k_2)) && (C.373) \\
 &= g_N(k_N(k_2)) && \text{(Pushouts kommutieren)} \\
 (C.376) \quad &\Rightarrow k_N(k_1) = k_N(k_2) && (g \text{ ist injektiv nach Lemma B.28})
 \end{aligned}$$

Daraus folgt:

$$\begin{aligned}
 source^G(m_E(\hat{e}_1)) &= m_N(source^L(\hat{e}_1)) && (m \text{ ist Homomorphismus}) \\
 &= m_N(l_N(k_1)) && (C.371) \\
 &= f_N(k_N(k_1)) && \text{(Pushouts kommutieren)} \\
 &= f_N(k_N(k_2)) && (C.376) \\
 &= m_N(l_N(k_2)) && \text{(Pushouts kommutieren)} \\
 &= m_N(source^L(\hat{e}_2)) && (C.374) \\
 (C.377) \quad &= source^G(m_E(\hat{e}_2)) && (m \text{ ist Homomorphismus})
 \end{aligned}$$

Ferner gilt:

$$\begin{aligned}
& \text{type}_{G,E}(m_E(\hat{e}_1)) = \text{type}_{L,E}(\hat{e}_1) && (\text{type}_L = \text{type}_G \circ m) \\
& = \text{type}_{R,E}(e'_1) && \text{(C.372)} \\
& = \text{type}_{R,E}(e'_2) && \text{(C.366)} \\
& = \text{type}_{L,E}(\hat{e}_2) && \text{(C.375)} \\
\text{(C.378)} \quad & = \text{type}_{G,E}(m_E(\hat{e}_2)) && (\text{type}_L = \text{type}_G \circ m)
\end{aligned}$$

Aus (C.377) und (C.378) folgt, dass die Variablenbelegung

$$\begin{aligned}
x &\mapsto m_E(\hat{e}_1) \\
y &\mapsto m_E(\hat{e}_2)
\end{aligned}$$

die Prämisse des Axioms (M.29) in $G \xrightarrow{\text{type}_G} S$ erfüllt. Da $G \xrightarrow{\text{type}_G} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$\text{(C.379)} \quad m_E(\hat{e}_1) = m_E(\hat{e}_2)$$

Ferner folgt aus (C.369):

$$\begin{aligned}
& \text{source}^R(e'_1) \neq \text{source}^R(e'_2) \Rightarrow r_N(k_1) \neq r_N(k_2) && \text{(C.370) und (C.373)} \\
& \Rightarrow l_N(k_1) \neq l_N(k_2) && (l \text{ ist injektiv}) \\
& \Rightarrow \text{source}^L(\hat{e}_1) \neq \text{source}^L(\hat{e}_2) && \text{(C.371) und (C.374)} \\
\text{(C.380)} \quad & \Rightarrow \hat{e}_1 \neq \hat{e}_2 && (\text{source}^L \text{ ist Abbildung})
\end{aligned}$$

Aus (C.379), (C.380) und Lemma B.27 folgt die Existenz zweier Elemente $\hat{e}'_1, \hat{e}'_2 \in K_E$ mit:

$$\text{(C.381)} \quad l_E(\hat{e}'_1) = \hat{e}_1$$

$$\text{(C.382)} \quad l_E(\hat{e}'_2) = \hat{e}_2$$

Daraus folgt:

$$\begin{aligned}
& f_E(k_E(\hat{e}'_1)) = m_E(l_E(\hat{e}'_1)) && \text{(Pushouts kommutieren)} \\
& = m_E(\hat{e}_1) && \text{(C.381)} \\
& = m_E(\hat{e}_2) && \text{(C.379)} \\
& = m_E(l_E(\hat{e}'_2)) && \text{(C.382)} \\
& = f_E(k_E(\hat{e}'_2)) && \text{(Pushouts kommutieren)} \\
\text{(C.383)} \quad & \Rightarrow k_E(\hat{e}'_1) = k_E(\hat{e}'_2) && (f \text{ ist injektiv nach Lemma B.28})
\end{aligned}$$

Nun gilt:

$$\begin{aligned}
 l_N(k_{1,2}) &= source^L(\hat{e}_{1,2}) && \text{(C.371) und (C.374)} \\
 &= source^L(l_E(\hat{e}'_{1,2})) && \text{(C.381) und (C.382)} \\
 &= l_N(source^K(\hat{e}'_{1,2})) && (l \text{ ist Homomorphismus}) \\
 \text{(C.384)} \quad &\Rightarrow k_{1,2} = source^K(\hat{e}'_{1,2}) && (l \text{ ist injektiv})
 \end{aligned}$$

Ferner gilt:

$$\begin{aligned}
 source^R(e'_{1,2}) &= r_N(k_{1,2}) && \text{(C.370) und (C.373)} \\
 &= r_N(source^K(\hat{e}'_{1,2})) && \text{(C.384)} \\
 \text{(C.385)} \quad &= source^R(r_E(\hat{e}'_{1,2})) && (r \text{ ist Homomorphismus})
 \end{aligned}$$

Schließlich gilt:

$$\begin{aligned}
 type_{R,E}(e'_{1,2}) &= type_{L,E}(\hat{e}_{1,2}) && \text{(C.372) und (C.375)} \\
 &= type_{L,E}(l_E(\hat{e}'_{1,2})) && \text{(C.381) und (C.382)} \\
 \text{(C.386)} \quad &= type_{R,E}(r_E(\hat{e}'_{1,2})) && (type_L \circ l = type_R \circ r)
 \end{aligned}$$

Seien

$$\begin{aligned}
 x &\mapsto e'_i \\
 y &\mapsto r_E(\hat{e}'_i)
 \end{aligned}$$

mit $i \in \{1, 2\}$ zwei Variablenbelegungen in R . Diese Variablenbelegungen erfüllen nach (C.385) und (C.386) jeweils die Prämisse des Axioms (M.29) in $R \xrightarrow{type_R} S$. Weil $R \xrightarrow{type_R} S$ ein M -System ist, folgt die Wahrheit der Konklusionen:

$$\text{(C.387)} \quad e'_1 = r_E(\hat{e}'_1)$$

$$\text{(C.388)} \quad e'_2 = r_E(\hat{e}'_2)$$

Daraus folgt:

$$\begin{aligned}
 e_1 &= n_E(e'_1) && \text{(Definition von } e'_1) \\
 &= n_E(r_E(\hat{e}'_1)) && \text{(C.387)} \\
 &= g_E(k_E(\hat{e}'_1)) && \text{(Pushouts kommutieren)} \\
 &= g_E(k_E(\hat{e}'_2)) && \text{(C.383)} \\
 &= n_E(r_E(\hat{e}'_2)) && \text{(Pushouts kommutieren)}
 \end{aligned}$$

$$\begin{aligned}
&= n_E(e'_2) && \text{(C.388)} \\
&= e_2 && \text{(Definition von } e'_1)
\end{aligned}$$

(3) Gelte $e'_1 \in R_E$ und $e'_2 \in D_E$ mit:

$$(C.389) \quad e_1 = n_E(e'_1)$$

$$(C.390) \quad e_2 = g_E(e'_2)$$

Daraus folgt:

$$\begin{aligned}
&type_{R,E}(e'_1) = type_{H,E}(n_E(e'_1)) && (type_R = type_H \circ n) \\
&= type_{H,E}(e_1) && \text{(C.389)} \\
&= type_{H,E}(e_2) && \text{(C.357)} \\
&= type_{H,E}(g_E(e'_2)) && \text{(C.390)} \\
(C.391) \quad &= type_{D,E}(e'_2) && (type_D = type_H \circ g)
\end{aligned}$$

Weil $L \xleftarrow{l} K \xrightarrow{r} R$ eine konsistente Aktion ist, existiert ein Knoten $k_1 \in K_N$ und eine Kante $\hat{e}_1 \in L_E$ mit:

$$(C.392) \quad r_N(k_1) = source^R(e'_1)$$

$$(C.393) \quad l_N(k_1) = source^L(\hat{e}_1)$$

$$(C.394) \quad type_{L,E}(\hat{e}_1) = type_{R,E}(e'_1)$$

Nun gilt:

$$\begin{aligned}
&g_N(k_N(k_1)) = n_N(r_N(k_1)) && \text{(Pushouts kommutieren)} \\
&= n_N(source^R(e'_1)) && \text{(C.392)} \\
&= source^H(n_E(e'_1)) && (n \text{ ist Homomorphismus)} \\
&= source^H(e_1) && \text{(C.389)} \\
&= source^H(e_2) && \text{(C.358)} \\
&= source^H(g_E(e'_2)) && \text{(C.390)} \\
&= g_N(source^D(e'_2)) && (g \text{ ist Homomorphismus)} \\
(C.395) \quad &\Rightarrow k_N(k_1) = source^D(e'_2) && (g \text{ ist injektiv nach Lemma B.28)}
\end{aligned}$$

Daraus folgt:

$$\begin{aligned}
\text{(C.396)} \quad \text{source}^G(m_E(\hat{e}_1)) &= m_N(\text{source}^L(\hat{e}_1)) && (m \text{ ist Homomorphismus}) \\
&= m_N(l_N(k_1)) && \text{(C.393)} \\
&= f_N(k_N(k_1)) && (\text{Pushouts kommutieren}) \\
&= f_N(\text{source}^D(e'_2)) && \text{(C.395)} \\
&= \text{source}^G(f_E(e'_2)) && (f \text{ ist Homomorphismus})
\end{aligned}$$

Ferner gilt:

$$\begin{aligned}
\text{(C.397)} \quad \text{type}_{G,E}(m_E(\hat{e}_1)) &= \text{type}_{L,E}(\hat{e}_1) && (\text{type}_L = \text{type}_G \circ m) \\
&= \text{type}_{R,E}(e'_1) && \text{(C.394)} \\
&= \text{type}_{D,E}(e'_2) && \text{(C.391)} \\
&= \text{type}_{G,E}(f_E(e'_2)) && (\text{type}_D = \text{type}_G \circ f)
\end{aligned}$$

Aus (C.396) und (C.397) folgt, dass die Variablenbelegung

$$\begin{aligned}
x &\mapsto m_E(\hat{e}_1) \\
y &\mapsto f_E(e'_2)
\end{aligned}$$

die Prämisse des Axioms (M.29) in $G \xrightarrow{\text{type}_G} S$ erfüllt. Da $G \xrightarrow{\text{type}_G} S$ ein M -System ist, folgt die Wahrheit der Konklusion $m_E(\hat{e}_1) = f_E(e'_2)$. Aus Lemma B.29 folgt, dass eine Kante $e \in K_E$ existiert mit:

$$\text{(C.398)} \quad l_E(e) = \hat{e}_1$$

$$\text{(C.399)} \quad k_E(e) = e'_2$$

Daraus folgt:

$$\begin{aligned}
\text{(C.400)} \quad l_N(\text{source}^K(e)) &= \text{source}^L(l_E(e)) && (l \text{ ist Homomorphismus}) \\
&= \text{source}^L(\hat{e}_1) && \text{(C.398)} \\
&= l_N(k_1) && \text{(C.393)} \\
&\Rightarrow \text{source}^K(e) = k_1 && (l \text{ ist injektiv})
\end{aligned}$$

Weiter gilt:

$$\begin{aligned}
\text{(C.401)} \quad \text{source}^R(r_E(e)) &= r_N(\text{source}^K(e)) && (r \text{ ist Homomorphismus}) \\
&= r_N(k_1) && \text{(C.400)} \\
&= \text{source}^R(e'_1) && \text{(C.392)}
\end{aligned}$$

Ferner stellt sich heraus:

$$\begin{aligned}
 \text{(C.402)} \quad \text{type}_{R,E}(r_E(e)) &= \text{type}_{L,E}(l_E(e)) && (\text{type}_R \circ r = \text{type}_L \circ l) \\
 &= \text{type}_{L,E}(\hat{e}_1) && \text{(C.398)} \\
 &= \text{type}_{R,E}(e'_1) && \text{(C.394)}
 \end{aligned}$$

Aus (C.401) und (C.402) folgt, dass die Variablenbelegung

$$\begin{aligned}
 x &\mapsto r_E(e) \\
 y &\mapsto e'_1
 \end{aligned}$$

die Prämisse des Axioms (M.29) in $R \xrightarrow{\text{type}_R} S$ erfüllt. Da $R \xrightarrow{\text{type}_R} S$ ein M -System ist, folgt die Wahrheit der Konklusion:

$$\text{(C.403)} \quad r_E(e) = e'_1$$

Daraus folgt:

$$\begin{aligned}
 e_1 &= n_E(e'_1) && \text{(Definition von } e'_1\text{)} \\
 &= n_E(r_E(e)) && \text{(C.403)} \\
 &= g_E(k_E(e)) && \text{(Pushouts kommutieren)} \\
 &= g_E(e'_2) && \text{(C.399)} \\
 &= e_2 && \text{(Definition von } e'_2\text{)}
 \end{aligned}$$

(4) Gelte $e'_1 \in D_E$ und $e'_2 \in R_E$ mit:

$$\text{(C.404)} \quad e_1 = g_E(e'_1)$$

$$\text{(C.405)} \quad e_2 = n_E(e'_2)$$

Dann führt dies analog zum letzten Punkt durch Vertauschen von e_1 und e_2 sowie von e'_1 und e'_2 zu der Wahrheit der Konklusion $e_1 = e_2$. \square

C.3 Beweise aus Kapitel 11

C.3.1 Beweis von Lemma 11.9

Die folgenden Einbettungen werden im nachfolgenden Beweis vorausgesetzt und nicht explizit erwähnt:

- Jedes $\mathbf{Sys}(S)$ -Objekt und jeder $\mathbf{Sys}(S)$ -Morphismus mit einem beliebigen Schema S ist ein $\mathbf{Alg}(MP)\downarrow S$ -Objekt bzw. ein $\mathbf{Alg}(MP)\downarrow S$ -Morphismus (Def. 9.10);
- jedes $\mathbf{Alg}(MP)\downarrow S$ -Objekt und jeder $\mathbf{Alg}(MP)\downarrow S$ -Morphismus mit einem beliebigen Schema S ist ein $\mathbf{Alg}(MP)^2$ -Objekt bzw. ein $\mathbf{Alg}(MP)^2$ -Morphismus (dies folgt aus der Konstruktion von Komma- und Pfeilkategorien).

Weiter werden die folgenden Funktoren benötigt:

- der Pullback-Funktor $\mathcal{P}^{fs}: \mathbf{Alg}(MP)\downarrow S^A \rightarrow \mathbf{Alg}(MP)\downarrow S^B$ (Konstruktion 11.4); und
- der Reflektor $\mathcal{F}: \mathbf{Alg}(MP)^2 \rightarrow \mathbf{Sys}$, der nach Satz 9.7, Satz 9.9 und Satz A.79 existiert.

Sei ein M -System $A \xrightarrow{type_A} S^A \in \mathbf{Ob}^{\mathbf{Sys}(S^A)}$ gegeben. Die Anwendung des Pullback-Funktors \mathcal{P}^{fs} auf dieses System liefert das System $B \xrightarrow{type_B} S^B$ in der Kategorie $\mathbf{Alg}(MP)\downarrow S^B$. Sei

$$(B' \xrightarrow{type_{B'}} S^{B'}, u_{type_{B'}}: (B \xrightarrow{type_B} S^B) \rightarrow (B' \xrightarrow{type_{B'}} S^{B'}))$$

die \mathbf{Sys} -Reflexion dieses Systems, wobei der Epimorphismus $u_{type_{B'}}$ aus den beiden $\mathbf{Alg}(MP)$ -Morphismen $u_B: B \rightarrow B'$ und $u_{S^B}: S^B \rightarrow S^{B'}$ besteht (Abb. C.1). Es wird nun gezeigt, dass $u_{type_{B'}}$ ein Isomorphismus ist, dass also gilt:

$$(C.406) \quad \mathcal{F}(B \xrightarrow{type_B} S^B) \cong B \xrightarrow{type_B} S^B$$

Aus (C.406) folgt dann sofort, dass $B \xrightarrow{type_B} S^B$ ein M -System ist, weil alle betrachteten Unterkategorien und somit auch $\mathbf{Sys}(S^B)$ unter Isomorphie abgeschlossen sind.

$$\begin{array}{ccccc}
 S^A & \xleftarrow{f_S} & S^B & \xrightarrow{u_{S^B}} & S^{B'} \\
 \uparrow type_A & & \uparrow type_B & & \uparrow type_{B'} \\
 A & \xleftarrow{f_I} & B & \xrightarrow{u_B} & B'
 \end{array}$$

P.B.

Abb. C.1: Pullback mit anschließender Faktorisierung nach den M -Axiomen in $\mathbf{Alg}(MP)$

Zuerst gilt, dass u_B ein $\mathbf{Alg}(MP)$ -Epimorphismus ist. Nach Konstruktion A.92 ist $u_{type_{B'}}$ und somit auch u_B surjektiv. Aus Lemma A.81 folgt schließlich, dass $u_{type_{B'}}$ Epimorphismus ist.

Weiter existiert nun, weil der Reflektor \mathcal{F} ein freier Funktor ist, ein $\mathbf{Alg}(MP)^2$ -Morphismus

$$m: (B' \xrightarrow{type_{B'}} S^{B'}) \rightarrow (A \xrightarrow{type_A} S^A) = (m_S: S^{B'} \rightarrow S^A, m_I: B' \rightarrow A)$$

mit:

$$(C.407) \quad m \circ u_{type_{B'}} = f$$

Ferner folgt nach Lemma 9.13 $S^B = S^{B'}$. Somit gilt:

$$(C.408) \quad u_{S^B} = id_{S^B}$$

Daraus folgt:

$$f_S = m_S \circ u_{S^B} \quad (C.407)$$

$$= m_S \circ id_{S^B} \quad (C.408)$$

$$(C.409) \quad = m_S \quad (\text{Identitat ist rechtsneutral})$$

Schlielich gilt:

$$type_{B'} \circ u_B = u_{S^B} \circ type_B \quad (\text{Pfeilkategorie-Morphismen kommutieren})$$

$$= id_{S^B} \circ type_B \quad (C.408)$$

$$(C.410) \quad = type_B \quad (\text{Identitat ist linksneutral})$$

Es ergibt sich somit die Situation in Abb. C.2.

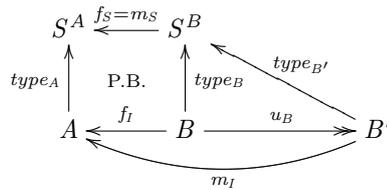


Abb. C.2: Pullback, Faktorisierung und universeller Morphismus

Nun gilt:

$$f_S \circ type_{B'} \circ u_B = f_S \circ type_B \quad (C.410)$$

$$= type_A \circ f_I \quad (\text{Pullbacks kommutieren})$$

$$= type_A \circ m_I \circ u_B \quad (C.407)$$

$$(C.411) \quad \Rightarrow f_S \circ type_{B'} = type_A \circ m_I \quad (u_B \text{ ist Epimorphismus})$$

Aus (C.411) folgt, dass $A \xleftarrow{m_I} B' \xrightarrow{type_{B'}} S^B$ Vergleichsobjekt zum Pullback $A \xleftarrow{f_I} B \xrightarrow{type_B} S^B$ in $\mathbf{Alg}(MP)$ ist. Somit existiert ein eindeutiger $\mathbf{Alg}(MP)$ -Morphismus $v: B' \rightarrow B$ mit (Abb. C.3):

$$(C.412) \quad f_I \circ v = m_I$$

$$(C.413) \quad type_B \circ v = type_{B'}$$

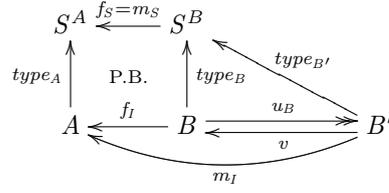


Abb. C.3: Mittler-Morphismus ins Pullback-Objekt

Daraus folgt:

$$\begin{aligned}
 f_I \circ v \circ u_B &= m_I \circ u_B & (C.412) \\
 &= f_I & (C.407) \\
 (C.414) \quad &= f_I \circ id_B & (\text{Identitat ist rechtsneutral})
 \end{aligned}$$

Weiter gilt:

$$\begin{aligned}
 type_B \circ v \circ u_B &= type_{B'} \circ u_B & (C.413) \\
 &= type_B & (C.410) \\
 (C.415) \quad &= type_B \circ id_B & (\text{Identitat ist rechtsneutral})
 \end{aligned}$$

Aus (C.414) und (C.415) folgt auf Grund der Eindeutigkeit des Mittler-Morphismus ins Pullback-Objekt:

$$(C.416) \quad v \circ u_B = id_B$$

Daraus folgt:

$$\begin{aligned}
 u_B \circ v \circ u_B &= u_B \circ id_B & (C.416) \\
 &= u_B & (\text{Identitat ist rechtsneutral}) \\
 &= id_{B'} \circ u_B & (\text{Identitat ist linksneutral}) \\
 (C.417) \quad &\Rightarrow u_B \circ v = id_{B'} & (u_B \text{ ist Epimorphismus})
 \end{aligned}$$

Aus (C.416) und (C.417) folgt, dass u_B Isomorphismus ist. Somit gilt (C.406). \square

C.3.2 Beweis von Lemma 11.17

Offensichtlich ist $\mathcal{P}^{l^t}(f)$ auf allen Relationen im Schema $S^\#$ vervollstandigt. Es reicht somit, die Auspragungen zu untersuchen (Abb. C.4). Sei ein Pradikat $p \in P_w$ mit $w \in S^*$ und ein

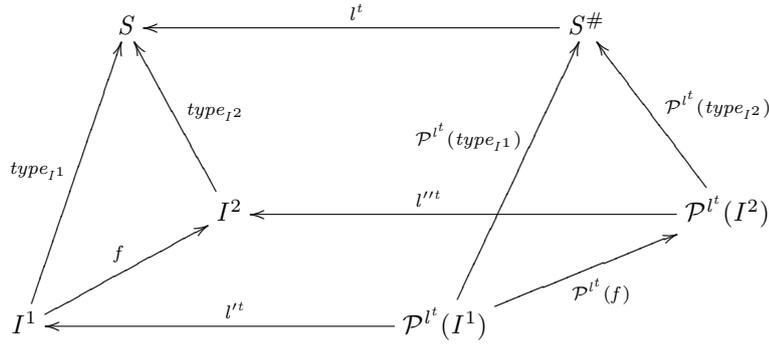


Abb. C.4: Anwendung des Pullback-Funktors

nichtleeres Sortenwort w' , das durch Streichung beliebiger Sorten aus w entsteht, gegeben. Seien weiter zwei Tupel $x \in \mathcal{P}^{l^t}(I^2)_w$ und $x' \in \mathcal{P}^{l^t}(I^1)_{w'}$ gegeben mit:

$$(C.418) \quad x \in p^{\mathcal{P}^{l^t}(I^2)}$$

$$(C.419) \quad \mathcal{P}^{l^t}(f)_{w'}(x') = \langle x \rangle_{w'}$$

Zu zeigen ist, dass ein Tupel $y \in \mathcal{P}^{l^t}(I^1)_w$ existiert mit:

$$(C.420) \quad \langle y \rangle_{w'} = x'$$

$$(C.421) \quad \mathcal{P}^{l^t}(f)_w(y) = x$$

$$(C.422) \quad y \in p^{\mathcal{P}^{l^t}(I^1)}$$

Aus (C.418) folgt, weil l''^t Homomorphismus ist:

$$(C.423) \quad l''^t_w(x) \in p^{I^2}$$

Weiter ist $I^1 \xleftarrow{l^t} \mathcal{P}^{l^t}(I^1) \xrightarrow{\mathcal{P}^{l^t}(f)} \mathcal{P}^{l^t}(I^2)$ Pullback von $I^1 \xrightarrow{f} I^2 \xleftarrow{l''^t} \mathcal{P}^{l^t}(I^2)$ nach Lemma 11.15. Somit gilt:

$$(C.424) \quad f \circ l^t = l''^t \circ \mathcal{P}^{l^t}(f)$$

Daraus ergibt sich:

$$f_{w'}(l^t_{w'}(x')) = l''^t_{w'}(\mathcal{P}^{l^t}(f)_{w'}(x')) \quad (C.424)$$

$$= l''^t_{w'}(\langle x \rangle_{w'}) \quad (C.419)$$

$$(C.425) \quad = \langle l''^t_w(x) \rangle_{w'} \quad (l''^t \text{ ist Homomorphismus})$$

Nach Voraussetzung ist f vervollständigend. Somit existiert nach (C.423) und (C.425) ein $x'' \in I_w^1$ mit:

$$(C.426) \quad \langle x'' \rangle_{w'} = l_{w'}^{t'}(x')$$

$$(C.427) \quad f_w(x'') = l_w^{t''}(x)$$

$$(C.428) \quad x'' \in p^{I^1}$$

Aus (C.418), (C.427) und (C.428) folgt schließlich nach Korollar A.87, dass ein $z \in \mathcal{P}^{l^t}(I^1)_w$ existiert mit:

$$(C.429) \quad z \in p^{\mathcal{P}^{l^t}(I^1)}$$

$$(C.430) \quad l_w^{t'}(z) = x''$$

$$(C.431) \quad \mathcal{P}^{l^t}(f)_w(z) = x$$

Aus (C.431) folgt:

$$(C.432) \quad \mathcal{P}^{l^t}(f)_{w'}(\langle z \rangle_{w'}) = \langle x \rangle_{w'}$$

Nach Lemma 10.13 und Lemma A.24 ist f injektiv. Nach Lemma 11.16 folgt, dass $\mathcal{P}^{l^t}(f)$ ebenfalls injektiv ist. Daraus folgt:

$$(C.433) \quad \begin{aligned} \mathcal{P}^{l^t}(f)_{w'}(x') &= \mathcal{P}^{l^t}(f)_{w'}(\langle z \rangle_{w'}) && (C.432) \text{ und } (C.419) \\ \Rightarrow \langle z \rangle_{w'} &= x' && (\mathcal{P}^{l^t}(f) \text{ ist injektiv}) \end{aligned}$$

Mit (C.433), (C.431) und (C.429) erfüllt $y := z$ offensichtlich (C.420), (C.421) und (C.422). \square

C.3.3 Beweis von Lemma 11.18

Offensichtlich zieht der Span $\mathcal{P}^{l^t}(I^1 \xrightarrow{\text{type}_{I^1}} S) \xleftarrow{\mathcal{P}^{l^t}(l)} \mathcal{P}^{l^t}(I^2 \xrightarrow{\text{type}_{I^2}} S) \xrightarrow{\mathcal{P}^{l^t}(r)} \mathcal{P}^{l^t}(I^3 \xrightarrow{\text{type}_{I^3}} S)$ im Schema $S^\#$ alle Kanten zurück. Es reicht somit, die Ausprägungen zu untersuchen (Abb. C.5). Sei eine Kante $e_r \in \mathcal{P}^{l^t}(I^3)_E$ gegeben. Zu zeigen ist, dass ein Knoten $k \in \mathcal{P}^{l^t}(I^2)_N$ und eine Kante $e_l \in \mathcal{P}^{l^t}(I^1)_E$ existieren mit:

$$(C.434) \quad \text{source}^{\mathcal{P}^{l^t}(I^1)}(e_l) = \mathcal{P}^{l^t}(l)_N(k)$$

$$(C.435) \quad \text{source}^{\mathcal{P}^{l^t}(I^3)}(e_r) = \mathcal{P}^{l^t}(r)_N(k)$$

$$(C.436) \quad \text{type}_{\mathcal{P}^{l^t}(I^1)_E}(e_l) = \text{type}_{\mathcal{P}^{l^t}(I^3)_E}(e_r)$$

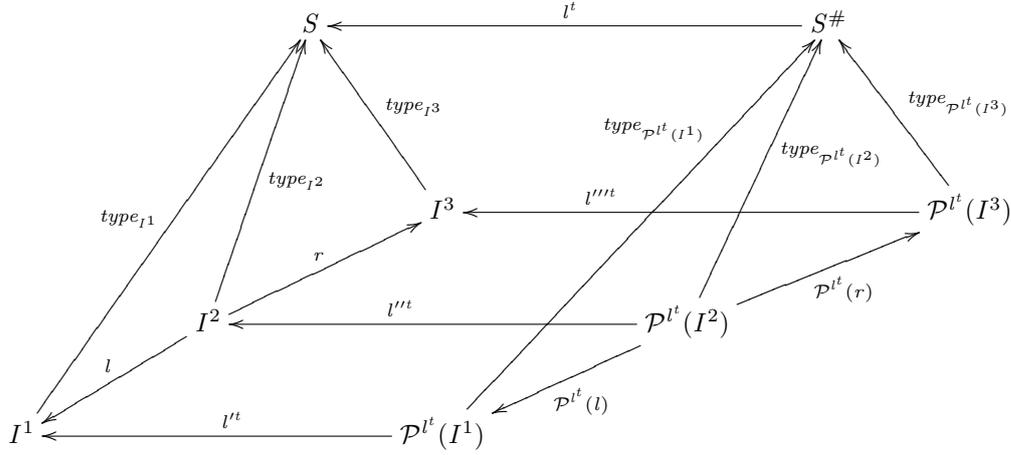


Abb. C.5: Anwendung des Pullback-Funktors auf Spans

Aus der Definition von e_r folgt $l_E^t(e_r) \in I_E^3$. Nach Voraussetzung zieht der Span $(I^1 \xrightarrow{type_{I^1}} S) \xleftarrow{l} (I^2 \xrightarrow{type_{I^2}} S) \xrightarrow{r} (I^3 \xrightarrow{type_{I^3}} S)$ Kanten zurück. Somit existieren ein Knoten $k' \in I_N^2$ und eine Kante $e'_l \in I_E^1$ mit:

$$(C.437) \quad source^{I^1}(e'_l) = l_N(k')$$

$$(C.438) \quad source^{I^3}(l_E^t(e_r)) = r_N(k')$$

$$(C.439) \quad type_{I^1,E}(e'_l) = type_{I^3,E}(l_E^t(e_r))$$

Daraus folgt:

$$r_N(k') = source^{I^3}(l_E^t(e_r)) \quad (C.438)$$

$$(C.440) \quad = l_N^t(source^{P^{l^t}(I^3)}(e_r)) \quad (l^t \text{ ist Homomorphismus})$$

Weiter ist $I^2 \xleftarrow{l^t} P^{l^t}(I^2) \xrightarrow{P^{l^t}(r)} P^{l^t}(I^3)$ Pullback von $I^2 \xrightarrow{r} I^3 \xleftarrow{l^t} P^{l^t}(I^3)$ nach Lemma 11.15. Aus (C.440) und Korollar A.86 folgt somit, dass ein $z \in P^{l^t}(I^2)_N$ existiert mit:

$$(C.441) \quad l_N^t(z) = k'$$

$$(C.442) \quad P^{l^t}(r)_N(z) = source^{P^{l^t}(I^3)}(e_r)$$

Ebenfalls gilt, dass $I^3 \xleftarrow{l^t} P^{l^t}(I^3) \xrightarrow{type_{P^{l^t}(I^3)}} S\#$ Pullback von $I^3 \xrightarrow{type_{I^3}} S \xleftarrow{l^t} S\#$ nach Konstruktion 11.4 ist. Daraus folgt:

$$\begin{aligned}
& \text{type}_{I^1, E}(e'_l) = \text{type}_{E, I^3, E}(l''^t(e_r)) & (\text{C.439}) \\
(\text{C.443}) \quad & = l_E^t(\text{type}_{\mathcal{P}^{l^t}(I^3), E}(e_r)) & (\text{Pullbacks kommutieren})
\end{aligned}$$

Ferner ist auch $I^1 \xleftarrow{l'^t} \mathcal{P}^{l^t}(I^1) \xrightarrow{\text{type}_{\mathcal{P}^{l^t}(I^1)}} S^\#$ Pullback von $I^1 \xrightarrow{\text{type}_{I^1}} S \xleftarrow{l^t} S^\#$ nach Konstruktion 11.4. Zusammen mit (C.443) folgt nach Korollar A.86, dass ein $e_l \in \mathcal{P}^{l^t}(I^1)_E$ existiert mit:

$$\begin{aligned}
(\text{C.444}) \quad & l_E^t(e_l) = e'_l \\
(\text{C.445}) \quad & \text{type}_{\mathcal{P}^{l^t}(I^1)_E}(e_l) = \text{type}_{\mathcal{P}^{l^t}(I^3)_E}(e_r)
\end{aligned}$$

Weiter gilt:

$$\begin{aligned}
& l_N^t(\text{source}^{\mathcal{P}^{l^t}(I^1)}(e_l)) = \text{source}^{I^1}(l_E^t(e_l)) & (l^t \text{ ist Homomorphismus}) \\
& = \text{source}^{I^1}(e'_l) & (\text{C.444}) \\
(\text{C.446}) \quad & = l_N(k') & (\text{C.437})
\end{aligned}$$

Schließlich ist $I^2 \xleftarrow{l''^t} \mathcal{P}^{l^t}(I^2) \xrightarrow{\mathcal{P}^{l^t}(l)} \mathcal{P}^{l^t}(I^1)$ Pullback von $I^2 \xrightarrow{l} I^1 \xleftarrow{l'^t} \mathcal{P}^{l^t}(I^1)$ nach Lemma 11.15. Zusammen mit (C.446) folgt nach Korollar A.86, dass ein $z' \in \mathcal{P}^{l^t}(I^2)_N$ existiert mit:

$$\begin{aligned}
(\text{C.447}) \quad & l_N^t(z') = k' \\
(\text{C.448}) \quad & \mathcal{P}^{l^t}(l)_N(z') = \text{source}^{\mathcal{P}^{l^t}(I^1)}(e_l)
\end{aligned}$$

Nun ergibt sich:

$$\begin{aligned}
& \text{type}_{\mathcal{P}^{l^t}(I^2)_N}(z') = \text{type}_{\mathcal{P}^{l^t}(I^1)_N}(\mathcal{P}^{l^t}(l)_N(z')) & (\text{Definition einer Kommakategorie}) \\
& = \text{type}_{\mathcal{P}^{l^t}(I^1)_N}(\text{source}^{\mathcal{P}^{l^t}(I^1)}(e_l)) & (\text{C.448}) \\
& = \text{source}^{S^\#}(\text{type}_{\mathcal{P}^{l^t}(I^1)_E}(e_l)) & (\text{type}_{\mathcal{P}^{l^t}(I^1)} \text{ ist Homomorphismus}) \\
& = \text{source}^{S^\#}(\text{type}_{\mathcal{P}^{l^t}(I^3)_E}(e_r)) & (\text{C.445}) \\
& = \text{type}_{\mathcal{P}^{l^t}(I^3)_N}(\text{source}^{\mathcal{P}^{l^t}(I^3)}(e_r)) & (\text{type}_{\mathcal{P}^{l^t}(I^3)} \text{ ist Homomorphismus}) \\
& = \text{type}_{\mathcal{P}^{l^t}(I^3)_N}(\mathcal{P}^{l^t}(r)_N(z)) & (\text{C.442}) \\
(\text{C.449}) \quad & = \text{type}_{\mathcal{P}^{l^t}(I^2)_N}(z) & (\text{Definition einer Kommakategorie})
\end{aligned}$$

Nach Konstruktion 11.4 ist $I^2 \xleftarrow{l''^t} \mathcal{P}^{l^t}(I^2) \xrightarrow{\text{type}_{\mathcal{P}^{l^t}(I^2)}} S^\#$ Pullback von $I^2 \xrightarrow{\text{type}_{I^2}} S \xleftarrow{l^t} S^\#$. Somit sind l''^t und $\text{type}_{\mathcal{P}^{l^t}(I^2)}$ nach Lemma A.87 gemeinsam strikt voll. Zusammen mit (C.441), (C.447) und (C.449) folgt somit $z = z'$. Mit $k ::= z = z'$ folgt die Wahrheit der Gleichungen (C.434) und (C.435) aus (C.448) und (C.442). Die Wahrheit der Gleichung (C.436) gilt wegen (C.445). \square

C.3.4 Beweis von Lemma 11.20

Nach Voraussetzung ist die Transformation t konsistent. Somit ist es nach Anmerkung 9.19 erlaubt, $\mathcal{F}^{S'}$ als Einschränkung des Funktors $\mathcal{F}^{M^*}: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M^*)$ anzunehmen. Sei im Folgenden $\mathcal{F}^{S'}(I^1 \xrightarrow{\text{type}_{I^1}} S') = I'^1 \xrightarrow{\text{type}_{I'^1}} S'$ und $\mathcal{F}^{S'}(I^2 \xrightarrow{\text{type}_{I^2}} S') = I'^2 \xrightarrow{\text{type}_{I'^2}} S'$.¹¹ Seien ferner die MP-Homomorphismen $u^1: I^1 \rightarrow I'^1$ und $u^2: I^2 \rightarrow I'^2$ die universellen Surjektionen auf der Ausprägungsebene. Weil $\mathcal{F}^{S'}$ freier Funktor ist, gilt schließlich:

$$(C.450) \quad \mathcal{F}^{S'}(f) \circ u^1 = u^2 \circ f$$

Es wird nun gezeigt, dass $\mathcal{F}^{S'}(f)$ für jedes Prädikat $p \in P_w$ mit $w \in S^*$ vervollständigend ist. Dazu werden die Prädikate in vier Gruppen unterteilt.

- (1) Gelte $p = =_s$ für eine beliebige Sorte $s \in S$. Es wird gezeigt, dass $\mathcal{F}^{S'}(f)_s$ injektiv ist. Nach Lemma 10.14 folgt daraus, dass $\mathcal{F}^{S'}(f)$ auf dem Prädikat $=_s$ vervollständigend ist.

Seien also zwei Elemente $x, y \in I_s^1$ gegeben mit:

$$(C.451) \quad \mathcal{F}^{S'}(f)_s(x) = \mathcal{F}^{S'}(f)_s(y)$$

Zu zeigen ist die Wahrheit der Gleichung:

$$(C.452) \quad x = y$$

Weil u^1 surjektiv ist, gibt es zwei Elemente $x', y' \in I_s^1$ mit:

$$(C.453) \quad u_s^1(x') = x$$

$$(C.454) \quad u_s^1(y') = y$$

Nun gibt es zwei Fälle zu unterscheiden:

- (a) $x' = y'$. Daraus folgt unmittelbar $u_s^1(x') = u_s^1(y')$ und zusammen mit (C.453) und (C.454) die Wahrheit von (C.452).
- (b) $x' \neq y'$. Weil f nach Voraussetzung vervollständigend und nach Lemma 10.13 und Lemma A.24 injektiv ist, gilt:

$$(C.455) \quad f_s(x') \neq f_s(y')$$

Aus (C.451), (C.453) und (C.454) ergibt sich:

¹¹ Das Schema S' bleibt nach Lemma 9.13 unverändert.

$$(C.456) \quad \mathcal{F}^{S'}(f)_s(u_s^1(x')) = \mathcal{F}^{S'}(f)_s(u_s^1(y'))$$

Aus (C.456) und (C.450) folgt:

$$(C.457) \quad \begin{aligned} u_s^2(f_s(x')) &= u_s^2(f_s(y')) \\ \Leftrightarrow (f_s(x'), f_s(y')) &\in \ker_{\underline{=}_s}^{u^2} \end{aligned}$$

Gelte die folgende Implikation für beliebige $z_1, z_2 \in I_s^1$:

$$(C.458) \quad (f_s(z_1), f_s(z_2)) \in \ker_{\underline{=}_s}^{u^2} \Rightarrow (z_1, z_2) \in \ker_{\underline{=}_s}^{u^1}$$

Nach (C.457) erfüllt die Variablenbelegung

$$\begin{aligned} z_1 &\mapsto x' \\ z_2 &\mapsto y' \end{aligned}$$

die Prämisse von (C.458). Da die Gültigkeit der Implikation angenommen wird, folgt die Wahrheit der Konklusion:

$$(C.459) \quad (x', y') \in \ker_{\underline{=}_s}^{u^1}$$

Zusammen mit (C.453) und (C.454) folgt aus (C.459) die Wahrheit von (C.452).

Es bleibt die Gültigkeit von (C.458) zu zeigen. Aus (C.455) und (C.457) folgt nach der Konstruktion der Epireflexion in Konstruktion A.92, dass ein M^* -Axiom und eine entsprechende Variablenbelegung existieren, so dass die Variablenbelegung die Prämisse des Axioms in $I^2 \xrightarrow{\text{type}_{I^2}} S'$ erfüllt und in der Konklusion die gewünschte Gleichheit entweder direkt erzwingt oder über den Kongruenzabschluss herbeiführt. Das einzige M^* -Axiom, das Elemente in der Konklusion zusammenlegt, ist Axiom (M.28). Dabei werden Elemente zur Sorte O in $I^2 \xrightarrow{\text{type}_{I^2}} S'$ bzw. zur Sorte N in I^2 identifiziert. Es reicht somit aus, sich vollständig auf Elemente zur Sorte N zu beschränken, weil die Identifikation von Elementen in anderen Trägemengen über den Kongruenzabschluss sichergestellt wird. Gelte also $s ::= N$. Somit gilt nach (C.455) und (C.457):

$$(C.460) \quad f_N(x') \neq f_N(y')$$

$$(C.461) \quad u_N^2(f_N(x')) = u_N^2(f_N(y'))$$

Aus (C.460) und (C.461) und der angenommenen Anwendung des M^* -Axioms (M.28) in der Konstruktion der Epireflexion folgt die Wahrheit der Prämisse:

$$(C.462) \quad (f_N(x'), f_N(y')) \in rel^{I^2}$$

$$(C.463) \quad type_{I^2}(f_N(x')) = type_{I^2}(f_N(y'))$$

Nach Voraussetzung ist f vervollständigend und nach Lemma 10.13 strikt voll. Zusammen mit (C.462) folgt:

$$(C.464) \quad (x', y') \in rel^{I^1}$$

Aus der Definition einer Kommakategorie folgt $type_{I^1} = type_{I^2} \circ f$. Zusammen mit (C.463) ergibt sich:

$$(C.465) \quad type_{I^1}(x') = type_{I^1}(y')$$

Nach (C.464) und (C.465) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto x' \\ y &\mapsto y' \end{aligned}$$

die Prämisse des M^* -Axioms (M.28) in $I^1 \xrightarrow{type_{I^1}} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt $u_N^1(x') = u_N^1(y')$ und somit die Wahrheit von (C.458).

- (2) Gelte $p = software$. Nach Anmerkung 10.12 muss lediglich gezeigt werden, dass $\mathcal{F}^{S'}(f)$ auf diesem Prädikat strikt voll ist. Sei also ein Element $x \in I_s^{I^1}$ gegeben mit:

$$(C.466) \quad \mathcal{F}^{S'}(f)_s(x) \in p^{I^2}$$

Weil u^1 surjektiv ist, existiert ein Element $x' \in I_s^1$ mit:

$$(C.467) \quad u_s^1(x') = x$$

Aus (C.466), (C.467) und (C.450) folgt:

$$(C.468) \quad \begin{aligned} u_s^2(f_s(x')) &\in p^{I^2} \\ \Leftrightarrow f_s(x') &\in ker_{=s}^{u^2} \end{aligned}$$

Aus (C.468) und nach Konstruktion der Epireflexion in Konstruktion A.92 gilt offenbar

$$(C.469) \quad f_s(x') \in p^{I^2}$$

weil kein M^* -Axiom das Prädikat *message* wahr macht. Weil f nach Voraussetzung vervollständigend und nach Lemma 10.13 strikt voll ist, folgt aus (C.469):

$$(C.470) \quad x' \in p^{I^1}$$

Daraus folgt unmittelbar $u_s^1(x') \in p^{I^1}$ und mit (C.467) schließlich $x \in p^{I^1}$.

- (3) Gelte $p = rel$ und somit $w = N N$. Analog zum letzten Punkt ergibt sich, dass $\mathcal{F}^{S'}(f)$ auf *rel* strikt voll ist, weil kein M^* -Axiom das Prädikat *relO* wahr macht. Es bleibt zu zeigen, dass $\mathcal{F}^{S'}(f)$ für $w' = N$ vervollständigend ist. Sei also ein Tupel $(\underline{x}, \bar{x}) \in I_w^{I^2}$ und ein Element $\underline{y} \in I_N^1$ gegeben mit:¹²

$$(C.471) \quad (\underline{x}, \bar{x}) \in rel^{I^2}$$

$$(C.472) \quad \mathcal{F}^{S'}(f)_N(\underline{y}) = \underline{x}$$

Es ist nachzuweisen, dass ein Element $\bar{y} \in I_N^1$ existiert mit:

$$(C.473) \quad \mathcal{F}^{S'}(f)_N(\bar{y}) = \bar{x}$$

$$(C.474) \quad (\underline{y}, \bar{y}) \in rel^{I^1}$$

Weil u^1 surjektiv ist, existiert ein Element $\underline{y}' \in I_N^1$ mit:

$$(C.475) \quad u_N^1(\underline{y}') = \underline{y}$$

Weil u^2 surjektiv ist, existiert ein Tupel $(\underline{x}', \bar{x}') \in I_w^2$ mit:

$$(C.476) \quad u_N^2(\underline{x}') = \underline{x}$$

$$(C.477) \quad u_N^2(\bar{x}') = \bar{x}$$

$$(C.478) \quad (\underline{x}', \bar{x}') \in rel^{I^2}$$

Dabei gilt (C.478), weil kein M^* -Axiom das Prädikat *relO* wahr macht und somit ein Urbild existieren muss, für welches das Prädikat wahr ist. Nun gibt es zwei Fälle zu unterscheiden:

- (a) $f_N(\underline{y}') = \underline{x}'$. Nach (C.478) und weil f nach Voraussetzung vervollständigend ist, gibt es ein Element $\bar{y}' \in I_N^1$ mit:

$$(C.479) \quad f_N(\bar{y}') = \bar{x}'$$

$$(C.480) \quad (\underline{y}', \bar{y}') \in rel^{I^1}$$

¹² Der Fall $\mathcal{F}^{S'}(f)_N(\underline{y}) = \bar{x}$ wird analog durch Vertauschen von \bar{x} und \underline{x} bewiesen.

Aus (C.480) und (C.475) folgt:

$$(C.481) \quad (\underline{y}, u_N^1(\bar{y}')) \in rel^{I^1}$$

Daraus folgt:

$$\mathcal{F}^{S'}(f)_N(u_N^1(\bar{y}')) = u_N^2(f_N(\bar{y}')) \quad (C.450)$$

$$= u_N^2(\underline{x}') \quad (C.479)$$

$$(C.482) \quad = \bar{x} \quad (C.477)$$

Aus (C.480) und (C.482) folgt, dass (C.473) und (C.474) mit $\bar{y} ::= u_N^1(\bar{y}')$ erfüllt sind.

(b) $f_N(\underline{y}') \neq \underline{x}'$. Es gilt:

$$u_N^2(f_N(\underline{y}')) = \mathcal{F}^{S'}(f)_N(u_N^1(\underline{y}')) \quad (C.450)$$

$$= \mathcal{F}^{S'}(f)_N(\underline{y}) \quad (C.475)$$

$$= \underline{x} \quad (C.472)$$

$$(C.483) \quad = u_N^2(\underline{x}') \quad (C.476)$$

Aus (C.483) folgt $(f_N(\underline{y}'), \underline{x}') \in ker_{\underline{N}}^{u^2}$. Nach Konstruktion der Epireflexion in Konstruktion A.92 gilt offenbar

$$(C.484) \quad (f_N(\underline{y}'), \underline{x}') \in rel^{I^2}$$

weil nur das M^* -Axiom (M.28) Elemente in der Konklusion zusammenlegt und für die Anwendung des Axioms in der Konstruktion der Epireflexion dessen Prämisse erfüllt sein muss. Aus (C.484) und (C.478) folgt, dass die Variablenbelegung

$$x \mapsto f_N(\underline{y}')$$

$$y \mapsto \underline{x}'$$

$$z \mapsto \bar{x}'$$

die Prämisse des MP -Axioms (MP.8) in I^2 erfüllt. Weil I^2 nach Voraussetzung ein MP -System ist, folgt die Wahrheit der Konklusion:

$$(C.485) \quad (f_N(\underline{y}'), \bar{x}') \in rel^{I^2}$$

Weil f nach Voraussetzung vervollständigend ist, folgt aus (C.485), dass ein Element $z \in I_N^1$ existiert mit:

$$(C.486) \quad f_N(z) = \bar{x}'$$

$$(C.487) \quad (\underline{y}', z) \in \text{rel}^{I^1}$$

Aus (C.487) und (C.475) folgt:

$$(C.488) \quad (\underline{y}, u_N^1(z)) \in \text{rel}^{I^1}$$

Daraus folgt:

$$\mathcal{F}^{S'}(f)_N(u_N^1(z)) = u_N^2(f_N(z)) \quad (C.450)$$

$$= u_N^2(\bar{x}') \quad (C.486)$$

$$(C.489) \quad = \bar{x} \quad (C.477)$$

Aus (C.488) und (C.489) folgt, dass (C.473) und (C.474) mit $\bar{y} ::= u_N^1(z)$ erfüllt sind.

(4) Gelte $p = \text{under}$. Sei ein Tupel $x = (\underline{x}, \bar{x}) \in I_w'^2$ und ein Tupel $y \in I_{w'}^1$ gegeben mit:

$$(C.490) \quad x \in \text{under}^{I'^2}$$

$$(C.491) \quad \mathcal{F}^{S'}(f)_{w'}(y) = \langle x \rangle_{w'}$$

Es ist zu zeigen, dass ein Tupel $\hat{y} \in I_w^1$ existiert mit:

$$(C.492) \quad \langle \hat{y} \rangle_{w'} = y$$

$$(C.493) \quad \mathcal{F}^{S'}(f)_w(\hat{y}) = x$$

$$(C.494) \quad \hat{y} \in \text{under}^{I^1}$$

Weil u^1 surjektiv ist, existiert ein Tupel $y' \in I_{w'}^1$, mit:

$$(C.495) \quad u_{w'}^1(y') = y$$

Weil u^2 surjektiv ist, existiert ein Tupel $x' = (\underline{x}', \bar{x}') \in I_w^2$ mit:

$$(C.496) \quad u_w^2(x') = x$$

Nun sind vier Fälle zu unterscheiden:

(a) $x' \in \text{under}^{I^2}$ und $f_{w'}(y') = \langle x' \rangle_{w'}$. Weil f nach Voraussetzung vervollständigend ist, folgt daraus die Existenz eines Tupels $\hat{y}' \in I_w^1$ mit:

$$(C.497) \quad \langle \hat{y}' \rangle_{w'} = y'$$

$$(C.498) \quad f_w(\hat{y}') = x'$$

$$(C.499) \quad \hat{y}' \in \text{under}^{I^1}$$

Aus (C.499) folgt unmittelbar:

$$(C.500) \quad u_w^1(\hat{y}') \in \text{under}^{I^1}$$

Weiter gilt:

$$\langle u_w^1(\hat{y}') \rangle_{w'} = u_{w'}^1(\langle \hat{y}' \rangle_{w'}) \quad (u^1 \text{ ist Homomorphismus})$$

$$= u_{w'}^1(y') \quad (C.497)$$

$$(C.501) \quad = y \quad (C.495)$$

Schließlich gilt:

$$\mathcal{F}^{S'}(f)_w(u_w^1(\hat{y}')) = u_w^2(f_w(\hat{y}')) \quad (C.450)$$

$$= u_w^2(x') \quad (C.498)$$

$$(C.502) \quad = x \quad (C.496)$$

Aus (C.500), (C.501) und (C.502) folgt, dass (C.492), (C.493) und (C.494) mit $\hat{y} ::= u_w^1(\hat{y}')$ erfüllt sind.

(b) $x' \in \text{under}^{I^2}$ und $f_{w'}(y') \neq \langle x' \rangle_{w'}$. Es wird nach den möglichen Sortenwörtern w' unterschieden:

(i) $w' = w$. Nach (C.491) erfüllt $\hat{y} ::= y$ offenbar die Gleichungen (C.492) und (C.493). Es bleibt die Gültigkeit von (C.494) zu zeigen. Weil $w = w'$, gilt $y = (\underline{y}, \bar{y})$ und $y' = (\underline{y}', \bar{y}')$. Nach (C.495) und (C.491) gilt weiter:

$$(C.503) \quad u_N^1(\underline{y}') = \underline{y}$$

$$(C.504) \quad u_N^1(\bar{y}') = \bar{y}$$

$$(C.505) \quad \mathcal{F}^{S'}(f)_N(\underline{y}) = \underline{x}$$

$$(C.506) \quad \mathcal{F}^{S'}(f)_N(\bar{y}) = \bar{x}$$

Daraus folgt:

$$u_N^2(f_N(\underline{y}')) = \mathcal{F}^{S'}(f)_N(u_N^1(\underline{y}')) \quad (C.450)$$

$$= \mathcal{F}^{S'}(f)_N(\underline{y}) \quad (C.503)$$

$$= \underline{x} \quad (\text{C.505})$$

$$(C.507) \quad = u_N^2(\underline{x}') \quad (\text{C.496})$$

Analog folgt:

$$(C.508) \quad u_N^2(f_N(\bar{y}')) = u_N^2(\bar{x}')$$

Aus (C.507) und (C.508) folgt $(f_N(\underline{y}'), \underline{x}') \in \ker_{\underline{N}}^{u^2}$ und $(f_N(\bar{y}'), \bar{x}') \in \ker_{\bar{N}}^{u^2}$. Nach Konstruktion der Epireflexion in Konstruktion A.92 gilt offenbar

$$(C.509) \quad (f_N(\underline{y}'), \underline{x}') \in \text{rel}^{I^2}$$

$$(C.510) \quad (f_N(\bar{y}'), \bar{x}') \in \text{rel}^{I^2}$$

$$(C.511) \quad \text{type}_{I^2, w}(f_w(\underline{y}')) = \text{type}_{I^2, w}(x')$$

weil nur das M^* -Axiom (M.28) Elemente in der Konklusion zusammenlegt und für die Anwendung des Axioms in der Konstruktion der Epireflexion dessen Prämisse erfüllt sein muss. Weil f nach Voraussetzung vervollständigend ist, folgt aus (C.509) und (C.510), dass ein Tupel $z = (\underline{z}, \bar{z}) \in I_w^1$ existiert mit:

$$(C.512) \quad f_w(z) = x'$$

$$(C.513) \quad (\underline{y}', \underline{z}) \in \text{rel}^{I^1}$$

$$(C.514) \quad (\bar{y}', \bar{z}) \in \text{rel}^{I^1}$$

Weil f nach Voraussetzung vervollständigend und nach Lemma 10.13 strikt voll ist, folgt aus (C.512) und der Voraussetzung $x' \in \text{under}^{I^2}$:

$$(C.515) \quad z \in \text{under}^{I^1}$$

Weiter gilt:

$$\text{type}_{I^1, w}(z) = \text{type}_{I^2, w}(f_w(z)) \quad (\text{Definition einer Kommakategorie})$$

$$= \text{type}_{I^2, w}(x') \quad (\text{C.512})$$

$$= \text{type}_{I^2, w}(f_w(\underline{y}')) \quad (\text{C.511})$$

$$(C.516) \quad = \text{type}_{I^1, w}(\underline{y}') \quad (\text{Definition einer Kommakategorie})$$

Nach (C.513), (C.514) und (C.516) erfüllen die Variablenbelegungen

$$\begin{aligned} x &\mapsto \underline{y}' \\ y &\mapsto \underline{z} \end{aligned}$$

und

$$\begin{aligned} x &\mapsto \bar{y}' \\ y &\mapsto \bar{z} \end{aligned}$$

jeweils die Prämisse des M' -Axioms (M.28) in $I^1 \xrightarrow{\text{type}_{I^1}} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt die Wahrheit der Konklusion in $I'^1 \xrightarrow{\text{type}_{I'^1}} S'$:

$$(C.517) \quad u_w^1(y') = u_w^1(z)$$

Daraus folgt:

$$\begin{aligned} z &\in \text{under}^{I^1} && (C.515) \\ \Rightarrow u_w^1(z) &\in \text{under}^{I'^1} && (u^1 \text{ ist Homomorphismus}) \\ \Rightarrow u_w^1(y') &\in \text{under}^{I'^1} && (C.517) \\ (C.518) \quad \Rightarrow y &\in \text{under}^{I'^1} && (C.503) \text{ und } (C.504) \end{aligned}$$

Aus (C.518) folgt mit $\hat{y} ::= y$ die Gültigkeit von (C.494).

(ii) $w' = N$. O. B. d. A. gelte:

$$(C.519) \quad \mathcal{F}^{S'}(f)_N(y) = \underline{x}$$

Es ist zu zeigen, dass ein Element $\bar{y} \in I_N^1$ existiert mit:

$$(C.520) \quad \mathcal{F}^{S'}(f)_N(\bar{y}) = \bar{x}$$

$$(C.521) \quad (y, \bar{y}) \in \text{under}^{I'^1}$$

Mit $\hat{y} ::= (y, \bar{y})$ folgt dann aus (C.520) und (C.521) die Gültigkeit von (C.492) bis (C.494).

Es gilt:

$$u_N^2(f_N(y')) = \mathcal{F}^{S'}(f)_N(u_N^1(y')) \quad (C.450)$$

$$= \mathcal{F}^{S'}(f)_N(y) \quad (C.495)$$

$$= \underline{x} \quad (\text{C.519})$$

$$(C.522) \quad = u_N^2(\underline{x}') \quad (\text{C.496})$$

Aus (C.522) folgt $(f_N(y'), \underline{x}') \in \ker_{\underline{=}_N}^{u^2}$. Weiter gilt nach Voraussetzung $f_N(y') \neq \underline{x}'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 gilt offenbar

$$(C.523) \quad (f_N(y'), \underline{x}') \in \text{rel}^{I^2}$$

$$(C.524) \quad \text{type}_{I^2, N}(f_N(y')) = \text{type}_{I^2, N}(\underline{x}')$$

weil nur das M^* -Axiom (M.28) Elemente in der Konklusion zusammenlegt und für die Anwendung des Axioms in der Konstruktion der Epireflexion dessen Prämisse erfüllt sein muss. Weil f nach Voraussetzung vervollständigend ist, folgt aus (C.523), dass ein Element $\underline{z} \in I_N^1$ existiert mit:

$$(C.525) \quad f_N(\underline{z}) = \underline{x}'$$

$$(C.526) \quad (y', \underline{z}) \in \text{rel}^{I^1}$$

Nach Voraussetzung gilt $x' \in \text{under}^{I^2}$. Ferner ist f nach Voraussetzung vervollständigend. Zusammen mit (C.525) folgt die Existenz eines Elements $\bar{z} \in I_N^1$ mit:

$$(C.527) \quad f_N(\bar{z}) = \bar{x}'$$

$$(C.528) \quad (\underline{z}, \bar{z}) \in \text{under}^{I^1}$$

Weiter gilt:

$$\text{type}_{I^1, N}(\underline{z}) = \text{type}_{I^2, N}(f_N(\underline{z})) \quad (\text{Definition einer Kommakategorie})$$

$$= \text{type}_{I^2, N}(\underline{x}') \quad (\text{C.525})$$

$$= \text{type}_{I^2, N}(f_N(y')) \quad (\text{C.524})$$

$$(C.529) \quad = \text{type}_{I^1, N}(y') \quad (\text{Definition einer Kommakategorie})$$

Nach (C.526) und (C.529) erfüllt die Variablenbelegungen

$$x \mapsto y'$$

$$y \mapsto \underline{z}$$

die Prämisse des M' -Axioms (M.28) in $I^1 \xrightarrow{\text{type}_{I^1}} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt die Wahrheit der Konklusion in $I'^1 \xrightarrow{\text{type}_{I'^1}} S'$:

$$(C.530) \quad u_N^1(y') = u_N^1(\underline{z})$$

Daraus folgt:

$$(z, \bar{z}) \in \text{under}^{I^1} \quad (\text{C.528})$$

$$\Rightarrow (u_N^1(z), u_N^1(\bar{z})) \in \text{under}^{I^1} \quad (u^1 \text{ ist Homomorphismus})$$

$$\Rightarrow (u_N^1(y'), u_N^1(\bar{z})) \in \text{under}^{I^1} \quad (\text{C.530})$$

$$(\text{C.531}) \quad \Rightarrow (y, u_N^1(\bar{z})) \in \text{under}^{I^1} \quad (\text{C.495})$$

Schließlich gilt:

$$\mathcal{F}^{S'}(f)_N(u_N^1(\bar{z})) = u_N^2(f_N(\bar{z})) \quad (\text{C.450})$$

$$= u_N^2(\bar{x}') \quad (\text{C.527})$$

$$(\text{C.532}) \quad = \bar{x} \quad (\text{C.496})$$

Aus (C.531) und (C.532) folgt, dass (C.520) und (C.521) mit $\bar{y} ::= u_N^1(\bar{z})$ erfüllt sind.

- (c) $x' \notin \text{under}^{I^2}$ und $f_{w'}(y') = \langle x' \rangle_{w'}$. Nach Satz 9.17 folgt, dass in I^2 die Elemente \underline{x}' und \bar{x}' über einen $(n+2)$ -elementigen erweiterten Pfad ($n \in \mathbb{N}_0$) miteinander verbunden sind. Dies ist in Abb. C.6 dargestellt, wobei die Notation $x \rightsquigarrow y$ für $(x, y) \in \text{rel}^{I^2}$ und die Notation $x \longrightarrow y$ für $(x, y) \in \text{under}^{I^2}$ steht. Dabei können die Elemente jedes Paares auch zusammenfallen.

$$\underline{x}' \longrightarrow x'_1 \rightsquigarrow x'_2 \longrightarrow x'_3 \rightsquigarrow x'_4 \longrightarrow x'_5 \rightsquigarrow \dots \rightsquigarrow x'_n \longrightarrow \bar{x}'$$

Abb. C.6: Beziehung zwischen \underline{x}' und \bar{x}' in I^2

Für diese Elemente gilt außerdem wegen der zu erfüllenden Prämisse des M^* -Axioms (M.28) für alle $i \in \mathbb{N}^+$:

$$(\text{C.533}) \quad \text{type}_{I^2, N}(x'_{2i-1}) = \text{type}_{I^2, N}(x'_{2i})$$

Weil f nach Voraussetzung vervollständigend ist und weil nach Voraussetzung $f_{w'}(y') = \langle x' \rangle_{w'}$ gilt, existiert eine äquivalente Situation in I^1 mit den Elementen $\underline{y}', y'_1, \dots, y'_n, \bar{y}' \in I_N^1$, für die gilt:

$$(\text{C.534}) \quad \begin{aligned} f_N(\underline{y}') &= \underline{x}' \\ f_N(y'_1) &= x'_1 \\ &\dots \\ f_N(y'_n) &= x'_n \\ f_N(\bar{y}') &= \bar{x}' \end{aligned}$$

(Abb. C.7).

$$\underline{y}' \longrightarrow \triangleright y'_1 \rightsquigarrow y'_2 \longrightarrow \triangleright y'_3 \rightsquigarrow y'_4 \longrightarrow \triangleright y'_5 \rightsquigarrow \dots \rightsquigarrow y'_n \longrightarrow \triangleright \bar{y}'$$

Abb. C.7: Urbild-Situation in I^1

Nun sei definiert:

$$(C.535) \quad \hat{y}' ::= (\underline{y}', \bar{y}')$$

Nach (C.535) und (C.534) gilt offenbar:

$$(C.536) \quad f_w(\hat{y}') = x'$$

$$(C.537) \quad \Rightarrow f_{w'}(\langle \hat{y}' \rangle_{w'}) = \langle x' \rangle_{w'}$$

Nach Voraussetzung gilt $f_{w'}(y') = \langle x' \rangle_{w'}$. Weil f vervollständigend und nach Lemma 10.13 und Lemma A.24 injektiv ist, folgt aus (C.537):

$$(C.538) \quad \langle \hat{y}' \rangle_{w'} = y'$$

Daraus ergibt sich:

$$y = u_w^1(y') \quad (C.495)$$

$$= u_w^1(\langle \hat{y}' \rangle_{w'}) \quad (C.538)$$

$$(C.539) \quad = \langle u_w^1(\hat{y}') \rangle_{w'} \quad (u^1 \text{ ist Homomorphismus})$$

Aus (C.533) und (C.534) folgt weiter für alle $i \in \mathbb{N}^+$:

$$(C.540) \quad type_{I^1, N}(y'_{2i-1}) = type_{I^1, N}(y'_{2i})$$

Aus (C.540) ergibt sich, dass auf Grund des “Zurückziehens” der Elemente entlang f inklusive der Relationen zwischen ihnen in I^1 dieselben Voraussetzungen für die Konstruktion der Epireflexion wie in I^2 vorliegen. Daraus folgt:

$$(C.541) \quad u_w^1(\hat{y}') \in \text{under}^{I^1}$$

Schließlich folgt:

$$\mathcal{F}^{S'}(f)_w(u_w^1(\hat{y}')) = u_w^2(f_w(\hat{y}')) \quad (\text{C.450})$$

$$= u_w^2(x') \quad (\text{C.536})$$

$$(C.542) \quad = x \quad (\text{C.496})$$

Aus (C.539), (C.541) und (C.542) folgt, dass (C.492), (C.493) und (C.494) mit $\hat{y} ::= u_w^1(\hat{y}')$ erfüllt sind.

- (d) $x' \notin \text{under}^{I^2}$ und $f_{w'}(y') \neq \langle x' \rangle_{w'}$. Dies wird durch eine Kombination der letzten beiden Fälle bewiesen. \square

C.3.5 Beweis von Lemma 11.21

Sei im Folgenden $\mathcal{F}^{S'}(I^1 \xrightarrow{\text{type}_{e_1^1}} S') = I'^1 \xrightarrow{\text{type}_{e_1'^1}} S'$, $\mathcal{F}^{S'}(I^2 \xrightarrow{\text{type}_{e_1^2}} S') = I'^2 \xrightarrow{\text{type}_{e_1'^2}} S'$ und $\mathcal{F}^{S'}(I^3 \xrightarrow{\text{type}_{e_1^3}} S') = I'^3 \xrightarrow{\text{type}_{e_1'^3}} S'$.¹³ Seien ferner die MP-Homomorphismen $u^1: I^1 \rightarrow I'^1$, $u^2: I^2 \rightarrow I'^2$ und $u^3: I^3 \rightarrow I'^3$ die universellen Surjektionen auf der Ausprägungsebene. Sei nun $e'_R \in I'_E^3$ eine Kante in I'^3 . Zu zeigen ist, dass ein Knoten $k' \in I'_N^2$ und eine Kante $e'_L \in I'_E^1$ existieren mit:

$$(C.543) \quad \text{source}^{I'^1}(e'_L) = \mathcal{F}^{S'}(l)_N(k')$$

$$(C.544) \quad \text{source}^{I'^3}(e'_R) = \mathcal{F}^{S'}(r)_N(k')$$

$$(C.545) \quad \text{type}_{I'^1, E}(e'_L) = \text{type}_{I'^3, E}(e'_R)$$

Aus der Surjektivität von u^3 resultiert die Existenz einer Kante $e_R \in I_E^3$ mit:

$$(C.546) \quad u_E^3(e_R) = e'_R$$

Weil die Aktion $(I^1 \xrightarrow{\text{type}_{e_1^1}} S) \xleftarrow{l} (I^2 \xrightarrow{\text{type}_{e_1^2}} S) \xrightarrow{r} (I^3 \xrightarrow{\text{type}_{e_1^3}} S)$ nach Voraussetzung konsistent ist, zieht sie Kanten zurück. Nach (C.546) existiert somit ein Knoten $k \in I_N^2$ und eine Kante $e_L \in I_E^1$ mit:

$$(C.547) \quad \text{source}^{I^1}(e_L) = l_N(k)$$

$$(C.548) \quad \text{source}^{I^3}(e_R) = r_N(k)$$

$$(C.549) \quad \text{type}_{I^1, E}(e_L) = \text{type}_{I^3, E}(e_R)$$

Weiter gilt, weil $\mathcal{F}^{S'}$ freier Funktor ist:

$$(C.550) \quad \mathcal{F}^{S'}(l) \circ u^2 = u^1 \circ l$$

$$(C.551) \quad \mathcal{F}^{S'}(r) \circ u^2 = u^3 \circ r$$

¹³ Das Schema S' bleibt nach Lemma 9.13 unverändert.

Daraus folgt zum einen:

$$\begin{aligned}
 \text{(C.552)} \quad \text{source}^{I^1}(u_E^1(e_L)) &= u_N^1(\text{source}^{I^1}(e_L)) && (u^1 \text{ ist Homomorphismus}) \\
 &= u_N^1(l_N(k)) && \text{(C.547)} \\
 &= \mathcal{F}^{S'}(l)_N(u_N^2(k)) && \text{(C.550)}
 \end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
 \text{(C.553)} \quad \text{source}^{I^3}(u_E^3(e_R)) &= u_N^3(\text{source}^{I^3}(e_R)) && (u^3 \text{ ist Homomorphismus}) \\
 &= u_N^3(r_N(k)) && \text{(C.548)} \\
 &= \mathcal{F}^{S'}(r)_N(u_N^2(k)) && \text{(C.551)}
 \end{aligned}$$

Schließlich ergibt sich:

$$\begin{aligned}
 \text{(C.554)} \quad \text{type}_{I^1,E}(u_E^1(e_L)) &= \text{type}_{I^1,E}(e_L) && \text{(Definition einer Kommakategorie)} \\
 &= \text{type}_{I^3,E}(e_R) && \text{(C.549)} \\
 &= \text{type}_{I^3,E}(u_E^3(e_R)) && \text{(Definition einer Kommakategorie)}
 \end{aligned}$$

Aus (C.552), (C.553) und (C.554) folgt mit $k' ::= u_N^2(k)$ und $e'_L ::= u_E^1(e_L)$ die Wahrheit der Gleichungen (C.543), (C.544) und (C.545). \square

C.3.6 Beweis von Lemma 11.28

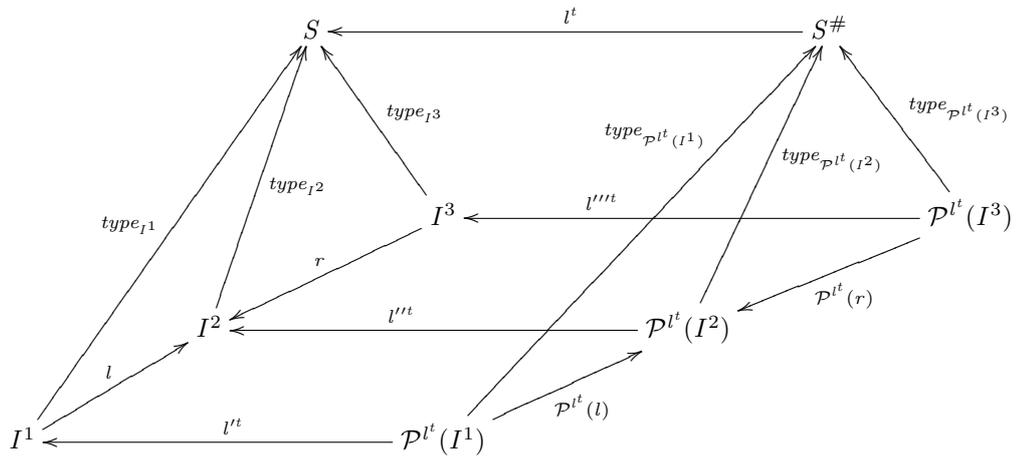


Abb. C.8: Anwendung des Pullback-Funktors auf Kospanns

Es reicht, die Ausprägungs-Anteile der Morphismen $\mathcal{P}^{l^t}(l)$ und $\mathcal{P}^{l^t}(r)$ zu betrachten (Abb. C.8). Sei $x \in \mathcal{P}^{l^t}(I^2)_s$ mit $s \in S$ ein beliebiges Element. Nach Voraussetzung sind l und r gemeinsam surjektiv. Somit existiert für $l_s''^t(x)$ ein Urbild:

(1) Gelte

$$(C.555) \quad l_s'''^t(x) = l_s(y)$$

mit $y \in I_s^1$. Nach Lemma 11.15 ist $I^1 \xleftarrow{l''^t} \mathcal{P}^{l^t}(I^1) \xrightarrow{\mathcal{P}^{l^t}(l)} \mathcal{P}^{l^t}(I^2)$ Pullback von $I^1 \xrightarrow{l} I^2 \xleftarrow{l''^t} \mathcal{P}^{l^t}(I^2)$. Nach Lemma A.52 folgt, dass ein eindeutiges $z \in \mathcal{P}^{l^t}(I^1)_s$ existiert mit:

$$(C.556) \quad l_s^t(z) = y$$

$$(C.557) \quad \mathcal{P}^{l^t}(l)_s(z) = x$$

Aus (C.557) folgt die zu beweisende Behauptung.

(2) Gelte

$$(C.558) \quad l_s'''^t(x) = r_s(y)$$

mit $y \in I_s^3$. Dies wird analog zum letzten Punkt bewiesen. □

C.3.7 Beweis von Lemma 11.29

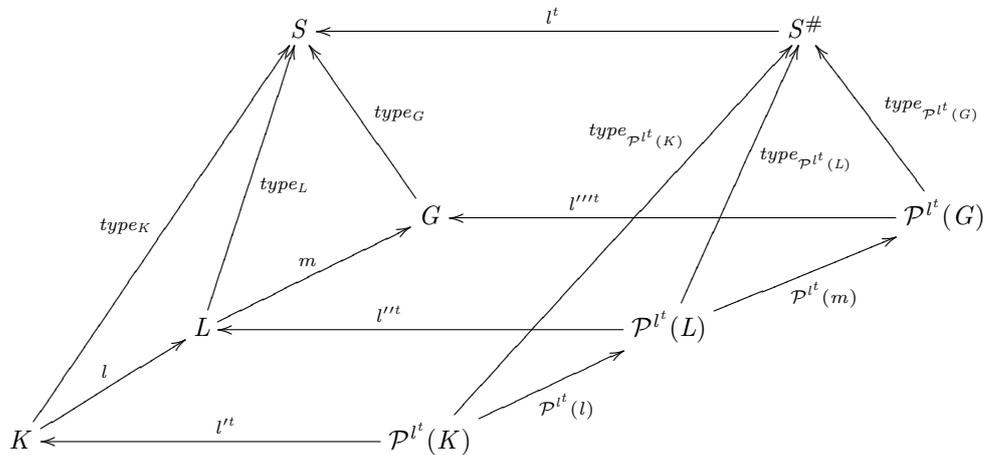


Abb. C.9: Anwendung des Pullback-Funktors auf hintereinander geschaltete Morphismen

Es reicht, die Ausprägungs-Anteile der Morphismen $\mathcal{P}^{l^t}(l)$ und $\mathcal{P}^{l^t}(r)$ zu betrachten (Abb. C.9). Weil m injektiv bis auf l ist, gilt nach Definition B.25 die Implikation

$$(C.559) \quad m_s(x) = m_s(y) \wedge x \neq y \Rightarrow x, y \in \text{Im } l_s$$

für alle $x, y \in L_s$ mit $s \in S$. Es ist zu zeigen, dass dann die folgende Implikation für alle $x, y \in \mathcal{P}^{l^t}(L)_s$ mit $s \in S$ gilt:

$$(C.560) \quad \mathcal{P}^{l^t}(m)_s(x) = \mathcal{P}^{l^t}(m)_s(y) \wedge x \neq y \Rightarrow x, y \in \text{Im } \mathcal{P}^{l^t}(l)_s$$

Seien nun zwei Elemente $x, y \in \mathcal{P}^{l^t}(L)_s$ gegeben, welche die Prämisse von (C.560) erfüllen. Nach Lemma 11.15 ist $L \xleftarrow{l''^t} \mathcal{P}^{l^t}(L) \xrightarrow{\mathcal{P}^{l^t}(m)} \mathcal{P}^{l^t}(G)$ Pullback von $L \xrightarrow{m} G \xleftarrow{l'''^t} \mathcal{P}^{l^t}(G)$. Weil die Pullback-Morphismen l''^t und $\mathcal{P}^{l^t}(m)$ nach Lemma A.53 gemeinsam strikt voll sind, folgt aus der angenommenen Wahrheit der Prämisse von (C.560):

$$(C.561) \quad l_s'''^t(x) \neq l_s'''^t(y)$$

Weil Pullbacks kommutieren, folgt weiterhin:

$$\begin{aligned} m_s(l_s'''^t(x)) &= l_s'''^t(\mathcal{P}^{l^t}(m)_s(x)) && \text{(Pullbacks kommutieren)} \\ &= l_s'''^t(\mathcal{P}^{l^t}(m)_s(y)) && \text{(Prämisse von (C.560))} \\ (C.562) \quad &= m_s(l_s'''^t(y)) && \text{(Pullbacks kommutieren)} \end{aligned}$$

Aus (C.561) und (C.562) folgt, dass die Variablenbelegung

$$\begin{aligned} x &\mapsto l_s'''^t(x) \\ y &\mapsto l_s'''^t(y) \end{aligned}$$

die Prämisse von (C.559) erfüllt. Weil m injektiv bis auf l ist, folgt die Wahrheit der Konklusion:

$$(C.563) \quad l_s'''^t(x), l_s'''^t(y) \in \text{Im } l_s$$

Seien x' und y' die Urbilder von $l_s'''^t(x)$ und $l_s'''^t(y)$ unter l . Nach Lemma 11.15 ist $K \xleftarrow{l''^t} \mathcal{P}^{l^t}(K) \xrightarrow{\mathcal{P}^{l^t}(l)} \mathcal{P}^{l^t}(L)$ Pullback von $K \xrightarrow{l} L \xleftarrow{l'''^t} \mathcal{P}^{l^t}(L)$. Nach Lemma A.52 folgt, dass zwei eindeutige Elemente $z, z' \in \mathcal{P}^{l^t}(K)_s$ existieren mit:

$$(C.564) \quad l_s''^t(z) = x'$$

$$(C.565) \quad \mathcal{P}^{l^t}(l)_s(z) = x$$

$$(C.566) \quad l_s''^t(z') = y'$$

$$(C.567) \quad \mathcal{P}^{l^t}(l)_s(z') = y$$

Aus (C.565) und (C.567) folgt jedoch die zu beweisende Wahrheit der Konklusion von (C.560). \square

C.3.8 Beweis von Lemma 11.32

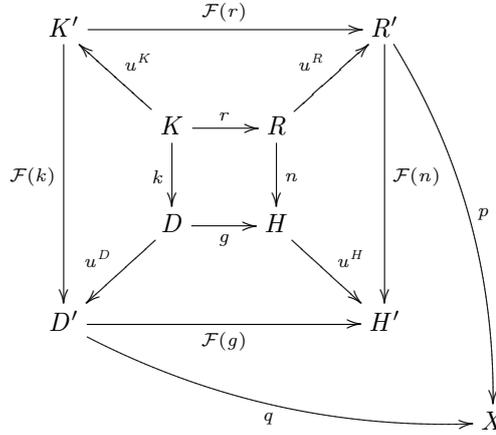


Abb. C.10: Anwendung eines Epireflectors auf ein Pushout-Diagramm

Es sei im Folgenden definiert:

$$\begin{aligned} K' \xrightarrow{\text{type}_{K'}} S &::= \mathcal{F}(K \xrightarrow{\text{type}_K} S) \\ R' \xrightarrow{\text{type}_{R'}} S &::= \mathcal{F}(R \xrightarrow{\text{type}_R} S) \\ D' \xrightarrow{\text{type}_{D'}} S &::= \mathcal{F}(D \xrightarrow{\text{type}_D} S) \\ H' \xrightarrow{\text{type}_{H'}} S &::= \mathcal{F}(H \xrightarrow{\text{type}_H} S) \end{aligned}$$

Um nachzuweisen, dass

$$(D' \xrightarrow{\text{type}_{D'}} S) \xrightarrow{\mathcal{F}(g)} (H' \xrightarrow{\text{type}_{H'}} S) \xleftarrow{\mathcal{F}(n)} (R' \xrightarrow{\text{type}_{R'}} S)$$

Pushout von

$$(D' \xrightarrow{\text{type}_{D'}} S) \xleftarrow{\mathcal{F}(k)} (K' \xrightarrow{\text{type}_{K'}} S) \xrightarrow{\mathcal{F}(r)} (R' \xrightarrow{\text{type}_{R'}} S)$$

in $\mathbf{Alg}(\Sigma)\downarrow S$ ist, seien ein $\mathbf{Alg}(\Sigma)\downarrow S$ -System $X \xrightarrow{\text{type}_X} S$ sowie zwei $\mathbf{Alg}(\Sigma)\downarrow S$ -Morphismen $p: (R' \xrightarrow{\text{type}_{R'}} S) \rightarrow (X \xrightarrow{\text{type}_X} S)$ und $q: (D' \xrightarrow{\text{type}_{D'}} S) \rightarrow (X \xrightarrow{\text{type}_X} S)$ gegeben mit:

$$(C.568) \quad p \circ \mathcal{F}(r) = q \circ \mathcal{F}(k)$$

Daraus folgt:

$$\begin{aligned}
 p \circ u^R \circ r &= p \circ \mathcal{F}(r) \circ u^K && (\mathcal{F} \text{ ist freier Funktor}) \\
 &= q \circ \mathcal{F}(k) \circ u^K && (\text{C.568}) \\
 (\text{C.569}) \quad &= q \circ u^D \circ k && (\mathcal{F} \text{ ist freier Funktor})
 \end{aligned}$$

Nach Voraussetzung ist

$$(D \xrightarrow{\text{type}_D} S) \xrightarrow{g} (H \xrightarrow{\text{type}_H} S) \xleftarrow{n} (R \xrightarrow{\text{type}_R} S)$$

Pushout von

$$(D \xrightarrow{\text{type}_D} S) \xleftarrow{k} (K \xrightarrow{\text{type}_K} S) \xrightarrow{r} (R \xrightarrow{\text{type}_R} S)$$

in $\mathbf{Alg}(MP_*) \downarrow S$. Zusammen mit (C.569) folgt, dass ein eindeutiger $\mathbf{Alg}(MP_*) \downarrow S$ -Morphismus $u: (H \xrightarrow{\text{type}_H} S) \rightarrow (X \xrightarrow{\text{type}_X} S)$ existiert mit:

$$(\text{C.570}) \quad u \circ n = p \circ u^R$$

$$(\text{C.571}) \quad u \circ g = q \circ u^D$$

(Abb. C.11).

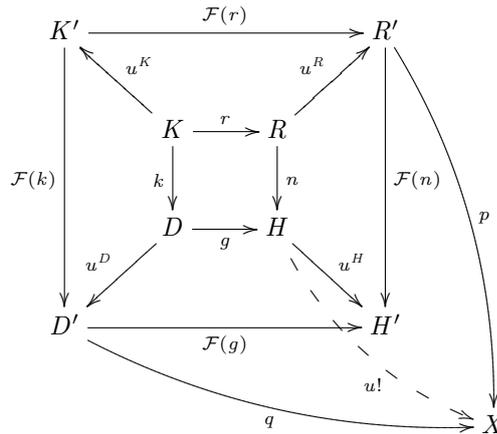


Abb. C.11: Universeller Morphismus u

Nun wird gezeigt, dass $\ker u^H \subseteq \ker u$ gilt. Sei $p \in P_w$ mit $w \in S^*$ ein beliebiges Prädikat und $x \in (H \xrightarrow{\text{type}_H} S)_w$ ein Tupel mit $x \in \ker_p^{u^H}$. Für $x \in p^H \xrightarrow{\text{type}_H} S$ folgt trivialerweise $x \in \ker_p^u$. Gelte also $x \notin p^H \xrightarrow{\text{type}_H} S$. Nach Voraussetzung existiert ein Urbild-Tupel von x unter g oder n . O. B. d. A. gelte:

$$(\text{C.572}) \quad g_s(x') = x$$

$$(\text{C.573}) \quad x' \in \ker_p^{u^D}$$

Daraus folgt:

$$\begin{aligned}
 x' &\in \ker_p^{u^D} && \text{C.573} \\
 \Rightarrow u_w^D(x') &\in \ker_p^q && (q \text{ ist Homomorphismus}) \\
 \Rightarrow g_w(x') &\in \ker_p^u && \text{(C.571)} \\
 \text{(C.574)} \quad \Rightarrow x &\in \ker_p^u && \text{(C.572)}
 \end{aligned}$$

Aus (C.574) folgt $\ker u^H \subseteq \ker u$. Aus Satz A.44 folgt, dass ein eindeutiger $\mathbf{Alg}(\Sigma) \downarrow S$ -Morphismus $u': (H' \xrightarrow{\text{type}_{H'}} S) \rightarrow (X \xrightarrow{\text{type}_X} S)$ existiert mit:

$$\text{(C.575)} \quad u' \circ u^H = u$$

(Abb. C.12).

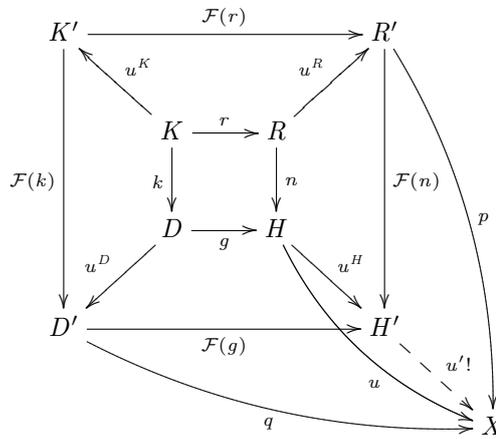


Abb. C.12: Universeller Morphismus u'

Nun gilt zum einen:

$$\begin{aligned}
 u' \circ F(n) \circ u^R &= u' \circ u^H \circ n && (\mathcal{F} \text{ ist freier Funktor}) \\
 &= u \circ n && \text{(C.575)} \\
 &= p \circ u^R && \text{(C.570)} \\
 \text{(C.576)} \quad \Rightarrow u' \circ F(n) &= p && (u^R \text{ ist Epimorphismus})
 \end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
u' \circ \mathcal{F}(g) \circ u^D &= u' \circ u^H \circ g && (\mathcal{F} \text{ ist freier Funktor}) \\
&= u \circ g && (\text{C.575}) \\
&= q \circ u^D && (\text{C.571}) \\
(\text{C.577}) \quad \Rightarrow u' \circ \mathcal{F}(g) &= q && (u^D \text{ ist Epimorphismus})
\end{aligned}$$

Aus (C.576) und (C.577) folgt, dass u' mit den Vergleichsmorphismen p und q verträglich ist. Es bleibt die Eindeutigkeit von u' zu zeigen. Sei ein weiterer $\mathbf{Alg}(\Sigma) \downarrow S$ -Morphismus $u'' : (H' \xrightarrow{\text{type}_{H'}} S) \rightarrow (X \xrightarrow{\text{type}_X} S)$ gegeben mit:

$$\begin{aligned}
(\text{C.578}) \quad u'' \circ \mathcal{F}(n) &= p \\
(\text{C.579}) \quad u'' \circ \mathcal{F}(g) &= q
\end{aligned}$$

Daraus folgt zum einen:

$$\begin{aligned}
u'' \circ u^H \circ n &= u'' \circ \mathcal{F}(n) \circ u^R && (\mathcal{F} \text{ ist freier Funktor}) \\
&= p \circ u^R && (\text{C.578}) \\
&= u' \circ \mathcal{F}(n) \circ u^R && (\text{C.576}) \\
(\text{C.580}) \quad &= u' \circ u^H \circ n && (\mathcal{F} \text{ ist freier Funktor})
\end{aligned}$$

Zum anderen gilt:

$$\begin{aligned}
u'' \circ u^H \circ g &= u'' \circ \mathcal{F}(g) \circ u^D && (\mathcal{F} \text{ ist freier Funktor}) \\
&= q \circ u^D && (\text{C.579}) \\
&= u' \circ \mathcal{F}(g) \circ u^D && (\text{C.577}) \\
(\text{C.581}) \quad &= u' \circ u^H \circ g && (\mathcal{F} \text{ ist freier Funktor})
\end{aligned}$$

Aus der Pushout-Eigenschaft des inneren Diagramms folgt, dass n und g gemeinsam epimorph sind. Aus (C.580) und (C.581) folgt somit $u'' \circ u^H = u' \circ u^H$. Weil u^H Epimorphismus ist, folgt schließlich $u'' = u'$. \square

C.3.9 Beweis von Satz 11.33

Nach Voraussetzung ist die Transformation t konsistent. Somit ist es nach Anmerkung 9.19 erlaubt, $\mathcal{F}^{S'}$ als Einschränkung des Funktors $\mathcal{F}^{M^*} : \mathbf{Alg}(\Sigma) \rightarrow \mathbf{Alg}(M^*)$ anzunehmen. Nach Lemma 11.32 ist lediglich die Kern-Bedingung zu prüfen. Es reicht, die Ausprägungen zu untersuchen. Sei also $p \in P_w$ mit $w \in S^*$ ein beliebiges Prädikat und $x \in H_s$ ein Tupel mit $x \in \ker_p^{u^H}$, wobei u^H der universelle Morphismus der Epireflexion für $H \xrightarrow{\text{type}_H} S'$ ist. Es ist zu zeigen, dass ein Tupel $x' \in (D \xrightarrow{\text{type}_D} S')_w$ mit

$$(C.582) \quad g_w(x') = x$$

$$(C.583) \quad x' \in \ker_p^{u^D}$$

oder ein Tupel $x' \in (R \xrightarrow{\text{type}_R} S')_w$ mit

$$(C.584) \quad n_s(x') = x$$

$$(C.585) \quad x' \in \ker_p^{u^R}$$

existiert. Es gibt nun fünf Fälle zu unterscheiden:

- (1) $x \in p^H$. Das ist in $\mathbf{Alg}(MP_*) \downarrow S'$ nach Konstruktion A.94 äquivalent zu der Existenz eines Elements $y \in H_p$ mit:

$$(C.586) \quad \pi_{p,w}^H(y) = x$$

Weil $(D \xrightarrow{\text{type}_D} S') \xrightarrow{g} (H \xrightarrow{\text{type}_H} S') \xleftarrow{n} (R \xrightarrow{\text{type}_R} S')$ nach Voraussetzung Pushout ist, existiert somit nach Lemma B.26 ein Urbild y' von y unter mindestens einem der beiden Homomorphismen g oder n . O. B. d. A. gelte

$$(C.587) \quad y = g_p(y')$$

für ein Element $y' \in D_p$. Daraus folgt offensichtlich:

$$(C.588) \quad \begin{aligned} & \pi_{p,w}^D(y') \in p^D \\ \Rightarrow & \pi_{p,w}^D(y') \in \ker_p^{u^D} \end{aligned} \quad (u^D \text{ ist Homomorphismus})$$

Weiter folgt:

$$(C.589) \quad \begin{aligned} g_w(\pi_{p,w}^D(y')) &= \pi_{p,w}^H(g_p(y')) && (g \text{ ist Homomorphismus}) \\ &= \pi_{p,w}^H(y) && (C.587) \\ &= x && (C.586) \end{aligned}$$

Aus (C.588) und (C.589) folgt, dass (C.582) und (C.583) mit $x' ::= \pi_{p,w}^D(y')$ erfüllt sind.

- (2) $x \notin p^H$ und $p = =_s$ für $s \in S$. Das Axiom (M.28) ist das einzige M^* -Axiom, das in der Konklusion Elemente zusammenlegt, und zwar zur Sorte O . Es reicht somit aus, sich vollständig auf Elemente zur Sorte N zu beschränken, weil die Identifikation von Elementen in anderen Trägemengen über den Kongruenzabschluss sichergestellt wird, da nach Lemma A.43 der Kern eines Homomorphismus' Kongruenzrelation ist. Gelte somit $s = N$. Nach der Konstruktion der Epireflexion in Satz 9.17 gilt somit:

$$(C.590) \quad x = (\underline{x}, \bar{x})$$

$$(C.591) \quad x \in rel^H$$

$$(C.592) \quad type_{H,N}(\underline{x}) = type_{H,N}(\bar{x})$$

Nach Konstruktion A.94 folgt aus (C.591) die Existenz eines Elements $\hat{x} \in H_{rel}$ mit:

$$(C.593) \quad \pi_{rel,1}^H(\hat{x}) = \underline{x}$$

$$(C.594) \quad \pi_{rel,2}^H(\hat{x}) = \bar{x}$$

Nach Lemma B.26 existiert ein Urbild \hat{x}' von \hat{x} unter mindestens einem der beiden Homomorphismen g und n . O. B. d. A. gelte

$$(C.595) \quad \hat{x} = g_{rel}(\hat{x}')$$

mit $\hat{x}' \in D_{rel}$. Aus (C.595) folgt nach Konstruktion A.94:

$$(C.596) \quad \pi_{rel,w}^H(\hat{x}') \in rel^H$$

Nun gilt zum einen:

$$\begin{aligned} g_N(\pi_{rel,1}^D(\hat{x}')) &= \pi_{rel,1}^H(g_{rel}(\hat{x}')) && (g \text{ ist Homomorphismus}) \\ &= \pi_{rel,1}^H(\hat{x}) && (C.595) \\ (C.597) \quad &= \underline{x} && (C.593) \end{aligned}$$

Zum anderen ergibt sich:

$$\begin{aligned} g_N(\pi_{rel,2}^D(\hat{x}')) &= \pi_{rel,2}^H(g_{rel}(\hat{x}')) && (g \text{ ist Homomorphismus}) \\ &= \pi_{rel,2}^H(\hat{x}) && (C.595) \\ (C.598) \quad &= \bar{x} && (C.594) \end{aligned}$$

Schließlich gilt:

$$\begin{aligned} type_{D,N}(\pi_{rel,1}^D(\hat{x}')) &= type_{H,N}(g_N(\pi_{rel,1}^D(\hat{x}'))) && (g \text{ ist } \mathbf{Alg}(MP_*) \downarrow S' \text{-Morphismus}) \\ &= type_{H,N}(\underline{x}) && (C.597) \\ &= type_{H,N}(\bar{x}) && (C.592) \\ &= type_{H,N}(g_N(\pi_{rel,2}^D(\hat{x}'))) && (C.598) \\ (C.599) \quad &= type_{D,N}(\pi_{rel,2}^D(\hat{x}')) && (g \text{ ist } \mathbf{Alg}(MP_*) \downarrow S' \text{-Morphismus}) \end{aligned}$$

Nach (C.596) und (C.599) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto \pi_{rel,1}^D(\hat{x}') \\ y &\mapsto \pi_{rel,2}^D(\hat{x}') \end{aligned}$$

die Prämisse des M^* -Axioms (M.28) in $D \xrightarrow{type_D} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt:

$$(C.600) \quad u_N^D(\pi_{rel,1}^D(\hat{x}')) = u_N^D(\pi_{rel,2}^D(\hat{x}'))$$

Aus (C.600), (C.597), (C.598) und (C.590) folgt, dass (C.582) und (C.583) mit $x' ::= \pi_{rel,w}^H(\hat{x}')$ erfüllt sind.

- (3) $x \notin p^H$ und $p = software$. Daraus folgt $w = N$. Weil kein M^* -Axiom existiert, das in der Konklusion das Prädikat *software* wahr macht, gibt es nach Konstruktion der Epireflexion in Konstruktion A.92 ein Element $y \in H_N$ mit:

$$(C.601) \quad u_N^H(y) = u_N^H(x)$$

$$(C.602) \quad y \in p^H$$

Das Axiom (M.28) ist das einzige M^* -Axiom, das in der Konklusion Elemente zusammenlegt, und zwar zur Sorte O . Nach der Konstruktion der Epireflexion in Satz 9.17 folgt aus (C.601):

$$(C.603) \quad (x, y) \in rel^H$$

$$(C.604) \quad type_{H,N}(x) = type_{H,N}(y)$$

Aus (C.603) folgt nach Konstruktion A.94 die Existenz eines Elements $\hat{x} \in H_{rel}$ mit:

$$(C.605) \quad \pi_{rel,1}^H(\hat{x}) = x$$

$$(C.606) \quad \pi_{rel,2}^H(\hat{x}) = y$$

Nach Lemma B.26 existiert ein Urbild \hat{x}' von \hat{x} unter mindestens einem der beiden Homomorphismen g und n :

(a) Gelte

$$(C.607) \quad \hat{x} = g_{rel}(\hat{x}')$$

mit $\hat{x}' \in D_{rel}$. Daraus folgt offensichtlich:

$$(C.608) \quad (\pi_{rel,1}^D(\hat{x}'), \pi_{rel,2}^D(\hat{x}')) \in rel^D$$

Nun gilt zum einen:

$$\begin{aligned}
 g_N(\pi_{rel,1}^D(\hat{x}')) &= \pi_{rel,1}^H(g_{rel}(\hat{x}')) && (g \text{ ist Homomorphismus}) \\
 &= \pi_{rel,1}^H(\hat{x}) && \text{(C.607)} \\
 \text{(C.609)} \quad &= x && \text{(C.605)}
 \end{aligned}$$

Zum anderen ergibt sich:

$$\begin{aligned}
 g_N(\pi_{rel,2}^D(\hat{x}')) &= \pi_{rel,2}^H(g_{rel}(\hat{x}')) && (g \text{ ist Homomorphismus}) \\
 &= \pi_{rel,2}^H(\hat{x}) && \text{(C.607)} \\
 \text{(C.610)} \quad &= y && \text{(C.606)}
 \end{aligned}$$

Weil $(L \xrightarrow{type_L} S') \xleftarrow{l} (K \xrightarrow{type_K} S') \xrightarrow{r} (R \xrightarrow{type_R} S')$ nach Voraussetzung eine konsistente Aktion ist, ist r vervollständigend und nach Lemma 10.13 strikt voll. Nach Lemma 9.21 ist auch g strikt voll. Aus (C.610) und (C.602) folgt somit:

$$\begin{aligned}
 \pi_{rel,2}^D(\hat{x}') &\in p^D \\
 \text{(C.611)} \quad &\Rightarrow \pi_{rel,2}^D(\hat{x}') \in \ker u^D && (u^D \text{ ist Homomorphismus})
 \end{aligned}$$

Schließlich gilt:

$$\begin{aligned}
 type_{D,N}(\pi_{rel,1}^D(\hat{x}')) &= type_{H,N}(g_N(\pi_{rel,1}^D(\hat{x}'))) && (g \text{ ist } \mathbf{Alg}(MP_*)\downarrow S'\text{-Morphismus}) \\
 &= type_{H,N}(x) && \text{(C.609)} \\
 &= type_{H,N}(y) && \text{(C.604)} \\
 &= type_{H,N}(g_N(\pi_{rel,2}^D(\hat{x}'))) && \text{(C.610)} \\
 \text{(C.612)} \quad &= type_{D,N}(\pi_{rel,2}^D(\hat{x}')) && (g \text{ ist } \mathbf{Alg}(MP_*)\downarrow S'\text{-Morphismus})
 \end{aligned}$$

Nach (C.608) und (C.612) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto \pi_{rel,1}^D(\hat{x}') \\
 y &\mapsto \pi_{rel,2}^D(\hat{x}')
 \end{aligned}$$

die Prämisse des M^* -Axioms (M.28) in $D \xrightarrow{type_D} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt:

$$\text{(C.613)} \quad u_N^D(\pi_{rel,1}^D(\hat{x}')) = u_N^D(\pi_{rel,2}^D(\hat{x}'))$$

Aus (C.611) und (C.613) folgt schließlich:

$$(C.614) \quad \pi_{rel,1}^D(\hat{x}') \in \ker_p^{u^D}$$

Aus (C.609) und (C.614) folgt, dass (C.582) und (C.583) mit $x' ::= \pi_{rel,1}^D(\hat{x}')$ erfüllt sind.

(b) Gelte

$$(C.615) \quad \hat{x} = n_{rel}(\hat{x}')$$

mit $\hat{x}' \in R_{rel}$. Daraus folgt offensichtlich:

$$(C.616) \quad (\pi_{rel,1}^R(\hat{x}'), \pi_{rel,2}^R(\hat{x}')) \in rel^R$$

Nun gilt zum einen:

$$\begin{aligned} n_N(\pi_{rel,1}^R(\hat{x}')) &= \pi_{rel,1}^H(n_{rel}(\hat{x}')) && (n \text{ ist Homomorphismus}) \\ &= \pi_{rel,1}^H(\hat{x}) && (C.615) \\ (C.617) \quad &= x && (C.605) \end{aligned}$$

Zum anderen ergibt sich:

$$\begin{aligned} n_N(\pi_{rel,2}^R(\hat{x}')) &= \pi_{rel,2}^H(n_{rel}(\hat{x}')) && (n \text{ ist Homomorphismus}) \\ &= \pi_{rel,2}^H(\hat{x}) && (C.615) \\ (C.618) \quad &= y && (C.606) \end{aligned}$$

Weiter gilt:

$$\begin{aligned} type_{R,N}(\pi_{rel,1}^R(\hat{x}')) &= type_{H,N}(n_N(\pi_{rel,1}^R(\hat{x}'))) && (n \text{ ist } \mathbf{Alg}(MP_*) \downarrow S' \text{-Morphismus}) \\ &= type_{H,N}(x) && (C.617) \\ &= type_{H,N}(y) && (C.604) \\ &= type_{H,N}(n_N(\pi_{rel,2}^R(\hat{x}'))) && (C.618) \\ (C.619) \quad &= type_{R,N}(\pi_{rel,2}^R(\hat{x}')) && (n \text{ ist } \mathbf{Alg}(MP_*) \downarrow S' \text{-Morphismus}) \end{aligned}$$

Schließlich folgt aus (C.602) nach Konstruktion A.94 die Existenz eines Elements $\hat{y} \in H_p$ mit:

$$(C.620) \quad \pi_{p,1}^H(\hat{y}) = y$$

Nach Lemma B.26 existiert ein Urbild \hat{y}' von \hat{y} unter mindestens einem der beiden Homomorphismen g und n :

(i) Gelte

$$(C.621) \quad \hat{y} = g_p(\hat{y}')$$

mit $\hat{y}' \in D_p$. Daraus folgt:

$$\begin{aligned} g_N(\pi_{p,1}^D(\hat{y}')) &= \pi_{p,1}^H(g_p(\hat{y}')) && (g \text{ ist Homomorphismus}) \\ &= \pi_{p,1}^H(\hat{y}) && (C.621) \\ (C.622) \quad &= y && (C.620) \end{aligned}$$

Nach Lemma B.29 folgt aus (C.622) und (C.618) die Existenz eines eindeutigen Elements $z \in K_N$ mit:

$$\begin{aligned} (C.623) \quad & r_N(z) = \pi_{rel,2}^R(\hat{x}') \\ (C.624) \quad & k_N(z) = \pi_{p,1}^D(\hat{y}') \end{aligned}$$

Weil $(L \xrightarrow{type_L} S') \xleftarrow{l} (K \xrightarrow{type_K} S') \xrightarrow{r} (R \xrightarrow{type_R} S')$ nach Voraussetzung eine konsistente Aktion ist, ist r vervollständigend. Somit folgt aus (C.616) und (C.623), dass ein Element $z' \in K_N$ existiert mit:

$$\begin{aligned} (C.625) \quad & r_N(z') = \pi_{rel,1}^R(\hat{x}') \\ (C.626) \quad & (z', z) \in rel^K \end{aligned}$$

Aus (C.626) folgt unmittelbar:

$$\begin{aligned} & (k_N(z'), k_N(z)) \in rel^D && (k \text{ ist Homomorphismus}) \\ (C.627) \quad & \Rightarrow (k_N(z'), \pi_{p,1}^D(\hat{y}')) \in rel^D && (C.624) \end{aligned}$$

Weiter gilt:

$$\begin{aligned} g_N(k_N(z')) &= n_N(r_N(z')) && (\text{Pushouts kommutieren}) \\ &= n_N(\pi_{rel,1}^R(\hat{x}')) && (C.625) \\ (C.628) \quad &= x && (C.617) \end{aligned}$$

Schließlich gilt:

$$\begin{aligned} type_{D,N}(k_N(z')) &= type_{H,N}(g_N(k_N(z'))) && (g \text{ ist } \mathbf{Alg}(MP_*) \downarrow S' \text{-Morphismus}) \\ &= type_{H,N}(x) && (C.628) \end{aligned}$$

$$\begin{aligned}
 &= \text{type}_{H,N}(y) && \text{(C.604)} \\
 &= \text{type}_{H,N}(g_N(\pi_{p,1}^D(\hat{y}'))) && \text{(C.622)} \\
 \text{(C.629)} \quad &= \text{type}_{D,N}(\pi_{p,1}^D(\hat{y}')) && (g \text{ ist } \mathbf{Alg}(MP_*)\downarrow S'\text{-Morphismus})
 \end{aligned}$$

Nach (C.627) und (C.629) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto k_N(z') \\
 y &\mapsto \pi_{p,1}^D(\hat{y}')
 \end{aligned}$$

die Prämisse des M^* -Axioms (M.28) in $D \xrightarrow{\text{type}_D} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt:

$$\text{(C.630)} \quad u_N^D(k_N(z')) = u_N^D(\pi_{p,1}^D(\hat{y}'))$$

Aus (C.630) folgt:

$$\text{(C.631)} \quad k_N(z') \in \ker_p^{u^D}$$

Aus (C.628) und (C.631) folgt, dass (C.582) und (C.583) mit $x' ::= k_N(z')$ erfüllt sind.

(ii) Gelte

$$\text{(C.632)} \quad \hat{y} = n_p(\hat{y}')$$

$$\text{(C.633)} \quad \pi_{p,1}^R(\hat{y}') = \pi_{rel,2}^R(\hat{x}')$$

mit $\hat{y}' \in R_p$. Nach (C.616) und (C.619) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto \pi_{rel,1}^R(\hat{x}') \\
 y &\mapsto \pi_{rel,2}^R(\hat{x}')
 \end{aligned}$$

die Prämisse des M^* -Axioms (M.28) in $R \xrightarrow{\text{type}_R} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt:

$$\text{(C.634)} \quad u_N^R(\pi_{rel,1}^R(\hat{x}')) = u_N^R(\pi_{rel,2}^R(\hat{x}'))$$

Aus (C.634) und (C.633) folgt:

$$\text{(C.635)} \quad \pi_{rel,1}^R(\hat{x}') \in \ker_p^{u^R}$$

Aus (C.617) und (C.635) folgt, dass (C.584) und (C.585) mit $x' ::= \pi_{rel,1}^R(\hat{x}')$ erfüllt sind.

(iii) Gelte

$$(C.636) \quad \hat{y} = n_p(\hat{y}')$$

$$(C.637) \quad \pi_{p,1}^R(\hat{y}') \neq \pi_{rel,2}^R(\hat{x}')$$

mit $\hat{y}' \in R_p$. Nun gilt:

$$n_N(\pi_{p,1}^R(\hat{y}')) = \pi_{p,1}^H(n_p(\hat{y}')) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{p,1}^H(\hat{y}) \quad (C.636)$$

$$= y \quad (C.620)$$

$$(C.638) \quad = n_N(\pi_{rel,2}^R(\hat{x}')) \quad (C.618)$$

Weil $(L \xrightarrow{type_L} S') \xleftarrow{l} (K \xrightarrow{type_K} S') \xrightarrow{r} (R \xrightarrow{type_R} S')$ nach Voraussetzung eine konsistente Aktion ist, ist r vervollständigend. Nach Lemma 10.13 und Lemma A.24 ist r injektiv. Nach Lemma B.27 folgt somit aus (C.637) und (C.638), dass zwei Elemente $y'', \bar{z} \in K_N$ existieren mit:

$$(C.639) \quad r_N(y'') = \pi_{p,1}^R(\hat{y}')$$

$$(C.640) \quad r_N(\bar{z}) = \pi_{rel,2}^R(\hat{x}')$$

Weil r strikt voll ist, folgt aus (C.639):

$$(C.641) \quad y'' \in p^K$$

$$(C.642) \quad \Rightarrow k_N(y'') \in p^D \quad (k \text{ ist Homomorphismus})$$

Weil r vervollständigend ist, folgt aus (C.640) und (C.616), dass ein Element $\underline{z} \in K_N$ existiert mit:

$$(C.643) \quad r_N(\underline{z}) = \pi_{rel,1}^R(\hat{x}')$$

$$(C.644) \quad (\underline{z}, \bar{z}) \in rel^K$$

Aus (C.644) folgt, da k Homomorphismus ist:

$$(C.645) \quad (k_N(\underline{z}), k_N(\bar{z})) \in rel^D$$

Weiter gilt:

$$\begin{aligned}
 \text{type}_{D,N}(k_N(\underline{z})) &= \text{type}_{K,N}(\underline{z}) && (k \text{ ist } \mathbf{Alg}(MP_*)\downarrow S'\text{-Morphismus}) \\
 &= \text{type}_{R,N}(r_N(\underline{z})) && (r \text{ ist } \mathbf{Alg}(MP_*)\downarrow S'\text{-Morphismus}) \\
 &= \text{type}_{R,N}(\pi_{rel,1}^R(\hat{x}')) && \text{(C.643)} \\
 &= \text{type}_{R,N}(\pi_{rel,2}^R(\hat{x}')) && \text{(C.619)} \\
 &= \text{type}_{R,N}(r_N(\bar{z})) && \text{(C.640)} \\
 &= \text{type}_{K,N}(\bar{z}) && (r \text{ ist } \mathbf{Alg}(MP_*)\downarrow S'\text{-Morphismus}) \\
 \text{(C.646)} \quad &= \text{type}_{D,N}(k_N(\bar{z})) && (k \text{ ist } \mathbf{Alg}(MP_*)\downarrow S'\text{-Morphismus})
 \end{aligned}$$

Nach (C.645) und (C.646) erfüllt die Variablenbelegung

$$\begin{aligned}
 x &\mapsto k_N(\underline{z}) \\
 y &\mapsto k_N(\bar{z})
 \end{aligned}$$

die Prämisse des M^* -Axioms (M.28) in $D \xrightarrow{\text{type}_D} S'$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt:

$$\text{(C.647)} \quad u_N^D(k_N(\underline{z})) = u_N^D(k_N(\bar{z}))$$

Weiter folgt:

$$\begin{aligned}
 g_N(k_N(y'')) &= n_N(r_N(y'')) && \text{(Pushouts kommutieren)} \\
 &= n_N(\pi_{p,1}^R(\hat{y}')) && \text{(C.639)} \\
 &= n_N(\pi_{rel,2}^R(\hat{x}')) && \text{(C.638)} \\
 &= n_N(r_N(\bar{z})) && \text{(C.640)} \\
 \text{(C.648)} \quad &= g_N(k_N(\bar{z})) && \text{(Pushouts kommutieren)}
 \end{aligned}$$

Weil $(L \xrightarrow{\text{type}_L} S') \xleftarrow{l} (K \xrightarrow{\text{type}_K} S') \xrightarrow{r} (R \xrightarrow{\text{type}_R} S')$ nach Voraussetzung eine konsistente Aktion ist, ist r vervollständigend und nach Lemma 10.13 strikt voll. Nach Lemma 9.21 ist auch g strikt voll. Aus Lemma A.24 folgt, dass g injektiv ist. Somit ergibt sich aus (C.648):

$$\text{(C.649)} \quad k_N(y'') = k_N(\bar{z})$$

Aus (C.649) und (C.642) folgt:

$$\text{(C.650)} \quad k_N(\bar{z}) \in p^D$$

Aus (C.650) und (C.647) folgt weiter:

$$(C.651) \quad k_N(\underline{z}) \in \ker_p^{u^D}$$

Schließlich gilt:

$$\begin{aligned} g_N(k_N(\underline{z})) &= n_N(r_N(\underline{z})) && \text{(Pushouts kommutieren)} \\ &= n_N(\pi_{rel,1}^R(\hat{x}')) && (C.643) \\ (C.652) \quad &= x && (C.617) \end{aligned}$$

Aus (C.652) und (C.651) folgt, dass (C.582) und (C.583) mit $x' ::= k_N(\underline{z})$ erfüllt sind.

- (4) $x \notin p^H$ und $p = rel$. Offensichtlich gilt $x = (\underline{x}, \bar{x})$ mit $\underline{x}, \bar{x} \in N$. Weil kein M^* -Axiom existiert, das in der Konklusion das Prädikat $relO$ wahr macht, bleibt der Kongruenzabschluss der Gleichheits-Prädikate zu untersuchen. Nach der Konstruktion der Epireflexion in Satz 9.17 existieren somit zwei Elemente $\underline{y}, \bar{y} \in H_N$ mit:

$$(C.653) \quad (\underline{y}, \underline{x}) \in rel^H$$

$$(C.654) \quad type_{H,N}(\underline{y}) = type_{H,N}(\underline{x})$$

$$(C.655) \quad (\bar{y}, \bar{x}) \in rel^H$$

$$(C.656) \quad type_{H,N}(\bar{y}) = type_{H,N}(\bar{x})$$

$$(C.657) \quad (\underline{y}, \bar{y}) \in rel^H$$

Nach Voraussetzung ist $H \xrightarrow{type_H} S'$ ein M' -System, somit sind die M' -Axiome (M.24) und (M.25) gültig in $H \xrightarrow{type_H} S'$. Nach (C.653) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto \underline{y} \\ y &\mapsto \underline{x} \end{aligned}$$

die Prämisse des M^* -Axioms (M.24) in $H \xrightarrow{type_H} S'$. Aus der Gültigkeit dieses Axioms in $H \xrightarrow{type_H} S'$ folgt die Wahrheit der Konklusion:

$$(C.658) \quad (\underline{x}, \underline{y}) \in rel^H$$

Nach (C.658) und (C.657) erfüllt die Variablenbelegung

$$x \mapsto \underline{x}$$

$$y \mapsto \underline{y}$$

$$z \mapsto \bar{y}$$

die Prämisse des M^* -Axioms (M.25) in $H \xrightarrow{\text{type}_H} S'$. Aus der Gültigkeit dieses Axioms in $H \xrightarrow{\text{type}_H} S'$ folgt die Wahrheit der Konklusion:

$$(C.659) \quad (\underline{x}, \bar{y}) \in \text{rel}^H$$

Nach (C.659) und (C.655) erfüllt die Variablenbelegung

$$x \mapsto \underline{x}$$

$$y \mapsto \bar{y}$$

$$z \mapsto \bar{x}$$

die Prämisse des M^* -Axioms (M.25) in $H \xrightarrow{\text{type}_H} S'$. Aus der Gültigkeit dieses Axioms in $H \xrightarrow{\text{type}_H} S'$ folgt die Wahrheit der Konklusion:

$$(C.660) \quad (\underline{x}, \bar{x}) \in \text{rel}^H$$

Das ist jedoch ein Widerspruch zu der ursprünglichen Annahme $x \notin \text{rel}^H$.

(5) $x \notin p^H$ und $p = \text{under}$. Offensichtlich gilt

$$(C.661) \quad x = (\underline{x}, \bar{x})$$

mit $\underline{x}, \bar{x} \in N$. Aus Satz 9.17 folgt, dass die Elemente \underline{x} und \bar{x} über einen $(n+2)$ -elementigen

$$\underline{x} \longrightarrow \triangleright x_1 \rightsquigarrow x_2 \longrightarrow \triangleright x_3 \rightsquigarrow x_4 \longrightarrow \triangleright x_5 \rightsquigarrow \dots \rightsquigarrow x_{2n} \longrightarrow \triangleright \bar{x}$$

Abb. C.13: Beziehung zwischen \underline{x} und \bar{x} in H

erweiterten Pfad ($n \in \mathbb{N}_0$) miteinander verbunden sind (Abb. C.13), wobei die Elemente jedes Paares auch zusammenfallen können. Nach Konstruktion A.94 entsprechen die Beziehungen Elementen $\hat{x}_{2i} \in H_{\text{under}}$ mit $0 \leq i \leq n$ und Elementen $\hat{x}_{2i+1} \in H_{\text{rel}}$ mit $0 \leq i < n$. Es gelte:

$$(C.662) \quad \begin{aligned} \underline{x} &= \pi_{\text{under},1}^H(\hat{x}_0) \\ x_1 &= \pi_{\text{under},2}^H(\hat{x}_0) = \pi_{\text{rel},1}^H(\hat{x}_1) \\ x_2 &= \pi_{\text{rel},2}^H(\hat{x}_1) = \pi_{\text{under},1}^H(\hat{x}_2) \\ x_3 &= \pi_{\text{under},2}^H(\hat{x}_2) = \pi_{\text{rel},1}^H(\hat{x}_3) \\ &\dots \end{aligned}$$

$$(C.663) \quad \begin{aligned} x_{2n} &= \pi_{rel,2}^H(\hat{x}_{2n-1}) = \pi_{under,1}^H(\hat{x}_{2n}) \\ \bar{x} &= \pi_{under,2}^H(\hat{x}_{2n}) \end{aligned}$$

und allgemein

$$(C.664) \quad x_{2i} = \pi_{rel,2}^H(\hat{x}_{2i-1}) = \pi_{under,1}^H(\hat{x}_{2i}) \quad \text{für alle } 0 \leq i \leq n$$

$$(C.665) \quad x_{2i+1} = \pi_{under,2}^H(\hat{x}_{2i}) = \pi_{rel,1}^H(\hat{x}_{2i+1}) \quad \text{für alle } 0 \leq i < n$$

Weil $(D \xrightarrow{type_D} S') \xrightarrow{g} (H \xrightarrow{type_H} S') \xleftarrow{n} (R \xrightarrow{type_R} S')$ nach Voraussetzung Pushout ist, existiert nach Lemma B.26 für jedes Element \hat{x}_i mit $0 \leq i \leq 2n$ ein Urbild \hat{x}'_i unter mindestens einem der beiden Homomorphismen g und n :

(a) Gelte

$$(C.666) \quad \hat{x}_i \in \text{Im } g_{under} \cup \text{Im } g_{rel}$$

für ein beliebiges i mit $0 \leq i \leq 2n$. Mit (C.666) folgt, dass jedes \hat{x}_i mit $0 \leq i \leq 2n$ ein Urbild \hat{x}'_i unter g hat. Es gilt somit:

$$(C.667) \quad \hat{x}_{2i} = g_{under}(\hat{x}'_{2i}) \quad \text{für alle } 0 \leq i \leq n$$

$$(C.668) \quad \hat{x}_{2i+1} = g_{rel}(\hat{x}'_{2i+1}) \quad \text{für alle } 0 \leq i < n$$

Weil $(L \xrightarrow{type_L} S') \xleftarrow{l} (K \xrightarrow{type_K} S') \xrightarrow{r} (R \xrightarrow{type_R} S')$ nach Voraussetzung eine konsistente Aktion ist, ist r vervollständigend und nach Lemma 10.13 strikt voll. Nach Lemma 9.21 ist auch g strikt voll. Weiter folgt nach Lemma A.24, dass g injektiv ist. Zusammen mit (C.664), (C.665), (C.667) und (C.668) folgt:

$$(C.669) \quad \pi_{under,1}^D(\hat{x}'_{2i}) = \pi_{rel,2}^D(\hat{x}'_{2i-1}) \quad \text{für alle } 0 < i \leq n$$

$$(C.670) \quad \pi_{under,2}^D(\hat{x}'_{2i}) = \pi_{rel,1}^D(\hat{x}'_{2i+1}) \quad \text{für alle } 0 \leq i < n$$

Aus (C.669) und (C.670) folgt, dass für $\pi_{under,1}^D(\hat{x}'_0)$ und $\pi_{under,2}^D(\hat{x}'_{2n})$ in D dieselbe Situation für die Epireflexion wie für \underline{x} und \bar{x} in H existiert. Daraus folgt:

$$(C.671) \quad (\pi_{under,1}^D(\hat{x}'_0), \pi_{under,2}^D(\hat{x}'_{2n})) \in \ker_{under}^{u^D}$$

Nun gilt zum einen:

$$\begin{aligned}
g_N(\pi_{\text{under},1}^D(\hat{x}'_0)) &= \pi_{\text{under},1}^H(g_{\text{under}}(\hat{x}'_0)) && (g \text{ ist Homomorphismus}) \\
&= \pi_{\text{under},1}^H(\hat{x}_0) && \text{(C.667)} \\
\text{(C.672)} \quad &= \underline{x} && \text{(C.662)}
\end{aligned}$$

Zum anderen ergibt sich:

$$\begin{aligned}
g_N(\pi_{\text{under},2}^D(\hat{x}'_{2n})) &= \pi_{\text{under},2}^H(g_{\text{under}}(\hat{x}'_{2n})) && (g \text{ ist Homomorphismus}) \\
&= \pi_{\text{under},2}^H(\hat{x}_{2n}) && \text{(C.667)} \\
\text{(C.673)} \quad &= \bar{x} && \text{(C.663)}
\end{aligned}$$

Aus (C.671), (C.672), (C.673) und (C.661) folgt, dass (C.582) und (C.583) mit

$$x' ::= (\pi_{\text{under},1}^D(\hat{x}'_0), \pi_{\text{under},2}^D(\hat{x}'_{2n}))$$

erfüllt sind.

(b) Gelte

$$\text{(C.674)} \quad \hat{x}_{2i} = n_{\text{under}}(\hat{x}'_{2i}) \quad \text{für alle } 0 \leq i \leq n$$

$$\text{(C.675)} \quad \hat{x}_{2i+1} = n_{\text{rel}}(\hat{x}'_{2i+1}) \quad \text{für alle } 0 \leq i < n$$

und zusätzlich

$$\text{(C.676)} \quad \pi_{\text{rel},2}^R(\hat{x}'_{2i-1}) = \pi_{\text{under},1}^R(\hat{x}'_{2i}) \quad \text{für alle } 0 < i \leq n$$

$$\text{(C.677)} \quad \pi_{\text{under},2}^R(\hat{x}'_{2i}) = \pi_{\text{rel},1}^R(\hat{x}'_{2i+1}) \quad \text{für alle } 0 \leq i < n$$

Dann ergibt sich analog zu Punkt 5a, dass (C.584) und (C.585) mit

$$x' ::= (\pi_{\text{under},1}^R(\hat{x}'_0), \pi_{\text{under},2}^R(\hat{x}'_{2n}))$$

erfüllt sind, wobei \hat{x}'_0 und \hat{x}'_{2n} die jeweiligen Urbilder von \hat{x}_0 und \hat{x}_{2n} unter n sind.

(c) Gelte

$$\text{(C.678)} \quad \hat{x}_{2i} = n_{\text{under}}(\hat{x}'_{2i}) \quad \text{für alle } 0 \leq i \leq n$$

$$\text{(C.679)} \quad \hat{x}_{2i+1} = n_{\text{rel}}(\hat{x}'_{2i+1}) \quad \text{für alle } 0 \leq i < n$$

und

$$\text{(C.680)} \quad \pi_{\text{rel},2}^R(\hat{x}'_{2i-1}) \neq \pi_{\text{under},1}^R(\hat{x}'_{2i})$$

für mindestens ein i mit $0 < i \leq n$ oder

$$(C.681) \quad \pi_{\text{under},2}^R(\hat{x}'_{2i}) \neq \pi_{\text{rel},1}^R(\hat{x}'_{2i+1})$$

für mindestens ein i mit $0 \leq i < n$. Nun gilt zum einen:

$$n_N(\pi_{\text{rel},2}^R(\hat{x}'_{2i-1})) = \pi_{\text{rel},2}^H(n_{\text{rel}}(\hat{x}'_{2i-1})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{\text{rel},2}^H(\hat{x}_{2i-1}) \quad (C.679)$$

$$= \pi_{\text{under},1}^H(\hat{x}_{2i}) \quad (C.664)$$

$$= \pi_{\text{under},1}^H(n_{\text{under}}(\hat{x}'_{2i})) \quad (C.678)$$

$$(C.682) \quad = n_N(\pi_{\text{under},1}^R(\hat{x}'_{2i})) \quad (n \text{ ist Homomorphismus})$$

Zum anderen ergibt sich:

$$n_N(\pi_{\text{under},2}^R(\hat{x}'_{2i})) = \pi_{\text{under},2}^H(n_{\text{under}}(\hat{x}'_{2i})) \quad (n \text{ ist Homomorphismus})$$

$$= \pi_{\text{under},2}^H(\hat{x}_{2i}) \quad (C.674)$$

$$= \pi_{\text{rel},1}^H(\hat{x}_{2i+1}) \quad (C.665)$$

$$= \pi_{\text{rel},1}^H(n_{\text{rel}}(\hat{x}'_{2i+1})) \quad (C.675)$$

$$(C.683) \quad = n_N(\pi_{\text{rel},1}^R(\hat{x}'_{2i+1})) \quad (n \text{ ist Homomorphismus})$$

Nach Lemma B.27 folgt aus (C.680) und (C.682) bzw. aus (C.681) und (C.683), dass die Elemente $\pi_{\text{rel},2}^R(\hat{x}'_{2i-1})$ und $\pi_{\text{under},1}^R(\hat{x}'_{2i})$ bzw. $\pi_{\text{under},2}^R(\hat{x}'_{2i})$ und $\pi_{\text{rel},1}^R(\hat{x}'_{2i+1})$ Urbilder unter r besitzen. Weil $(L \xrightarrow{\text{type}_L} S') \xleftarrow{l} (K \xrightarrow{\text{type}_K} S') \xrightarrow{r} (R \xrightarrow{\text{type}_R} S')$ nach Voraussetzung eine konsistente Aktion ist, ist r vervollständigend. Daraus folgt, dass jedes \hat{x}'_i mit $0 \leq i \leq 2n$ ein Urbild \hat{x}''_i unter r hat. Es gilt somit:

$$(C.684) \quad \hat{x}'_{2i} = r_{\text{under}}(\hat{x}''_{2i}) \quad \text{für alle } 0 \leq i \leq n$$

$$(C.685) \quad \hat{x}'_{2i+1} = r_{\text{rel}}(\hat{x}''_{2i+1}) \quad \text{für alle } 0 \leq i < n$$

Nun gilt zum einen für alle $0 < i \leq n$:

$$g_N(\pi_{\text{rel},2}^K(k_{\text{rel}}(\hat{x}''_{2i-1}))) = \pi_{\text{rel},2}^H(g_{\text{rel}}(k_{\text{rel}}(\hat{x}''_{2i-1}))) \quad (g \text{ ist Homomorphismus})$$

$$= \pi_{\text{rel},2}^H(n_{\text{rel}}(r_{\text{rel}}(\hat{x}''_{2i-1}))) \quad (\text{Pushouts kommutieren})$$

$$= \pi_{\text{rel},2}^H(n_{\text{rel}}(\hat{x}'_{2i-1})) \quad (C.685)$$

$$= n_N(\pi_{\text{rel},2}^R(\hat{x}'_{2i-1})) \quad (n \text{ ist Homomorphismus})$$

$$= n_N(\pi_{\text{under},1}^R(\hat{x}'_{2i})) \quad (C.682)$$

$$= \pi_{\text{under},1}^H(n_{\text{under}}(\hat{x}'_{2i})) \quad (n \text{ ist Homomorphismus})$$

$$\begin{aligned}
&= \pi_{\text{under},1}^H(n_{\text{under}}(r_{\text{under}}(\hat{x}_{2i}''))) && \text{(C.684)} \\
&= \pi_{\text{under},1}^H(g_{\text{under}}(k_{\text{under}}(\hat{x}_{2i}''))) && \text{(Pushouts kommutieren)} \\
\text{(C.686)} \quad &= g_N(\pi_{\text{under},1}^K(k_{\text{under}}(\hat{x}_{2i}''))) && \text{(}g\text{ ist Homomorphismus)}
\end{aligned}$$

Zum anderen ergibt sich für alle $0 \leq i < n$:

$$\begin{aligned}
g_N(\pi_{\text{under},2}^K(k_{\text{under}}(\hat{x}_{2i}''))) &= \pi_{\text{under},2}^H(g_{\text{under}}(k_{\text{under}}(\hat{x}_{2i}''))) && \text{(}g\text{ ist Homomorphismus)} \\
&= \pi_{\text{under},2}^H(n_{\text{under}}(r_{\text{under}}(\hat{x}_{2i}''))) && \text{(Pushouts kommutieren)} \\
&= \pi_{\text{under},2}^H(n_{\text{under}}(\hat{x}_{2i}')) && \text{(C.684)} \\
&= n_N(\pi_{\text{under},2}^R(\hat{x}_{2i}')) && \text{(}n\text{ ist Homomorphismus)} \\
&= n_N(\pi_{\text{rel},1}^R(\hat{x}_{2i+1}')) && \text{(C.683)} \\
&= \pi_{\text{rel},1}^H(n_{\text{rel}}(\hat{x}_{2i+1}')) && \text{(}n\text{ ist Homomorphismus)} \\
&= \pi_{\text{rel},1}^H(n_{\text{rel}}(r_{\text{rel}}(\hat{x}_{2i+1}''))) && \text{(C.685)} \\
&= \pi_{\text{rel},1}^H(g_{\text{rel}}(k_{\text{rel}}(\hat{x}_{2i+1}''))) && \text{(Pushouts kommutieren)} \\
\text{(C.687)} \quad &= g_N(\pi_{\text{rel},1}^K(k_{\text{rel}}(\hat{x}_{2i+1}''))) && \text{(}g\text{ ist Homomorphismus)}
\end{aligned}$$

Weil $(L \xrightarrow{\text{type}_L} S') \xleftarrow{l} (K \xrightarrow{\text{type}_K} S') \xrightarrow{r} (R \xrightarrow{\text{type}_R} S')$ nach Voraussetzung eine konsistente Aktion ist, ist r vervollständigend und nach Lemma 10.13 strikt voll. Nach Lemma 9.21 ist auch g strikt voll. Aus Lemma A.24 folgt, dass g injektiv ist. Somit ergibt sich aus (C.686) und (C.687):

$$\text{(C.688)} \quad \pi_{\text{rel},2}^K(k_{\text{rel}}(\hat{x}_{2i-1}'')) = \pi_{\text{under},1}^K(k_{\text{under}}(\hat{x}_{2i}'')) \quad \text{für alle } 0 < i \leq n$$

$$\text{(C.689)} \quad \pi_{\text{under},2}^K(k_{\text{under}}(\hat{x}_{2i}'')) = \pi_{\text{rel},1}^K(k_{\text{rel}}(\hat{x}_{2i+1}'')) \quad \text{für alle } 0 \leq i < n$$

Aus (C.688) und (C.689) folgt, dass für $\pi_{\text{under},1}^D(k_{\text{under}}(\hat{x}_0''))$ und $\pi_{\text{under},2}^D(k_{\text{under}}(\hat{x}_{2n}''))$ in D dieselbe Situation für die Epireflexion wie für \underline{x} und \bar{x} in H existiert. Daraus folgt:

$$\text{(C.690)} \quad (\pi_{\text{under},1}^D(k_{\text{under}}(\hat{x}_0'')), \pi_{\text{under},2}^D(k_{\text{under}}(\hat{x}_{2n}''))) \in \ker_{\text{under}}^{u^D}$$

Nun gilt zum einen:

$$\begin{aligned}
g_N(\pi_{\text{under},1}^D(k_{\text{under}}(\hat{x}_0''))) &= \pi_{\text{under},1}^H(g_{\text{under}}(k_{\text{under}}(\hat{x}_0''))) && \text{(}g\text{ ist Homomorphismus)} \\
&= \pi_{\text{under},1}^H(n_{\text{under}}(r_{\text{under}}(\hat{x}_0''))) && \text{(Pushouts kommutieren)} \\
&= \pi_{\text{under},1}^H(n_{\text{under}}(\hat{x}_0')) && \text{(C.684)} \\
&= \pi_{\text{under},1}^H(\hat{x}_0) && \text{(C.678)} \\
\text{(C.691)} \quad &= \underline{x} && \text{(C.662)}
\end{aligned}$$

Zum anderen ergibt sich:

$$\begin{aligned}
 g_N(\pi_{\text{under},2}^D(k_{\text{under}}(\hat{x}_{2n}''))) &= \pi_{\text{under},2}^H(g_{\text{under}}(k_{\text{under}}(\hat{x}_{2n}''))) && (g \text{ ist Homomorphismus}) \\
 &= \pi_{\text{under},2}^H(n_{\text{under}}(r_{\text{under}}(\hat{x}_{2n}''))) && (\text{Pushouts kommutieren}) \\
 &= \pi_{\text{under},2}^H(n_{\text{under}}(\hat{x}_{2n}')) && (\text{C.684}) \\
 &= \pi_{\text{under},2}^H(\hat{x}_{2n}) && (\text{C.678}) \\
 (\text{C.692}) \qquad \qquad \qquad &= \bar{x} && (\text{C.663})
 \end{aligned}$$

Aus (C.690), (C.691), (C.692) und (C.661) folgt, dass (C.582) und (C.583) mit

$$x' ::= (\pi_{\text{under},1}^D(k_{\text{under}}(\hat{x}_0'')), \pi_{\text{under},2}^D(k_{\text{under}}(\hat{x}_{2n}'')))$$

erfüllt sind. □

C.3.10 Beweis von Lemma 11.42

Die gewünschte Eigenschaft wird mit Hilfe von Induktion über der Länge des Pfades in B gezeigt:

- *Induktionsbeginn* $n = 1$. Nach (11.2) ist (x, z) ein zweielementiger Pfad, und es gilt:

$$y_1 = f_N(x) = f_N(x_1)$$

Somit ist (11.3) erfüllt.

- *Induktionsbeginn* $n = 2$. Nach (11.2) ist (x, z) ein zweielementiger Pfad, und es gilt:

$$y_1 = f_N(x) = f_N(x_1)$$

$$y_2 = f_N(z) = f_N(x_2)$$

Somit ist (11.3) erfüllt.

- *Induktionsvoraussetzung.* Gelte (11.3) für Pfade der Länge n mit $n \geq 2$.
- *Induktionsschritt* $n \rightarrow n + 1$. Der $n + 1$ -elementige Pfad lässt sich in einen n -elementigen Pfad (y_1, y_2, \dots, y_n) und einen zweielementigen Pfad (y_n, y_{n+1}) aufteilen. Nach Voraussetzung gilt $y_1 ::= f_N(x)$ und $y_{n+1} ::= f_N(z)$. Weil f nach Voraussetzung abkürzende Pfade zurückzieht, folgt aus (11.2), dass ein Element $x' \in A_N$ existiert mit:

$$(\text{C.693}) \qquad \qquad \qquad f_N(x') = y_n$$

$$(\text{C.694}) \qquad \qquad \qquad (x, x') \in \text{under}^A$$

$$(\text{C.695}) \qquad \qquad \qquad (x', z) \in \text{under}^A$$

Nach Induktionsvoraussetzung gilt (11.3) für die Pfade (y_1, y_2, \dots, y_n) und (y_n, y_{n+1}) in B . Somit existieren ein A -Pfad (x_1, x_2, \dots, x_m) mit

$$(C.696) \quad \forall y_i \exists x_j : y_i = f_N(x_j)$$

für $1 \leq i \leq n$ und $1 \leq j \leq m$ sowie ein A -Pfad $(x_m, x_{m+1}, \dots, x_{m'})$ mit

$$(C.697) \quad \forall y_i \exists x_j : y_i = f_N(x_j)$$

für $n \leq i \leq n+1$ und $m \leq j \leq m'$. Aus (C.696) und (C.697) folgt die Gültigkeit von (11.3) für $n+1$ und somit für alle $n \in \mathbb{N}^+$. \square

C.3.11 Beweis von Lemma 11.44

Seien zwei Elemente $x, z \in P_N^S$ und ein Element $y' \in S_N^\#$ gegeben mit:

$$(C.698) \quad (x, z) \in \text{under}^{P^S}$$

$$(C.699) \quad (\bar{l}_N^{t_2}(x), y') \in \text{under}^{S^\#}$$

$$(C.700) \quad (y', \bar{l}_N^{t_2}(z)) \in \text{under}^{S^\#}$$

Aus (C.699) und (C.700) folgt, weil r^{t_1} und \bar{r}^{t_1} Homomorphismen sind:

$$(C.701) \quad (\bar{r}_N^{t_1}(x), \bar{r}_N^{t_1}(z)) \in \text{under}^{S^{\#\#}}$$

$$(C.702) \quad (r_N^{t_1}(\bar{l}_N^{t_2}(x)), r_N^{t_1}(y')) \in \text{under}^{S'}$$

$$(C.703) \quad (r_N^{t_1}(y'), r_N^{t_1}(\bar{l}_N^{t_2}(z))) \in \text{under}^{S'}$$

Weil Pullbacks kommutieren, folgt aus (C.702) und (C.703):

$$(C.704) \quad (l_N^{t_2}(\bar{r}_N^{t_1}(x)), r_N^{t_1}(y')) \in \text{under}^{S'}$$

$$(C.705) \quad (r_N^{t_1}(y'), l_N^{t_2}(\bar{r}_N^{t_1}(z))) \in \text{under}^{S'}$$

Weil l^{t_2} nach Voraussetzung abkürzende Pfade zurückzieht, folgt aus (C.701), (C.704) und (C.705), dass ein $\hat{y} \in S_N^{\#\#}$ existiert mit:

$$(C.706) \quad l_N^{t_2}(\hat{y}) = r_N^{t_1}(y')$$

$$(C.707) \quad (\bar{r}_N^{t_1}(x), \hat{y}) \in \text{under}^{S^{\#\#}}$$

$$(C.708) \quad (\hat{y}, \bar{r}_N^{t_1}(z)) \in \text{under}^{S^{\#\#}}$$

Aus (C.706) folgt nach Korollar A.86, dass ein eindeutiges $y \in P_N^S$ existiert mit:

$$(C.709) \quad \bar{l}_N^{t_2}(y) = y'$$

$$(C.710) \quad \bar{r}_N^{t_1}(y) = \hat{y}$$

Aus (C.709), (C.699) und (C.700) folgt:

$$(C.711) \quad (\bar{l}_N^{t_2}(x), \bar{l}_N^{t_2}(y)) \in \text{under}^{S^\#}$$

$$(C.712) \quad (\bar{l}_N^{t_2}(y), \bar{l}_N^{t_2}(z)) \in \text{under}^{S^\#}$$

Aus (C.710), (C.707) und (C.708) folgt:

$$(C.713) \quad (\bar{r}_N^{t_1}(x), \bar{r}_N^{t_1}(y)) \in \text{under}^{S^{\#\#}}$$

$$(C.714) \quad (\bar{r}_N^{t_1}(y), \bar{r}_N^{t_1}(z)) \in \text{under}^{S^{\#\#}}$$

Weil die Pullback-Morphismen nach Korollar A.87 gemeinsam strikt voll sind, folgt zum einen aus (C.711) und (C.713) und zum anderen aus (C.712) und (C.714):

$$(C.715) \quad (x, y) \in \text{under}^{P^S}$$

$$(C.716) \quad (y, z) \in \text{under}^{P^S}$$

Schließlich folgt aus (C.709), (C.715) und (C.716), dass \bar{l}^{t_2} abkürzende Pfade zurückzieht. \square

C.3.12 Beweis von Lemma 11.47

Nach Konstruktion ist $I^\# \xleftarrow{l''^{t_2}} P \xrightarrow{t} S^{\#\#}$ Pullback von $I^\# \xrightarrow{\text{type}_{I^\#}} S' \xleftarrow{l'^2} S^{\#\#}$. Somit gilt:

$$(C.717) \quad \text{type}_{I^\#} \circ l''^{t_2} = l'^2 \circ t$$

Weil \mathcal{F}^{M^*} freier Funktor ist und Funktoren kommutative Diagramme übertragen, folgt aus (C.717):

$$(C.718) \quad \begin{aligned} \mathcal{F}^{M^*}(\text{type}_{I^\#}) \circ \mathcal{F}^{M^*}(l''^{t_2}) &= \mathcal{F}^{M^*}(l'^2) \circ \mathcal{F}^{M^*}(t) \\ \Rightarrow \text{type}_{I'} \circ \mathcal{F}^{M^*}(l''^{t_2}) &= l'^2 \circ \mathcal{F}^{M^*}(t) \end{aligned}$$

Aus (C.718) und der Pullback-Eigenschaft von $I' \xleftarrow{l'^2} I^{\#\#} \xrightarrow{\text{type}_{I^{\#\#}}} S^{\#\#}$ folgt, dass ein eindeutiger MP -Homomorphismus $i: \mathcal{F}^{M^*}(P) \rightarrow I^{\#\#}$ existiert (Abb. C.14) mit:

$$(C.719) \quad l'^2 \circ i = \mathcal{F}^{M^*}(l''^{t_2})$$

$$(C.720) \quad \text{type}_{I^{\#\#}} \circ i = \mathcal{F}^{M^*}(t)$$

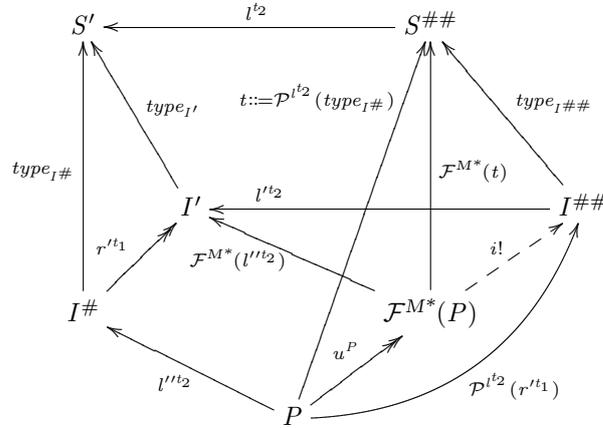


Abb. C.14: Universeller Morphismus i

Aus (C.720) folgt, dass i ein $\mathbf{Alg}(MP) \downarrow S^{##}$ -Morphismus ist.

Nun gilt:

$$\begin{aligned}
 l'^{t_2} \circ i \circ u^P &= \mathcal{F}^{M^*}(l''^{t_2}) \circ u^P && (C.719) \\
 &= r'^{t_1} \circ l''^{t_2} && (\mathcal{F}^{M^*} \text{ ist Funktor}) \\
 (C.721) \quad &= l'^{t_2} \circ \mathcal{P}^{l'^{t_2}}(r'^{t_1}) && (\mathcal{P}^{l'^{t_2}} \text{ ist Funktor})
 \end{aligned}$$

Weil $\mathcal{P}^{l'^{t_2}}$ kofreier Funktor ist, ist $\mathcal{P}^{l'^{t_2}}(r'^{t_1})$ der einzige MP -Homomorphismus, der das Diagramm kommutativ schließt. Zusammen mit (C.721) folgt:

$$(C.722) \quad i \circ u^P = \mathcal{P}^{l'^{t_2}}(r'^{t_1})$$

Nach Lemma A.88 überträgt der Pullback-Funktor surjektive Homomorphismen. Aus der Surjektivität von r'^{t_1} folgt somit die Surjektivität von $\mathcal{P}^{l'^{t_2}}(r'^{t_1})$. Zusammen mit (C.722) folgt, dass i surjektiv ist¹⁴ (Abb. C.15).

Es wird nun gezeigt, dass i voll und injektiv ist. Aus Lemma A.32 folgt zusammen mit der bereits nachgewiesenen Surjektivität von i , dass i MP -Isomorphismus und somit auch Isomorphismus in der Kategorie $\mathbf{Alg}(MP) \downarrow S^{##}$ ist.

- *Injektivität:* Es wird gezeigt, dass i Monomorphismus ist. Weil in der betrachteten Kategorie Monomorphie und Injektivität zusammenfallen, folgt daraus, dass i injektiv ist. Seien also ein System X und zwei Morphismen $f: X \rightarrow \mathcal{F}^{M^*}(P)$ und $g: X \rightarrow \mathcal{F}^{M^*}(P)$ gegeben mit:

¹⁴ Wenn $g \circ f$ surjektiv ist, dann auch g .

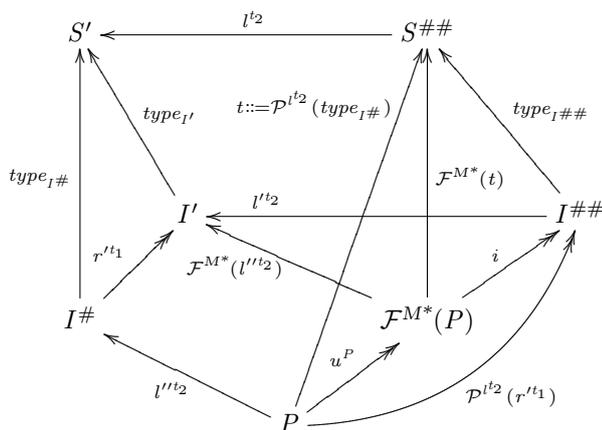


Abb. C.15: Surjektionen

$$(C.723) \quad i \circ f = i \circ g$$

Daraus folgt zum einen:

$$\mathcal{F}^{M^*}(l''t2) \circ f = l'^t2 \circ i \circ f \tag{C.719}$$

$$= l'^t2 \circ i \circ g \tag{C.723}$$

$$(C.724) \quad = \mathcal{F}^{M^*}(l''t2) \circ g \tag{C.719}$$

Zum anderen gilt:

$$\mathcal{F}^{M^*}(t) \circ f = type_{I##} \circ i \circ f \tag{C.720}$$

$$= type_{I##} \circ i \circ g \tag{C.723}$$

$$(C.725) \quad = \mathcal{F}^{M^*}(t) \circ g \tag{C.720}$$

Nach Voraussetzung sind $\mathcal{F}^{M^*}(l''t2)$ und $\mathcal{F}^{M^*}(t)$ gemeinsam strikt voll und somit gemeinsam injektiv. Gemeinsame Injektivität ist in der betrachteten Kategorie identisch mit gemeinsamer Monomorphie. Somit folgt aus (C.724) und (C.725):

$$f = g$$

Also ist i Monomorphismus und somit injektiv.

- *Vollheit:* Sei $p \in P_w$ mit $w \in S^*$ beliebig gegeben. Sei ferner ein Element $x \in \mathcal{F}^{M^*}(P)_w$ gegeben mit:

$$(C.726) \quad i_w(x) \in p^{I##}$$

Daraus folgt zum einen:

$$(C.727) \quad \begin{aligned} i_w(x) \in p^{I^{\#\#}} &\Rightarrow l_w^{t_2}(i_w(x)) \in p^{I'} && (l^{t_2} \text{ ist Homomorphismus}) \\ &\Rightarrow \mathcal{F}^{M^*}(l^{t_2})_w(x) \in p^{I'} && (C.719) \end{aligned}$$

Zum anderen gilt:

$$(C.728) \quad \begin{aligned} i_w(x) \in p^{I^{\#\#}} &\Rightarrow type_{I^{\#\#},w}(i_w(x)) \in p^{S^{\#\#}} && (type_{I^{\#\#}} \text{ ist Homomorphismus}) \\ &\Rightarrow \mathcal{F}^{M^*}(t)_w(x) \in p^{S^{\#\#}} && (C.720) \end{aligned}$$

Nach Voraussetzung sind $\mathcal{F}^{M^*}(l^{t_2})$ und $\mathcal{F}^{M^*}(t)$ gemeinsam strikt voll. Somit folgt aus (C.727) und (C.728):

$$x \in p^{\mathcal{F}^{M^*}(P)}$$

Also ist i strikt voll. □

C.3.13 Beweis von Lemma 11.48

Sei $p \in P_w$ mit $w \in S^*$ beliebig gegeben. Sei $x \in \mathcal{F}^{M^*}(P)_w$ ein beliebiges Tupel, für das gilt:

$$(C.729) \quad \mathcal{F}^{M^*}(l^{t_2})_w(x) \in p^{I'}$$

$$(C.730) \quad \mathcal{F}^{M^*}(t)_w(x) \in p^{S^{\#\#}}$$

Es ist nachzuweisen, dass gilt:

$$(C.731) \quad x \in p^{\mathcal{F}^{M^*}(P)}$$

Weil u^P surjektiv ist, existiert ein Urbild $x' \in P_w$ von x unter u^P . Nun gilt:

$$\mathcal{F}^{M^*}(t)_w(x) \in p^{S^{\#\#}} \quad (C.730)$$

$$\Rightarrow \mathcal{F}^{M^*}(t)_w(u_w^P(x')) \in p^{S^{\#\#}} \quad (\text{Definition von } x')$$

$$(C.732) \quad \Rightarrow t_w(x') \in p^{S^{\#\#}} \quad (u^P \text{ ist } \mathbf{Alg}(MP) \downarrow S^{\#\#}\text{-Morphismus})$$

Es werden zwei Fälle unterschieden:

- (1) $l_w^{t_2}(x') \in p^{I'}$. Weil Pullback-Morphismen nach Lemma A.87 gemeinsam strikt voll sind, folgt zusammen mit (C.732), dass $x' \in p^P$ gilt. Daraus folgt $u_w^P(x') \in p^{\mathcal{F}^{M^*}(P)}$, weil u^P Homomorphismus ist. Zusammen mit der Definition von x' folgt schließlich die Wahrheit von (C.731).

(2) $l_w''t_2(x') \notin p^{I^\#}$. Es gilt:

$$\begin{aligned}
 & \mathcal{F}^{M^*}(l_w''t_2)_w(x) \in p^{I'} && \text{(C.729)} \\
 \Rightarrow & \mathcal{F}^{M^*}(l_w''t_2)_w(u_w^P(x')) \in p^{I'} && \text{(Definition von } x') \\
 & \Rightarrow r_w^{t_1}(l_w''t_2(x')) \in p^{I'} && \text{(Diagramm kommutiert)} \\
 \text{(C.733)} & \Rightarrow l_w''t_2(x') \in \ker_p^{r^{t_1}}
 \end{aligned}$$

Gelte die folgende Implikation für beliebige $x \in P_w$:

$$(C.734) \quad l_w''t_2(x) \in \ker_p^{r^{t_1}} \Rightarrow x \in \ker_p^{u^P}$$

Dann folgt aus (C.733) und (C.734) $u_w^P(x') \in p^{\mathcal{F}^{M^*}(P)}$. Zusammen mit der Definition von x' folgt schließlich die Wahrheit von (C.731).

Es bleibt die Wahrheit von (C.734) zu zeigen. Gelte also die Prämisse:

$$(C.735) \quad l_w''t_2(x') \in \ker_p^{r^{t_1}}$$

Nun wird nach den Prädikaten unterschieden:

(a) $p = =_s$ für $s \in S$. Aus der Voraussetzung $l_w''t_2(x') \notin p^{I^\#}$ und (C.735) folgt nach der Konstruktion der Epireflexion in Konstruktion A.92, dass ein M^* -Axiom und eine entsprechende Variablenbelegung existieren, so dass die Variablenbelegung die Prämisse des Axioms in $P \xrightarrow{t} S^{\#\#}$ erfüllt und in der Konklusion die gewünschte Gleichheit entweder direkt erzwingt oder über den Kongruenzabschluss herbeiführt. Der Kongruenzabschluss muss nicht explizit berücksichtigt werden, weil nach Lemma A.43 der Kern eines Homomorphismus' Kongruenzrelation ist. Somit reicht es aus, diejenigen M^* -Axiome zu betrachten, die in der Konklusion Elemente identifizieren. Das einzige Axiom dieses Typs ist Axiom (M.28), das Elemente zur Sorte N identifiziert. Somit ist es möglich, sich auf $s = N$ zu beschränken. Gelte also $p = =_N$ und $x' = (\underline{x}', \bar{x}')$ mit $\underline{x}', \bar{x}' \in P_N$. Aus der vorausgesetzten Identifikation der Elemente $l_N''t_2(\underline{x}')$ und $l_N''t_2(\bar{x}')$ durch Axiom (M.28) folgt die Wahrheit der Prämisse in $I^\# \xrightarrow{\text{type}_{I^\#}} S'$:

$$(C.736) \quad (l_N''t_2(\underline{x}'), l_N''t_2(\bar{x}')) \in \text{rel}^{I^\#}$$

$$(C.737) \quad \text{type}_{I^\#, N}(l_N''t_2(\underline{x}')) = \text{type}_{I^\#, N}(l_N''t_2(\bar{x}'))$$

Weiter folgt aus (C.732):

$$(C.738) \quad t_N(\underline{x}') = t_N(\bar{x}')$$

Weil $S^{\#\#}$ nach Voraussetzung ein MP -System ist, sind die MP -Axiome (MP.4) und (MP.9) gültig in $S^{\#\#}$. Zusammen mit (C.738) folgt:

$$(C.739) \quad (t_N(\underline{x}'), t_N(\overline{x}')) \in rel^{S^{\#\#}}$$

Nach Voraussetzung ist $I^{\#} \xleftarrow{l''t_2} P \xrightarrow{t} S^{\#\#}$ Pullback von $I^{\#} \xrightarrow{type_{I^{\#}}} S' \xleftarrow{l't_2} S^{\#\#}$. Somit sind nach Lemma A.87 die MP -Homomorphismen $l''t_2$ und t gemeinsam strikt voll. Aus (C.736) und (C.739) folgt somit:

$$(C.740) \quad (\underline{x}', \overline{x}') \in rel^P$$

Nach (C.740) und (C.738) erfüllt die Variablenbelegung

$$\begin{aligned} x &\mapsto \underline{x}' \\ y &\mapsto \overline{x}' \end{aligned}$$

die Prämisse des M^* -Axioms (M.28) in $P \xrightarrow{t} S^{\#\#}$. Nach Konstruktion der Epireflexion in Konstruktion A.92 folgt:

$$(C.741) \quad u_N^P(\underline{x}') = u_N^P(\overline{x}')$$

Aus (C.741) folgt unmittelbar die Wahrheit der Konklusion von (C.734).

- (b) $p = software$. Daraus folgt $w = N$. Weil kein M^* -Axiom existiert, das in der Konklusion das Prädikat *software* wahr macht, gibt es nach Konstruktion der Epireflexion in Konstruktion A.92 ein Element $y \in I_N^{\#}$ mit:

$$(C.742) \quad r_N^{t_1}(l_N''t_2(x')) = r_N^{t_1}(y)$$

$$(C.743) \quad y \in p^{I^{\#}}$$

Aus (C.742) folgt analog zu Punkt 2a:

$$(C.744) \quad (l_N''t_2(x'), y) \in rel^{I^{\#}}$$

$$(C.745) \quad type_{I^{\#}, N}(l_N''t_2(x')) = type_{I^{\#}, N}(y)$$

Daraus folgt:

$$type_{I^{\#}, N}(y) = type_{I^{\#}, N}(l_N''t_2(x')) \quad (C.745)$$

$$(C.746) \quad = l_N^{t_2}(t_N(x')) \quad (\text{Pullbacks kommutieren})$$

Nach Voraussetzung ist $I^\# \xleftarrow{l''t_2} P \xrightarrow{t} S^{\#\#}$ Pullback von $I^\# \xrightarrow{\text{type}_{I^\#}} S' \xleftarrow{l^t_2} S^{\#\#}$. Aus (C.746) folgt somit nach Korollar A.86, dass ein $z \in P_N$ existiert mit:

$$(C.747) \quad t_N(z) = t_N(x')$$

$$(C.748) \quad l''t_2(z) = y$$

Mit der Variablenbelegung $x \mapsto (x', z)$ folgt aus (C.742) und (C.748) nach Punkt 2a:

$$(C.749) \quad u_N^P(x') = u_N^P(z)$$

Weiter folgt aus (C.747) und (C.732):

$$(C.750) \quad t_N(z) \in p^{S^{\#\#}}$$

Schließlich folgt aus (C.748) und (C.743):

$$(C.751) \quad l''t_2(z) \in p^{I^\#}$$

Nach Voraussetzung ist $I^\# \xleftarrow{l''t_2} P \xrightarrow{t} S^{\#\#}$ Pullback von $I^\# \xrightarrow{\text{type}_{I^\#}} S' \xleftarrow{l^t_2} S^{\#\#}$. Weil Pullback-Morphismen nach Lemma A.87 gemeinsam strikt voll sind, folgt zusammen mit (C.750) und (C.751):

$$(C.752) \quad z \in p^P$$

Zusammen mit (C.749) folgt aus (C.752)

$$(C.753) \quad x' \in \ker_p^{u^P}$$

und somit die Wahrheit der Konklusion von (C.734).

- (c) $p = \text{rel}$. Analog zu Punkt 4 im Beweis von Satz 11.33 in Abschnitt C.3.9 folgt $l''t_2(x') \in \text{rel}^{I^\#}$, was ein Widerspruch zur Annahme $l''t_2(x') \notin \text{rel}^{I^\#}$ ist.
- (d) $p = \text{under}$. Offenbar gilt $x' = (\underline{x}', \bar{x}')$. Aus der Voraussetzung $l''t_2(x') \notin p^{I^\#}$ und (C.735) folgt nach Satz 9.17, dass ein erweiterter Pfad von $l''t_2(\underline{x}')$ nach $l''t_2(\bar{x}')$ existiert, wobei die Epireflexion im ersten Schritt alle Elemente, die lediglich über die *rel*-Relation in Beziehung stehen, zusammenlegt, und im zweiten Schritt den transitiven Abschluss der *under*-Relation durchführt. Es gibt somit einen erweiterten Pfad $p_{I^\#} ::= (y_1 ::= l''t_2(\underline{x}'), y_2, \dots, y_n ::= l''t_2(\bar{x}'))$, für den

$$(C.754) \quad (y_i, y_{i+1}) \in \text{under}^{I^\#}$$

oder

$$(C.755) \quad (y_i, y_{i+1}) \in \text{rel}^{I^\#}$$

$$(C.756) \quad \text{type}_{I^\#,N}(y_i) = \text{type}_{I^\#,N}(y_{i+1})$$

für alle $1 \leq i < n$ gilt. Daraus folgt, dass die Abbildung des erweiterten Pfades in das System S' einen (einfachen) Pfad ergibt. Denn im Fall $(y_i, y_{i+1}) \in \text{under}^{I^\#}$ gilt $(\text{type}_{I^\#,N}(y_i), \text{type}_{I^\#,N}(y_{i+1})) \in \text{under}^{S'}$, weil $\text{type}_{I^\#}$ Homomorphismus ist. Und im zweiten Fall fallen nach Voraussetzung die Elemente unter der Abbildung $\text{type}_{I^\#,N}$ zusammen. Somit gibt es in S' von $\text{type}_{I^\#,N}(y_1)$ zu $\text{type}_{I^\#,N}(y_n)$ einen m -elementigen Pfad $p_{S'} ::= (z_1 ::= \text{type}_{I^\#,N}(y_1), z_2, \dots, z_m ::= \text{type}_{I^\#,N}(y_n))$ in S' .¹⁵

Weil Pullbacks kommutieren, folgt aus den Definitionen von z_1 und z_m sowie von y_1 und y_n :

$$(C.757) \quad z_1 = l_N^{t_2}(t_N(\underline{x}'))$$

$$(C.758) \quad z_m = l_N^{t_2}(t_N(\bar{x}'))$$

Schließlich folgt aus (C.732):

$$(C.759) \quad (t_N(\underline{x}'), t_N(\bar{x}')) \in \text{under}^{S^{\#\#}}$$

Weil l^{t_2} nach Voraussetzung abkürzende Pfade zurückzieht, folgt aus (C.757), (C.758), (C.759) und Lemma 11.42, dass ein Pfad von $t_N(\underline{x}')$ zu $t_N(\bar{x}')$ in $S^{\#\#}$ existiert, der zu jedem Element von $p_{S'}$ ein Urbild unter l^{t_2} besitzt. Sei $p_{S^{\#\#}} ::= (z'_1 ::= t_N(\underline{x}'), z'_2, \dots, z'_{m'} := t_N(\bar{x}'))$ dieser m' -elementige Pfad.

Es wird nun gezeigt, dass ein erweiterter Pfad von \underline{x}' zu \bar{x}' existiert. Zunächst besitzt jedes Element y_i mit $1 \leq i \leq n$ ein Urbild unter $l_N^{t_2}$. Denn $\text{type}_{I^\#,N}(y_i) = z_j$ für ein $j \in \{1, \dots, m\}$. Weiter hat jedes z_i mit $1 \leq i \leq m$ ein Urbild $z'_k \in p_{S^{\#\#}}$ unter $l_N^{t_2}$ mit $1 \leq k \leq m'$. Nach Korollar A.86 existiert somit ein Element $x'_i \in P_N$ mit:

$$(C.760) \quad l_N^{t_2}(x'_i) = y_i$$

$$(C.761) \quad t_N(x'_i) = z'_k$$

Seien nun x'_i und x'_{i+1} zwei benachbarte Urbilder für $1 \leq i < n$. Nach (C.761) gibt es $k, k' \in \{1, \dots, m'\}$ mit:

¹⁵ $p_{S'}$ kann kürzer als $p_{I^\#}$ sein (und somit $m < n$ gelten), nämlich wenn $\text{type}_{I^\#}$ einige Elemente von $p_{I^\#}$ identifiziert.

$$(C.762) \quad t_N(x'_i) = z'_k$$

$$(C.763) \quad t_N(x'_{i+1}) = z'_{k'}$$

Aus (C.762) folgt zum einen:

$$l_N^{t_2}(z'_k) = l_N^{t_2}(t_N(x'_i)) \quad (C.762)$$

$$= \text{type}_{I\#,N}(l_N^{t_2}(x'_i)) \quad (\text{Pullbacks kommutieren})$$

$$(C.764) \quad = \text{type}_{I\#,N}(y_i) \quad (C.760)$$

Zum anderen folgt aus (C.763):

$$l_N^{t_2}(z'_{k'}) = l_N^{t_2}(t_N(x'_{i+1})) \quad (C.763)$$

$$= \text{type}_{I\#,N}(l_N^{t_2}(x'_{i+1})) \quad (\text{Pullbacks kommutieren})$$

$$(C.765) \quad = \text{type}_{I\#,N}(y_{i+1}) \quad (C.760)$$

Nach (C.754) und (C.756) gilt

$$\text{type}_{I\#,N}(y_i) = \text{type}_{I\#,N}(y_{i+1})$$

oder

$$(\text{type}_{I\#,N}(y_i), \text{type}_{I\#,N}(y_{i+1})) \in p_{S'}$$

je nachdem, ob $\text{type}_{I\#}$ die beiden Elemente y_i und y_{i+1} identifiziert oder nicht. Im ersten Fall kann unter der Annahme, dass unter l^{t_2} immer dasselbe Urbild ausgewählt wird, o. B. d. A. $k = k'$ und somit $z'_k = z'_{k'}$ angenommen werden. Weil $S^{\#\#}$ ein MP-System ist, ist das MP-Axiom (MP.4) gültig in $S^{\#\#}$. Daraus folgt $(z'_k, z'_{k'}) \in \text{under}^{S^{\#\#}}$. Im zweiten Fall gibt es nach Konstruktion des Pfades $p_{S^{\#\#}}$ einen Pfad von z'_k nach $z'_{k'}$. Weil $S^{\#\#}$ ein MP-System ist, ist das MP-Axiom (MP.6) gültig in $S^{\#\#}$. Daraus folgt ebenfalls $(z'_k, z'_{k'}) \in \text{under}^{S^{\#\#}}$.

Zusammen mit (C.762) und (C.763) ergibt sich somit:

$$(C.766) \quad (t_N(x'_i), t_N(x'_{i+1})) \in \text{under}^{S^{\#\#}}$$

Nach Konstruktion von p_P gilt weiter $(l_N^{t_2}(x'_i), l_N^{t_2}(x'_{i+1})) = (y_i, y_{i+1}) \in p_{I\#}$. Es gibt nun zwei Möglichkeiten:

- (i) $(y_i, y_{i+1}) \in \text{under}^{I\#}$. Aus (C.760) und (C.766) folgt nach Korollar A.87:

$$(C.767) \quad (x'_i, x'_{i+1}) \in \text{under}^P$$

(ii) $(y_i, y_{i+1}) \in \text{rel}^{I^\#}$ und $\text{type}_{I^\#, N}(y_i) = \text{type}_{I^\#, N}(y_{i+1})$. Weil $S^{\#\#}$ ein MP-System ist, ist das MP-Axiom (MP.9) gültig in $S^{\#\#}$. Somit folgt aus (C.766):

$$(C.768) \quad (t_N(x'_i), t_N(x'_{i+1})) \in \text{rel}^{S^{\#\#}}$$

Aus (C.760) und (C.768) folgt nach Korollar A.87:

$$(C.769) \quad (x'_i, x'_{i+1}) \in \text{rel}^P$$

Nach der obigen Argumentation folgt aus der Voraussetzung $\text{type}_{I^\#, N}(y_i) = \text{type}_{I^\#, N}(y_{i+1})$ o. B. d. A. $z'_k = z'_{k'}$ gefolgert werden. Somit folgt aus (C.762) und (C.763):

$$(C.770) \quad t_N(x'_i) = z'_k = z'_{k'} = t_N(x'_{i+1})$$

Aus (C.767) und (C.769) folgt die Existenz eines erweiterten Pfades von $\underline{x}' = x'_1$ nach $\bar{x}' = x'_n$. Zusammen mit (C.770) ergibt sich, dass die Epireflexion nach Satz 9.17 eine Vererbungsverknüpfung zwischen \underline{x}' und \bar{x}' hinzufügt. Somit gilt $(\underline{x}', \bar{x}') \in \text{ker}_{\text{under}}^{r'^{t_1}}$ und folglich die Wahrheit der Konklusion von (C.734). \square

Literaturverzeichnis

- [1] ABITEBOUL, SERGE: *Querying Semi-Structured Data*. In: *Proceedings of the 6th International Conference on Database Theory (ICDT 1997)*, Seiten 1–18, 1997.
- [2] ADÁMEK, JIŘÍ, HORST HERRLICH und GEORGE E. STRECKER: *Abstract and Concrete Categories: The Joy of Cats*. Free Software Foundation, 2004.
- [3] AMBLER, SCOTT W.: *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley, 2006.
- [4] AMBLER, SCOTT W.: *Test-Driven Development of Relational Databases*. IEEE Software, 24:37–43, 2007.
- [5] AMERICAN NATIONAL STANDARDS INSTITUTE: *ANSI/INCITS 319:1998: Programming languages — Smalltalk*. New York, USA, 1998.
- [6] BANERJEE, JAY, HONG-TAI CHOU, JORGE F. GARZA, WON KIM, DARRELL WOELK, NAT BALLOU und HYOUNG-JOO KIM: *Data model issues for object-oriented applications*. ACM Transactions on Information Systems, 5(1):3–26, 1987.
- [7] BANERJEE, JAY, WON KIM, HYOUNG-JOO KIM und HENRY F. KORTH: *Semantics and implementation of schema evolution in object-oriented databases*. ACM SIGMOD Record, 16(3):311–322, 1987.
- [8] BECK, KENT: *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
- [9] BIERMANN, ENRICO, CLAUDIA ERMEL und GABRIELE TAENTZER: *Precise Semantics of EMF Model Transformations by Graph Transformation*. In: *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008)*, Band 5301 der Reihe *Lecture Notes in Computer Science*, Seiten 53–67. Springer-Verlag, 2008.
- [10] BIERMANN, ENRICO, CLAUDIA ERMEL und GABRIELE TAENTZER: *Lifting Parallel Graph Transformation Concepts to Model Transformation based on the Eclipse Modeling Framework*. In: *Manipulation of Graphs, Algebras and Pictures: Essays Dedicated to Hans-Jörg Kreowski on the Occasion of His 60th Birthday*, Seiten 59–76. Universität Bremen, 2009.
- [11] BRÈCHE, PHILIPPE: *Advanced Principles for Changing Schemas of Object Databases*. In: *Proceedings of the 8th International Conference on Advanced Information Systems Engineering (CAiSE 1996)*, Seiten 476–495, 1996.
- [12] BRÈCHE, PHILIPPE und MARTIN WÖRNER: *High level Primitives for Schema Update - User Manual: Prototype 2*, 1995.
- [13] BRÈCHE, PHILIPPE und MARTIN WÖRNER: *Schema Updates Primitives for ODB Design*. Technischer Bericht, Universität Frankfurt, Fachbereich Informatik, 1995.
- [14] BRUNI, ROBERTO und FABIO GADDUCCI: *Some algebraic laws for spans (and their connection with multi-relations)*. Electronic Notes in Theoretical Computer Science, 44(3):175–193, May 2003.

- [15] BUNEMAN, PETER: *Semistructured data*. In: *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS 1997)*, Seiten 117–121. ACM, 1997.
- [16] BURMEISTER, PETER, FRANCESC ROSSELLÓ, JOAN TORRENS und GABRIEL VALIENTE: *Algebraic Transformation of Unary Partial Algebras I: Double-Pushout Approach*. *Theoretical Computer Science*, 184(1–2):145–193, 1997.
- [17] CICHETTI, ANTONIO, DAVIDE DI RUSCIO und ALFONSO PIERANTONIO: *Model Patches in Model-Driven Engineering*. In: *Proceedings of the 3rd International Workshop on Model-Driven Software Evolution (MoDSE-MCCM 2009)*, October 2009.
- [18] CORRADINI, ANDREA, FERNANDO LUÍS DOTTI, LUCIANA FOSS und LEILA RIBEIRO: *Translating Java Code to Graph Transformation Systems*. In: *Proceedings of the 2nd International Conference on Graph Transformation (ICGT 2004)*, Seiten 383–398, 2004.
- [19] DIG, DANNY und RALPH JOHNSON: *The Role of Refactorings in API Evolution*. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005)*, Seiten 389–398. IEEE Computer Society, 2005.
- [20] EHRIG, H., K. EHRIG, U. PRANGE und G. TAENTZER: *Fundamentals of Algebraic Graph Transformation*. Springer-Verlag, 2006.
- [21] EHRIG, H., B. MAHR, F. CORNELIUS, M. GROSSE-RHODE, P. ZEITZ, G. SCHRÖTER und K. ROBERING: *Mathematisch-strukturelle Grundlagen der Informatik, 2. überarbeitete Auflage*. Springer-Verlag, 2001.
- [22] EHRIG, HARTMUT und CLAUDIA ERMEL: *Semantical Correctness and Completeness of Model Transformations using Graph and Rule Transformation*. In: *Proceedings of the 4th International Conference on Graph Transformation (ICGT 2008)*, Band 5214 der Reihe *Lecture Notes in Computer Science*, Seiten 194–210. Springer-Verlag, 2008.
- [23] EHRIG, HARTMUT, CLAUDIA ERMEL und KARSTEN EHRIG: *Evolution of Model Transformations by Model Refactoring (Long Version)*. Technischer Bericht 2009/04, Technische Universität Berlin, Institut für Softwaretechnik und Theoretische Informatik, 2009.
- [24] EHRIG, HARTMUT, CLAUDIA ERMEL und KARSTEN EHRIG: *Refactoring of Model Transformations*. *Electronic Communications of the EASST*, 18, 2009.
- [25] EHRIG, HARTMUT und BERND MAHR: *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, 1985.
- [26] FOLLI, ALESSANDRO und TOM MENS: *Refactoring of UML models using AGG*. *Electronic Communications of the EASST*, 8, 2007.
- [27] FOWLER, MARTIN: *Refactoring. Oder: Wie Sie das Design vorhandener Software verbessern*. Addison-Wesley, 2000.
- [28] FOWLER, MARTIN: *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- [29] FOWLER, MARTIN und KENDALL SCOTT: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2003.
- [30] GHEYI, ROHIT und TIAGO MASSONI: *Formal refactorings for object models*. In: *Companion to the 20th annual ACM SIGPLAN International Conference on Object-oriented programming, systems, languages, and applications (OOPSLA 2005)*, Seiten 208–209. ACM, 2005.
- [31] HABEL, ANNEGRET, JÜRGEN MÜLLER und DETLEF PLUMP: *Double-pushout approach with injective matching*. In: *Proceedings of the 6th International Workshop on Theory and Application of Graph Transformations (TAGT 1998)*, Seiten 431–444. Springer-Verlag, 1998.
- [32] HABEL, ANNEGRET, JÜRGEN MÜLLER und DETLEF PLUMP: *Double-pushout graph transformation revisited*. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.
- [33] HERMANN, FRANK, HARTMUT EHRIG und CLAUDIA ERMEL: *Transformation of Type Graphs with Inheritance for Ensuring Security in E-Government Networks*. In: *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering (FASE 2009)*,

- Band 5503 der Reihe *Lecture Notes in Computer Science*, Seiten 325–339. Springer-Verlag, 2009.
- [34] HERRMANNSDÖRFER, MARKUS und DANIEL RATIU: *Limitations of Automating Model Migration in Response to Metamodel Adaptation*. In: *Proceedings of the 3rd International Workshop on Model-Driven Software Evolution (MoDSE-MCCM 2009)*, October 2009.
- [35] HORN, ALFRED: *On Sentences Which are True of Direct Unions of Algebras*. *Journal of Symbolic Logic*, 16(1):14–21, 1951.
- [36] JOHNSON, RALPH E. und WILLIAM F. OPDYKE: *Refactoring and Aggregation*. In: *Proceedings of the 1st JSSST International Symposium on Object Technologies for Advanced Software*, Band 742, Seiten 264–278. Springer-Verlag, 1993.
- [37] KASTENBERG, HARMEN: *Towards Attributed Graphs in Groove*. In: *Proceedings of the Workshop on Graph Transformation for Verification and Concurrency (GT-VC 2005)*, Band 154 der Reihe *Electronic Notes in Theoretical Computer Science*, Seiten 47–54, Amsterdam, Netherlands, June 2006. Elsevier.
- [38] KASTENBERG, HARMEN, ANNEKE G. KLEPPE und AREND RENSINK: *Defining Object-Oriented Execution Semantics Using Graph Transformations*. In: GORRIERI, R. und H. WEHRHEIM (Herausgeber): *Proceedings of the 8th IFIP International Conference on Formal Methods for Open-Object Based Distributed Systems (FMOODS 2006)*, Band 4037 der Reihe *Lecture Notes in Computer Science*, Seiten 186–201, London, June 2006. Springer-Verlag.
- [39] KASTENBERG, HARMEN, ANNEKE G. KLEPPE und AREND RENSINK: *Engineering object-oriented semantics using graph transformations*. Technischer Bericht TR-CTIT-06-12, University of Twente, Department of Computer Science, 2006.
- [40] KERIEVSKI, JOSHUA: *Refactoring to Patterns*. Addison-Wesley, 2004.
- [41] KIM, WON und HONG-TAI CHOU: *Versions of Schema for Object-Oriented Databases*. In: *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB 1988)*, Seiten 148–159. Morgan Kaufmann Publishers Inc., 1988.
- [42] KNUTH, DONALD E.: *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 3. Auflage, 1997.
- [43] KÖNIG, HARALD, MICHAEL LÖWE und CHRISTOPH SCHULZ: *Functor Semantics for Refactoring-Induced Data Migration*. Technischer Bericht 02007/01, Fachhochschule für die Wirtschaft Hannover, 2007.
- [44] KUNO, HARUMI A., YOUNG-GOOK RA und ELKE A. RUNDENSTEINER: *The Object-Slicing Technique: A Flexible Object Representation and Its Evaluation*. Technischer Bericht CSE-TR-241-95, University of Michigan, Department of Elec. Engineering and Computer Science, 1995.
- [45] LEE, SANG-WON, JUNG-HO AHN und HYOUNG-JOO KIM: *A schema version model for complex objects in object-oriented databases*. *Journal of Systems Architecture*, 52(10):563–577, 2006.
- [46] LEE, SANG-WON und HYOUNG-JOO KIM: *Rich base schema (RiBS): a unified framework for OODB schema version management*. *Journal of Database Management*, 11(1):29–37, 2000.
- [47] LERNER, BARBARA STAUDT: *A model for compound type changes encountered in schema evolution*. *ACM Transactions on Database Systems*, 25(1):83–127, 2000.
- [48] LERNER, BARBARA STAUDT und A. NICO HABERMANN: *Beyond schema evolution to database reorganization*. In: *Proceedings of the European conference on object-oriented programming on Object-oriented programming, systems, languages, and applications (OOPSLA/ECOOP 1990)*, Seiten 67–76. ACM, 1990.

- [49] LLABRÉS, MERCÈ und FRANCESC ROSSELLÓ: *The uniqueness condition for the double pushout transformation of algebras*. Information Sciences, 171(1–3):93–124, 2005.
- [50] LÖWE, MICHAEL: *Konstruktive Spezifikation mit Initialer Semantik*. Technische Universität Berlin, Forschungsgruppe Theoretische Informatik/Formale Spezifikation, 1992.
- [51] LÖWE, MICHAEL, HARALD KÖNIG, CHRISTOPH SCHULZ und MICHAEL PETERS: *Refactoring Information Systems – A Formal Framework*. In: *Proceedings of the 10th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2006)*, Band 1, Seiten 75–80, 2006. Auch erschienen in: *Journal on Systemics, Cybernetics and Informatics*, 5(2):66–71, 2007.
- [52] LÖWE, MICHAEL, HARALD KÖNIG, CHRISTOPH SCHULZ und MICHAEL PETERS: *Refactoring Information Systems – Handling partial composition*. Electronic Communications of the EASST, 3, 2006.
- [53] MAL'CEV, ANATOLY IVANOVICH: *Algebraic systems*. Springer-Verlag, 1973.
- [54] MENS, TOM, SERGE DEMEYER, BART DU BOIS, HANS STENTEN und PIETER VAN GORP: *Refactoring: Current Research and Future Trends*. Electronic Notes in Theoretical Computer Science, 82(3):483–499, 2003.
- [55] MENS, TOM, SERGE DEMEYER und DIRK JANSSENS: *Formalising Behaviour Preserving Program Transformations*. In: GOOS, G., J. HARTMANIS und J. VAN LEEUWEN (Herausgeber): *Proceedings of the 1st International Conference on Graph Transformation (ICGT 2002)*, Band 2505 der Reihe *Lecture Notes in Computer Science*, Seiten 286–301. Springer-Verlag, 2002.
- [56] MILLER, RENÉE J., LAURA M. HAAS und MAURICIO A. HERNÁNDEZ: *Schema Mapping as Query Discovery*. In: *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000)*, Seiten 77–88. Morgan Kaufmann Publishers Inc., 2000.
- [57] MILLER, RENÉE J., YANNIS E. IOANNIDIS und RAGHU RAMAKRISHNAN: *The Use of Information Capacity in Schema Integration and Translation*. In: *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB 1993)*, Seiten 120–133. Morgan Kaufmann Publishers Inc., 1993.
- [58] MILLER, RENÉE J., YANNIS E. IOANNIDIS und RAGHU RAMAKRISHNAN: *Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice*. Information Systems, 19(1):3–31, 1994.
- [59] MILLER, RENÉE J., YANNIS E. IOANNIDIS und RAGHU RAMAKRISHNAN: *Schema Intension Graphs: A Formal Model for the Study of Schema Equivalence*. Technischer Bericht CS-TR-1994-1185, University of Wisconsin-Madison, Department of Computer Sciences, 1994.
- [60] OPDYKE, WILLIAM F.: *Refactoring Object-Oriented Frameworks*. Doktorarbeit, University of Illinois, Urbana-Champaign, IL, USA, 1992.
- [61] OPDYKE, WILLIAM F. und RALPH E. JOHNSON: *Creating Abstract Superclasses by Refactoring*. In: *Proceedings of the 1993 ACM conference on Computer science (CSC 1993)*, Seiten 66–73. ACM, 1993.
- [62] PENNEY, D. JASON und JACOB STEIN: *Class modification in the GemStone object-oriented DBMS*. SIGPLAN Notices, 22(12):111–117, 1987.
- [63] PÉREZ, JAVIER, OLGA RUNGE und GABRIELE TAENTZER: *Specifying and Analyzing Program Refactorings With AGG*. In: *Proceedings of the 4th International Workshop on Graph-Based Tools: The Contest (GraBaTs 2008)*, September 2008.
- [64] PILONE, DAN: *UML 2.0 in a Nutshell*. O'Reilly, 2006.
- [65] RA, YOUNG-GOOK und ELKE A. RUNDENSTEINER: *OODB support for providing transparent schema changes*. In: *Proceedings of the 1994 Conference of the Centre for Advanced Studies on Collaborative research (CASCON 1994)*, Seite 57. IBM Press, 1994.

- [66] RAHM, ERHARD und PHILIP A. BERNSTEIN: *A survey of approaches to automatic schema matching*. The International Journal on Very Large Data Bases, 10(4):334–350, 2001.
- [67] RATZINGER, JACEK, MICHAEL FISCHER und HARALD GALL: *Improving Evolvability through Refactoring*. In: *Proceedings of the 2005 International Workshop on Mining software repositories (MSR 2005)*, Seiten 1–5. ACM, 2005.
- [68] ROBERTS, DONALD BRADLEY: *Practical analysis for refactoring*. Doktorarbeit, University of Illinois, Urbana-Champaign, IL, USA, 1999.
- [69] RODDICK, JOHN F.: *Schema evolution in database systems: an annotated bibliography*. ACM SIGMOD Record, 21(4):35–40, 1992.
- [70] RODDICK, JOHN F.: *A Survey of Schema Versioning Issues for Database Systems*. Information and Software Technology, 37(7):383–393, 1995.
- [71] ROZENBERG, GRZEGORZ (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.
- [72] RUNDENSTEINER, ELKE A.: *MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases*. In: *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB 1992)*, Seiten 187–198. Morgan Kaufmann Publishers Inc., 1992.
- [73] RUTLE, ADRIAN, UWE WOLTER und YNGVE LAMO: *A Diagrammatic Approach to Model Transformations*. In: *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems (EATIS 2008)*, Seiten 1–8. ACM, 2008.
- [74] RUTLE, ADRIAN, UWE WOLTER und YNGVE LAMO: *A formal approach to modeling and model transformations in software engineering*. Technischer Bericht 48, University of Turku, Department of Informatics, 2008.
- [75] SCHULZ, CHRISTOPH: *Algebraische Graphstrukturen und deren Transformation mit dem DPO-Ansatz*. Technischer Bericht 02009/07, Fachhochschule für die Wirtschaft Hannover, 2009.
- [76] SCHULZ, CHRISTOPH: *Konzeptionelles Modell objektorientierter Systeme*. Technischer Bericht 02009/08, Fachhochschule für die Wirtschaft Hannover, 2009.
- [77] SCHULZ, CHRISTOPH: *Mehrsortige algebraische Systeme*. Technischer Bericht 02009/06, Fachhochschule für die Wirtschaft Hannover, 2009.
- [78] SKARRA, ANDREA H. und STANLEY B. ZDONIK: *The Management of Changing Types in an Object-Oriented Database*. In: *Proceedings of the Conference on Object-oriented programming, systems, languages, and applications (OOPSLA 1986)*, Seiten 483–495. ACM, 1986.
- [79] STAHELI, DAVID G.: *Enhanced Debugging with Edit and Continue in Microsoft Visual C++ 6.0*. MSDN Library, 1998.
- [80] TAENTZER, GABRIELE: *AGG: A Graph Transformation Environment for System Modeling and Validation*. In: PFALTZ, J., M. NAGL und B. BOEHLEN (Herausgeber): *Proceedings of the 2nd International Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2003)*, Band 3062 der Reihe *Lecture Notes in Computer Science*, Seiten 446–453. Springer-Verlag, 2004.
- [81] TAENTZER, GABRIELE und MARTIN BEYER: *Amalgamated Graph Transformations and Their Use for Specifying AGG - an Algebraic Graph Grammar System*. In: *Proceedings of the 1993 International Workshop on Graph Transformations in Computer Science*, Seiten 380–394. Springer-Verlag, 1994.
- [82] TAMMA, VALENTINA, STEPHEN CRANFIELD, TIMOTHY W. FININ und STEVEN WILLMOTT (Herausgeber): *Ontologies for Agents: Theory and Experiences*. Whitestein Series in Software Agent Technologies. Birkhäuser, 2005.
- [83] TANENBAUM, ANDREW S.: *Moderne Betriebssysteme*. Prentice Hall/Pearson Studium, 2. Auflage, 2002.

- [84] TICHELAAR, SANDER, STÉPHANE DUCASSE, SERGE DEMEYER und OSCAR NIERSTRASZ: *A Meta-model for Language-Independent Refactoring*. In: *Proceedings of the International Symposium on Principles of Software Evolution (ISPSE 2000)*, Seiten 157–167. IEEE, 2000.
- [85] TOKUDA, LANCE AIJI: *Evolving Object-Oriented Designs with Refactorings*. Doktorarbeit, University of Texas, Austin, Texas, USA, December 1999.
- [86] VOSSEN, GOTTFRIED: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. Oldenbourg, 2000.
- [87] WIDMER, TOBIAS: *Unleashing the Power of Refactoring*. Eclipse Magazine, 1:7–18, August 2006.
- [88] YOUNG-GOOK, RA und ELKE A. RUNDENSTEINER: *A Transparent Schema-Evolution System Based on Object-Oriented View Technology*. IEEE Transactions on Knowledge and Data Engineering, 9(4):600–624, 1997.
- [89] ZICARI, ROBERTO: *A Framework for Schema Updates in an Object-Oriented Database System*. In: *Proceedings of the 7th International Conference on Data Engineering (ICDE 1991)*, Seiten 2–13, 1991.