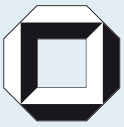
Philipp Bender

Informationserhaltende Sichten und ihre Änderungsoperationen



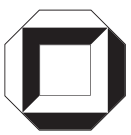
universitätsverlag karlsruhe

Philipp Bender

Informationserhaltende Sichten und ihre Änderungsoperationen

Informationserhaltende Sichten und ihre Änderungsoperationen

von
Philipp Bender



universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH)
Fakultät für Informatik, 2006

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de

Universitätsverlag Karlsruhe 2006
Print on Demand

ISBN 3-86644-041-3

Informationserhaltende Sichten und ihre Änderungsoperationen

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Philipp Bender

aus Pforzheim

Tag der mündlichen Prüfung: 14. Februar 2006

Erster Gutachter: Prof. Dr.-Ing. Dr. h.c. Peter C. Lockemann

Zweiter Gutachter: Prof. Dr.-Ing. Roland Vollmar

Vorwort

Die vorliegende Arbeit ist meine Dissertation, die ich im Rahmen meiner Tätigkeit am Institut für Programmstrukturen und Datenorganisation an der Universität Karlsruhe anfertigte. Zur Entstehung dieser Arbeit haben eine Vielzahl von Personen beigetragen, wobei ich an dieser Stelle gerne die Gelegenheit wahrnehme, mich bei allen Beteiligten zu bedanken.

Mein Dank gebührt meinem Doktorvater, Herrn Prof. Dr. Lockemann, der mit seinen kritischen sowie konstruktiven Fragen es in einzigartiger Weise verstand, die Arbeit auf den richtigen Weg zu bringen. Danken möchte ich ferner Prof. Dr. Vollmar, der sich bereit erklärte, das Korreferat zu übernehmen.

Auch meinen derzeitigen und ehemaligen Kollegen am IPD möchte ich für die Unterstützung in vielerlei Hinsicht danken. Es hat mir sehr viel Spaß gemacht, in einem solchen Team zu arbeiten. Auf Dr. Gerd Hillebrand gehen die ersten Anregungen zum Thema zurück. Michael Klein und Daniel Pfeifer leisteten inhaltliche und moralische Unterstützung während diverser mittäglicher Kaffeerunden. Danken möchte ich auch meinen Projektkollegen Heiko Schepperle und Jutta Mülle, die mir in der Schlussphase der Arbeit oftmals den Rücken von Projektarbeiten freihielten.

Pforzheim, im Februar 2006 Philipp Bender

Inhaltsverzeichnis

1 Einleitung

1.1 Abbildungen zwischen Schemata

Betriebliche Anwendungsprogramme werden für eine Vielzahl von Aufgaben eingesetzt, zu denen etwa die Auftragsverwaltung, die Pflege von Produktkatalogen oder die Prozessüberwachung gehören. Wird ein solches System innerhalb einer monolithischen Architektur realisiert, erfolgt die Präsentation, die Verarbeitung und Speicherung der Daten innerhalb eines einzelnen integrierten Programms. Hierbei operiert jede Programmkomponente auf demselben Datenbestand, der mit Hilfe eines Schemas modelliert wird (Abb. 1.1). Dessen Instanzen $\mathbf{I_1}$, $\mathbf{I_2}$, $\mathbf{I_3}$, ... spiegeln den jeweiligen Programmzustand bzw. die Applikationsdaten wider. Ein Beispiel für ein solches System ist ein objektorientiertes Programm zur Verwaltung des Buchbestands einer Bibliothek, wobei die einzelnen vorhandenen Bücher und deren Eigenschaften, wie Titel und Autoren, in Form von Objekten mit entsprechenden Attributwerten innerhalb der Anwendung gehalten werden. Die Attributwerte der Objekte können sowohl durch eine Präsentationskomponente auf der Benutzungsoberfläche angezeigt als auch von der eigentlichen Programmlogik bearbeitet werden und schließlich zur dauerhaften Speicherung serialisiert und auf Platte geschrieben werden.

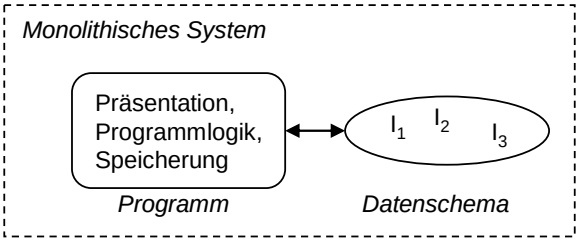


Abbildung 1.1: *Architektur eines monolithischen Systems*

Um die Realisierung komplexer Programme beherrschbar zu machen, folgt ihr Aufbau nicht einem komplett integrierten Ansatz, sondern häufig einer geschichteten Architektur, wobei jede Schicht innerhalb der Anwendung eine ihr zugewiesene Funktionalität erbringt. In einer Referenzarchitektur (Abb. 1.2) für derartige Systeme wird zumeist von drei Schichten ausgegangen, wobei diese strenge Einteilung im realen Anwendungsfall nicht unbedingt eingehalten werden muss, wenn sich dadurch etwa Vorteile bei der

Anwendungsperformanz ergeben. Auf der untersten Ebene dieses Architekturmodells findet sich eine Datenhaltungsschicht, in der die von der Anwendung benötigten Daten persistent hinterlegt sind. Für die Verwaltung großer Datenbestände kommen hierbei Datenbankmanagementsysteme zum Einsatz. Die eigentliche Verarbeitung der gespeicherten Daten und die Erbringung der Programmfunktionalität findet in der mittleren Schicht statt, deren Realisierung durch Applikationsserver oder herstellerspezifische Systeme erfolgt. Letztendlich muss ein Anwender auf das System zugreifen können, so dass sich auf oberster Ebene der Architektur eine Präsentationsschicht zur Darstellung der Benutzungsoberfläche anschließt, über die ein Nutzer mit dem System arbeitet. Hier findet sich entweder eine vollständige Klientapplikation oder es wird ein leichtgewichtiger Klient etwa in Form eines Webbrowsers eingesetzt.

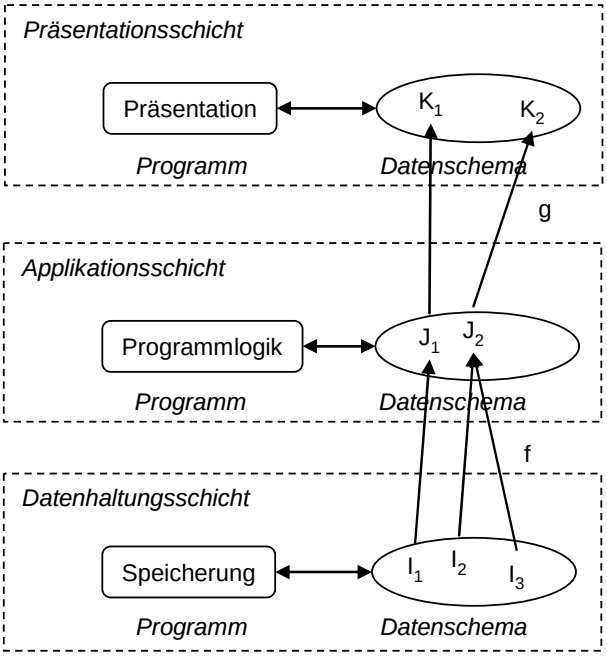


Abbildung 1.2: *Mehr-Schichten-Architektur*

Um den gemeinsamen Datenbestand für die einzelnen Programmteile wieder zugänglich zu machen, wird dieser durch das Anlegen von Kopien ebenfalls auf die Schichten verteilt, so dass jede Schicht eine eigene Repräsentation der aktuellen Programmdaten besitzt. Diese werden jeweils wiederum durch ein eigenes Schema beschrieben. In Abb. 1.2 sind durch das entsprechende Schema die Instanzen $\mathbf{I_1}$, $\mathbf{I_2}$ und $\mathbf{I_3}$ auf Datenbankebene, die Instanzen $\mathbf{J_1}$ und $\mathbf{J_2}$ auf Anwendungsebene und die Instanzen $\mathbf{K_1}$ und $\mathbf{K_2}$ auf der Präsentationsschicht definiert. Welchem Datenmodell ein Schema jeweils folgt, kann von Schicht zu Schicht variieren, je nachdem welches Modell sich am besten für den gegebenen Einsatzzweck eignet. Geht man innerhalb der Applikationsschicht von einem

objektorientierten Programmiermodell aus, bietet es sich an, dort die Daten ebenfalls objektorientiert zu beschreiben, wohingegen ein Datenbanksystem meist einem Schema eines relationalen Modells folgt. Innerhalb der Schichtenarchitektur taucht eine einzelne Datenentität der Realwelt in mehreren Kopien auf. Im Literaturverwaltungssystem wird ein Buch durch seine ISBN, seinen Titel und sein Erscheinungsjahr repräsentiert. Diese Eigenschaften bilden einmal die Attributwerte für ein Objekt der Klasse `Buch` auf der Anwendungsschicht und gleichzeitig sind diese Daten aber auch in der relationalen Datenbank in Form eines Tupels einer Relation *Buch* vorhanden (Abb. 1.3). Somit existiert ein in der Realwelt vorhandenes Buch mit dem Titel „Database theory“ in Form von mehreren Kopien innerhalb des Systems.

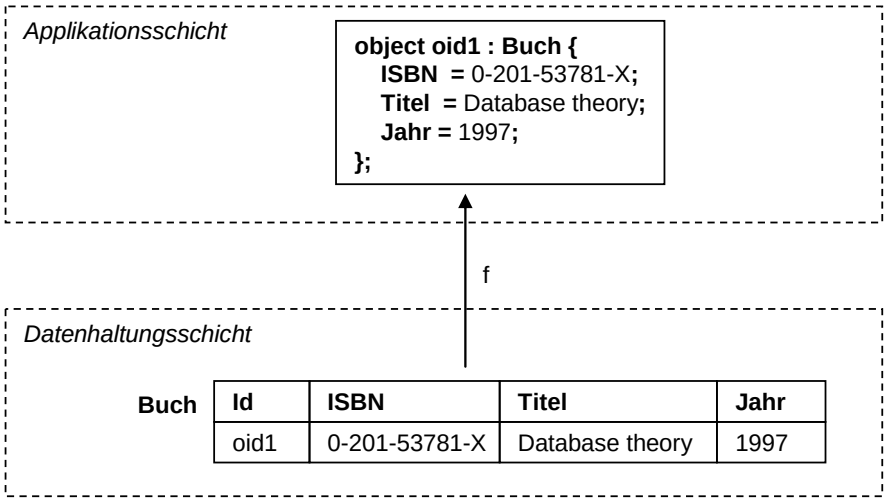


Abbildung 1.3: *Daten auf mehreren Schichten*

Durch die Aufteilung in mehrere Schichten und der Zuweisung von bestimmten Aufgaben zu den jeweiligen Schichten ergeben sich eine Reihe von Vorteilen. So kann auf jeder Ebene eine existierende Implementierung relativ einfach ausgetauscht und durch eine neue ersetzt werden, ohne das gesamte System an sich umbauen zu müssen. Wird auf ein neues Datenbanksystem umgestellt, bleibt die Präsentationsschicht von dieser Änderung unberührt. Außerdem wird es möglich, die einzelnen Anwendungsteile räumlich zu trennen oder redundant auszulegen. Ein Firmenserver kann eine Reihe von Webklienten bedienen, die weltweit verteilt auf den zentral hinterlegten Datenbestand zugreifen. Genauso können mehrere redundant ausgelegte Instanzen eines Applikationsservers aktiv sein und so eine größere Menge von Anfragen von Klienten parallel abarbeiten als wenn nur ein Exemplar des Servers zur Verfügung stünde.

Die Trennung in die verschiedenen Schichten hat nun aber auch zur Folge, dass Daten an den entstandenen Schnittstellen umgewandelt bzw. zwischen den Schichten ausgetauscht werden müssen, so dass auf jeder Ebene der richtige Datenzustand als aktuell

gültig angesehen wird. Die einzelnen Schichten müssen in dieser Hinsicht synchron gehalten werden. Unterscheidet sich der Titel eines Buchs in einem Buchobjekt von dem zugehörigen Tupelattributwert in der Datenbank und werden beide Werte jeweils als gültig betrachtet, liegt eine Inkonsistenz vor, da dann das Buch innerhalb des Systems zwei verschiedene Titel besitzt. Dies entspricht aber nicht der modellierten Welt, da dort jedes Buch nur einen Titel hat. Der Datenaustausch ist nicht auf eine Richtung beschränkt, etwa nur von der Datenhaltungs- zur Applikationsschicht, sondern erfolgt sowohl datenlesend, wenn bestimmte Daten gesucht oder angezeigt werden sollen und hierzu aus der Datenbank ausgelesen werden, als auch datenändernd, wenn innerhalb der Anwendung neue Datensätze angelegt und diese in die Datenbank eingepflegt werden oder bestehende Datenbestände geändert werden. Ebenso findet ein solcher Austausch nicht nur auf zwei Schichten beschränkt statt, sondern erfolgt auch zwischen Programmlogik und Präsentationsschicht, wenn Daten einem Nutzer angezeigt werden oder dieser Daten über die Benutzungsoberfläche des Programms einträgt.

Wie die Daten jeweils zu transformieren sind, wird durch eine Abbildung bzw. Sicht festgelegt, welche letztendlich eine Zuordnung von Schemainstanzen über die Schichtengrenzen hinweg vornimmt und synchrone Datenzustände einander zuordnet. In Abb. 1.2 wird diese Rolle von den beiden Abbildungen f und g übernommen. Die Datenbankinstanz $\mathbf{I_1}$ wird der Anwendungsinstanz $\mathbf{J_1}$ und die Instanzen $\mathbf{I_2}$ und $\mathbf{I_3}$ werden der Instanz $\mathbf{J_2}$ zugewiesen. Durch g erfolgt analog eine Zuordnung von $\mathbf{J_1}$ zu $\mathbf{K_1}$ und $\mathbf{J_2}$ zu $\mathbf{K_2}$.

Da die Daten im Datenbanksystem persistent hinterlegt und somit dort zeitlich gesehen am längsten vorhanden sind, werden ausgehend von diesem Datenbestand die Daten der anderen Schichten abgeleitet. So ist bei einer Beendigung des Programms oder bei einem etwaigen Absturz sichergestellt, dass das Anwendungsprogramm auf dem letzten aktuellen Zustand wieder aufsetzen kann, wenn dieser vorher gespeichert wurde. Die zugehörige Abbildung f beschreibt also, wie die Schemainstanzen der Anwendungsschicht aus den Datenbankdaten abgeleitet werden, so dass etwa der Zustand eines Buchobjekts der Applikationsschicht aus den gespeicherten Tupeln in der Datenbank berechnet werden kann. Die Abbildung f bildet im Beispiel eine Relation des Relationstyps *Buch* auf Objekte der Klasse `Buch` ab, indem die Relationsattribute den gleich benannten Objektattributen zugewiesen werden (Abb. 1.3). Sind in einem Schema mehrere Relationstypen vorhanden, bildet f einen Datenbankzustand, der sich aus den Relationen zusammensetzt, auf eine Anwendungsinstanz in Form von Objekten ab. Wenn die Abbildung entsprechend gewählt wird, ist auch eine Umkehrabbildung möglich, so dass die Attributwerte der Objekte in passenden persistenten Tupeln abgelegt werden können.

Ein Schema bestimmt nicht nur die Struktur der Daten, sondern definiert ebenfalls die Operationen, welche auf den zugehörigen Daten ausgeführt werden können. Sie erlauben es innerhalb einer Schicht, auf Datensätze lesend zuzugreifen oder diese neu anzulegen, zu löschen oder zu modifizieren. Bei Objekten sind Methoden durch die Schnittstellen der zugehörigen Objektklassen spezifiziert, mit denen etwa Attributwerte eines Objekts gelesen oder Attribute mit neuen Werten belegt werden können. Im relationalen Modell gestatten die Operatoren der relationalen Algebra wie die Selektion oder Projektion das

Auswählen von vorhandenen Tupeln und Änderungsoperatoren deren Bearbeitung.

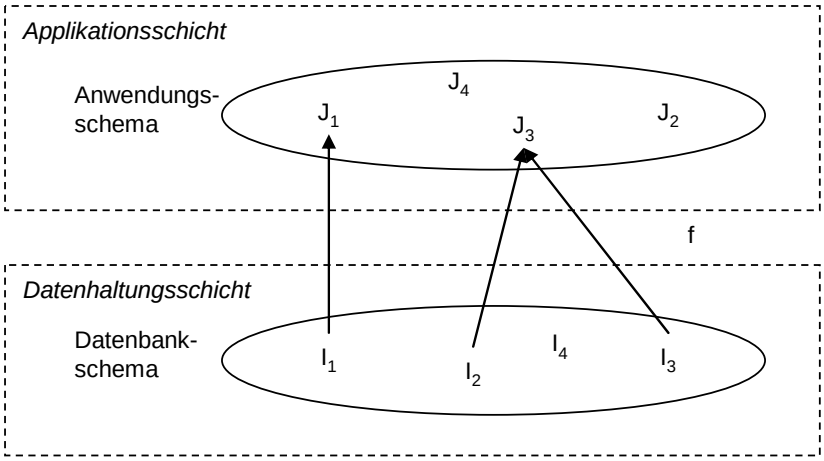


Abbildung 1.4: *Die Instanzen* $\mathbf{I_1}$, $\mathbf{I_2}$ *und* $\mathbf{I_3}$ *des Datenbankschemas werden mit den Instanzen* $\mathbf{J_1}$ *und* $\mathbf{J_3}$ *des Anwendungsschemas durch eine Abbildung* f *verknüpft.*

Das Vorhandensein einer Abbildung hat zum einen zur Folge, dass die Daten im Grunde nun redundant über die unterschiedlichen Schichten verteilt vorliegen, da sich die Daten einer Schicht aus den Daten einer anderen Schicht mittels der Abbildung berechnen lassen. Abb. 1.4 zeigt eine Zuordnung von Schemainstanzen am Beispiel der Anwendungs- und der Datenhaltungsschicht, wobei f die persistenten Instanzen $\mathbf{I_1}$, $\mathbf{I_2}$, $\mathbf{I_3}$ innerhalb des Datenbankschemas mit den Instanzen $\mathbf{J_1}$, $\mathbf{J_3}$ des Anwendungsschemas verknüpft. Zum anderen ziehen Änderungen an den Daten in einer Schicht zumeist Modifikationen der Daten in der anderen Schicht nach sich, weil die aus den Änderungen resultierenden Instanzen wieder durch die Abbildung einander zugewiesen werden müssen. Wird die Änderung auf der jeweils anderen Seite nicht nachgezogen, liegt auf das Gesamtsystem betrachtet ein inkonsistenter Programmzustand vor, da dann in einer Schicht die geänderten Daten gültig sind und in der anderen der alte Zustand weiter bestehen würde.

Durch die Verknüpfung der Instanzen eines Schemas mit den Instanzen eines anderen Schemas über eine gegebene Abbildung ergibt sich die Fragestellung, der im Folgenden nachgegangen werden soll:

> *Sind zwei Schemata gegeben, deren Instanzen durch eine Abbildung miteinander verknüpft sind, welche Änderungsoperationen sind dann auf einem Schema semantisch korrekt durchführbar, so dass die Änderungen auf dem anderen Schema nachgezogen werden können?*

Ein Nutzer von Daten möchte natürlich die durch das Schema definierten Operationen auch tatsächlich ausführen können. Leseoperationen auf dem Datenbankschema stellen

keine Schwierigkeit dar, da sie auf der vorhandenen Datenbankinstanz ausgeführt werden und diese nicht modifizieren. Ändert sich eine Datenbankinstanz, ergibt sich die zum Ergebnis gehörende Anwendungsinstanz durch das Anwenden der Abbildung f auf die geänderte Datenbankinstanz. Betrachtet man das Anwendungsschema, entstehen auch hier bei Leseoperationen keine Probleme, da diese eine Anwendungsinstanz unverändert lassen. Eine solche Operation wird durchgeführt, indem die Abbildung f auf die aktuelle Datenbankinstanz angewendet wird und das Ergebnis mit der lesenden Anfrage des Nutzers verkettet wird. Ist die Anwendungsinstanz materialisiert, fällt der erste Schritt weg. Soll eine ändernde Operation durchgeführt werden, also von einer Instanz zu einer anderen innerhalb des Anwendungsschemas gewechselt werden, muss die Ergebnisinstanz wieder über die Abbildung mit einer passenden Instanz im Datenbankschema verbunden sein. Es muss auf dem Datenbankschema eine passende Operationsfolge gefunden werden, so dass deren Ausführung in einer Datenbankinstanz endet, aus der durch die Abbildung das Ergebnis der ursprünglichen Operation im Anwendungsschema abgeleitet wird. Ändert sich im Beispiel das Erscheinungsjahr in einem Buchobjekt von 1997 auf 2001, so wird diese Änderung in der Datenbank nachgezogen, indem dort der Attributwert im zugehörigen Tupel ebenfalls von 1997 auf 2001 geändert wird. Durch die Abbildung ist das modifizierte Tupel mit dem modifizierten Objekt wieder verknüpft. Kann die Modifikation auf der Datenbank nicht durchgeführt werden oder würde sich hier keine passende Operation finden, existiert das Buch mit dem Erscheinungsjahr 2001 in der Anwendungsschicht und mit dem Jahr 1997 in der Datenbank innerhalb des Systems. Beide Werte würden dann nicht mehr zusammenpassen.

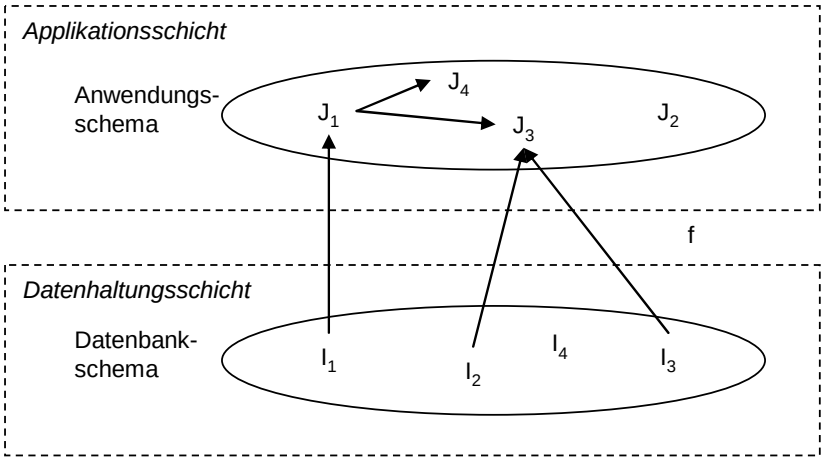


Abbildung 1.5: *Operationen auf dem Anwendungsschema*

Unter semantisch korrekt ist zu verstehen, dass die Durchführung einer Operation den Effekt hat, der durch ihre Semantik beschrieben wird. Soll eine Operation einen neuen Datenwert einfügen und ist nach ihrer Ausführung im Vergleich zur Ausgangsinstanz im

Ergebnis ausschließlich der spezifizierte Wert hinzugekommen, hat die Operation den vom Aufrufer erwarteten Effekt. Ist hingegen der angegebene neue Wert nicht im Ergebnis vorhanden, ist dies sicherlich nicht im Sinne des Aufrufers und folglich nicht semantisch korrekt. Solche Ungereimtheiten zwischen der Definition einer Operation und dem Ergebnis ihrer tatsächlichen Ausführung können entstehen, da eine angegebene Änderungsoperation nicht nur auf dem Anwendungsschema ausgeführt werden muss, sondern auch die Instanz auf Datenbankseite durch eine Operationsfolge angepasst werden muss. Die Existenz einer solchen Folge ist aber weder garantiert, noch muss diese eindeutig bestimmbar sein. Geht man davon aus, dass grundsätzlich zwischen allen Instanzen eines Schema durch einzelne Operationen bzw. Operationsfolgen hin- und hergewechselt werden kann, ist in Abb. 1.5 eine Änderung von $\mathbf{J_1}$ nach $\mathbf{J_4}$ auf Seiten des Anwendungsschemas zulässig, da beide gültige Instanzen dieses Schemas sind. Diese Operation ist aber nicht durchführbar, da die Abbildung f der Instanz $\mathbf{J_4}$ keine Datenbankinstanz zuordnet. Es gibt somit Anwendungsoperationen, die zwar definiert, aber beim Vorhandensein einer Abbildung nicht ausführbar sind. Ebenfalls ein Problem tritt bei einer Änderung von $\mathbf{J_1}$ nach $\mathbf{J_3}$ auf. Um diese Änderung auf der Datenbank nachzuziehen, wäre dort ein Wechsel von $\mathbf{I_1}$ sowohl zu $\mathbf{I_2}$ als auch alternativ zu $\mathbf{I_3}$ möglich. Um die Operation durchführen zu können, muss bei derartigen Mehrdeutigkeiten eine Entscheidung getroffen werden, welche Variante gewählt wird.

Obwohl eine Reihe von Schwierigkeiten auftreten, soll natürlich eine möglichst große Zahl von den auf dem Anwendungsschema definierten Operationen auch durchführbar sein. Allen Problemen aus dem Wege würde man gehen, wenn Änderungsoperationen auf dem Anwendungsschema grundsätzlich verboten wären. Ein Programm mit einem konstanten Zustand ist aber auch nicht unbedingt wünschenswert.

Zu den Anwendungsfällen der formulierten Fragestellung gehört das View-Update-Problem im Fall von Sichten innerhalb des relationalen Datenmodells, das sich durch die Historie der Datenbanktheorie (vgl. etwa [Cod74, Ull89, LL95, Dat00, Vos00, GMUW02]) zieht. Das Datenbankschema ist hier durch ein relationales Schema gegeben, auf dem eine Abbildung als Sicht mittels der Operatoren der relationalen Algebra formuliert wird. Als Ergebnis der Abbildungsdefinition selbst wird das Anwendungsschema in Form eines Relationstyps festgelegt, welcher die durch die Sicht erzeugten Relationen aufnimmt und über welchen ein Anwender auf die hinterlegten Daten zugreift. Als problematisch stellt sich das Durchführen von Änderungsoperationen auf der Sicht heraus, da hierfür eine entsprechende Operationsfolge auf den persistenten Daten gefunden werden muss. Damit eine solche angegeben werden kann, legen bisherige Betrachtungen dieser Problemstellung enge Restriktionen an die Definitionen von Sichten, so dass jeweils nur Teilmengen überhaupt möglicher Sichtendefinitionen untersucht werden. Es wird etwa die Menge der zur Sichtendefinition einsetzbaren Operatoren künstlich eingeschränkt, das verwendete Datenbankschema muss bestimmte Voraussetzungen erfüllen, so dass etwa einattributige Schlüssel vorhanden sein müssen, oder es werden nur wenige ausgewählte Änderungsoperationen auf den Sichten zugelassen, für die eine Umsetzung in Datenbankoperationen angegeben werden kann. Eine allgemeine Betrachtung der Pro-

blemstellung ohne die Festlegung von zu erfüllenden Voraussetzungen erfolgt nicht.

Die Fragestellung betrifft auch das Object/Relational-Mapping-Problem, bei dem objektorientierte Anwendungsdaten in einem relationalen Datenbanksystem persistent hinterlegt werden (vgl. etwa [LL95, KW01, WK02, Top05, Hib05]). Bei einer Abbildungsdefinition muss gewährleistet sein, dass auf dem Anwendungsschema jede dort definierte Änderungsoperation auch tatsächlich wie angegeben durchführbar ist, da innerhalb der Anwendung jeder erlaubte Programmzustand auch irgendwann erreicht werden kann. Um die unterschiedlichen Ausdrucksmächtigkeiten der beteiligten Datenmodelle zu überwinden, werden in der Regel feste Abbildungsmuster vorgegeben, so dass zum Beispiel durch eine Abbildung einem Objekt einer Objektklasse immer ein Tupel eines Relationstyps zugewiesen wird und folglich die Attributwerte eines Objekts innerhalb eines Tupels gespeichert werden. Durch den Einsatz dieser vorbestimmten Abbildungsmuster wird zwar gewährleistet, dass Änderungen der Anwendung an den Daten immer in die Datenbank geschrieben werden können, doch ob die Abbildung an sich den Anforderungen der Anwendung etwa in Bezug auf die Leseperformanz entspricht, bleibt offen. Hier existieren weit mehr Abbildungsmöglichkeiten als durch einen starren Abbildungsmechanismus vorgegeben werden können.

Beide Anwendungsfälle sind nicht neu und doch beschäftigen sie die Datenbankgemeinde seit ihrem Aufkommen. Für beide existieren eine Vielzahl von Untersuchungen und Lösungsvorschlägen, wobei zumeist nicht die komplette Problemstellung betrachtet wird, sondern dadurch, dass die Menge der untersuchten Abbildungen von vorneherein eingeschränkt wird, jeweils nur ein Ausschnitt. Beim View-Update-Problem werden nur Sichten zugelassen, die bestimmte Voraussetzungen erfüllen, und beim O/R-Mapping-Problem ist die Menge der überhaupt erlaubten Abbildungen beschränkt. Über eine Vielzahl von Sichten, welche nicht die jeweils geforderten Vorbedingungen erfüllen, wird folglich keine Aussagen gemacht.

1.2 Idee der Arbeit

Bisherige Untersuchungen zur vorgestellten Fragestellung versuchen, für eine gegebene Probleminstanz etwa in Form einer Sicht durch eine direkte Untersuchung derselben eine Lösung abzuleiten (Abb. 1.6). Der Weg zur Lösungsfindung nimmt eine Probleminstanz PI als Eingabe, führt auf deren Grundlage Analysen durch und liefert die auf dem Anwendungsschema durchführbaren Operationen bzw. bestimmt, wie eine zuvor spezifizierte Operation durchführbar ist. Bei dieser Vorgehensweise stellt sich aber in den meisten Fällen die Probleminstanz selbst als unhandlich heraus. Im Fall von relationalen Sichten liegt es etwa nahe, für die einzelnen Operatoren der Algebra, mit deren Hilfe die Abbildung definiert wird, eine Umkehrabbildung zu suchen und somit für ein Tupel, das in der Sicht vorkommt, diejenigen Tupel in der Datenbank zu finden, aus denen es abgeleitet wird. Bei Änderungen am Sichttupel sind dann die gefundenen Tupel auf Datenbankseite anzupassen. Wie die Umkehrabbildung aussieht, hängt allerdings

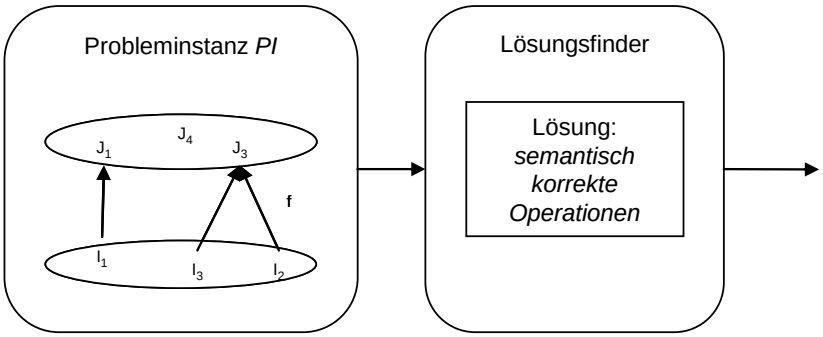


Abbildung 1.6: *Direkte Suche nach einer Lösung*

nicht nur von der Art des Operators also der Abbildung selbst ab, sondern auch von eventuell auf dem Datenbankschema vorkommenden Konsistenzbedingungen. Bei einer Projektion auf einem Schlüsselattribut erfolgt eine eindeutige Zuordnung zwischen einem Tupel der Sicht und einem Tupel in den Datenbankdaten. Bei einer Projektion auf einem Nichtschlüsselattribut fällt diese Eindeutigkeit weg. Die Eigenschaften der Umkehrabbildung hängen also nicht nur von der Abbildungsdefinition als Projektion ab, sondern auch von auf dem Datenbankschema vorkommenden Konsistenzbedingungen, so dass im Grunde bei einer Untersuchung einer Abbildung eine Vielzahl von möglichen Varianten einer Umkehrabbildung betrachtet werden muss. Um die Variantenvielfalt wieder einzuschränken, werden bei bisherigen Untersuchungen an betrachtete Probleminstanzen gewisse Voraussetzungen gestellt; etwa dass bei einer Datenbankrelation immer ein einattributiger Schlüssel vorhanden sein muss und dieser mit in die Sicht übernommen werden muss.

Betrachtet man Abbildungen, so sind die existierenden Darstellungsformen unhandlich, um die semantisch korrekten Operationen zu bestimmen. Im Folgenden ist die Idee, die Herangehensweise an die Fragestellung zu ändern und eine andere Repräsentation einer Abbildungsbeschreibung einzusetzen, die sich als geeigneter zur Lösungsfindung erweisen soll, und eine Ausgangsprobleminstanz darauf abzubilden bzw. zu reduzieren. Ein Problem A kann auf ein andere Problemstellung B reduziert werden, wenn jede Instanz α von A einfach in eine Instanz β von B umformuliert werden kann und eine Lösung von β eine Lösung für α liefert. Kann bei einer neuen Problemstellung B für ein breiteres Spektrum von Instanzen einfacher eine Lösung angegeben werden, entsteht auch für das alte Problem A ein neuer Lösungsweg, wenn das alte auf das neue reduziert werden kann. Da es sich sowohl beim View-Update-Problem als auch beim O/R-Mapping-Problem um Abbildungsprobleme handelt, ist die Idee folglich, beide auf eine andere, passender formulierte Abbildungsproblematik zu reduzieren, bei der die gegebene Fragestellung beantwortet werden kann, ohne die Menge der überhaupt betrachteten Probleminstanzen einzuschränken. Eine Instanz eines Problems besteht hierbei aus den

beiden vorgegebenen Schemata und den jeweils zugehörigen Schemainstanzen, sowie der angegebenen Abbildungsdefinition und den Anwendungs- bzw. Datenbankoperationen.

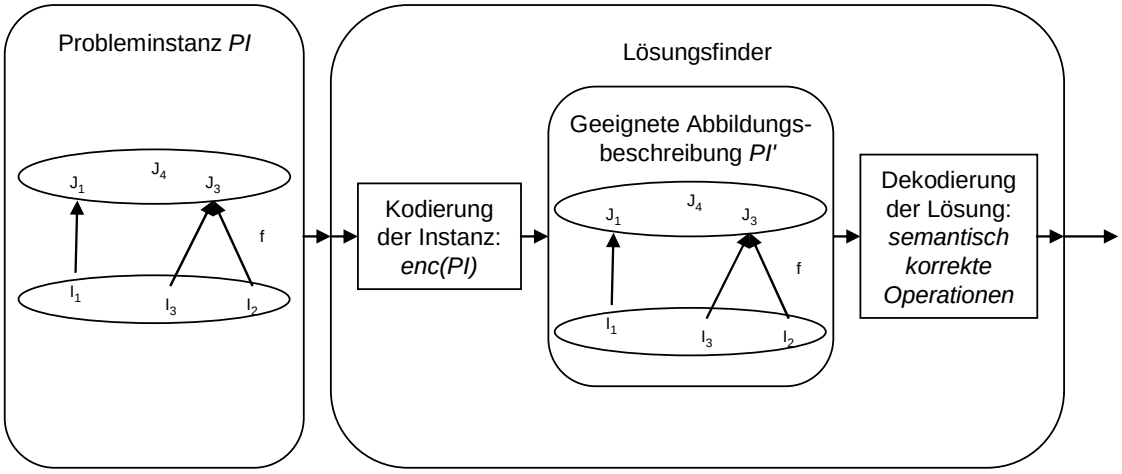


Abbildung 1.7: *Reduktion eines Problems auf ein anderes*

Abb. 1.7 fasst das geplante Vorgehen zusammen. Ist etwa eine relationale Sicht gegeben, so besteht diese aus dem zugrunde liegenden Datenbankschema und dem sich ergebenden Anwendungsschema bzw. Sichtschema. Beide Schemata beschreiben jeweils eine Menge von Schemainstanzen in Form von Relationen, die durch die Sicht aufeinander abgebildet werden. Diese Abbildung wird in der Regel durch einen Ausdruck der relationalen Algebra beschrieben. Die auf dem Sichtschema definierten Änderungsoperatoren gestatten es, neue Tupel in eine Relation einzufügen, existierende Tupel zu löschen oder Werte von Tupelattributen zu ändern. All diese Komponenten zusammengenommen bilden eine Probleminstanz PI. Die Repräsentation einer solchen Probleminstanz wird nun durch eine Funktion *enc* passend umkodiert, so dass sich hieraus eine Probleminstanz PI' einer neuen Problemstellung ergibt, wobei sich diese neue Abbildungsbeschreibung als geeigneter für eine Bestimmung der semantisch korrekten Operationen gestalten soll als die ursprüngliche Problemdarstellung. Die in der neuen Darstellungsform gewonnenen Erkenntnisse werden dann auf das Ursprungsproblem übertragen.

Folgt man dem beschriebenen Verfahren, soll dieses die These der Arbeit unterstützen:

> *Ist eine Abbildung gegeben, kann durch ein formales Berechnungsverfahren auf Basis der auf dem Anwendungsschema durchzuführenden Änderungsoperation, der aktuellen Anwendungsinstanz und letztendlich der Abbildung selbst entschieden werden, ob die Operation semantisch korrekt durchführbar ist oder nicht.*

Die Entscheidung soll auf einem Verfahren beruhen, dass ohne den Eingriff eines Abbildungserstellers auskommt, d.h. das Ergebnis der Entscheidung kann berechnet werden

und ist unabhängig von einem menschlichen Eingriff. Basis eines solchen Entscheidungsverfahrens sollen die durchzuführende Operation und das Anwendungsschema selbst sowie die aktuelle Anwendungsinstanz und letztendlich die Abbildung sein. Die betrachteten Probleminstanzen sollen keinen Einschränkungen bezüglich untersuchter Abbildungen, Datenmodelle oder Schemata unterliegen.

1.3 Gliederung der Arbeit

Die Idee der Arbeit impliziert eine Zweiteilung des weiteren Vorgehens. Als erstes muss eine geeignete Abbildungsproblemformulierung gefunden werden, so dass im zweiten Schritt die eigentliche Reduktion des View-Update-Problems und des O/R-Mapping-Problems auf diese erfolgen kann.

Kapitel 2 untersucht die Anforderungen an eine geeignete Problemformulierung, indem die bei existierenden Darstellungsformen einer Abbildung vorhandenen Defizite genauer beleuchtet werden, welche die Bestimmung der durchführbaren Operationen behindern. Die Spezifikation der neuen Problemstellung und wie hierin eine Lösung gefunden werden kann, folgt in Kapitel 3. Ist die Suche nach einer geeigneten Problemformulierung erfolgreich verlaufen, kann das vorgestellte Lösungsverfahren bei den zwei Anwendungsfällen umgesetzt werden, indem die entsprechenden Problemstellungen reduziert werden. Kapitel 4 führt die generelle Vorgehensweise bei einer Reduktion ein, woran sich zum einen die Reduktion des View-Update-Problems im Fall von relationalen Sichten in Kapitel 5 und zum anderen die Reduktion des O/R-Mapping-Problems in Kapitel 6 anschließt. Kapitel 7 zeigt einige verkürzende Schreibweisen für die Darstellung einer Abbildung innerhalb der neuen Problemstellung auf. Kapitel 8 fasst die gewonnenen Erkenntnisse zusammen und nimmt eine Bewertung vor.

2 Abbildungen zwischen Schemata

Um bei einem Abbildungsproblem die Menge der semantisch korrekten Operationen ermitteln zu können, gilt es zuerst zu klären, welche Kriterien eine Operation erfüllen muss, um überhaupt semantisch korrekt zu sein. Es wird ein Kriterienkatalog benötigt, mit dessen Hilfe entschieden wird, wann die Ausführung einer Operation auch den durch sie definierten Effekt bewirkt und wann nicht.

Die aufgestellten Kriterien sind anhand einer Abbildungsbeschreibung zu überprüfen, wobei diese so dargestellt werden muss, dass ein Verfahren angegeben werden kann, welches die Überprüfung vornimmt. Eine einzelne Abbildungsbeschreibung setzt sich aus vier Gruppen von Bestandteilen zusammen, zu denen die beiden beteiligten Schemata, deren Instanzen, die jeweils auf den Schemata definierten Operationen und die Abbildungsdefinition selbst gehören. Ziel dieses Kapitels ist es, Anforderungen an eine geeignete Repräsentation jeder dieser Komponenten aufzustellen.

Abschnitt 2.1 führt die zur Beschreibung einer Abbildung verwendete Notation ein. In Abschnitt 2.2 wird ein Kriterienkatalog aufgestellt, den eine Operation erfüllen muss, um korrekt durchführbar zu sein. Die nachfolgenden Abschnitte 2.3, 2.4 und 2.5 untersuchen, wie die Bestandteile einer Abbildungsbeschreibung formuliert sein müssen, damit die aufgestellten Kriterien anhand einer Abbildungsbeschreibung überprüft werden können. Abschnitt 2.6 fasst die Kriterien noch einmal zusammen.

2.1 Abbildungsbeschreibung

Damit eine Abbildung auf ihre Eigenschaften hin untersucht werden kann, muss zuerst festgelegt werden, was unter einer Abbildungsbeschreibung zu verstehen ist und wie diese definiert wird. Die Bestandteile einer Beschreibung können in vier Kategorien eingeteilt werden:

- *Werte*
  Daten werden mit Hilfe von Werten beschrieben, wobei einfache als auch zusammengesetzte Werte auftreten können.

- *Schemata*
  Ein Schema modelliert eine Menge von Werten. Gehört ein Wert zu einer von einem Schema beschriebenen Menge von Werten, ist er eine *Instanz* dieses Schemas.

- *Operationen*
  Eine Operation definiert eine Zuordnung zwischen den Instanzen innerhalb eines

einzelnen Schemas.

- *Abbildung*
  Eine Abbildung legt eine Beziehung zwischen Instanzen aus zwei unterschiedlichen Schemata fest.

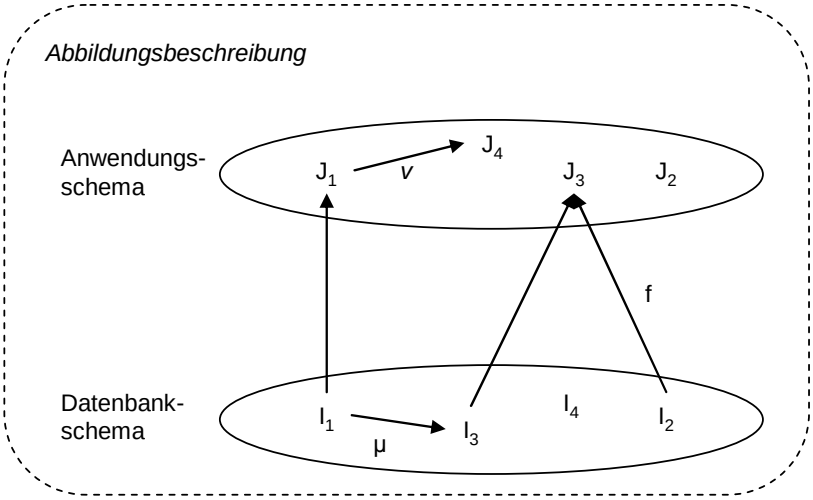


Abbildung 2.1: *Abbildungsbeschreibung*

Graphisch dargestellt ergibt sich für eine Abbildungsbeschreibung ein Bild wie in Abb. 2.1. Ein Datenbankschema $\mathbf{S}$ umfasst eine Reihe von Instanzen $\mathbf{I_1}$, $\mathbf{I_2}$, $\mathbf{I_3}$ und $\mathbf{I_4}$, wobei zu einem bestimmten Zeitpunkt eine davon dem aktuellen Zustand der Datenbank entspricht. Dieser Zustand wird geändert, wenn eine Änderungsoperation $\mu : dom(\mathbf{S}) \rightarrow dom(\mathbf{S})$ auf dem Schema durchgeführt wird. Im Beispiel ist etwa ein Wechsel von $\mathbf{I_1}$ nach $\mathbf{I_3}$ angegeben. Analog zu den Operationen μ auf dem Datenbankschema gestatten Operationen $\nu : dom(\mathbf{T}) \rightarrow dom(\mathbf{T})$ eine Zustandsänderung auf den Instanzen $\mathbf{J_1}$, $\mathbf{J_2}$, $\mathbf{J_3}$ und $\mathbf{J_4}$ des Anwendungsschemas $\mathbf{T}$. Im Folgenden wird von „der“ Änderungsoperation gesprochen, wobei es sich dabei auch durchaus um eine Folge von Einzeloperationen handeln kann und ebenso verschiedene Arten von Operationen wie Einfüge-, Lösch- oder wertändernde Operationen darunter zu verstehen sind. Grundsätzlich wird angenommen, dass zwischen je zwei beliebig gewählten Schemainstanzen durch mindestens eine Operation ein Wechsel möglich ist. Dies ist trivial immer dadurch gegeben, dass die Eingabeinstanz durch die Operation komplett gelöscht und als Ergebnis eine neue Instanz frisch aufgebaut wird.

Beispiel 2.1 (*Schema und Instanzen*)
Unter Verwendung des relationalen Datenmodells [AHV95] modelliert ein Schema $\mathbf{R}$ die Daten eines Anwendungsprogramms, das Literaturangaben zu Büchern verwaltet. Das

Schema besteht aus den drei Relationstypen *Buch*, *Autor* und *Institut* mit den jeweils in eckigen Klammern angegebenen Attributen:

- *Buch* [*ISBN*, *Titel*, *Jahr*]
  enthält Angaben zu einzelnen Büchern in Form von ISBN, Titel und Erscheinungsjahr.
- *Autor* [*Autorenname*, *Institutsname*, *ISBN*]
  beschreibt Buchautoren mit ihrem Namen, dem Institut, bei dem sich tätig sind oder waren, und den ISBN der von ihnen verfassten Bücher.
- *Institut* [*Institutsname*, *Ortsname*]
  listet Institute und deren Standorte auf.

Konsistenzbedingungen wie etwa funktionale Abhängigkeiten sind auf dem Beispielschema nicht definiert.

Eine gültige Schemainstanz **I** für dieses Schema bilden die Relationen aus Abb. 2.2, bei denen die Daten von zwei Büchern, drei Autoren und drei Instituten jeweils als ein Tupel der entsprechenden Relation hinterlegt sind.

Buch

ISBN	Titel	Jahr
0-201-53781-X	Database theory	1997
1-55867-622-X	Data and the Web	2001

Autor

Autorenname	Institutsname	ISBN
S. Maier	I.N.R.I.A.	0-201-53781-X
P. Hill	I.N.R.I.A.	1-55867-622-X
R. Scholl	Bell Labs	0-201-53781-X

Institut

Institutsname	Ortsname
I.N.R.I.A	Rocquencourt
I.N.R.I.A.	Rennes
Bell Labs	Murray Hill

Abbildung 2.2: *Drei Beispielrelationen Buch, Autor und Institut*

Es existieren drei Arten von Änderungsoperatoren auf diesem Schema. Ein Tupel kann in eine Relation eingefügt werden, ein Tupel kann aus einer Relation gelöscht werden oder die Werte innerhalb eines Tupels können modifiziert werden:

- $ins(R, t)$ fügt ein Tupel t in die Relation R ein,

- $del(R, t)$ löscht ein Tupel t aus der Relation R und
- $mod(R, t_1 \rightarrow t_2)$ ersetzt das Tupel t_1 durch t_2 in R.

Hierbei besitzen die Tupel t, t_1 und t_2 jeweils dieselben Attribute wie die Relation, in die sie eingefügt bzw. aus der sie gelöscht werden.

Eine Operation auf dem Beispielschema ist etwa durch eine Folge von zwei Änderungsoperatoren gegeben

$ins(Buch, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1990 $\rangle)$
$del(Autor, \langle$ R. Scholl, Bell Labs, 0-201-53781-X $\rangle)$

mit denen ein neues Tupel in *Buch* eingefügt und ein bestehendes Tupel aus *Autor* gelöscht wird. Nach Ausführung der Operation wird eine neue Schemainstanz $\mathbf{I}'$ erreicht, welche Relationen mit drei Büchern und nur noch zwei Autoren enthält.

Ebenso ist

$mod(Institut, \langle$ I.N.R.I.A., Rennes $\rangle \rightarrow \langle$ Bell Labs, Rennes $\rangle)$

eine Operation, welche die Attributwerte eines Tupels ändert und nur aus einem Operator besteht.

Werden alle bestehenden Tupel aus den drei Relationen gelöscht und diese mit neuen Tupeln gefüllt, kann ein Zustandswechsel zwischen zwei beliebigen Instanzen des Schemas durchgeführt werden. □

Ist ein Datenbankschema $\mathbf{S}$ mit den Instanzen $dom(\mathbf{S})$ gegeben, so weist eine Abbildung $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ einer Instanz $\mathbf{I} \in dom(\mathbf{S})$ des Datenbankschemas eine Instanz $\mathbf{J} \in dom(\mathbf{T})$ des Anwendungsschemas $\mathbf{T}$ zu. Hierbei muss nicht jede Instanz des Datenbankschemas abgebildet werden, so dass der *Definitionsbereich* von f, $dom(f)$, auch durch eine Untermenge von $dom(\mathbf{S})$ gebildet werden kann. Ebenso muss nicht jede Instanz des Anwendungsschemas erreicht werden, so dass der *Bildbereich* der Abbildung, $range(f) = \{f(\mathbf{I}) \mid \mathbf{I} \in dom(\mathbf{S})\}$, eine Teilmenge von $dom(\mathbf{T})$ sein kann. Ohne Beschränkung der Allgemeinheit wird im Folgenden als Bezeichnung des Schemas, das den Definitionsbereich der Abbildung enthält, der Begriff Datenbankschema verwendet und entsprechend das Schema, welches den Bildbereich umfasst, als Anwendungsschema bezeichnet. Somit führt die Abbildungsrichtung immer vom Datenbank- zum Anwendungsschema, also wird einer Datenbankinstanz genau eine Anwendungsinstanz zugewiesen. Welche Modellierungsmöglichkeiten ein Datenbankschema bietet, beschreibt das Datenbankdatenmodell, und entsprechend wird diese Rolle vom Anwendungsdatenmodell auf der anderen Seite der Abbildung übernommen.

Beispiel 2.2 (*Abbildung*)
Im relationalen Datenmodell wird eine Abbildung durch die Operatoren der relationalen Algebra definiert, wie etwa durch die Selektion $\sigma_c(R)$ von Tupeln aus einer Relation R unter Angabe einer Selektionsbedingung c, durch die Projektion $\pi_A(R)$ einzelner Relationsattribute A von R oder durch das kartesische Produkt $R_1 \times R_2$ als Verknüpfung

zweier Relationen R_1 und R_2. Mit dem Schema aus Beispiel 2.1 als Datenbankschema $\mathbf{S}$ definiert eine Abbildung

$$f := \sigma_{ISBN=0-201-53781-X}(Buch)$$

eine Selektion auf der Relation *Buch*, bei der alle Tupel mit der ISBN 0-201-53781-X ausgewählt werden.

Unter den Operatoren der Algebra ist das relationale Datenmodell abgeschlossen, d.h. wird einer der Operatoren auf eine oder mehrere Relationen angewendet, wird als Ergebnis wieder eine Relation erzeugt. Somit kann als Anwendungsschema ein relationales Schema angegeben werden, welches die Bildinstanzen aufnimmt. Auf diesem Schema sind dann ebenso die Operatoren der Algebra und die Änderungsoperatoren definiert. Ein mögliches Anwendungsschema $\mathbf{T}$ für die Beispielabbildung f ist ein Schema, das nur aus dem folgenden Relationstyp besteht:

$$Buch_f\ [\ ISBN,\ Titel,\ Jahr\ ]$$

Die Abbildung f bildet alle Datenbankinstanzen auf Instanzen dieses Anwendungsschemas ab, so dass $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$.

Das angegebene Anwendungsschema $\mathbf{T}$ ist nicht mit dem Bildbereich der Abbildung f identisch, da das Anwendungsschema mehr Instanzen umfasst als der Bildbereich. In einer zum Relationstyp $Buch_f$ gehörenden Relation kann ein Tupel

$$\langle\ \text{1-55867-622-X, Data and the Web, 2001}\ \rangle$$

auftreten, aber in einer durch die Abbildung erzeugten Relation niemals, da dieses Tupel nicht die bei f spezifizierte Selektionsbedingung erfüllt. Die ISBN ist eine andere. Folglich unterscheiden sich die durch den Relationstyp $Buch_f$ beschriebenen Relationen und die durch f erzeugten. Offensichtlich gilt in diesem Beispiel $range(f) \subset dom(\mathbf{T})$, d.h. die Werte des Bildbereichs sind eine echte Untermenge der Instanzen des Anwendungsschemas. □

Die Semantik einer Operation, also der Effekt, der durch sie erreicht werden soll, ist durch ihre Definition $\nu(\mathbf{J_1}) := \mathbf{J_2}$ als ein Übergang zwischen zwei Instanzen $\mathbf{J_1}$ und $\mathbf{J_2}$ eines Schemas gegeben. Bei *Änderungsoperationen* unterscheiden sich diese beiden Instanzen, $\mathbf{J_1} \neq \mathbf{J_2}$, und bei *Leseoperationen* sind sie identisch, $\mathbf{J_1} = \mathbf{J_2}$. Ist der aktuell gültigen Anwendungsinstanz eine Datenbankinstanz zugeordnet, liegt ein konsistenter Systemzustand vor und auf diese Anwendungsinstanz kann eine Operation ν angewendet werden. Wird eine definierte Operation ν tatsächlich auf dem Schema ausgeführt, wird als Ergebnis der Ausführung eine Instanz erwartet, die der in der Definition angegebenen entsprechen sollte. Passend dazu muss eine Operation μ auf dem Datenbankschema gefunden werden, so dass deren Ausführungsergebnis durch die Abbildung f auf das Ergebnis der Durchführung der Anwendungsoperation abgebildet wird. Es liegt dann wieder ein konsistenter Systemzustand vor. Eine solche Operation μ ist eine *Umsetzung* von ν auf dem Datenbankschema. In Abb. 2.3 ist $f(\mathbf{I_1}) = \mathbf{J_1}$ der Ausgangspunkt. Für

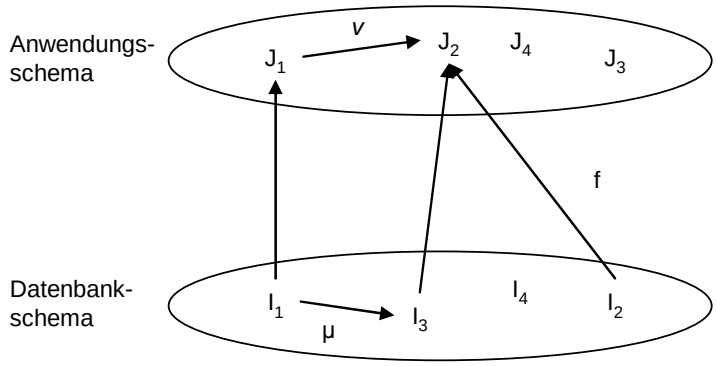


Abbildung 2.3: *Umsetzung einer Operation ν als Operation μ*

die Operation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ ist $\mu(\mathbf{I_1}) := \mathbf{I_3}$ eine mögliche Umsetzung, wobei das Ergebnis der Ausführung von ν die Instanz $\mathbf{J_2}$ und das Ergebnis von μ die Instanz $\mathbf{I_3}$ ist. Nach der Ausführung werden die beiden Ergebnisinstanzen durch f einander zugeordnet, d.h. es gilt hier $f(\mathbf{I_3}) = \mathbf{J_2}$, so dass wie vor der Operationsausführung ein konsistenter Systemzustand vorliegt.

Ist eine Abbildungsbeschreibung gegeben, so treten einige Schwierigkeiten bei der Bestimmung von geeigneten Umsetzungen auf, so dass bei der Durchführung einer Operation unerwünschte Effekte auftreten können. Eine Umsetzung muss nicht eindeutig bestimmbar sein, kann eventuell überhaupt nicht vorhanden sein oder zu Seiteneffekten auf der Anwendungsseite führen. In den folgenden Abschnitten soll untersucht werden,

- welche Kriterien erfüllt sein müssen, dass eine Änderungsoperation ν auf dem Anwendungsschema entsprechend ihrer Definition ausgeführt werden kann, d.h. wann eine geeignete Umsetzung existiert, und
- wie anhand einer Abbildungsbeschreibung die aufgestellten Kriterien für die Ausführbarkeit überprüft werden können.

Bei der Suche nach einer passenden Abbildungsbeschreibung kommt eine Reihe von Beispielen zum Einsatz. Für Beispielabbildungen werden das relationale Datenmodell [Cod70, AHV95], das objektorientierte Modell der ODMG [CB00], sowie das semistrukturierte Modell XML [BPSM+04] mit einer jeweils dazugehörigen Abbildungssprache verwendet.

2.2 Operationen

Sind Änderungsoperationen auf dem Anwendungsschema definiert, werden Kriterien benötigt, anhand derer ermittelt werden kann, wann eine Operation gemäß ihrer Defini-

tion ausgeführt werden kann und wann nicht. Dieser Abschnitt untersucht die Effekte, die bei einer Ausführung auftreten können, und stellt die Kriterien auf, welche durch eine korrekt ausführbare Operation erfüllt sein müssen.

2.2.1 Ausführen von Operationen

Bei der Ausführung von Änderungsoperationen auf dem Anwendungsschema treten beim Vorhandensein einer Abbildung eine Reihe von Schwierigkeiten auf. So existiert zu einer Änderungsoperation auf dem Anwendungsschema im Regelfall nicht nur eine gültige Umsetzung, sondern eine Vielzahl möglicher Umsetzungen, die alle zum gewünschten Effekt auf dem Anwendungsschema führen.

Beispiel 2.3 (*Umsetzung: Mehrdeutigkeiten*)
Betrachtet man das relationale Schema aus Beispiel 2.1 mit dem Relationstyp $Buch$, wird eine Abbildung $Buch_{ISBN,Titel}$, welche die Nummern und Titel der hinterlegten Bücher ausliest, mit Hilfe einer Projektion auf dem Datenbankschema definiert:

$$Buch_{ISBN,Titel} := \pi_{ISBN,Titel}(Buch)$$

Ein passendes Anwendungsschema hierzu ist

$$Buch_{ISBN,Titel}\ [\ ISBN, Titel\ ]$$

zu welchem die Ergebnisrelationen der Abbildung Instanzen sind. Mit der Relation $Buch$ aus dem Beispiel als Datenbankinstanz enthält die zugehörige Anwendungsrelation $Buch_{ISBN,Titel}$ die Werte der beiden angegebenen Attribute in Form von zwei Tupeln:

ISBN	Titel
0-201-53781-X	Database theory
1-55867-622-X	Data and the Web

Eine auf dem Anwendungsschema erlaubte Änderungsoperation ist das Einfügen eines neuen Tupels, da hier die Operation

$$ins(Buch_{ISBN,Titel},\ \langle\ \text{3-569-45667-X, Einsatz von Datenbanken}\ \rangle)$$

definiert ist. Als Ergebnis der Operationsausführung wird erwartet, dass die Relation $Buch_{ISBN,Titel}$ drei Tupel enthält; die beiden vorhandenen und das durch die Operation neu einzufügende.

Passend gilt es eine Änderungsoperation auf dem Datenbankschema anzugeben, so dass deren Ergebnis durch die Abbildung $Buch_{ISBN,Titel}$ auf eine Relation mit den beiden bestehenden und dem neuen Tupel im Anwendungsschema abgebildet wird. Da das

Erscheinungsjahr eines Buchs nicht im Anwendungsschema vorkommt, in der Datenbankrelation aber angegeben werden muss, kann ein passender Wert „geraten“ werden, um die Einfügeoperation umzusetzen. Nimmt man 1997 als Wert für die Jahreszahl, wird in die Datenbankrelation $Buch$ das folgende Tupel eingefügt:

$ins(Buch, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1997 $\rangle)$

Bildet man die sich ergebende Relation von $Buch$ durch die Abbildung $Buch_{ISBN,Titel}$ in das Anwendungsschema ab, ist dort die gewünschte Zielrelation mit der angegebenen Änderung vorhanden. Mit dieser Umsetzung hat die Einfügeoperation auf dem Anwendungsschema den definierten Effekt; das neue Tupel ist eingefügt.

Eine weitere Umsetzung wäre das Einfügen nicht nur eines Tupels, sondern mehrerer Tupel in die Relation $Buch$, welche alle dieselben Werte bei $ISBN$ und $Titel$ besitzen, aber unterschiedliche Jahreszahlen enthalten:

$ins(Buch, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1996 $\rangle)$
$ins(Buch, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1997 $\rangle)$
$ins(Buch, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1998 $\rangle)$

Auch hier entsteht nach der Abbildung die erwartete Anwendungsinstanz. Ob es sinnvoll ist, mehrere Tupel mit unterschiedlichen Jahreszahlen zu verwenden, ist eine Frage der Semantik und soll hier außer Acht gelassen werden.

Eine dritte Variante einer gültigen Umsetzung wäre die folgende Operation, bei der ein Tupel in der Datenbank eingefügt und ein anders modifiziert wird:

$ins(Buch, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1997 $\rangle)$
$mod(Buch, \langle$ 0-201-53781-X, Database theory, 1997 $\rangle \rightarrow$
$\langle$ 0-201-53781-X, Database theory, 2002 $\rangle)$

Hiernach entsteht ebenfalls eine Datenbankinstanz, aus welcher die Abbildung $Buch_{ISBN,Titel}$ die gewünschte Anwendungsinstanz ableitet.

Eine zweite Abbildung $Buch_{Jahr2000}$ ist definiert als eine Selektion

$Buch_{Jahr2000} := \sigma_{Jahr>2000}(Buch)$

und wählt diejenigen Tupel aus der Datenbankrelation in die Anwendungsrelation, die nach dem Jahr 2000 erschienen sind:

ISBN	Titel	Jahr
1-55867-622-X	Data and the Web	2001

Die Rolle des Anwendungsschemas bei dieser Abbildung übernimmt der Relationstyp

$Buch_{Jahr2000}$ [$ISBN$, $Titel$, $Jahr$]

Er erlaubt das Löschen des vorhandenen Tupels

$$del(Buch_{Jahr2000}, \langle \text{ 1-55867-622-X, Data and the Web, 2001 } \rangle)$$

aus der Relation $Buch_{Jahr2000}$, wobei diese Operation umgesetzt werden kann durch ein Löschen desselben Tupels in der Datenbankrelation $Buch$:

$$del(Buch, \langle \text{ 1-55867-622-X, Data and the Web, 2001 } \rangle)$$

Eine weitere Möglichkeit einer Umsetzung ist aber auch, das zu löschende Tupel in $Buch$ durch ein anderes Tupel zu ersetzen, welches die Selektionsbedingung der Abbildung nicht erfüllt:

$$mod(Buch, \langle \text{ 1-55867-622-X, Data and the Web, 2001 } \rangle \rightarrow \langle \text{ 1-55867-622-X, Data and the Web, 1999 } \rangle),$$

Hier wird das Erscheinungsjahr so geändert, dass das Tupel nicht mehr von $Buch_{Jahr2000}$ ausgewählt wird. Dies ist ebenfalls eine gültige Umsetzung, da das zu entfernende Tupel nicht mehr in der resultierenden Anwendungsinstanz auftritt. □

Die Umsetzung einer Änderungsoperation auf der Anwendungsseite in eine Operation auf der Datenbankseite ist nicht zwingend eindeutig festgelegt. Es gibt oft mehrere Möglichkeiten, um die gewünschte Anwendungsinstanz zu erhalten, wobei die Art der Änderungsoperation bei der Umsetzung nicht erhalten bleiben muss. Bei der zweiten Abbildung in Beispiel 2.3 kann ein Löschen eines Tupels in der Anwendungsrelation durch ein Löschen eines Tupels in der Datenbankrelation oder aber auch durch ein Ändern eines Tupels erfolgen.

Abb. 2.4 zeigt den Fall von nicht eindeutigen Umsetzungen anhand einer Abbildungsbeschreibung. Soll eine Operation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ durchgeführt werden, kann bei der angegebenen Abbildung f entweder $\mu_1(\mathbf{I_1}) := \mathbf{I_2}$ oder $\mu_2(\mathbf{I_1}) := \mathbf{I_3}$ als Umsetzung ausgewählt werden, da die Ergebnisse beider Operationen durch f auf die gewünschte Anwendungsinstanz $\mathbf{J_2}$ abgebildet werden. Bei einer Ausführung von ν muss hier folglich eine Entscheidung getroffen werden, welche der beiden Operationen μ_1 und μ_2 letztendlich verwendet wird.

Neben Operationen mit mehr als einer Umsetzung können auch Änderungsoperationen auf dem Anwendungsschema definiert sein, die zwar laut Definition eine Änderung an den Daten bewirken sollen, deren Ausführung aber keinerlei Effekt besitzt.

Beispiel 2.4 (*Seiteneffekte: Effektfrei*)
Auf dem Anwendungsschema der Abbildung $Buch_{Jahr2000}$ aus Beispiel 2.3 sind ebenfalls Einfügeoperationen definiert. Soll die Relation $Buch_{Jahr2000}$ um ein weiteres Tupel ergänzt werden, kann die Einfügeoperation

$$ins(Buch_{Jahr2000}, \langle \text{ 0-13-881333-0, Einführung in Datenbanksysteme, 2002 } \rangle)$$

als das Einfügen desselben Tupels in die Datenbankrelation $Buch$ umgesetzt werden:

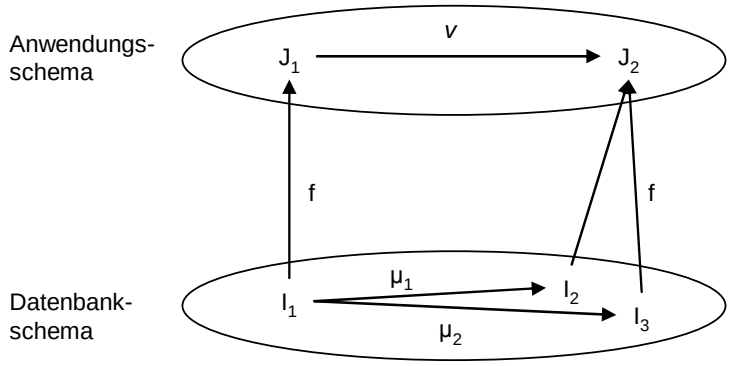


Abbildung 2.4: *Für die Operation ν gibt es mehrere Umsetzungen: μ_1 und μ_2*

$ins(Buch, \langle$ 0-13-881333-0, Einführung in Datenbanksysteme, 2002 $\rangle)$

Aus der geänderten Datenbankrelation berechnet die Abbildung die gewünschte Bildrelation, die neben dem vorhandenen das neu eingefügte Tupel enthält.

Soll ein weiteres Tupel durch

$ins(Buch_{Jahr2000}, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1997 $\rangle)$

in $Buch_{Jahr2000}$ ergänzt werden, liegt es nahe, wie beim zuvor eingefügten Tupel vorzugehen und das neue Tupel ebenfalls in $Buch$ einzufügen:

$ins(Buch, \langle$ 3-569-45667-X, Einsatz von Datenbanken, 1997 $\rangle)$

Nur, das zuletzt eingefügte Tupel erscheint nicht in der Anwendungsrelation, da die Jahreszahl 1997 die Selektionsbedingung von $Buch_{Jahr2000}$ verletzt. Auch mit einer anderen Umsetzung der Einfügeoperation auf Operationen auf dem Datenbankschema kann das Ziel – dass die Anwendungsrelation das neu ergänzte Tupel enthält – nicht erreicht werden, da dieses Tupel immer von der Selektionsbedingung der Abbildung herausgefiltert wird. □

Auf dem Anwendungsschema können Änderungsoperationen definiert sein, die dort keinerlei Effekt besitzen. Eine solche Änderungsoperation ist bei ihrer Durchführung effektfrei, d.h. die Instanz vor der Ausführung ist mit der nach der Ausführung identisch. Abb. 2.5 zeigt dies anhand einer Operation $\nu(\mathbf{J_1}) := \mathbf{J_2}$. Von ihrer Definition ausgehend würde eine neue Instanz $\mathbf{J_2}$ erreicht, doch gehört diese nicht zum Bildbereich der Abbildung f, d.h. ihr ist keine Datenbankinstanz zugeordnet. Würde die Operation wie angegeben ausgeführt, würde ein inkonsistenter Systemzustand entstehen. Eine Möglichkeit, einen solchen Zustand zu vermeiden, ist die Durchführung zusätzlicher Korrekturmaßnahmen in Form einer nachgeschalteten Operation ν', die zu einer Ergebnisinstanz führt, der eine Datenbankinstanz zugewiesen ist. Im Beispiel kann diese Korrektur aufgrund der

vorhandenen Umsetzung $\mu(\mathbf{I_1}) := \mathbf{I_2}$ bestimmt werden und besteht in einem Zurücknehmen der Änderungsoperation auf dem Anwendungsschema, indem das eingefügte Tupel dort gleich wieder entfernt wird, so dass eine Durchführung von ν wieder zurück nach $\mathbf{J_1}$ führt. Letztendlich wird also $\nu'(\nu(\mathbf{J_1})) = \mathbf{J_1}$ ausgeführt und nicht das in der Definition der Operation angegebene $\nu(\mathbf{J_1}) = \mathbf{J_2}$. Die Ausführung von ν ist somit effektfrei. Die Definition und die Ausführung der Operation differieren, so dass die Ausführung nicht das bewirkt, was die Definition spezifiziert.

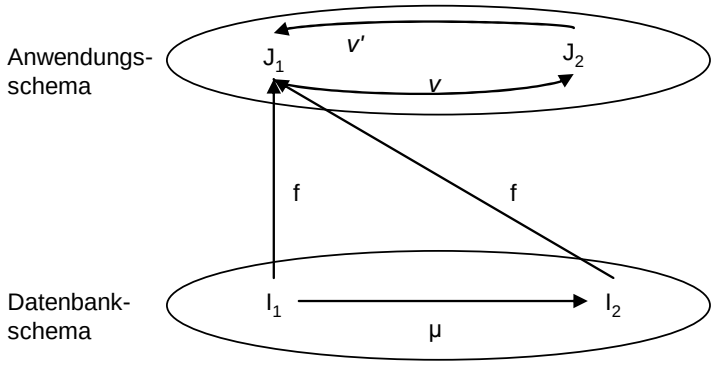


Abbildung 2.5: *Operation ν ist effektfrei*

Existiert auf dem Anwendungsschema eine Änderungsoperation, deren Durchführung zwar den gewünschten Effekt einschließt, aber zusätzlich zur vom Nutzer angegebenen Änderung Werte hinzugefügt oder löscht, folgt die Ausführung ebenfalls nicht der Definition der Operation.

Beispiel 2.5 (*Seiteneffekte: Zusätzliche Werte werden entfernt oder kommen hinzu*) Betrachtet man die Abbildung

$$AutorOrt := \pi_{Autorenname,Ortsname} (\sigma_{Autor.Institutsname=Institut.Institutsname}(Autor \times Institut))$$

welche die Autoren und ihre zugehörigen Institutsorte ausgibt, dann enthält im Anwendungsschema die zum Relationstyp

$$AutorOrt\ [\ Autorenname, Ortsname\ ]$$

gehörige Anwendungsrelation die in Abb. 2.6 angegebenen Einträge.

Möchte ein Nutzer das erste Tupel dieser Relation mit der Operation

$$del(AutorOrt, \langle\ \text{S. Maier, Rocquencourt}\ \rangle)$$

löschen, so gibt die Definition dieser Operation vor, dass das Ergebnis eine Relation ist, bei der das angegebene Tupel fehlt und die übrigen vier Tupel noch vorhanden sind.

Autorenname	Ortsname
S. Maier	Rocquencourt
S. Maier	Rennes
P. Hill	Rocquencourt
P. Hill	Rennes
R. Scholl	Murray Hill

Abbildung 2.6: *Anwendungsrelation*

Eine Möglichkeit zur Umsetzung dieser Löschoperation auf der Datenbank ist das Entfernen eines Tupels aus der Relation *Autor*:

$$del(Autor, \langle \text{ S. Maier, I.N.R.I.A., 0-201-53781-X } \rangle)$$

Berechnet man mit der Abbildung *AutorOrt* aus dem Ergebnis die zugehörige Bildinstanz, hat diese Umsetzung neben dem Entfernen des angegebenen Tupels aus der Anwendungsinstanz aber auch zur Folge, dass das Tupel ⟨ S. Maier, Rennes ⟩ ebenfalls aus der Relation *AutorOrt* gelöscht wird. Die Durchführung der Operation resultiert folglich auf dem Anwendungsschema in einer Relation, die statt der erwarteten vier Tupel nur noch drei Tupel enthält.

Wählt man als Umsetzung das Löschen des ersten Tupels aus $Institut$

$$del(Institut, \langle \text{ I.N.R.I.A., Rocquencourt } \rangle)$$

kommt in der Anwendungsrelation zusätzlich das Tupel ⟨ P. Hill, Rocquencourt ⟩ abhanden. Beide Umsetzungen lösen also neben dem definierten Effekt – dem Löschen des angegebenen Tupels – weitere Seiteneffekte in Form von zusätzlich verschwindenden Tupeln auf der Anwendungsseite aus.

Eine weitere Abbildung, welche die Namen der Institute auswählt, ist durch die Projektion

$$Institut_{Name} := \pi_{Institutsname}(Institut)$$

gegeben. Als Anwendungsinstanz wird der Datenbankrelation die Relation

Institutsname
I.N.R.I.A
Bell Labs

zugeordnet passend zum Relationstyp

$Institut_{Name}$ [$Institutsname$]

Wird hier ein Institutsname geändert,

$mod(Institut_{Name}, \langle$ I.N.R.I.A. $\rangle \rightarrow \langle$ Sojo Labs $\rangle)$

kann dies als eine Modifikation

$mod(Institut, \langle$ I.N.R.I.A., Rennes $\rangle \rightarrow \langle$ Sojo Labs, Rennes $\rangle)$

auf der Datenbankinstanz umgesetzt werden. Die sich ergebende Anwendungsinstanz enthält dann allerdings statt der bisherigen zwei Tupel nun drei Tupel, da bei der Datenbankinstanz zwei Tupel mit dem Eintrag I.N.R.I.A. vorhanden sind, aber nur eines davon modifiziert wird. Im Anwendungsschema kann dies so interpretiert werden, dass der Institutsname des einen Tupels von I.N.R.I.A. auf Sojo Labs geändert wurde und ein zusätzliches Tupel mit dem Wert I.N.R.I.A. wieder eingefügt wurde:

Institutsname
Sojo Labs
Bell Labs
I.N.R.I.A

□

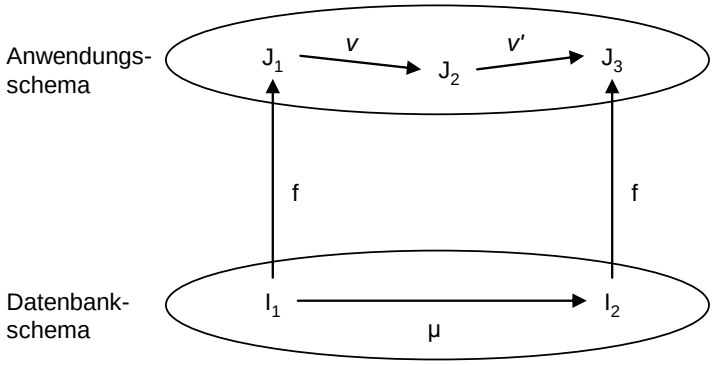


Abbildung 2.7: *Operation ν führt zu Seiteneffekten*

Umsetzungen können einen anderen als den von der Operation definierten Effekt im Anwendungsschema auslösen, so dass neben den durch die Operation spezifizierten Werten weitere Werte gelöscht oder hinzugefügt werden. Beides gibt die Definition der Änderungsoperation nicht explizit an. Solche Seiteneffekte entstehen, wenn die vom Nutzer

durchgeführte Änderungsoperation auf dem Anwendungsschema dort zwar zu einer Instanz des Anwendungsschemas führt, diese Instanz aber nicht zum Bildbereich der Abbildung gehört; es also keine Datenbankinstanz gibt, aus der sie hergeleitet werden kann. Abb. 2.7 zeigt eine Anwendungsoperation $\nu(\mathbf{J_1}) := \mathbf{J_2}$, die zu einer Instanz $\mathbf{J_2}$ führt, die zwar innerhalb des Anwendungsschemas gültig ist, aber über die Abbildung f nicht erreicht wird. Wird eine solche Operation dennoch durchgeführt, muss an diese wieder eine Korrekturmaßnahme ν' angefügt werden, so dass eine Instanz $\mathbf{J_3}$ erreicht wird, der eine Datenbankinstanz zugeordnet ist, so dass wieder ein konsistenter Zustand erreicht wird. Auf Basis der Umsetzung $\mu(\mathbf{I_1}) = \mathbf{I_2}$ kann das Ergebnis der Korrekturmaßnahme als $\mathbf{J_3} = f(\mathbf{I_2})$ bestimmt werden. In Beispiel 2.5 entspricht ν' dem Löschen bzw. Einfügen zusätzlicher Tupel auf Anwendungsseite. Eine Ausführung von ν resultiert nicht in $\nu(\mathbf{J_1}) = \mathbf{J_2}$, sondern in $\nu'(\nu(\mathbf{J_1})) = \mathbf{J_3}$. Wieder unterscheiden sich Definition und Ausführung einer Operation, wobei die Ausführung mehr oder weniger Änderungen bewirkt als die Definition spezifiziert.

2.2.2 Verwandte Arbeiten

Abbildungen innerhalb des relationalen Modells in Form von Sichten werden schon seit langer Zeit untersucht. Ist hierbei eine Änderungsoperation auf der Sicht gegeben, werden Kriterien definiert, wann eine solche Operation ausgeführt werden kann und wann nicht.

Einen ersten Kriterienkatalog zur Beschreibung ausführbarer Änderungsoperationen auf einer Sicht stellen [DB78] auf. Eine Änderungsoperation ist durchführbar, wenn es eine eindeutig bestimmbare Umsetzung auf der Datenbank gibt, die minimal, seiteneffektfrei, konsistenz- und arterhaltend ist. Eine Umsetzung ist minimal, wenn alle anderen möglichen Umsetzungen mehr Änderungen an der Datenbankinstanz vornehmen, um dasselbe Ergebnis in der Sicht zu erreichen. In Beispiel 2.3 wird somit das Hinzufügen eines einzigen Tupels in der Datenbankrelation dem Hinzufügen von drei Tupeln vorgezogen. Eine Umsetzung soll keine Seiteneffekte im Anwendungsschema bewirken. Dieses Kriterium erfüllt bei Beispiel 2.5 keine der angegebenen Umsetzungen, da zusätzlich in der Anwendungsrelation Tupel hinzukommen oder wegfallen. Sind auf dem Datenbankschema Konsistenzbedingungen definiert, dürfen diese durch eine Umsetzung nicht verletzt werden. Als weiteres Kriterium wird die Arterhaltung der Operation bei einer Umsetzung gefordert. Fügt die Operation auf dem Anwendungsschema Tupel hinzu, so soll die Umsetzung ebenfalls nur Tupel hinzufügen. Die erste Umsetzung aus Beispiel 2.3 erfüllt dieses Kriterium, die letzte aus demselben Beispiel erfüllt es nicht. Ist anhand dieser Kriterien für eine Anwendungsoperation ν auf einer Instanz $f(\mathbf{I_1})$ eine Umsetzung μ eindeutig bestimmbar, dann ist die Operation auf dem Anwendungsschema durch eben diese Umsetzung durchführbar und es gilt $\nu(f(\mathbf{I_1})) = f(\mu(\mathbf{I_1}))$ (Abb. 2.8).

Die aufgestellten Kriterien besitzen einige Schwachpunkte. So schränkt die Forderung nach einer eindeutigen Bestimmbarkeit der Umsetzung die Menge der durchführbaren Operationen ein, da Operationen als nicht ausführbar angesehen werden, obwohl sie es eigentlich doch sind. Dieser Fall tritt bei Operationen ein, für die eine Menge von gülti-

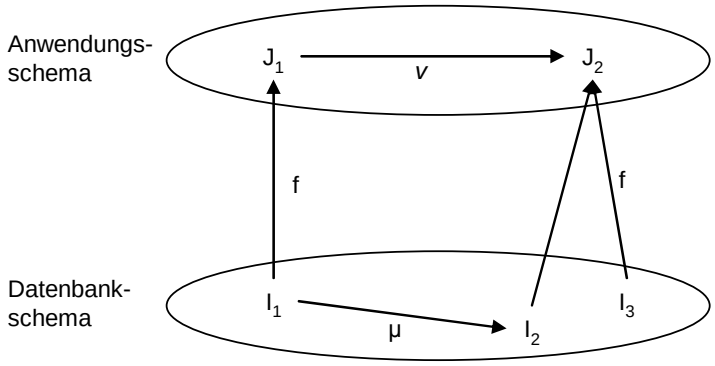


Abbildung 2.8: *Die Umsetzung nach* $\mathbf{I_2}$ *wird der nach* $\mathbf{I_3}$ *aufgrund der aufgestellten Kriterien vorgezogen*

gen Umsetzungen vorhanden ist, bei der aber kein Kriterium für eine Auswahl einer der Operationen aus dieser Menge angegeben werden kann. In Beispiel 2.3 ist beim Einfügen eines Tupels im Anwendungsschema bei einer Umsetzung in eine Datenbankoperation der Wert für das Erscheinungsjahr des einzufügenden Buchs grundsätzlich frei wählbar. Somit gibt es eine Menge möglicher Umsetzungen, bei denen jeweils die Jahreszahl variiert. Eine Wahl einer dieser Umsetzungen kann aber nicht getroffen werden, da kein Kriterium existiert, das eine der Varianten den anderen vorzieht. Die Forderung nach Arterhaltung bei einer Umsetzung wird in diesem Ansatz als natürlich angesehen. Es gibt aber durchaus Umsetzungen, bei denen die Art der Operation nicht erhalten wird und die ebenfalls zum gewünschten Ergebnis führen, wie in Beispiel 2.3 zu sehen ist. Wie die Autoren von [DB78] selbst bemerken, sind auf Basis des definierten Kriterienkatalogs nur in wenigen Situationen Änderungsoperationen überhaupt umsetzbar.

Da die eindeutige Bestimmbarkeit einer Umsetzung ein schwer zu erfüllendes Kriterium darstellt, das dazu führt, dass viele Operationen nicht ausgeführt werden können, weil eine vorhandene Mehrdeutigkeit nicht aufgelöst werden kann, schlägt [Mas84] auf Basis eines ähnlichen Kriterienkatalogs vor, den Benutzer oder Abbildungsersteller im Fall von Mehrdeutigkeiten eine der möglichen Umsetzung auswählen zu lassen.

Diese Vorgehensweise involviert naturgemäß einen menschlichen Anwender in die Entscheidung und führt somit zu keinem automatischen Berechnungsverfahren, das zu einer gültigen Umsetzung führt.

Einen abstrakteren Blickwinkel nehmen [BS81] ein, indem sie durchführbare Änderungsoperationen ν auf dem Anwendungsschema zu einer Menge U zusammenfassen,

- bei der die Verkettung zweier Operationen ν_1 und ν_2 wieder als Operation in U vorhanden ist: $\nu_1, \nu_2 \in U \Rightarrow \nu_1 \circ \nu_2 \in U$
- und bei der es zu jeder Operation eine weitere Operation gibt, welche die durchgeführte Änderung zurücknimmt: $\forall \mathbf{J} \in dom(\mathbf{T}) \forall \nu_1 \in U \exists \nu_2 \in U : \nu_2(\nu_1(\mathbf{J})) = \mathbf{J}$

Eine Änderungsoperation, die keinerlei Effekt besitzt wie in Beispiel 2.4, erfüllt aber auch die beiden Kriterien und macht dabei wohl nicht das, was sie sollte. Des Weiteren fallen die Operationen mit Seiteneffekten aus Beispiel 2.5 ebenfalls in diese Menge, da diese Operationen verkettet und auch zurückgenommen werden können. Sie führen aber wie gesehen zu unerwünschten Seiteneffekten.

Zusammenfassend betrachtet, weisen bestehende Kriterienkataloge zur Bestimmung von ausführbaren Operationen einige Lücken auf. Es werden eigentlich durchführbare Operationen abgelehnt, bei der Suche nach einer Umsetzung ist die Unterstützung eines Anwenders notwendig oder Operationen werden trotz unerwünschter Seiteneffekte als ausführbar akzeptiert.

2.2.3 Semantisch korrekte Operationen

Bewirkt die Durchführung einer Änderungsoperation auf einer Instanz des Bildbereichs genau den Effekt, der von ihr erwartet wird, und nicht mehr oder weniger Änderungen, dann ist diese Operation semantisch korrekt. Sie bewirkt genau das, was in ihrer Definition angegeben wird, so dass das in der Definition angegebene Ergebnis mit dem Resultat der Durchführung übereinstimmt.

Bei der Ausführung einer Operation dürfen somit keine zusätzlichen Korrekturmaßnahmen notwendig sein, um ausgehend von einem konsistenten Systemzustand wieder einen konsistenten Systemzustand zu erreichen. Korrekturen sind nicht notwendig, wenn das Ergebnis der Operation wieder zum Bildbereich der Abbildung gehört. Eine Operation ist dann seiteneffektfrei.

Definition 2.1 (*Seiteneffektfrei*)
Sei $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ und $\mathbf{J_1} \in range(f)$. Eine Operation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ auf dem Anwendungsschema $\mathbf{T}$ der Abbildung f ist *seiteneffektfrei (durchführbar)*, wenn die Ergebnisinstanz $\mathbf{J_2}$ zum Bildbereich von f gehört: $\mathbf{J_2} \in range(f)$.

Eine seiteneffektfreie Änderungsoperation führt von einer Instanz des Bildbereichs von f zu einer anderen Instanz des Bildbereichs und somit von einem konsistenten Systemzustand zum nächsten. In Beispiel 2.5 gehört bei der Abbildung *AutorOrt* die bei der Löschoperation angegebene Ergebnisinstanz zwar zum Anwendungsschema, jedoch nicht zum Bildbereich der Abbildung, so dass sich Seiteneffekte ergeben. Bei einer seiteneffektfreien Operation ist sichergestellt, dass sie ohne Seiteneffekte durchgeführt werden kann, wenngleich noch nicht gesagt ist, wie eine zu ihr passende Umsetzung aussieht. Es ist nur die Existenz mindestens einer Umsetzung gewährleistet, da es mindestens eine Datenbankinstanz gibt, welche durch die Abbildung auf das Ergebnis der Operation abgebildet wird.

Das eigentliche Problem beim Vorhandensein mehrerer möglicher Umsetzungen wie in Abb. 2.4 ist nicht, dass es auf dem Datenbankschema zwei Operationen μ_1 und μ_2 gibt, sondern dass diese zu zwei unterschiedlichen Datenbankinstanzen $\mathbf{I_2}$ und $\mathbf{I_3}$ führen, die beide durch f auf dieselbe Anwendungsinstanz $\mathbf{J_2}$ abgebildet werden. Anstatt nun

Kriterien für eine Auswahl einer der Operationen μ_1 oder μ_2 zu definieren, ist die in diesem Ansatz verfolgte Idee, zu fordern, dass durch die Abbildung jeder Anwendungsinstanz höchstens eine Datenbankinstanz zugewiesen wird. Soll im Anwendungsschema durch eine Operation zu einer bestimmten Instanz gewechselt werden, legt eine solche Abbildung dann eindeutig fest, welche dazu passende Datenbankinstanz zu wählen ist.

Definition 2.2 (*Eindeutigkeit*)
Sei $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ und $\mathbf{J_1} \in range(f)$. Eine Operation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ auf dem Anwendungsschema $\mathbf{T}$ der Abbildung f ist *eindeutig (durchführbar)*, wenn es im Datenbankschema $\mathbf{S}$ genau eine Instanz $\mathbf{I} \in dom(\mathbf{S})$ gibt, die der Ergebnisinstanz der Operation durch f zugeordnet wird: $f(\mathbf{I}) = \mathbf{J_2}$.

Bei einer eindeutigen Operation steht fest, welche Datenbankinstanz eine mögliche Umsetzung als Ergebnis erreichen muss. Auf welchem Wege diese Datenbankinstanz erreicht wird, kann dem System überlassen werden, das aus mehreren vorhandenen Operationen μ_i eine passende Operation auswählen kann. Die Einfügeoperation in Beispiel 2.3 mit der Abbildung $Buch_{ISBN,Titel}$ ist nicht eindeutig, da der Ergebnisinstanz der Anwendungsoperation mehr als eine Datenbankinstanz zugewiesen wird.

Gelten beide Eigenschaften bei einer Operation, dann bewirkt ihre Ausführung den durch ihre Definition beschriebenen Effekt und bei einer Ausführung steht fest, wie die Datenbank nachzuführen ist. Die Operation ist korrekt ausführbar. Verwendet man die Definition der Seiteneffektfreiheit und der Eindeutigkeit, kann der Begriff der semantischen Korrektheit formal gefasst werden.

Definition 2.3 (*Semantisch korrekt*)
Sei $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ und $\mathbf{J_1} \in range(f)$. Eine Änderungsoperation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ auf dem Anwendungsschema $\mathbf{T}$ der Abbildung f ist *semantisch korrekt (durchführbar)*, wenn ν sowohl seiteneffektfrei als auch eindeutig ist.

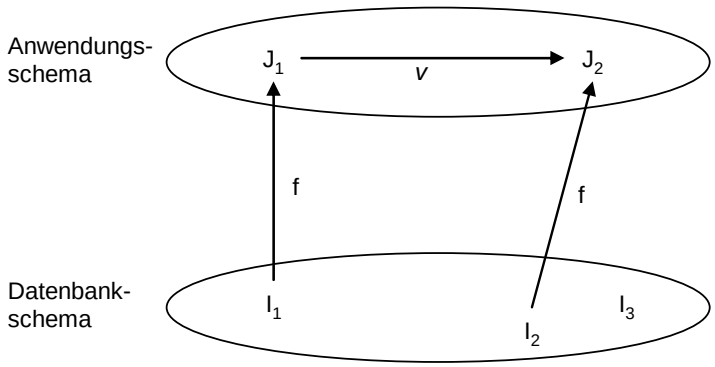


Abbildung 2.9: *Die Operation ν ist semantisch korrekt*

Anders ausgedrückt, gibt es für eine semantisch korrekte Änderungsoperation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ genau eine Instanz $\mathbf{I_2}$, für die $\nu(f(\mathbf{I_1})) = f(\mathbf{I_2})$ gilt mit $\mathbf{J_1} = f(\mathbf{I_1})$, $\mathbf{J_2} = f(\mathbf{I_2})$, $\mathbf{J_1} \neq \mathbf{J_2}$ und $\mathbf{I_1}, \mathbf{I_2} \in dom(\mathbf{S})$. Abb. 2.9 zeigt eine solche Konstellation für die Operation ν, die somit semantisch korrekt ist.

Gibt es eine Menge von Instanzen, von denen alle im Bildbereich der Abbildung liegen, und ist jeder Instanz eindeutig eine Datenbankinstanz zugewiesen, dann sind alle Operationen zwischen den Instanzen dieser Menge semantisch korrekt. Die zugehörige Abbildung ist eine Eins-zu-eins-Zuordnung zwischen den Instanzen ihres Definitionsbereichs und ihres Bildbereichs. Die Abbildung ist eine informationserhaltende Sicht auf den Daten.

Definition 2.4 (*Informationserhaltende Sicht*)
Eine Abbildung ist eine *informationserhaltende Sicht*, wenn sie eine Eins-zu-eins-Zuordnung zwischen Instanzen ihres Definitions- und ihres Bildbereichs beschreibt.

Abb. 2.10 zeigt eine informationserhaltende Sicht f zwischen einem Datenbankschema und einem Anwendungsschema mit jeweils vier Instanzen. Die Abbildung ordnet der Instanz $\mathbf{I_1}$ eindeutig die Instanz $\mathbf{J_1}$ zu, der Instanz $\mathbf{I_3}$ eindeutig die Instanz $\mathbf{J_2}$ und der Instanz $\mathbf{I_4}$ eindeutig die Instanz $\mathbf{J_4}$. Die Instanz $\mathbf{I_2}$ wird nicht abgebildet und $\mathbf{J_3}$ nicht durch die Abbildung erreicht. f beschreibt eine Eins-zu-eins-Zuordnung zwischen den Instanzen des Definitions- und des Bildbereichs. Alle Änderungsoperationen zwischen $\mathbf{J_1}$, $\mathbf{J_2}$ und $\mathbf{J_4}$ sind semantisch korrekt, da alle diese Instanzen im Bildbereich liegen und ihnen eindeutig eine Datenbankinstanz zugewiesen wird. Operationen nach $\mathbf{J_3}$ sind nicht durchführbar, da es für diese Instanz keine passende Datenbankinstanz gibt.

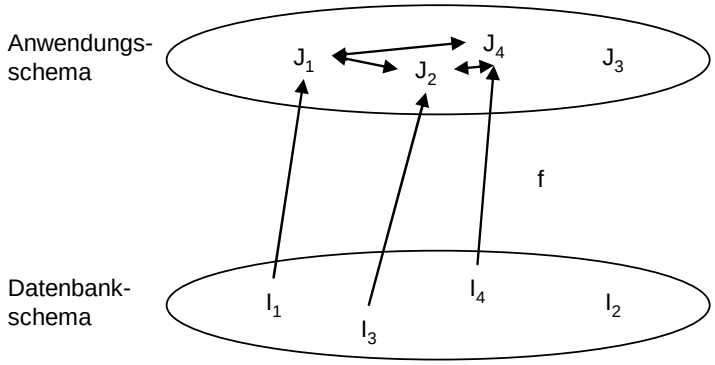


Abbildung 2.10: *Informationserhaltende Sicht*

Die auf einem Anwendungsschema definierten Operationen können danach unterschieden werden, ob sie semantisch korrekt sind oder nicht. Erfüllen sie die beiden genannten Bedingungen, dann können sie durchgeführt werden und ihre Wirkung entspricht der erwarteten. Erfüllen sie die Bedingungen nicht, steht nicht eindeutig fest, wie sie durch-

zuführen sind, oder eine Durchführung würde Seiteneffekt auslösen, so dass folglich ihre Ausführung abgelehnt wird. Folgt man der Definition für semantische Korrektheit, können die auf einem Anwendungsschema definierten Änderungsoperationen $\nu(\mathbf{J_1}) := \mathbf{J_2}$ mit $\mathbf{J_1} \in range(f)$ in die in Abb. 2.11 angegebene Klassifikation eingeteilt werden.

Eigenschaft einer Operation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ *mit* $\mathbf{J_1} \in range(f)$	*Klasse*
Nicht seiteneffektfrei $(\nu(\mathbf{J_1}) \notin range(f))$	nicht durchführbar
Seiteneffektfrei $(\nu(\mathbf{J_1}) \in range(f))$ und	
Nicht eindeutig $(\exists \mathbf{I_1}, \mathbf{I_2} \in dom(\mathbf{S}) : \nu(\mathbf{J_1}) = f(\mathbf{I_1}) = f(\mathbf{I_2}))$	nicht durchführbar
Eindeutig $(\exists_1 \mathbf{I_1} \in dom(\mathbf{S}) : \nu(\mathbf{J_1}) = f(\mathbf{I_1}))$	durchführbar

Abbildung 2.11: *Klassifikation von Anwendungsoperationen* $\nu(\mathbf{J_1}) := \mathbf{J_2}$ *mit* $\mathbf{J_1} \in range(f)$

Eine Operation wird auf semantische Korrektheit getestet, indem die beiden Einzelkriterien – Seiteneffektfreiheit und Eindeutigkeit – anhand der Abbildungsbeschreibung überprüft werden. Das Kriterium für einen Test auf Seiteneffektfreiheit ist die Frage, ob durch die Operation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ ausgehend von einer Instanz des Bildbereichs wieder eine Instanz innerhalb des Bildbereichs der Abbildung oder eine Instanz außerhalb des Bildbereichs erreicht wird. Liegt die Instanz außerhalb, ist die Operation nicht ohne Seiteneffekte ausführbar, da Korrekturmaßnahmen notwendig sind. Liegt sie innerhalb, ist die Seiteneffektfreiheit erfüllt und die Eindeutigkeit gibt den Ausschlag auf Korrektheit.

Um eine Aussage über die Seiteneffektfreiheit treffen zu können, muss also überprüft werden, ob $\nu(\mathbf{J_1}) \in range(f)$ gilt. Da der Bildbereich $range(f) = \{f(\mathbf{I}) \mid \mathbf{I} \in dom(\mathbf{S})\}$ nur durch die Abbildung selbst beschrieben wird, müssen sämtliche Instanzen $\mathbf{I} \in dom(\mathbf{S})$ des Datenbankschemas durch die Abbildung f abgebildet werden und es muss getestet werden, ob ein Bild $f(\mathbf{I})$ und das Ergebnis der Änderungsoperation übereinstimmen, d.h. ob für irgendein $\mathbf{I}$ gilt $\nu(\mathbf{J_1}) = f(\mathbf{I})$. Da im Regelfall ein Datenbankschema eine Vielzahl von Instanzen beschreibt, ist diese Vorgehensweise ein mühseliger Weg.

Ist eine Beschreibung des Bildbereichs durch ein Schema $\mathbf{T_f}$ mit $range(f) = dom(\mathbf{T_f})$ möglich, vereinfacht sich der Test auf eine Überprüfung auf das Enthaltensein des Operationsergebnisses in der Instanzenmenge dieses Schemas: $\nu(\mathbf{J_1}) \in dom(\mathbf{T_f})$. Im Folgenden ist die Idee, für eine Abbildung ein *Zielschema* $\mathbf{T_f}$ zu fordern, das exakt die Werte des Bildbereichs als Instanzenmenge umfasst. Ist das Ergebnis der Operation Instanz des Zielschemas $\mathbf{T_f}$ der Abbildung, ist die Operation seiteneffektfrei, ansonsten nicht. In Abb. 2.12 umfasst der Bildbereich die drei Instanzen $\mathbf{J_1}$, $\mathbf{J_2}$ und $\mathbf{J_4}$ nicht aber $\mathbf{J_3}$, welche aber wie die drei anderen ebenfalls zum Anwendungsschema gehört. Benötigt wird also ein Schema, dass $\mathbf{J_1}$, $\mathbf{J_2}$ und $\mathbf{J_4}$ beschreibt, um die Seiteneffektfreiheit zu überprüfen.

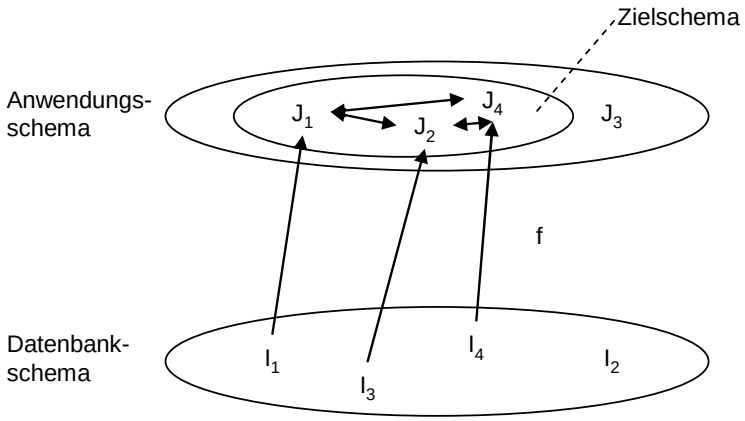


Abbildung 2.12: *Zielschema*

Ob eine Operation eindeutig ist, kann mittels der Abbildungsdefinition bestimmt werden. Ordnet die Abbildung einer einzelnen Anwendungsinstanz zwei oder mehr Datenbankinstanzen zu, gibt es nicht eindeutige Anwendungsoperationen; nämlich diejenigen, die in diesen Anwendungsinstanzen resultieren. In diesem Fall muss bei einer Durchführung einer Operation, die eine solche Anwendungsinstanz als Ergebnis besitzt, aus einer Menge von möglichen Umsetzungen, die jeweils andere Datenbankinstanzen erreichen können, eine ausgewählt werden. Da für eine Auswahl aber zumeist keine geeigneten Kriterien angegeben werden können, findet die Auswahl nicht statt und die Operationsausführung wird abgelehnt, obwohl passende Umsetzungen vorhanden sind.

Wird die Zuordnung mehrerer Datenbankinstanzen zur selben Anwendungsinstanz bei einer Abbildungsdefinition vermieden, ist automatisch sichergestellt, dass nur noch eindeutige Operationen auftreten. Mathematisch ausgedrückt muss die Abbildung f hierfür injektiv sein, d.h. für die Abbildung und zwei beliebige Datenbankinstanzen $\mathbf{I_1}$ und $\mathbf{I_2}$ gilt $f(\mathbf{I_1}) = f(\mathbf{I_2}) \Rightarrow \mathbf{I_1} = \mathbf{I_2}$. Die Abbildung f aus Bild 2.12 ist injektiv, da keiner der Bildinstanzen $\mathbf{J_1}$, $\mathbf{J_2}$ und $\mathbf{J_4}$ mehr als eine Datenbankinstanz zugewiesen wird.

Um zu testen, ob eine Operation semantisch korrekt ist, ergeben sich als Anforderungen an eine Abbildungsbeschreibung folglich,

- dass bei jeder Abbildung ein Zielschema vorhanden sein muss, um die Seiteneffektfreiheit der Operationen zu testen, und
- dass eine Abbildung auf Injektivität getestet werden muss, um die Eindeutigkeit der Operationen zu bestimmen.

In den nächsten Abschnitten werden die Anforderungen an die Komponenten einer Abbildungsbeschreibung bestimmt, deren Erfüllung sicherstellt, dass bei jeder formulierbaren Abbildung ein Zielschema vorhanden ist und bei jeder Abbildung deren Eigenschaften überprüfbar sind.

2.3 Schemata

Wird bei jeder Abbildung ein Zielschema gefordert, muss dieses zum einen anhand einer gegebenen Abbildung ermittelt werden können und zum anderen muss das Datenmodell seine Modellierung erlauben. Die Existenz eines solchen Schemas hängt davon ab, welche Wertemengen eine Abbildung erzeugen kann und welche Wertemengen durch ein Schema modelliert werden können. Dieser Abschnitt zeigt die Eigenschaften von Abbildungssprachen auf, welche die Existenz eines Zielschemas verhindern können.

2.3.1 Existenz eines Zielschemas

Um die Seiteneffektfreiheit einer Operation zu überprüfen, wird ein Zielschema benötigt, das $range(f)$ beschreibt. Im Gegensatz zum Anwendungsschema, welches bei der Abbildungsdefinition angegeben wird, wird das Zielschema allein durch die Abbildung bzw. deren Bildbereich bestimmt und nicht explizit vorgegeben. Es muss daher aus der Abbildungsdefinition abgeleitet werden, was allerdings nicht immer gelingen muss.

Beispiel 2.6 (*Zielschema nicht vorhanden*)
Eine Möglichkeit, Abbildungen innerhalb des semistrukturierten Datenmodells XML zu definieren, ist die Verwendung von XQuery [BCF+05] als Abbildungssprache. Will man die Mengen der Eingabe- und Ausgabewerte einer Abbildung beschreiben, bietet sich als Schemabeschreibungssprache etwa DTD [BPSM+04] an.

Folgen die Datenbankinstanzen der DTD

```
<!ELEMENT autoren (autor*)>
<!ELEMENT autor (autorenname, institutsname, isbn)>
<!ELEMENT autorenname (#PCDATA)>
<!ELEMENT institutsname (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
```

so enthalten dazu konforme XML-Dokumente eine Liste von Autoren, wobei bei jedem Autor sein Name, das Institut, bei dem er tätig ist, und die ISBN, eines von ihm verfassten Buchs angegeben ist. Das XML-Dokument

```
<autoren>
   <autor>
     <autorenname>S. Maier</autorenname>
     <institutsname>I.N.R.I.A.</institutsname>
     <isbn>0-201-53781-X</isbn>
   </autor>
   <autor>
     <autorenname>P. Hill</autorenname>
     <institutsname>I.N.R.I.A.</institutsname>
     <isbn>1-55867-622-X</isbn>
```

```
    </autor>
    <autor>
      <autorenname>R. Scholl</autorenname>
      <institutsname>Bell Labs</institutsname>
      <isbn>0-201-53781-X</isbn>
    </autor>
</autoren>
```

ist zur angegebenen DTD konform, umfasst die Daten von drei Autoren und dient im Folgenden als Datenbankinstanz.

Eine in XQuery formulierte Abbildung ist die Anfrage

```
<ergebnis> {
   for $x in //autoren/autor/isbn
     return <a> { $x/text() } </a>
   } {
   for $y in //autoren/autor/isbn
     return <b> { $y/text() } </b>
   }
</ergebnis>
```

Die Anfrage iteriert über die bei den Autoren hinterlegten ISBN-Nummern durch die Angabe des Pfades `//autoren/autor/isbn` in den `for`-Klauseln, verpackt in jeder `return`-Klausel den Text des jeweils ausgewählten `isbn`-Elements in ein `a`- bzw. ein `b`-Element und kombiniert diese zu einem Ergebnisdokument.

Mit dem obigen Beispieldokument als Eingabe entsteht durch die Abbildung das folgende XML-Dokument als Ergebnis:

```
<ergebnis>
   <a>0-201-53781-X</a>
   <a>1-55867-622-X</a>
   <a>0-201-53781-X</a>
   <b>0-201-53781-X</b>
   <b>1-55867-622-X</b>
   <b>0-201-53781-X</b>
</ergebnis>
```

Durch die Semantik von XQuery [DFF+05] werden auch Regeln vorgegeben, mit denen aus einer vorhandenen Abbildungsdefinition ein Schema abgeleitet werden kann, welches die von der Abbildung erzeugten Werte als Instanzen besitzt. Ein solches Schema wird auf Basis der in der Abbildung angegebenen XML-Elemente und der `return`-Klauseln abgeleitet. Die Beispielabbildung definiert ein `ergebnis`-Element als umschließendes Element für die Ausgabe. Die beiden `return`-Klauseln erzeugen jeweils beliebig viele `a`- und

b-Elemente, da die Anzahl der Autoren auf Eingabeseite durch das Datenbankschema nicht nach oben beschränkt ist. Aus der Abbildungsdefinition wird somit das Schema

```
<!ELEMENT ergebnis (a*, b*)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>
```

abgeleitet. Jede Bildinstanz der Anfrage ist zu dieser DTD konform.

Die erzeugte DTD beschreibt zwar die Werte des Bildbereichs, verwendet hierzu aber die Beschreibung einer echten Obermenge der Bildinstanzen. In einem durch die Abbildung erzeugten Dokument stimmt die Anzahl der a-Elemente immer mit der Anzahl der b-Elemente überein, da diese Elemente aus denselben isbn-Eingabeelementen erzeugt werden. Enthält das Eingabedokument n ISBN-Einträge, entstehen im Ausgabedokument genau n a- und genau n b-Elemente. Somit müsste das Schema zur Beschreibung des Bildbereichs eigentlich

<!ELEMENT ergebnis (a^n, b^n)> mit $n \geq 0$
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>

lauten, das offensichtlich nicht mit dem aus der Abbildungsdefinition abgeleiteten Schema übereinstimmt. Allerdings stellt dies keine gültige DTD dar, da mit Hilfe einer DTD nur Dokumente beschrieben werden können, bei denen die Reihenfolge von Elementen durch eine reguläre Sprache formuliert werden kann. Die durch den Ausdruck ergebnis(a^{n}b^n) gegebene Sprache, welche den Aufbau von Instanzen des Bildbereichs beschreibt, gehört hier nicht dazu.

Folglich ist die aus der Abbildung abgeleitete DTD ein Anwendungsschema, aber kein Zielschema, und das Zielschema kann nicht innerhalb des Datenmodells modelliert werden, da die Abbildung eine Menge von Werten erzeugt, die nicht durch ein Schema erfasst werden kann. □

Abbildungssprachen sind in der Regel so angelegt, dass die Ergebniswerte einer Abbildung in dem Datenmodell ausgedrückt werden können, für das die Abbildungssprache formuliert wird. So erzeugt XQuery immer XML-Dokumente und die relationale Algebra liefert Relationen. Folglich ist für eine Abbildung ein Anwendungsschema innerhalb des entsprechenden Datenmodells immer definierbar, wobei das Anwendungsschema die Ausgabewerte umfasst. Obwohl eine Abbildung dann eine Zuordnung von Instanzen zwischen zwei Schemata – Datenbankschema und Anwendungsschema – beschreibt, bedeutet dies aber nicht automatisch, dass auch ein Zielschema abgeleitet bzw. angegeben werden kann. Wie gesehen kann bei einer gegebenen Abbildung f gelten, dass ein Anwendungsschema $\mathbf{T}$ vorhanden ist, aber kein Zielschema $\mathbf{T_f}$ existiert:

$$\begin{aligned} \exists \mathbf{T} &: range(f) \subset dom(\mathbf{T}) \\ \nexists \mathbf{T_f} &: range(f) = dom(\mathbf{T_f}) \end{aligned}$$

Das Problem ist hierbei, dass die Menge an Werten, die durch eine Abbildung erzeugt wird, nicht durch ein Schema innerhalb des Datenmodells beschrieben werden kann.

2.3.2 Verwandte Arbeiten

Ob innerhalb eines Datenmodells ein Zielschema bestimmt werden kann, hängt vom betrachteten Datenmodell und der verwendeten Abbildungssprache ab. Das Modell definiert, welche Wertemengen ein gültiges Schema beschreiben kann, und die Sprache legt fest, welche Wertemengen durch Abbildungen erzeugt werden können.

Beim relationalen Datenmodell [Cod70, AHV95] werden Abbildungen innerhalb dieses Datenmodells mit Hilfe der Operatoren der relationalen Algebra formuliert. Die Operatoren sind so definiert, dass bei jedem Operator als Ergebnis wieder eine Relation entsteht, für die ein Relationstyp angegeben werden kann. Enthält ein Schema diesen Typ, kann es als Anwendungsschema verwendet werden.

Beim relationalen Modell ist ein Anwendungsschema immer vorhanden bzw. aus der Abbildung ableitbar, aber ein Zielschema für eine Abbildung kann in den meisten Fällen nicht angegeben werden. In Beispiel 2.2 wird eine Abbildung durch eine Selektion erzeugt und ein Anwendungsschema mit dem passenden Relationstyp angegeben. Aufgrund der Selektion kommen nur Tupel, welche das spezifizierte Selektionskriterium erfüllen, in einer Instanz im Bildbereich vor. Bei Änderungsoperationen auf dem Anwendungsschema können aber auch Tupel eingefügt werden, welche die Selektionsbedingung nicht erfüllen, da anhand des Relationstyps nicht überprüft werden kann, welche Kriterien ein einfügbares Tupel erfüllen muss. Mit Hilfe gängiger Konsistenzbedingungen wie funktionalen Abhängigkeiten können solche Kriterien auf dem Anwendungsschema nicht ausgedrückt werden. Hierzu wäre eine Erweiterung des relationalen Modells um speziellere Konsistenzbedingungen notwendig, um etwa die durch eine Selektion erzwungenen Bedingungen zu formulieren. Wenn eine Sicht aus Projektionen auf Datenbankrelationen mit funktionalen Abhängigkeiten besteht, kann die Existenz eines Zielschemas ebenfalls nicht garantiert werden, wie [Heg84] zeigt.

Objektorientierte Modelle wie [Clu98, CB00] gestatten die Modellierung von Objekten und deren strukturellem Aufbau. Zur Definition von Abbildungen wird zumeist eine Erweiterung von SQL aus dem relationalen Modell verwendet wie etwa OQL [CB00]. Die Abbildungssprache wird im Vergleich zu SQL dahingehend ergänzt, dass zusätzlich Pfadausdrücke innerhalb von Anfragen verwendet werden können, um über Referenzen zwischen Objekten zu navigieren, und der Aufruf von Objektmethoden in einer Anfrage gestattet wird.

Da eine ähnliche Abbildungssprache wie im relationalen Modell verwendet wird, ist hier die Festlegung eines Anwendungsschemas für eine Abbildung in den meisten Fällen möglich. Problematisch wird die Bestimmung eines Anwendungsschemas, wenn die Abbildung neue Objekte erzeugen kann und diese Objekte einer Klasse zugewiesen werden müssen. Hier ist ein menschlicher Eingriff notwendig, um diese Einordnung vorzunehmen, da nicht nur die Schnittstelle des Objekts für eine Klassifizierung ausschlaggebend ist,

sondern auch die dahinter stehende Semantik. Bei objektorientierten Modellen ergeben sich also schon bei der Angabe eines passenden Anwendungsschemas Schwierigkeiten, die bei der Herleitung eines Zielschemas dann ebenfalls auftreten. Hinzu kommen die Problemfälle aus dem relationalen Modell, dass etwa Domänen mit Einschränkungen aus Selektionsbedingungen auf dem Anwendungsschema nicht formuliert werden können.

Semistrukturierte Modelle wie XML [BPSM+04] modellieren ihre Daten in Form einer geschachtelten Baumstruktur. Für derartige Modelle existiert eine vielfältige Auswahl an Abbildungssprachen, zu denen etwa XQuery [BCF+05] und XSLT [Kay05] für die Abbildung von XML-Dokumenten zählen.

Bei semistrukturierten Modellen tritt der Fall ein, dass ein Anwendungsschema in der Regel zwar abgeleitet werden kann, dieses aber eine echte, viel zu große Obermenge der Werte des Bildbereichs beschreibt, wie in Beispiel 2.6 gesehen. Es existieren Abbildungen, für die ein Zielschema innerhalb des Datenmodells nicht definierbar ist. Einige Untersuchungen [MS99, MSV00, PV00, Suc01, AMN+01, Toz01, Suc02, MN04] betrachten, ob zu einem gegebenen Schema getestet werden kann, ob dies den Bildbereich einer Abbildung beinhaltet, um enger umschriebene Anwendungsschemata zu erhalten. Doch es zeigt sich, dass nur unter Einschränkung der Abbildungssprache überhaupt überprüft werden kann, ob ein gegebenes Schema den Bildbereich einer Abbildung umfasst.

Bei den drei betrachteten Datenmodellen und Abbildungssprachen kann nicht garantiert werden, dass bei einer Abbildung ein Zielschema vorhanden ist. Es tritt der Fall ein, dass ein solches Schema innerhalb des entsprechenden Datenmodells überhaupt nicht modelliert werden kann. Betrachtet man etwa das relationale Datenmodell, entsteht bei einer Abbildung zwar wieder eine Relation, doch gibt es keinen Relationstyp, der nur die Menge der Bildrelationen umfasst. Ähnliches gilt bei objektorientierten und semistrukturierten Modellen. Es ist also häufig der Bildbereich einer Abbildung eine echte Teilmenge eines Anwendungsschemas.

2.3.3 Wohltypisierte Abbildungen

Eine Abbildungsbeschreibung muss gewährleisten, dass ein Zielschema aus der Abbildungsdefinition hergeleitet werden kann, und das Datenmodell muss die Formulierung eines solchen Schemas erlauben. Für die Bildwerte einer Abbildung f muss also ein Schema definierbar sein, das genau diese Wertemenge als Instanzen umfasst. Eine solche Abbildung ist wohltypisiert.

Definition 2.5 (*Wohltypisiert*)
Eine Abbildung $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ ist *wohltypisiert*, wenn es ein Zielschema $\mathbf{T_f}$ gibt, so dass $range(f) = dom(\mathbf{T_f})$.

Die Abbildung f in Abb. 2.13 ist wohltypisiert, wenn ein Schema existiert, welches $\mathbf{J_1}$ und $\mathbf{J_2}$ als Instanzen besitzt.

Da es ein Ziel ist, bei allen erlaubten Abbildungen auch Aussagen über die semantisch korrekten Operationen machen zu können, muss somit bei jeder definierbaren Abbildung

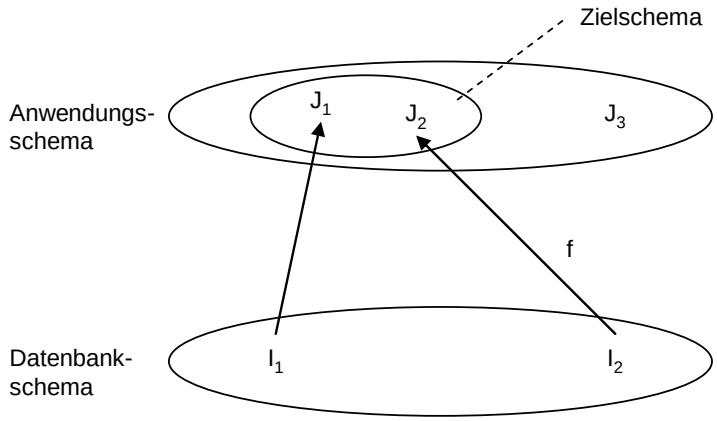


Abbildung 2.13: *Wohltypisierte Abbildung*

auch ein Zielschema vorhanden sein. Es dürfen keine Abbildungen ausgeschlossen sein.

Als Anforderung an eine Abbildungsbeschreibung ergibt sich,

- dass jede Abbildung genau ein Zielschema besitzen muss, um die Seiteneffektfreiheit zu überprüfen, und somit wohltypisiert ist.

In den bisherigen Beispielen sind zwei Ursachen aufgetreten, die eine Formulierung eines Zielschemas verhindern. Bei der Selektion in Beispiel 2.2 wird durch die Selektionsbedingung die Domäne des ISBN-Attributs auf Anwendungsseite auf genau einen Wert eingeschränkt; nämlich auf den in der Selektionsbedingung angegebenen. Eine solche Domäne kann aber innerhalb des relationalen Modells nicht modelliert werden, so dass kein Zielschema vorhanden ist. Ist ein Datenmodell nicht auf vorgegebene Domänen beschränkt, sondern erlaubt die Modellierung beliebiger Wertemengen, ist auch eine passende Domäne für die Anwendungsseite einer solchen Abbildung modellierbar.

Bei der XML-Abbildung in Beispiel 2.6 tritt der Fall ein, dass auf einen Teilwert der Eingabe bei der Abbildung mehrfach zugegriffen wird und somit dieser mehrfach abgebildet wird. Aus einem einzelnen `isbn`-Element des Eingabedokuments wird durch die erste `for`-Klausel ein `a`-Element und durch die zweite `for`-Klausel ein `b`-Element in der Ausgabe, d.h. ein und dasselbe `isbn`-Element wird zweifach abgebildet. Durch derartige Mehrfachzugriffe werden auf der Anwendungsseite Abhängigkeiten zwischen Teilwerten impliziert, die dort nicht ausgedrückt werden können. Im Fall des Beispiels müssen in einer Anwendungsinstanz zum einen immer genauso viele `a`- wie `b`-Elemente vorhanden sein und zum anderen müssen diese immer paarweise dieselben Textwerte enthalten. Steht innerhalb des ersten `a`-Elements eine gewisse ISBN, muss diese Nummer auch innerhalb des ersten `b`-Elements auftauchen usw. Ändert sich eine ISBN, muss auch die Nummer innerhalb des Partnerelements geändert werden, um eine gültige Anwendungsinstanz zu erhalten. Konsistenzbedingungen, um diese Eigenschaften der Anwendungsin-

stanzen ausdrücken zu können, bietet das XML-Datenmodell nicht. Abb. 2.14 zeigt eine Mehrfachabbildung von Teilwerten auf einer abstrakteren Ebene, bei der einem Teilwert v der Eingabeinstanz zwei Werte w_1 und w_2 in der Ausgabeinstanz zugewiesen werden.

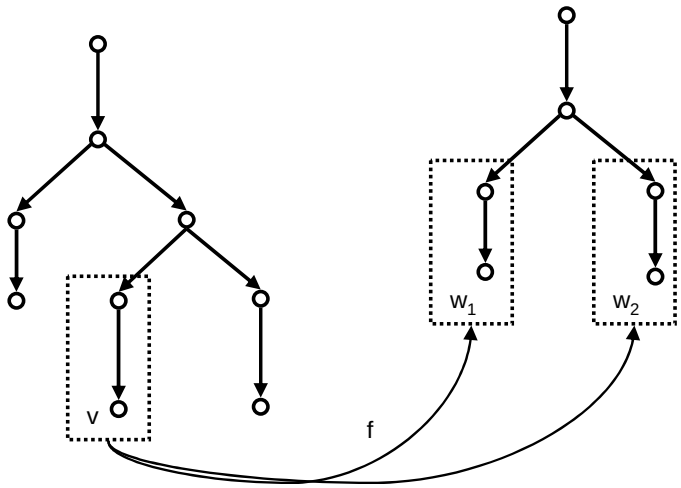


Abbildung 2.14: *Mehrfachabbildung*

Werden Mehrfachzugriffe auf Teilwerte bei einer Abbildung vermieden, entstehen keine durch sie verursachten Abhängigkeiten auf der Anwendungsseite und die Existenz eines Zielschemas wird nicht verhindert.

2.4 Instanzen

Mehrfachzugriffe können durch unterschiedliche Ursachen entstehen. Einmal kann das Datenmodell die Modellierung von Werten erlauben, bei denen über mehrere verschiedene Zugriffspfade innerhalb einer Instanz auf denselben Teilwert zugegriffen wird. Werden diese Pfade bei einer Abbildungsdefinition verwendet, ist anhand der Definition selbst nicht entscheidbar, ob über diese Pfade derselbe Teilwert erreicht wird oder nicht, da sich die Pfade ja unterscheiden. Das Datenmodell kann solche Mehrfachzugriffe auf einen Teilwert über unterschiedliche Pfade verbieten, indem bestimmte Anforderungen an den Aufbau von Werten gestellt werden, wobei auf Basis der Wertdarstellung auch Abbildungen formulierbar sein müssen.

2.4.1 Zugriffe auf Teilwerte von Instanzen

Eine Abbildung wird als Zuordnung zwischen den Instanzen zweier Schemata beschrieben, wobei Abbildungen so formuliert werden, dass Teilwerte von Instanzen einander

zugeordnet werden. Die Instanzen werden als zusammengesetzte Werte betrachtet und durch die Abbildung wird ein Teilwert einer Instanz einem Teilwert der anderen Instanz zugewiesen.

Beispiel 2.7 (*Werte auf Werte abbilden*)
Beschreibt ein objektorientiertes Anwendungsschema eine Klasse `Institut` mit zwei Attributen

```
class Institut {
   attribute string Institutsname;
   attribute string Ortsname;
}
```

und das relationale Datenbankschema einen Relationstyp

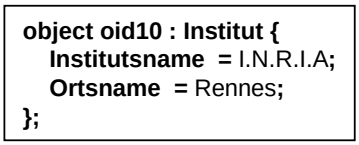
Institut [*Oid*, *Institutsname*, *Ortsname*]

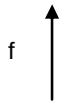
so kann durch die Definition einer geeigneten Abbildung festgelegt werden, dass aus einem einzelnen Tupel die Zustandsdaten eines Objekts abgeleitet werden. Die Tupelwerte beim relationalen Attribut *Oid* enthalten hierbei die impliziten eindeutigen Objektkennungen aus dem objektorientierten Modell.

Die Beschreibung einer Abbildung erfolgt nicht anhand von einzelnen Instanzen, sondern auf Basis der Schemata, indem mit deren Hilfe angegeben wird, welche Teilwerte der Datenbankinstanz auf welche Teilwerte der Anwendungsinstanz abgebildet werden. So wird im Beispiel der Klasse `Institut` der Relationstyp *Institut* zugewiesen und das Attribut `Institutsname` der Klasse wird dem Attribut *Institutsname* des Relationstyps zugeordnet. Entsprechendes gilt für das Attribut `Ortsname`. Wird ein Teilwert einer zugehörigen Relation über den Bezeichner des Relationstyps und einen Attributbezeichner adressiert, wird diesem Wert der Attributwert innerhalb eines Objekts zugeordnet, der über den Klassenbezeichner, Objektbezeichner und Attributbezeichner angesprochen wird. Das Datenbankschema enthält somit Tupel, aus dessen Attributwerten die Abbildung die Werte der Objektattribute berechnet (Abb. 2.15), und Teilwerte werden auf Teilwerte abgebildet.

□

Fasst man Schemainstanzen als zusammengesetzte Werte auf, so bestehen diese aus weiteren Werten, die einzeln abgebildet werden können. Eine Instanz eines relationalen Schemas besteht aus einer Menge von Relationen und diese wiederum enthalten jeweils eine Menge von Tupeln, die ihrerseits aus einer Menge von Attributwerten bestehen. Im Fall eines objektorientierten Schemas beinhaltet eine Instanz eine Reihe von Objektmengen, wobei ein einzelnes Objekt eine Menge von Attributen umfasst, deren Werte atomar oder wiederum zusammengesetzt sein können. Bei einem semistrukturierten Modell ergibt sich eine Schachtelungsstruktur automatisch durch das Schachteln der einzelnen Elemente. Bei einer Abbildung werden Teilwerte von Datenbankinstanzen den Teilwerten von Anwendungsinstanzen zugeordnet.





Oid	Institutsname	Ortsname
oid10	I.N.R.I.A	Rennes

Abbildung 2.15: *Zuordnung eines Tupels zu einem Objekt*

Unterscheiden sich das Anwendungsdatenmodell und das Datenbankdatenmodell, werden häufig nicht nur Daten aufeinander abgebildet, sondern auch Metadaten des einen Schemas als Daten des anderen interpretiert und umgekehrt.

Beispiel 2.8 (*Werte auf Metadaten abbilden*)
Eine Abbildung auf die Objektklasse aus Beispiel 2.7 kann auch spezifiziert werden, wenn eine Relation als

$$Institut\ [\ Oid,\ Attribut,\ Wert\ ]$$

vorhanden ist. Das Attribut *Oid* enthält wieder die Objektkennung. Die Spalten *Attribut* und $Wert$ modellieren die einzelnen Objektattribute und deren Werte, indem ein Tupelwert in der ersten der beiden Spalte den Bezeichner des Objektattributs und der Wert in der zweiten Spalte den Attributwert angibt. Der Zustand eines einzelnen Objekts wird wie in Abb. 2.16 dargestellt aus zwei Tupeln errechnet, wobei aus den Werten des Attributs *Attribut* in der Relation die Bezeichner der objektorientierten Attribute werden und somit Daten in Metadaten transformiert werden.

Es ist auch eine Abbildung möglich, bei welcher der Zustand des Objekts als Text kodiert vorliegt und das Objekt aus einem einzelnen Tupel $\langle$ oid10+I.N.R.I.A.+Rennes $\rangle$ einer Relation

$$Institut\ [\ Wert\ ]$$

generiert wird (Abb. 2.17). Wie der Text dekodiert wird, muss der Abbildungsberechnung bekannt sein, so dass im Beispiel etwa die „+“-Zeichen die Rolle von Trennzeichen zwischen den Attributwerten übernehmen.

□

Neben einer Zuordnung von Teilwerten zueinander können bei einer Abbildung auch Metadaten auf Daten abgebildet werden und umgekehrt Daten in Metadaten überführt

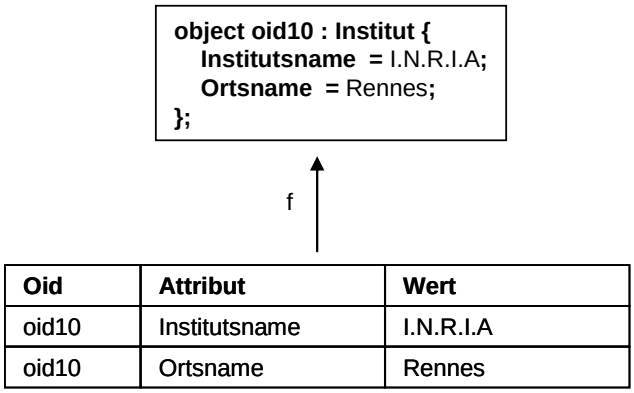


Abbildung 2.16: *Abbildung von zwei Tupeln auf ein Objekt*

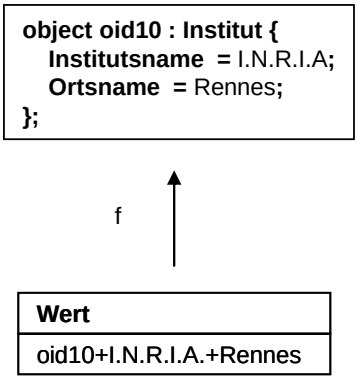


Abbildung 2.17: *Speicherung eines Objektzustands in Textform*

werden. In Beispiel 2.8 ist der Bezeichner eines Objektattributs im objektorientierten Modell Teil der Schemabeschreibung, im relationalen Schema wird daraus ein Wert eines Relationsattributs und gehört somit zur Instanz.

Damit ein einzelner Teilwert der Eingabeinstanz nicht mehrfach abgebildet wird, ist bei der Darstellung der Instanzen sicherzustellen, dass auf einen Teilwert nur über einen eindeutigen Zugriffspfad zugegriffen wird. Gängige Datenmodelle wie das relationale oder ein objektorientiertes Modell implizieren zumeist eine Schachtelungsstruktur auf einem Wert. Abstrahiert man von einem konkreten Datenmodell, ist eine häufig anzutreffende Variante, Daten in Form von Graph- oder Baumstrukturen zu repräsentieren, da diese Datenstrukturen einfach aufgebaut sind und sich alles irgendwie als Graph darstellen lässt. Ein Knoten des Graphen repräsentiert hier einen Teilwert und eine gerichtete Kante eine Enthaltensseinbeziehung zwischen Werten. Je nachdem wie ein Wert aufgebaut ist,

kann bei einer Abbildung auf einen Teilwert über unterschiedliche Zugriffspfade mehrfach zugegriffen werden.

Beispiel 2.9 (*Wertbasierter Zugriff*)
Betrachtet man XML, so müssen hier die Elemente im Kontext eines sie umgebenden Elements nicht eindeutig benannt sein. Im XML-Dokument

```
<autoren>
   <autor>
     <autorenname>S. Maier</autorenname>
     <institutsname>I.N.R.I.A.</institutsname>
     <isbn>0-201-53781-X</isbn>
   </autor>
   <autor>
     <autorenname>P. Hill</autorenname>
     <institutsname>I.N.R.I.A.</institutsname>
     <isbn>1-55867-622-X</isbn>
   </autor>
   <autor>
     <autorenname>R. Scholl</autorenname>
     <institutsname>Bell Labs</institutsname>
     <isbn>0-201-53781-X</isbn>
   </autor>
</autoren>
```

sind innerhalb eines `autoren`-Elements mehrere `autor`-Elemente vorhanden, die jeweils die Daten eines Autors beschreiben. Das gesamte Dokument repräsentiert somit eine Menge von Autoren, wobei die Elemente der Menge – die einzelnen Autoren und ihre Daten – nicht eindeutig sein müssen.

Da die Daten aller Autoren über den gleichen Pfad `//autoren/autor` angesprochen werden, wird ein Auswahlkriterium in Form einer Selektionsbedingung auf den Eigenschaften der Autoren angegeben, um einen oder mehrere der Autoren auszuwählen. Soll eine Abbildung die Institute aller Autoren in die Ausgabe übernehmen, die S. Maier heißen oder ein Buch mit der ISBN 0-201-53781 geschrieben haben, so sieht eine Anfrage in XQuery etwa folgendermaßen aus:

```
<ergebnis> {
   for $a in //autoren/autor
   where $a/autorenname/text() = ''S. Maier''
   return <institut> { $a/institutsname/text() } </institut>
   }{
   for $a in //autoren/autor
   where $a/isbn/text() = ''0-201-53781-X''
```

```
   return <institut> { $a/institutsname/text() } </institut>
   }
</ergebnis>
```

und als Ergebnis entsteht

```
<ergebnis>
   <institut>I.N.R.I.A.</institut>
   <institut>I.N.R.I.A.</institut>
   <institut>Bell Labs</institut>
</ergebnis>
```

Die Abbildung wählt einen Autor anhand seiner Eigenschaften aus, wobei durch die beiden `for`-Klauseln zweimal über die Menge der Autoren iteriert wird und einmal die Autoren mit dem angegebenen Namen ausgewählt werden und das andere Mal die Autoren, bei denen die spezifizierte ISBN vorhanden ist. Diese Abbildung impliziert eine Abhängigkeit auf dem Anwendungsschema, da der Text I.N.R.I.A. im ersten und im zweiten `institut`-Element der Ausgabe aus demselben Element der Eingabe entstanden ist. Wird in der Bildinstanz einer der beiden Werte geändert, muss der andere ebenfalls modifiziert werden.

Wählt man die Abbildung anders, tritt dieses Problem nicht auf:

```
<ergebnis> {
   for $a in //autoren/autor
   where $a/autorenname/text() = ''S. Maier''
   or $a/isbn/text() = ''0-201-53781-X''
   return <institut> { $a/institutsname/text() } </institut>
   }
</ergebnis>
```

und als Ergebnis entsteht

```
<ergebnis>
   <institut>I.N.R.I.A.</institut>
   <institut>Bell Labs</institut>
</ergebnis>
```

Im Gegensatz zur ersten Abbildung, die zweimal über die Menge der Autoren iteriert, läuft die zweite Abbildung nur einmal über alle Autoren. Im zweiten Fall wird jedes Element der Eingabe nur einmal angefasst. □

Beide Anfragen aus Beispiel 2.9 nehmen einen wertbasierten Zugriff auf die `autor`-Elemente vor, d.h. die Daten eines Autoren werden nicht über ihre Position im Dokument angesprochen, sondern über eine Selektionsbedingung ausgewählt. Auf alle Autorendaten wird über denselben Pfad `//autoren/autor` zugegriffen, so dass nur ein

wertbasierter Zugriff möglich ist, um einzelne Autoren aus der Menge auszuwählen (Abb. 2.18 links), d.h. sie werden anhand ihrer Eigenschaften ausgewählt. Treffen aber mehrere Auswahlkriterien innerhalb einer Abbildung auf dasselbe Mengenelement zu, so existieren mehrere Zugriffspfade auf dieses Element und es wird mehrfach abgebildet. In Beispiel 2.9 wird in der ersten Abbildung auf den ersten Autor sowohl über den Pfad `//autoren/autor [ autorenname = ''S.Maier'' ]` als auch über den Pfad `//autoren/autor [ isbn = ''0-201-53781-X'' ]` zugegriffen. Problematisch ist hier, dass anhand der ersten Abbildung oder der von ihr erzeugten Bildinstanz nicht ermittelt werden kann, ob die ersten beiden `institut`-Elemente der Ausgabe vom gleichen Eingabeelement erzeugt werden oder von zwei unterschiedlichen. Anstatt eines Autoren, der sowohl die Bedingung der ersten `for`-Klausel als auch die der zweiten `for`-Klausel erfüllt, kann es genau so gut ein Eingabedokument mit zwei unterschiedlichen Autoren geben, die jeweils eine Bedingung erfüllen und je eines der ISBN-Element der Ausgabe erzeugen. Somit ist auf Anwendungsseite nicht entscheidbar, ob die Gleichheit der ersten beiden `institut`-Elemente in der Ausgabe mit einer Konsistenzbedingung durchgesetzt werden muss oder nicht. Es ist anhand der Abbildungsdefinition nicht feststellbar, ob eine Mehrfachabbildung stattfindet oder nicht, da das Datenmodell mehrere Zugriffspfade auf ein und dasselbe Element der Eingabe zulässt. Ein Zielschema ist daher nicht definierbar.

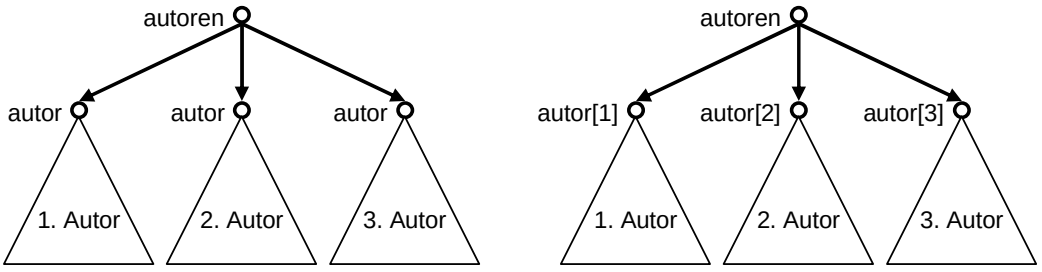


Abbildung 2.18: *Wertbasierter Zugriff vs. pfadbasierter Zugriff auf einzelne Autoren*

Werden Elemente einer Schachtelungsebene unterschiedlich benannt oder wird auf gleich benannten Elementen eine Ordnung definiert, kann über diesen eindeutigen Bezeichner bzw. die Position ein Einzelelement aus der Menge ausgewählt werden (Abb. 2.18 rechts). Im Beispiel ist der erste Autor des XML-Dokuments etwa über den Pfad `//autoren/autor[1]` adressiert, wenn seine Position innerhalb des Dokuments mit im Pfad angegeben wird. Da sich die Zugriffspfade pro Element dann unterscheiden, ist kein wertbasierter Zugriff mehr notwendig, da jedes Element über einen eindeutigen Pfad einzeln angesprochen werden kann. Werden bei einer Abbildungsdefinition nur solche Pfade verwendet, ist bei einer Abbildung feststellbar, ob ein Element mehrfach abgebildet wird; nämlich dann, wenn derselbe Pfad innerhalb der Abbildung zweimal verwendet wird.

Bei einem pfadbasierten Zugriff können allerdings auch Schwierigkeiten auftreten, wenn die Länge des Zugriffspfads nicht nach oben beschränkt ist.

Beispiel 2.10 (*Zugriff über Referenzen*)
Modelliert man einen Autoren und ein von ihm geschriebenes Buch als Graph, enthält dieser etwa die in Abb. 2.19 dargestellten Knoten und Kanten.

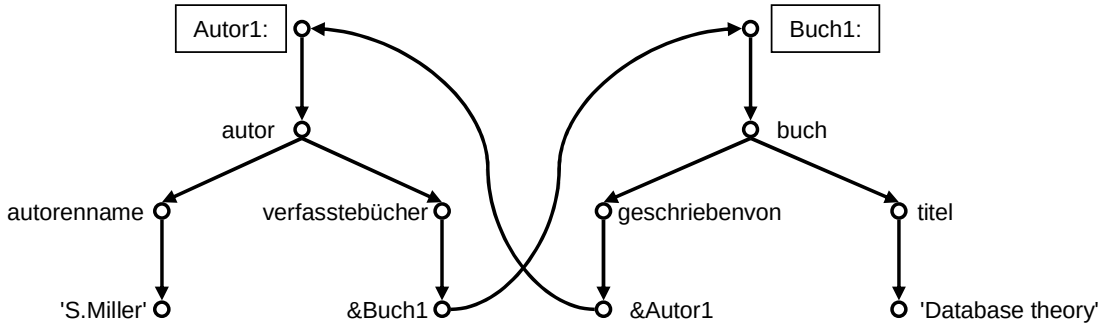


Abbildung 2.19: *Rekursive Daten*

Ein Autor wird durch den linken Teilbaum dargestellt. Er wird durch seinen Namen beschrieben und verweist auf die von ihm verfassten Bücher. Ein Buch als rechter Teilbaum besitzt einen Titel und referenziert seine Autoren. Eingerahmte Knotenbezeichner kennzeichnen jeweils die beiden Teilbäume eindeutig, so dass ein Autor durch die Angabe von `&Buch1` auf das Buch verweisen kann und umgekehrt das Buch über den `&Autor1` Eintrag zurück auf den Autoren verweist.

Wird auf einem solchen Graphen eine Abbildung formuliert, welche alle Daten zum gegebenen Autor ausgeben soll, so enthält das Ergebnis alle Pfade, die mit `//Autor1/autor` beginnen. Eine Schwierigkeit entsteht, weil die Anzahl dieser Pfade nach oben nicht beschränkt ist, da der Graph einen Zyklus enthält. Wird der `verfasstebücher`-Eintrag beim Autor ausgewertet, führt dieser zu `Buch1` und dessen `geschriebenvon`-Eintrag wieder zurück zu `Autor1` und dessen `verfasstebücher`-Eintrag wieder zu `Buch1` usw. Dadurch, dass der Graph einen Zyklus enthält, kommt die Abbildungsberechnung zu keinem Ergebnis.

□

Sind Werte als beliebige Graphen modelliert, können Zyklen entstehen, so dass wieder ein Mehrfachzugriff auf Teilwerte innerhalb eines solchen Zyklus über eine Menge unterschiedlicher Pfade möglich ist. In Beispiel 2.10 ist der Name des Autors etwa sowohl direkt über den Pfad `//Autor1/autor/autorenname` adressierbar als auch über den Umweg vom Autor über das Buch zurück zum Autor. Zudem wird, wenn eine Abbildung einen Zyklus entlangläuft, eine Endlosschleife erzeugt und für den Eingabewert nie eine Ausgabe generiert, da die Anfrageausführung nie terminiert. Bei einem Graphen

kann im Grunde jeder Knoten mit jedem anderen über eine Kante verbunden sein. Ein Baum entsteht, wenn durch die Kanten eine strikte Schachtelungsstruktur zwischen den Knoten beschrieben wird, so dass ein Knoten das Kind von nur einem Vaterknoten sein darf. Hierdurch sind keine Zyklen innerhalb der Darstellung mehr möglich.

2.4.2 Verwandte Arbeiten

Es liegt nahe, Instanzen als zusammengesetzte Werte zu betrachten und ihre einzelnen enthaltenen Teilwerte abzubilden. Datenmodelle selbst geben zumeist eine natürliche Aufteilung in derartige Komponenten vor.

Eine Möglichkeit, die semantische Grenze zwischen zwei unterschiedlichen Datenmodellen zu überwinden, stellt der Einsatz von Mediatorarchitekturen dar [PGMW95, Wie92, ACM97, CDSS98, CR03]. Eine gängige Vorgehensweise ist hierbei, ein Zwischendatenmodell zumeist in Form eines semistrukturierten Datenmodells einzuführen und die Datenmodelle des Datenbank- und des Anwendungsschemas hierauf abzubilden. Dies sind zumeist relationale oder objektorientierte Modelle. Um Daten im Zwischendatenmodell darzustellen, werden oft Graph- oder Baumstrukturen eingesetzt, da diese Datenstruktur die meiste Flexibilität bietet und Instanzen anderer Datenmodelle einfach in eine Graphenstruktur überführt werden können.

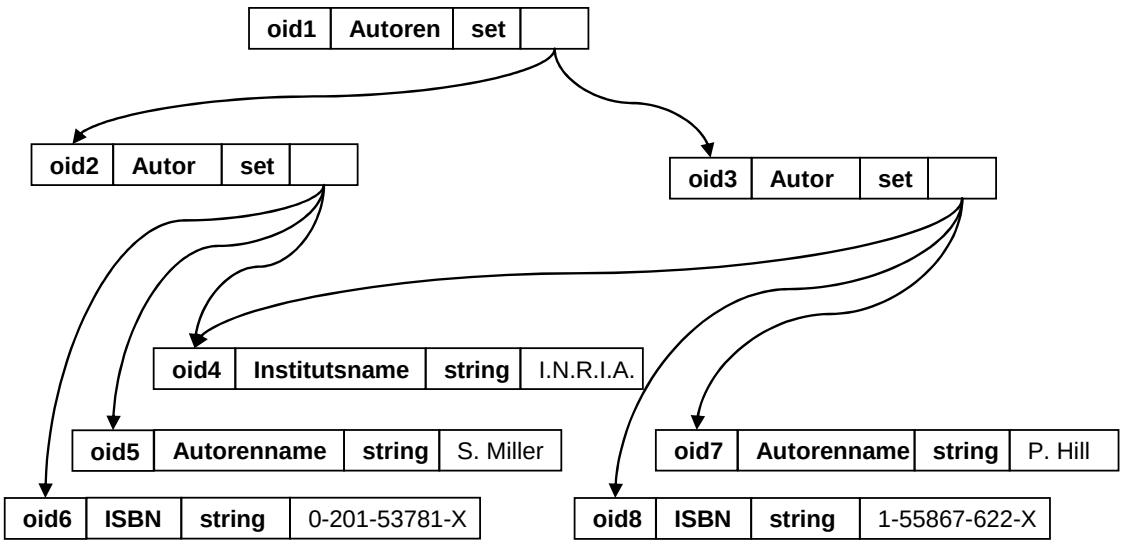


Abbildung 2.20: *Daten in OEM*

Im Rahmen einer Mediatorarchitektur stellen [GMQP+95, PGMW95] das semistrukturierte Datenmodell „Object Exchange Model (OEM)“ vor, mit dessen Hilfe Daten von sowohl objektorientierten Modellen als auch Relationen des relationalen Datenmodells nachgebildet werden können. Ein Wert wird in Form eines Graphen beschrieben, wobei

jeder Graphknoten eine eindeutige Identität, eine Benennung, einen Typ und einen Wert besitzt. Die Typen können entweder atomar sein, dann enthält der Knoten diesen Wert, oder Mengen modellieren, wobei dann der Wert der Menge aus Verweisen auf andere Knoten besteht. Abb. 2.20 zeigt die Modellierung einer Menge von Autoren, wobei bei jedem Knoten dessen eindeutige Kennung, der Bezeichner, der Knotentyp und schließlich der Knotenwert in dieser Reihenfolge angegeben ist. Die Menge der Autoren ist mit `Autoren` gekennzeichnet, `Autor` steht für einen Autoren und dessen Eigenschaften sind entsprechend ihres Bezeichners angegeben.

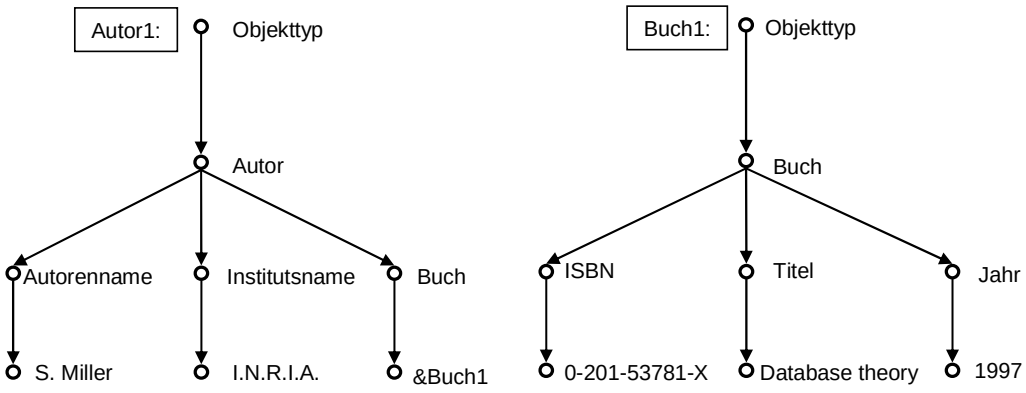


Abbildung 2.21: *Daten in YAT*

Die Daten innerhalb des Datenmodells „Yet Another Tree Model (YAT)“ [CDSS98, MZ98, BM99] werden durch mehrere Bäume dargestellt, wobei diese jeweils durch einen Identifikator gekennzeichnet werden, damit Referenzen zwischen Bäumen definiert werden können. Jeder Knoten innerhalb eines Baums ist gekennzeichnet, wobei die Blattknoten Werte repräsentieren und die übrigen Knoten eine Strukturierung der Werte vornehmen. In Abb. 2.21 ist ein Autor und ein Buch angegeben, wobei Baumidentifikatoren eingerahmt dargestellt sind. Der Autor verweist durch die Angabe der Kennung `&Buch1` auf den entsprechenden anderen Baum.

Problematisch für eine Abbildungsdefinition erweist sich, dass sowohl bei OEM als auch bei YAT auf einen einzelnen Knoten eines Graphen mehrere Verweise vorliegen können, wie etwa auf Knoten `oid4` in Abb. 2.20. Durch solche Verweise existieren zum einen mehrere Zugriffspfade auf einen Teilwert und zum anderen können Zyklen im Graphen gebildet werden.

Eine Reihe von Datenmodellen [PVM+02, GMPS03, BPSM+04] – unter anderem XML – verwendet Daten in Form von Bäumen, bei denen solche Mehrfachreferenzen auf einen Knoten nicht auftreten, da eine strikte Schachtelungsstruktur gefordert wird. Bei diesen Modellen müssen die Kinder eines Baumknotens zumeist nicht eindeutig gekennzeichnet sein, da hiermit Mengen von Werten ausgedrückt werden. Das XML-Dokument aus

Beispiel 2.9 verwendet innerhalb des `autoren`-Elements mehrere `autor`-Elemente, um die Liste der Autoren zu beschreiben.

Baumbasierte Datenmodelle erlauben im Regelfall gleich benannte Kindknoten, so dass der Zugriff auf einzelne Knoten wertbasiert erfolgt und daher auf Teilwerte mehrere Zugriffspfade vorhanden sein können, wie in Beispiel 2.9 gesehen. Ist eine Ordnung auf den Kindknoten vorgegeben, ist auch ein pfadbasierter Zugriff möglich, da die Kinder über ihre Position eindeutig adressierbar sind.

Für Baumstrukturen existieren nicht nur graphische Darstellungsformen, sondern auch textbasierte Formen. Eine Möglichkeit ist die Darstellung in Form von Termen etwa in der Prädikatenlogik [CDG+99]. Ist eine Menge von Funktionssymbolen F_Σ über einem Alphabet Σ gegeben, so weist eine Funktion $\alpha_\Sigma : F_\Sigma \rightarrow I\!N_0$ jedem Funktionssymbol eine Stelligkeit zu. Nullstellige Symbole bilden die Menge der Konstanten. Die Menge der Terme $Term_\Sigma$ kann durch die Regel

- $f \in F_\Sigma$, $\alpha_\Sigma(f) = n$, $t_1, \ldots, t_n \in Term_\Sigma$ ist auch $f(t_1, \ldots, t_n) \in Term_\Sigma$

aufgebaut werden. Ein Wert mit einem Autor ergibt sich als Term betrachtet als

```
autoren(autor(autorenname(S.Miller),
              institutsname(I.N.R.I.A.),
              isbn(0-201-53781-X)))
```

In der Prädikatenlogik besitzt allerdings jedes Funktionssymbol eine feste Stelligkeit, so dass es nicht an verschiedenen Stellen innerhalb eines Werts unterschiedlich verwendet werden kann. Besteht der Autorenname einmal aus einem einzigen Text, der Vor- und Nachname enthält, und woanders im Schema aus Einzelwerten für Vor- und Nachname, müsste das Symbol `autorenname` im ersten Fall einstellig und im zweiten zweistellig sein. Außerdem spielt bei Termen die Reihenfolge der Funktionsparameter eine Rolle. So ist `autor(autorenname(...), institutsname(...), isbn(...))` ein anderer Term als `autor(isbn(...), autorenname(...), institutsname(...))`. Die als Parameter auftauchenden Funktionssymbole müssen hier nicht eindeutig sein, so dass enthaltene Elemente nicht eindeutig adressiert werden.

Werden graph- bzw. baumbasierte Datenmodelle betrachtet, erlauben diese die Modellierung von Zyklen oder gestatten es, dass gleich benannte Kindknoten verwendet werden können, um etwa Mengen von Werten zu modellieren. Da im ersten Fall eine Abbildung entlang eines Zyklus navigieren kann, wird auf einen Teilwert innerhalb des Zyklus mehrfach zugegriffen und somit werden Konsistenzbedingungen auf Anwendungsseite notwendig. Ebenso wird die Modellierung eines Zielschemas schwierig, wenn gleich benannte Kindknoten vorhanden sind und ein wertbasierter Zugriff erfolgt, da hier ein einzelner Teilwert der Eingabe durch unterschiedliche Selektionsbedingungen mehrfach ausgewählt werden kann. Dies gilt sowohl bei einer graphischen als auch einer textbasierten Notation von Bäumen.

2.4.3 Instanzen als Bäume

Werden Werte in Form von Graphen modelliert, hat diese Darstellungsform den Vorteil, dass sich die Werte gängiger Datenmodelle wie des relationalen Modells oder eines objektorientierten Modells relativ einfach in eine Graphstruktur übertragen lassen. Somit ist diese Datenstruktur recht universell, was die Darstellung von Werten angeht. Allerdings geht Semantik verloren, da ein graphbasiertes Modell außer einer Schachtelungsstruktur zwischen Werten keine weiteren Modellierungsmöglichkeiten bietet. Es wird außer Zusammenhängen zwischen einzelnen Teilwerten nichts weiter dargestellt. Da eine Beschreibung einer Abbildung auf der Ebene von Werten stattfindet und keinerlei Semantik involviert ist, ist eine solche Darstellung von Werten für den hier betrachteten Anwendungszweck aber ausreichend.

Da Abbildungen auch zwischen Daten und Metadaten möglich sind, liegt der Übergang zu einem selbstbeschreibenen Datenmodell für die Verwendung in einer Abbildungsbeschreibung nahe, welches Daten und Metadaten gemeinsam behandelt. In der Graphstruktur tauchen diese beiden Datenarten dann gemeinsam auf – etwa als Knotenbezeichner.

Besitzt ein Knoten mehrere Vorgängerknoten, können Zyklen angelegt werden, so dass das Zielschema einer Abbildung nicht angegeben werden kann, da zum einen ein Mehrfachzugriff auf Teilwerte möglich ist und zum anderen eine Abbildungsberechnung nicht zwingend terminieren muss. Werden anstatt allgemeiner Graphen nur Bäume zur Modellierung von Daten eingesetzt, treten diese Schwierigkeiten nicht auf.

Werden auf einen Knoten nachfolgende Knoten nicht eindeutig benannt, muss ein wertbasierter Zugriff auf die folgenden Teilwerte definiert werden, da kein eindeutiger Zugriffspfad definiert werden kann, der die Elemente einzeln adressiert. Wird die Auswahl von Elementen anhand von Selektionsbedingungen vorgenommen, dann ist ein Mehrfachzugriff auf einzelne Elemente möglich, ohne dass dies anhand der Abbildung festgestellt werden kann. Wird eine eindeutige Benennung von Kindknoten durch das Datenmodell erzwungen, wird dies verhindert.

Insgesamt betrachtet, ergibt sich als Anforderung für eine Abbildungsbeschreibung somit die Forderung,

- dass Instanzen als Bäume mit eindeutig benannten Kindknoten modelliert werden müssen, um die Existenz eines Zielschemas nicht zu verhindern.

Definiert man eine Ordnung auf gleich benannten Kindknoten, kann man diese auch in eine eindeutige Benennung der Knoten überführen. Umgekehrt wird eine Ordnung auf eindeutigen Knoten impliziert, wenn man von einer Ordnungsbeziehung auf der Menge der Knotenbezeichner ausgeht. Beide Eigenschaften sind in diesem Sinne durcheinander austauschbar. Zählt man die Bezeichner der Kinder auf

$$\begin{array}{cccccccc} \text{a}, & \text{b}, & \text{a}, & \text{b}, & \text{b}, & \text{c}, & \ldots & \text{(nicht eindeutig und geordnet)} \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & & \\ \text{a[0]}, & \text{b[0]}, & \text{a[1]}, & \text{b[1]}, & \text{b[2]}, & \text{c[0]}, & \ldots & \text{(eindeutig und nicht geordnet)} \end{array}$$

entsteht zwischen den Bezeichnern in beiden Darstellungsarten eine eindeutige Zuordnung.

2.5 Abbildungen

Neben der Modellierung von Instanzen bzw. Werten hängt es des Weiteren auch von der gewählten Abbildungssprache ab, ob diese einen jetzt eindeutig adressierten Teilwert mehrfach abbilden kann, indem etwa derselbe Zugriffspfad in einer Abbildung mehrfach verwendet wird. Dieser Abschnitt untersucht die Abbildungsmöglichkeiten von Bäumen aufeinander und wie ein Abbildungsmechanismus formuliert sein muss, damit Teilwerte nur einfach abgebildet werden können.

2.5.1 Abbildungen auf Bäumen

Werden Werte als Bäume formuliert, beschreibt eine Abbildung eine Zuordnung von Teilbäumen der Eingabe zu Teilbäumen der Ausgabe. Garantiert das Datenmodell, dass auf einen Teilwert innerhalb der Eingabe nur über einen einzigen Pfad zugegriffen werden kann, verbietet dies einer Abbildung nicht, einen Teilwert mehrfach über denselben Pfad auszuwählen und somit Abhängigkeiten auf dem Anwendungsschema zu errichten, so dass kein Zielschema angegeben werden kann.

Beispiel 2.11 (*Teilwerte mehrfach abbilden*)
Ist das Datenbankschema wieder die DTD aus Beispiel 2.6 und ein zugehöriges Eingabedokument das XML-Dokument mit der Liste der Autoren aus Beispiel 2.6, so kann eine Abbildung als folgende XQuery-Anfrage formuliert werden:

```
<ergebnis> {
   for $i in (1 to fn:count(//autoren/autor))
     for $j in (1 to fn:count(//autoren/autor))
       return <a> { //autoren/autor[$i]/autorenname,
                    //autoren/autor[$j]/isbn } </a>
   }
</ergebnis>
```

Innerhalb der `return`-Klausel wird auf die Daten eines einzelnen Autors mehrfach zugegriffen, indem etwa der Pfad `//autoren/autor[1]/autorenname` bei der Berechnung der Abbildung mehrfach verwendet wird. Die Abbildung erzeugt so für n Autoren ein Ausgabedokument mit n^2 `a`-Elementen. Für eine solche Abbildung existiert zwar wieder ein Anwendungsschema, aber es gibt kein Zielschema, da `ergebnis((a(autorenname, isbn))`$^{n^2}$`)` kein regulärer Ausdruck ist. □

Eine Abbildung auf Bäumen kann definiert werden, indem bei einer Abbildungsdefinition ein Teilbaum angegeben wird und wie dieser abgebildet wird. Wenn dieser Teilbaum

im Eingabewert auftritt, wird der in der Abbildungsdefinition zugehörige Teilwert in der Ausgabe generiert. Der selektierte Teilbaum kann hierbei nur ein Pfad wie in Beispiel 2.11 sein oder auch ein Baummuster, das einen Knoten und dessen Kinder bezeichnet – also einen zusammenhängenden Teilbaum beschreibt. Die Kombination aus Pfadausdruck und Baummuster kann variiert werden, so dass entweder nur ein Pfad, nur ein Muster oder ein über einen Pfad adressiertes Baummuster angegeben werden kann, um einen Teil der Eingabe zu selektieren.

Neben der Adressierung und nachfolgender Abbildung bestimmter Teile der Eingabe ist eine weitere Möglichkeit, eine Abbildung auf Bäumen zu beschreiben, die Knoten des Eingabebaums zu durchlaufen und abhängig vom besuchten Knoten eine Ausgabe zu produzieren.

Beispiel 2.12 (*Teilwerte einmal abbilden*)
Neben XQuery ist XSLT [Kay05] eine weitere Abbildungssprache für XML-Dokumente. Im Gegensatz zu XQuery wird hier nicht ein Teil des Eingabebaums adressiert und dann abgebildet, sondern die Knoten des Baums werden nacheinander durchlaufen und hierbei eine Ausgabe generiert.

Eine Abbildungsdefinition in XSLT ergibt sich als

```
<xsl:transform xmlns:xsl=''http://www.w3.org/1999/XSL/Transform''
               version=''2.0''>

  <xsl:template match=''/''>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match=''autor''>
    <xsl:element name=''personenname''>
      <xsl:value-of select=''./autorenname''/>
    </xsl:element>
  </xsl:template>

</xsl:transform>
```

Das erste durch `xsl:template` definierte Baummuster spezifiziert, dass beginnend beim Wurzelkonten des Eingabedokuments, der über `/` adressiert wird, die angegebenen Muster beim Durchlaufen eines Baums angewendet werden sollen. Trifft der XSLT-Prozessor auf ein `autor`-Element, feuert das zweite Muster und gibt den Autorenname in einem `personname`-Element aus. Als Ausgabe für das XML-Dokument aus Beispiel 2.9 entsteht

```
<personenname>S. Maier</personenname>
<personenname>P. Hill</personenname>
```

```
<personenname>R. Scholl</personenname>
```

Die Reihenfolge der Elemente in der Ausgabe ist durch die Reihenfolge der Elemente im Eingabewert festgelegt, da das XML-Dokument vom XSLT-Prozessor nach der impliziten Ordnung der Elemente vom Beginn des Dokuments bis zu dessen Ende durchlaufen wird, falls keine andere Reihenfolge angegeben wird. □

Wird eine Abbildungsberechnung basierend auf einem Durchlaufen eines Baums durchgeführt, wird bei einem Vorbeikommen an einem Knoten eine Ausgabe erzeugt. Wird jeder Knoten nur einmal besucht, wird jedes Element nur ein einziges Mal abgebildet. Ausschlaggebend für die generierte Ausgabe können dann einmal die Position des besuchten Knotens innerhalb des Eingabewerts sein bzw. die Bezeichnung des passierten Knotens oder die Bezeichnungen der Knoten in der Umgebung des besuchten Knotens.

2.5.2 Verwandte Arbeiten

Auf baumbasierten Datenmodellen wird eine Reihe unterschiedlicher Abbildungsmechanismen vorgeschlagen. Diese werden entsprechend ihrer Vorgehensweise bei einer Abbildungsberechnung in zwei Gruppen eingeteilt. Die Sprachen der ersten Gruppe selektieren einen Teil der Eingabe und generieren hierfür eine Ausgabe. Die der zweiten Gruppe durchlaufen den Eingabewert in einer bestimmten Reihenfolge und erzeugen dabei den Ausgabewert.

Zu den Sprachen der ersten Gruppe gehören etwa YATL [CDSS98] auf dem YAT-Modell, LOREL [QRS+95] auf dem OEM-Datenmodell, XQuery [BCF+05] auf XML-Dokumenten und UnQL [BFS00]. Für eine Abbildungsdefinition wird auf Eingabeseite ein Muster eines Teilbaums angegeben und diesem ein Ausgabebaum zugeordnet, der dann erzeugt wird, wenn das Baummuster innerhalb des Eingabewerts vorkommt. Innerhalb eines solchen Musters sind Variablen erlaubt, so dass Teile der Eingabe mit in die Ausgabe kopiert werden können. Da ein Pfad ebenfalls einen Teilbaum darstellt, der von der Wurzel bis zu einem Blatt führt, kann eine Abbildung auch dadurch spezifiziert werden, dass man die Pfade einer Eingabeinstanz denen einer Ausgabeinstanz zuweist. In [PVM+02, FKMP03, RF03] wird auf einem geschachtelten relationalen Modell eine Abbildung in Form von derartigen Pfadkorrespondenzen definiert.

Bei diesen Sprachen ist es erlaubt, dass in einer Abbildung mehrere Baummuster überlappende Teile des Eingabewerts selektieren und hierbei die Knoten in der Schnittmenge der Muster mehrfach abgebildet werden, wie in Beispiel 2.11 zu sehen ist. Somit sind unerwünschte Mehrfachabbildungen von Teilen der Eingabe bei dieser Gruppe von Abbildungssprachen möglich.

[GMPS03] betrachtet einen Wert als Baum, bei dem ein Knoten jeweils eindeutig benannte Kinder besitzt. Eine Abbildung adressiert einen einzelnen Teil der Eingabe und nimmt an dieser Position lokale Umbauten am Baum vor, so dass hierdurch der Ausgabebaum entsteht. Ein Umbau wird durch einen Operator beschrieben, der Teilwerte

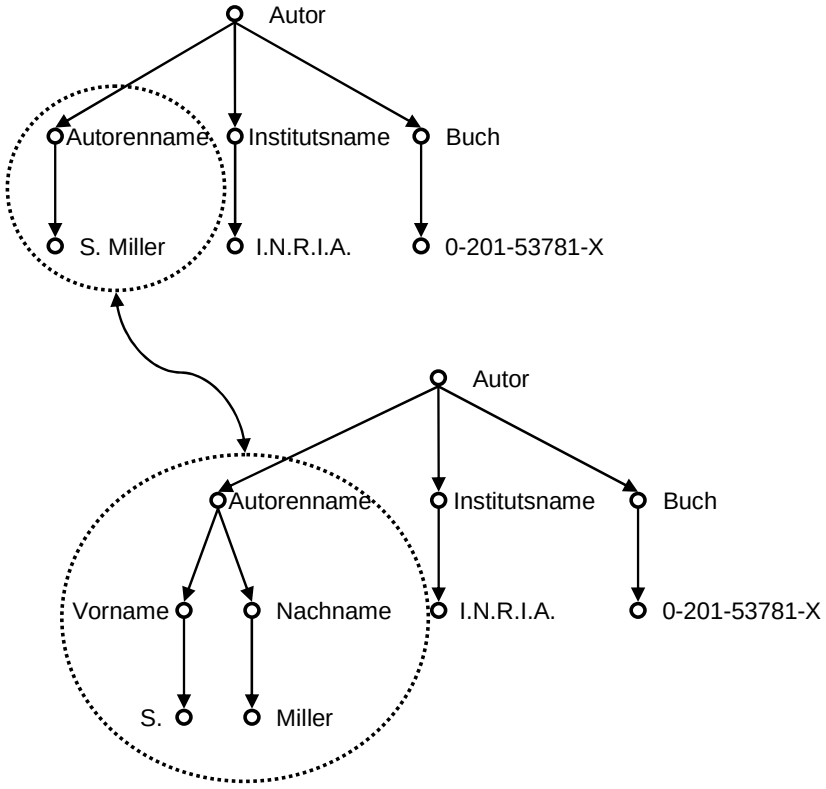


Abbildung 2.22: *Abbildung von Teilbäumen*

eines Baums durch andere ersetzt, wobei für komplexere Abbildungen diese Operatoren kombiniert werden können. Insbesondere werden hier Operatoren betrachtet, die einen inversen Operator besitzen, so dass die Abbildung in beide Richtungen eindeutig bestimmt ist. Abb. 2.22 zeigt eine Abbildung, die über den Pfad `//Autor` eine Teil der Eingabe adressiert und am darunter hängenden Teilbaum einen Umbau vornimmt. Der Name des Autors wird in der Eingabe als ein Text bestehend aus Vor- und Nachname dargestellt und im erzeugten Baum sind die beiden Namensbestandteile getrennt. Diese Beispielabbildung ist in beide Richtungen eindeutig definiert, da sowohl der Name auseinander genommen als auch wieder zusammengesetzt werden kann.

Bei einer derartigen Abbildungsdefinition wird jeweils nur ein Teil eines Baums herausgeschnitten bzw. wieder eingefügt. Somit ist zwar garantiert, dass ein Knoten nur einmal abgebildet wird, doch komplexere Abbildungen müssen durch eine Menge von Operatoren beschrieben werden, d.h. eine Abbildungsdefinition gestaltet sich schwierig.

Eine Abbildungssprache, die beim Durchlaufen eines Baums eine Ausgabe erzeugt, ist XSLT [Kay05]. Ein Abbildungsersteller kann festlegen, bei welchem Knoten des Eingabedokuments ein Durchlaufen beginnen soll und in welche Richtung dann weiter navigiert wird. Trifft auf den gerade besuchten Baumknoten eines der bei der Abbildungsdefi-

nition angegebenen Baummuster zu, wird die Abbildung für diesen Teil durchgeführt. Treffen mehrere Muster zu, ist die Abbildung nicht eindeutig bestimmt, da immer nur ein Muster feuern kann und nicht festgelegt ist, welches dies ist. Beim Durchlaufen ist es möglich, die Richtung zu ändern, in welcher der nächste zu besuchende Knoten liegt, so dass beliebig innerhalb eines Baums navigiert werden kann. Es kann mit den Kindern fortgefahren werden oder beim Vaterknoten weitergegangen werden. Wird keine explizite Reihenfolge vorgegeben, wird ein XML-Dokument gemäß der Ordnung seiner Elemente abgearbeitet.

Bei dieser Abbildungsform wird zwar garantiert, dass beim Vorbeikommen an einem Knoten dieser nur einmal abgebildet wird, aber es kann beliebig innerhalb des Baums navigiert werden und hierbei kann ein Knoten mehrfach besucht werden, wobei bei jedem Passieren des Knotens eine – eventuell jedes Mal auch andere – Ausgabe erzeugt wird. Es können folglich Mehrfachabbildungen stattfinden. Zudem ist die Abbildung nicht spezifiziert, wenn mehrere Muster auf denselben Teil der Eingabe passen, so dass anhand der Abbildungsdefinition nicht bestimmt werden kann, welche Ausgabewerte erzeugt werden.

[ACM98] verwendet eine textbasierte Darstellung von Bäumen und attributierte Grammatiken [WG85] zur Beschreibung von Abbildungen. Ist ein Wert als

```
inst{name=I.N.R.I.A, ort=Rennes}
```

gegeben, werden solche Werte als Wörter einer Sprache aufgefasst und Mengen von Werten mit Hilfe einer kontextfreien Grammatik formuliert. Die Grammatik übernimmt hierbei die Rolle eines Schemas und die von der Grammatik akzeptierten Werte die Rolle von Instanzen. Eine Abbildung zwischen Werten wird dadurch definiert, dass während der Akzeptanz eines Werts durch die einzelnen Grammatikregeln bei jeder durchgeführten Regel ein Ausgabewert berechnet wird. Die Attributierung einer Regel beschreibt hierbei, welche Berechnungsvorschriften jeweils zur Anwendung kommen. Eine attributierte Grammatik enthält etwa die Regeln

```
<Institute> → <Institut> <Institute> { $$ := {$1} ∪ $2 } |
              ϵ {$$ := {} }
<Institut> → ''inst{name='' #String '',ort='' #String ''}''
              { $$ := { Name=$1 } }
```

Terminale sind in Anführungszeichen dargestellt, Nichtterminale in spitzen Klammern und Variablen werden durch ein # gekennzeichnet. Die Attributierungen sind jeweils am Ende einer Grammatikregel in geschweiften Klammern angegeben. $-Symbole markieren hier Variablen, wobei der Ausgabewert mit $$ angegeben wird. Solche Variablen können durch die Angabe einer Positionsnummer an Werte gebunden werden, mit denen während der Akzeptanz Nichtterminale und #-Variablen belegt sind. Die Attributierung der ersten Regel erzeugt als Ausgabewert eine Menge aus dem Ausgabewert, der an das `<Institut>`-Nichtterminal gebunden ist, vereinigt mit der Menge von Ausgabewerten, die an `<Institute>` gebunden sind. Entsprechend nimmt die zweite Regel den Wert der

ersten String-Variablen und baut diesen in den Ausgabewert ein. Dem Beispielwert als Eingabe wird die Ausgabe

{`Name=I.N.R.I.A.`}

zugeordnet.

Dadurch, dass ein Eingabewert abschnittsweise mit Hilfe der Grammatikregeln durchlaufen wird, wird verhindert, dass ein Teilwert der Eingabe mehrfach in die Ausgabe abgebildet wird. Ist ein Terminalsymbol beim Übergang mit einer Regel akzeptiert worden, wird es bei keinem weiteren Übergang mehr verwendet. Allerdings können keine Baummuster bei der Eingabe angegeben werden, so dass der Kontext, innerhalb dessen ein Teilwert verwendet wird, bei der Abbildung nicht berücksichtigt werden kann.

Werden Bäume textbasiert als Terme repräsentiert, können Abbildungen zwischen Termen durch Termersetzungssysteme definiert werden, zu denen Baumautomaten [GS97, CDG+99] gehören. Ein Baumautomat erlaubt es, eine Menge von Termen zu beschreiben. Aufgebaut ist ein solcher Automat aus Zuständen, einer Menge von Funktionssymbolen und Übergangsregeln zwischen Zuständen. Eine Regel eines Automaten besitzt die Form

$$f(q_1(x_1), \ldots, q_n(x_n)) \quad \longrightarrow q(f(x_1, \ldots x_n))$$

mit dem Funktionssymbol f, den Zuständen q, q_1, ..., q_n und Variablen x_1, ..., x_n. Enthält ein Eingabeterm an einer beliebigen Stelle eine Funktion f mit n Parametern und sind diese Parameter jeweils durch den Zustand q_i markiert worden, dann werden die Parameterwerte an die Variablen x_i gebunden und der Übergang in den Zustand q durchgeführt, so dass der Term beginnend mit f mit q markiert wird. Wird einer der Finalzustände q_f des Automaten bei der Ableitung eines Terms t erreicht, gilt also $t \longrightarrow \ldots \longrightarrow q_f(t)$ mit einer Folge von Zustandsübergängen, dann ist der Term t vom Automaten akzeptiert. Ein Automat kann mit den Regeln

$$\begin{array}{lcl}
\texttt{inst}(q_{name}(x_1),q_{ort}(x_2)) & \longrightarrow & q_f(\texttt{inst}(x_1,x_2)) \\
\texttt{name}(q_{text}(x_1)) & \longrightarrow & q_{name}(\texttt{name}(x_1)) \\
\texttt{ort}(q_{text}(x_1)) & \longrightarrow & q_{ort}(\texttt{ort}(x_1)) \\
\texttt{I.N.R.I.A.} & \longrightarrow & q_{text}(\texttt{I.N.R.I.A.}) \\
\texttt{Rennes} & \longrightarrow & q_{text}(\texttt{Rennes})
\end{array}$$

und dem Finalzustand q_f angegeben werden, so dass dieser Baumautomat den Term

`inst(name(I.N.R.I.A),ort(Rennes))`

akzeptiert. Das Konzept von Baumautomaten kann dahingehend erweitert werden, dass während der Akzeptanz eines Eingabeterms ein zweiter Term als Ausgabe generiert wird. Ein solcher Baumtransduktor beschreibt somit eine Beziehung zwischen zwei Mengen von Termen; denen die akzeptiert werden und denen die bei einer Akzeptanz generiert werden.

Bei dieser Art der Abbildungsdefinition wird ebenfalls sichergestellt, dass ein einzelner Teilwert der Eingabe nur einmal abgebildet wird. Ist ein Funktionssymbol im Eingabewert durch eine Regel akzeptiert worden, steht es auf der Eingabeseite nicht weiter zur Verfügung. Allerdings eignen sich die verwendeten Terme nicht, um Werte darzustellen, wie in Abschnitt 2.4.2 gesehen.

Wird eine Abbildung auf Bäumen definiert, kann ein Teil der Eingabe ausgewählt und eine Ausgabe generiert werden. Da hierbei ein Teilbaum der Eingabe mehrfach selektiert werden kann, entstehen auf Seite des Anwendungsschemas Abhängigkeiten, die dort nicht ausgedrückt werden können. Somit ist kein Zielschema modellierbar. Auch im Fall eines Durchlaufens eines Baums ist nicht immer gewährleistet, dass nur eine Einfachabbildung erfolgt, wenn etwa beliebig durch den Baum navigiert werden kann. Sind Mehrfachabbildungen ausgeschlossen, wie im Fall einer Grammatik oder eines Baumautomaten, wird der Kontext, innerhalb dessen ein Teilwert in der Eingabe vorkommt, nicht bei der Abbildung berücksichtigt, so dass etwa ein Teilwert nicht je nach Position, an der er vorkommt, unterschiedlich abgebildet werden kann.

2.5.3 Abbildungen nach Partitionierung

Abbildungen auf Bäumen werden dadurch beschrieben, dass auf Eingabeseite einer Abbildungsdefinition ein oder mehrere Baummuster angegeben werden. Ein Muster kann hierbei ein einzelner Teilbaum, ein Pfad oder ein durch einen Pfad adressierter Teilbaum sein. Abb. 2.23 zeigt eine Abbildung f, die zwei Muster definiert. Das eine Muster beschreibt einen Teilbaum mit einem mit `b` markierten Knoten und den Knoten `c` und `e` als Kindern und einen Knoten `0` als Ausgabe. Das andere Muster entsprechend einen Baum aus den Knoten `e`, `k` und `l` und einem Knoten `1` als Ausgabe.

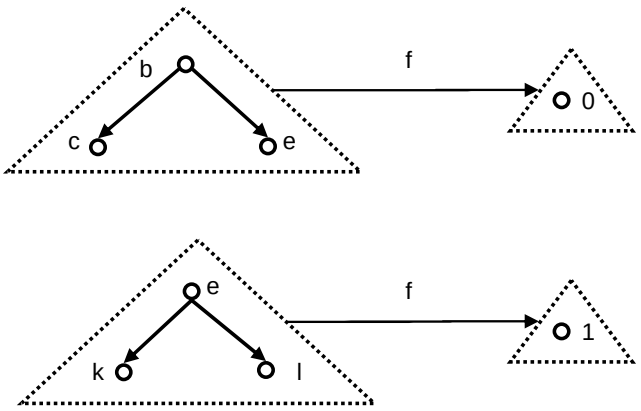


Abbildung 2.23: *Abbildung von Teilbäumen*

Eine Möglichkeit einer Abbildungsausführung ist es, den Eingabebaum nach einem in

der Abbildungsdefinition gegebenen Muster zu durchsuchen und bei jedem Auftreten eines Musters die zugehörige Ausgabe zu erzeugen. Hier kann aber der Fall eintreten, dass Knoten mehrfach abgebildet werden, wie beim Beispielbaum in Abb. 2.24 der Knoten `e`, der in beiden Mustern der Abbildung f auftritt. Ändert sich der Wert `1` in der Bildinstanz, wäre `e` im Eingabebaum zu ändern und somit auch `0` in der Ausgabe. Zwischen den beiden Elementen der Ausgabe besteht also eine Abhängigkeit.

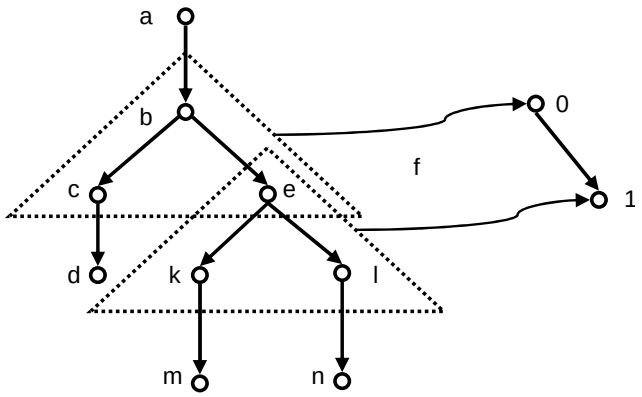


Abbildung 2.24: *Mehrfachauswahl von Teilbäumen*

Eine alternative Abbildungsausführung besteht darin, die Knoten eines Baums in einer gegebenen Reihenfolge zu durchlaufen und beim Passieren von Knoten eine Ausgabe zu generieren, wenn ein Muster passt. Da bei einem Durchlaufen jeder Knoten nur einmal besucht wird, wenn die Richtung fest vorgegeben ist, wird jeder Knoten auch nur einmal abgebildet. Doch auch hier können sich auf Eingabeseite Baummuster überlappen, so dass die Abbildung nicht mehr eindeutig bestimmt ist. Wird beim Baum in Abb. 2.25 beim Knoten `a` gestartet und die Knoten in Postfix-Ordnung durchlaufen, dann wird die Knotenfolge `a`, `b`, `e`, `l`, `n`, `k`, `m`, `c`, `d` passiert. Das zweite Muster der Abbildung f feuert zuerst, da die zu diesem Muster passenden Knoten zuerst abgearbeitet werden. Da der Knoten `e` dann schon verwendet wurde, erzeugt das erste Muster von f keine Ausgabe mehr (Abb. 2.25 links). Wird umgekehrt der Eingabebaum in Präfix-Ordnung durchlaufen, generiert das erste Muster eine Ausgabe und das zweite keine (Abb. 2.25 rechts). Folglich ist anhand der Abbildungsdefinition selbst das Zielschema nicht ableitbar, da die Ausgabe alleine davon abhängt, wie die Berechnung der Abbildung erfolgt.

Solche Probleme werden vermieden, wenn bei Abbildungsdefinitionen solche Überlappungen von Baummustern auf der Eingabeseite nicht zugelassen sind. Es muss also gewährleistet sein, dass die Baummuster auf der Eingabeseite eine Partitionierung der Knoten eines Eingabewerts definieren. Wird dann eine Abbildung bei einem Durchlaufen eines Baums berechnet, ist sichergestellt, dass auf einen Knoten nicht mehrfach zugegriffen wird, da sich keine Muster überlappen. Abb. 2.26 zeigt beispielhaft eine Partitionie-

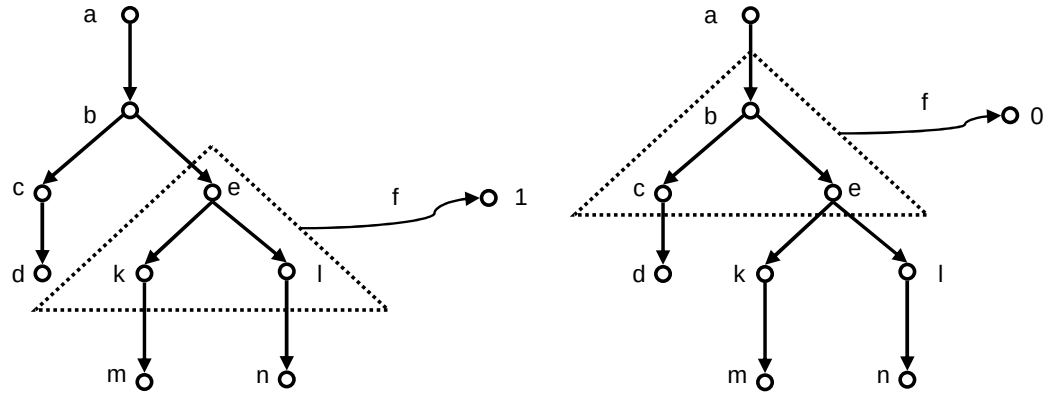


Abbildung 2.25: *Nicht eindeutig bestimmte Abbildung*

rung eines Eingabebaums durch eine Abbildung g. Dem Muster mit den Knoten `b`, `c` und `e` wird die Ausgabe `0`, `1` zugeordnet und entsprechend dem Teilwert mit den Knoten `l`, `n` die Ausgabe `2`. Für die anderen Teile der Eingabe wird keine Ausgabe definiert. Während eines Durchlaufs kann über die Position, an der man sich aktuell befindet, Buch geführt werden, so dass die Abbildung nicht nur abhängig von den Knotenbezeichnern, sondern auch abhängig von der Position im Eingabewert definiert werden kann.

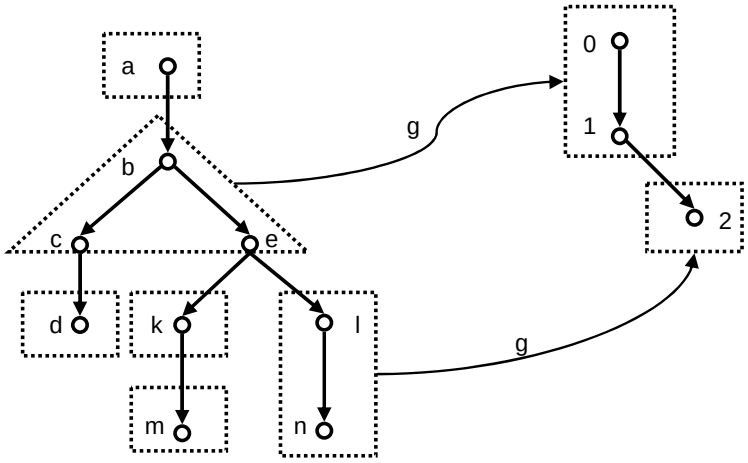


Abbildung 2.26: *Abbildung nach Partitionierung*

Somit ergeben sich für eine Abbildungsbeschreibung die Anforderungen,

- dass Abbildungen die Knoten eines Eingabewerts partitionieren müssen und eine

Abbildung durch das Durchlaufen eines Eingabebaums ausgeführt werden muss, um die Existenz eines Zielschemas nicht zu verhindern.

Wird eine Partitionierung vorgenommen, ist eine Überprüfung der Injektivität der Abbildungsdefinition ebenfalls möglich. Werden zwei unterschiedliche Baummuster der Partitionierung, die an denselben Stellen von Eingabebäumen verwendet werden können, auf dasselbe Muster in der Ausgabe abgebildet, ist die Abbildung nicht mehr injektiv. Dann werden zwei verschiedene Eingabewerte einem einzigen Ausgabewert zugeordnet.

2.6 Fazit

Eine Abbildungsbeschreibung besteht aus vier Gruppen von Komponenten, zu denen die beiden an der Abbildung beteiligten Schemata, deren Instanzen, die dann abgebildet werden, sowie die auf den Schemata definierten Operationen und schließlich die Abbildungsdefinition an sich gehören.

Ausgangspunkt für eine Untersuchung sind die auf dem Anwendungsschema definierten Änderungsoperationen. Bei diesen soll überprüft werden, ob sie semantisch korrekt sind, d.h. ob die Ausführung einer Operation auch den durch ihre Definition angegebenen Effekt besitzt. Dies ist dann der Fall, wenn sowohl die Ausgangsinstanz als auch die Ergebnisinstanz der Operation zum Bildbereich der Abbildung gehören. Dann beschreibt die Operation einen Übergang zwischen zwei konsistenten Systemzuständen. Eine solche Operation ist seiteneffektfrei und es steht fest, dass sie gemäß ihrer Definition durchgeführt werden kann. Um die Seiteneffektfreiheit von Operationen testen zu können, wird das Zielschema der Abbildung verwendet, welches genau die Werte des Bildbereichs der Abbildung als Instanzen umfasst. Um bestimmen zu können, wie eine Operation durchgeführt wird, muss die Abbildung injektiv sein, so dass jeder Anwendungsinstanz höchstens eine Datenbankinstanz zugeordnet wird. Die zur Ergebnisinstanz der Änderungsoperation gehörende Datenbankinstanz kann dann über die Abbildung selbst bestimmt werden. Für eine Abbildungsbeschreibung ergeben sich somit die Anforderungen, dass bei jeder Abbildung

- ein Zielschema vorhanden sein muss und dass
- die Injektivität einer Abbildung getestet werden können muss.

Die Existenz eines Zielschemas hängt vom Datenmodell ab, das beschreibt, welche Wertemengen durch eine Abbildung erzeugt werden können und welche Wertemengen durch ein Schema modellierbar sind. Wenn Teilwerte einer Datenbankinstanz mehrfach abgebildet werden, können Abhängigkeiten zwischen den jeweils erzeugten Bilder der Teilwerte entstehen, die nicht mit einem Schema modelliert werden können. Solche Mehrfachabbildungen können erkannt werden, wenn das Datenmodell nur eindeutige Zugriffspfade auf Teilwerte zulässt und eine Abbildung bei der Berechnung der Anwendungsinstanz nicht mehrfach auf einen Teilwert zugreift.

Wie an den vorgestellten Beispielen zu sehen ist, erfüllen häufig eingesetzte Datenmodelle wie das relationale, objektorientierte oder semistrukturierte Modelle und Abbildungsmechanismus wie die relationale Algebra, OQL oder XQuery die aufgestellten Anforderungen nicht. Daher ist es notwendig, ein passendes Modell und eine Abbildungssprache einzuführen.

3 Annotierte Bäume und ihre Automaten

Ziel dieses Kapitels ist, eine Abbildungsbeschreibung derart zu formulieren, dass mit deren Hilfe bei jeder formulierten Abbildung die semantisch korrekten Operationen bestimmt werden können. Hierzu wird ein neues Datenmodell und eine Abbildungssprache entwickelt, mit welchen die vier an einer Abbildungsbeschreibung beteiligten Komponenten – Werte, Schemata, Abbildung und Operationen – so dargestellt werden, dass für jede mit dieser Darstellung formulierten Abbildung ein Zielschema bestimmt und die Injektivität einer Abbildung getestet werden kann.

Abschnitt 3.1 führt die Formulierung von Werten als Bäume und die Beschreibung von Schemata als Baumautomaten ein. Ein Baumautomat mit Ausgabe wird verwendet, um Abbildungen auf Bäumen zu definieren. Den Nachweis, dass diese Darstellungsform den Anforderungen aus Kapitel 2 genügt, erbringt Abschnitt 3.2, worin gezeigt wird, dass bei jeder durch einen Automaten formulierten Abbildung ein Zielschema vorhanden ist und die Abbildungseigenschaften überprüfbar sind. Dieses Kapitel schließt in Abschnitt 3.3 mit einer Beschreibung einer Beispielabbildung zwischen Bäumen, bei der die semantisch korrekten Operationen bestimmt werden, und einem Fazit in Abschnitt 3.4.

3.1 Abbildungsbeschreibung durch Baumautomaten

Gesucht wird eine Abbildungsbeschreibung, bei der bei jeder formulierbaren Abbildung ein Zielschema vorhanden ist und bei der die Injektivität einer Abbildung überprüft werden kann. Hierzu wird eine geeignete Darstellung von Werten, Schemata, Abbildungsdefinitionen und Operationen benötigt. Auf Grundlage der Untersuchungen in Kapitel 2 werden die vier Komponenten einer Abbildungsbeschreibung geeignet formuliert.

3.1.1 Bäume als Werte

Um Abbildungen beschreiben zu können, muss zunächst festgelegt werden, was abgebildet wird, d.h. es wird eine Darstellung für Werte benötigt. In Abschnitt 2.4 wird ermittelt, dass diese eine Baumstruktur besitzen sollten, wobei die Kinder von Knoten eindeutig benannt sind, damit auf der Bildseite einer Abbildung keine Abhängigkeiten zwischen Teilwerten entstehen, welche die Existenz eines Zielschemas verhindern können. Dies wird in eine formale Beschreibung von Werten umgesetzt.

Eine Menge von nicht weiter unterteilbaren Basiswerten bildet die Grundlage des Datenmodells. Um komplexer strukturierte Daten zu beschreiben, ist es des Weiteren erlaubt, Werte in einander zu schachteln und somit zusammengesetzte Werte aufzubauen. Hierbei enthält ein zusammengesetzter Wert eine Reihe von Basiswerten oder wiederum zusammengesetzte Werte, die jeweils eindeutig durch ein Annotationssymbol innerhalb des Kontextes des sie umgebenden Werts gekennzeichnet werden.

Im Folgenden wird eine abzählbar unendliche Menge von *Annotationssymbolen* mit Σ und eine abzählbar unendliche Menge von *Variablen* mit $\mathcal{X}$ bezeichnet, wobei beide Mengen disjunkt sind, also $\Sigma \cap \mathcal{X} = \emptyset$ gilt. Neben jeder Variablen stellt jede endliche Menge $\{ a_1, \ldots, a_n \}$ aus der Potenzmenge der Annotationssymbole $I\!P^{fin}(\Sigma)$ einen *Basiswert* dar. Da die leere Menge auch ein Element der Potenzmenge der Annotationssymbole ist, ist somit der leere Wert $\{ \}$. Sind beliebige Werte $v_1, \ldots, v_n$ vorhanden, kann ein zusammengesetzter Wert $\{ a_1 : v_1, \ldots, a_n : v_n \}$ aufgebaut werden, indem eine geschachtelte Struktur mit paarweise verschiedenen Annotationen $a_1, \ldots, a_n$ zur Kennzeichnung der enthaltenen Werte $v_1, \ldots, v_n$ aufgebaut wird. Dadurch, dass die Menge $a_1, \ldots, a_n$ auf einer Ebene eines Werts immer endlich sein muss, also die Breite einer Ebene nach oben beschränkt ist, kann es keinen Wert mit einer unbeschränkten Anzahl von direkt enthaltenen Teilwerten geben. Die Schachtelungstiefe hingegen besitzt keine obere Schranke.

Formal betrachtet, wird die Menge $\mathcal{V}(\Sigma, \mathcal{X})$ aller *Werte* über einer Menge von Annotationen Σ und Variablen $\mathcal{X}$ rekursiv durch die folgenden Regeln aufgebaut:

- Für alle $x \in \mathcal{X}$ gilt $x \in \mathcal{V}(\Sigma, \mathcal{X})$.
- Für alle $\{a_1, \ldots, a_n\} \in I\!P^{fin}(\Sigma)$ gilt $\{ a_1, \ldots, a_n \} \in \mathcal{V}(\Sigma, \mathcal{X})$.
- Wenn $\{a_1, \ldots, a_n\} \in I\!P^{fin}(\Sigma)$ und $v_1, \ldots, v_n \in \mathcal{V}(\Sigma, \mathcal{X})$, dann gilt $\{ a_1 : v_1, \ldots, a_n : v_n \} \in \mathcal{V}(\Sigma, \mathcal{X})$.

Die Menge der innerhalb eines Werts v vorkommenden Variablen wird mit $var(v)$ bezeichnet. Ist diese Menge leer, enthält ein Wert also keine Variable, dann ist er *variablenfrei*, wobei die Menge aller variablenfreien Werte im Folgenden mit $\mathcal{V}(\Sigma)$ bezeichnet wird. Ein Wert ist *endlich*, wenn seine Schachtelungstiefe eine obere Schranke besitzt, also die Anzahl der in ihm vorkommenden Ebenen endlich ist. Die Menge der endlichen Werte wird mit $\mathcal{V}^{fin}(\Sigma, \mathcal{X})$ bezeichnet. Ein Wert ist *linear*, wenn er jede in ihm vorkommende Variable genau einmal enthält. Ist ein Wert $v = \{ \ldots, a_i : v_i, \ldots \}$ zusammengesetzt, so adressiert $v.a_i = v_i$ den Wert v_i und die Annotation a_i entspricht der Adresse von v_i innerhalb von v.

Beispiel 3.1 (*Werte*)
Ist die Menge der Annotationssymbole als $\Sigma = \{ a, b, c, d \}$ und eine Variablenmenge $\mathcal{X} = \{x_1, x_2\}$ gegeben, so können etwa die zusammengesetzten Werte

$$v_1 = \{ a : \{ a : \{ c \} , b : \{ a \} \} , b : \{ c \} \} ,$$

$v_2 = \{\ c : x_1\ \}$,
$v_3 = \{\ a : x_1,\ b : x_1,\ c : \{\ d : x_2\ \}\ \}$

gebildet werden, wobei der erste Wert v_1 variablenfrei ist. v_2 ist linear, da eine Variable höchstens einmal auftritt. v_3 ist weder variablenfrei noch linear, da v_3 die Variable x_1 an zwei Stellen enthält. $v_3.c$ adressiert den Teilwert $\{\ d : x_2\ \}$ innerhalb von v_3.

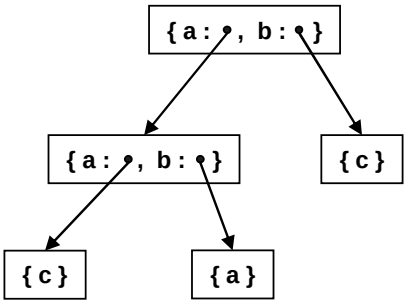


Abbildung 3.1: *Der Wert v_1 in seiner graphischen Darstellung*

Ein Wert besitzt neben der textbasierten Darstellung auch eine graphische in Form eines Baums, indem für jede Ebene des Werts ein Knoten im Baum und für jede Enthaltensseinbeziehung zwischen zwei Ebenen eine Baumkante angelegt wird. Abb. 3.1 zeigt den Wert v_1 in seiner graphischen Darstellung. □

Enthält ein Wert v eine Menge von Variablen, so können diese mit Werten belegt werden. Eine Variablenbelegung wird durch eine Substitution beschrieben, wobei eine *Substitution* eine Abbildung von der Menge der Variablen $\mathcal{X}$ in die Menge der Werte $\mathcal{V}(\Sigma, \mathcal{X})$ ist. Eine Substitution $\sigma = [x_1\ /\ v_1, \ldots, x_n\ /\ v_n]$ ist definiert als eine Identitätsabbildung auf $\mathcal{X} - \{x_1, \ldots, x_n\}$ und ordnet für $1 \leq i \leq n$ jedem $x_i \in \mathcal{X}$ einen Wert $v_i \in \mathcal{V}(\Sigma, \mathcal{X})$ zu. Substitutionen werden auf Werte erweitert als $\sigma(\ \{\ a_1 : v_1,\ \ldots,\ a_n : v_n\ \}\ ) = \{\ a_1 : \sigma(v_1),\ \ldots,\ a_n : \sigma(v_n)\ \}$. Die Notation $v\sigma$ steht für die Anwendung der Substitution σ auf den Wert v. Ist etwa $v = \{\ a : \{\ b : x_1\ \}\ ,\ c : x_2\ \}$ gegeben, so ist $v\ [\ x_1\ \ /\ \{\ d\ \}\ ] = \{\ a : \{\ b : \{\ d\ \}\ \}\ ,\ c : x_2\ \}$. Durch diese Substitution wird die Variable x_1 in v durch den Wert $\{\ d\ \}$ ersetzt.

Neben Einzelwerten werden auch Mengen von Werten betrachtet, wobei hierzu festgelegt werden muss, wann zwei Werte als gleich angesehen werden, um die Eindeutigkeit der Mengenelemente sicherstellen zu können. Auf der Menge der variablenfreien Werte wird eine Äquivalenzrelation $= \subseteq \mathcal{V}(\Sigma) \times \mathcal{V}(\Sigma)$ definiert, welche die Gleichheit von Werten beschreibt. Zwei Werte sind genau dann gleich, wenn beide mit derselben Menge von Annotationssymbolen mit jeder Annotation aus der Menge jeweils die gleichen enthaltenen Werte adressieren. Somit sind zwei Werte $v = \{\ a_1 : v_1,\ \ldots,\ a_n : v_n\ \}$ und

$w = \{\ b_1 : w_1, \ldots, b_n : w_n\ \}$ genau dann *gleich*, $v = w$, wenn

$$v = w :\Leftrightarrow \{a_1, \ldots, a_n\} = \{b_1, \ldots, b_n\} \wedge \\ \forall a \in \{a_1, \ldots, a_n\} \exists b \in \{b_1, \ldots, b_n\} : a = b \Rightarrow v.a = w.b$$

Diese Definition der Gleichheit drückt aus, dass die Reihenfolge der Annotationssymbole auf einer Ebene keine Rolle spielt, solange über sie dieselben Teilwerte adressiert werden.

Die in diesem Abschnitt vorgestellte Art, Werte zu modellieren, garantiert, dass durch einen Wert eine strikte Schachtelungsstruktur beschrieben wird. Somit sind keine Verweise zwischen Teilwerten modellierbar und daher treten auch keine Zyklen innerhalb von Werten auf. Zudem sind dadurch, dass die Annotationen einer Ebene eindeutig sein müssen, über die Annotationen einer Ebene die enthaltenen Teilwerte auf den nachfolgenden Ebenen eindeutig adressiert. Folglich stellt die gewählte Wertdarstellung für alle Werte die Erfüllung der Anforderung aus Abschnitt 2.4 sicher, dass ein Wert eine Baumstruktur mit eindeutig benannten Kindknoten besitzen muss.

3.1.2 Baumautomaten als Schemata

Bei einer Abbildung werden Werte nicht isoliert betrachtet, sondern zumeist eine Menge von Werten abgebildet, die zusammen die Elemente des Definitionsbereichs einer Abbildung bilden. Um Abbildungen formulieren zu können, wird also zunächst ein Beschreibungskonstrukt für Mengen von Bäumen benötigt.

Um eine Wertemenge zu modellieren, bedient man sich im Regelfall eines Schemas, welches den strukturellen Aufbau gültiger Werte festlegt. Da ein Wert die Form eines Baums aufweist, kann eine Menge von Werten durch ein Schema in Form eines Baumautomaten beschrieben werden, wobei die Menge der vom Automaten akzeptierten Werte als Menge der Instanzen interpretiert wird. Die Zustandsübergangsregeln des Automaten legen fest, wie die einzelnen Ebenen eines zum Schema gehörigen Werts aufgebaut sein müssen.

Ein Automat setzt sich aus einer Menge von Zuständen und einer Menge von Übergangsregeln zusammen, die beschreiben, wie zwischen diesen Zuständen gewechselt werden kann. Übergibt man dem Automaten einen Eingabewert, so legt der Aufbau dieses Werts fest, wann welcher Zustandsübergang innerhalb des Automaten stattfinden kann. Ein einzelner Übergang beschreibt hierbei, welche Annotationen auf welcher Schachtelungsebene innerhalb eines gültigen Werts erwartet werden. Der Automat bewegt sich bei den Zustandsübergängen bei den Blättern beginnend nach oben durch den Wert, bis der Finalzustand des Automaten erreicht wird oder keine weitere Übergangsregel mehr anwendbar ist.

Definition 3.1 (*Baumautomat*)
Ein *Baumautomat* über einer Menge von Annotationen Σ ist ein Tupel $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ mit

- Q als einer endlichen Menge von *Zuständen*,

- Σ als einer endlichen Menge von *Annotationssymbolen*,
- $q_f \in Q$ als dem *Finalzustand* und
- Δ als einer endlichen Menge von *Übergangsregeln* der Form

$$u \quad \longrightarrow \quad q$$

wobei $u \in \mathcal{V}^{fin}(\Sigma, Q)$ ein endlicher Wert über den Annotationssymbolen Σ und Zuständen Q ist, $u \notin Q$ nicht aus nur einem Zustand besteht und $q \in Q$.

Eine Übergangsregel $u \longrightarrow q$ enthält auf ihrer linken Seite einen endlichen Wert u, der über den Annotationssymbolen Σ gebildet wird und der beliebige Zustände aus Q als Blätter besitzen kann. Auf der rechten Seite einer Regel findet sich ein Zustand q. Sind etwa $\Sigma = \{a, b, c\}$ und $Q = \{q_1, q_2, q_3\}$, so sind

$$\begin{array}{lcl} \{\ a : q_1,\ b : q_1,\ c : \{\ a\ \}\ \} & \longrightarrow & q_2 \\ \{\ a : \{\ b : q_1\ \},\ c : q_3\ \} & \longrightarrow & q_2 \\ \{\ a,\ c\ \} & \longrightarrow & q_1 \end{array}$$

gültige Regeln. Eine Regel der Form $q_1 \longrightarrow q_2$ ist nicht gestattet, da hier auf der linken Seite nur ein Zustand und kein Symbol aus Σ vorkommt.

Wird ein Wert einem Automaten als Eingabe übergeben, führt dieser eine Ableitung in Form einer Reihe von Zustandsübergängen durch und akzeptiert hierdurch den Wert oder akzeptiert ihn nicht. Ein Übergang wird durchgeführt, indem ein Teilbaum des Eingabewerts durch einen Zustand ersetzt wird, wobei die Übergangsregeln des Automaten angeben, welche Ersetzungen erlaubt sind. Durch die Angabe von Übergangsregeln impliziert ein Baumautomat $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ eine Übergangsrelation $v \longrightarrow_{\mathcal{A}} v'$ zwischen Werten mit $v, v' \in \mathcal{V}(\Sigma, Q)$, die als

$$v \longrightarrow_{\mathcal{A}} v' :\Leftrightarrow \begin{cases} \exists w \in \mathcal{V}(\Sigma, \{x\} \cup Q) \\ \exists u \longrightarrow q \in \Delta, \\ v = w[x\ /\ u], \\ v' = w[x\ /\ q] \end{cases}$$

definiert ist. Ein Übergang zwischen zwei Werten v und v' kann stattfinden, wenn es für einen Teilwert u von v eine Regel $u \longrightarrow q$ gibt, die genau diesen Teilwert auf ihrer linken Seite besitzt. Der Übergang wird durchgeführt, indem in v der Teilwert u durch den auf der rechten Seite der Regel angegebenen Zustand q ersetzt wird. Ist ein Wert v gegeben, der einen Teilwert $\{\ a : q_1,\ b : q_2\ \}$ enthält, so zeigt Abb. 3.2 den Übergang von v nach v' mit einer Regel $\{\ a : q_1,\ b : q_2\ \} \longrightarrow q$, bei dem $\{\ a : q_1,\ b : q_2\ \}$ durch q ersetzt wird.

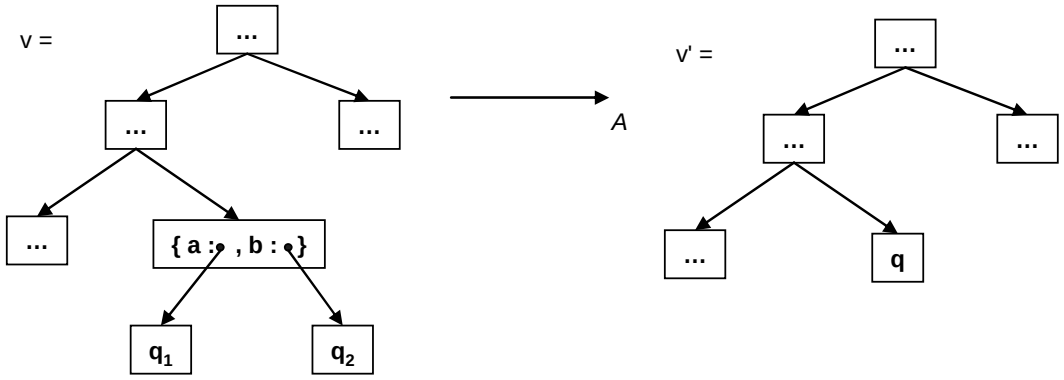


Abbildung 3.2: *Bei einem Übergang zwischen den Werten v und v' mit Hilfe der Regel* $\{\ a : q_1,\ b : q_2\ \} \longrightarrow q$ *wird ein Teilbaum durch einen Zustand ersetzt.*

Ein Startzustand muss nicht angegeben werden, da die Abarbeitung eines Werts von unten nach oben – also bei den Blättern beginnend – erfolgt und somit zuerst die Übergangsregeln durchgeführt werden, deren linke Seiten keine Zustände enthalten. Wenn gleichzeitig Übergänge mit mehreren Regeln möglich sind, ist die Reihenfolge, in der die einzelnen Übergänge durchgeführt werden, nicht zwingend festgelegt. Ebenso können für einen Teilwert der Eingabe mehrere alternative Übergänge vorhanden sein, wenn der Teilwert den linken Seiten mehrerer Regeln entspricht.

Ist ein Baumautomat vorgegeben, ist überprüfbar, ob ein gegebener Wert zu der durch den Automaten definierten Wertemenge gehört oder nicht. Ein Baumautomat $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ *akzeptiert* einen Wert $v \in \mathcal{V}(\Sigma)$, wenn es mindestens einen Ableitungsweg $v \xrightarrow{*}_{\mathcal{A}} q_f$ gibt, der zum Finalzustand q_f des Automaten führt, wobei $\xrightarrow{*}_{\mathcal{A}}$ die reflexive, transitive Hülle der Übergangsrelation $\longrightarrow_{\mathcal{A}}$ ist. Für einen Wert v kann es nicht nur einen, sondern auch mehrere Ableitungswege geben, da innerhalb der Regelmenge eines Automaten mehrere Regeln mit passender linker Seite vorhanden sein können, so dass für einen Teilwert alternative Übergänge möglich sind. Zu einem Eingabewert gehörige Ableitungswege können sich folglich dahingehend unterscheiden, dass jeweils andere Übergangsregeln durchlaufen werden oder verschiedene Zustände erreicht werden. Für einen Wert v werden zwei Ableitungswege als *gleich* betrachtet, wenn sie für v und jeden Teilwert von v in Anzahl und Menge der jeweils zur Ableitung verwendeten Regeln übereinstimmen und nur die Reihenfolge, in der die Regeln durchgeführt werden, vertauscht sind. Bei gleichen Wegen kann die Reihenfolge, in der einzelne Teilbäume abgeleitet werden, verschieden sein.

Für einen endlichen Eingabewert terminiert eine Ableitung immer, da bei einem Übergang mit einer Regel immer Annotationen aus dem Eingabewert entfernt werden. Jede Regel besitzt mindestens ein Annotationssymbol auf ihrer linken Seite. Der Eingabewert wird also gemessen an der Zahl der in ihm enthaltenen Annotations- und Zustandssym-

bol bei jedem Ableitungsschritt kleiner.

Die Menge der *Instanzen* eines Baumautomaten $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ wird mit

$$dom(\mathcal{A}) := \{v \in \mathcal{V}(\Sigma) \mid v \xrightarrow{*}_{\mathcal{A}} q_f\}$$

bezeichnet und besteht aus den Werten v, für die es mindestens je einen Ableitungsweg $v \xrightarrow{*}_{\mathcal{A}} q_f$ in den Finalzustand q_f gibt. $dom(\mathcal{A})$ enthält somit alle von $\mathcal{A}$ akzeptierten Werte. Zwei Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ sind *äquivalent*, wenn die Mengen ihrer Instanzen identisch sind, wenn also $dom(\mathcal{A}_1) = dom(\mathcal{A}_2)$ gilt. Analog gibt es für jeden Zustand $q \in Q$ eine Menge von Werten $dom(q)$, die in ihn abgeleitet werden, und seinen *Wertebereich* bilden als

$$dom(q) := \{v \in \mathcal{V}(\Sigma) \mid v \xrightarrow{*}_{\mathcal{A}} q\}.$$

Beispiel 3.2 (*Baumautomat*)
Sei ein Baumautomat durch $\mathcal{A} = (\{q_f, q_1, q_2, q_3\}, \{ a, b, c \}, q_f, \Delta)$ mit $\Delta = \{$

$$\begin{array}{lcl}
\{ a : q_1, b : q_3 \} & \longrightarrow & q_f \\
\{ a : q_2, b : q_3 \} & \longrightarrow & q_f \\
\{ a : q_3, b : q_3 \} & \longrightarrow & q_1 \\
\{ a : q_3, b : q_3 \} & \longrightarrow & q_2 \\
\{ c \} & \longrightarrow & q_3 \\
\{ a \} & \longrightarrow & q_3
\end{array}$$

$\}$ gegeben. Die Ableitung des Werts $v_1 = \{ a : \{ a : \{ c \} , b : \{ a \} \} , b : \{ c \} \}$ aus Beispiel 3.1 erfolgt durch eine Reihe von Übergängen.

Ein möglicher Ableitungsweg nach q_f wird durch die folgenden Übergänge beschrieben, wobei bei jedem Ableitungsschritt die verwendete Übergangsregel aus Δ angegeben ist:

$$\begin{array}{lcll}
v_1 & \longrightarrow_{\mathcal{A}} & \{ a : \{ a : q_3, b : \{ a \} \} , b : \{ c \} \} & \textit{5.Regel} \\
 & \longrightarrow_{\mathcal{A}} & \{ a : \{ a : q_3, b : \{ a \} \} , b : q_3 \} & \textit{5.Regel} \\
 & \longrightarrow_{\mathcal{A}} & \{ a : \{ a : q_3, b : q_3 \} , b : q_3 \} & \textit{6.Regel} \\
 & \longrightarrow_{\mathcal{A}} & \{ a : q_1, b : q_3 \} & \textit{3.Regel} \\
 & \longrightarrow_{\mathcal{A}} & q_f & \textit{1.Regel}
\end{array}$$

In Abb. 3.3 ist zu sehen, wie bei dieser Ableitung der Wert v_1 von unten nach oben durchlaufen wird. Da für v_1 eine Ableitung vorhanden ist, bei welcher der Finalzustand q_f des Automaten erreicht wird, ist der Wert v_1 eine gültige Instanz von $\mathcal{A}$, also $v_1 \in dom(\mathcal{A})$. Ebenso gehört v_1 zum Wertebereich des Finalzustands: $v_1 \in dom(q_f)$.

Für v_1 ist auch der Ableitungsweg

$$\begin{array}{lcll}
v_1 & \longrightarrow_{\mathcal{A}} & \{ a : \{ a : q_3, b : \{ a \} \} , b : \{ c \} \} & \textit{5.Regel} \\
 & \longrightarrow_{\mathcal{A}} & \{ a : \{ a : q_3, b : q_3 \} , b : \{ c \} \} & \textit{6.Regel} \\
 & \longrightarrow_{\mathcal{A}} & \{ a : q_1, b : \{ c \} \} & \textit{3.Regel} \\
 & \longrightarrow_{\mathcal{A}} & \{ a : q_1, b : q_3 \} & \textit{5.Regel} \\
 & \longrightarrow_{\mathcal{A}} & q_f & \textit{1.Regel}
\end{array}$$

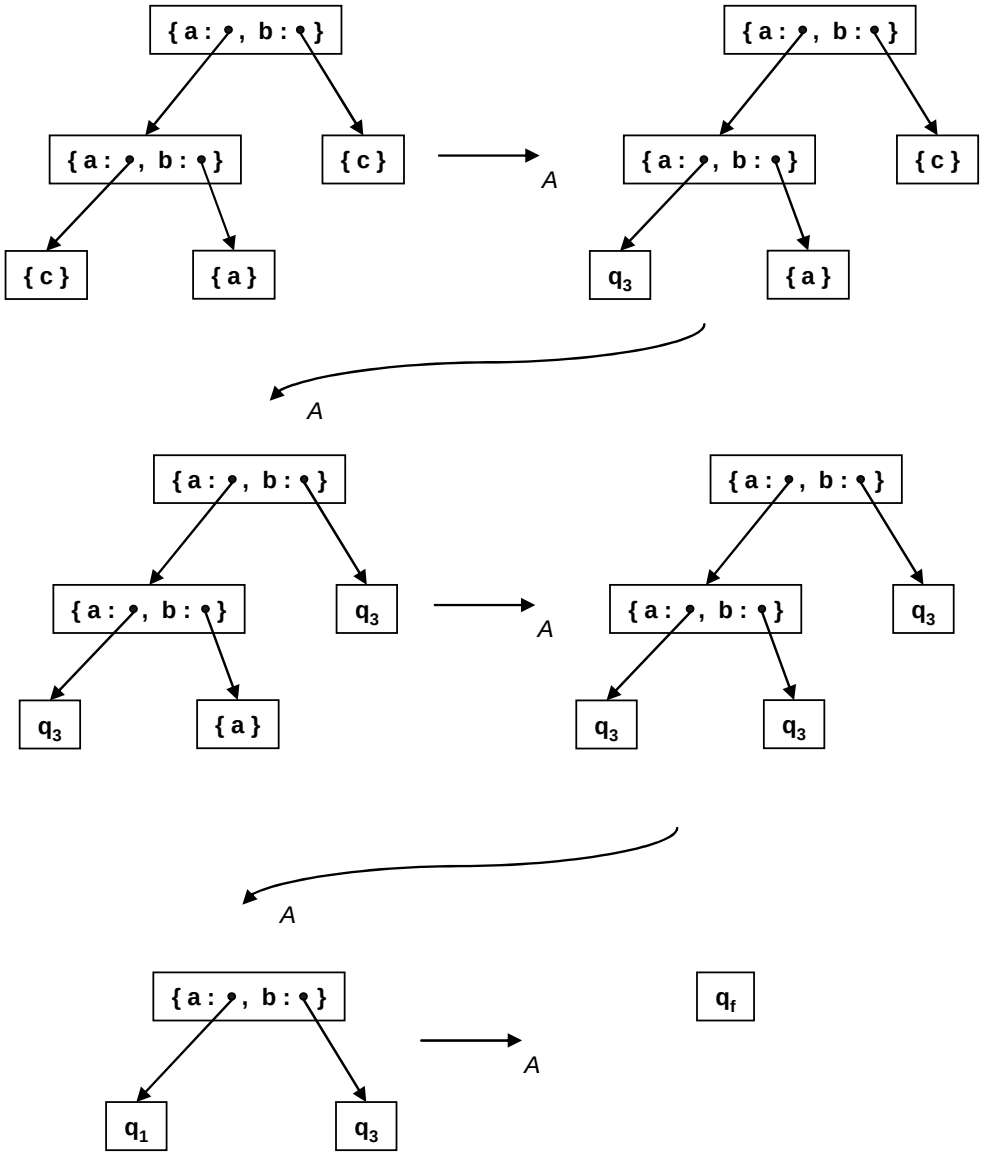


Abbildung 3.3: *Akzeptanz von v_1 durch den Automaten $\mathcal{A}$*

möglich, bei dem im Vergleich zur ersten Ableitung die Reihenfolge der ausgeführten Regeln vertauscht ist, aber für einen Teilbaum dieselben Regeln verwendet werden. Dieser und der erste Ableitungsweg werden als gleich angesehen, da sie sich nur in der Reihenfolge unterscheiden, in der Teilbäume abgeleitet werden. Beide Wege bestehen jeweils aus einmal der ersten, einmal der dritten, zweimal der fünften und einmal der sechsten Regel.

Für v_1 existiert noch ein weiterer Ableitungsweg, der ebenfalls den Zustand q_f erreicht,

hierzu aber eine andere Menge von Übergangsregeln verwendet. Diese Ableitung besteht aus einmal der zweiten, einmal der vierten, zweimal der fünften und einmal der sechsten Regel. □

Mit Hilfe eines Automaten wird es möglich, Mengen von Bäumen zu beschreiben. Ein Baumautomat wird hierbei in der Rolle eines Akzeptors für Werte verwendet, wobei die Menge der akzeptierten Werte seine Instanzen bilden. Übertragen gesprochen liegt mit einem Automaten also ein Schema vor, da sowohl ein Automat als auch ein Schema Wertemengen modellieren.

3.1.3 Abbildungen durch Baumautomaten

Grundlage einer Abbildungsdefinition ist ein Baumautomat, da dieser die Menge der abzubildenden Werte beschreibt, und somit den Definitionsbereich einer Abbildung absteckt. Um eine geeignete Formulierung von Abbildungen zwischen Bäumen zu erhalten, wird den Untersuchungen aus Abschnitt 2.5 gefolgt. Bei einer Abbildung darf somit ein Teilwert nur einmal abgebildet werden. Auf Bäume übertragen bedeutet dies, dass bei einem Baum als Eingabewert für jede Ebene des Baums durch eine Abbildung nur einmal eine Ausgabe festgelegt werden darf.

3.1.3.1 Baumautomaten mit Ausgabe

Bei der Akzeptanz eines Werts werden Übergänge durchgeführt, bei denen ein Teil des Eingabewerts gelöscht und durch einen Zustand ersetzt wird. Nimmt man statt des Entfernens eines Teilwerts eine Ersetzung von Teilwerten vor, wird bei der Akzeptanz eines Eingabewerts ein Ausgabewert generiert und durch den Automaten wird eine Beziehung zwischen Ein- und Ausgabewerten festgelegt. Wo welche Ersetzung stattfinden kann, wird hierbei durch die Übergangsregeln des Automaten beschrieben.

Definition 3.2 (*Baumautomat mit Ausgabe*)
Ein *Baumautomat mit Ausgabe* über Σ und Σ' ist ein Tupel $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ mit

- Q als einer endlichen Menge von *Zuständen*,
- Σ einer endlichen Menge von *Eingabeannotationen*,
- Σ' einer endlichen Menge von *Ausgabeannotationen*,
- $q_f \in Q$ als dem *Finalzustand* und
- Δ als einer endlichen Menge von *Übergangsregeln* der Form

$$u \quad \longrightarrow \quad q\ (\ u'\ )$$

wobei bei jeder Regel gilt, dass $u \in \mathcal{V}^{fin}(\Sigma, Q)$ ein endlicher Wert über den Eingabeannotationen Σ ist, $u \notin Q$ nicht aus nur einem Zustand besteht, jedem Vorkommen eines Zustands in u eine andere Variable x_i aus $x_1, \ldots, x_n \in \mathcal{X}$ in Klammern folgt, $q \in Q$ und $u' \in \mathcal{V}^{fin}(\Sigma', \{x_1, \ldots, x_n\})$ ein endlicher Wert über den Ausgabeannotationen Σ' ist, wobei u' alle die auf der linken Seite der Regel in u auftretenden Variablen x_i genau einmal enthält.

Im Vergleich zu der Definition eines Automaten ohne Ausgabe unterscheidet sich ein Automat mit Ausgabe durch die zusätzliche Angabe einer Menge von Ausgabeannotationen Σ' und durch um Ausgabewerte u' erweiterte Übergangsregeln. Eine Regel enthält auf der linken Seite einen endlichen Wert u über der Menge der Eingabeannotationen Σ, wobei Zustände aus Q als Blätter auftreten dürfen. Enthält ein Blatt in u einen Zustand, so folgt diesem Zustand eine Variable x_i in Klammern, wobei eine einzelne Variable höchstens einmal in u vorkommen darf. Die Ausgabe auf der rechten Seite einer Regel enthält einen Zustand q, einen endlichen Wert u' über den Ausgabeannotationen Σ' und über sämtlichen Variablen $x_1, \ldots, x_n$ der linken Seite der Regel, wobei jede Variable genau einmal in u' vorhanden sein muss. Sind $\Sigma = \Sigma' = \{a, b, c\}$, $Q = \{q_1, q_2, q_3\}$ und $\mathcal{X} = \{x_1, x_2\}$, so sind etwa

$$\begin{array}{lcl}
\{\ a : q_1(x_1),\ b : q_1(x_2)\ \} & \longrightarrow & q_2\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a : \{\ b : q_1(x_1)\ \}\ ,\ c : q_3(x_2)\ \} & \longrightarrow & q_2\ (\ \{\ a : x_2,\ b : \{\ a : x_1\ \}\ \}\ ) \\
\{\ a,\ c\ \} & \longrightarrow & q_1\ (\ \{\ \}\ )
\end{array}$$

gültige Regeln. Jedem Zustand auf der linken Seite einer Regel folgt eine Variable und jede dieser Variablen ist jeweils auf der rechten Seite vorhanden. Eine Substitution auf den Zuständen der Eingabeseiten von Regeln wird als $q_1(x_1)\ [\ x_1\ /\ v_1\ ] = q_1(v_1)$ definiert.

Eine Übergangsregel bei einem Automaten mit Ausgabe legt fest, wie ein Teilwert u eines Werts durch einen anderen Teilwert u' ersetzt wird, während der Automat entlang der Baumstruktur des Eingabewerts läuft. Die Variablen einer Regel sorgen dafür, dass bereits erzeugte Ausgabeteile nicht verloren gehen, sondern bei einer Ersetzung an die richtige Stelle kopiert werden. Die Übergangsrelation $v \longrightarrow_{\mathcal{T}} v'$ ist im Fall eines Automaten mit Ausgabe $\mathcal{T}$ definiert als

$$v \longrightarrow_{\mathcal{T}} v' :\Leftrightarrow \left\{ \begin{array}{l}
\exists w \in \mathcal{V}(\Sigma, \{x\} \cup Q) \\
\exists v'_1, \ldots, v'_n \in \mathcal{V}(\Sigma'), \\
\exists u \longrightarrow q(u') \in \Delta \\
v = w[x\ /\ u[x_1\ /\ v'_1, \ldots, x_n\ /\ v'_n]], \\
v' = w[x\ /\ q(u'[x_1\ /\ v'_1, \ldots, x_n\ /\ v'_n])]
\end{array} \right.$$

Enthält ein Wert v einen Teilwert u, bei dem die Zustände von Werten v'_i gefolgt werden, und gibt es eine Regel $u \longrightarrow q(u')$, welche den Teilwert auf ihrer linken Seite besitzt, dann wird u durch $q(u')$ ersetzt und die Bäume v'_i entsprechend den Variablen aus v nach v' umkopiert. Enthält ein Wert einen Teilwert $\{\ a,\ b\ \}$ und besitzt der Automat

eine Übergangsregel $\{\ a, b\ \} \longrightarrow q\ (\ \{\ 0, 1\ \}\ )$, so wird bei einem Übergang mit dieser Regel der Teilwert $\{\ a, b\ \}$ durch $q\ (\ \{\ 0, 1\ \}\ )$ ersetzt. Enthält ein Wert v einen Teilwert $\{\ a : q_1(v'_1),\ b : q_2(v'_2)\ \}$, bei dem jedem q_i ein Teilbaum v'_i folgt und gibt es eine Übergangsregel der Form $\{\ a : q_1(x_1),\ b : q_2(x_2)\ \} \longrightarrow q\ (\ \{\ 0 : x_1,\ 1 : x_2\ \}\ )$, bei der die Annotationen a, b und die Zustände q_1, q_2 mit dem Teilwert übereinstimmen, so wird bei einem Übergang mit dieser Regel der Teilwert $\{\ a : q_1(v'_1),\ b : q_2(v'_2)\ \}$ durch $q\ (\ \{\ 0 : v'_1,\ 1 : v'_2\ \}\ )$ ersetzt. Hierbei sind die Teilbäume v'_i jeweils an die Variablen x_i der Eingabeseite gebunden und werden an den durch die entsprechenden Variablen markierten Positionen auf der Ausgabeseite eingesetzt (Abb. 3.4). Da der bei einer Regel angegebene Ausgabewert linear sein muss, also eine Variable der Eingabeseite höchstens einmal auf der Ausgabeseite vorkommt, wird vermieden, dass in der Ausgabe mehrere Kopien desselben Teilbaums der Eingabe angelegt werden.

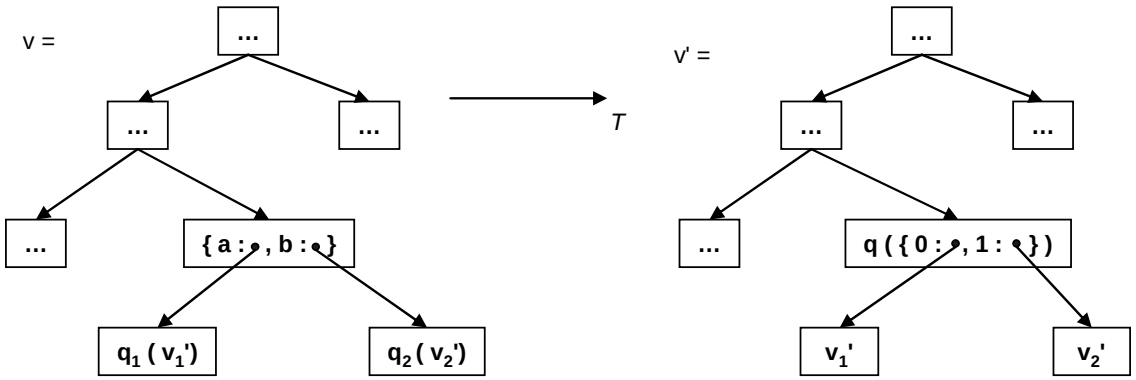


Abbildung 3.4: *Bei einem Übergang zwischen den Werten v und v' mit Hilfe der Regel $\{\ a : q_1(x_1),\ b : q_2(x_2)\ \} \longrightarrow q\ (\ \{\ 0 : x_1,\ 1 : x_2\ \}\ )$ wird ein Teilbaum durch einen anderen ersetzt.*

Die Markierungen der Teilbäume in Form von Zuständen wandern von unten nach oben durch den zu akzeptierenden Wert. Wird letztendlich bei einer Ableitung eines Werts $v \xrightarrow{*}_{\mathcal{T}} q_f(v')$ der Finalzustand q_f des Automaten erreicht, ist der Eingabewert v akzeptiert. Die Menge der Instanzen von $\mathcal{T}$ umfasst alle akzeptierten Werte:

$$dom(\mathcal{T}) := \{v \in \mathcal{V}(\Sigma) \mid v \xrightarrow{*}_{\mathcal{T}} q_f(v')\}.$$

Durch einen Baumautomaten mit Ausgabe $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ wird eine Zuordnung $f_{\mathcal{T}} \subseteq \mathcal{V}(\Sigma) \times \mathcal{V}(\Sigma')$ zwischen Werten über den Eingabeannotationen und Werten über den Ausgabeannotationen impliziert. Akzeptiert ein Automat einen Wert, wird der Finalzustand erreicht und gleichzeitig eine Ausgabe generiert. Die durch den Automaten mit Ausgabe implizierte Relation ergibt sich als

$$f_{\mathcal{T}} = \{(v, v') \in \mathcal{V}(\Sigma) \times \mathcal{V}(\Sigma') \mid v \xrightarrow{*}_{\mathcal{T}} q_f(v')\}$$

Die Relation $f_{\mathcal{T}}$ besteht aus Paaren (v, v') von akzeptierten Werten v und denen bei ihrer Akzeptanz generierten Ausgabewerten v'. Zum Definitionsbereich der Relation gehören alle Werte, die in den Finalzustand abgeleitet werden und somit $dom(f_{\mathcal{T}}) = \{v \mid (v, v') \in f_{\mathcal{T}}\}$ bilden. Umgekehrt ergibt sich der Bildbereich der Relation als $range(f_{\mathcal{T}}) = \{v' \mid (v, v') \in f_{\mathcal{T}}\}$.

Beispiel 3.3 (*Baumautomat mit Ausgabe*)
Ein Automat mit Ausgabe ist $\mathcal{T} = (\{q_f, q_1, q_2, q_3\}, \{ a, c, b \}, \{ 0, 1, 2 \}, q_f, \Delta)$ mit $\Delta = \{$

$$
\begin{array}{lcl}
\{ a : q_1(x_1), b : q_3(x_2) \} & \longrightarrow & q_f(\{ 0 : x_1, 1 : x_2 \}) \\
\{ a : q_2(x_1), b : q_3(x_2) \} & \longrightarrow & q_f(\{ 0 : x_1, 1 : x_2 \}) \\
\{ a : q_3(x_1), b : q_3(x_2) \} & \longrightarrow & q_1(\{ 1 : x_1, 2 : x_2 \}) \\
\{ a : q_3(x_1), b : q_3(x_2) \} & \longrightarrow & q_2(\{ 2 : x_1, 0 : x_2 \}) \\
\{ c \} & \longrightarrow & q_3(\{ 1 \}) \\
\{ a \} & \longrightarrow & q_3(\{ 1 \})
\end{array}
$$

$\}$.

Wird der Wert v_1 aus Beispiel 3.1 akzeptiert, wird ein Ausgabewert erzeugt, indem die entsprechenden Ebenen des Baums ersetzt werden (Abb. 3.5):

$$
\begin{array}{lcll}
v_1 & \longrightarrow_{\mathcal{T}} & \{ a : \{ a : q_3(\{ 1 \}), b : \{ a \} \} , b : \{ c \} \} & \textit{5.Regel} \\
& \longrightarrow_{\mathcal{T}} & \{ a : \{ a : q_3(\{ 1 \}), b : \{ a \} \} , b : q_3(\{ 1 \}) \} & \textit{5.Regel} \\
& \longrightarrow_{\mathcal{T}} & \{ a : \{ a : q_3(\{ 1 \}), b : q_3(\{ 1 \}) \} , b : q_3(\{ 1 \}) \} & \textit{6.Regel} \\
& \longrightarrow_{\mathcal{T}} & \{ a : q_1(\{ 1 : \{ 1 \} , 2 : \{ 1 \} \}), b : q_3(\{ 1 \}) \} & \textit{3.Regel} \\
& \longrightarrow_{\mathcal{T}} & q_f(\{ 0 : \{ 1 : \{ 1 \} , 2 : \{ 1 \} \} , 1 : \{ 1 \} \}) & \textit{1.Regel}
\end{array}
$$

Der Automat ordnet also dem Eingabewert v_1 bei diesem Ableitungsweg als Ausgabe den Wert $\{ 0 : \{ 1 : \{ 1 \} , 2 : \{ 1 \} \} , 1 : \{ 1 \} \}$ zu.

□

Im Folgenden wird eine Übergangsregel $u \longrightarrow q(u')$ eines Automaten mit Ausgabe auch mit Hilfe von Substitutionen als

$$
w \, [\, y_1 \ / \, q_1(x_1), \ldots, y_n \ / \, q_n(x_n) \,] \quad \longrightarrow \quad q \, (\, w' \, [\, y_1 \ / \, x_1, \ldots, y_n \ / \, x_n \,] \,)
$$

notiert, wobei $u = w \, [\, y_1 \ / \, q_1(x_1), \ldots, y_n \ / \, q_n(x_n) \,]$ und $u' = w' \, [\, y_1 \ / \, x_1, \ldots, y_n \ / \, x_n \,]$ gilt, um auszudrücken, dass die linke Seite der Regel einen Wert u enthält, in dem Zustände q_i gefolgt von Variablen x_i auftreten und entsprechend der Wert u' der rechten Seite diese Variablen x_i beinhaltet. Für jede Variable x_i wird bei dieser Schreibweise eine neue Variable y_i eingeführt und dieser auf der linken Seite der Zustand $q_i(x_i)$ mit der Variablen x_i zugeordnet und auf der rechten Seite die Variable x_i selbst. Jede Variable y_i kommt also in w bzw. w' genau einmal vor. Eine Übergangsregel

$$
\{ a : q_1(x_1), b : q_1(x_2) \} \quad \longrightarrow \quad q_2 \, (\, \{ a : x_1, b : x_2 \} \,)
$$

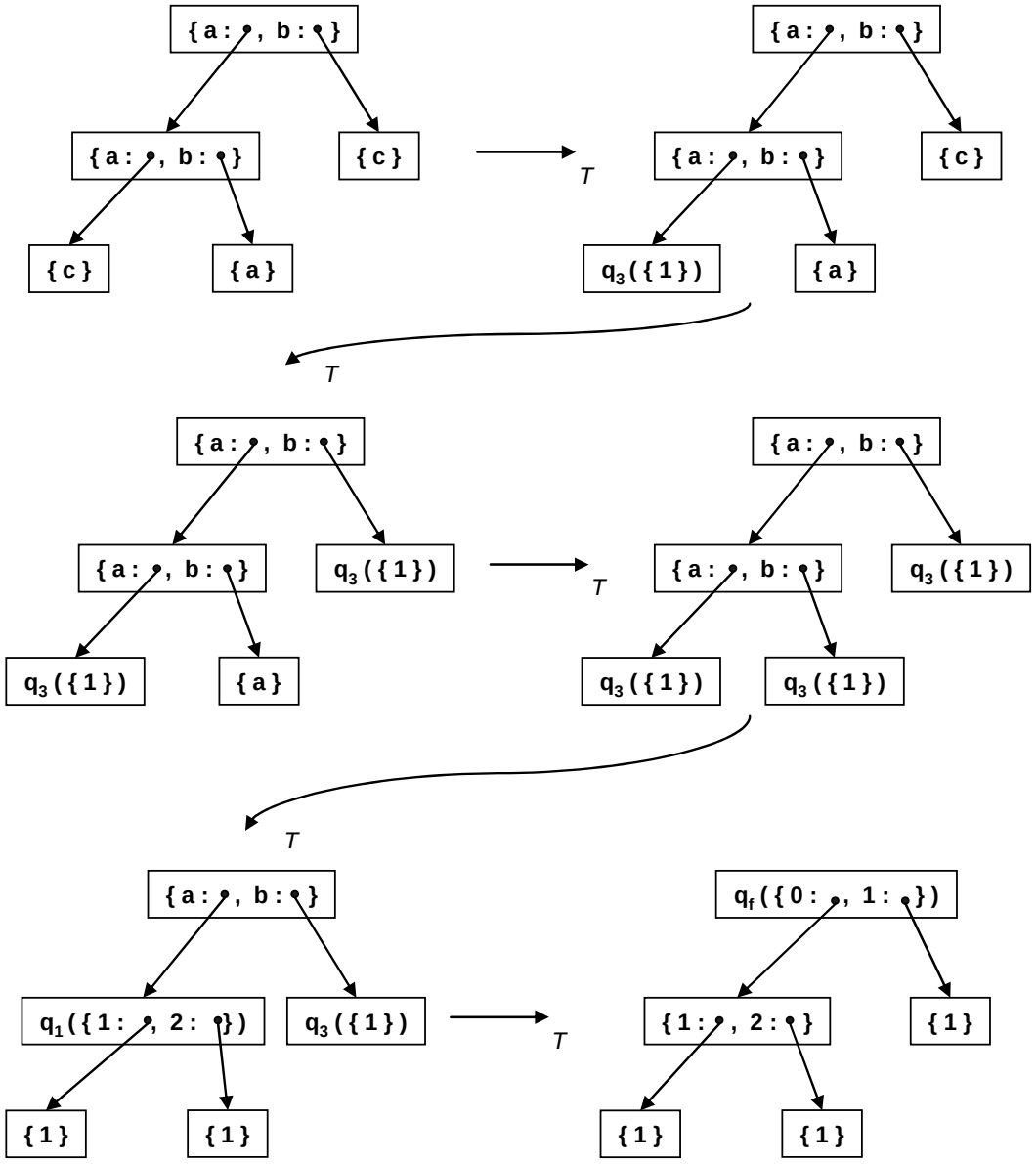


Abbildung 3.5: *Definition einer Beziehung zwischen zwei Werten*

kann also mit Hilfe von Substitutionen auch als

$$\{\ a : y_1,\ b : y_2\ \}\ [\ y_1\ \ /\ q_1(x_1),\ y_2\ \ /\ q_1(x_2)\ ] \longrightarrow q_2\ (\ \{\ a : y_1,\ b : y_2\ \}\ [\ y_1\ \ /\ x_1,\ y_2\ \ /\ x_2\ ]\ )$$

geschrieben werden, wobei y_1 für x_1 und y_2 für x_2 eingeführt wird.

Ist ein Baumautomat mit Ausgabe $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ gegeben, so ist die durch ihn definierte Relation $f_{\mathcal{T}}$ nicht zwingend eine Abbildung im mathematischen Sinn, d.h.

einem Eingabewert v kann nicht nur ein Ausgabewert v' zugeordnet sein, sondern es können mehrere Ausgabewerte $v'_1, v'_2, \ldots$ vorhanden sein. Dies ist etwa dann der Fall, wenn für v mehrere unterschiedliche Ableitungswege existieren und bei diesen unterschiedliche Ausgabewerte erzeugt werden. Ein Automat $\mathcal{T}$ mit den Übergangsregeln

$$\begin{array}{lcl} \{\ a : q_1(x_1),\, b : q_1(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 0 : x_1,\, 1 : x_2\ \}\ ) \\ \{\ c\ \} & \longrightarrow & q_1\ (\ \{\ 3\ \}\ ) \\ \{\ c\ \} & \longrightarrow & q_1\ (\ \{\ 4\ \}\ ) \end{array}$$

impliziert eine Relation $f_{\mathcal{T}}$, welche dem Eingabewert $v = \{\ a : \{\ c\ \}\ ,\ b : \{\ c\ \}\ \}$ unter anderem die Ausgabewerte $v'_1 = \{\ 0 : \{\ 3\ \}\ ,\ 1 : \{\ 3\ \}\ \}$ und $v'_2 = \{\ 0 : \{\ 3\ \}\ ,\ 1 : \{\ 4\ \}\ \}$ zuweist. Die unterschiedlichen Ausgaben entstehen, je nachdem, ob bei der Ableitung von v die zweite oder die dritte Regel für einen Teilwert $\{\ c\ \}$ verwendet wird. Ein Automat mit einer solchen Regelmenge beschreibt somit keine Abbildung, sondern nur eine Relation und ist daher ungeeignet für die weiteren Untersuchungen.

3.1.3.2 Definition einer Abbildung zwischen Werten

Da mit Hilfe von Automaten mit Ausgabe letztendlich Abbildungen formuliert werden sollen, wird ein Kriterium benötigt, anhand dessen erkannt werden kann, ob ein Automat eine Abbildung beschreibt oder ob er nur eine Relation festlegt. Die Automaten mit der letztgenannten Eigenschaft sind ungeeignet für die weiteren Betrachtungen. Um ein solches Kriterium festzulegen, kann der Aufbau der Regelmenge eines Automaten herangezogen werden, da durch die Regeln die Beziehung zwischen Ein- und Ausgabewerten definiert wird. Ob ein Automat das Kriterium dann erfüllt, kann anhand des Aufbaus der Regelmenge untersucht werden.

Durch einen Baumautomaten werden Ebenen des Eingabewerts durch Ebenen des Ausgabewerts ersetzt. Wann welche Ersetzung vorgenommen wird, entscheiden die Übergangsregeln des Automaten. Werden bei einer Akzeptanz eines Eingabewerts bei allen Ableitungswegen dieselben Regeln durchlaufen, werden auch immer die gleichen Ersetzungen vorgenommen und somit immer derselbe Wert als Ausgabe generiert.

Theorem 3.1 (*Abbildungseigenschaft*)
Ist ein Automat mit Ausgabe $\mathcal{T}$ gegeben und sind für jeden Wert $v \in dom(\mathcal{T})$ alle Ableitungswege gleich, dann definiert $f_{\mathcal{T}}$ eine Abbildung. □

Beweis Sei $\mathcal{T}$ ein Automat und $v = w[y_1 \,/\, v_1, \ldots, y_n \,/\, v_n] \in dom(\mathcal{T})$ ein akzeptierter Wert mit einem Ableitungsweg nach q_f:

$$v \xrightarrow{*}_{\mathcal{T}} q_f(v')$$

Betrachtet man den letzten Ableitungsschritt dieses Wegs, der schließlich nach q_f führt,

$$v \xrightarrow{*}_{\mathcal{T}} w[y_1 \,/\, q_1(v'_1), \ldots, y_n \,/\, q_n(v'_n)] \longrightarrow_{\mathcal{T}} q_f(v')$$

ergibt sich daraus, dass in der Regelmenge des Automaten eine Regel der Form w [y_1 / $q_1(x_1)$, ..., y_n / $q_n(x_n)$] $\longrightarrow q_f$ (w' [y_1 / x_1, ..., y_n / x_n]) vorhanden ist, bei welcher der für den Übergang passende Wert und die passenden Zustände q_i auf der linken Seite vorhanden sind und die rechts den Zustand q_f enthält. Bei allen Ableitungswegen von v wird diese Regel als letzte verwendet, da alle Wege als gleich vorausgesetzt sind. Es gibt in der Regelmenge mindestens eine Regel mit dieser linken Seite, da vorausgesetzt wird, dass v akzeptiert wird. Es gibt innerhalb der Regelmenge sogar genau nur eine Regel mit dieser linken Seite, da alle Ableitungswege für v gleich sind. Würde es mehrere Regeln geben, die diese linke Seite und q_f rechts besitzen, gäbe es unterschiedliche Ableitungswege von v, da dann der Übergang alternativ mit einer von diesen Regeln möglich wäre. Dies widerspricht aber der Annahme, dass für v alle Wege gleich sind. Da es nur eine Regel gibt, ist die Ersetzung von w [y_1 / $q_1(v'_1)$, ..., y_n / $q_n(v'_n)$] durch einen Ausgabewert mit dieser Regel eindeutig festgelegt. In der Regelmenge des Automaten gibt es genau eine Regel für den letzten Ableitungsschritt, über den v in den gegebenen Zustand q_f abgeleitet wird.

Sind alle Ableitungswege von v nach q_f gleich, dann bedeutet dies auch, dass für alle Teilwerte v_i die Ableitungen in den jeweiligen Zustand q_i gleich sind:

$$v_1 \xrightarrow{*}_{\mathcal{T}} q_1(v'_1), \ldots, v_n \xrightarrow{*}_{\mathcal{T}} q_n(v'_n)$$

Wäre diese nicht der Fall und gäbe es für ein v_i zwei unterschiedliche Wege zu einem q_i, dann gäbe es auch zwei unterschiedliche Wege für v nach q_f, was der Annahme widerspricht. Entsprechend der Argumentation des vorangegangenen Absatzes für $v \xrightarrow{*}_{\mathcal{T}} q_f(v')$ kann nun für ein Teilwert v_i und ein Zustand q_i bei $v_i \xrightarrow{*}_{\mathcal{T}} q_i(v'_i)$ gefolgert werden, dass in der Regelmenge des Automaten genau eine Regel vorhanden ist, welche im letzten Ableitungsschritt nach q_i verwendet werden kann und somit auch diese Ersetzung eindeutig ist. Folgt man der Schachtelungsstruktur von v bis zu den Blattebenen, gilt auf jeder Ebene, dass es für jeden Teilwert gleiche Ableitungswege in einen entsprechenden Zustand q und genau eine Regel in der Regelmenge gibt, welche im letzten Ableitungsschritt nach q führt.

Folglich wird jede Ebene von v durch den Automaten eindeutig ersetzt, so dass v ein eindeutiger Ausgabewert v' zugewiesen wird. Da v beliebig aus $dom(\mathcal{T})$ gewählt wird, gilt die eindeutige Zuordnung einer Ausgabe für alle akzeptierten Werte, so dass $f_{\mathcal{T}}$ eine Abbildung beschreibt. □

Ist ein Automat mit Ausgabe $\mathcal{T}$ vorhanden und gibt es für jeden akzeptierten Wert nur gleiche Ableitungswege $v \xrightarrow{*}_{\mathcal{T}} q_f(v')$, dann wird ein solcher Automat im Folgenden als *kontextbezogen deterministisch* bezeichnet. Ein solcher Automat beschreibt eine Abbildung.

Die Existenz von gleichen Ableitungswegen ist ein hinreichendes, aber kein notwendiges Kriterium, damit ein Automat eine Abbildung definiert. Es gibt Automaten, die eine Abbildung definieren, bei denen aber mehrere unterschiedliche Ableitungswege für einen Eingabewert existieren. Ein Automat $\mathcal{T}$ mit den Regeln

$$\begin{array}{lcl} \{\ a : \{\ a\ \}\ \} & \longrightarrow & q_f\ (\ \{\ a : \{\ b\ \}\ \}\ ) \\ \{\ a : q_1(x_1)\ \} & \longrightarrow & q_f\ (\ \{\ a : x_1\ \}\ ) \\ \{\ a\ \} & \longrightarrow & q_1\ (\ \{\ b\ \}\ ) \end{array}$$

ist hierfür ein Beispiel. Er beschreibt für den einzig akzeptierten Wert eine Abbildung $f_{\mathcal{T}}$ $(\ \{\ a : \{\ a\ \}\ \}\ ) = \{\ a : \{\ b\ \}\ \}$, besitzt aber für den Wert $\{\ a : \{\ a\ \}\ \}$ zwei unterschiedliche Ableitungswege einmal bestehend aus der ersten Regel und das andere mal aus den Regeln zwei und drei. Beide Wege sind zwar verschieden, doch wird auf jedem Weg derselbe Ausgabewert generiert. Bei solchen Wegen unterscheiden sich die jeweils besuchten Zustände, wobei die Ein- und Ausgabewerte der entsprechenden Regeln auf beiden Wegen gleich sind, so dass solche Automaten redundante Wege enthalten. Automaten mit redundanten Wegen spielen im Folgenden keine weitere Rolle.

Das Vorhandensein von gleichen Wegen für jeden akzeptierten Weg bedeutet nicht, dass der Automat in dem Sinne deterministisch sein muss, dass für einen bestimmten Teilwert der Eingabe immer nur genau eine Regel zutrifft. Passt immer nur eine Regel, gibt es zwar auch nur gleiche Ableitungswege für einen Eingabewert, doch sind deterministische Automaten für die Beschreibung von Abbildungen eine zu starke Einschränkung, da es Automaten gibt, die nicht deterministisch sind, aber trotzdem eine Abbildung beschreiben. Ein Automat $\mathcal{T}$ mit den Regeln

$$\begin{array}{lcl} \{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\ \{\ c\ \} & \longrightarrow & q_1\ (\ \{\ a\ \}\ ) \\ \{\ c\ \} & \longrightarrow & q_2\ (\ \{\ b\ \}\ ) \end{array}$$

ist hierfür ein Beispiel. Dieser enthält mit der zweiten und dritten Regel zwei Regeln, bei denen die linken Seiten identisch sind. Enthält ein Eingabewert einen Teilwert $\{\ c\ \}$, so sind beide Regeln alternativ anwendbar. Doch existieren für den einzig akzeptierten Wert $\{\ a : \{\ c\ \}\ ,\ b : \{\ c\ \}\ \}$ nur gleiche Ableitungswege, die aus den einmaligen Anwendungen der Regeln eins, zwei und drei bestehen. Durch diesen Automaten wird also eine Abbildung beschrieben, obwohl der nicht deterministisch ist.

Der Automat $\mathcal{T}$ zeigt auch, dass bei einem kontextbezogen deterministischen Automaten für einen Teilwert eines Eingabewerts nicht gewährleistet sein muss, dass bei jeder Ableitung für diesen Teilwert immer derselbe Zustand erreicht wird. Für die Eingabe $\{\ c\ \}$ gibt es im Beispiel zwei unterschiedliche Ableitungswege $\{\ c\ \} \longrightarrow_{\mathcal{T}} q_1\ (\ \{\ a\ \}\ )$ und $\{\ c\ \} \longrightarrow_{\mathcal{T}} q_2\ (\ \{\ b\ \}\ )$, bei denen jeweils ein anderer Zustand q_1 bzw. q_2 erreicht wird.

Überprüfung auf eine Abbildung

Ist ein Automat gegeben, so muss eine Überprüfung möglich sein, ob der Automat kontextbezogen deterministisch ist oder nicht. Hierfür wird der Aufbau der Regelmenge des Automaten betrachtet, ob für eine Instanz des Automaten jeweils nur gleiche Ableitungswege vorhanden sind.

Theorem 3.2 (*Partitionierung*)
Ist ein Automat $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ kontextbezogen deterministisch und sind die Regeln

$\Delta_q \subseteq \Delta$ für jedes $q \in Q$ die Mengen aller Regeln, welche jeweils den Zustand q auf ihrer rechten Seite besitzen, dann erfolgt durch die Regeln eines jeden Δ_q eine Partitionierung des Wertebereichs $dom(q)$ des entsprechenden Zustands q. □

Beweis Wie im Beweis zu Theorem 3.1 gezeigt, gilt bei einem kontextbezogen deterministischen Automaten, dass für einen Wert $v \in dom(q)$ die im letzten Übergang nach q verwendete Regel eindeutig festgelegt ist. Ist $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ und sind $u_i \longrightarrow q(u_i') \in \Delta_q$ die Regeln aus Δ, die in q enden, dann sind die Mengen der Werte, welche je eine der Regeln bei der Ableitung nach q verwenden

$$
\begin{aligned}
dom(q, u_1 \longrightarrow q(u_1')) &:= \{\ v_1 \mid v_1 \xrightarrow{*}_{\mathcal{T}} \ldots \overset{u_1 \longrightarrow q(u_1')}{\longrightarrow}_{\mathcal{T}} q\ (\ v_1'\ )\ \} \\
dom(q, u_2 \longrightarrow q(u_2')) &:= \{\ v_2 \mid v_2 \xrightarrow{*}_{\mathcal{T}} \ldots \overset{u_2 \longrightarrow q(u_2')}{\longrightarrow}_{\mathcal{T}} q\ (\ v_2'\ )\ \} \\
&\ldots \\
dom(q, u_n \longrightarrow q(u_n')) &:= \{\ v_n \mid v_n \xrightarrow{*}_{\mathcal{T}} \ldots \overset{u_n \longrightarrow q(u_n')}{\longrightarrow}_{\mathcal{T}} q\ (\ v_n'\ )\ \}
\end{aligned}
$$

disjunkt, d.h. $dom(q, u_i \longrightarrow q(u_i')) \cap dom(q, u_j \longrightarrow q(u_j')) = \emptyset$ für $i \neq j$. Ebenfalls ist $dom(q) = \bigcup_{i=1}^{n} dom(q, u_i \longrightarrow q(u_i'))$, da alle Regeln nach q betrachtet werden. Die Regeln nehmen folglich eine Partitionierung der Wertemenge $dom(q)$ von q vor. □

Ist ein Automat kontextbezogen deterministisch, dann beschreiben die Regeln jedes Zustands q eine Partitionierung von $dom(q)$. Ist umgekehrt ein Automat gegeben und beschreiben die Regeln für jeden Zustand q eine Partitionierung von $dom(q)$, dann ist der Automat kontextbezogen deterministisch. Um die Eigenschaft zu überprüfen, sind also für alle Zustände q eines Automaten die entsprechenden Wertemengen zu konstruieren und auf paarweise Disjunktheit zu überprüfen.

Für die einzelnen Wertemengen $dom(q, u_i \longrightarrow q(u_i'))$ kann jeweils ein Automat angelegt werden, welcher die Werte einer Menge als Instanzen besitzt. Ist ein Automat $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ gegeben, sei $q \in Q$ ein Zustand und $\Delta_q \subseteq \Delta$ die Regeln, die q auf der rechten Seite enthalten. Für jede Regel $u_i \longrightarrow q(u_i')$ aus Δ_q wird ein Automat $\mathcal{T}_{u_i \longrightarrow q(u_i')} = (Q, \Sigma, \Sigma', q, (\Delta - \Delta_q) \cup \{u_i \longrightarrow q(u_i')\})$ konstruiert. Ein solcher Automat besitzt q als Finalzustand und nur eine der Regeln aus Δ_q, die somit in den Finalzustand führt. Dies bedeutet, dass ein solcher Automat genau die Werte aus $dom(q, u_i \longrightarrow q(u_i'))$ akzeptiert. Sind die Mengen der akzeptierten Werte für die für einen Zustand q konstruierten Automaten $\mathcal{T}_{u_i \longrightarrow q(u_i')}$ paarweise disjunkt, dann liegt eine Partitionierung von $dom(q)$ durch die Regeln aus Δ_q vor. Liegt für alle Zustände des Automaten $\mathcal{T}$ eine entsprechende Partitionierung vor, dann ist der Automat kontextbezogen deterministisch.

Zusammenfassend ergibt sich für eine Überprüfung von $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ der folgende Ablauf:

1. Konstruiere für alle $q \in Q$ jeweils die Menge Δ_q, die alle Regeln enthält, die nach q führen.

2. Überprüfe für alle Δ_q, ob sie je eine Partitionierung von $dom(q)$ definieren:

a) Konstruiere $\mathcal{T}_{u_i \longrightarrow q(u_i')} = (Q, \Sigma, \Sigma', (\Delta - \Delta_q) \cup \{u_i \longrightarrow q(u_i')\})$ für alle $u_i \longrightarrow q(u_i') \in \Delta_q$

b) Gilt $dom(\mathcal{T}_{u_i \longrightarrow q(u_i')}) \cap dom(\mathcal{T}_{u_j \longrightarrow q(u_j')}) = \emptyset$ für alle $i \neq j$, dann liegt eine Partitionierung vor.

3. Beschreiben alle Δ_q je eine Partitionierung von $dom(q)$, dann ist der Automat $\mathcal{T}$ kontextbezogen deterministisch.

Während der Überprüfung sind Instanzenmengen von Automaten auf Disjunktheit zu testen. Die Entscheidung, ob es einen Wert gibt, der von einer endlichen Folge von Automaten akzeptiert wird – also der Schnitt der akzeptierten Werte nicht leer ist –, besitzt bei Baumautomaten der Prädikatenlogik einen Aufwand von EXPTIME [CDG+99], so dass für die Überprüfung im obigen Verfahren im allgemeinen Fall von einem hohen Aufwand auszugehen ist.

Es gibt aber Regelmengen, bei denen sich der Aufwand auf paarweise Vergleiche von Regeln reduziert. Ein Automat $\mathcal{T}_{u_i \longrightarrow q(u_i')}$ besitzt als Schlussregel genau eine der Regeln, die in q enden, so dass jeder von diesem Automaten akzeptierte Wert auf der obersten Ebene die entsprechenden in u_i angegebenen Annotationen enthält. Unterscheiden sich bereits diese Annotationen bei den Regeln $u_i \longrightarrow q(u_i')$, sind die Instanzenmengen der Automaten auf jeden Fall disjunkt, da durch die Annotationen garantiert ist, dass die Automaten keine gemeinsamen Instanzen besitzen können. Es ist also nur ein Vergleich von Annotationsmengen notwendig, um die Disjunktheit der Wertemengen festzustellen.

Enthält eine Menge Δ_q für einen Automaten die beiden Regeln

$$\begin{array}{lcl} \{\ a : \{\ a : q_2(x_1)\ \}\ ,\ b : q_1(x_2)\ \} & \longrightarrow & q\ (\ \{\ a : \{\ a : x_1,\ b : x_2\ \}\ \}\ ) \\ \{\ a : q_1(x_1)\ \} & \longrightarrow & q\ (\ \{\ a : x_1\ \}\ ) \end{array}$$

so besitzen alle Instanzen des Automaten $\mathcal{T}_{1.Regel}$ die Form $\{\ a : v_1,\ b : v_2\ \}$ und alle Instanzen des Automaten $\mathcal{T}_{2.Regel}$ die Form $\{\ a : v_3\ \}$, so dass die Instanzenmengen disjunkt sind. Werden die Annotationen $\{\ a, b\ \}$ und $\{\ a\ \}$ auf oberster Ebene der linken Seiten der beiden Regeln miteinander verglichen, erkennt man, dass diese verschieden sind und daher die Wertemengen disjunkt. Es ist somit nur ein Vergleich von Regeln notwendig, um die Disjunktheit festzustellen. Wären die Annotationsmengen auf oberster Ebene beider Regeln hingegen gleich, könnten mit diesem einfachen Vergleich keine Aussagen gemacht werden.

Auswahl eines Übergangs durch Vorausschau

Bei einem kontextbezogen deterministischen Automaten sind in der Menge der Übergangsregeln Regeln erlaubt, die auf denselben Teilwert einer Eingabe passen. Sind solche Regeln vorhanden, besteht eine Auswahl, mit welcher dieser Regeln der nächste Übergang auszuführen ist. Enthält ein Eingabewert an einer beliebigen Stelle einen Teilwert $\{\ a : q_1(v_1'),\ b : q_2(v_2')\ \}$ – hat er also die Form

$$w\ [\ x\ /\ \{\ a : q_1(v_1'),\ b : q_2(v_2')\ \}\ ]$$

– und besitzt ein kontextbezogen deterministischer Automat zwei Übergangsregeln

$$\begin{array}{lcl} \{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q\ (\ u_1'\ ) \in \Delta \\ \{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q'\ (\ u_2'\ ) \in \Delta \end{array}$$

so passen die linken Seiten beider Regeln auf den Teilwert und es kann grundsätzlich mit beiden ein Übergang durchgeführt werden.

Bei einem kontextbezogen deterministischen Automaten gibt es aber für einen akzeptierten Wert nur gleiche Ableitungswege, so dass der Eingabewert nicht mit beiden Regeln nach q_f abgeleitet werden kann, wenn es sich bei ihm um eine Instanz des Automaten handelt. Da es nur gleiche Ableitungswege für eine Instanz gibt, gilt entweder

$$w\ [\ x\ /\ \{\ a : q_1(v_1'),\ b : q_2(v_2')\ \}\ ]\ \overset{1.Regel}{\longrightarrow}_{\mathcal{T}}\ \ldots\ \overset{*}{\longrightarrow}_{\mathcal{T}}\ q_f(v_1')$$

oder

$$w\ [\ x\ /\ \{\ a : q_1(v_1'),\ b : q_2(v_2')\ \}\ ]\ \overset{2.Regel}{\longrightarrow}_{\mathcal{T}}\ \ldots\ \overset{*}{\longrightarrow}_{\mathcal{T}}\ q_f(v_2')$$

Beides ist nicht möglich. Wird im zu akzeptierenden Wert eine Vorausschau auf den weiteren Ableitungsweg des Kontextes w vorgenommen, kann eine Entscheidung getroffen werden, ob ein Übergang mit der ersten oder der zweiten Regeln als nächstes durchzuführen ist, da einer der Wege in eine Sackgasse führt. Auf den Kontext bezogen ist die Auswahl der passenden Regel eindeutig festgelegt und somit deterministisch.

Beispiel 3.4 (*Kontextbezogen deterministische Automaten*)

1. Ist ein Automat $\mathcal{T} = (\{q_f, q_1, q_2\}, \{a, b, c\}, \{0, 1, 2\}, q_f, \Delta)$ mit den Regeln

$$\begin{array}{lcll} \{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) & \textit{1.Regel} \\ \{\ a : q_2(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) & \textit{2.Regel} \\ \{\ c\ \} & \longrightarrow & q_2\ (\ \{\ 0\ \}\ ) & \textit{3.Regel} \end{array}$$

   gegeben, kann überprüft werden, ob dieser kontextbezogen deterministisch ist. Dem Verfahren folgend werden zuerst die Δ_q für alle $q \in Q$ berechnet:

$$\begin{array}{l} \Delta_{q_f} = \{1.Regel\} \\ \Delta_{q_1} = \{2.Regel\} \\ \Delta_{q_2} = \{3.Regel\} \end{array}$$

   Offensichtlich beschreiben alle Δ_q eine Partitionierung des entsprechenden $dom(q)$ mit jeweils nur einer Partition. Somit ist der Automat kontextbezogen deterministisch und beschreibt eine Abbildung.

2. Ist ein Automat $\mathcal{T} = (\{q_f, q_1, q_2, q_3\}, \{a, b, c\}, \{0, 1, 2\}, q_f, \Delta)$ mit den Regeln

$$\begin{array}{lcll} \{\ a : q_1(x_1),\ b : q_3(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) & \textit{1.Regel} \\ \{\ a : q_2(x_1),\ b : q_3(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) & \textit{2.Regel} \\ \{\ c\ \} & \longrightarrow & q_2\ (\ \{\ 0\ \}\ ) & \textit{3.Regel} \\ \{\ c\ \} & \longrightarrow & q_3\ (\ \{\ 1\ \}\ ) & \textit{4.Regel} \end{array}$$

gegeben, kann wieder eine Überprüfung erfolgen, indem die Δ_q berechnet werden. Es gilt:

$$\Delta_{q_f} = \{1.Regel\}$$
$$\Delta_{q_1} = \{2.Regel\}$$
$$\Delta_{q_2} = \{3.Regel\}$$
$$\Delta_{q_3} = \{4.Regel\}$$

Jedes Δ_q beschreibt wieder eine Partitionierung des entsprechenden Wertebereichs $dom(q)$ mit nur einer Partition. Folglich ist dieser Automat ebenfalls kontextbezogen deterministisch.

Bei einer Ableitung für einen Wert $v = \{\ a : \{\ a : \{\ c\ \}\ , b : \{\ c\ \}\ \}\ , b : \{\ c\ \}\ \}$ hängt es von der Position des Werts $\{\ c\ \}$ innerhalb von v ab, ob der Übergang mit der dritten oder der vierten Regel durchgeführt wird. Tritt er als Teilwert unterhalb von a auf, wird er nach $\{\ 0\ \}$ abgebildet und unterhalb von b nach $\{\ 1\ \}$. Steht eine Entscheidung an, wann welche der beiden Regeln auszuführen ist, kann diese anhand des Kontextes getroffen werden. Gilt etwa $w = \{\ a : \{\ a : \{\ c\ \}\ , b : \{\ c\ \}\ \}\ , b : x\ \}$ und ist der Eingabewert $v = w\ [\ x\ /\ \{\ c\ \}\ ]$, so muss eine Ableitung mit der vierten Regel erfolgen, um den Finalzustand zu erreichen, da nur $w\ [\ x\ /\ \{\ c\ \}\ ] \overset{4.Regel}{\longrightarrow}_{\mathcal{T}} w\ [\ x\ /\ q_3(\ \{\ 1\ \}\ )\ ] \overset{*}{\longrightarrow}_{\mathcal{T}} \ldots \overset{*}{\longrightarrow}_{\mathcal{T}} q_f(v')$. Es erfolgt also während der Ableitung eine Betrachtung des Kontextes, innerhalb dessen der Wert auftritt, und anhand dessen eindeutig entschieden werden kann, welche der beiden Regeln auszuführen ist. Hierzu wird eine Vorausschau auf den weiteren Ableitungsweg vorgenommen.

3. Ist ein Automat $\mathcal{T} = (\{q_f, q_1, q_2, q_3\}, \{a, b, c\}, \{0, 1, 2\}, \Delta)$ mit den Regeln

$$\begin{array}{lcll}
\{\ a : q_1(x_1),\ b : q_3(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) & \textit{1.Regel} \\
\{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 2 : x_1,\ 2 : x_2\ \}\ ) & \textit{2.Regel} \\
\{\ a : q_2(x_1),\ b : q_3(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) & \textit{3.Regel} \\
\{\ c\ \} & \longrightarrow & q_2\ (\ \{\ 0\ \}\ ) & \textit{4.Regel} \\
\{\ c\ \} & \longrightarrow & q_3\ (\ \{\ 1\ \}\ ) & \textit{5.Regel}
\end{array}$$

gegeben, kann wieder eine Überprüfung erfolgen. Es gilt

$$\Delta_{q_f} = \{1.Regel, 2.Regel\}$$
$$\Delta_{q_1} = \{3.Regel\}$$
$$\Delta_{q_2} = \{4.Regel\}$$
$$\Delta_{q_3} = \{5.Regel\}$$

Bis auf Δ_{q_f} beschreiben alle Regelmengen eine Partitionierung mit einer Partition. Für Δ_{q_f} muss der Nachweis noch erbracht werden, dass diese Regeln eine Partitionierung von $dom(q_f)$ vornehmen.

Die Regeln aus Δ_{q_f} besitzen bei den Werten der Eingabeseiten beide dieselbe Menge $\{a, b\}$ an Annotationen auf der obersten Ebene, so dass eine Entscheidung anhand dieses Vergleichs nicht getroffen werden kann. Es müssen die beiden Automaten $\mathcal{T}_{1.Regel}$ und $\mathcal{T}_{2.Regel}$ konstruiert werden und deren Instanzenmengen auf Disjunktheit untersucht werden. Die Automaten ergeben sich als $\mathcal{T}_{1.Regel} = (\{q_f, q_1, q_2, q_3\}, \{a, b, c\}, \{0, 1, 2\}, \Delta_{1.Regel})$ mit den Regeln $\Delta_{1.Regel} = \{$

$$
\begin{array}{lcl}
\{\ a : q_1(x_1),\ b : q_3(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) \\
\{\ a : q_2(x_1),\ b : q_3(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) \\
\{\ c\ \} & \longrightarrow & q_2\ (\ \{\ 0\ \}\ ) \\
\{\ c\ \} & \longrightarrow & q_3\ (\ \{\ 1\ \}\ )
\end{array}
$$

$\}$ und $\mathcal{T}_{2.Regel} = (\{q_f, q_1, q_2, q_3\}, \{a, b, c\}, \{0, 1, 2\}, \Delta_{2.Regel})$ mit den Regeln $\Delta_{2.Regel} = \{$

$$
\begin{array}{lcl}
\{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 2 : x_1,\ 2 : x_2\ \}\ ) \\
\{\ a : q_2(x_1),\ b : q_3(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ 1 : x_1,\ 2 : x_2\ \}\ ) \\
\{\ c\ \} & \longrightarrow & q_2\ (\ \{\ 0\ \}\ ) \\
\{\ c\ \} & \longrightarrow & q_3\ (\ \{\ 1\ \}\ )
\end{array}
$$

$\}$. Es muss $dom(\mathcal{T}_{1.Regel}) \cap dom(\mathcal{T}_{2.Regel}) = \emptyset$ gezeigt werden, wobei die Überprüfung ähnlich wie in [CDG+99] beschrieben erfolgen kann. Es zeigt sich, dass der Schnitt den Wert $\{\ a : \{\ a : \{\ c\ \}\ ,\ b : \{\ c\ \}\ \}\ ,\ b : \{\ c\ \}\ \}$ enthält. Somit beschreibt Δ_{q_f} keine Partitionierung und folglich ist $\mathcal{T}$ nicht kontextbezogen deterministisch.

□

Ist ein kontextbezogen deterministischer Automat mit Ausgabe $\mathcal{T}$ gegeben, so erfüllt dieser folgende Eigenschaften:

- Es existieren nur gleiche Ableitungswege für alle Instanzen $v \in dom(\mathcal{T})$.
- $f_{\mathcal{T}}$ ist eine Abbildung auf Instanzenebene (Abb. 3.6).

Ein Baumautomat mit Ausgabe erlaubt die Abbildung von Werten aufeinander. Dadurch, dass eine Abbildung beim Durchlaufen eines Werts durch den Automaten vorgenommen wird, wobei jede Ebene des Eingabebaums durch einen Ausgabewert ersetzt wird, wird ein Teilwert der Eingabe nur einmal abgebildet. Somit sind die Anforderungen aus Abschnitt 2.5 durch diese Art der Formulierung einer Abbildung erfüllt.

3.1.4 Operationen

Um eine Abbildungsbeschreibung zu komplettieren, werden noch Änderungsoperatoren auf Bäumen benötigt, wobei ein Operator einen Baum in einen anderen Baum umwandelt. Da Operationen zwischen zwei beliebigen Bäumen möglich sein sollen, müssen

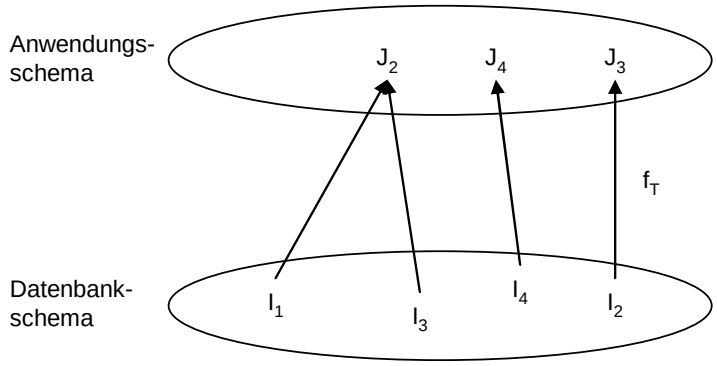


Abbildung 3.6: *Abbildung zwischen Instanzen*

die Operatoren so angelegt sein, dass Änderungsoperationen zwischen zwei beliebigen Bäumen durch Folgen von Operatoren beschrieben werden können.

Verfolgt man die Annotationssymbole von der Wurzel eines Baums bis zu einem Blatt, entsteht ein Pfad durch die Aneinanderreihung der besuchten Annotationssymbole. Jeder Baum aus $\mathcal{V}(\Sigma)$ impliziert daher eine Menge von Pfaden über Σ. Für einen Wert $v \in \mathcal{V}(\Sigma)$ entsteht die Menge der Pfade $L(v) \subseteq \Sigma^*$ durch folgende rekursive Vorschriften:

- Ist $v = \{\ a_1 : v_1, \ldots, a_n : v_n\ \}$, dann ist $L(v) = \bigcup_{1=1}^{n} a_i L(v_i)$.
- Ist $v = \{\ a_1, \ldots, a_n\ \}$, dann ist $L(v) = \{a_1, \ldots, a_n\}$.
- Ist $v = \{\ \}$, dann ist $L(v) = \{\epsilon\}$.

Hierbei ist $aL(v) = \{aw \mid w \in L(v)\}$ als Verkettung einer Annotation a mit jedem Pfad aus $L(v)$ zu verstehen. ϵ wird als spezielles Zeichen behandelt, dass für den leeren Wert steht.

Zu einer Menge von Pfaden P ergibt sich wieder ein Wert, indem die Pfade entsprechend gruppiert und dann die Annotationen einer Ebene erzeugt werden. $L^{-1}(P)$ liefert zu einer Pfadmenge P einen Wert durch

- Ist $P = \bigcup_{i=1}^{n} a_i P_i$, dann $L^{-1}(P) = \{\ a_1 : L^{-1}(P_i), \ldots, a_n : L^{-1}(P_n)\ \}$
- Ist $P = \{a_1, \ldots, a_n\}$, dann $L^{-1}(P) = \{\ a_1, \ldots, a_n\ \}$
- Ist $P = \{\epsilon\}$, dann $L^{-1}(P) = \{\ \}$

Die Verkettung von a mit allen Pfaden aus P wird durch aP dargestellt.

Bei dieser Betrachtungsweise bewirkt eine Änderungsoperation zwischen zwei Werten einen Wechsel von einer Pfadmenge zu einer anderen Pfadmenge. Eine solche Änderung kann durch das Hinzu- oder Wegnehmen von Pfaden zur Ausgangspfadmenge formuliert

werden. Auf Basis der Pfaddarstellung von Werten werden zwei Änderungsoperatoren auf der Menge der Werte $\mathcal{V}(\Sigma)$ definiert:

- Ein Vereinigungsoperator

  $$union : \mathcal{V}(\Sigma) \times \mathcal{V}(\Sigma) \rightarrow \mathcal{V}(\Sigma)$$

  auf Bäumen ist definiert als $union(v,w) := L^{-1}(L(v) \cup L(w))$ und berechnet den Wert, der sich aus der Vereinigung der Pfadmengen der beiden Eingabewerte ergibt.

- Ein Differenzoperator

  $$diff : \mathcal{V}(\Sigma) \times \mathcal{V}(\Sigma) \rightarrow \mathcal{V}(\Sigma)$$

  ist definiert als $diff(v,w) := L^{-1}(L(v) - L(w))$ und berechnet den Wert, der sich aus der Differenzbildung der Pfadmengen der beiden Eingabewerte ergibt.

Sind zwei beliebige Bäume v und w gegeben, kann ein Wechsel vom einen zum anderen Wert mit Hilfe der beiden definierten Operatoren *union* und $diff$ beschrieben werden. Eine Operation $\nu(v) = w$ ergibt sich als Operatorenfolge

$$union(diff(v, v_{\nu-}), v_{\nu+}) = w$$

wobei die Definition von ν den Wert $v_{\nu-}$ als $diff(v, w)$ und den Wert $v_{\nu+}$ als $diff(w, v)$ definiert. $v_{\nu-}$ enthält die Pfade, die aus v entfernt werden müssen, um zu w zu gelangen, und $v_{\nu+}$ enthält die Pfade, die zu v hinzugefügt werden müssen.

Beispiel 3.5 (*Baumoperatoren*)
Sind zwei Werte $v = \{\ a : \{\ a : \{\ c\ \}\ ,\ b : \{\ a\ \}\ \}\ ,\ b : \{\ c\ \}\ \}$ und $w = \{\ a : \{\ b\ \}\ ,\ b : \{\ c\ \}\ \}$ gegeben, so ergibt sich $L(v) = \{aac, aba, bc\}$ und $L(w) = \{ab, bc\}$. Die Operation $diff(v,w)$ liefert den Wert $L^{-1}(\{aac, aba\}) = \{\ a : \{\ a : \{\ c\ \}\ ,\ b : \{\ a\ \}\ \}\ \}$. Abb. 3.7 zeigt die Differenzbildung zwischen den beiden Werten.

□

Die Operatoren *union* und $diff$ sind auf der Menge der Bäume definiert und somit unabhängig von einer durch einen Automaten beschriebenen Menge von Bäumen. Ein Operator führt erst dann eine Operation auf den Instanzen eines gegebenen Automaten durch, wenn sowohl der Ausgangswert als auch das Ergebnis der Operation beide durch den Automaten akzeptiert werden.

Der *union*-Operator ist nicht für alle Wertpaare aus $\mathcal{V}(\Sigma)$ definiert. Eine Vereinigung der beiden Werte $v = \{\ a,\ b\ \}$ und $w = \{\ a : \{\ c\ \}\ \}$ ist nicht möglich, da L^{-1} für die Pfadmenge $L(v) \cup L(w)$ undefiniert ist. Der Ergebniswert einer Vereinigung wäre $\{\ a : \{\ c\ \}\ ,\ b\ \}$, wobei dieser ungültig ist, da bei der Annotation a auf der obersten Ebene ein Teilwert folgt und hinter der Annotation b kein Teilwert angegeben ist. Da die beiden Operatoren aber verwendet werden, um den Weg von einem gültigen Wert zu einem anderen gültigen Wert aus $\mathcal{V}(\Sigma)$ zu beschreiben, treten undefinierte Operationen bei den weiteren Betrachtungen nicht auf.

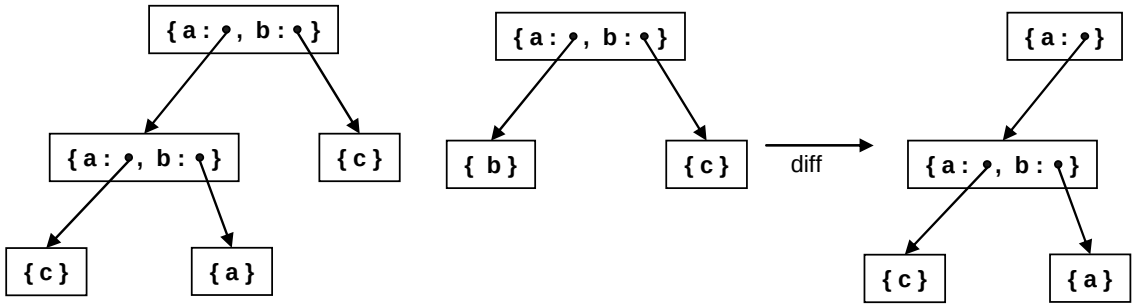


Abbildung 3.7: *Differenz zweier Werte*

3.2 Erfüllung der Anforderungen

In Abschnitt 3.1 wird ein Abbildungsmechanismus vorgestellt, mit dessen Hilfe Abbildungen zwischen Bäumen beschrieben werden können. Es gilt zu zeigen, dass dieser Mechanismus geeignet ist, die semantisch korrekten Operationen bei einer Abbildung zu bestimmen. Hierfür ist die Existenz eines Zielschemas für jede definierbare Abbildung nachzuweisen und zu zeigen, wie anhand einer Abbildung getestet werden kann, ob diese injektiv ist.

3.2.1 Existenz eines Zielschemas

Werden Abbildungen durch Baumautomaten beschrieben, muss es diese Abbildungsbeschreibung erlauben, ein Zielschema für jede Abbildung anzugeben, so dass die Seiteneffektfreiheit von Änderungsoperationen überprüft werden kann.

Ist ein Baumautomat mit Ausgabe $\mathcal{T}$ gegeben, so legt dieser ein Quellschema in Form eines Automaten ohne Ausgabe $\mathcal{A}_S$ fest, dessen Instanzen genau den Definitionsbereich der Relation $f_\mathcal{T}$ beschreiben. Dieses Schema $\mathcal{A}_S$ leitet sich aus dem Automaten $\mathcal{T}$ ab, indem bei den Übergangsregeln die Ausgabewerte wegfallen, wobei $dom(f_\mathcal{T}) = dom(\mathcal{A}_S)$ gilt. Kann man für die Bildmengen von Relationen ebenfalls jeweils ein Schema in Form eines Automaten ohne Ausgabe finden, welches die dort vorkommenden Werte als Instanzen akzeptiert, dann gibt es für alle durch einen Automaten mit Ausgabe definierten Relationen je ein Zielschema $\mathcal{A}_T$ mit $range(f_\mathcal{T}) = dom(\mathcal{A}_T)$.

Theorem 3.3 (*Existenz eines Quell- und Zielschemas*)
Für jede durch einen Baumautomaten mit Ausgabe $\mathcal{T}$ definierte Relation $f_\mathcal{T} \subseteq \mathcal{V}(\Sigma) \times \mathcal{V}(\Sigma')$ existiert ein Quellschema $\mathcal{A}_S$ mit $dom(\mathcal{A}_S) = dom(f_\mathcal{T})$ und ein Zielschema $\mathcal{A}_T$ mit $dom(\mathcal{A}_T) = range(f_\mathcal{T})$. □

Beweis Sei ein Baumautomat mit Ausgabe $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ gegeben, dann wird

die Menge der Eingabewerte durch $\mathcal{A}_S = (Q, \Sigma, q_f, \Delta_S)$ mit

$$\begin{array}{lcl} w\ [\ y_1\ /\ q_1, \ldots, y_n\ /\ q_n\ ] & \longrightarrow & q \in \Delta_S \\ & \text{gdw} & \\ w\ [\ y_1\ /\ q_1(x_1), \ldots, y_n\ /\ q_n(x_n)\ ] & \longrightarrow & q\ (\ w'\ [\ y_1\ /\ x_1, \ldots, y_n\ /\ x_n\ ]\ ) \in \Delta \end{array}$$

beschrieben. Die Übergangsregeln von $\mathcal{A}_S$ ergeben sich dadurch, dass der Ausgabewert bei der entsprechenden Regel von $\mathcal{T}$ gestrichen wird. Offensichtlich akzeptieren $\mathcal{T}$ und $\mathcal{A}_S$ dieselbe Menge von Werten, und da genau diese Werte abgebildet werden, modelliert $\mathcal{A}_S$ das Quellschema von $\mathcal{T}$. Es gilt $dom(\mathcal{A}_S) = dom(f_\mathcal{T})$.

Auf ähnliche Weise wird ein Schema $\mathcal{A}_T = (Q, \Sigma', q_f, \Delta_T)$ konstruiert, welches die Menge der Ausgabewerte von $\mathcal{T}$ umfasst mit

$$\begin{array}{lcl} w'\ [\ y_1\ /\ q_1, \ldots, y_n\ /\ q_n\ ] & \longrightarrow & q \in \Delta_T \\ & \text{gdw} & \\ w\ [\ y_1\ /\ q_1(x_1), \ldots, y_n\ /\ q_n(x_n)\ ] & \longrightarrow & q\ (\ w'\ [\ y_1\ /\ x_1, \ldots, y_n\ /\ x_n\ ]\ ) \in \Delta \end{array}$$

Bei jeder Übergangsregel aus $\mathcal{T}$ wird in $\mathcal{A}_T$ anstelle des Eingabewerts w der Ausgabewert w' verwendet und entsprechend die Vorkommen von x_i auf der rechten Seite durch q_i ersetzt. Da bei einem Automaten mit Ausgabe jeder Wert der Ausgabe auch tatsächlich durch einen Eingabewert generiert wird, akzeptiert der so erzeugte Automat $\mathcal{A}_T$ genau die Menge der Ausgabewerte von $\mathcal{T}$, so dass $dom(\mathcal{A}_T) = range(\mathcal{T})$. $\mathcal{A}_T$ modelliert das Zielschema der Abbildung. □

Jede Relation zwischen Wertemengen, die durch einen Baumautomaten mit Ausgabe beschrieben wird, besitzt somit ein Zielschema, das aus der Abbildungsdefinition selbst abgeleitet werden kann und genau die Werte des Bildbereichs der Relation als Instanzen umfasst. Die Forderung aus Abschnitt 2.2.3 nach der Existenz eines Zielschemas für jede formulierbare Abbildung ist somit erfüllt.

Dadurch, dass ein Zielschema vorhanden ist, können Baumautomaten mit Ausgabe auch miteinander verkettet werden, indem das Zielschema des ersten Automaten $\mathcal{T}_1$ die Rolle des Quellschemas des zweiten Automaten $\mathcal{T}_2$ übernimmt. Bei einer Verkettung ergibt sich die Relation des entstehenden Automaten als $f_{\mathcal{T}_2}(f_{\mathcal{T}_1}(v))$, welche einem Eingabewert v die entsprechenden Ausgabewerte zuweist.

Würde die Forderung nach Linearität im Ausgabewert einer Übergangsregel bei der Definition eines Automaten mit Ausgabe wegfallen, könnten Baumautomaten definiert werden, die kein Zielschema besitzen, da dann Abhängigkeiten zwischen Unterbäumen auftreten könnten, die nicht durch einen Automaten modelliert werden können. Betrachtet man zum Beispiel einen (ungültigen) Automaten mit folgenden Übergangsregeln

$$\begin{array}{lcl} \{\ a : q_f(x_1)\ \} & \longrightarrow & q_f\ (\ \{\ a : x_1, b : x_1\ \}\ ) \\ \{\ a\ \} & \longrightarrow & q_f\ (\ \{\ a\ \}\ ) \end{array}$$

wobei bei der ersten Übergangsregel die Variable x_1 mehrfach in der Ausgabe auftritt, dann ist der Ausgabewert nicht linear. Dies hat zur Folge, dass bei einer Ableitung zwei

Kopien von dem an die Variable x_1 gebundenen Teilwert in den Ausgabewert eingefügt werden. Wird ein Wert

$$\{\ a : \underbrace{\{\ a : \{\ \ldots\}\ \}}_{n}\ \}$$

der neben dem a auf oberster Ebene noch n geschachtelte a enthält, akzeptiert, dann entsteht die Ausgabe

$$\{\ a : \underbrace{\{\ a : \{\ \ldots\}\ \}}_{n}, b : \underbrace{\{\ a : \{\ \ldots\}\ \}}_{n}\ \}$$

Der Ausgabewert enthält identische Unterbäume mit je n geschachtelten a unterhalb der obersten Ebene. Soll ein Zielschema konstruiert werden, welches diese Ausgabewerte beschreibt, müsste bei diesem eine Regel $\{\ a : q_1, b : q_2\ \} \longrightarrow q_f$ mit q_f als Finalzustand auftreten, wobei garantiert werden muss, dass die nach q_1 und q_2 abgeleiteten Werte identisch sind. Die einzige Möglichkeit, dies zu garantieren, besteht darin, für jedes mögliche n einen entsprechenden Zustand q_n anzulegen, der dann erreicht wird, wenn im abgeleiteten Wert die Annotation a n-mal vorkommt, und Finalregeln der Form $\{\ a : q_n, b : q_n\ \} \longrightarrow q_f$ anzugeben. Da n nach oben nicht beschränkt ist, wäre die Anzahl der benötigten Zustände q_n ebenfalls nach oben unbeschränkt, was der Definition eines Automaten widerspricht, da dessen Zustandsmenge endlich sein muss.

3.2.2 Injektivität einer Abbildung

Neben der Existenz eines Zielschemas spielt die Injektivität einer Abbildung eine Rolle bei einem Test auf semantische Korrektheit von Operationen, um die Eindeutigkeit von Änderungsoperationen nachzuweisen. Auch diese Eigenschaft einer Abbildung kann anhand eines Baumautomaten überprüft werden.

Da bei einem Baumautomaten bei allen Übergangsregeln die Menge der Variablen auf der linken Seite gleich der Menge der Variablen auf der rechten Seite ist, kann zu jedem Automaten mit Ausgabe ein inverser Baumautomat definiert werden, welcher den Ausgabewerten wieder Eingabewerte zuweist.

Definition 3.3 (*Inverser Baumautomat*)
Sei ein Baumautomat $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ gegeben, dann ergibt sich der *inverse Baumautomat* $\mathcal{T}^{-1} = (Q, \Sigma', \Sigma, q_f, \Delta^{-1})$ mit

$$\Delta^{-1} \quad = \quad \{\ invert(\ u \longrightarrow q(u')\ ) \mid u \longrightarrow q(u') \in \Delta\ \}$$

wobei die Funktion *invert* auf der Menge der Regeln als

$$invert(w[y_1\ /\ q_1(x_1), \ldots, y_n\ /\ q_n(x_n)] \longrightarrow q(w'[y_1\ /\ x_1, \ldots, y_n\ /\ x_n])) :=$$
$$w'[y_1\ /\ q_1(x_1), \ldots, y_n\ /\ q_n(x_n)] \longrightarrow q(w[y_1\ /\ x_1, \ldots, y_n\ /\ x_n])$$

definiert ist.

Ein inverser Automat entsteht durch das jeweilige Vertauschen der Ein- und Ausgabeseiten aller Übergangsregeln eines Automaten. Insgesamt betrachtet, werden so Quell- und Zielschema gegeneinander vertauscht. Aus den ursprünglichen Eingabewerten werden somit Ausgabewerte und umgekehrt (Abb. 3.8).

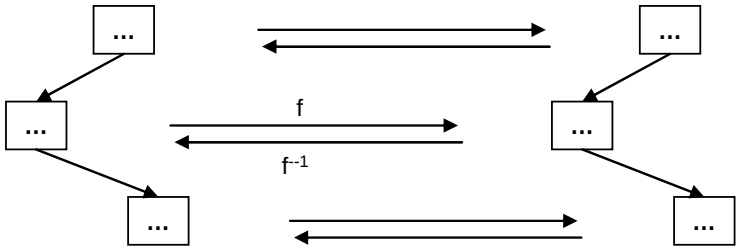


Abbildung 3.8: *Eine inverse Abbildung f^{-1} macht das Ersetzen von Ebenen durch eine Abbildung f bei einer Ableitung wieder rückgängig.*

Beispiel 3.6 (*Inverser Automat*)
Ein Baumautomat mit Ausgabe ist zum Beispiel $\mathcal{T} = (\{q_f, q_1, q_2\}, \{a, b, c\}, \{0, 1, 2, 3\}, q_f, \Delta)$ mit $\Delta = \{$

$$
\begin{array}{lcl}
\{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 0 : x_1,\ 1 : x_2\ \}\ ) \\
\{\ a : q_2(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ 2 : x_1,\ 3 : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_2\ (\ \{\ 0\ \}\ ) \\
\{\ c\ \} & \longrightarrow & q_2\ (\ \{\ 1\ \}\ )
\end{array}
$$

$\}$. Der inverse Automat $\mathcal{T}^{-1} = (\{q_f, q_1, q_2\}, \{0, 1, 2, 3\}, \{a, b, c\}, q_f, \Delta^{-1})$ entsteht durch das Vertauschen der Ein- und Ausgabewerte der einzelnen Übergangsregeln mit $\Delta^{-1} = \{$

$$
\begin{array}{lcl}
\{\ 0 : q_1(x_1),\ 1 : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ 2 : q_2(x_1),\ 3 : q_2(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ 0\ \} & \longrightarrow & q_2\ (\ \{\ a\ \}\ ) \\
\{\ 1\ \} & \longrightarrow & q_2\ (\ \{\ c\ \}\ )
\end{array}
$$

$\}$. □

Die Existenz eines inversen Automaten garantiert noch nicht, dass der inverse Automat auch eine Abbildung beschreibt, d.h. es ist nicht sichergestellt, dass die ursprüngliche Abbildung injektiv ist. Besitzt ein Automat mit Ausgabe die Regeln

$$
\begin{array}{lcl}
\{\ a : q_1(x_1),\ a : q_1(x_2)\ \} & \longrightarrow & q_f(\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_1(\ \{\ b\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_1(\ \{\ b\ \}\ )
\end{array}
$$

beschreibt dieser eine Abbildung. Der inverse Automat ergibt sich durch das Vertauschen der Ein- und Ausgabewerte als

$$\begin{array}{lcl} \{\ a : q_1(x_1),\ b : q_1(x_2)\ \} & \longrightarrow & q_f(\ \{\ a : x_1,\ a : x_2\ \}\ ) \\ \{\ b\ \} & \longrightarrow & q_1(\ \{\ a\ \}\ ) \\ \{\ b\ \} & \longrightarrow & q_1(\ \{\ b\ \}\ ) \end{array}$$

Der inverse Automat beschreibt keine Abbildung, da einem Eingabewert $\{\ a : \{\ b\ \}\ ,\ b : \{\ b\ \}\ \}$ unter anderem sowohl $\{\ a : \{\ b\ \}\ ,\ a : \{\ b\ \}\ \}$ als auch $\{\ a : \{\ a\ \}\ ,\ a : \{\ a\ \}\ \}$ als Ausgabe zugewiesen wird.

Existiert zu einem gegeben Baumautomaten ein inverser Automat und sind beide kontextbezogen deterministisch, d.h. wird durch beide eine Abbildung beschrieben, dann wird durch den Automaten selbst jedem Eingabewert eindeutig ein Ausgabewert zugewiesen und durch den inversen Automaten jedem Ausgabewert eindeutig ein Eingabewert. Die durch den Automaten beschriebene Abbildung ist also injektiv auf Instanzebene.

Theorem 3.4 (*Injektive Abbildung*)
Sind ein Baumautomat $\mathcal{T}$ und sein inverser Automat $\mathcal{T}^{-1}$ beide kontextbezogen deterministisch, dann ist die Abbildung zwischen Instanzen der Ein- und Ausgabe injektiv. □

Beweis Sei ein Baumautomat $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ gegeben. Da $\mathcal{T}$ kontextbezogen deterministisch ist, gibt es für jede Instanz des Automaten genau einen Ausgabewert. $\mathcal{T}$ definiert also eine Abbildung $f_{\mathcal{T}} : dom(\mathcal{A}_S) \rightarrow dom(\mathcal{A}_T)$ zwischen dem Quell- und dem Zielschema des Automaten mit $f_{\mathcal{T}}(v) = v'$ wenn $v \xrightarrow{*}_{\mathcal{T}} q_f(v')$. Ist der inverse Automat ebenfalls kontextbezogen deterministisch, gilt hier $v' \xrightarrow{*}_{\mathcal{T}^{-1}} q_f(v)$, da im inversen Automaten die Seiten aller Übergangsregeln vertauscht sind. Es ergibt sich eine Abbildung $f_{\mathcal{T}^{-1}} : dom(\mathcal{A}_T) \rightarrow dom(\mathcal{A}_S)$ und es gilt $f_{\mathcal{T}^{-1}}(f_{\mathcal{T}}(v)) = v$ für alle $v \in dom(\mathcal{A}_S)$. Somit ist die Abbildung auch injektiv, da eine Umkehrabbildung existiert. □

Sind ein Automat und sein inverser Automat kontextbezogen deterministisch, dann wird durch ersteren eine informationserhaltende Sicht auf der Menge der Werte des Definitionsbereichs beschrieben, da jeder Instanz des Definitionsbereichs eindeutig eine Instanz des Bildbereichs zugeordnet wird und umgekehrt (Abb. 3.9). Anhand einer Abbildungsdefinition kann überprüft werden, ob die Abbildung injektiv ist, indem überprüft wird, ob die beteiligten Automaten kontextbezogen deterministisch sind. Somit ist die Anforderung aus Abschnitt 2.2.3 erfüllt.

3.3 Bestimmung semantisch korrekter Operationen

Ist ein gegebener Baumautomat $\mathcal{T}$ kontextbezogen deterministisch, dann beschreibt dieser eine Abbildung $f_{\mathcal{T}}$. Eine Instanz $\mathbf{I_1}$ des Definitionsbereichs wird durch den Automaten auf eine Instanz $\mathbf{J_1}$ des Bildbereichs abgebildet (Abb. 3.10). Wird auf dieser

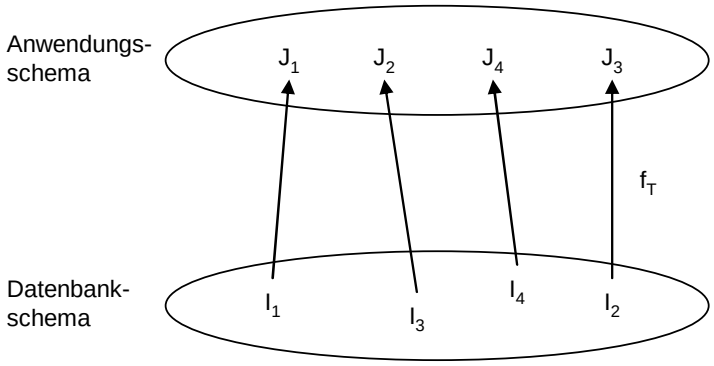


Abbildung 3.9: *Informationserhaltende Sicht zwischen Werten*

Instanz eine Änderungsoperation ν ausgeführt, gehört der resultierende Wert entweder zum Bildbereich der Abbildung oder er gehört nicht dazu. Wird der Wert vom immer vorhandenen inversen Automaten $\mathcal{T}^{-1}$ als Instanz akzeptiert, gehört er dazu und die Änderungsoperation ν ist somit seiteneffektfrei. Ist der inverse Automat kontextbezogen deterministisch, dann sind alle Operationen zwischen Instanzen des Bildbereichs eindeutig und durch $f_{\mathcal{T}^{-1}}$ wird bei der Akzeptanz der geänderten Bildinstanz $\nu(\mathbf{J_1})$ die zugehörige Instanz $\mu(\mathbf{I_1})$ des Definitionsbereichs berechnet. Somit ist die Änderungsoperation ν semantisch korrekt durchführbar. Eine Operatorfolge für μ ergibt sich als $union(diff(\mathbf{I_1}, diff(\mathbf{I_1}, \mathbf{I_2})), diff(\mathbf{I_2}, \mathbf{I_1}))$ mit $\mu(\mathbf{I_1}) = \mathbf{I_2}$.

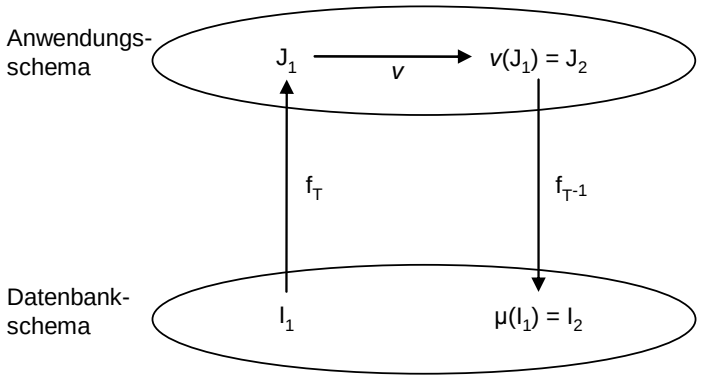


Abbildung 3.10: *Beispielabbildung* $f_{\mathcal{T}}$

Ist der inverse Automat kontextbezogen deterministisch, dann beschreibt die Abbildung eine Eins-zu-eins-Zuordnung zwischen den Instanzen des Definitions- und des Bildbereichs und jede Änderungsoperation zwischen den Instanzen des Bildbereichs ist se-

mantisch korrekt.

Beispiel 3.7 (*Operationsausführung*)
Ein Baumautomat mit Ausgabe ist zum Beispiel $\mathcal{T} = (\{q_f, q_1, q_2\}, \{a, b, c\}, \{0, 1, 2, 3\}, q_f, \Delta)$ mit $\Delta = \{$

$$\begin{array}{lcl}
\{\ a : q_1(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ 0 : x_1,\ 1 : x_2\ \}\ ) \\
\{\ a : q_2(x_1),\ b : q_2(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ 2 : x_1,\ 3 : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_2\ (\ \{\ 0\ \}\ ) \\
\{\ c\ \} & \longrightarrow & q_2\ (\ \{\ 1\ \}\ )
\end{array}$$

$\}$. Der Automat ist kontextbezogen deterministisch, da alle Δ_q je eine Partitionierung für $dom(q)$ festlegen. Alle Δ_q bis auf Δ_{q_2} enthalten eine Regel und somit nur ein Partition. Bei den Regeln in Δ_{q_2} unterscheiden sich die Annotationsmengen $\{a\}$ und $\{c\}$ auf den Eingabeseiten, so dass hier ebenfalls eine Partitionierung vorliegt. Der Automat definiert eine Abbildung $f_{\mathcal{T}} : dom(\mathcal{A}_S) \to dom(\mathcal{A}_T)$, welche die Instanzen des Quellschemas $\mathcal{A}_S$ den Instanzen des Zielschemas $\mathcal{A}_T$ zuweist.

Es existiert ein inverser Automat $\mathcal{T}^{-1}$, bei dem die Regelmenge aus der Regelmenge von $\mathcal{T}$ entsteht, indem die Eingabe- und Ausgabewerte bei jeder Regel vertauscht werden. Somit ist $\mathcal{T}^{-1} = (\{q_f, q_1, q_2\}, \{0, 1, 2, 3\}, \{a, b, c\}, q_f, \Delta^{-1})$ mit $\Delta^{-1} = \{$

$$\begin{array}{lcl}
\{\ 0 : q_1(x_1),\ 1 : q_2(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ 2 : q_2(x_1),\ 3 : q_2(x_2)\ \} & \longrightarrow & q_1\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ 0\ \} & \longrightarrow & q_2\ (\ \{\ a\ \}\ ) \\
\{\ 1\ \} & \longrightarrow & q_2\ (\ \{\ c\ \}\ )
\end{array}$$

$\}$. Der inverse Automat ist ebenfalls kontextbezogen deterministisch, da alle Δ_q jeweils die Mengen $dom(q)$ partitionieren. Bei den Regeln aus Δ_{q_2} unterscheiden sich die Annotationsmengen $\{0\}$ und $\{1\}$ der Regeln, so dass auch hier eine Partitionierung vorliegt.

Ist ein Wert $\mathbf{I_1} = \{\ a : \{\ a : \{\ c\ \}\ , b : \{\ a\ \}\ \}\ , b : \{\ c\ \}\ \}$ als Instanz des Quellschemas gegeben (Abb. 3.11), so wird dieser durch den Automaten $\mathcal{T}$ auf $\mathbf{J_1} = \{\ 0 : \{\ 2 : \{\ 1\ \}\ , 3 : \{\ 0\ \}\ \}\ , 1 : \{\ 1\ \}\ \}$ im Zielschema $\mathcal{A}_T$ der Abbildung abgebildet. Eine Änderungsoperation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ auf diesem Wert führt etwa zu einem Ergebniswert $\mathbf{J_2} = \{\ 0 : \{\ 2 : \{\ 0\ \}\ , 3 : \{\ 0\ \}\ \}\ , 1 : \{\ 0\ \}\ \}$, bei der die $\{\ 1\ \}$-Werte nach $\{\ 0\ \}$ geändert werden, indem die entsprechenden Pfade entfernt und neue eingefügt werden. $\mathbf{J_2}$ ist ebenfalls eine Instanz des Zielschemas $\mathcal{A}_T$, da $\mathbf{J_2}$ von $\mathcal{T}^{-1}$ akzeptiert wird. Folglich ist die Änderungsoperation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ seiteneffektfrei, da sie zu einem Wert innerhalb des Bildbereichs der Abbildung führt. Da beide Automaten kontextbezogen deterministisch sind, ist die durch $\mathcal{T}$ beschriebene Abbildung eine informationserhaltende Sicht auf den Werten und somit die Änderungsoperation $\nu(\mathbf{J_1}) := \mathbf{J_2}$ auch eindeutig. Es gibt genau eine Instanz $\mathbf{I_2}$ innerhalb des Quellschemas, die der Instanz $\mathbf{J_2}$ durch $f_{\mathcal{T}}$ zugewiesen ist. Diese Instanz $\mathbf{I_2}$ kann über die Akzeptanz von $\mathbf{J_2}$ durch $\mathcal{T}^{-1}$ berechnet werden und ergibt sich hierbei als $\mathbf{I_2} = \{\ a : \{\ a : \{\ a\ \}\ , b : \{\ a\ \}\ \}\ , b : \{\ a\ \}\ \}$. Die Änderungsoperation ν auf dem Zielschema ist somit semantisch korrekt durchführbar und die Operation $\mu(\mathbf{I_1}) = \mathbf{I_2}$

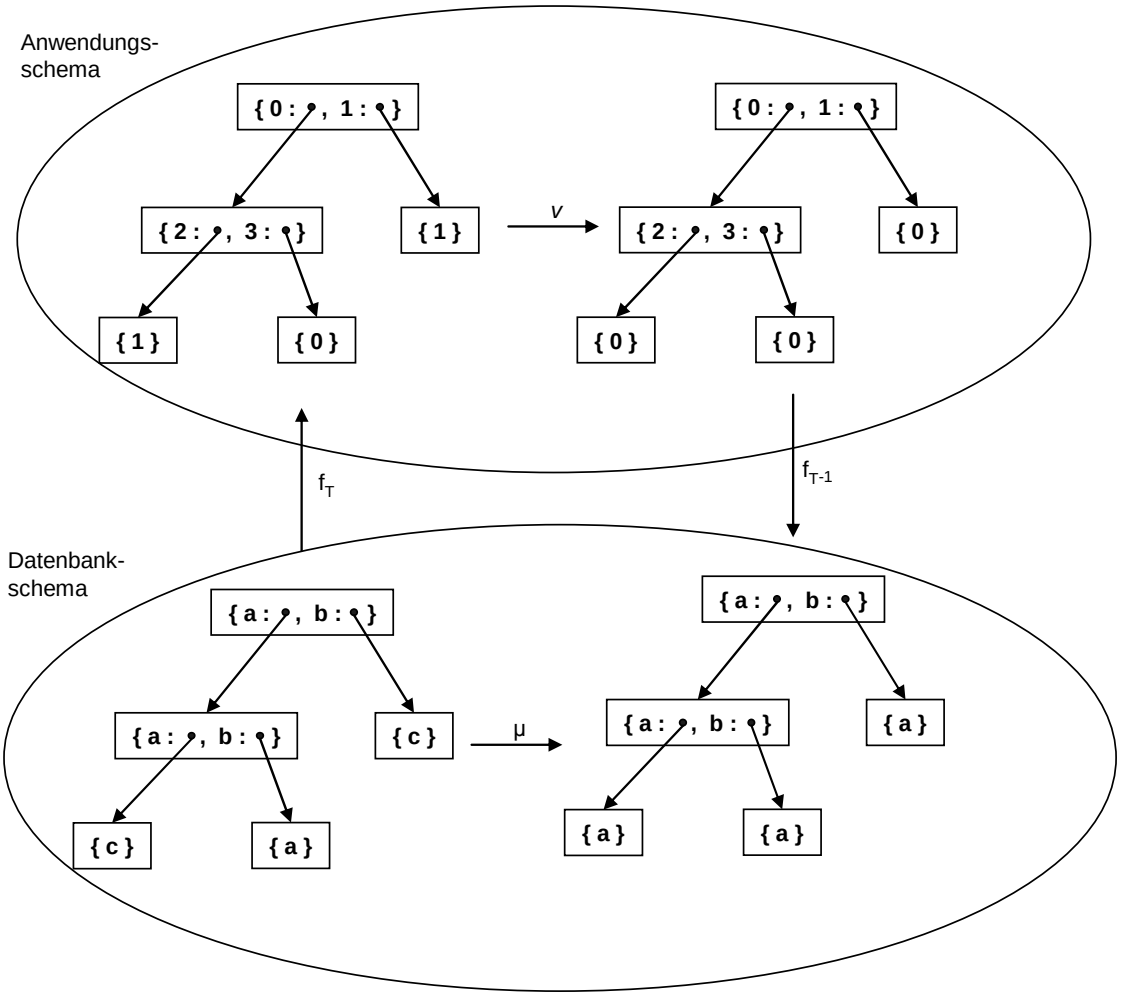


Abbildung 3.11: *Durchführung einer Operation* ν

ist eine Umsetzung für die Anwendungsoperation $\nu(\mathbf{J_1}) = \mathbf{J_2}$. Die Operatorfolge für μ ergibt sich als $union(diff(\mathbf{I_1}, diff(\mathbf{I_1}, \mathbf{I_2})), diff(\mathbf{I_2}, \mathbf{I_1}))$, wobei $diff(\mathbf{I_1},\mathbf{I_2}) = \{aac, bc\}$ und $diff(\mathbf{I_2},\mathbf{I_1}) = \{aaa, ba\}$. Die zu $\mathbf{I_2}$ fehlenden Pfade werden in $\mathbf{I_1}$ ergänzt und die überflüssigen Pfade entfernt. □

3.4 Fazit

In diesem Abschnitt wird eine Abbildungsbeschreibung in Form von Baumautomaten eingeführt und gezeigt, dass diese den Anforderungen aus Kapitel 2 genügt. Werte werden in Form von Bäumen repräsentiert, wobei die Kindknoten über Annotationen eindeutig

ansprechbar sind. Eine Menge von Bäumen wird durch einen Baumautomaten beschrieben, der Werte akzeptieren oder ablehnen kann. Die Menge der akzeptierten Bäume bildet seine Instanzen. Wird bei der Akzeptanz eines Werts durch den Automaten eine Ausgabe generiert, definiert der Automat eine Relation zwischen zwei Wertemengen. Gibt es für jeden akzeptierten Wert nur gleiche Ableitungswege, dann definiert der Automat eine Abbildung, wobei Automaten mit dieser Eigenschaft als kontextbezogen deterministisch bezeichnet werden. Für jede durch einen Automaten definierte Abbildung ist ein Zielschema vorhanden und für jeden Automat ist ein inverser Automat definiert. Ist der inverse Automat ebenfalls kontextbezogen deterministisch, wird eine informationserhaltende Sicht beschrieben.

Da es immer ein Zielschema gibt und die Abbildungseigenschaften anhand des Aufbaus der Regelmenge eines Automaten bestimmt werden können, ist eine Entscheidung, ob eine Änderungsoperation semantisch korrekt ist, für alle Abbildungen und alle Operationen auf dem Anwendungsschema möglich. Die Abbildungsbeschreibung durch annotierte Bäume eignet sich also dazu, die semantisch korrekten Operationen bei einem Abbildungsproblem zu bestimmen. In den folgenden Kapiteln soll nun dieses Datenmodell bei der Betrachtung des View-Update-Problems und des O/R-Mapping-Problems Verwendung finden, indem dortige Abbildungen als Baumabbildungen formuliert werden.

4 Reduktionen auf Baumautomaten

Mit einer Abbildungsbeschreibung in Form von Baumautomaten ist ein Abbildungsmechanismus vorhanden, bei dem jede formulierbare Abbildung ein Zielschema besitzt und die Eigenschaften einer Abbildung überprüft werden können. Somit eignet er sich zur Bestimmung von semantisch korrekten Operationen. Um die Abbildungen des View-Update-Problems und des O/R-Mapping-Problems anhand von Baumabbildungen zu untersuchen, müssen diese Abbildungen auf Baumautomaten mit Ausgabe reduziert werden. Dieses Kapitel untersucht die generelle Vorgehensweise bei einer Reduktion und zeigt auf, wie eine gegebene Abbildung f als eine durch einen Baumautomaten formulierte Abbildung kodiert wird.

Abschnitt 4.1 beschreibt den Ablauf einer Reduktion. Hierbei wird zuerst das Datenbankschema der Abbildung als Baumautomat ohne Ausgabe kodiert (Abschnitt 4.2). Der entstandene Automat dient dann als Quellschema für die Kodierung der Abbildung durch einen Automaten mit Ausgabe (Abschnitt 4.3). Es folgt eine Zusammenfassung des Kapitels in Abschnitt 4.4.

4.1 Durchführung einer Reduktion

Das Ziel einer Reduktion auf Baumautomaten ist, eine Abbildung so darzustellen, dass die durchführbaren Änderungsoperationen auf einem Anwendungsschema anhand der neuen Darstellungsform bestimmt werden können. Hierzu müssen die an einer Abbildung beteiligten Komponenten durch eine Funktion *enc* entsprechend umkodiert werden. Bei der Wahl der Kodierungsfunktion ist darauf zu achten, dass sich Erkenntnisse, die bei einer Untersuchung der kodierten Fassung einer Abbildung gewonnen werden, auch wieder auf die ursprüngliche Abbildung übertragen lassen.

Ist eine Abbildung als $f : dom(\mathbf{S}) \longrightarrow dom(\mathbf{T})$ zwischen den Instanzen zweier Schemata $\mathbf{S}$ und $\mathbf{T}$ gegeben, so wird eine Kodierungsfunktion *enc* benötigt, welche das Datenbankschema $\mathbf{S}$ in einen Baumautomaten $\mathcal{A}_S$ verwandelt. Dieser stellt den Definitionsbereich der Abbildung dar. Die Abbildung f wird danach als Baumtransformation basierend auf diesem Automaten als Quellschema kodiert. Um eine Beschreibung einer Abbildung als Baumabbildung zu erhalten, ergeben sich somit die folgenden Schritte:

1. **Definitionsbereich der Abbildung kodieren**
   Das Datenbankschema $\mathbf{S}$ der Abbildung f wird durch die Anwendung der Kodierungsfunktion $enc(\mathbf{S}) = \mathcal{A}_S$ als Automat $\mathcal{A}_S$ kodiert, wobei gewährleistet sein muss, dass wenn ein Wert $\mathbf{I}$ Instanz des Schemas $\mathbf{S}$ ist, dann der kodierte Wert

$enc(\mathbf{I})$ Instanz des Automaten $\mathcal{A}_S$ ist. Es muss also für alle Instanzen $\mathbf{I}$ des Schemas $\mathbf{S}$ gelten:

$$\mathbf{I} \in dom(\mathbf{S}) \Leftrightarrow enc(\mathbf{I}) \in dom(\mathcal{A}_S)$$

Der durch die Kodierung erzeugte Automat ist somit eine andere Darstellungsform für das Schema und Aussagen darüber, ob ein Wert Instanz ist oder nicht, können von der einen Form in die andere übertragen werden.

2. **Definition der Abbildung kodieren**
   Der generierte Automat $\mathcal{A}_S$ bildet die Grundlage für die Kodierung der Abbildung f, da er den Definitionsbereich der Abbildung vorgibt. Um eine Abbildung auf Basis dieses Definitionsbereichs aufzustellen, eignet sich grundsätzlich jeder Automat mit Ausgabe $\mathcal{T}$, der $\mathcal{A}_S$ als Quellschema besitzt, da ein solcher Automat genau die durch $\mathcal{A}_S$ beschriebenen Werte abbildet. Für die Kodierung der Abbildung f wird der Automat $\mathcal{T}$ so gewählt, dass

   $$f(\mathbf{I}) = \mathbf{J} \Leftrightarrow f_{\mathcal{T}}(enc(\mathbf{I})) = enc(\mathbf{J})$$

   gilt, d.h. wird eine Instanz $\mathbf{I}$ des Definitionsbereichs durch f auf eine Instanz $\mathbf{J}$ des Bildbereichs abgebildet, dann muss die kodierte Instanz $enc(\mathbf{I})$ durch den Automaten $\mathcal{T}$ auf die kodierte Instanz $enc(\mathbf{J})$ abgebildet werden. $f_{\mathcal{T}}$ ist folglich eine andere Darstellungsform für f, wobei sich die Eigenschaften von $f_{\mathcal{T}}$ auf f übertragen lassen.

3. **Änderungsoperationen kodieren**
   Sollen die Eigenschaften einer Änderungsoperation $\nu(f(\mathbf{I})) = \mathbf{J}$ auf dem Anwendungsschema untersucht werden, muss diese Operation als Wechsel zwischen zwei Bäumen kodiert werden, wobei

   $$\nu(f(\mathbf{I})) := \mathbf{J} \Leftrightarrow enc(\nu)(enc(f(\mathbf{I}))) = enc(\mathbf{J})$$

   sichergestellt sein muss. Ist die Änderung ν von $f(\mathbf{I})$ nach $\mathbf{J}$ definiert, bewirkt die kodierte Operation einen Wechsel zwischen den entsprechenden kodierten Werten.

   Da es zu jedem $\mathcal{T}$ einen inversen Automaten $\mathcal{T}^{-1}$ gibt, können die Eigenschaften der Operation ν untersucht werden. Wird $enc(\mathbf{J})$ vom inversen Automaten akzeptiert, dann gehört der Wert zum Zielschema der Abbildung $f_{\mathcal{T}}$ und ist somit seiteneffektfrei. Ist $\mathcal{T}^{-1}$ zudem kontextbezogen deterministisch, ist ν ebenfalls eindeutig und somit semantisch korrekt.

Das Anwendungsschema $\mathbf{T}$ einer Abbildung muss nicht kodiert werden, da dieses für die Bestimmung der semantisch korrekten Operationen nicht benötigt wird. Hierfür ist nur das Zielschema notwendig und dieses ergibt sich bei der Kodierung von f automatisch. Das Schema $\mathbf{T}$ legt allerdings die Menge der definierten Änderungsoperationen ν fest, so dass es indirekt über die Kodierung der Änderungsoperationen mit hineinspielt.

4.2 Definitionsbereich der Abbildung kodieren

Für den aus einer Kodierung generierten Automaten $\mathcal{A}_S$ ist sicherzustellen, dass sich dieser auch als Grundlage einer Abbildungskodierung eignet. Die Automaten mit Ausgabe $\mathcal{T}$, die ihn als Quellschema besitzen, müssen alle eine Abbildung auf Bäumen beschreiben, damit sie jeweils als mögliche Kodierung einer Abbildung f geeignet sind. Eine Abbildung wird etwa dann beschrieben, wenn alle Automaten $\mathcal{T}$ mit diesem Quellschema kontextbezogen deterministisch sind. Dann ergibt jede erlaubte Kodierung $f_{\mathcal{T}}$ einer Abbildung f auch eine Abbildung auf Bäumen und nicht nur eine Relation.

Da die Definition eines kontextbezogen deterministischen Automaten nicht auf den Ausgabewerten eines Automaten mit Ausgabe beruht, sondern fordert, dass für einen Eingabewert alle Ableitungswege gleich sind, kann diese Eigenschaft auch für Automaten ohne Ausgabe definiert werden. Ist ein generierter Automat $\mathcal{A}_S$ kontextbezogen deterministisch, ist automatisch jeder Automat mit Ausgabe $\mathcal{T}$, der $\mathcal{A}_S$ als Quellschema besitzt, ebenfalls kontextbezogen deterministisch und beschreibt somit eine Abbildung.

Definition 4.1 (*Kontextbezogen deterministisch*)
Ein Automat $\mathcal{A}_S = (Q, \Sigma, q_f, \Delta)$ ohne Ausgabe ist *kontextbezogen deterministisch*, wenn für jeden Wert $v \in dom(\mathcal{A}_S)$ alle Ableitungswege $v \xrightarrow{*}_{\mathcal{A}_S} q_f$ gleich sind.

Das Verfahren zur Überprüfung dieser Eigenschaft kann von den Automaten mit Ausgabe aus Abschnitt 3.1.3.2 übernommen werden, da die Ausgabewerte der Regeln bei einer Überprüfung keine Rolle spielen.

Die Forderung nach einem kontextbezogen deterministischen Automaten ohne Ausgabe geht über die Rolle eines solchen Automaten als reinem Akzeptor hinaus. Wird ein Automat als Akzeptor verwendet, muss es für einen Wert nur einen beliebigen Ableitungsweg zum Finalzustand gegeben, damit der Wert akzeptiert wird. Es können also auch mehrere unterschiedliche Wege zu einem Wert vorhanden sein und der Automat wird immer noch seiner Aufgabe gerecht. Im hier betrachteten Umfeld, in dem ein Automat die Grundlage einer Abbildungsdefinition bildet, macht es aber Sinn, strengere Kriterien bei den Automateneigenschaften anzulegen, um sicherzustellen, dass letztendlich eine Abbildung definiert wird.

Für ein Kodierungsverfahren, welches aus gegebenen Schemata Baumautomaten ohne Ausgabe erzeugt, ist folglich sicherzustellen, dass dieses so gewählt wird, dass ein generierter Automat immer kontextbezogen deterministisch ist. Um eine Abbildung zu erhalten, müssen dann nur bei seinen Regeln die Ausgabewerte ergänzt werden.

4.3 Definition der Abbildung kodieren

Der durch eine Kodierung eines Schemas entstandene Automat $\mathcal{A}_S$ dient als Grundlage, um eine Abbildung zu definieren, indem er als Quellschema eines Automaten mit Ausgabe $\mathcal{T}$ verwendet wird. Ist ein Automat $\mathcal{A}_S = (Q, \Sigma, q_f, \Delta_S)$ gegeben, so beschreibt

jeder Baumautomat $\mathcal{T} = (Q, \Sigma, \Sigma', q_f, \Delta)$ eine Abbildung basierend auf $\mathcal{A}_S$ als Definitionsbereich, wenn für die Regelmengen gilt:

$$\begin{array}{lcl} w\ [\ y_1\ /\ q_1, \ldots, y_n\ /\ q_n\ ] & \longrightarrow & q \in \Delta_S \\ & \text{gdw} & \\ w\ [\ y_1\ /\ q_1(x_1), \ldots, y_n\ /\ q_n(x_n)\ ] & \longrightarrow & q\ (\ w'\ [\ y_1\ /\ x_1, \ldots, y_n\ /\ x_n\ ]\ ) \in \Delta \end{array}$$

Bildlich gesprochen, werden die Regeln des Automaten $\mathcal{A}_S$ so um Ausgabewerte erweitert, dass sich die Regelmenge eines Automaten mit Ausgabe $\mathcal{T}$ ergibt.

Die Wahl einer Ausgabe kann von unterschiedlichen Kriterien abhängen. Ist ein Wert der Form

$$w\ [\ x\ /\ \{\ \ldots, a : v, \ldots\}\ ]$$

gegeben, dann enthält dieser Wert eine Ebene mit einer Annotation a, der ein Teilwert v folgt, und die Ebene von a ist in einen umgebenden Wert w eingebettet. Für die im Wert vorkommende Annotation a kann die Abbildung anhand von drei Kriterien festgemacht werden:

- Die Ausgabe für a basiert auf der Annotation selbst, d.h. etwa, dass ein a an dieser Stelle bei jedem Wert immer auf ein b abgebildet wird.
- Die Ausgabe für a kann vom nachfolgenden Teilwert v abhängen, d.h. etwa, dass wenn bei einem Wert ein Teilwert v_1 folgt, wird a auf b abgebildet, und wenn bei einem Wert ein anderer Teilwert v_2 folgt, erfolgt die Abbildung von a nach c.
- Die Ausgabe für a hängt vom Kontext w ab, der a umgibt, d.h. etwa, dass wenn bei einem Wert in der weiteren Ableitung ein Kontext w_1 erreicht wird, erfolgt die Abbildung von a nach b und sonst nach c.

Eine Abbildung wird auf Basis der Regelmenge eines Automaten formuliert, so dass Abbildungen, die diese drei Kriterien verwenden, um eine Ausgabe festzulegen, auch anhand der Regelmenge definierbar sein müssen. Ist eine Regel

$$\{\ \ldots, a : q, \ldots\}\ \longrightarrow q'$$

vorhanden, kann eine Abbildung basierend auf dem Vorhandensein einer Annotation formuliert werden, indem einfach die entsprechende Ausgabe angegeben wird. Die Regel

$$\{\ \ldots, a : q(x), \ldots\}\ \longrightarrow q'\ (\ \{\ \ldots, b : x, \ldots\}\ )$$

bildet die Annotation a auf die Annotation b ab.

Eine Abbildungsdefinition in Abhängigkeit des der Annotation a nachfolgenden Teilwerts v kann anhand einer Regel

$$\{\ \ldots, a : q, \ldots\}\ \longrightarrow q'$$

nicht formuliert werden, da nach q alle Teilwerte, die der Annotation a folgenden dürfen, abgeleitet werden und keine Unterscheidung nach v_1 und v_2 möglich ist. Um eine Abbildung dennoch möglich zu machen, ist die Idee, einen Automaten in einen äquivalenten Automaten umzuformen, der statt der einen Regel $\{ \ldots, a : q, \ldots \} \longrightarrow q'$ zwei Regeln $\{ \ldots, a : q_1, \ldots \} \longrightarrow q'$ und $\{ \ldots, a : q_2, \ldots \} \longrightarrow q'$ enthält, wobei $dom(q_1) \cup dom(q_2) = dom(q)$ gilt. Wenn hierbei $v_1 \in dom(q_1)$ und $v_2 \in dom(q_2)$ gilt, kann anhand der beiden Regeln im entstandenen Automaten die Abbildung in Abhängigkeit von v_1 und v_2 gewählt werden. Hier beschreibt der Ausgabewert der ersten Regel die Ausgabe für a, wenn ein v_1 folgt, und der Ausgabewert der zweiten Regel die Ausgabe für a, wenn ein v_2 folgt. Die Unterscheidung wird anhand der Zustände q_1 und q_2 festgemacht.

Eine Abbildungsdefinition anhand des umgebenden Kontexts ist bei einer Regel

$$\{ \ldots, a : q, \ldots \} \longrightarrow q'$$

ebenfalls nicht formulierbar, da q' alle erlaubten Verwendungspositionen der Ebene mit a kennzeichnet und hier keine Unterscheidung zwischen bestimmten Positionen trifft. Um eine Abbildung in Abhängigkeit des Kontexts zu formulieren, wird auch hier der Automat in einen äquivalenten Automaten umgeformt, der statt der Regel $\{ \ldots, a : q, \ldots \} \longrightarrow q'$ zwei Regeln $\{ \ldots, a : q, \ldots \} \longrightarrow q_1$ und $\{ \ldots, a : q, \ldots \} \longrightarrow q_2$ mit $dom(q_1) = dom(q_2) = dom(q)$ enthält, wobei q_1 und q_2 jeweils für unterschiedliche Verwendungspositionen des ursprünglichen q stehen. Der Ausgabewert der ersten Regel beschreibt dann die Ausgabe, wenn ein q_1 in einem bestimmten Wert w_1 in der weiteren Ableitung erreicht wird, und die Ausgabe der zweiten Regel die Ausgabe, wenn ein q_2 in einem anderer Wert w_2 erreicht wird.

Da für eine gegebene Wertemenge im Regelfall nicht nur ein Automat vorhanden ist, der diese akzeptiert, sondern zumeist mehrere äquivalente Automaten existieren, kann ein Automat in einen äquivalenten Automaten umgeformt werden. Eine *Umformung* erzeugt einen neuen Automaten, indem eine Menge von Regeln im Ausgangsautomat durch eine andere Regelmenge im Ergebnisautomat ersetzt wird, wobei der Ergebnisautomat zum Ausgangsautomat äquivalent ist und, wenn der Ausgangsautomat kontextbezogen deterministisch ist, es dann der Ergebnisautomat ebenfalls ist. Es gehen also bei einer Umformung keine der wichtigen Automateneigenschaften verloren. Beim *Aufteilen eines Wertebereichs* eines Zustands q wird ein an einer bestimmten Position einer Regel vorkommendes q durch zwei neue Zustände q_1 und q_2 mit $dom(q) = dom(q_1) \cup dom(q_2)$ ersetzt und entsprechend zwei neue Regeln angelegt. Beim *Duplizieren eines Wertebereichs* von q werden die Vorkommen von q an zwei unterschiedlichen Positionen in den Regeln durch je einen neuen Zustand q_1 bzw. q_2 mit $dom(q) = dom(q_1) = dom(q_2)$ ersetzt.

Theorem 4.1 (*Äquivalenzumformungen*)
Ist ein Automat gegeben, dann sind das Aufteilen und das Duplizieren von Wertebereichen Umformungen, welche zu einem äquivalenten Automaten führen. Ist der Aus-

gangsautomat kontextbezogen deterministisch, bleibt diese Eigenschaft bei beiden Umformungen erhalten. □

Der Beweis zu Theorem 4.1 definiert zunächst eine Reihe von einfachen Umformungen, welche hintereinander ausgeführt, die Behauptung zeigen.

Einsetzen und Aufteilen von Regeln

Eine Umformung kann durchgeführt werden, indem Regeln in einander eingesetzt werden. Sind bei einem Automaten Regeln der Form

$$\begin{array}{lcl} \{ \ldots, a : q', \ldots \} & \longrightarrow & q \\ u_1 & \longrightarrow & q' \\ & \ldots & \\ u_n & \longrightarrow & q' \end{array}$$

vorhanden, kann ein äquivalenter Automat erzeugt werden, der statt diesen Regeln die Regelmenge

$$\begin{array}{lcl} \{ \ldots, a : u_1, \ldots \} & \longrightarrow & q \\ & \ldots & \\ \{ \ldots, a : u_n, \ldots \} & \longrightarrow & q \\ u_1 & \longrightarrow & q' \\ & \ldots & \\ u_n & \longrightarrow & q' \end{array}$$

enthält. Die linken Seiten u_i der Regeln $u_i \longrightarrow q'$ werden jeweils an Stelle von q' in die erste Regel $\{ \ldots, a : q', \ldots \} \longrightarrow q$ eingesetzt, wodurch sich die neue Regelmenge ergibt.

Theorem 4.2 (*Einsetzen von Regeln*)
Seien ein Automat $\mathcal{A}_1 = (Q_1, \Sigma, q_f, \Delta_1)$, eine Regel $w[x \,/\, q'] \longrightarrow q \in \Delta_1$ mit $q' \in Q_1$, $q' \neq q_f$ und $\Delta_{q'} \subseteq \Delta_1$ als die Menge aller Regeln, die q' auf der rechten Seite enthalten, gegeben, dann entsteht durch das *Einsetzen der Regeln* aus $\Delta_{q'}$ in $w[x \,/\, q'] \longrightarrow q$ ein Automat $\mathcal{A}_2 = (Q_1, \Sigma, q_f, \Delta_2)$ mit

$$\begin{array}{rcl} \Delta_2 := \Delta_1 & \cup & \{w[x \,/\, u_i] \longrightarrow q \mid \forall u_i \longrightarrow q' \in \Delta_{q'} \wedge w[x \,/\, q'] \longrightarrow q \in \Delta_1\} \\ & - & \{w[x \,/\, q'] \longrightarrow q\} \end{array}$$

Es gilt $dom(\mathcal{A}_1) = dom(\mathcal{A}_2)$. Ist $\mathcal{A}_1$ kontextbezogen deterministisch, ist $\mathcal{A}_2$ kontextbezogen deterministisch. □

Beweis Ist für einen Wert $w\ [\ x\ \ /\ u_i\ ]$ in $\mathcal{A}_1$ ein Ableitungsweg mit zwei Regeln $u_i \longrightarrow q'$ und $w\ [\ x\ \ /\ q'\ ] \longrightarrow q$ vorhanden, so gibt es in $\mathcal{A}_2$ einen Weg mit nur einem Übergang $w\ [\ x\ \ /\ u_i\ ] \longrightarrow q$ und umgekehrt. Es werden durch ein Einsetzen

zwei Übergänge zu einem zusammengefasst. Da es sich bei q' nicht um den Finalzustand handelt, sind die Mengen der Instanzen von $\mathcal{A}_1$ und $\mathcal{A}_2$ identisch.

Da $\mathcal{A}_1$ kontextbezogen deterministisch ist, gibt es nur gleiche Ableitungswege für einen Wert $w\ [\ x\ /\ u_i\ ]$ nach q. In $\mathcal{A}_2$ gibt es für einen solchen Wert ebenfalls nur gleiche Wege nach q, da zwar durch eine Regel $w\ [\ x\ /\ u_i\ ] \longrightarrow q$ neue Wege mit diesem Übergang hinzukommen, aber gleichzeitig durch den Wegfall der Regel $w\ [\ x\ /\ q'\ ] \longrightarrow q$ die in $\mathcal{A}_1$ vorkommenden Wege entfernt werden. $\mathcal{A}_2$ ist somit ebenfalls kontextbezogen deterministisch. □

Beim Einsetzen von Regeln werden Regeln in einander geschachtelt. Eine umgekehrte Umformung gestattet es, geschachtelte Regeln wieder auseinander zunehmen.

Theorem 4.3 (*Aufteilen von Regeln*)
Seien ein Automat $\mathcal{A}_1 = (Q_1, \Sigma, q_f, \Delta_1)$, eine Menge von Regeln $w\ [\ x\ /\ u_1\ ] \longrightarrow q, \ldots, w\ [\ x\ /\ u_n\ ] \longrightarrow q \in \Delta$ und $q' \notin Q_1$ gegeben, dann entsteht durch das *Aufteilen der Regeln* ein Automat $\mathcal{A}_2 = (Q_1 \cup \{q'\}, \Sigma, q_f, \Delta_2)$ mit

$$\begin{aligned}\Delta_2 := \Delta_1 \quad \cup \quad & \{w[x\ /\ q'] \longrightarrow q, u_1 \longrightarrow q', \ldots, u_n \longrightarrow q' \mid \\ & \qquad w[x\ /\ u_1] \longrightarrow q, \ldots, w[x\ /\ u_n] \longrightarrow q \in \Delta_1\} \\ - \quad & \{w[x\ /\ u_1] \longrightarrow q, \ldots, w[x\ /\ u_n] \longrightarrow q\}\end{aligned}$$

Es gilt $dom(\mathcal{A}_1) = dom(\mathcal{A}_2)$. Ist $\mathcal{A}_1$ kontextbezogen deterministisch, ist $\mathcal{A}_2$ kontextbezogen deterministisch. □

Beweis Dadurch, dass ein neuer Zustand q' eingeführt wird, wird ein Teilwert $w\ [\ x\ /\ u_i\ ]$, der in $\mathcal{A}_1$ durch eine Übergangsregel akzeptiert wird, in $\mathcal{A}_2$ durch zwei Regeln $w\ [\ x\ /\ q'\ ] \longrightarrow q$ und $u_i \longrightarrow q'$ akzeptiert und umgekehrt. Die beiden Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ sind äquivalent.

Es gibt für einen Wert $w\ [\ x\ /\ u_i\ ]$ in $\mathcal{A}_1$ nur gleiche Ableitungswege nach q. In $\mathcal{A}_2$ kommen neue Wege durch das Hinzufügen der Regeln $w\ [\ x\ /\ q'\ ] \longrightarrow q$ und $u_i \longrightarrow q'$ hinzu, aber die alten Wege fallen weg, da eine Regel wie $w\ [\ x\ /\ u_i\ ] \longrightarrow q$ nicht übernommen wird. $\mathcal{A}_2$ ist somit kontextbezogen deterministisch. □

Austauschen von Zuständen

Neben dem Einsetzen und Aufteilen von Regeln kann die Regelmenge auch so umgeformt werden, dass Zustände mit demselben Wertebereich alternativ verwendet werden können. Ist die Regelmenge

$$
\begin{array}{lcl}
\{ \ldots, a : q_1, \ldots \} & \longrightarrow & q \\
u_1 & \longrightarrow & q_1 \\
 & \ldots & \\
u_n & \longrightarrow & q_1 \\
u_1 & \longrightarrow & q_2 \\
 & \ldots & \\
u_n & \longrightarrow & q_2
\end{array}
$$

vorhanden und gilt $dom(q_1) = dom(q_2)$, dann kann ein äquivalenter Automat erzeugt werden, der stattdessen die Regelmenge

$$
\begin{array}{lcl}
\{ \ldots, a : q_2, \ldots \} & \longrightarrow & q \\
u_1 & \longrightarrow & q_1 \\
 & \ldots & \\
u_n & \longrightarrow & q_1 \\
u_1 & \longrightarrow & q_2 \\
 & \ldots & \\
u_n & \longrightarrow & q_2
\end{array}
$$

enthält. Der Zustand q_1 wird in der ersten Regel durch den Zustand q_2 ersetzt.

Theorem 4.4 (*Austauschen von Zuständen*)
Seien ein Automat $\mathcal{A}_1 = (Q_1, \Sigma, q_f, \Delta_1)$, eine Regel $w\ [\ x\ /\ q_1\ ] \longrightarrow q \in \Delta_1$ und zwei Zustände $q_1, q_2 \in Q_1$ mit $dom(q_1) = dom(q_2)$ gegeben, so ergibt sich durch das *Austauschen von Zuständen* q_1 mit q_2 ein Automat $\mathcal{A}_2 = (Q_1, \Sigma, q_f, \Delta_2)$ mit

$$
\begin{array}{rcl}
\Delta_2 := \Delta_1 & \cup & \{w[x\ /\ q_2] \longrightarrow q\} \\
 & - & \{w[x\ /\ q_1] \longrightarrow q\}
\end{array}
$$

Es gilt $dom(\mathcal{A}_1) = dom(\mathcal{A}_2)$. Ist $\mathcal{A}_1$ kontextbezogen deterministisch, ist $\mathcal{A}_2$ kontextbezogen deterministisch. □

Beweis Wird ein Wert $w\ [\ x\ /\ v\ ]$ mit $v \in dom(q_1)$ von $\mathcal{A}_1$ akzeptiert, wird derselbe Wert $w\ [\ x\ /\ v\ ]$ mit $v \in dom(q_2)$ von $\mathcal{A}_2$ akzeptiert und umgekehrt, da $dom(q_1)$ und $dom(q_2)$ identisch sind. Die beiden Automaten sind äquivalent.

Es gibt für einen Wert $w\ [\ x\ /\ v\ ]$ mit $v \in dom(q_1) = dom(q_2)$ in $\mathcal{A}_1$ nur gleiche Ableitungswege nach q. In $\mathcal{A}_2$ kommen neue Wege durch das Hinzufügen der Regel $u\ [\ x\ /\ q_2\ ] \longrightarrow q$ dazu und die alten Wege über q_1 fallen weg. Folglich ist $\mathcal{A}_2$ kontextbezogen deterministisch. □

Durch das Austauschen von Zuständen kann ein Zustand q_1 aus dem Automaten entfernt werden, wenn es einen anderen Zustand q_2 gibt, welcher denselben Wertebereich beschreibt. Der zu entfernende Zustand q_1 wird an allen Verwendungspositionen durch q_2 ersetzt, so dass dann die Übergangsregeln, welche q_1 auf der rechten Seite enthalten, überflüssig werden, da q_1 auf keiner linken Seite einer Regel mehr verwendet wird. Die Zustände q_1 und q_2 werden somit zusammengefasst.

Aufteilen von Wertebereichen

Mit Hilfe der drei vorgestellten Umformungen ist es möglich, Wertebereiche von Zuständen aufzuteilen. Ist eine Regelmenge

$$\begin{array}{lcl} \{\ \ldots, a : q, \ldots\} & \longrightarrow & q' \\ u_1 & \longrightarrow & q \\ & \ldots & \\ u_n & \longrightarrow & q \end{array}$$

vorhanden und soll der Wertebereich von q an der Position hinter a aufgeteilt werden, werden die Regeln aus Δ_q zuerst in die erste Regel eingesetzt. Es entsteht die Regelmenge:

$$\begin{array}{lcl} \{\ \ldots, a : u_1, \ldots\} & \longrightarrow & q' \\ & \ldots & \\ \{\ \ldots, a : u_n, \ldots\} & \longrightarrow & q' \\ u_1 & \longrightarrow & q \\ & \ldots & \\ u_n & \longrightarrow & q \end{array}$$

Danach werden die Regeln gemäß der gewünschten Einteilung der Domäne aufgeteilt. Sollen u_1 bis u_i zu einer Domäne und u_{i+1} bis u_n zur anderen gehören, werden zwei neue Zustände q_1 und q_2 eingeführt und die Regeln aufgeteilt:

$$\begin{array}{lcl} \{\ \ldots, a : q_1, \ldots\} & \longrightarrow & q' \\ \{\ \ldots, a : q_2, \ldots\} & \longrightarrow & q' \\ u_1 & \longrightarrow & q_1 \\ & \ldots & \\ u_i & \longrightarrow & q_1 \\ u_{i+1} & \longrightarrow & q_2 \\ & \ldots & \\ u_n & \longrightarrow & q_2 \\ u_1 & \longrightarrow & q \\ & \ldots & \\ u_n & \longrightarrow & q \end{array}$$

Der Wertebereich von q wird also auf die beiden Zustände q_1 und q_2 aufgeteilt. q_1 kennzeichnet die Werte der ersten Teilmenge des ursprünglichen Wertebereichs von q und q_2 die Werte der zweiten Teilmenge. Der entstehende Automat ist äquivalent zum Ausgangsautomat und kontextbezogen deterministisch, wenn dieser es ist, da Umformungen verwendet werden, welche dies garantieren.

Das Aufteilen von Wertebereichen kann an einer bestimmten Verwendungsposition eines Zustands q in einer Regel erfolgen, so dass q in anderen Regeln weiterverwendet wird, oder der Zustand q kann komplett durch neue Zustände ersetzt werden, d.h. an jeder Verwendungsposition von q findet eine Aufteilung statt. Es ergibt sich dann eine

Reihe von neuen Zuständen q_i, welche je eine Teilmenge der Domäne von q beschreiben. Zustände, welche dieselbe Teilmenge beschreiben, können durch Austauschen und Entfernen der dann nicht mehr benötigten Regeln zusammengefasst werden.

Duplizieren von Wertebereichen

Die vorgestellten Umformungen erlauben auch ein Duplizieren von Wertebereichen. Sind die Regeln

$$\begin{array}{ll} \{\ldots, a : q, \ldots, b : q, \ldots\} & \longrightarrow\ q' \\ u_1 & \longrightarrow\ q \\ & \ldots \\ u_n & \longrightarrow\ q \end{array}$$

gegeben, werden die Regeln Δ_q an beiden Positionen von q in die erste Regel $\{\ldots, a : q, \ldots, b : q, \ldots\} \longrightarrow q'$ eingesetzt, wonach sich eine Regelmenge der Form

$$\begin{array}{ll} \{\ldots, a : u_1, \ldots, b : u_1, \ldots\} & \longrightarrow\ q' \\ \{\ldots, a : u_1, \ldots, b : u_2, \ldots\} & \longrightarrow\ q' \\ & \ldots \\ \{\ldots, a : u_n, \ldots, b : u_{n-1}, \ldots\} & \longrightarrow\ q' \\ \{\ldots, a : u_n, \ldots, b : u_n, \ldots\} & \longrightarrow\ q' \\ u_1 & \longrightarrow\ q \\ & \ldots \\ u_n & \longrightarrow\ q \end{array}$$

ergibt. Diese Regeln werden entsprechend der Werte u_i hinter a sortiert und für jede dieser Gruppen die Regeln aufgeteilt, indem jeweils ein neuer Zustand q_i eingefügt wird, so dass die Regelmenge

$$\begin{array}{ll} \{\ldots, a : u_1, \ldots, b : q_1, \ldots\} & \longrightarrow\ q' \\ u_1 & \longrightarrow\ q_1 \\ & \ldots \\ u_n & \longrightarrow\ q_1 \\ \{\ldots, a : u_2, \ldots, b : q_2, \ldots\} & \longrightarrow\ q' \\ u_1 & \longrightarrow\ q_2 \\ & \ldots \\ u_n & \longrightarrow\ q_2 \\ & \ldots \\ \{\ldots, a : u_n, \ldots, b : q_n, \ldots\} & \longrightarrow\ q' \\ u_1 & \longrightarrow\ q_n \\ & \ldots \\ u_n & \longrightarrow\ q_n \\ & \ldots \\ u_1 & \longrightarrow\ q \\ & \ldots \\ u_n & \longrightarrow\ q \end{array}$$

entsteht. Die neu eingefügten Zustände q_i enthalten alle denselben Wertebereich, so dass diese durch Austauschen zu einem einzigen Zustand q_1 zusammengefasst werden können:

$$\begin{array}{lcl}
\{\ \ldots, a : u_1, \ldots, b : q_1, \ldots\} & \longrightarrow & q' \\
\{\ \ldots, a : u_2, \ldots, b : q_1, \ldots\} & \longrightarrow & q' \\
 & \ldots & \\
\{\ \ldots, a : u_n, \ldots, b : q_1, \ldots\} & \longrightarrow & q' \\
u_1 & \longrightarrow & q_1 \\
 & \ldots & \\
u_n & \longrightarrow & q_1 \\
u_1 & \longrightarrow & q \\
 & \ldots & \\
u_n & \longrightarrow & q
\end{array}$$

Die Regeln für q_2 bis q_n können wegfallen, da es sich hierbei nicht um Endzustände handelt und diese Zustände auf keinen linken Seiten mehr verwendet werden.

Danach wird wieder ein Aufteilen der Regeln vorgenommen und die Zustände zusammengefasst. Es entsteht die Regelmenge

$$\begin{array}{lcl}
\{\ \ldots, a : q_2, \ldots, b : q_1, \ldots\} & \longrightarrow & q' \\
u_1 & \longrightarrow & q_1 \\
 & \ldots & \\
u_n & \longrightarrow & q_1 \\
 & \ldots & \\
u_1 & \longrightarrow & q_2 \\
 & \ldots & \\
u_n & \longrightarrow & q_2 \\
u_1 & \longrightarrow & q \\
 & \ldots & \\
u_n & \longrightarrow & q
\end{array}$$

Der Wertebereich von q wird durch die beiden Zustände q_1 und q_2 dupliziert. q_1 kennzeichnet die Werte, die hinter der Annotation b stehen, und q_2 die Werte, die der Annotation a folgen. Der entstehende Automat ist äquivalent zum Ausgangsautomat und kontextbezogen deterministisch, da der Ausgangsautomat es ist. Dies stellen die verwendeten Umformungen sicher.

Das Duplizieren von Wertebereichen kann an zwei bestimmten Verwendungspositionen eines Zustands q innerhalb der Regeln erfolgen, so dass q in anderen Regeln weiterverwendet wird, oder der Zustand q kann komplett durch neue Zustände ersetzt werden, d.h. an jeder Verwendungsposition von q findet eine Duplizieren statt. Da alle neuen Zustände denselben Wertebereich beschreiben, können einige von ihnen bei Bedarf durch Austauschen und Entfernen der dann überflüssig gewordenen Regeln zusammengefasst

werden.

Beispiel 4.1 (*Umformungen*)
Ausgangspunkt ist ein Automat $\mathcal{A} = (\{q_f, q_1\}, \{a, b, c, d\}, q_f, \Delta)$ mit den Regeln $\Delta = \{$

$$\begin{array}{lcl} \{\ a : q_1,\ b : q_1\ \} & \longrightarrow & q_f \\ \{\ c\ \} & \longrightarrow & q_1 \\ \{\ d\ \} & \longrightarrow & q_1 \end{array}$$

$\}$. Dieser Automat ist kontextbezogen deterministisch.

1. Ist $\mathcal{A}$ gegeben, so wird ein äquivalenter Automat $\mathcal{A}_{sep} = (\{q_f, q_2, q_3\}, \{a, b\}, q_f, \Delta_{sep})$ durch das Aufteilen des Wertebereichs von q_1 auf zwei neue Zustände q_2 und q_3 erzeugt, wobei $dom(q_2) \cup dom(q_3) = dom(q_1)$. Die Regelmenge ergibt sich zu $\Delta_{sep} = \{$

   $$\begin{array}{lcl} \{\ a : q_2,\ b : q_2\ \} & \longrightarrow & q_f \\ \{\ a : q_2,\ b : q_3\ \} & \longrightarrow & q_f \\ \{\ a : q_3,\ b : q_2\ \} & \longrightarrow & q_f \\ \{\ a : q_3,\ b : q_3\ \} & \longrightarrow & q_f \\ \{\ c\ \} & \longrightarrow & q_2 \\ \{\ d\ \} & \longrightarrow & q_3 \end{array}$$

   $\}$. An jeder Stelle in der ersten Regel von $\mathcal{A}_1$, an der q_1 auftritt, kommt in $\mathcal{A}_2$ einmal der Zustand q_2 und einmal der Zustand q_3 vor. Da es zwei Positionen gibt und zwei neue Zustände eingeführt werden, entstehen vier Regeln. q_2 kennzeichnet die Domäne mit dem Wert c und q_3 die Domäne mit den Wert d. Wird auf Basis von $\mathcal{A}_{sep}$ eine Ausgabe ergänzt, können die Abbildungen von a und b in den ersten vier Regeln in Abhängigkeit von q_2 und q_3 erfolgen, d.h. abhängig davon, ob der Wert c oder d darunter folgt. Die Regeln mit q_1 auf der rechten Seite fallen heraus, da dieser Zustand nicht der Finalzustand ist und q_1 auf keiner linken Seite mehr verwendet wird.

2. Eine Umformung von $\mathcal{A}$ in einen Automaten $\mathcal{A}_{dup} = (\{q_f, q_2, q_3\}, \{a, b\}, q_f, \Delta_{dup})$ ist durch ein Duplizieren des Wertebereichs von q_1 durch das Einführen von q_2 und q_3 möglich. Die Regeln ergeben sich zu $\Delta_{dup} = \{$

   $$\begin{array}{lcl} \{\ a : q_2,\ b : q_3\ \} & \longrightarrow & q_f \\ \{\ c\ \} & \longrightarrow & q_2 \\ \{\ d\ \} & \longrightarrow & q_2 \\ \{\ c\ \} & \longrightarrow & q_3 \\ \{\ d\ \} & \longrightarrow & q_3 \end{array}$$

   $\}$. q_2 kennzeichnet die Verwendungsposition von c und d hinter a und q_3 die Position hinter b. Hier ist eine Abbildungsdefinition bei den letzten vier Regeln abhängig

von q_2 und q_3 möglich, d.h. abhängig davon, ob im weiteren Ableitungsweg ein a oder ein b erreicht wird. Die Regeln mit q_1 sind hier überflüssig und werden weggelassen.

□

Mit Hilfe der Umformungen können Abbildungen in Abhängigkeit eines Annotationssymbols, in Abhängigkeit eines bereits abgeleiteten Werts oder in Abhängigkeit der weiteren Ableitung definiert werden. Im Reduktionsverfahren gliedern sich die Umformungen zwischen der Kodierung des Definitionsbereichs der Abbildung und der Kodierung der Abbildung selbst in den Ablauf ein.

4.4 Fazit

Bei der Reduktion einer Abbildung f auf eine Baumtransformationen ergeben sich die folgenden Schritte. Zuerst wird das Datenbankschema der Abbildung als Automat ohne Ausgabe kodiert, wobei das Kodierungsverfahren einen kontextbezogen deterministischen Automaten erzeugt. Werden dessen Regeln nach einer optionalen Umformung um Ausgabewerte ergänzt, entsteht ein Automat mit Ausgabe, der die Abbildung kodiert. Werden ebenfalls die Änderungsoperationen auf dem Anwendungsschema kodiert, können deren Eigenschaften untersucht werden.

5 Sichten im relationalen Datenmodell

Innerhalb des relationalen Datenmodells sind Abbildungen in Form von Sichten schon seit der Einführung dieses Datenmodells Gegenstand von Untersuchungen. Ein besonderes Augenmerk liegt auf dem View-Update-Problem, bei dem Änderungsoperationen, die auf einem Anwendungsschema spezifiziert sind, in passende Operationen auf dem Datenbankschema umzusetzen sind. Ziel ist es hierbei, für eine möglichst große Klasse von Sichten und unterschiedlichen Operationen auf dem Sichtschema eine gültige Umsetzung bestimmen zu können.

Als ein erster Anwendungsfall sollen in diesem Kapitel Sichten des View-Update-Problems auf Abbildungen zwischen Bäumen reduziert und hierdurch die semantisch korrekten Änderungsoperationen auf einer Sicht bestimmt werden. Durch die Kodierung einer Sicht als Baumautomat wird erreicht, dass bei jeder Sicht, die mit Hilfe von Selektionen, Projektionen oder kartesischen Produkten formuliert wird, die semantisch korrekten Operationen bestimmt werden können. Es zeigt sich, dass die Kodierung jeder dieser Abbildungen eine informationserhaltende Sicht beschreibt, so dass alle Änderungsoperationen auf einer Sicht entweder eindeutig durchführbar sind oder eine Durchführung abgelehnt wird, wenn die Operation nicht seiteneffektfrei ist.

Abschnitt 5.1 führt die im Weiteren verwendete Notation für Relationstypen und Relationen ein, woran sich eine Formulierung des View-Update-Problems in Abschnitt 5.2 anschließt. Abschnitt 5.3 befasst sich mit bisherigen Lösungsvorschlägen zum View-Update-Problem und zeigt deren Einschränkungen auf. Die notwendige Kodierungsfunktion *enc* für eine Reduktion einer Sicht auf einen Baumautomaten wird in Abschnitt 5.4 definiert, wobei gezeigt wird, dass eine Sicht eine informationserhaltende Abbildung beschreibt. Die Vorgehensweise, um bei einer Sicht die semantisch korrekten Operationen zu bestimmen, zeigt Abschnitt 5.5. Dieses Kapitel schließt mit einem Beispiel zum View-Update-Problem in Abschnitt 5.6 und einem Fazit in 5.7.

5.1 Das relationale Datenmodell

Da bei einer Reduktion des View-Update-Problems relationale Schemata und deren Instanzen in einen Baumautomaten zu überführen sind, gilt es zunächst eine formale Beschreibung des relationalen Modells anzugeben, auf der die Definition der Kodierungsfunktion *enc* basiert.

Das relationale Datenmodell wurde von [Cod70] eingeführt und beschreibt Daten als Mengen von Tupeln, wobei innerhalb eines Tupels die Zustandsdaten einer bestimmten

Entität der Realwelt zusammengefasst werden. Eine Menge von Tupeln mit demselben strukturellen Aufbau bildet eine Relation. Obwohl die semantische Ausdrucksmächtigkeit des Datenmodells nicht sehr ausgeprägt ist, beruhen aufgrund seiner Einfachheit und seiner formalen Beschreibbarkeit die meisten heute im Einsatz befindlichen Datenbanksysteme auf diesem Modell. Die folgende Beschreibung des relationalen Datenmodells und die verwendete Notation ist an [AHV95] angelehnt.

5.1.1 Werte

Werte innerhalb des Modells werden durch Tupel und darauf aufbauenden Relationen beschrieben. Die Menge **const** der atomaren Werte oder *Konstanten* wird durch die Vereinigung einer Menge von paarweise disjunkten, abzählbar unendlichen *Domänen* gebildet. Beispiele für Domänen sind die Menge der Zahlenwerte oder die Menge der Zeichenfolgen, welche einzelne Zahlen bzw. Texte als Konstanten enthalten.

Ein *Tupel* ist ein Element eines kartesischen Produkts von Domänen. Ein Tupel t ist ein geordnetes n-Tupel ($n \geq 1$) von Konstanten, d.h. $t \in dom_1 \times \ldots \times dom_n$, wobei die einzelnen dom_i nicht paarweise disjunkt sein müssen. Ein Tupel wird in spitzen Klammern geschrieben $t = \langle v_1, v_2, \ldots, v_n \rangle$. Auf die einzelnen Elemente eines Tupels t kann durch $t(i)$ zugegriffen werden, wobei $t(i)$ das $i$te Element v_i adressiert und $1 \leq i \leq n$. Eine *Relation* ist eine möglicherweise leere, endliche Menge R von gleich aufgebauten Tupeln t mit $t \in dom_1 \times \ldots \times dom_n$.

Die drei Relationen aus Abschnitt 2.1 sind also Relationen im Sinne dieser Definition und werden noch einmal in Abb. 5.1 dargestellt.

5.1.2 Schema

Ein Schema besteht aus einer Menge von Relationstypen, wobei jeder Typ eine Menge von erlaubten Relationen beschreibt. Es existiert eine abzählbar unendliche Menge **att** von *Attributbezeichnern*. Eine Abbildung *dom* auf **att** weist jedem Attribut eine bestimmte Domäne zu, wobei $dom(A)$ die *Domäne* des Attributs A bildet.

Weiterhin gibt es eine abzählbar unendliche Menge **relname** von *Relationsbezeichnern*. Eine partielle Funktion $sort : \mathbf{relname} \rightarrow I\!P^{fin}(\mathbf{att})$ bildet die Relationsbezeichner in die Menge der endlichen Teilmengen von **att** ab und ordnet so einem Relationsbezeichner eine Menge von Attributen zu. Es wird davon ausgegangen, dass die Menge der Attribute geordnet ist, wenn eine Attributliste angegeben wird. Ein *Relationstyp* ist ein Relationsbezeichner R oder $R\ [\ A_1, \ldots, A_n\ ]$, falls angezeigt werden soll, dass dieser die Attribute $A_1, \ldots, A_n$ enthält also $sort(R)=\{\ A_1, \ldots, A_n\ \}$ gilt. Ein *relationales Schema* ist eine nichtleere, endliche Menge **R** von Relationstypen.

Eine *Instanz* eines Relationstyps $R\ [\ A_1, \ldots, A_n\ ]$ ist eine Relation R von Tupeln t mit $t \in dom(A_1) \times \ldots \times dom(A_n)$. Auf einen einzelnen Wert eines Tupels kann auch über einen Attributbezeichner A mit $t(A)$ zugegriffen werden, wenn $t \in R\ [\ A_1, \ldots, A,$

Buch

ISBN	Titel	Jahr
0-201-53781-X	Database theory	1997
1-55867-622-X	Data and the Web	2001

Autor

Autorenname	Institutsname	ISBN
S. Maier	I.N.R.I.A.	0-201-53781-X
P. Hill	I.N.R.I.A.	1-55867-622-X
R. Scholl	Bell Labs	0-201-53781-X

Institut

Institutsname	Ortsname
I.N.R.I.A	Rocquencourt
I.N.R.I.A.	Rennes
Bell Labs	Murray Hill

Abbildung 5.1: *Drei Relationen*

$\ldots, A_n$]. Eine *Schemainstanz* über einem relationalen Schema $\mathbf{R}$ ist eine Abbildung $\mathbf{I}$ mit Domäne $\mathbf{R}$, so dass $\mathbf{I}(R)$ eine Relation über dem Relationstyp R ist für alle $R \in \mathbf{R}$.

Beispiel 5.1 (*Relationales Schema*)
Verwendet man die eingeführte Notation, wird das relationale Schema aus Beispiel 2.1 und die zugehörige Datenbankinstanz aus Abb. 5.1 wie folgt definiert:

$$
\begin{array}{lcl}
\mathbf{R} & = & \{ \text{ Buch, Autor, Institut } \} \\
 & & \\
sort(Buch) & = & \{ \text{ ISBN, Titel, Jahr } \} \\
sort(Autor) & = & \{ \text{ Autorenname, Institutsname, ISBN } \} \\
sort(Institut) & = & \{ \text{ Institutsname, Ortsname } \} \\
 & & \\
dom(ISBN) & = & \text{string} \\
dom(Titel) & = & \text{string} \\
dom(Jahr) & = & \text{integer} \\
dom(Autorenname) & = & \text{string} \\
dom(Institutsname) & = & \text{string} \\
dom(Ortsname) & = & \text{string}
\end{array}
$$

$$
\begin{aligned}
\mathbf{I}(Buch) &= \{ \langle \text{ 0-201-53781-X, Database theory, 1997 } \rangle, \\
&\quad \langle \text{ 1-55867-622-X, Data and the Web, 2001 } \rangle \} \\
\mathbf{I}(Autor) &= \{ \langle \text{ S. Maier, I.N.R.I.A., 0-201-53781-X } \rangle, \\
&\quad \langle \text{ P. Hill, I.N.R.I.A., 1-55867-622-X } \rangle, \\
&\quad \langle \text{ R. Scholl, Bell Labs, 0-201-53781-X } \rangle \} \\
\mathbf{I}(Institut) &= \{ \langle \text{ I.N.R.I.A., Rocquencourt } \rangle, \\
&\quad \langle \text{ I.N.R.I.A., Rennes } \rangle, \\
&\quad \langle \text{ Bell Labs, Murray Hill } \rangle \}
\end{aligned}
$$

R bezeichnet die drei vorkommenden Relationstypen, wobei die Funktion *sort* jedem Relationstyp seine Attribute zuweist. *dom* legt bei jedem Attribut die zugehörige Domäne fest. Die Bezeichner *string* und *integer* stehen hierbei für die Domäne der Zeichenfolgen bzw. für die Domäne der Zahlen. Die Attribute $ISBN$ und $Institutsname$ treten jeweils bei zwei Relationstypen auf. Sollen diese Attribute in diesen beiden Relationen unterschiedliche Domänen besitzen, sind sie so umzubenennen, dass die Zuordnung der Attributbezeichner zu den Relationstypen eindeutig wird. Dann können verschiedene Domänen angegeben werden.

Die Funktion **I** weist jedem Relationstyp eine Relation zu, welche die zu den Attributen passenden Tupel enthält. $\mathbf{I}(R)$ wird auch vereinfacht als R geschrieben, wenn aus dem Zusammenhang ersichtlich ist, dass es sich um eine Relation handelt. □

Das Beispielschema macht bis auf die Vorgabe von Domänen keine Einschränkungen bezüglich der Menge an Tupeln, die zu einer Relation gehören können. Will man aber etwa modellieren, dass ein Buch durch seine ISBN eindeutig gekennzeichnet ist und daher der Wert beim Attribut $ISBN$ die Werte der anderen Relationsattribute $Titel$ und $Jahr$ eindeutig bestimmt, sind nicht alle aufgrund der Domänen möglichen Tupel auch erlaubt. Um solche Abhängigkeiten zwischen Werten auszudrücken, werden Konsistenzbedingungen in Form von funktionalen Abhängigkeiten verwendet.

Definition 5.1 (*Funktionale Abhängigkeit*)
Sei U eine Menge von Attributen, dann ist eine *funktionale Abhängigkeit* über U ein Ausdruck der Form $X \rightarrow Y$, wobei $X, Y \subseteq U$. Eine Relation R über U erfüllt $X \rightarrow Y$, $R \models X \rightarrow Y$, wenn für alle Tupel t und s von R aus $\pi_X(s) = \pi_X(t)$ folgt, dass $\pi_Y(s) = \pi_Y(t)$ gilt. Eine funktionale Abhängigkeit auf einem Schema **R** ist ein Ausdruck $R : X \rightarrow Y$, wobei $R \in \mathbf{R}$ und $X \rightarrow Y$ eine Abhängigkeit über $sort(R)$ ist. Eine Instanz **I** über **R** erfüllt $R : X \rightarrow Y$, wenn $\mathbf{I}(R)$ die Abhängigkeit $X \rightarrow Y$ erfüllt.

Soll die Abhängigkeit der Attribute $Titel$ und $Jahr$ von $ISBN$ innerhalb des Beispielschemas ausgedrückt werden, könnte eine funktionale Abhängigkeit $Buch : ISBN \rightarrow Titel, Jahr$ formuliert werden, die von der Beispielinstanz erfüllt werden würde.

Neben Konsistenzbedingungen, die sich auf die Tupel einer einzelnen Relation beziehen, existieren auch Bedingungen, die zwischen den Tupeln mehrerer Relationen gelten sollen. So macht es etwa im Fall des Beispiels Sinn, zu fordern, dass zu den ISBNs, die bei den Autoren angegeben sind, auch die entsprechenden Bücher in der Relation $Buch$

eingetragen sein müssen. Solche Bedingungen können im Datenmodell durch Inklusionsabhängigkeiten der Form $Autor[ISBN] \subseteq Buch[ISBN]$ modelliert werden.

Definition 5.2 (*Inklusionsabhängigkeit*)
Sei $\mathbf{R}$ ein relationales Schema. Eine *Inklusionsabhängigkeit* über $\mathbf{R}$ ist ein Ausdruck der Form $R_1 [A_1 , \ldots, A_n] \subseteq R_2 [B_1, \ldots, B_n]$. Eine Instanz $\mathbf{I}$ über $\mathbf{R}$ erfüllt $R_1 [A_1, \ldots, A_n] \subseteq R_2 [B_1, \ldots, B_n]$, $\mathbf{I} \models R_1 [A_1, \ldots, A_n] \subseteq R_2 [B_1, \ldots, B_n]$, falls $\mathbf{I}(R_1) [A_1, \ldots, A_n] \subseteq \mathbf{I}(R_2) [B_1, \ldots, B_n]$.

Eine Menge von funktionalen und Inklusionsabhängigkeiten wird mit $\mathcal{D}$ bezeichnet, wobei eine Schemainstanz $\mathbf{I}$ über $\mathbf{R}$ diese erfüllt, $\mathbf{I} \models \mathcal{D}$, falls $\mathbf{I} \models d$ für alle $d \in \mathcal{D}$.

Neben diesen beiden gängigen Vertretern wurde im Laufe der Zeit noch eine ganze Reihe weiterer Konsistenzbedingungen vorgeschlagen, durch deren Einsatz mehr Semantik in das relationale Modell eingebracht wird. Da funktionale Abhängigkeiten und Inklusionsabhängigkeiten oder Unterarten davon häufig in realen Datenbanksystemen Verwendung finden, sollen die weiteren Betrachtungen aber auf diese beiden beschränkt bleiben.

5.1.3 Abbildungen

Abbildungen zwischen zwei Schemata werden im relationalen Modell mit Hilfe der Operatoren der relationalen Algebra definiert. Diese wählen aus den Relationen des Datenbankschemas eine Menge von Tupeln aus und erzeugen aus ihnen eine Ergebnisrelation. Da jeder Operator wieder eine Relation liefert, können die Operatoren auch hintereinander verkettet und hierdurch komplexere Abbildung definiert werden. Eine Abbildung selbst impliziert ein für sie passendes Anwendungsschema, da ein Operator als Ergebnis wieder eine Relation liefert und zu dieser ein Relationstyp abgeleitet werden kann.

Zur relationalen Algebra gehören unter anderem die Selektion, die Projektion und das kartesische Produkt.

Selektion

Bei einer Selektion werden Tupel anhand ihrer Attributwerte aus einer Relation $R [A_1, \ldots, A_n]$ ausgewählt und in die Bildrelation übernommen. Das Selektionskriterium besteht aus einem Vergleich, welcher den Wert eines Attributs A_i innerhalb eines Tupels gegen eine Konstante c vergleicht

$$\sigma_{A_i=c}(R) = \{t \mid t \in R \wedge t(A_i) = c\}$$

und diejenigen Tupel t der Relation R auswählt, die den gesuchten Wert c beim Attribut A_i enthalten. Neben einem Vergleich mit einer Konstanten ist auch der Vergleich der Werte zweier Attribute $A_i = A_j$ der Relation R in der Selektionsbedingung möglich.

Projektion

Bei einer Projektion werden aus allen Tupeln einer Relation $R [A_1, \ldots, A_n]$ die Werte

eines oder mehrerer Attribute ausgewählt und zu einer Ergebnisrelation vereinigt:

$$\pi_{A_i,\ldots,A_j}(R) = \{\langle t(A_i), \ldots, t(A_j)\rangle \mid t \in R\}$$

Innerhalb der Ergebnisrelation sind die Tupel wieder eindeutig.

Kartesisches Produkt
Bei einem kartesischen Produkt werden die Tupel zweier Relationen R_1 [A_1, ..., A_n] und R_2 [B_1, ..., B_m] so miteinander verknüpft, dass aus jedem Tupel t der einen Relation in Kombination mit jedem Tupel s der anderen Relation ein neues Tupel im Ergebnis angelegt wird:

$$R_1 \times R_2 = \{\langle t(A_1), \ldots, t(A_n), s(B_1), \ldots, s(B_m)\rangle \mid t \in R_1 \wedge s \in R_2\}$$

Abbildungen, in denen Selektion, Projektion und das kartesische Produkt verwendet werden, bilden die Menge der konjunktiven Anfragen. Diese drei Operatoren formen zusammen die *SPC-Algebra* über Relationen.

5.1.4 Operationen

Operatoren, die eine Schemainstanz in eine Instanz desselben Schemas umwandeln, liegen außerhalb der Algebra, da die Operatoren der Algebra nur lesend auf Daten zugreifen und somit eine existierende Instanz nicht verändern können. Mit datenändernden Operatoren wird ein Tupel in eine Relation eingefügt, ein Tupel gelöscht oder die Attributwerte eines Tupels geändert:

- $ins(R, t)$ fügt ein Tupel t in die Relation R ein,
- $del(R, t)$ löscht ein Tupel t aus der Relation R und
- $mod(R, t_1 \rightarrow t_2)$ ersetzt das Tupel t_1 durch t_2 in R.

Hier enthält das einzufügende bzw. das zu löschende Tupel dieselben Attribute wie die jeweils angegebene Relation. In existierenden Datenbanksystemen werden der *del*- und der *mod*-Operator häufig mit einer Selektion verbunden, so dass nicht nur ein einzelnes Tupel gelöscht bzw. modifiziert wird, sondern eine Menge von Tupeln ausgewählt wird. Dieses Verhalten kann durch eine Folge mehrerer Einzeloperationen *del* bzw. *mod* nachgebildet werden.

5.2 View-Update-Problem

Auf Grundlage des relationalen Datenmodells wird das View-Update-Problem formuliert. Eine Abbildungsbeschreibung stellt sich im relationalen Modell wie in Abb. 5.2 gezeigt dar. Das Datenbankschema ist hierbei ein relationales Schema bestehend aus

einer Menge von Relationstypen. Diese legen eine Menge von Relationen fest, die zusammen eine Schemainstanz wie $\mathbf{I_1}$ bilden. Eine Abbildung bzw. Sicht f ist durch einen Ausdruck der SPC-Algebra gegeben, indem Selektion, Projektion und kartesisches Produkt zu einer Anfrage kombiniert werden. Passend zur Abbildungsdefinition gibt es ein relationales Anwendungsschema, welches einen Relationstyp enthält und zu dessen Instanzenmenge die Ergebnisrelation $f(\mathbf{I_1})$ der Abbildung f gehört. Die Operationen ν und μ werden durch die Operatoren *ins*, *del* und *mod* oder einer Kombination dieser drei angegeben.

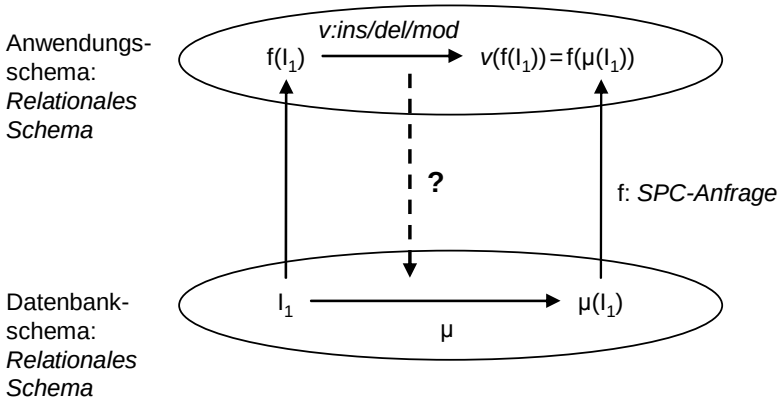


Abbildung 5.2: *View-Update-Problem: Finde für eine Anwendungsoperation ν eine geeignete Umsetzung in Form einer Operation μ*

Ist eine Sicht gegeben, stellt sich die Frage, wie Änderungsoperationen ν über den Anwendungsdaten in Änderungen μ in der Datenbasis umgesetzt werden können (Abb. 5.2). Es ergibt sich das *View-Update-Problem*.

Definition 5.3 (*View-Update-Problem*)
Sind eine Abbildung $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$, Instanzen $\mathbf{I_1} \in dom(\mathbf{S})$, $f(\mathbf{I_1})$ und eine Änderungsoperation ν auf $f(\mathbf{I_1})$ gegeben, finde eine Umsetzung μ, so dass $\nu(f(\mathbf{I_1})) = f(\mu(\mathbf{I_1}))$ gilt.

Ziel ist es, für eine möglichst große Klasse von Sichten f und für viele Operationen ν ein geeignetes μ angeben zu können. Ein geeignetes μ muss nicht immer existieren. Ebenso kann zu einem ν eine Vielzahl von möglichen Umsetzungen μ vorhanden sein, aus denen eine auszuwählen ist. Etliche Schwierigkeiten, die bei der Suche nach einem μ auftreten können, listen etwa [FC85] auf.

5.3 Verwandte Arbeiten

Untersuchungen zum View-Update-Problem ziehen sich von dessen erster Formulierung etwa in [Cod74] durch die Historie der Datenbanktheorie (vgl. etwa [Ull89, LL95, Dat00, Vos00, GMUW02]). Im Laufe der Zeit wurden eine Vielzahl von Untersuchungen und Ergebnissen zu diesem Thema vorgestellt, die basierend auf ihrer Grundidee in mehrere Gruppen eingeteilt werden können. Hier gibt es Verfahren, die eine Umkehrabbildung zu einer Sicht bestimmen, Verfahren, welche die Daten, die nicht durch die Abbildung abgebildet werden, mit in die Betrachtungen einbeziehen, und Ansätze, die den Sichtenersteller selbst passende Umsetzungen für Änderungsoperationen bestimmen lassen.

Betrachtet man als Beispiel den Relationstyp $Buch$ [$ISBN$, $Titel$, $Jahr$] mit einer funktionalen Abhängigkeit $ISBN \rightarrow Titel$ und eine Sicht als $\sigma_{Jahr \geq 2000}(Buch)$, ist also die Frage zu klären, welche Änderungsoperationen auf der Sicht durchgeführt werden können und welche nicht.

5.3.1 Umkehrabbildung

Es liegt nahe, bei einer Abbildung diejenigen Tupel in der Datenbank zu suchen, aus denen ein Tupel in der Sicht hergeleitet wird. Wird eine Änderungsoperation auf der Sicht durchgeführt, so werden die Tupel in der Datenbank bestimmt, aus denen das zugehörige Sichttupel abgeleitet wird. Wird ein Wert im Sichttupel geändert, muss dieser Wert in den zugehörigen Datenbanktupeln geändert werden; wird ein Sichttupel gelöscht, werden die zugehörigen Datenbanktupel gelöscht, etc.

[DB78, DB82] betrachten Sichten über Relationen mit einattributigen Schlüsseln, wobei zur Sichtendefinition nur Projektionen $\pi_A(R)$ und kartesische Produkte mit nachgeschalteter Selektion auf Gleichheit von gleichbenannten Attributen $\sigma_{R_1.A=R_2.A}(R_1 \times R_2)$ (Equi-Joins) zugelassen werden. Zusätzlich sind funktionale Abhängigkeiten auf dem Datenbankschema erlaubt. Ein Sichtgraph gibt an, welche Attributwerte innerhalb eines Tupels Werte bei anderen Attributen eindeutig festlegen. Die Attribute der Datenbankrelationstypen und des Sichtrelationstyps bilden die Knotenmenge des Graphen. Zwischen Sichtattributen und den zugehörigen Datenbankattributen werden jeweils bidirektionale Kanten in den Graph eingefügt, da sich der Wert eines Sichtattributs aus dem zugehörigen Datenbankattribut ableitet und umgekehrt. Ebenso werden zwischen den an der Selektionsbedingung von Equi-Joins beteiligten Attributen jeweils bidirektionale Kanten eingefügt, da bei einem Tupel, das aus einem solchen Join entsteht, diese Attributwerte sich gegenseitig eindeutig festlegen. Für funktionale Abhängigkeiten werden unidirektionale Kanten entsprechend der Abhängigkeitsrichtung angelegt. Je nach betrachteter Sicht kann anhand des zugehörigen Sichtgraphen bestimmt werden, ob Einfüge- oder Löschoperationen von Tupeln auf der Sicht durchgeführt werden können. Ausschlaggeben ist hierbei, ob von einem Schlüsselattribut aus alle zu ändernden Attribute in der Sicht und der Datenbank über den Graph erreicht werden.

[BDH03a, BDH03b] nehmen eine Erweiterung der Idee des Sichtgraphen auf XML-

Sichten über relationalen Datenbanken vor, wobei zusätzlich Schachtelungsoperatoren zur Sichtendefinition zugelassen werden.

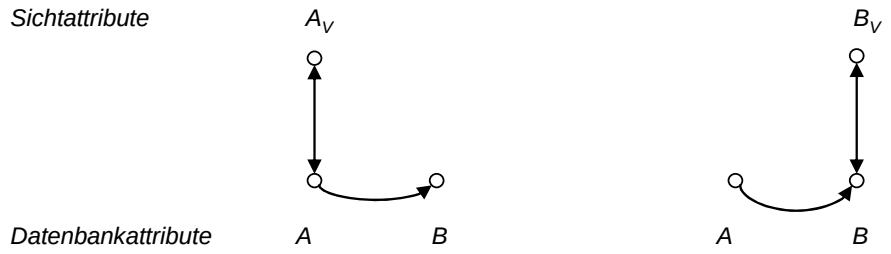


Abbildung 5.3: *Sichtgraphen für zwei unterschiedliche Projektionen*

Ist ein Datenbankrelationstyp als R [A, B] mit einer funktionalen Abhängigkeit $A \rightarrow B$ gegeben und wird hierauf eine Sicht $\pi_A(R)$ definiert, entsteht ein Sichtgraph wie er in Abb. 5.3 links zu sehen ist. Für die Attribute A und B des Datenbankrelationstyps wird je ein Knoten angelegt und ebenso wird mit dem Attribut A_v des Sichtrelationstyps verfahren. Ein Wert des Sichtattributs A_v leitet sich aus A ab, so dass sich eine bidirektionale Kante zwischen den entsprechenden Knoten ergibt. Das Attribut A des Datenbankrelationstyps bestimmt durch die funktionale Abhängigkeit den Wert beim Attribut B, so dass in den Graph eine unidirektionale Kante von A nach B eingefügt wird. Wird ein Tupel in der Sicht eingefügt, also ein Wert des Attributs A_v ändert sich, sind vom Knoten A_v aus alle anderen Knoten des Graphen erreichbar, insbesondere der Schlüssel A. Somit ist es erlaubt, das Tupel in der Sicht einzufügen, wobei der fehlende Wert für das Attribut B in der Datenbankrelation R als NULL gesetzt wird. Bei einer zweiten Sicht $\pi_B(R)$, die anstatt des Attributs A das Attribut B projiziert, ist ein Einfügen eines Tupels in die Sichtrelation nicht mehr möglich (Abb. 5.3 rechts). Von Knoten B_v ausgehend sind nicht alle anderen Knoten des Graphen erreichbar.

Dieser Ansatz nimmt offensichtlich bei der Menge der betrachteten Sichten eine Auswahl vor, da etwa Sichten, die eine Selektion verwenden, überhaupt nicht untersucht werden. Zum anderen wird bei allen Datenbankrelationen, die abgebildet werden, ein einattributiger Schlüssel gefordert. Für Sichten, welche diese Voraussetzungen nicht erfüllen, werden keinerlei Aussagen gemacht. Zu diesen gehört die Beispielsicht über dem Relationstyp *Buch*, da zum einen *Buch* keinen Schlüssel besitzt und die angegebene Beispielabbildung eine Selektion verwendet.

[Kel95] verfolgt einen ähnlichen Ansatz, lässt aber die Selektion als Abbildungsoperation zu. Es werden Datenbankrelationstypen untersucht, die jeweils einen einattributigen Schlüssel besitzen, wobei diese Schlüssel durch die Abbildung mit auf die Anwendungsseite abgebildet werden müssen. Hierdurch wird über die Schlüsselwerte eine eindeutige Zuordnung von Tupeln in der Sicht zu Tupeln in der Datenbank sichergestellt. Wird auf der Sicht ein Tupel geändert, steht durch die Zuordnung fest, welche Tupel in der Daten-

bank zu ändern sind, und nur diese werden dann auch bei einer Umsetzung modifiziert.

Dieser Ansatz macht zwar keine Einschränkung bei den zur Sichtendefinition verwendbaren Operatoren – eine Selektion ist hier zulässig – doch werden wieder nur Relationstypen mit Schlüsseln betrachtet. Zudem müssen diese Schlüssel mit in die Sicht übertragen werden, so dass wieder etliche Sichten überhaupt nicht untersucht werden können. Für die Beispielsicht sind wieder keine Aussagen möglich.

Mit Hilfe der ersten beiden in diesem Abschnitt vorgestellten Ansätze lässt sich bestimmen, woher die Tupel kommen, aus denen ein Tupel in der Sicht gebildet wird. Verfolgt man diesen Gedanken weiter, kann versucht werden, zu jedem Operator, der in einer Sichtendefinition Anwendung findet, einen inversen Operator zu bestimmen [BL98]. Der inverse Operator gibt dann zu einem Operator der SPC-Algebra an, welche seiner Ergebnistupel in der Sicht durch welche Eingangstupel in der Datenbank erzeugt werden. Wird eine Sicht durch eine Verkettung von Operatoren beschrieben, kann durch die Verkettung der jeweils inversen Operatoren bestimmt werden, welche Tupel in der Sicht durch welche Tupel in der Datenbank erzeugt werden. Da ein inverser Operator nicht eindeutig festgelegt sein muss, werden im Rahmen des Ansatzes für ein Sichttupel alle möglichen Tupel-Konstellationen in der Datenbank angegeben, aus denen das Sichttupel entstehen kann.

In [Mas84, Kel86, KU84] werden inverse Operatoren nicht pauschal für einen gegebenen Operator bestimmt, sondern es wird ein Satz von Regeln angegeben, mit denen aus einer Menge möglicher inverser Operatoren ein passender ausgewählt werden kann. In diese Auswahl wird ein Sichtenersteller mit einbezogen. Untersucht werden hier wieder nur Sichten, bei denen einattributige Schlüsselattribute vorhanden sind.

Um den Ursprung eines Tupels in der Sicht zu beschreiben, verwendet [Shu98] in der Sicht Relationen mit zugehörigen Konsistenzbedingungen in Form von boolschen Ausdrücken, wobei diese angeben, welche Bedingungen in der Datenbankrelation vorliegen müssen, um ein bestimmtes Sichtentupel zu generieren. Wird dann auf der Sicht ein Tupel eingefügt oder gelöscht, lassen sich die nötigen Bedingungen in der Datenbank ermitteln und somit die Änderung auf der Datenbank durchführen.

Bei der Bestimmung von inversen Operatoren gibt es einige Probleme. So müssten auf dem Datenbankschema vorhandene Konsistenzbedingungen ebenfalls bei der Betrachtung mit einbezogen werden. Wird eine Projektion auf einem Schlüsselattribut definiert, ist jedem Tupel in der Sicht eindeutig ein Tupel in der Datenbank zugewiesen. Bei einer Projektion auf einem Nichtschlüsselattribut kommt dies Eindeutigkeit abhanden, da mehrere Datenbanktupel auf ein Sichttupel abgebildet werden können. Die Eigenschaften der Umkehrabbildung hängen also nicht nur vom eigentlichen Operator ab, sondern ebenso von eventuell vorhandenen Konsistenzbedingungen. Des Weiteren fehlt bei diesen Ansätzen eine Einbeziehung des Bildbereichs der Abbildung, da davon ausgegangen wird, dass für jedes Tupel auf dem Sichtschema durch die Umkehroperatoren ein oder mehrere passende Datenbanktupel gefunden werden können. Kann das angegebene Sichttupel aber nie durch die Abbildung erzeugt werden, gibt es folglich auch keine zugehörigen Datenbanktupel und ein inverser Operator auf einem solchen Tupel

macht keinen Sinn. Ergänzt man in der Beispielsicht über der Relation *Buch* etwa ein Tupel, welches die Selektionsbedingung der Sichtdefinition nicht erfüllt, können für dieses Tupel keine Datenbanktupel bestimmt werden, da keine vorhanden sind, aus denen es generiert werden könnte.

Ein inheräntes Problem bei alle diesen Ansätzen, die eine Umkehrabbildung bestimmen, ist, dass davon ausgegangen wird, dass durch die Operation auf dem Anwendungsschema eine Instanz des Bildbereichs der Abbildung erreicht wird. Erst dann macht es Sinn, eine Umkehrung zu berechnen, da nur dann mindestens eine passende Datenbankinstanz vorhanden ist. Führt die Anwendungsoperation zu einer Instanz außerhalb des Bildbereichs, existiert keine zugehörige Datenbankinstanz, so dass diese auch nicht berechnet werden kann. Eine Überprüfung, ob die Instanz zum Bildbereich gehört, findet aber nicht statt. Stattdessen wird davon ausgegangen, dass alle auf der Sicht definierten Änderungsoperationen auch auf der Sicht ausgeführt werden können.

5.3.2 Komplement

Da zu einer Anwendungsoperation zumeist mehrere mögliche Umsetzungen vorhanden sind, ist eine Idee zur Auflösung dieser Mehrdeutigkeit, diejenigen Datenbankdaten, die nicht durch die Sicht abgebildet werden, mit in die Auswahlentscheidung einzubeziehen.

Bei der Definition einer Sicht gehen in der Regel Informationen über den Zustand im Datenbankschema verloren, da mehrere Datenbankinstanzen auf dieselbe Sichtinstanz abgebildet werden. Somit lassen sich Sichtinstanzen nicht mehr eindeutig einer Datenbankinstanz zuordnen, da Mehrdeutigkeiten vorhanden sind. In [BS81] wird vorgeschlagen, diese Mehrdeutigkeit dadurch aufzulösen, dass die Daten, die nicht in der Sicht auftauchen, als Entscheidungshilfe zu Rate gezogen werden. Hierzu wird die Sicht f durch eine weitere Abbildung g geeignet ergänzt, so dass beide zusammengenommen auf allen Datenbankinstanzen eine injektive Abbildung in die Menge der Sichtinstanzen beschreiben. Wird zu einer Sicht $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ eine Komplementabbildung $g : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ derart gewählt, dass für alle Datenbankinstanzen mit $\mathbf{I_1} \neq \mathbf{I_2}$

$$f(\mathbf{I_1}) = f(\mathbf{I_2}) \Rightarrow g(\mathbf{I_1}) \neq g(\mathbf{I_2})$$

gilt – d.h. wenn die Sicht f zwei unterschiedliche Datenbankinstanzen auf dieselbe Sichtinstanz abbildet, dann das Komplement g diesen beiden Datenbankinstanzen immer verschiedene Sichtinstanzen zuweist – dann ergibt sich bei kombinierter Betrachtung der Sicht und des Komplements als eine einzige Abbildung $f \times g : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$, dass

$$(f(\mathbf{I_1}), g(\mathbf{I_1})) = (f(\mathbf{I_2}), g(\mathbf{I_2})) \Rightarrow \mathbf{I_1} = \mathbf{I_2}$$

gilt. Dies bedeutet, dass die kombinierte Abbildung injektiv ist und somit eine Eins-zu-eins-Zuordnung zwischen Instanzen des Definitions- und des Bildbereichs festlegt. Folglich kann aus einem Sichtenzustand $f(\mathbf{I_1})$ und dem Zustand des Komplements $g(\mathbf{I_1})$

eindeutig der Zustand $\mathbf{I_1}$ der Datenbasis abgeleitet werden. Diese Zuordnung wird in Abb. 5.4 mit $(f \times g)^{-1}$ bezeichnet. Als triviales Komplement zu jeder Abbildung kann immer die Identitätsabbildung auf dem Datenbankschema gewählt werden, welche den kompletten Datenbasiszustand übernimmt. Zu einer Sicht kann mehr als ein mögliches Komplement existieren.

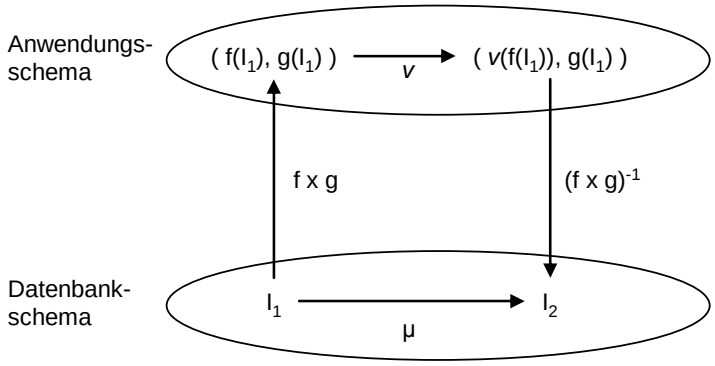


Abbildung 5.4: *Konstantes Komplement*

Wird auf der Sichtinstanz $f(\mathbf{I_1})$ eine Änderungsoperation ν ausgeführt, muss auf der kombinierten Abbildung $f \times g$ die Instanz des Komplements konstant bleiben, da hierauf im eigentlichen Sinne ja nicht zugegriffen werden kann. Es gilt also eine Umsetzung μ zu finden, bei der gilt, dass

$$\nu(f(\mathbf{I_1})) = f(\mu(\mathbf{I_1})) = f(\mathbf{I_2}) \text{ und } g(\mathbf{I_1}) = g(\mu(\mathbf{I_1})) = g(\mathbf{I_2})$$

d.h. das Komplement bleibt bei der Änderung konstant, so dass $(\nu(f(\mathbf{I_1})), g(\mathbf{I_1})) = (f(\mu(\mathbf{I_1})), g(\mu(\mathbf{I_1}))) = (f(\mathbf{I_2}), g(\mathbf{I_2}))$. Je weniger Daten ein Komplement folglich umfasst, desto geeigneter ist es, da so die Chancen, dass es von einer Änderung nicht betroffen ist, größer sind.

Sei ein Datenbankschema, eine Sicht f auf diesem Schema und eine Menge von Änderungsoperationen auf der Sicht gegeben, wobei das Verketten und das Zurücknehmen von Änderungen ebenfalls zur Menge gehört. Sind alle Operationen der Menge eindeutig umsetzbar, dann gibt es zu f ein konstantes Komplement g. Dies bedeutet umgekehrt, dass wenn zu einer Sicht f ein Komplement g gefunden werden kann, das bei einer Änderungsoperation konstant bleibt, dann diese Änderungsoperation eindeutig auf die Datenbank umsetzbar ist. Bei diesem Ansatz muss also zu einer gegebenen Abbildung f ein minimales, konstantes Komplement gefunden werden, um Aussagen über die durchführbaren Änderungsoperationen machen zu können.

Ist eine Relation R [A, B, C] mit $A \rightarrow B$ gegeben und die Sicht f als $\pi_{AB}(R)$ definiert, so kann als Komplement g die Abbildung $\pi_{AC}(R)$ gewählt werden. Sind die

Bilder beider Abbildungen $\pi_{AB}(R)$ und $\pi_{AC}(R)$ bekannt, kann hieraus der Zustand der Datenbankrelation R rekonstruiert werden. Ist in R ein Tupel $\langle$ a, b, c $\rangle$ vorhanden, so ergibt sich in der Sicht f ein Tupel $\langle$ a, b $\rangle$. Eine Änderungsoperation auf der Sicht, die das Komplement konstant lässt, ist die Modifikation dieses Tupels nach $\langle$ a, d $\rangle$, wobei deren Umsetzung das Datenbanktupel nach $\langle$ a, d, c $\rangle$ ändert. Ein Einfügen oder Löschen von Tupeln in der Sicht f kann nicht umgesetzt werden, da sich hierzu der Zustand des Komplements ändern müsste, weil dann ein Wert für das Tupelattribut C einzufügen oder zu löschen wäre.

Für die Suche nach einem passenden Komplement für eine gegebene Sicht werden in [BS81] keine Hinweise gegeben. [CP84] gibt eine Aufwandsabschätzung für eine Suche an. Werden für die Definition einer Sicht nur Projektionen auf einer einzelnen Relation mit funktionalen Abhängigkeiten zugelassen, so kann ein Komplement gefunden werden, indem mit der Identitätsabbildung begonnen wird und nach und nach Attribute aus dem Komplement entfernt werden, wobei bei jedem Schritt die Komplementeigenschaft zu überprüfen ist. Die Suche nach einem Komplement mit vorgegebener Größe – also auch die Suche nach einem minimalen – erweist sich als NP-vollständig.

[LV02, LV03, Lec03] betrachten nur Datenbankrelationen, die ein Schlüsselattribut besitzen und Sichten, bei denen immer die Schlüsselattribute mit in die Sicht übernommen werden. Mit diesen Voraussetzungen kann ein genügend kleines Komplement in Form einer Menge von Komplementabbildungen mit polynomialem Aufwand gefunden werden. Die Idee ist hierbei, das Komplement derart zu wählen, dass in eine Instanz des Komplements diejenigen Tupel übernommen werden, deren Schlüsselwerte nicht in der Sicht auftauchen, und bei Tupeln, die nicht vollständig in der Sicht erscheinen, die Schlüssel und die fehlenden Attribute ebenfalls mit in das Komplement zu nehmen. Hierdurch können aus den Sichtdaten und den Daten der Komplementabbildungen die Datenbankdaten wiederhergestellt werden.

Wie die Menge der durchführbaren Änderungsoperationen bei einer konkreten Sicht aussieht, wird in [BS81] ebenfalls nicht beantwortet. Um einer solchen Beschreibung aus dem Weg zu gehen, betrachtet [KU84] nur solche Abbildungen f, bei denen die aus Sicht und konstantem Komplement kombinierte Abbildung $f \times g$ surjektiv ist, d.h. durch die Abbildung werden alle Instanzen des Anwendungsschemas erreicht. Da $f \times g$ somit eine Bijektion darstellt, sind alle Operationen auf dem Anwendungsschema durchführbar. Ein Komplement g zu einer Sicht f mit dieser Eigenschaft ist eindeutig festgelegt. Beispiele für solche Abbildungen werden in [KU84] nicht gegeben.

Die Ansätze, die der Idee des konstanten Komplements aus [BS81] folgen, liefern Ergebnisse meist theoretischer Natur, die aber oft nicht auf eine konkrete Sicht übertragen werden können. Es ist zum einen unklar, wie ein Komplement überhaupt ermittelt werden kann. Bei einer gegebenen Sicht muss dieses geeignet geraten werden, da Berechnungsverfahren nur für eine Teilmenge der möglichen Sichten vorhanden sind. Für die Beispielabbildung über dem Relationstyp *Buch* kann etwa mit keinem Verfahren ein Komplement berechnet werden. Ist ein Komplement gefunden, steht allerdings noch nicht fest, welche konkreten Änderungsoperationen – Einfügen oder Löschen von Tu-

peln – auf der Sicht überhaupt durchgeführt werden können. Hierzu werden bei den bestehenden Ansätzen ebenfalls keine Angaben gemacht.

Unter Umständen ist die Forderung nach einem konstanten Komplement zu rigide, d.h. es gibt Sichten, bei denen unter konstantem Komplement eine Änderungsoperation nicht umsetzbar ist, obwohl eine passende Umsetzung vorhanden ist. [Kel87] zeigt an einem Beispiel, dass es neben den Sichten mit konstantem Komplement Sichten ohne konstantem Komplement gibt, für die ebenfalls Umsetzungen bestimmt werden können.

Eine echt größere Klasse von Sichten als die mit konstantem Komplement, bei denen ebenfalls Umsetzungen bestimmt werden können, beschreibt [GPZ88] und bezeichnet diese Sichten als konsistente Sichten. Ist eine Menge von Änderungsoperationen U auf der Sicht gegeben und ist $\tau(\nu)$ die passende Umsetzung der Operation ν auf der Datenbank, so muss bei einer konsistenen Sicht f für Instanzen $\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3} \in dom(\mathbf{S})$ des Datenbankschemas gelten:

$$\exists \nu \in U : \tau_1(\nu)(\mathbf{I_1}) = \mathbf{I_2} \wedge \tau_2(\nu)(\mathbf{I_1}) = \mathbf{I_3} \wedge f(\mathbf{I_2}) = f(\mathbf{I_3}) \Rightarrow \mathbf{I_2} = \mathbf{I_3}$$

Es darf also für eine Änderungsoperation nicht zwei mögliche Umsetzungen $\tau_1(\nu)$ und $\tau_2(\nu)$ geben, welche in der Datenbank zu verschiedenen Zuständen $\mathbf{I_2}$ und $\mathbf{I_3}$ führen, die wiederum in denselben Sichtzustand abgebildet werden (Abb. 5.5). Konsistente Sichten sind eine echte Obermenge der Sichten mit konstantem Komplement und werden so interpretiert, dass das Ergebnis eines fiktiven Komplements $g(\mu(\mathbf{I}))$ nicht konstant, sondern nur eindeutig bestimmbar sein muss. Hierbei kann es auch kleiner werden, d.h. es kann nach einer Änderung weniger Tupel umfassen als vor der Änderung.

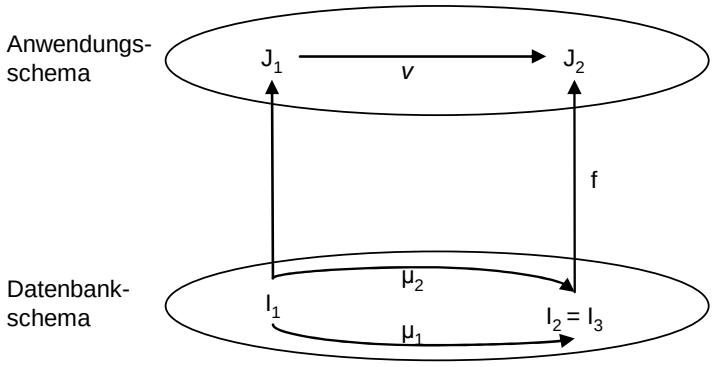


Abbildung 5.5: *Konsistente Sicht*

Ist eine Relation R [A, B] und die Sicht $\pi_A(R)$ gegeben, so muss als Komplement die Identitätsabbildung gewählt werden, um aus Sicht und Komplement den Datenbankzustand rekonstruieren zu können. Folglich würde der Ansatz des konstanten Komplements keine Änderungsoperationen erlaubt, da sich bei jeder der Operationen auch der Komplementzustand ändern würde. Gibt es in der Datenbank aber nur ein Tupel ⟨ a, b ⟩ mit

einem Wert von a beim ersten Attribut, gehört hierzu ein Tupel ⟨ a ⟩ in der Sicht. Wird dieses Tupel in der Sicht gelöscht, gibt es für diese Löschoperation eine eindeutige Umsetzung, die darin besteht, das Tupel ⟨ a, b ⟩ aus der Datenbankrelation zu entfernen. Das Komplement bleibt nicht konstant, da der Wert b, der nicht durch die Sicht abgebildet wird, ebenfalls aus der Datenbankrelation verschwindet. Doch ist das Ergebnis einer Umsetzung eindeutig bestimmt und die Sicht somit konsistent.

Ähnlich wie beim Komplement werden auch in [GPZ88] keine Hinweise gegeben, wie eine Übertragung der Ergebnisse auf eine konkrete Sicht aussehen kann. Es wird kein Verfahren angegeben, mit dem überprüft werden kann, ob eine Sicht konsistent ist oder nicht.

All diese Herangehensweisen an das View-Update-Problem auf abstrakter Ebene haben die Schwierigkeit, die theoretisch gewonnenen Ergebnisse bei einer konkreten Sichtendefinition anzuwenden. So wird kein Verfahren vorgeschlagen, mit dem bei einer gegebenen Abbildung ein Komplement berechnet werden kann. Genauso wenig wird angegeben, welche Operationen in Form von Einfüge- oder Löschoperationen nun tatsächlich durchführbar sind und welche nicht.

5.3.3 Abstrakte Datentypen

Neben Ansätzen, die versuchen, automatisch eine Umsetzung zu berechnen, gibt es auch Ansätze, bei denen ein menschlicher Sichtenersteller auf der Sicht erlaubte Änderungsoperationen bestimmt und gleichzeitig für diese eine gültige Umsetzung angibt.

Sichten können als abstrakte Datentypen aufgefasst werden, wobei die auf der Sicht angegebenen Änderungsoperationen dann die Operationen dieses Typs darstellen. Bei einer Sichtendefinition wird nicht nur angegeben, wie Daten aus dem Datenbankschema in das Anwendungsschema abgeleitet werden, sondern auch wie die Änderungsoperationen implementiert werden, d.h. für jede Änderungsoperation auf der Sicht wird bei der Sichtenerstellung jeweils die Umsetzung auf Operationen der Datenbank mit angegeben.

[RS79] und [HST99] betrachten Sichten als abstrakte Datentypen, wobei zur Definition von Sichten jeweils eine eigene, ausdrucksmächtige Abbildungssprache verwendet wird. Somit können auf vielfältige Art und Weise Sichten definiert werden, wobei ein Sichtenersteller für jede auf der Sicht durchführbare Operation eine Umsetzung angibt. Zu diesen Operationen gehören einmal die lesenden Operationen, die beschreiben, wie die Daten aus dem Datenbankschema ins Anwendungsschema übertragen werden, und zum anderen schreibende Operationen, bei denen angegeben wird, welche Operationen auf der Datenbankseite durchzuführen sind, wenn in der Sicht Daten gelöscht, hinzugefügt oder geändert werden.

Da bei diesen Ansätzen eine eigene Sichtendefinitionssprache eingesetzt wird, können die Eigenschaften einer mit ihr definierten Abbildung nicht formal bestimmt bzw. überprüft werden. Es kann etwa nicht getestet werden, ob die Lese- und Schreibrichtung einer Abbildung zusammenpassen. Da ein Korrektheitsbegriff für sinnvolle Abbildungen fehlt, können durch die Umsetzung einer Operation ungewollte Effekte ausgelöst werden, da

diese ja frei definierbar ist. Zudem muss für jede Operation auf dem Anwendungsschema eine Implementierung angegeben werden, wodurch der Aufwand, eine Sicht zu erstellen, wächst.

5.4 Reduktion des View-Update-Problems

Existierende Ansätze beschränken bei Betrachtungen des View-Update-Problems zumeist die Menge der Sichten, für die überhaupt Aussagen gemacht werden. Im Folgenden sollen Abbildungsbeschreibungen in Form von relationalen Sichten durch eine Reduktion auf die in Kapitel 3 eingeführten Baumautomaten untersucht werden.

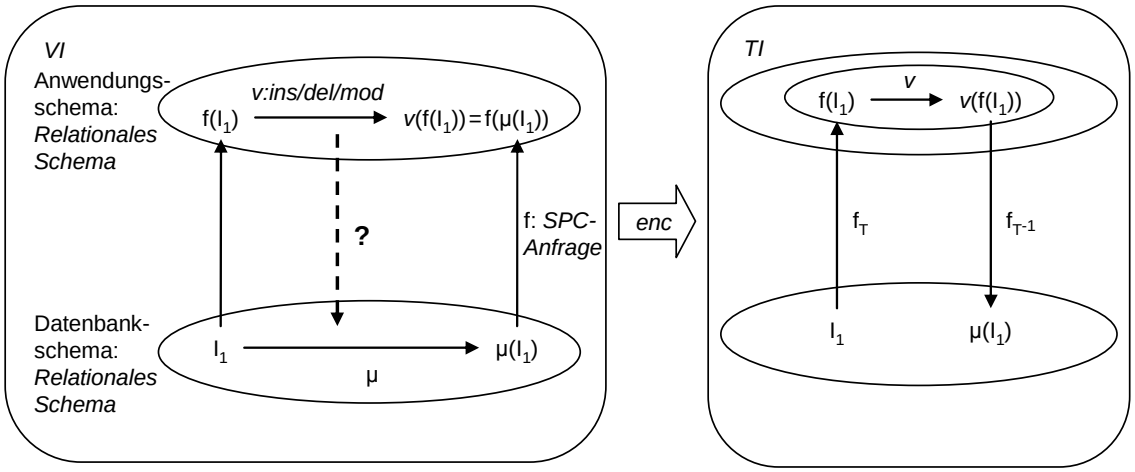


Abbildung 5.6: *Kodierung einer relationalen Sicht als Abbildung zwischen Bäumen*

Abb. 5.6 gibt einen Überblick über die einzuschlagende Vorgehensweise. Eine Abbildungsbeschreibung VI ist gegeben durch ein relationales Datenbankschema, einen Ausdruck der relationalen Algebra als Abbildung f und ein ebenfalls relationales Anwendungsschema. Auf diesem soll eine durch die Änderungsoperatoren *ins*, *del* und *mod* formulierte Operation ν auf der Instanz $f(\mathbf{I_1})$ durchgeführt werden, wobei nicht feststeht, wie eine dazu passende Umsetzung μ auf dem Datenbankschema auszusehen hat. Bei der Reduktion wird die Abbildungsbeschreibung VI durch eine Kodierungsfunktion *enc* in eine Formulierung als Baumtransformation TI überführt. Das Datenbankschema in TI entsteht durch eine Kodierung des Datenbankschemas aus VI als Baumautomat $\mathcal{A}_S$ und legt den Definitionsbereich der Abbildung fest. Dieser Baumautomat dient als Quellschema für einen Automaten mit Ausgabe $\mathcal{T}$, welcher die Abbildung f dadurch beschreibt, dass die aus den Relationen entstandenen Bäume des Datenbankschemas durch die Abbildung $f_{\mathcal{T}}$ den Bäumen im Bildbereich zuordnet werden.

In der gewonnenen Darstellungsform als Baumabbildung ist die Existenz eines Zielschemas immer sichergestellt. Liegt die durch eine kodierte Änderungsoperation ν erreichte Anwendungsinstanz $\nu(f(\mathbf{I_1}))$ im Zielschema der Abbildung, ist die Operation seiteneffektfrei. Liegt sie außerhalb, ist ihr keine Datenbankinstanz zugeordnet und die Durchführung wird abgelehnt. Ergibt sich aus der Kodierung von f, dass $f_{\mathcal{T}}$ eine informationserhaltende Sicht beschreibt – ist also jeder Instanz des Definitionsbereichs genau eine Instanz des Bildbereichs zugeordnet und umgekehrt –, dann ist eine seiteneffektfreie Operation ν auch eindeutig und somit semantisch korrekt. Der inverse Baumautomat legt $f_{\mathcal{T}^{-1}}$ fest und berechnet dann die Datenbankinstanz $\mu(\mathbf{I_1})$, die sich als Ergebnis der Umsetzung ergibt. Abb. 5.7 zeigt eine als Baumautomat kodierte Sicht und zwei Änderungsoperationen ν_1 und ν_2, wobei die Operation ν_1 durchführbar ist und ν_2 nicht ausgeführt werden kann, da das Ergebnis von ν_1 innerhalb des Zielschemas liegt und das von ν_2 nicht.

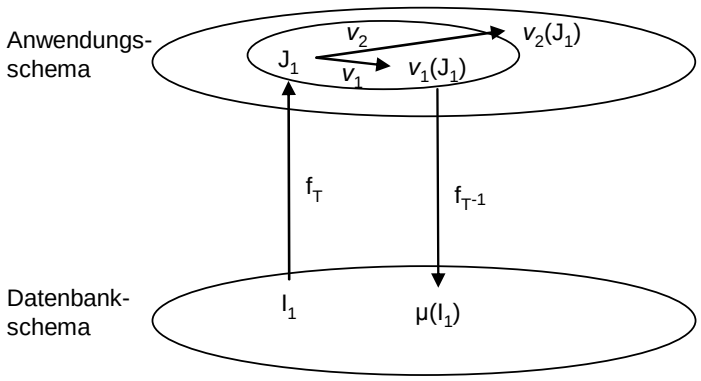


Abbildung 5.7: *Bestimmung semantisch korrekter Operationen*

Gelingt die Kodierung einer durch die SPC-Algebra formulierten Sicht f als Baumautomat mit Ausgabe und ist die aus der Kodierung resultierende Abbildung $f_{\mathcal{T}}$ eine informationserhaltende Sicht, dann bedeutet dies, dass bei beliebigen Sichten, die mit Hilfe der SPC-Algebra formuliert werden, für jede Anwendungsoperation bestimmt werden kann, ob sie ausführbar ist, und dass, wenn sie ausführbar ist, es eine eindeutige Umsetzung gibt. Dies gilt für alle Sichten über allen relationalen Schemata. Die Aufgabe der nächsten Abschnitte ist, die Existenz einer Kodierung nachzuweisen, mit der Abbildungen der SPC-Algebra eine informationserhaltende Sicht beschreiben.

5.4.1 Relationen als Baum

Im ersten Schritt der Reduktion muss die Funktion *enc* für relationale Instanzen festgelegt werden. Ein Kodierungsverfahren für eine Menge von Relationen muss gewährleisten, dass zum einen jede Instanz eines relationalen Schemas als Baum kodiert und dass

zum anderen auf Basis dieser Kodierung eine Abbildung definiert werden kann. Da die Operatoren der relationalen Algebra letztendlich bei einer Abbildung auf einzelne Attributwerte zugreifen, müssen sich diese Attributwerte folglich in der Baumdarstellung als Annotationen wieder finden. Bei einer Abbildungsdefinition mit Hilfe eines Baumautomaten kann dann entsprechend ein Zugriff auf diese einzelnen Werte innerhalb eines Baums erfolgen.

Das gewählte Kodierungsverfahren folgt der impliziten Schachtelungsstruktur einer relationalen Instanz. Eine Instanz $\mathbf{I}$ eines relationalen Schemas besteht aus einer Menge von Relationen, wobei jede Relation $\mathbf{I}(R_i)$ eine Menge von Tupeln enthält. Die Relationen R_i sind jeweils eindeutig über den Bezeichner des zugehörigen Relationstyps adressiert, so dass bei einer Kodierung einer Instanz auf oberster Ebene des entstehenden Baums diese Relationsbezeichner R_i angegeben werden:

$$\{\ R_1 : v_1,\ \ldots,\ R_n : v_n\ \}$$

Ein Teilwert v_i, der einer Annotation R_i folgt, entspricht dann der Kodierung der zum Typ R_i gehörenden Relation. Somit sind die kodierten Relationen v_i über die Annotationen der jeweils zugehörigen Relationsbezeichner im Baum adressierbar.

Im nächsten Schritt muss die Kodierung für einzelne Relationen festgelegt werden. Die Tupel einer Relation werden nach den Werten $a_1, \ldots, a_n$ des ersten Attributs A_1 gruppiert und diese Werte bilden die Annotationen der obersten Ebene eines der Teilwerte v_i:

$$v_i = \{\ a_1 : w_1,\ \ldots,\ a_n : w_n\ \}$$

Hinter jeder Annotation a_i dieser Ebene folgt ein Teilwert w_i, der die Werte der restlichen Attribute $A_2, \ldots, A_n$ derjenigen Tupel beschreibt, welche a_i als Wert beim ersten Attribut besitzen. Entsprechend dem ersten Attribut werden die Werte für die weiteren Attribute gruppiert und die Ebenen des Baums auf Grundlage einer Relation aufgebaut.

Ist ein relationales Schema $\mathbf{R}$ mit der Instanz $\mathbf{I}$ gegeben, so ist die Kodierungsfunktion $enc(\mathbf{I})$ rekursiv definiert:

- Für eine Schemainstanz $\mathbf{I} \in dom(\mathbf{R})$ gilt
  $enc(\mathbf{I}) = \{\ R_1 :\ enc(\mathbf{I}(R_1))\ \ , \ldots, R_n :\ \ enc(\mathbf{I}(R_n))\ \}$
  mit den Relationstypen $R_1, \ldots, R_n \in \mathbf{R}$.

- Für eine Relation I über den Attributen $A_1,\ldots,A_n$ gilt
  $enc(I) = \{\ a_1 :\ enc(\pi_{A_2,\ldots,A_n}(\sigma_{A_1=a_1}(I)))\ , \ldots, a_m :\ enc(\pi_{A_2,\ldots,A_n}(\sigma_{A_1=a_m}(I)))\ \}$
  mit $\{a_1, \ldots, a_m\} = \pi_{A_1}(I)$.

  Für eine Relation I mit einem Attribut A_1 gilt
  $enc(I) = \{\ a_1, \ldots, a_m\}$
  mit $\{a_1, \ldots, a_m\} = \pi_{A_1}(I)$.

Die bei einem Relationstyp verwendeten Attributbezeichner A_i tauchen in der Baumdarstellung nicht mehr auf, da von einer impliziten Ordnung auf den Attributen ausgegangen wird. Auf welcher Ebene sich die Werte welches Attributs befinden, wird durch die Schicht innerhalb eines Baums dargestellt (Abb. 5.8). Die Annotationen auf der obersten Schicht repräsentieren die Relationsbezeichner, die Annotationen auf der darunter folgenden Schicht stehen für die Werte der ersten Attribute von Relationen, die Annotationen auf der dritten Ebene stellen die Werte der zweiten Attribute dar usw.

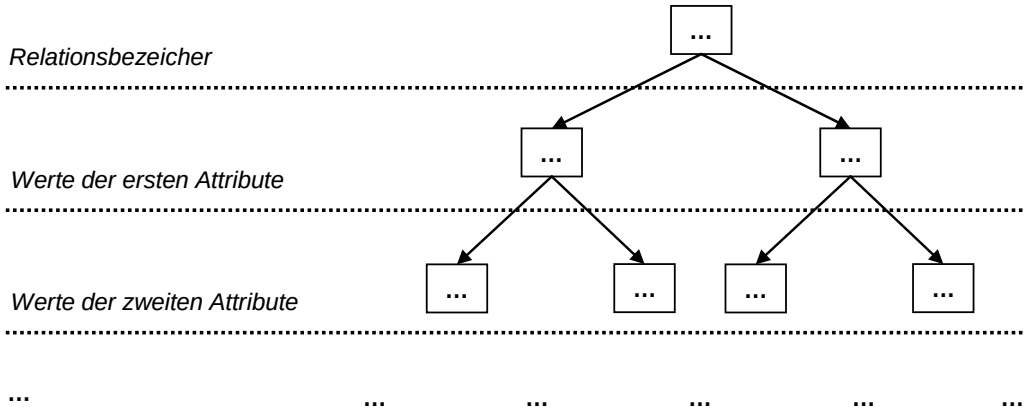


Abbildung 5.8: *Relationen als Baum*

Beispiel 5.2 (*Relationen als Baum*)
Ist ein relationales Schema **R** mit den zwei Relationstypen

R [A, B, C] mit $dom(A) = dom(B) = dom(C) = \{a, b\}$ und
S [D] mit $dom(D) = \{c, d\}$

gegeben, so sind die beiden Relationen in Abb. 5.9 eine Beispielinstanz **I** für dieses Schema.

R

A	B	C
a	a	a
a	b	a
a	b	b
b	a	b

S

D
c
d

Abbildung 5.9: *Beispielrelationen R und S*

Wendet man die Kodierungsfunktion $enc(\mathbf{I})$ auf diese Instanz an, entsteht im ersten Schritt

$$\{\ R : enc(\mathbf{I}(R\ [\ A,\ B,\ C\ ]\ )),\ S : enc(\mathbf{I}(S\ [\ D\ ]\ ))\ \}$$

als die oberste Ebene des Baums, welche die beiden vorkommenden Relationsbezeichner als Annotationen enthält. Wird im nächsten Schritt die Kodierungsfunktion auf den beiden Relationen $enc(\mathbf{I}(R\ [\ A,\ B,\ C\ ]\ ))$ bzw. $enc(\mathbf{I}(S\ [\ D\ ]\ ))$ berechnet, werden die Ebenen für die Werte der Attribute A bzw. D ergänzt:

$$\{\ R : \{\ a : enc(\pi_{B,C}(\sigma_{A=a}(\mathbf{I}(R)))),\ b : enc(\pi_{B,C}(\sigma_{A=b}(\mathbf{I}(R))))\ \}\ ,\ S : \{\ c,\ d\ \}\ \}$$

wobei $\{a, b\} = \pi_A(\mathbf{I}(R))$ und $\{c, d\} = \pi_D(\mathbf{I}(S))$. Wird die Kodierung der Werte der Attribute B und C ebenfalls berechnet, entsteht als Ergebnis von $enc(\mathbf{I})$ der Baum in Abb. 5.10. Auf der obersten Ebene finden sich die Bezeichner der zum Schema gehörenden Relationstypen wieder und in dem jeweils durch sie adressierten Unterbaum die Tupel der zugehörigen Relation. Der Teilbaum unter R beschreibt auf der obersten Ebene die vorhandenen Werte beim Attribut A. Auf den direkt folgenden Ebenen darunter sind die Werte für B und auf den Blattebenen die Werte für C. Werden die Annotationssymbole im Baum von oben nach unten gelesen, entsprechen sie den Attributwerten bei den Tupeln. Somit stellt sich ein Tupel einer Relation als ein Pfad von Annotationen innerhalb des Baums dar.

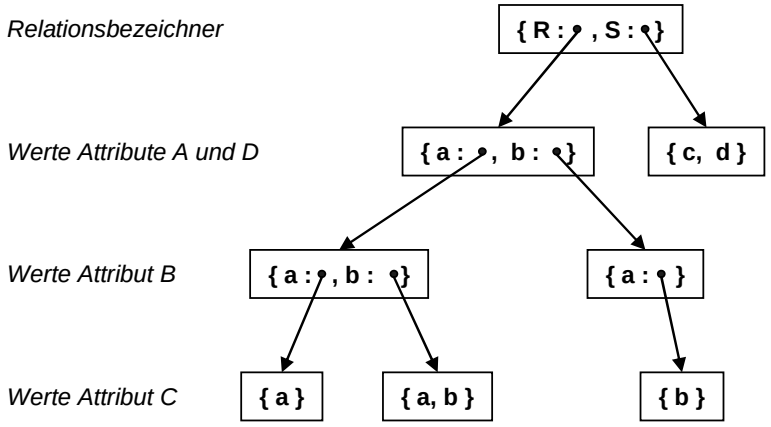


Abbildung 5.10: *Kodierung der Beispielrelationen aus Abb. 5.9 als Baum*

□

Das gewählte Kodierungsverfahren erlaubt es, jede beliebige Instanz eines relationalen Schemas als Baum darzustellen, da für jede Relation die Kodierung ausgeführt werden kann. Da die Annotationen einer Ebene eindeutig sein müssen, ist zudem gewährleistet, dass aus einem Baum wieder eindeutig die Relationen, aus denen er entstanden ist, hergeleitet werden können.

5.4.2 Relationstypen als Baumautomat

Passend zur Darstellung von Relationen als Bäume, gilt es die Kodierung für relationale Schemata als Baumautomaten ohne Ausgabe festzulegen. Bei der Definition der Kodierung eines Schemas muss zum einen gewährleistet sein, dass die Kodierung für alle relationalen Schemata definiert ist, und zum anderen sichergestellt sein, dass wenn eine Relationenmenge Instanz eines relationalen Schemas ist, dann die Kodierung dieser Relationen vom für das Schema erzeugten Baumautomaten akzeptiert wird. Schließlich muss jeder erzeugte Automat kontextbezogen deterministisch sein, um als Quellschema eines Automaten mit Ausgabe dienen zu können, der dann eine Abbildung beschreibt.

Schemata ohne Konsistenzbedingungen

Die Definition der Kodierung eines Schemas ohne Konsistenzbedingungen orientiert sich an der Kodierung von Relationen als Baum. Der Automat $\mathcal{A}_S$ für ein Schema $\mathbf{R}$ mit den Relationstypen $R_1, \ldots, R_n$ muss einen Finalzustand q_f besitzen, in den die Kodierungen aller Instanzen des Schemas abgeleitet werden. Nach den Kodierungsvorschriften für Instanzen enthält jede Instanz eines Schemas auf der obersten Ebene die Bezeichner der im Schema vorkommenden Relationstypen. Folglich muss es im Automaten eine Regel geben, welche in den Finalzustand mündet und diese Bezeichner als Annotationen zulässt:

$$\{\ R_1 : q_{R_1;A_1},\ \ldots,\ R_n : q_{R_n;A_1}\ \} \longrightarrow q_f$$

Im zu generierenden Automaten gibt es somit pro Relationstyp R_i einen Zustand $q_{R_i;A_1}$, den die kodierten Relationen $enc(\mathbf{I}(R_i))$ dieses Typs erreichen:

$$enc(\mathbf{I}(R_i)) \xrightarrow{*}_{\mathcal{A}_S} q_{R_i;A_1}$$

Da ein relationales Schema auch leere Relationen erlaubt, muss es pro Zustand $q_{R_i;A_1}$ eine Übergangsregel $\{\ \} \longrightarrow q_{R_i;A_1}$ geben, welche die leere Relation modelliert.

Eine kodierte Relation $enc(I)$ enthält auf der obersten Ebene die Annotationen für die Werte $\{a_1, \ldots, a_n\} = \pi_{A_1}(I)$ des ersten Attributs A_1 der Relation I. Da auf dieser Ebene beliebige Teilmengen der Domäne von A_1 vorkommen können, müssen im Automaten Regeln der Form

$$\{\ a_1 : q_{R_i;A_2},\ \ldots,\ a_n : q_{R_i;A_2}\ \} \longrightarrow q_{R_i;A_1}$$

vorhanden sein, wobei je eine Regel für jede nicht leere Teilmenge $\{a_1, \ldots, a_n\}$ der Domäne $dom(A_1)$ notwendig ist und diese Regeln jeweils in den Zustand $q_{R_i;A_1}$ münden. In den Zustand $q_{R_i;A_2}$ werden für alle k die als $enc(\pi_{A_2,\ldots,A_n}(\sigma_{A_1=a_k}(\mathbf{I}(R_i))))$ kodierten Relationen abgeleitet:

$$enc(\pi_{A_2,\ldots,A_n}(\sigma_{A_1=a_k}(\mathbf{I}(R_i)))) \xrightarrow{*}_{\mathcal{A}_S} q_{R_i;A_2}$$

Verfolgt man den rekursiven Aufbau der Bäume weiter, sind für jedes Attribut A_j eines Relationstyps R_i im Automaten ein passender Zustand $q_{R_i;A_j}$ und Regeln der Form $\{\ a_1 : q_{R_i;A_{j+1}},\ \ldots,\ a_n : q_{R_i;A_{j+1}}\ \} \longrightarrow q_{R_i;A_j}$ für alle Teilmengen $\{a_1, \ldots, a_n\}$ der Domäne von A_j vorhanden.

Die Berechnungsvorschrift für die Kodierung eines Schemas folgt der Berechnungsvorschrift für die Kodierung einer Schemainstanz und ist ebenfalls rekursiv aufgebaut. Die Regeln geben jeweils an, welche Kombinationen an Annotationen auf welcher Ebene eines Werts erlaubt sind. Insgesamt ergibt sich für ein relationales Schema $\mathbf{R}$ die Kodierung als Baumautomat ohne Ausgabe $enc(\mathbf{R}) = (Q, \Sigma, q_f, \Delta)$ durch die folgenden rekursiven Vorschriften, indem die Automaten für die einzelnen Relationstypen zu einem Schemaautomaten kombiniert werden:

- Ist $\mathbf{R} = \{\ R_1[U_1], \ldots, R_n[U_n]\ \}$ ein Schema und $enc(U_i) = (Q_{U_i}, \Sigma_{U_i}, q_{U_i}, \Delta_{U_i})$ für eine Attributmenge U_i, dann $enc(\mathbf{R}) = (Q, \Sigma, q_f, \Delta)$ mit

  $$\begin{aligned} Q &= \textstyle\bigcup_i Q_{U_i} \cup \{q_f\} \\ \Sigma &= \textstyle\bigcup_i \Sigma_{U_i} \cup \{R_1, \ldots, R_n\} \\ \Delta &= \textstyle\bigcup_i \Delta_{U_i} \cup \{\{\ R_1 : q_{U_1}, \ldots, R_n : q_{U_n}\} \longrightarrow q_f\} \\ &\quad \cup \{\{\ \} \longrightarrow q_{U_i} \mid i = 1, \ldots, n\} \end{aligned}$$

  wobei $q_f \notin Q_{U_i}$ für alle i und $Q_{U_i} \cap Q_{U_j} = \emptyset$ für $i \neq j$.

  Sind die $enc(U_i)$ die Automaten, welche pro Relationstyp R_i die kodierten Relationen akzeptieren, dann werden deren Regeln in den Automaten für das Schema übernommen, jeweils eine Regeln $\{\ \} \longrightarrow q_{U_i}$ für die leere Relation ergänzt und die Schlussregel mit den Relationsbezeichnern $\{\ R_1 : q_{U_1},\ \ldots,\ R_n : q_{U_n}\ \} \longrightarrow q_f$ eingefügt.

- Ist $U = [\ A_1,\ A_2,\ \ldots,\ A_n\ ]$ eine Menge von Attributen und $enc([\ A_2,\ \ldots,\ A_n\ ]) = (Q_{A_2}, \Sigma_{A_2}, q_{A_2}, \Delta_{A_2})$, dann $enc(U) = (Q_{A_1}, \Sigma_{A_1}, q_{A_1}, \Delta_{A_1})$ mit

  $$\begin{aligned} Q_{A_1} &= Q_{A_2} \cup \{q_{A_1}\} \\ \Sigma_{A_1} &= \Sigma_{A_2} \cup dom(A_1) \\ \Delta_{A_1} &= \Delta_{A_2} \cup \{\{\ a_1 : q_{A_2}, \ldots, a_n : q_{A_2}\} \longrightarrow q_{A_1} \mid \{a_1, \ldots, a_n\} \in I\!P^{fin}(dom(A_1)) \wedge \\ &\quad n \geq 1\} \end{aligned}$$

  wobei $q_{A_1} \notin Q_{A_2}$.

  Akzeptiert der Automat $enc([\ A_2,\ \ldots,\ A_n\ ])$ die kodierten Relationen über den Attributen A_2 bis A_n, dann werden dessen Regeln in den Automaten für die Attribute $A_1, \ldots, A_n$ übernommen und Regeln für das Attribut A_1 ergänzt. Um das Attribut zu kodieren, wird für jede endliche, nicht leere Teilmenge seiner Domäne $\{a_1, \ldots, a_n\}$ eine Regel eingefügt.

  Ist $U = [\ A_1\ ]$ eine Menge mit einem Attribut, dann $enc(U) = (Q_{A_1}, \Sigma_{A_1}, q_{A_1}, \Delta_{A_1})$ mit

$$
\begin{aligned}
Q_{A_1} &= \{q_{A_1}\} \\
\Sigma_{A_1} &= dom(A_1) \\
\Delta_{A_1} &= \{\{\ a_1, \ldots, a_n\} \ \longrightarrow q_{A_1} \mid \{a_1, \ldots, a_n\} \in I\!P^{fin}(dom(A_1)) \wedge n \geq 1\}
\end{aligned}
$$

Ein Automat für ein einzelnes Attribut enthält für jede endliche, nicht leere Teilmenge seiner Domäne $\{a_1, \ldots, a_n\}$ eine Regel.

Das Kodierungsverfahren für ein Schema impliziert wie das Verfahren für eine Instanz eine Ordnung auf den Attributen eines Relationstyps. Je nachdem in welcher Reihenfolge die Attributmenge abgearbeitet wird, entsteht ein anderer Automat. Dies hat Ähnlichkeit mit der Interpretation des relationalen Modells, bei der Attribute nicht über Bezeichner, sondern über die Nummer ihrer Position innerhalb einer Relation angesprochen werden. Ein kodiertes Attribut in einem Baum wird nur durch die Nummer der Schicht identifiziert, in der es vorkommt. Da im Automaten jeder Zustand einer Schicht zugeordnet werden kann, entsprechen – übertragen betrachtet – ein oder eventuell auch mehrere Zustände dann einem Attribut.

Bei der Kodierung in einen Automaten wird eine Einschränkung vorgenommen, da davon ausgegangen wird, dass die Domänen der Attribute endlich sind. Ansonsten wären unendlich viele Regeln notwendig, um eine Domäne zu kodieren, so dass kein gültiger Automat entstehen würde. Da im relationalen Modell nur endliche Relationen betrachtet werden, ist diese Einschränkung hinnehmbar, da die Domänen bei Bedarf als „genügend groß“ gewählt werden können.

Das Zusammensetzen von mehreren kleinen zu einem großen Automaten für ein Schema, wie es innerhalb der Kodierung verwendet wird, erzeugt wieder einen gültigen Automaten. Da ein Automat aus einer Menge von Zuständen und Regeln besteht und keine Anforderungen an den Aufbau einer zu akzeptierende Wertemenge gestellt werden, können Zustände und Regeln von Automaten durch eine Vereinigung entsprechend kombiniert werden. Die Zustandsmengen der zu kombinierenden Automaten sind beim Kodierungsverfahren disjunkt.

Beispiel 5.3 (*Relationstypen als Baumautomat*)
Betrachtet man das relationale Schema $\mathbf{R}$ aus Beispiel 5.2, so kann die Kodierung $enc(\mathbf{R})$ dieses Schemas als Baumautomat rekursiv berechnet werden. Es ergibt sich nach dem ersten Schritt des Verfahrens $enc(\mathbf{R}) = (Q, \Sigma, q_f, \Delta)$ als

$$
\begin{aligned}
Q &= Q_A \cup Q_D \cup \{q_f\} \\
\Sigma &= \Sigma_A \cup \Sigma_D \cup \{R, S\} \\
\Delta &= \Delta_A \cup \Delta_D \\
&\qquad \cup\{\ \{\ R : q_A,\ S : q_D\ \} \ \longrightarrow q_f\ \} \\
&\qquad \cup\{\ \{\ \}\ \longrightarrow q_A,\ \{\ \}\ \longrightarrow q_D\ \}
\end{aligned}
$$

mit dem Automaten $enc([\ A,\ B,\ C\ ]) = (Q_A, \Sigma_A, q_A, \Delta_A)$ für den Relationstyp R und dem Automaten $enc([\ D\ ]) = (Q_D, \Sigma_D, q_D, \Delta_D)$ für den Relationstyp S.

Durch den zweiten Schritt der Kodierung ergibt sich $enc([\ A, B, C\ ]) = (Q_A, \Sigma_A, q_A, \Delta_A)$ als

$$
\begin{aligned}
Q_A &= Q_B \cup \{q_A\} \\
\Sigma_A &= \Sigma_B \cup \{a, b\} \\
\Delta_A &= \Delta_B \\
&\quad \cup \{\ \{\ a : q_B\ \} \longrightarrow q_A,\ \{\ b : q_B\ \} \longrightarrow q_A,\ \{\ a : q_B,\ b : q_B\ \} \longrightarrow q_A\ \}
\end{aligned}
$$

mit $enc([\ B,\ C\ ]) = (Q_B, \Sigma_B, q_B, \Delta_B)$ als

$$
\begin{aligned}
Q_B &= Q_C \cup \{q_B\} \\
\Sigma_B &= \Sigma_C \cup \{a, b\} \\
\Delta_B &= \Delta_C \\
&\quad \cup \{\ \{\ a : q_C\ \} \longrightarrow q_B,\ \{\ b : q_C\ \} \longrightarrow q_B,\ \{\ a : q_C,\ b : q_C\ \} \longrightarrow q_B\ \}
\end{aligned}
$$

und mit $enc([\ C\ ]) = (Q_C, \Sigma_C, q_C, \Delta_C)$ als

$$
\begin{aligned}
Q_C &= \{q_C\} \\
\Sigma_C &= \{a, b\} \\
\Delta_C &= \{\ \{\ a\ \} \longrightarrow q_C,\ \{\ b\ \} \longrightarrow q_C,\ \{\ a,\ b\ \} \longrightarrow q_C\ \}
\end{aligned}
$$

Für den zweiten Relationstyp ergibt sich $enc([\ D\ ]) = (Q_D, \Sigma_D, q_D, \Delta_D)$ mit

$$
\begin{aligned}
Q_D &= \{q_D\} \\
\Sigma_D &= \{c, d\} \\
\Delta_D &= \{\ \{\ c\ \} \longrightarrow q_D,\ \{\ d\ \} \longrightarrow q_D,\ \{\ c,\ d\ \} \longrightarrow q_D\ \}
\end{aligned}
$$

Werden die Zustandsmengen und die Regelmengen vereinigt, entsteht insgesamt ein Automat $enc(\mathbf{R}) = (\{q_f, q_A, q_B, q_C, q_D\},\ \{a, b, c, d\},\ q_f,\ \Delta)$ mit den Übergangsregeln:

$$
\begin{aligned}
\{\ R : q_A,\ S : q_D\ \} &\longrightarrow q_f \\
\{\ \} &\longrightarrow q_A \\
\{\ a : q_B\ \} &\longrightarrow q_A \\
\{\ b : q_B\ \} &\longrightarrow q_A \\
\{\ a : q_B,\ b : q_B\ \} &\longrightarrow q_A \\
\{\ a : q_C\ \} &\longrightarrow q_B \\
\{\ b : q_C\ \} &\longrightarrow q_B \\
\{\ a : q_C,\ b : q_C\ \} &\longrightarrow q_B \\
\{\ a\ \} &\longrightarrow q_C \\
\{\ b\ \} &\longrightarrow q_C \\
\{\ a,\ b\ \} &\longrightarrow q_C \\
\{\ \} &\longrightarrow q_D \\
\{\ c\ \} &\longrightarrow q_D \\
\{\ d\ \} &\longrightarrow q_D \\
\{\ c,\ d\ \} &\longrightarrow q_D
\end{aligned}
$$

Die Schlussregel des Automaten, die in den Finalzustand q_f mündet, beschreibt die im Schema vorgegebenen Relationstypen. Da diese bei einer gültigen Instanz des Schemas immer vorhanden sein müssen, gibt es nur eine Regel mit q_f auf der rechten Seite.

Als nächstes enthält die Regelmenge eine Übergangsregel zur Beschreibung einer leeren Relation für den Typ R. Hierzu wird der leere Wert $\{\ \}$ in den zum ersten Attribut A von R gehörenden Zustand q_A abgeleitet. Somit akzeptiert der Automat einen Wert der Form $\{\ R : \{\ \}\ ,\ S : \ldots\}$ mit einer leeren Relation R.

Die folgenden drei angegebenen Regeln beschreiben die Domäne des Attributs A mit den Werten $\{a, b\}$. Da jede mögliche Kombination von Werten aus dieser Menge erlaubt ist, existieren entsprechend drei Regeln, die je eine der Möglichkeiten beschreiben. Eine Verknüpfung zum nachfolgenden Attribut B geschieht durch die Angabe des diesem Attribut zugewiesenen Zustands q_B auf den linken Seiten der Regeln. Bei den Regeln für das letzte Attribut C fehlen Zustandsangaben auf den linken Seiten, so dass diese Werte die Blätter innerhalb eines Baums bilden.

Mit den angegebenen Regeln wird der Baum aus Abb. 5.10 vom Automaten $enc(\mathbf{R})$ akzeptiert. Also stellt dieser Wert eine gültige Instanz des kodierten Schemas dar. □

Es gilt zu überprüfen, ob die gewählte Kodierungsfunktion den Anforderungen genügt, d.h. dass für jedes relationale Schema ohne Konsistenzbedingungen ein kontextbezogen deterministischer Automat erzeugt wird, der dann einen kodierten Wert als Instanz akzeptiert, wenn der Wert Instanz des relationalen Schemas ist und umgekehrt.

Jedes relationale Schema ohne Konsistenzbedingungen kann kodiert werden, da für alle entsprechenden Schemata und Relationstypen die Kodierungsfunktion definiert ist. Es gilt

$$\mathbf{I} \in dom(\mathbf{R}) \Leftrightarrow enc(\mathbf{I}) \in dom(enc(\mathbf{R}))$$

für alle Schemata $\mathbf{R}$ ohne Konsistenzbedingungen und alle ihre Instanzen $\mathbf{I}$, da das Kodierungsverfahren für Schemata dem Kodierungsverfahren für Instanzen folgt. Eine kodierte Instanz eines Schemas enthält auf oberster Ebene immer die Bezeichner der vorhandenen Relationstypen und ein für das Schema erzeugter Automat enthält eine entsprechende Regel, die in den Finalzustand mündet. Die Teilbäume für die einzelnen Relationen enthalten pro Schicht die Werte aus der Domäne eines Attributs des Relationstyps. Entsprechend besitzt der Automat pro Attribut einen Zustand, in den alle möglichen Kombinationen von Werten der Attributdomäne ableitbar sind, und der einer entsprechenden Schicht zugeordnet werden kann.

Das Kodierungsverfahren erzeugt für jedes relationale Schema ohne Konsistenzbedingungen einen kontextbezogen deterministischen Automaten. Jeder Schritt des Kodierungsverfahrens führt einen neuen Zustand q ein, der in keinem anderen Schritt ebenfalls auf der rechten Seite einer Regel verwendet wird. Somit wird in jedem Schritt die komplette Menge Δ_q festgelegt, wenn q der in diesem Schritt eingeführte Zustand ist. Δ_{q_f} enthält nur eine Regel, so dass Δ_{q_f} eine Partitionierung von $dom(q_f)$ mit einer Partition beschreibt. Für einen Zustand q_{A_i} für ein beliebiges Attribut A_i eines Relationstyps gilt, dass in einem Schritt Regeln der Form

$$\{\ a_1 : q_{A_{i+1}},\ \ldots,\ a_n : q_{A_{i+1}}\ \} \quad \longrightarrow \quad q_{A_i}$$

angelegt werden. Da für jede Teilmenge $\{a_1, \ldots, a_n\}$ aus der Domäne des Attributs A_i nur eine Regel angelegt wird, unterscheiden sich die Annotationen auf den linken Seiten zweier Regeln aus $\Delta_{q_{A_i}}$. Bei den Zuständen für die ersten Attribute einer Relation kommen durch den ersten Schritt des Kodierungsverfahrens noch je eine Regel mit { } hinzu. Die hier angegebene leere Annotationsmenge unterscheidet sich ebenfalls von den Annotationsmengen der anderen Regeln aus $\Delta_{q_{A_i}}$. Somit beschreibt jedes $\Delta_{q_{A_i}}$ eine Partitionierung von je $dom(q_{A_i})$. Ein erzeugter Automat ist folglich kontextbezogen deterministisch.

Schemata mit Konsistenzbedingungen

Das bisherige Kodierungsverfahren für relationale Schemata berücksichtigt noch keine Konsistenzbedingungen in Form von funktionalen Abhängigkeiten oder Inklusionsabhängigkeiten, die bei einem relationalen Schema angegeben sein können. Beide Arten von Konsistenzbedingungen können auf Ebene einer Schemainstanz ausgewertet werden, indem überprüft wird, ob eine Instanz die Abhängigkeiten einhält oder nicht. Abhängigkeiten haben hierbei eine Filterfunktion, da ein Schema mit Abhängigkeiten weniger Instanzen besitzt als dasselbe Schema ohne Abhängigkeiten. Bei einer funktionalen Abhängigkeit $R : X \rightarrow Y$ sind nur solche Instanzen erlaubt, bei denen unter einem Wert für X nur ein Werte für Y bei einer Relation R auftritt. Fehlt die Abhängigkeit, sind mehrere Werte unter Y bei festem Wert unter X zugelassen und somit auch mehr Relationen vorhanden.

Ist ein Schema $\mathbf{R}$ und eine Menge von Konsistenzbedingungen $\mathcal{D}$ gegeben, erfolgt eine Kodierung $enc(\mathbf{R},\mathcal{D})$ über die Menge der möglichen Instanzen des relationalen Schemas. Es ergibt sich ein Automat $enc(\mathbf{R},\mathcal{D})=(\{q_f\}, \Sigma, q_f, \Delta)$ mit

$$\Delta = \{v \longrightarrow q_f \mid v = enc(\mathbf{I}) \wedge \mathbf{I} \models \mathcal{D} \wedge \mathbf{I} \in dom(\mathbf{R})\}$$

und $\Sigma = \{ann(v) \mid v = enc(\mathbf{I})\}$, wobei $ann(v)$ die Menge der in v vorkommenden Annotationen bezeichnet. Jedes v stellt hierbei die Kodierung für eine komplette Schemainstanz dar und ergibt sich somit als geschachtelter Wert. Da die Menge der Schemainstanzen als endlich angenommen wird, wird ein Automat mit endlich vielen Übergangsregeln angelegt. Offensichtlich kodiert der entstandene Automat das Schema mit den zugehörigen Konsistenzbedingungen.

Das Kodierungsverfahren besteht aus einer reinen Aufzählung aller erlaubten Instanzen, wobei Regeln mit tiefer geschachtelten Werten auf der Eingabeseite verwendet werden. Aus diesen Regeln können Regeln mit flachen Werten erzeugt werden, indem die Umformungen aus Abschnitt 4.3 verwendet werden.

Beispiel 5.4 (*Kodierung von Schemata mit Konsistenzbedingungen*)
Sei ein Schema $\mathbf{R}$ mit einem Relationstyp R [A, B] mit $dom(A) = dom(B) = \{a, b\}$ und einer funktionalen Abhängigkeit $\mathcal{D} = \{R : A \rightarrow B\}$ gegeben. Die Kodierung liefert $enc(\mathbf{R},\mathcal{D}) = (\{q_f\}, \{R, a, b\}, q_f, \Delta)$ mit $\Delta = \{$

$$
\begin{array}{lcl}
\{\ R : \{\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ a : \{\ a\ \}\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ a : \{\ b\ \}\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ b : \{\ a\ \}\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ b : \{\ b\ \}\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ a : \{\ a\ \}\ , b : \{\ a\ \}\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ a : \{\ a\ \}\ , b : \{\ b\ \}\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ a : \{\ b\ \}\ , b : \{\ a\ \}\ \}\ \} & \longrightarrow & q_f \\
\{\ R : \{\ a : \{\ b\ \}\ , b : \{\ b\ \}\ \}\ \} & \longrightarrow & q_f
\end{array}
$$

}. Ist in einer Ebene des Attributs A eine Annotation gegeben, so ist auf der direkt darunter nachfolgenden Ebene des Attributs B nur eine Annotation vorhanden. Somit bestimmt jeder Wert für A den Wert für B eindeutig und die funktionale Abhängigkeit wird erfüllt.

Werden die Regeln mit den geschachtelten Werten durch die Umformungen nach Abschnitt 4.3 in flache Regeln aufgeteilt und Zustände mit selbem Wertebereich zusammengefasst, entsteht ein zu $enc(\mathbf{R},\mathcal{D})$ äquivalenter Automat $\mathcal{A} = (\{q_f, q_A, q_B\}, \Sigma, q_f, \Delta_1)$ mit den Regeln $\Delta_1 = \{$

$$
\begin{array}{lcl}
\{\ R : q_A\ \} & \longrightarrow & q_f \\
\{\ \} & \longrightarrow & q_A \\
\{\ a : q_B\ \} & \longrightarrow & q_A \\
\{\ b : q_B\ \} & \longrightarrow & q_A \\
\{\ a : q_B, b : q_B\ \} & \longrightarrow & q_A \\
\{\ a\ \} & \longrightarrow & q_B \\
\{\ b\ \} & \longrightarrow & q_B
\end{array}
$$

}. Im Vergleich zu einem für ein Schema ohne Konsistenzbedingungen erzeugten Automaten fehlt hier die Übergangsregel $\{\ a, b\ \} \longrightarrow q_B$, welche mehr als eine Annotation auf der Blattebene erlauben würde. Durch das Vorhandensein von Konsistenzbedingungen fallen also im Vergleich zu einem Schema ohne Konsistenzbedingungen Regeln weg. □

Jedes Schema mit Konsistenzbedingungen kann mit Hilfe des Verfahrens als Baumautomat kodiert werden. Der Automat akzeptiert nur die kodierten Werte, welche gültige Instanzen des relationalen Schemas unter Beachtung der Abhängigkeiten sind. Im Grunde lassen sich beliebige Konsistenzbedingungen modellieren, die sogar über die im relationalen Modell vorhandenen Abhängigkeiten hinausgehen können, da ein Automat so aufgebaut sein kann, dass jede Instanz einzeln akzeptiert wird. Sind ein Schema $\mathbf{R}$ und Konsistenzbedingungen $\mathcal{D}$ in Form von funktionalen oder Inklusionsabhängigkeiten gegeben, gilt

$$\mathbf{I} \in dom(\mathbf{R}) \wedge \mathbf{I} \models \mathcal{D} \Leftrightarrow enc(\mathbf{I}) \in dom(enc(\mathbf{R}, \mathcal{D}))$$

Der durch eine Kodierung entstehende Automat ist kontextbezogen deterministisch, da für jede Instanz des Automaten genau ein Ableitungsweg vorhanden ist. Dieser besteht für jede Instanz aus genau einer der im Automaten vorhandenen Regeln. Somit wird durch die Regelmenge die Menge $dom(q_f)$ partitioniert, wobei jede Partition genau eine Instanz des Automaten enthält.

Sowohl das Kodierungsverfahren für Schemata ohne Konsistenzbedingungen als auch das Verfahren für Schemata mit Konsistenzbedingungen erzeugt Automaten mit einer Vielzahl von Übergangsregeln, da entweder für alle Teilmengen von Attributdomänen jeweils einzelne Regeln angegeben werden oder jede relationale Instanz einzeln kodiert wird. Da die erzeugten Regelmengen aufgrund der Konstruktionsverfahren eine regelmäßige Struktur aufweisen, kann auf syntaktischer Ebene eine Menge von Regeln durch eine höherwertige Regel ersetzt werden, die nicht nur einen Übergang beschreibt, sondern eine Menge möglicher Übergänge zusammenfasst. Hierdurch wird die Zahl der anzugebenden Übergangsregeln verringert. Kapitel 7 zeigt verkürzte Schreibweisen für Regelmengen auf.

Durch die Angabe der beiden Kodierungsvorschriften wird die Zielsetzung dieses Abschnitts erreicht, dass jedes relationale Schema als kontextbezogen deterministischer Baumautomat kodiert werden kann. Diese Aussage schließt Schemata mit Konsistenzbedingungen ein. Die Existenz eines Quellschemas, das für eine Abbildungsdefinition verwendet werden kann, ist erbracht.

5.4.3 Abbildungen als Baumtransformationen

Die Kodierungsfunktion *enc* liefert für ein relationales Schema einen Automaten $\mathcal{A}_S$, der als Quellschema bei der Kodierung einer Sichtendefinition verwendet wird. Bei der Kodierung einer Sicht wird der Automat als Quellschema für einen Automat mit Ausgabe $\mathcal{T}$ benutzt, welcher die Sicht beschreibt. Die Kodierung für eine Sicht muss derart gewählt werden, dass wenn die Sicht eine Datenbankinstanz auf eine Bildinstanz abbildet, dann durch die kodierte Abbildung die kodierte Datenbankinstanz auf die kodierte Bildinstanz abgebildet wird.

Sichten im relationalen Modell werden durch die Operatoren der SPC-Algebra definiert, so dass es bei einer Kodierung für Sichten ausreichend ist, eine Kodierung für jeden der Operatoren der Algebra anzugeben, um aus diesen Operatoren bestehende Sichten kodieren zu können. Verwendet die Sicht mehrere Operatoren hintereinander, werden die Automaten der Operatoren entsprechend hintereinander geschaltet, so dass das Zielschema des einen Automaten das Quellschema des anderen Automaten bildet. Hierbei sind eventuell Umformungen als Zwischenschritte notwendig, um das Zielschema in eine geeignete Form zu bringen, so dass es als Quellschema der folgenden Abbildung verwendet werden kann.

Die Operatoren der SPC-Algebra beschreiben allerdings keine injektiven Abbildungen, da sie mehrere unterschiedliche Datenbankinstanzen auf dieselbe Bildinstanz abbilden. Ist für ein Datenbankschema mit einem Relationstyp $R\ [\ A\ ]$ mit $dom(A) = \{a, b\}$ eine

Sicht als $\sigma_{A=a}(R)$ definiert, so bildet diese Sicht sowohl eine Relation $R_1 = \{\langle\ \mathrm{a}\ \rangle\}$ also auch eine Relation $R_2 = \{\langle\ \mathrm{a}\ \rangle, \langle\ \mathrm{b}\ \rangle\}$ des Datenbankschemas auf die Relation $R_V = \{\langle\ \mathrm{a}\ \rangle\}$ im Anwendungsschema ab. Bei einer naiven Kodierung der Selektion ergibt sich ein Baumautomat mit Ausgabe $\mathcal{T} = (\{q_f, q_A\}, \{R, a, b\}, \{R, a\}, q_f, \Delta)$ mit $\Delta = \{$

$$
\begin{array}{lcl}
\{\ R : q_A(x_1)\ \} & \longrightarrow & q_f\ (\ \{\ R : x_1\ \}\ ) \\
\{\ \} & \longrightarrow & q_A\ (\ \{\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_A\ (\ \{\ a\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_A\ (\ \{\ \}\ ) \\
\{\ a, b\ \} & \longrightarrow & q_A\ (\ \{\ a\ \}\ )
\end{array}
$$

$\}$, der entsprechend den Eingabewerten $\{\ R : \{\ a\ \}\ \}$ und $\{\ R : \{\ a, b\ \}\ \}$ den Ausgabewert $\{\ R : \{\ a\ \}\ \}$ zuweist. Da der Automat nur eine andere Darstellungsform der Sicht ist, übertragen sich die Eigenschaften der Sicht auf die durch den Automaten beschriebene Abbildung. Diese ist ebenfalls nicht injektiv und somit keine informationserhaltende Sicht, da der inverse Automat wegen der zweiten und vierten Regeln nicht kontextbezogen deterministisch ist.

Freiraum, um doch eine informationserhaltende Sicht zu gewinnen, entsteht dadurch, dass die Kodierungsfunktion für die Abbildung innerhalb gewisser Grenzen frei gewählt werden kann, so dass eine informationserhaltende Sicht auf Seiten der Baumabbildungen entsteht und die Abbildungseigenschaften immer noch auf die Sichten übertragbar sind. Die Idee ist, ähnlich wie in [BS81] für eine Abbildung neben dem eigentlichen Ausgabewert ein Komplement zu definieren, welches den Teil der Datenbankinstanz beschreibt, welcher nicht durch die eigentliche Abbildung abgebildet wird. Übertragen auf Bäume bedeutet dies, dass ein Ausgabewert neben den gewöhnlichen Annotationen a_1, $\ldots$, a_n als Ausgabe auch ein Komplement enthält, dass aus mit einem hochgestellten C markierten Annotationen a_1^C, $\ldots$, a_n^C besteht. Hierbei handelt es sich lediglich um eine Markierung von Symbolen und nicht um zwei disjunkte Mengen von Annotationssymbolen, so dass zwei Annotationen a und a^C als identisch angesehen werden, wenn es um die Eindeutigkeit der Symbole auf einer Ebene eines Werts geht. Dass Komplement ist so zu wählen, dass insgesamt eine informationserhaltende Sicht entsteht und dass, wenn die mit C markierten Annotationen beim Ausgabewert gestrichen werden, dann die eigentliche Ausgabe entsteht. Aus einem Wert mit Komplement kann ein Wert ohne Komplement hergeleitet werden, indem die Annotationen des Komplements aus dem Wert entfernt werden. Ein Wert $\{\ a, b^C\ \}$ entspricht $\{\ a\ \}$, ein Wert $\{\ a : v^C\ \}$ entspricht $\{\ a\ \}$ und ein Wert $\{\ a^C : v\ \}$ entspricht v. Gehören mehrere Annotationen einer Ebene $\{\ a^C : v_1, b^C : v_2\ \}$ zum Komplement entspricht dies einem Wert $union(v_1,v_2)$.

Verwendet man bei der Beispielselektion ein Komplement, entsteht ein Automat mit den Regeln

$$
\begin{array}{lcl}
\{\ R : q_A(x_1)\ \} & \longrightarrow & q_f\ (\ \{\ R : x_1\ \}\ ) \\
\{\ \} & \longrightarrow & q_A\ (\ \{\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_A\ (\ \{\ a\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_A\ (\ \{\ b^C\ \}\ ) \\
\{\ a, b\ \} & \longrightarrow & q_A\ (\ \{\ a, b^C\ \}\ )
\end{array}
$$

Das Komplement ist hier so gewählt, dass die Annotationen der Eingabeseite, welche nicht zur eigentlichen Ausgabe gehören, im Komplement landen. Der inverse Automat ist kontextbezogen deterministisch, da sich die Mengen der Annotationen bei Regeln des Zustands q_A alle paarweise unterscheiden. Die Abbildung mit Komplement ist also eine informationserhaltende Sicht. Ein Eingabewert $\{\ R : \{\ a, b\ \}\ \}$ wird auf einen Ausgabewert $\{\ R : \{\ a, b^C\ \}\ \}$ abgebildet. Streicht man die mit C markierten Annotationen, entsteht der eigentliche Ausgabewert der Selektion wieder als $\{\ R : \{\ a\ \}\ \}$.

Selektionen, Projektionen und kartesische Produkte werden im Folgenden jeweils als Abbildungen mit Komplement kodiert.

5.4.3.1 Selektion

Eine Selektion wählt aus einer Menge von Eingabetupeln diejenigen aus, welche die spezifizierte Selektionsbedingung erfüllen. Die Selektionsbedingung ist im einfachsten Fall der Vergleich eines Attributs A_i mit einer Konstanten c. Ein Tupel erscheint in der Ausgabe, wenn es den angegebenen Attributwert besitzt. Da ein Tupel einem Pfad in einem Baum entspricht, bedeutet eine Selektion $\sigma_{A_i=c}$ übertragen auf Bäume, dass aus der Menge der Pfade eines Eingabewerts diejenigen ausgewählt werden, welche auf einer dem Selektionsattribut A_i zugeordneten Ebene eine Annotation c enthalten.

Abb. 5.11 zeigt die Instanz mit den zwei Relationen R und S aus Beispiel 5.2. Wird hier eine Selektion $\sigma_{B=b}(R)$ durchgeführt, bilden diejenigen Pfade den Ausgabewert, welche auf der Ebene des Attributs B eine Annotation b besitzen. Dies sind die beiden Pfade *Raba* und *Rabb*. Diese Pfade finden sich in der Baumdarstellung wieder als alle Teilbäume unterhalb der Ebenen von B, welche die Annotation b auf der B-Schicht erreichen – im Beispiel ist dies der Teilbaum $\{\ a, b\ \}$ auf der C-Schicht – und als Annotationen der Ebenen von A und von den Relationsbezeichnern darüber, bei denen der Teilbaum darunter auf der B-Schicht ein b enthält – dies ist a auf der A-Schicht und R bei den Bezeichnern. Sämtliche übrigen Annotationen der Eingabe werden in der Ausgabe durch die Angabe mit einem hochgestellten C als Komplement gekennzeichnet.

Grundlage einer Abbildungsdefinition bildet ein kontextbezogen deterministischer Automat, wie er durch $enc(\mathbf{R})$ bzw. $enc(\mathbf{R},\mathcal{D})$ entstanden ist. Wie im Beispiel gesehen hängt die Entscheidung, ob eine Annotation zur Ausgabe oder zum Komplement gehört, bei Teilwerte unterhalb der Schicht des Selektionsattributs davon ab, ob die Annotation der Selektionsbedingung bei der weiteren Ableitung erreicht wird, und bei Teilwerten oberhalb der Schicht des Selektionsattributs davon ab, ob in einem bereits abgeleiteten Teilbaum diese Annotation enthalten ist. Bei einem durch die Kodierung entstandenen Automaten kann diese Entscheidung aber noch nicht getroffen werden, da der Werte-

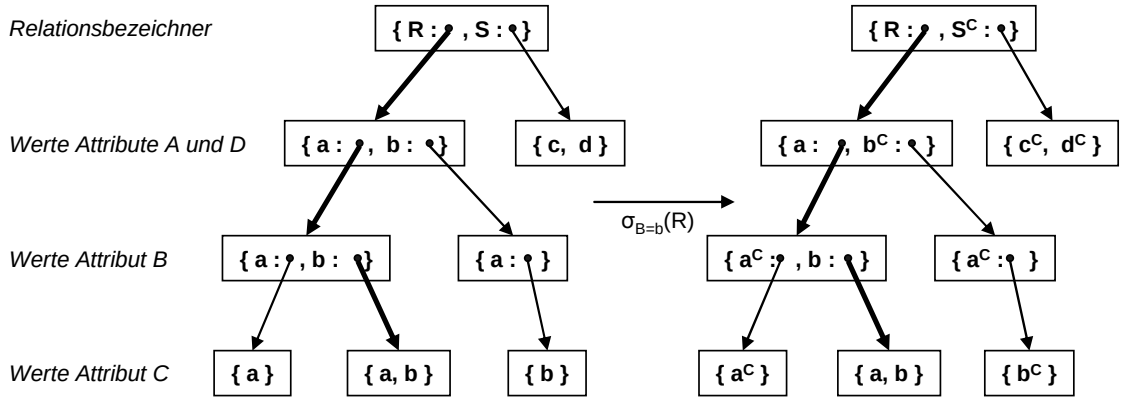


Abbildung 5.11: *Selektion $\sigma_{B=b}(R)$ auf einem Beispielwert*

bereich eines Zustandes q_{A_j} eines Attributs A_j alle erlaubten Werte enthält und hier nicht die notwendige Unterscheidung macht. Durch Umformungen nach Abschnitt 4.3 können die Bedingungen, die zur Festlegung der Ausgabe führen, in die Zustandsmenge hineinkodiert werden, indem der Wertebereich eines Zustands q_{A_j} eines Attributs A_j in zwei Zustände $q_{A_j;A_i=c}$ und $q_{A_j;A_i\neq c}$ aufgeteilt wird, wobei der Wertebereich von $q_{A_j;A_i=c}$ diejenigen Teilwerte enthält, welche die Annotation c der Selektionsbedingung enthalten, und der Wertebereich von $q_{A_j;A_i=c}$ diejenigen Teilwerte enthält, welche c nicht enthalten. Wird eine Annotation von einem positiv markierten Zustand gefolgt, kommt sie zum Ausgabewert, sonst zum Komplement. Bevor die Ausgabewerte festgelegt werden, ist also eine Umformung des Automaten notwendig.

Umformung des Automaten

Ist ein Automat für einen Relationstyp $R\ [\ A_1, \ldots, A_i, \ldots, A_n\ ]$ gegeben und soll eine Selektion $\sigma_{A_i=c}(R)$ kodiert werden, so enthält der Automat Regeln für das Selektionsattribut A_i der Form

$$\{\ a_1 : q_{A_{i+1}}, \ldots, a_n : q_{A_{i+1}}\ \} \quad \longrightarrow \quad q_{A_i}$$

Einige dieser Regeln enthalten die Annotation c aus der Selektionsbedingung und andere nicht. Der Wertebereich des Zustands q_{A_i} wird auf zwei neue Zustände $q_{A_i;A_i=c}$ und $q_{A_i;A_i\neq c}$ aufgeteilt. Werte, die ein c enthalten, werden nach $q_{A_i;A_i=c}$ abgeleitet und Werte, die kein c enthalten, nach $q_{A_i;A_i\neq c}$:

$$\begin{array}{lll} \{\ a_1 : q_{A_{i+1}}, \ldots, c : q_{A_{i+1}}, \ldots, a_n : q_{A_{i+1}}\ \} & \longrightarrow & q_{A_i;A_i=c} \\ \{\ a_1 : q_{A_{i+1}}, \ldots, a_m : q_{A_{i+1}}\ \} & \longrightarrow & q_{A_i;A_i\neq c} \end{array}$$

Im umgeformten Automaten enthalten die Übergangsregeln des direkten Vorgängerattributs A_{i-1} des Attributs A_i Regeln, bei denen die Zustände $q_{A_i;A_i=c}$ und $q_{A_i;A_i\neq c}$ auf den linken Seiten auftreten:

$$\begin{array}{lcl} \{\ a_1 : q_{A_i;A_i=c},\ \ldots,\ a_n : q_{A_i;A_i\neq c}\ \} & \longrightarrow & q_{A_{i-1}} \\ \{\ a_1 : q_{A_i;A_i\neq c},\ \ldots,\ a_n : q_{A_i;A_i\neq c}\ \} & \longrightarrow & q_{A_{i-1}} \end{array}$$

Kommt ein $q_{A_i;A_i=c}$ vor, enthält der dorthin abgeleitete Teilbaum die Annotation aus der Selektionsbedingung. Bei einem nach $q_{A_i;A_i\neq c}$ abgeleiteten Teilbaum triff dies nicht zu. Der Wertebereich des Zustands $q_{A_{i-1}}$ wird nun ebenfalls auf zwei neue Zustände $q_{A_{i-1};A_i=c}$ und $q_{A_{i-1};A_i\neq c}$ aufgeteilt, wobei bei Regeln, die mindestens einen positiv markierten Zustand auf der linken Seite enthalten, der Zustand $q_{A_{i-1}}$ ebenfalls positiv und ansonsten negativ markiert wird. Nach dieser Umformung gibt es für das Attribut A_{i-1} die Regeln:

$$\begin{array}{lcl} \{\ a_1 : q_{A_i;A_i=c},\ \ldots,\ a_n : q_{A_i;A_i\neq c}\ \} & \longrightarrow & q_{A_{i-1};A_i=c} \\ \{\ a_1 : q_{A_i;A_i\neq c},\ \ldots,\ a_n : q_{A_i;A_i\neq c}\ \} & \longrightarrow & q_{A_{i-1};A_i\neq c} \end{array}$$

Die Regeln für das Attribut A_{i-2} enthalten nun wieder positiv und negativ markierte Zustände, so dass der zu diesem Attribut gehörigen Zustand $q_{A_{i-2}}$ auf $q_{A_{i-2};A_i=c}$ und $q_{A_{i-2};A_i\neq c}$ aufgeteilt wird, wie dies beim Attribut A_{i-1} geschehen ist. Dieses Verfahren wird bis zum ersten Attribut A_1 fortgesetzt, dessen Zustand ebenfalls aufgeteilt wird. Beim Attribut A_1 ist eine Regel vorhanden, die links den leeren Wert enthält. Da die leere Relation ebenfalls die Selektionsbedingung erfüllt – sind keine Tupel vorhanden, wird als Ergebnis der Selektion die leere Relation erwartet –, wird der Zustand dieser Regel zu den positiv markierten zugeschlagen.

Der Zustand des Attributs A_{i+1}, welches dem Selektionsattribut A_i nachfolgt, wird ebenfalls positiv bzw. negativ markiert, je nachdem ob in der dann folgenden Ableitung des zugehörigen Teilbaums die Annotation c aus der Selektionsbedingung erreicht wird oder nicht. Sind die Regeln des Selektionsattributs A_i als

$$\begin{array}{lcl} \{\ a_1 : q_{A_{i+1}},\ \ldots,\ c : q_{A_{i+1}},\ \ldots,\ a_n : q_{A_{i+1}}\ \} & \longrightarrow & q_{A_i;A_i=c} \\ \{\ a_1 : q_{A_{i+1}},\ \ldots,\ a_m : q_{A_{i+1}}\ \} & \longrightarrow & q_{A_i;A_i\neq c} \end{array}$$

vorhanden, wird der Wertebereich von $q_{A_{i+1}}$ des folgenden Attributs A_{i+1} auf zwei Zustände $q_{A_{i+1};A_i=c}$ und $q_{A_{i+1};A_i\neq c}$ dupliziert. Der Zustand $q_{A_{i+1};A_i=c}$ beschreibt Werte, die im Eingabewert unterhalb der Annotation c liegen, und $q_{A_{i+1};A_i\neq c}$ beschreibt Werte, die im Eingabewert nicht unterhalb der Annotation c liegen. Nach dem Einfügen der Zustände ergeben sich die Regeln für das Selektionsattribut A_i zu

$$\begin{array}{lcl} \{\ a_1 : q_{A_{i+1};A_i\neq c},\ \ldots,\ c : q_{A_{i+1};A_i=c},\ \ldots,\ a_n : q_{A_{i+1};A_i\neq c}\ \} & \longrightarrow & q_{A_i;A_i=c} \\ \{\ a_1 : q_{A_{i+1};A_i\neq c},\ \ldots,\ a_m : q_{A_{i+1};A_i\neq c}\ \} & \longrightarrow & q_{A_i;A_i\neq c} \end{array}$$

und die für das Attribut A_{i+1} zu

$$\begin{array}{lcl} \{\ a_1 : q_{A_{i+2}},\ \ldots,\ a_n : q_{A_{i+2}}\ \} & \longrightarrow & q_{A_{i+1};A_i=c} \\ \{\ a_1 : q_{A_{i+2}},\ \ldots,\ a_n : q_{A_{i+2}}\ \} & \longrightarrow & q_{A_{i+1};A_i\neq c} \end{array}$$

Der Wertebereich des Zustands $q_{A_{i+2}}$ des nächsten Attributs A_{i+2} wird wieder auf zwei Zustände $q_{A_{i+2};A_i=c}$ und $q_{A_{i+2};A_i\neq c}$ dupliziert, wobei auf einer linken Seite nur der positiv markierte Zustand $q_{A_{i+2};A_i=c}$ verwendet wird, wenn auf der rechten Seite der positiv markierte Zustand $q_{A_{i+1};A_i=c}$ vorhanden ist, und nur der negativ markierte, wenn der Zustand rechts negativ markiert ist. Es entstehen die Regeln

$$\{\ a_1 : q_{A_{i+2};A_i=c},\ \ldots,\ a_n : q_{A_{i+2};A_i=c}\ \} \quad \longrightarrow \quad q_{A_{i+1};A_i=c}$$
$$\{\ a_1 : q_{A_{i+2};A_i\neq c},\ \ldots,\ a_n : q_{A_{i+2};A_i\neq c}\ \} \quad \longrightarrow \quad q_{A_{i+1};A_i\neq c}$$

Entsprechend wird für alle weiteren Attribute bis zum letzten Attribut A_n verfahren.

Definition der Abbildung
Durch die Umformungen ist aus dem durch die Kodierung erzeugten Automat ein äquivalenter Automat entstanden, der ebenfalls kontextbezogen deterministisch ist, und bei dem Teilwerte der Eingabe, welche den Wert der Selektionsbedingung enthalten, in positiv markierte Zustände und Teilwerte, welche den Wert nicht enthalten, in negativ markierte Zustände abgeleitet werden. Auf Basis dieses Automaten als Quellschema wird die Selektion definiert, indem die Regeln um passende Ausgabewerte ergänzt werden:

- Tritt hinter einer Annotation a_i auf der linken Seite ein positiv markierter Zustand auf, wird diese Annotation als a_i auf die Ausgabeseite der Regel übernommen.

- Ist der Zustand hinter der Annotation a_i negativ oder nicht markiert, wird die Annotation als ${a_i}^C$ auf die Ausgabeseite der Regel übernommen und gehört zum Komplement.

- Sind auf der linken Seite keine Zustände vorhanden, ist der Zustand auf der rechten Seite ausschlaggebend. Ist dieser positiv markiert, kommen die Annotation in die Ausgabe, sonst zum Komplement.

Für die Regeln des Attributs A_i entstehen etwa die um die Ausgabe ergänzten Regeln:

$$\begin{aligned}
&\{\ a_1 : q_{A_{i+1};A_i\neq c}(x_1),\ \ldots,\ c : q_{A_{i+1};A_i=c}(x),\ \ldots,\ a_n : q_{A_{i+1};A_i\neq c}(x_n)\ \} \\
&\qquad \longrightarrow q_{A_i;A_i=c}\ (\ \{\ a_1^C : x_1,\ \ldots,\ c : x,\ \ldots,\ a_n^C : x_n\ \}\ ) \\
&\{\ a_1 : q_{A_{i+1};A_i\neq c}(x_1),\ \ldots,\ a_m : q_{A_{i+1};A_i\neq c}(x_m)\ \} \\
&\qquad \longrightarrow q_{A_i;A_i\neq c}\ (\ \{\ a_1^C : x_1,\ \ldots,\ a_m^C : x_m\ \}\ )
\end{aligned}$$

Eine Ausnahme dieser Vorgehensweise stellen die Schlussregeln dar, welche die Relationsbezeichner modellieren. Hier wird immer die Annotation der selektierten Relation in die Ausgabe übernommen, da bei einer Selektion immer eine Relation zurückgegeben wird, auch wenn es die leere Relation ist, also kein Tupel die Selektionsbedingung erfüllt.

Beispiel 5.5 (*Kodierung einer Selektion*)
Ist das relationale Schema aus Beispiel 5.2 gegeben, sind die Übergangsregeln

$$
\begin{array}{lcl}
\{\ R : q_A,\ S : q_D\ \} & \longrightarrow & q_f \\
\{\ \} & \longrightarrow & q_A \\
\{\ a : q_B\ \} & \longrightarrow & q_A \\
\{\ b : q_B\ \} & \longrightarrow & q_A \\
\{\ a : q_B,\ b : q_B\ \} & \longrightarrow & q_A \\
\{\ a : q_C\ \} & \longrightarrow & q_B \\
\{\ b : q_C\ \} & \longrightarrow & q_B \\
\{\ a : q_C,\ b : q_C\ \} & \longrightarrow & q_B \\
\{\ a\ \} & \longrightarrow & q_C \\
\{\ b\ \} & \longrightarrow & q_C \\
\{\ a,\ b\ \} & \longrightarrow & q_C \\
\{\ \} & \longrightarrow & q_D \\
\{\ c\ \} & \longrightarrow & q_D \\
\{\ d\ \} & \longrightarrow & q_D \\
\{\ c,\ d\ \} & \longrightarrow & q_D
\end{array}
$$

vorhanden.

Eine Selektion $\sigma_{B=b}(R)$ wird nun wie folgt kodiert. Zuerst wird der Automat umgeformt, indem der Wertebereich des Zustands q_B des Selektionsattributs in zwei Zustände $q_{B;B=b}$ und $q_{B;B\neq b}$ aufgeteilt wird. Danach erfolgt eine Aufteilung von q_A. Für den Zustand q_C wird der Wertebereich auf zwei neue Zustände $q_{C;B=b}$ und $q_{C;B\neq b}$ dupliziert. Es ergibt sich die Regelmenge

$$
\begin{array}{lcl}
\{\ R : q_{A;B=b},\ S : q_D\ \} & \longrightarrow & q_f \\
\{\ R : q_{A;B\neq b},\ S : q_D\ \} & \longrightarrow & q_f \\
\{\ \} & \longrightarrow & q_{A;B=b} \\
\{\ a : q_{B;B=b}\ \} & \longrightarrow & q_{A;B=b} \\
\{\ a : q_{B;B\neq b}\ \} & \longrightarrow & q_{A;B\neq b} \\
\{\ b : q_{B;B=b}\ \} & \longrightarrow & q_{A;B=b} \\
\{\ b : q_{B;B\neq b}\ \} & \longrightarrow & q_{A;B\neq b} \\
\{\ a : q_{B;B=b},\ b : q_{B;B=b}\ \} & \longrightarrow & q_{A;B=b} \\
\{\ a : q_{B;B=b},\ b : q_{B;B\neq b}\ \} & \longrightarrow & q_{A;B=b} \\
\{\ a : q_{B;B\neq b},\ b : q_{B;B=b}\ \} & \longrightarrow & q_{A;B=b} \\
\{\ a : q_{B;B\neq b},\ b : q_{B;B\neq b}\ \} & \longrightarrow & q_{A;B\neq b} \\
\{\ a : q_{C;B\neq b}\ \} & \longrightarrow & q_{B;B\neq b} \\
\{\ b : q_{C;B=b}\ \} & \longrightarrow & q_{B;B=b} \\
\{\ a : q_{C;B\neq b},\ b : q_{C;B=b}\ \} & \longrightarrow & q_{B;B=b} \\
\{\ a\ \} & \longrightarrow & q_{C;B=b} \\
\{\ b\ \} & \longrightarrow & q_{C;B=b} \\
\{\ a,\ b\ \} & \longrightarrow & q_{C;B=b}
\end{array}
$$

$$
\begin{array}{ll}
\{\ a\ \} & \longrightarrow\ q_{C;B\neq b} \\
\{\ b\ \} & \longrightarrow\ q_{C;B\neq b} \\
\{\ a, b\ \} & \longrightarrow\ q_{C;B\neq b} \\
\{\ \} & \longrightarrow\ q_D \\
\{\ c\ \} & \longrightarrow\ q_D \\
\{\ d\ \} & \longrightarrow\ q_D \\
\{\ c, d\ \} & \longrightarrow\ q_D
\end{array}
$$

Dieser Automat dient als Grundlage, die Selektionsabbildung zu definieren. Ist auf der linken Seite einer Regel ein Zustand positiv markiert, wird die zugehörige Annotation der Ausgabe zugeschlagen, sonst dem Komplement. Sind auf der linken Seite keine Zustände vorhanden, gibt der Zustand rechts den Ausschlag:

$$
\begin{array}{ll}
\{\ R : q_{A;B=b}(x_1),\ S : q_D(x_2)\ \} & \longrightarrow\ q_f\ (\ \{\ R : x_1,\ S^C : x_2\ \}\ ) \\
\{\ R : q_{A;B\neq b}(x_1),\ S : q_D(x_2)\ \} & \longrightarrow\ q_f\ (\ \{\ R : x_1,\ S^C : x_2\ \}\ ) \\
\{\ \} & \longrightarrow\ q_{A;B=b}\ (\ \{\ \}\ ) \\
\{\ a : q_{B;B=b}(x_1)\ \} & \longrightarrow\ q_{A;B=b}\ (\ \{\ a : x_1\ \}\ ) \\
\{\ a : q_{B;B\neq b}(x_1)\ \} & \longrightarrow\ q_{A;B\neq b}\ (\ \{\ a^C : x_1\ \}\ ) \\
\{\ b : q_{B;B=b}(x_1)\ \} & \longrightarrow\ q_{A;B=b}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ b : q_{B;B\neq b}(x_1)\ \} & \longrightarrow\ q_{A;B\neq b}\ (\ \{\ b^C : x_1\ \}\ ) \\
\{\ a : q_{B;B=b}(x_1),\ b : q_{B;B=b}(x_2)\ \} & \longrightarrow\ q_{A;B=b}\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a : q_{B;B=b}(x_1),\ b : q_{B;B\neq b}(x_2)\ \} & \longrightarrow\ q_{A;B=b}\ (\ \{\ a : x_1,\ b^C : x_2\ \}\ ) \\
\{\ a : q_{B;B\neq b}(x_1),\ b : q_{B;B=b}(x_2)\ \} & \longrightarrow\ q_{A;B=b}\ (\ \{\ a^C : x_1,\ b : x_2\ \}\ ) \\
\{\ a : q_{B;B\neq b}(x_1),\ b : q_{B;B\neq b}(x_2)\ \} & \longrightarrow\ q_{A;B\neq b}\ (\ \{\ a^C : x_1,\ b^C : x_2\ \}\ ) \\
\{\ a : q_{C;B\neq b}(x_1)\ \} & \longrightarrow\ q_{B;B\neq b}\ (\ \{\ a^C : x_1\ \}\ ) \\
\{\ b : q_{C;B=b}(x_1)\ \} & \longrightarrow\ q_{B;B=b}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ a : q_{C;B\neq b}(x_1),\ b : q_{C;B=b}(x_2)\ \} & \longrightarrow\ q_{B;B=b}\ (\ \{\ a^C : x_1,\ b : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow\ q_{C;B=b}\ (\ \{\ a\ \}\ ) \\
\{\ b\ \} & \longrightarrow\ q_{C;B=b}\ (\ \{\ b\ \}\ ) \\
\{\ a, b\ \} & \longrightarrow\ q_{C;B=b}\ (\ \{\ a, b\ \}\ ) \\
\{\ a\ \} & \longrightarrow\ q_{C;B\neq b}\ (\ \{\ a^C\ \}\ ) \\
\{\ b\ \} & \longrightarrow\ q_{C;B\neq b}\ (\ \{\ b^C\ \}\ ) \\
\{\ a, b\ \} & \longrightarrow\ q_{C;B\neq b}\ (\ \{\ a^C, b^C\ \}\ ) \\
\{\ \} & \longrightarrow\ q_D\ (\ \{\ \}^{\,C}\ ) \\
\{\ c\ \} & \longrightarrow\ q_D\ (\ \{\ c^C\ \}\ ) \\
\{\ d\ \} & \longrightarrow\ q_D\ (\ \{\ d^C\ \}\ ) \\
\{\ c, d\ \} & \longrightarrow\ q_D\ (\ \{\ c^C, d^C\ \}\ )
\end{array}
$$

Für den Beispielwert aus Abb. 5.11 erzeugt der Automat die in der Abbildung angegebene Ausgabe. □

Im sich durch die Umformung ergebenden Automaten sind dem Attribut A_i zwei Zustände $q_{A_i;A_i=c}$ und $q_{A_i;A_i\neq c}$ zugewiesen, obwohl ursprünglich durch die Kodierung nur ein Zustand q_{A_i} für dieses Attribut vorgesehen war. Durch Umformungen kann sich

folglich die Zahl der Zustände pro Attribut ändern. Die einem Attribut zugeordnete Schicht innerhalb eines Baums bleibt hierbei gleich. Die Werte des Attributs B aus dem Beispiel liegen auch nach der Umformung auf der zweiten Schicht von unten.

Sind Konsistenzbedingungen vorhanden, kann einem Attribut eines Relationstyps schon bei der Kodierung nicht nur ein Zustand innerhalb des Automaten, sondern eine Reihe von Zuständen zugeordnet worden sein. Die Modellierung einer Selektion erfolgt dann analog der beschriebenen Vorgehensweise nicht nur für einen Zustand pro Attribut, sondern bei jedem Zustand des Attributs.

Der Automat $\mathcal{T}$, der sich aus der Kodierung des Schemas und den nachfolgenden Umformungen ergibt, ist kontextbezogen deterministisch. Die Kodierung beschreibt also eine Abbildung. Das Kodierungsverfahren für Abbildungen übernimmt bei jeder Regel die Annotationen der Eingabe- auf die Ausgabeseite, wobei sie dort eventuell mit C markiert sind. Wird der inverse Automat $\mathcal{T}^{-1}$ gebildet, werden die Ein- und die Ausgabewerte bei Regeln aus $\mathcal{T}$ vertauscht. Dies bedeutet, dass wenn eine Regelmenge aus Δ_q in $\mathcal{T}$ eine Partitionierung beschreibt, gilt dies auch für die entsprechende Regelmenge Δ_q aus $\mathcal{T}^{-1}$, da sich die Annotationen einer Regel auf Ein- und Ausgabeseite entsprechen. Folglich beschreibt die Kodierung einer Selektion eine informationserhaltende Sicht.

Aufbauend auf der Selektion gegen eine Konstante lassen sich komplexere Selektionskriterien zusammensetzen. Ist in der Selektionsbedingung eine Oder-Verknüpfung der Form $\sigma_{(A_i=c_1)\vee(A_i=c_2)}(R)$ auf einem Attribut A_i mit zwei Konstanten c_1 und c_2 vorhanden, werden die Regeln zum Attribut A_i positiv markiert, falls eine der Konstanten auf der linken Seite vorkommt, und sonst negativ. Die anderen Attribute werden analog zur obigen Vorgehensweise bei der Selektion mit einer Konstanten behandelt. Eine Und-Verknüpfung über mehrere Attribute $\sigma_{(A_i=c_1)\wedge(A_j=c_2)}(R)$ kann durch das Hintereinanderschalten von Selektionen modelliert werden, da $\sigma_{(A_i=c_1)\wedge(A_j=c_2)}(R) = \sigma_{A_i=c_1}(\sigma_{A_j=c_2}(R))$ gilt.

Weil die Domänen der Attribute als endlich angenommen werden, können auch Kriterien, die über einen Test auf Gleichheit hinausgehen, entsprechend kodiert werden. Alle Tupel mit einem Attributwert kleiner als c ergeben sich aus $\sigma_{A_i<c}(R) = \sigma_{(A_i=c_1)\vee\ldots\vee(A_i=c_n)}(R)$ mit $c_i < c$ für $1 \leq i \leq n$. Bei einer Selektion $\sigma_{A_i=A_j}(R)$, bei der die Werte unterschiedlicher Attribute innerhalb eines Tupels identisch sein sollen, kann der Operator ebenfalls umgeschrieben werden als $\sigma_{(A_i=c_1\wedge A_j=c_1)\vee(A_i=c_2\wedge A_j=c_2)\vee\ldots}(R)$ für alle Werte c_i der Domäne von A_i bzw. A_j.

5.4.3.2 Projektion

Eine Projektion wählt bei einer Relation die Werte der bei der Projektion angegebenen Attribute bei allen Tupeln aus und vereinigt diese zu einer neuen Relation. Im Fall von Bäumen entspricht ein Attribut einem Teil einer Schicht in einem Baum, so dass eine Projektion einen Baum horizontal zerteilt.

Ist wieder die Instanz mit den zwei Relationen R und S aus Beispiel 5.2 vorhanden und eine Projektion als $\pi_B(R)$ spezifiziert, dürfen nur die dem Attribut B zugeordneten

Ebenen ausgegeben werden (Abb. 5.12). Alle übrigen Ebenen werden dem Komplement zugewiesen. Eine Ausnahme stellt wieder die Annotation des Relationsbezeichners dar, die immer zur Ausgabe gehört.

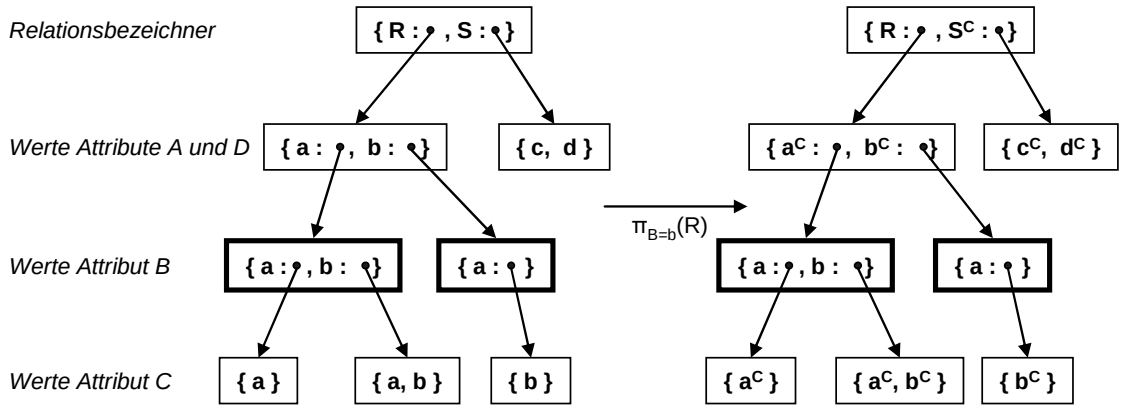


Abbildung 5.12: *Projektion $\pi_B(R)$ auf einem Beispielwert*

Eine Umformung des Automaten vor der Definition der Ausgabe ist nicht notwendig, da bei einer Projektion die Ausgabe nur von den Annotationen selbst abhängt und nicht von bereits abgeleiteten Werten wie bei der Selektion.

Definition der Abbildung

Ergibt eine Kodierung $enc(\mathbf{R})$ bzw. $enc(\mathbf{R},\mathcal{D})$ einen kontextbezogen deterministischen Automaten, so müssen bei einer Projektion als Abbildung alle Annotationen der Eingabeseite mit in die Ausgabe übernommen werden, die zu Regeln gehören, welche dem Projektionsattribut zugeordnet sind, und alle übrigen gehören zum Komplement.

Ist eine Relation $R\ [\ A_1, \ldots, A_i, \ldots, A_n\ ]$ gegeben und soll eine Projektion $\pi_{A_i}(R)$ kodiert werden, erzeugt die Kodierung des Schemas einen Automaten, der für das Projektionsattribut A_i unter anderem Übergangsregeln der Art

$$\{\ a_1 : q_{A_{i+1}}, \ldots, a_n : q_{A_{i+1}}\ \} \quad \longrightarrow \quad q_{A_i}$$

enthält. Da alle Annotationen bei diesen Regeln in die Ausgabe übernommen werden müssen, ergibt sich eine solche Regel mit Ausgabe als

$$\{\ a_1 : q_{A_{i+1}}(x_1) \ldots, a_n : q_{A_{i+1}}(x_n)\ \} \quad \longrightarrow \quad q_{A_i}\ (\ \{\ a_1 : x_1, \ldots, a_n : x_n\ \}\ )$$

Bei den Regeln der übrigen Attribute der Relation A_j, $i \neq j$, werden die Annotationen dem Komplement auf der Ausgabeseite zugeschlagen:

$$\{\ a_1 : q_{A_{j+1}}(x_1), \ldots, a_n : q_{A_{j+1}}(x_n)\ \} \quad \longrightarrow \quad q_{A_j}\ (\ \{\ a_1^C : x_1, \ldots, a_n^C : x_n\ \}\ )$$

Analog zum Fall der Selektion wird bei der Finalregel nur der Relationsbezeichner ausgegeben, welcher zur Relation gehört, auf der die Projektion durchgeführt wird, und eine leere Relation dieses Relationstyps ebenfalls der Ausgabe zugeschlagen.

Beispiel 5.6 (*Kodierung einer Projektion*)
Eine Projektion $\pi_B(R)$ auf dem Schema aus Beispiel 5.2 wird durch die folgenden Regeln definiert:

$$
\begin{array}{lcl}
\{\ R : q_A(x_1),\ S : q_D(x_2)\ \} & \longrightarrow & q_f\ (\ \{\ R : x_1,\ S^C : x_2\ \}\ ) \\
\{\ \} & \longrightarrow & q_A\ (\ \{\ \}\ ) \\
\{\ a : q_B(x_1)\ \} & \longrightarrow & q_A\ (\ \{\ a^C : x_1\ \}\ ) \\
\{\ b : q_B(x_1)\ \} & \longrightarrow & q_A\ (\ \{\ b^C : x_1\ \}\ ) \\
\{\ a : q_B(x_1),\ b : q_B(x_2)\ \} & \longrightarrow & q_A\ (\ \{\ a^C : x_1,\ b^C : x_2\ \}\ ) \\
\{\ a : q_C(x_1)\ \} & \longrightarrow & q_B\ (\ \{\ a : x_1\ \}\ ) \\
\{\ b : q_C(x_1)\ \} & \longrightarrow & q_B\ (\ \{\ b : x_2\ \}\ ) \\
\{\ a : q_C(x_1),\ b : q_C(x_2)\ \} & \longrightarrow & q_B\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_C\ (\ \{\ a^C\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_C\ (\ \{\ b^C\ \}\ ) \\
\{\ a,\ b\ \} & \longrightarrow & q_C\ (\ \{\ a^C,\ b^C\ \}\ ) \\
\{\ \} & \longrightarrow & q_D\ (\ \{\ \}^{\,C}\ ) \\
\{\ c\ \} & \longrightarrow & q_D\ (\ \{\ c^C\ \}\ ) \\
\{\ d\ \} & \longrightarrow & q_D\ (\ \{\ d^C\ \}\ ) \\
\{\ c,\ d\ \} & \longrightarrow & q_D\ (\ \{\ c^C,\ d^C\ \}\ )
\end{array}
$$

Bei den Regeln des Projektionsattributs B werden die Annotationen der Eingabeseite in die Ausgabe übernommen. Bei den anderen Attributen A, C und D kommen die Annotationen zum Komplement. Der Relationsbezeichner R wird ausgegeben.

Der Automat definiert die in Abb. 5.12 angegebene Abbildung. □

Der Automat $\mathcal{T}$, der sich aus der Kodierung des Schemas und den nachfolgenden Umformungen ergibt, ist kontextbezogen deterministisch. Die Kodierung beschreibt also eine Abbildung. Das Kodierungsverfahren für Abbildungen übernimmt bei jeder Regel die Annotationen der Eingabe- auf die Ausgabeseite, wobei sie dort eventuell mit C markiert sind. Wird der inverser Automat $\mathcal{T}^{-1}$ gebildet, werden die Ein- und die Ausgabewerte bei Regeln aus $\mathcal{T}$ vertauscht. Dies bedeutet, dass wenn eine Regelmenge aus Δ_q in $\mathcal{T}$ eine Partitionierung beschreibt, gilt dies auch für die entsprechende Regelmenge Δ_q aus $\mathcal{T}^{-1}$, da sich die Annotationen einer Regel auf Ein- und Ausgabeseite entsprechen. Mit dieser Kodierung beschreibt eine Projektion eine informationserhaltende Sicht.

Wird nicht nur ein Attribut, sondern werden mehrere Attribute projiziert, erfolgt eine Kodierung einer solchen Projektion analog, wobei bei den Regeln aller Projektionsattribute die Annotationen der Ausgabe zugeschlagen werden.

5.4.3.3 Kartesisches Produkt

Im Gegensatz zu Selektion und Projektion, bei denen aus Eingabewerten bestimmte Annotationen ausgewählt werden, um das Ergebnis zu generieren, tritt bei einem kartesischen Produkt eine Vervielfachung eines Werts in der Ausgabe auf. Aus einem einzelnen Tupel einer Eingaberelation werden Teile von mehreren Tupeln in der Bildinstanz.

Abb. 5.13 zeigt ein kartesisches Produkt $R \times S$ auf den beiden Beispielrelationen R und S. Die Ebenen, die der Relation S zugeordnet sind, werden vervielfacht und an die Blätter der Relation R angehängt, um das kartesische Produkt zu beschreiben. Die Werte der Relation S kommen zum Komplement.

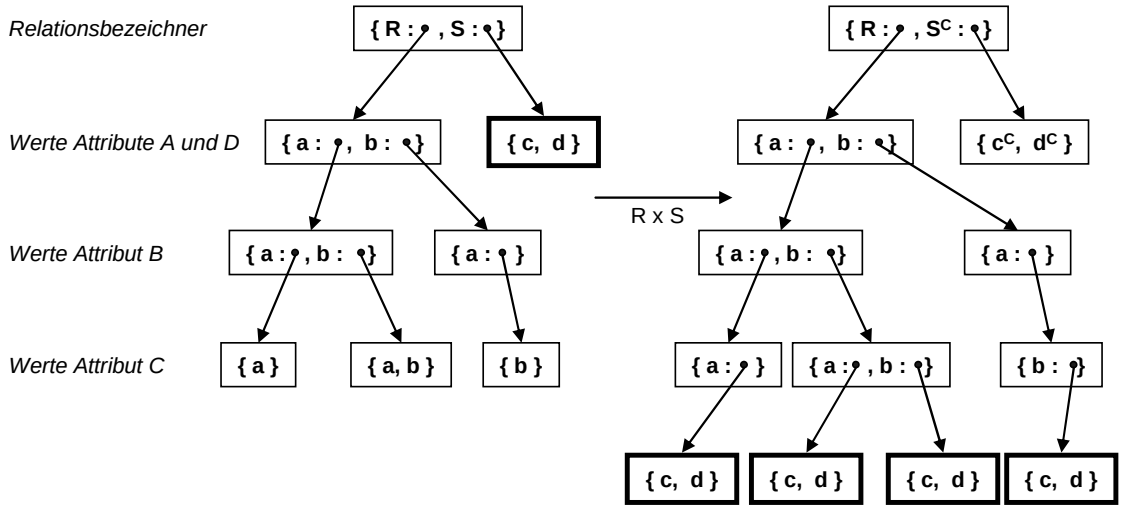


Abbildung 5.13: *Kartesisches Produkt* $R \times S$

Soll ein kartesisches Produkt mittels eines Baumautomaten umgesetzt werden, ergeben sich zunächst zwei Schwierigkeiten. Zum einen wird der Eingabewert bestehend aus den beiden Relationen von unten nach oben durchlaufen, d.h. der Wert, der an die eine Relation anzuhängen ist, ist erst nach dem Durchlaufen des Automaten bekannt. Somit kann er nicht an die Blätter in der Ausgabe platziert werden. Zum anderen erlauben es die Regeln eines Baumautomaten nicht, Teilbäume der Eingabe in der Ausgabe zu vervielfachen. Beide Schwierigkeiten können umgangen werden, indem der Automat vor der Abbildungsdefinition passend umgeformt wird.

Umformung des Automaten

Zur Beschreibung eines kartesischen Produkts $R_1 \times R_2$ werden nicht flache, sondern geschachtelte Ein- bzw. Ausgabewerte bei den Übergangsregeln verwendet. Hierzu erfolgt zuerst eine Umformung des Automaten.

Durch die Kodierung $enc(\mathbf{R})$ bzw. $enc(\mathbf{R}, \mathcal{D})$ zweier Relationstypen R_1 [$A_1, \ldots, A_n$] und R_2 [$B_1, \ldots, B_m$] entsteht ein Automat mit den Übergangsregeln

$$
\begin{array}{lcl}
\{\ R_1 : q_{R_1;A_1},\ R_2 : q_{R_2;B_1}\ \} & \longrightarrow & q_f \\
\{\ \} & \longrightarrow & q_{R_1;A_1} \\
 & \ldots & \\
\{\ a_1 : q_{R_1;A_{i+1}}, \ldots, a_{k_1} : q_{R_1;A_{i+1}}\ \} & \longrightarrow & q_{R_1;A_i} \\
 & \ldots & \\
\{\ b_1, \ldots, b_{k_2}\ \} & \longrightarrow & q_{R_1;A_n} \\
 & & \\
\{\ \} & \longrightarrow & q_{R_2;B_1} \\
 & \ldots & \\
\{\ c_1 : q_{R_2;B_{j+1}}, \ldots, c_{k_3} : q_{R_2;B_{j+1}}\ \} & \longrightarrow & q_{R_2;B_j} \\
 & \ldots & \\
\{\ d_1, \ldots, d_{k_4}\ \} & \longrightarrow & q_{R_2;B_m}
\end{array}
$$

wobei die Zustände $q_{R_1;A_i}$ Attribute A_i der ersten Relation R_1 repräsentieren und die Zustände $q_{R_2;B_j}$ Attribute B_j der zweiten Relation R_2. Werden aus diesem Automaten alle Zustände der zweiten Relation durch das Einsetzen der entsprechenden Regeln entfernt, besitzt der hieraus resultierende Automat die Übergangsregeln:

$$
\begin{array}{lcl}
\{\ R_1 : q_{R_1;A_1},\ R_2 : \{\ \}\ \} & \longrightarrow & q_f \\
\{\ R_1 : q_{R_1;A_1},\ R_2 : v_1\ \} & \longrightarrow & q_f \\
 & \ldots & \\
\{\ R_1 : q_{R_1;A_1},\ R_2 : v_l\ \} & \longrightarrow & q_f \\
\{\ \} & \longrightarrow & q_{R_1;A_1} \\
 & \ldots & \\
\{\ a_1 : q_{R_1;A_{i+1}}, \ldots, a_{k_1} : q_{R_1;A_{i+1}}\ \} & \longrightarrow & q_{R_1;A_i} \\
 & \ldots & \\
\{\ b_1, \ldots, b_{k_2}\ \} & \longrightarrow & q_{R_1;A_n}
\end{array}
$$

Die Regeln $\{\ R_1 : q_{R_1;A_1},\ R_2 : v_i\ \} \longrightarrow q_f$ enthalten einen geschachtelten Wert auf der Eingabeseite, wobei für jede mögliche Relation von R_2 eine Regel mit deren Kodierung v_i vorhanden ist. Jede Schlussregel gibt hierbei an, welchen Wert die zweite Relation besitzt. Abhängig davon, welche der Schlussregeln bei einer Ableitung erreicht wird – also abhängig davon welcher Wert bei der zweiten Relation vorhanden ist –, kann nun für die Regeln die Ausgabe definiert werden. Der Wertebereich von $q_{R_1;A_1}$ wird hierfür auf die Zustände $q_{R_1;A_1,v_1}$, $q_{R_1;A_1,v_2} \ldots q_{R_1;A_1,v_l}$ dupliziert. Bei jeder der Regeln $\{\ R_1 : q_{R_1;A_1},\ R_2 : v_j\ \} \longrightarrow q_f$ wird $q_{R_1;A_1}$ durch den entsprechenden Zustand $q_{R_1;A_1,v_j}$ ersetzt, so dass v_j bei Zustand und hinter R_2 übereinstimmen. Wird ein Wert nach $q_{R_1;A_1,v_j}$ abgeleitet, bedeutet dies, dass ein v_j bei R_2 vorhanden ist. Entsprechend werden die Wertebereiche der Zustände der nachfolgenden Attribute A_i von R_1 auf Zustände dupliziert, wobei ein Zustand $q_{R_1;A_i,v_j}$ auf der linken Seite einer Regel Verwendung findet, wenn auf der rechten Seite ein ebenfalls mit v_j markierter Zustand vorhanden ist:

$$
\begin{array}{lcl}
\{\ R_1 : q_{R_1;A_1,\{\}},\ R_2 : \{\ \}\ \} & \longrightarrow & q_f \\
\{\ R_1 : q_{R_1;A_1,v_1},\ R_2 : v_1\ \} & \longrightarrow & q_f \\
 & \ldots & \\
\{\ R_1 : q_{R_1;A_1,v_n},\ R_2 : v_l\ \} & \longrightarrow & q_f \\
\{\ \} & \longrightarrow & q_{R_1;A_1,\{\}} \\
 & \ldots & \\
\{\ a_1 : q_{R_1;A_{i+1},\{\}},\ \ldots,\ a_{k_1} : q_{R_1;A_{i+1},\{\}}\ \} & \longrightarrow & q_{R_1;A_i,\{\}} \\
 & \ldots & \\
\{\ b_1,\ \ldots,\ b_{k_2}\ \} & \longrightarrow & q_{R_1;A_n,\{\}} \\
\{\ \} & \longrightarrow & q_{R_1;A_1,v_1} \\
 & \ldots & \\
\{\ a_1 : q_{R_1;A_{i+1},v_1},\ \ldots,\ a_{k_1} : q_{R_1;A_{i+1},v_1}\ \} & \longrightarrow & q_{R_1;A_i,v_1} \\
 & \ldots & \\
\{\ b_1,\ \ldots,\ b_{k_2}\ \} & \longrightarrow & q_{R_1;A_n,v_1} \\
\{\ \} & \longrightarrow & q_{R_1;A_1,v_l} \\
 & \ldots & \\
\{\ a_1 : q_{R_1;A_{i+1},v_l},\ \ldots,\ a_{k_1} : q_{R_1;A_{i+1},v_l}\ \} & \longrightarrow & q_{R_1;A_i,v_l} \\
 & \ldots & \\
\{\ b_1,\ \ldots,\ b_{k_2}\ \} & \longrightarrow & q_{R_1;A_n,v_l}
\end{array}
$$

Definition der Abbildung

Auf Basis des umgeformten Automaten als Quellschema wird eine Ausgabe definiert. Wird ein Teilwert in einen mit v_j markierten Zustand abgeleitet, dann ist sichergestellt, dass bei der zweiten Relation R_2 ein Wert v_j vorhanden ist. Bei einer Regel für eine Blattebene $\{\ b_1, \ldots, b_n\ \} \longrightarrow q_{R_1;A_n,v_j}$ werden die Annotationen der Eingabe in die Ausgabe übernommen und an jede Annotation der Wert v_j angehängt, der beim Zustand der Regel angegeben ist. Bei den anderen Regeln für die Attribute werden die Annotationen der Eingabe in die Ausgabe übernommen. Eine Ausnahme hiervon stellen die Regeln dar, bei denen der im Zustand angegebene Wert v_j dem leeren Wert entspricht. Bei diesen Regeln steht fest, dass die zweite Relation leer ist und somit immer die leere Relation ausgegeben werden muss. Bei solchen Regeln kommen daher alle Annotationen bis auf den leeren Wert ins Komplement. Bei den Schlussregeln nach q_f gehören die Annotationen der zweiten Relation immer alle zum Komplement. Es ergibt sich ein Automat mit Ausgabe und den folgenden Regeln:

$$
\begin{array}{l}
\{\ R_1 : q_{R_1;A_1,\{\}}(x_1),\ R_2 : \{\ \}\ \} \\
\qquad\qquad \longrightarrow q_f\ (\ \{\ R_1 : x_1,\ R_2^C : \{\ \}^{\,C}\ \}\ ) \\
\{\ R_1 : q_{R_1;A_1,v_1}(x_1),\ R_2 : v_1\ \} \\
\qquad\qquad \longrightarrow q_f\ (\ \{\ R_1 : x_1,\ R_2^C : v_1^C\ \}\ ) \\
\ldots \\
\{\ R_1 : q_{R_1;A_1,v_l}(x_1),\ R_2 : v_l\ \} \\
\qquad\qquad \longrightarrow q_f\ (\ \{\ R_1 : x_1,\ R_2^C : v_l^C\ \}\ )
\end{array}
$$

$$
\begin{aligned}
&\{\ \} \\
&\qquad\longrightarrow q_{R_1;A_1,\{\}}\ (\ \{\ \}\ ) \\
&\ldots \\
&\{\ a_1 : q_{R_1;A_{i+1},\{\}}(x_1),\ \ldots,\ a_{k_1} : q_{R_1;A_{i+1},\{\}}(x_{k_1})\ \} \\
&\qquad\longrightarrow q_{R_1;A_i,\{\}}\ (\ \{\ a_1^C : x_1,\ \ldots,\ a_{k_1}^C : x_{k_1}\ \}\ ) \\
&\ldots \\
&\{\ b_1,\ \ldots,\ b_{k_2}\ \} \\
&\qquad\longrightarrow q_{R_1;A_n,\{\}}\ (\ \{\ b_1^C : \{\ \}\ ,\ \ldots,\ b_{k_2}^C : \{\ \}\ \}\ ) \\
&\{\ \} \\
&\qquad\longrightarrow q_{R_1;A_1,v_1}\ (\ \{\ \}\ ) \\
&\ldots \\
&\{\ a_1 : q_{R_1;A_{i+1},v_1}(x_1),\ \ldots,\ a_{k_1} : q_{R_1;A_{i+1},v_1}(x_{k_1})\ \} \\
&\qquad\longrightarrow q_{R_1;A_i,v_1}\ (\ \{\ a_1 : x_1,\ \ldots,\ a_{k_1} : x_{k_1}\ \}\ ) \\
&\ldots \\
&\{\ b_1,\ \ldots,\ b_{k_2}\ \} \\
&\qquad\longrightarrow q_{R_1;A_n,v_1}\ (\ \{\ b_1 : v_1,\ \ldots,\ b_{k_2} : v_1\ \}\ ) \\
&\{\ \} \\
&\qquad\longrightarrow q_{R_1;A_1,v_l}\ (\ \{\ \}\ ) \\
&\ldots \\
&\{\ a_1 : q_{R_1;A_{i+1},v_l}(x_1),\ \ldots,\ a_{k_1} : q_{R_1;A_{i+1},v_l}(x_{k_1})\ \} \\
&\qquad\longrightarrow q_{R_1;A_i,v_l}\ (\ \{\ a_1 : x_1,\ \ldots,\ a_{k_1} : x_{k_1}\ \}\ ) \\
&\ldots \\
&\{\ b_1,\ \ldots,\ b_{k_2}\ \} \\
&\qquad\longrightarrow q_{R_1;A_n,v_l}\ (\ \{\ b_1 : v_l,\ \ldots,\ b_{k_2} : v_l\ \}\ )
\end{aligned}
$$

Durch die Regeln des Automaten, welche die Blattebenen der Eingabe beschreiben, findet das Anhängen der kodierten Tupel statt, wobei durch die Markierungen der Zustände mit v_j sichergestellt ist, dass wenn ein Wert v_j in der Ausgabe angehängt wird, dann dieser Wert bei der Relation R_2 auch vorhanden ist. Das Vervielfachen der Werte der Relation R_2 findet jeweils innerhalb einer Regel durch den geschachtelten Ausgabewert statt.

Beispiel 5.7 (*Kartesisches Produkt*)
Für das kartesische Produkt zwischen den zwei Relationen $R \times S$ aus Beispiel 5.2 entsteht der Automat:

$$
\begin{aligned}
&\{\ R : q_{R;A,\{\}}(x_1),\ S : \{\ \}\ \} && \longrightarrow && q_f\ (\ \{\ R : x_1,\ S^C : \{\ \}^{\,C}\ \}\ ) \\
&\{\ R : q_{R;A,v_1}(x_1),\ S : \{\ c\ \}\ \} && \longrightarrow && q_f\ (\ \{\ R : x_1,\ S^C : \{\ c^C\ \}\ \}\ ) \\
&\{\ R : q_{R;A,v_2}(x_1),\ S : \{\ d\ \}\ \} && \longrightarrow && q_f\ (\ \{\ R : x_1,\ S^C : \{\ d^C\ \}\ \}\ ) \\
&\{\ R : q_{R;A,v_3}(x_1),\ S : \{\ c,\ d\ \}\ \} && \longrightarrow && q_f\ (\ \{\ R : x_1,\ S^C : \{\ c^C,\ d^C\ \}\ \}\ )
\end{aligned}
$$

$$
\begin{array}{lcl}
\{\ \} & \longrightarrow & q_{R;A,\{\}}\ (\ \{\ \}\ ) \\
\{\ a : q_{R;B,\{\}}(x_1)\ \} & \longrightarrow & q_{R;A,\{\}}\ (\ \{\ a^C : x_1\ \}\ ) \\
\{\ b : q_{R;B,\{\}}(x_1)\ \} & \longrightarrow & q_{R;A,\{\}}\ (\ \{\ b^C : x_1\ \}\ ) \\
\{\ a : q_{R;B,\{\}}(x_1),\ b : q_{R;B,\{\}}(x_2)\ \} & \longrightarrow & q_{R;A,\{\}}\ (\ \{\ a^C : x_1,\ b^C : x_2\ \}\ ) \\
\{\ a : q_{R;C,\{\}}(x_1)\ \} & \longrightarrow & q_{R;B,\{\}}\ (\ \{\ a^C : x_1\ \}\ ) \\
\{\ b : q_{R;C,\{\}}(x_1)\ \} & \longrightarrow & q_{R;B,\{\}}\ (\ \{\ b^C : x_1\ \}\ ) \\
\{\ a : q_{R;C,\{\}}(x_1),\ b : q_{R;C,\{\}}(x_2)\ \} & \longrightarrow & q_{R;B,\{\}}\ (\ \{\ a^C : x_1,\ b^C : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_{R;C,\{\}}\ (\ \{\ a^C : \{\ \}\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_{R;C,\{\}}\ (\ \{\ b^C : \{\ \}\ \}\ ) \\
\{\ a,\ b\ \} & \longrightarrow & q_{R;C,\{\}}\ (\ \{\ a^C : \{\ \}\ ,\ b^C : \{\ \}\ \}\ ) \\
\{\ \} & \longrightarrow & q_{R;A,v_1}\ (\ \{\ \}\ ) \\
\{\ a : q_{R;B,v_1}(x_1)\ \} & \longrightarrow & q_{R;A,v_1}\ (\ \{\ a : x_1\ \}\ ) \\
\{\ b : q_{R;B,v_1}(x_1)\ \} & \longrightarrow & q_{R;A,v_1}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ a : q_{R;B,v_1}(x_1),\ b : q_{R;B,v_1}(x_2)\ \} & \longrightarrow & q_{R;A,v_1}\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a : q_{R;C,v_1}(x_1)\ \} & \longrightarrow & q_{R;B,v_1}\ (\ \{\ a : x_1\ \}\ ) \\
\{\ b : q_{R;C,v_1}(x_1)\ \} & \longrightarrow & q_{R;B,v_1}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ a : q_{R;C,v_1}(x_1),\ b : q_{R;C,v_1}(x_2)\ \} & \longrightarrow & q_{R;B,v_1}\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_{R;C,v_1}\ (\ \{\ a : \{\ c\ \}\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_{R;C,v_1}\ (\ \{\ b : \{\ c\ \}\ \}\ ) \\
\{\ a,\ b\ \} & \longrightarrow & q_{R;C,v_1}\ (\ \{\ a : \{\ c\ \}\ ,\ b : \{\ c\ \}\ \}\ ) \\
\{\ \} & \longrightarrow & q_{R;A,v_2}\ (\ \{\ \}\ ) \\
\{\ a : q_{R;B,v_2}(x_1)\ \} & \longrightarrow & q_{R;A,v_2}\ (\ \{\ a : x_1\ \}\ ) \\
\{\ b : q_{R;B,v_2}(x_1)\ \} & \longrightarrow & q_{R;A,v_2}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ a : q_{R;B,v_2}(x_1),\ b : q_{R;B,v_2}(x_2)\ \} & \longrightarrow & q_{R;A,v_2}\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a : q_{R;C,v_2}(x_1)\ \} & \longrightarrow & q_{R;B,v_2}\ (\ \{\ a : x_1\ \}\ ) \\
\{\ b : q_{R;C,v_2}(x_1)\ \} & \longrightarrow & q_{R;B,v_2}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ a : q_{R;C,v_2}(x_1),\ b : q_{R;C,v_2}(x_2)\ \} & \longrightarrow & q_{R;B,v_2}\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_{R;C,v_2}\ (\ \{\ a : \{\ d\ \}\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_{R;C,v_2}\ (\ \{\ b : \{\ d\ \}\ \}\ ) \\
\{\ a,\ b\ \} & \longrightarrow & q_{R;C,v_2}\ (\ \{\ a : \{\ d\ \}\ ,\ b : \{\ d\ \}\ \}\ ) \\
\{\ \} & \longrightarrow & q_{R;A,v_3}\ (\ \{\ \}\ ) \\
\{\ a : q_{R;B,v_3}(x_1)\ \} & \longrightarrow & q_{R;A,v_3}\ (\ \{\ a : x_1\ \}\ ) \\
\{\ b : q_{R;B,v_3}(x_1)\ \} & \longrightarrow & q_{R;A,v_3}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ a : q_{R;B,v_3}(x_1),\ b : q_{R;B,v_3}(x_2)\ \} & \longrightarrow & q_{R;A,v_3}\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a : q_{R;C,v_3}(x_1)\ \} & \longrightarrow & q_{R;B,v_3}\ (\ \{\ a : x_1\ \}\ ) \\
\{\ b : q_{R;C,v_3}(x_1)\ \} & \longrightarrow & q_{R;B,v_3}\ (\ \{\ b : x_1\ \}\ ) \\
\{\ a : q_{R;C,v_3}(x_1),\ b : q_{R;C,v_3}(x_2)\ \} & \longrightarrow & q_{R;B,v_3}\ (\ \{\ a : x_1,\ b : x_2\ \}\ ) \\
\{\ a\ \} & \longrightarrow & q_{R;C,v_3}\ (\ \{\ a : \{\ c,\ d\ \}\ \}\ ) \\
\{\ b\ \} & \longrightarrow & q_{R;C,v_3}\ (\ \{\ b : \{\ c,\ d\ \}\ \}\ ) \\
\{\ a,\ b\ \} & \longrightarrow & q_{R;C,v_3}\ (\ \{\ a : \{\ c,\ d\ \}\ ,\ b : \{\ c,\ d\ \}\ \}\ )
\end{array}
$$

Diese enorme Anzahl von Regeln beschreibt das in Abb. 5.13 angegebene kartesische Produkt. □

Der Automat $\mathcal{T}$, der sich aus der Kodierung ergibt, ist kontextbezogen deterministisch. Der inverse Automat $\mathcal{T}^{-1}$ besitzt ebenfalls diese Eigenschaft, da die Annotationen der Ausgabe wieder aus der Eingabeseite übernommen werden. Mit dieser Kodierung beschreibt ein kartesisches Produkt eine informationserhaltende Sicht.

Selektion, Projektion und kartesisches Produkt können jeweils durch einen Baumautomaten mit Ausgabe kodiert werden. Relationale Anfragen, die durch mehrere dieser Operatoren gebildet werden, werden durch das Hintereinanderschalten der entsprechenden Automaten formuliert. Hierbei wird aus dem Zielschema des einen Automaten das Quellschema des nachfolgenden Automaten. Eventuell sind hierbei noch Umformungen notwendig, um ein Schema in die gewünschte Form zu bringen.

Es gilt

$$f(\mathbf{I}) = \mathbf{J} \Leftrightarrow f_{\mathcal{T}}(enc(\mathbf{I})) = enc(\mathbf{J})$$

wobei f ein Ausdruck der SPC-Algebra und $enc(f) = f_{\mathcal{T}}$.

5.4.4 Änderungsoperationen

In einem letzten Schritt sind für die Änderungsoperatoren aus dem relationalen Modell die Kodierungen zu definieren. Ist eine Relation als Baum gegeben, können die Änderungsoperatoren aus dem relationalen Datenmodell, wie das Einfügen oder Löschen von Tupeln, entsprechend auf Operationen über Bäume umgesetzt werden. Da ein Tupel einem Pfad entspricht, werden diese Operationen als Einfügen oder Löschen von Pfaden in bzw. aus einem Baum definiert. Hierbei besitzen der Baum der kodierten Relation und die Pfade immer dieselbe Tiefe, da im Pfad für alle im Baum kodierten Attribute Werte angegeben sein müssen.

Ist eine Änderungsoperation $ins(R, t)$ gegeben, wird durch diese zu der Menge der Tupel der Relation R ein einzelnes Tupel t hinzugefügt. Es werden somit zwei Tupelmengen vereinigt. Diese Vereinigung ist auf Baumseite durch den *union*-Operator gegeben, der in Abschnitt 3.1.4 definiert wurde. Es ergibt sich

$$enc(ins(R, t)) = union(enc(R), \{\ R : enc(t)\ \}).$$

Beim einzufügenden Tupel t wird in der Baumdarstellung die Relation R mit angegeben, in die das Tupel eingefügt werden soll.

Für das Entfernen eines Tupels durch $del(R, t)$ wird eine Kodierung festgelegt, welche die Differenz der Pfade von zwei Bäumen berechnet, wobei vom Baum, der die Relation kodiert, der Baum des zu entfernenden Tupels abgezogen wird. Die Kodierung eines relationalen Löschoperators ergibt sich als

$$enc(del(R, t)) = diff(enc(R), \{\ R : enc(t)\ \}).$$

Ein Ersetzen eines Tupels $mod(R, t_1 \rightarrow t_2)$ kann durch das Entfernen des Tupels t_1 und das Einfügen von t_2 erreicht werden.

Ob durch eine Operation wieder eine gültige Instanz eines Schemas entsteht, wird hierbei nicht ausgesagt. Soll überprüft werden, ob dem so ist, muss der Ergebniswert vom zum Schema gehörenden Automaten akzeptiert werden. Wenn das Ergebnis der Operation vom Automaten akzeptiert wird, dann gehört es zum Schema und erfüllt gleichzeitig dessen Konsistenzbedingungen.

Beispiel 5.8 (*Operationen auf Bäumen*)
Ist die Instanz aus Beispiel 5.2 mit den beiden Relationen R und S gegeben, so können die hier definierten Einfügeoperatoren als Operationen auf Bäumen kodiert werden. Ein $ins(R, \langle\ \text{b, b, b}\ \rangle)$ wird kodiert als $union(v_1, \{\ R : \{\ b : \{\ b : \{\ b\ \}\ \}\ \}\ \}\ )$. Abb. 5.14 zeigt das Ergebnis dieser Operation.

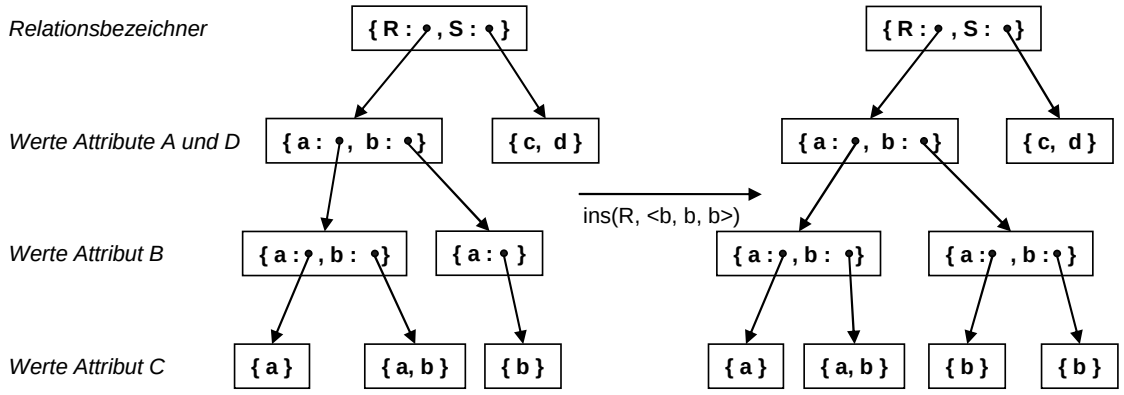


Abbildung 5.14: *Kodierung einer Einfügeoperation*

□

Operationen auf Bäumen mit normalen Annotationen können auf Bäume mit Komplement erweitert werden, indem die Definitionen der Operatoren übernommen werden. Allerdings ist zu beachten, dass die Markierungen der Annotationen durch C entsprechend angepasst werden, um die Eindeutigkeit der Annotationssymbole auf einer Ebene des Ergebniswerts herzustellen. Wird in eine Ebene eine Annotation ohne Markierung eingefügt, die bereits im Komplement vorhanden ist, dann wird deren Markierung als Komplement entfernt.

Bei der Kodierung eines Einfügens oder Löschens eines Tupels auf einem Baum mit Komplement ist zu beachten, dass der Fall eintreten kann, dass die relationale Operation nicht alle Attributwerte beschreibt, um einen Pfad der notwendigen Länge zu erhalten. Eine Operation $ins(R, \langle\ \text{c}\ \rangle)$ ist auf dem Anwendungsschema bei der Projektion $\pi_B(R)$ aus Beispiel 5.6 definiert. Als Ergebnis einer Projektion auf dem angegebenen Beispiel-

wert entsteht der Wert mit Komplement v, wie er in Abb. 5.15 links zu sehen ist. Die Einfügeoperation beschreibt nur einen Attributwert, den für das Attribut B, also einen Pfad der Länge eins, der Teilbaum für die Relation besitzt aber drei Schichten. Die bei solchen Operationen fehlenden Werte können berechnet werden, da die Domänen der fehlenden Attribute bekannt sind. Das einzufügende Tupel wird mit Werten aus den Domänen ergänzt, wobei deren Annotationen zum Komplement zugeschlagen werden. Eine Kodierung für ein $ins(R, \langle\ c\ \rangle)$ kann als $union(v, \{\ R : \{\ a^C : \{\ c : \{\ a^C\ \}\ \}\ \}\ )$ festgelegt werden, wobei sich als Ergebnis der Operation der in Abb. 5.15 angegebene Wert ergibt. Es sind auch andere Kodierungen möglich, welche jeweils andere Werte aus der Domäne der Attribute verwenden.

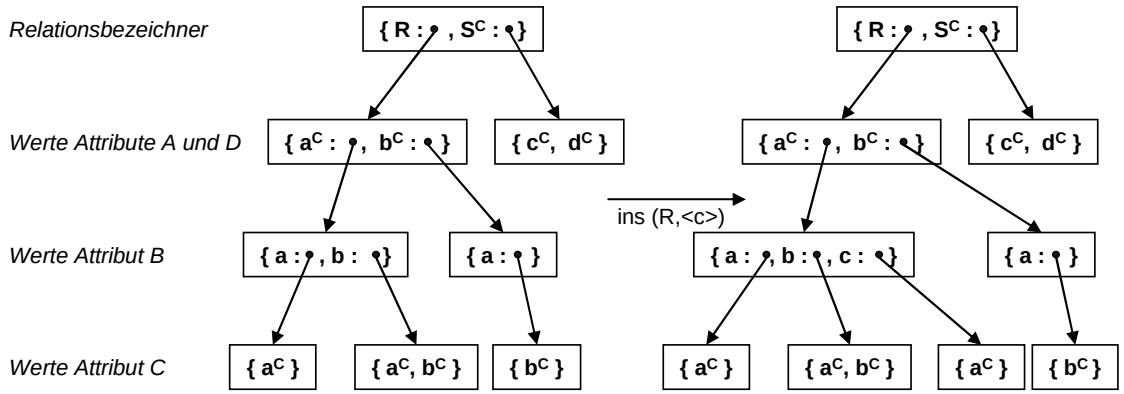


Abbildung 5.15: *Einfügen bei Werten mit Komplement*

Es gilt

$$\nu(\mathbf{J_1}) := \mathbf{J_2} \Leftrightarrow enc(\nu)(enc(\mathbf{J_1})) = enc(\mathbf{J_2})$$

wobei ν aus *ins*, *del* und *mod* Operatoren besteht.

5.5 Bestimmung semantisch korrekter Operationen

Die Abschnitte 5.4.2, 5.4.3 und 5.4.4 zeigen, dass eine relationale Sicht auf Basis eines kodierten relationalen Schemas so als Baumautomat dargestellt werden kann, dass für alle Operatoren der SPC-Algebra eine informationserhaltende Sicht entsteht. Dies bedeutet, dass jede Änderungsoperation auf der Anwendungsseite zwischen zwei Instanzen des Zielschemas semantisch korrekt ist. Ist folglich ein beliebiges relationales Schema mit oder ohne funktionale Abhängigkeiten und Inklusionsabhängigkeiten gegeben und wird hierauf eine Sicht als Ausdruck der SPC-Algebra definiert, dann können die semantisch korrekten Anwendungsoperationen bestimmt werden.

Die Vorgehensweise zur Lösungsfindung beim View-Update-Problem folgt der Beschreibung in Abschnitt 5.4. Ein gegebenes relationales Datenbankschema wird als Baumautomat kodiert, wobei die ebenfalls gegebene Sicht die Ausgabe des Baumautomaten festlegt. Das Komplement der Sicht wird aus der Regelmenge abgeleitet. Ist eine Sichtinstanz $\mathbf{J_1} = f_{\mathcal{T}}(\mathbf{I_1})$ geben, kann man diese mit einer Änderungsoperation in eine andere Instanz $\mathbf{J_2}$ umwandeln. Diese neue Instanz wird entweder vom inversen Automaten der Abbildung akzeptiert oder sie wird von ihm abgelehnt und kann somit nicht durchgeführt werden. Wird die Instanz $\mathbf{J_2}$ akzeptiert, kann mit der immer vorhandenen Umkehrabbildung $f_{\mathcal{T}^{-1}}$ die Instanz $\mathbf{I_2} = \mu(\mathbf{I_2})$ der Basis errechnet werden. Die auf der Datenbank durchzuführende Operation μ, um von $\mathbf{I_1}$ nach $\mathbf{I_2}$ zu gelangen, ergibt sich als Hinzufügen der Tupel $diff(\mathbf{I_2},\mathbf{I_1})$ und dem Entfernen der Tupel $diff(\mathbf{I_1},\mathbf{I_2})$.

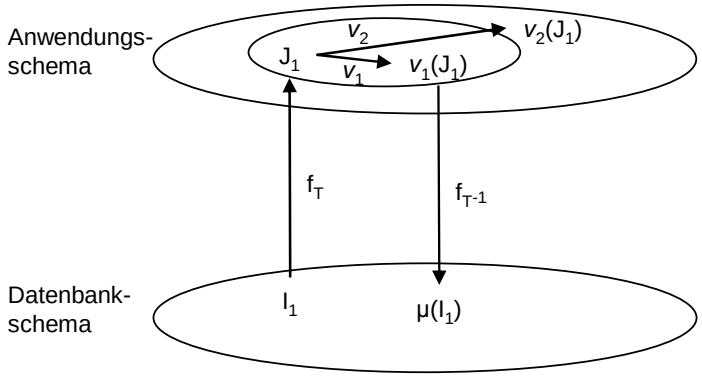


Abbildung 5.16: *Bei einer SPC-Sicht sind alle Operationen ν_1, die das Zielschema erreichen, durchführbar; alle anderen Operationen ν_2 nicht*

5.6 Beispiel für relationale Sichten

Dieses Kapitel schließt mit einem Beispiel, das aufzeigt, wie mit dem vorgestellten Lösungsverfahren das View-Update-Problem behandelt wird. Hierzu wird eine Sicht in eine Darstellung als Baumtransformation überführt und angegebene Änderungsoperationen werden auf Umsetzbarkeit getestet.

Beispiel 5.9 (*Selektion*)

Ein relationales Schema **S** mit dem Relationstyp

$$Buch[ISBN, Titel, Jahr] \text{ mit } ISBN \rightarrow Titel$$

stellt das Datenbankschema und modelliert Bücher. Die funktionale Abhängigkeit bringt zum Ausdruck, dass zu einer gegebenen ISBN eines Buchs der Titel immer eindeutig ist.

Ein Buch kann in mehreren Auflagen über unterschiedliche Jahre verteilt herausgegeben werden, so dass zum Attribut *Jahr* keine Abhängigkeit besteht. Somit sind keine einattributigen Schlüsselattribute vorhanden. Eine zugehörige Datenbankinstanz **I** besteht aus einer Relation und soll ein Tupel enthalten:

ISBN	Titel	Jahr
0-201-53781-X	Database theory	1997

Eine Abbildung bzw. Sicht wählt die nach dem Jahr 2000 erschienenen Bücher aus und ist als

$$V := \sigma_{Jahr \geq 2000}(Buch)$$

definiert. Als Anwendungsschema **T** wird

$$Buch_V[ISBN, Titel, Jahr]$$

abgeleitet.

Auf der Sicht sollen nun die Einfügeoperationen

$$\nu_1 = ins(Buch_V, \langle \text{ 3-544-685458-3, Einsatz von Datenbanken, 1997 } \rangle$$
$$\nu_2 = ins(Buch_V, \langle \text{ 0-201-53781-X, Einsatz von Datenbanken, 2001 } \rangle$$
$$\nu_3 = ins(Buch_V, \langle \text{ 3-544-685458-3, Einsatz von Datenbanken, 2001 } \rangle$$

auf $f(\mathbf{I})$ ausgeführt werden, wobei sich die Frage stellt, ob diese überhaupt durchgeführt werden können, und wenn sie durchgeführt werden können, wie diese umzusetzen sind (Abb. 5.17).

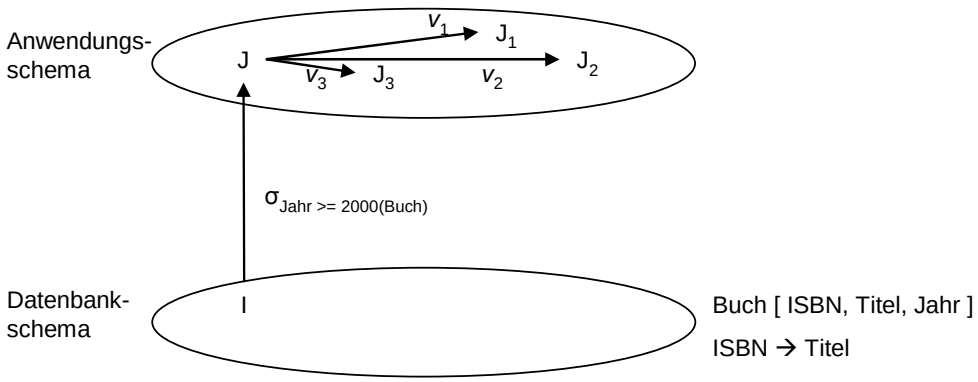


Abbildung 5.17: *Ausgangssituation*

Im Folgenden wird eine Reduktion auf einen Baumautomaten durchgeführt und die Fragestellung daran untersucht.

Definitionsbereich der Abbildung kodieren

Die Reduktion beginnt mit der Überführung des relationalen Schemas und der funktionalen Abhängigkeit in einen Baumautomaten gemäß der Beschreibung aus Abschnitt 5.4.2. Für den Relationstyp $Buch\,[\,ISBN, Titel, Jahr\,]$ entsteht der Automat $\mathcal{A}_{Buch;ISBN\rightarrow Titel} = (Q, \Sigma, q_f, \Delta)$ mit den Übergangsregeln $\Delta = \{$

$$
\begin{array}{ll}
\{\ Buch : q_{ISBN}\ \} & \longrightarrow\ q_f \\
\{\ \} & \longrightarrow\ q_{ISBN} \\
\{\ 0-201-53781-X : q_{Titel}\ \} & \longrightarrow\ q_{ISBN} \\
\{\ 3-544-685458-3 : q_{Titel}\ \} & \longrightarrow\ q_{ISBN} \\
\{\ 0-201-53781-X : q_{Titel}, 3-544-685458-3 : q_{Titel}\ \} & \longrightarrow\ q_{ISBN} \\
 & \ldots \\
\{\ Database\ theory : q_{Jahr}\ \} & \longrightarrow\ q_{Titel} \\
\{\ Einsatz\ von\ Datenbanken : q_{Jahr}\ \} & \longrightarrow\ q_{Titel} \\
 & \ldots \\
\{\ 1997\ \} & \longrightarrow\ q_{Jahr} \\
\{\ 2001\ \} & \longrightarrow\ q_{Jahr} \\
\{\ 1997, 2001\ \} & \longrightarrow\ q_{Jahr} \\
 & \ldots
\end{array}
$$

$\}$. Für die drei Attribute wird je ein Zustand gebildet und die Werte der zugehörigen Attributdomänen legen den Aufbau der Regeln fest. In der Darstellung sind nur die Regeln angegeben, die zu den Werten der Instanz gehören und nicht alle Regeln für die kompletten Attributdomänen. Die Abhängigkeit sagt aus, dass wenn ein Wert für die ISBN vorhanden ist, dann auch nur ein Wert für den Titel erlaubt ist. Im Automaten bedeutet dies, dass die Werte der Regeln, die in q_{Titel} enden, nur einen Titel sprich eine Annotation besitzen. Es ist $enc(\mathbf{S}) = \mathcal{A}_{Buch;ISBN\rightarrow Titel}$.

Definition der Abbildung kodieren

Im nächsten Schritt wird die gegebene Abbildung $\sigma_{Jahr\geq 2000}(Buch)$ auf Basis des erzeugten Baumautomaten $\mathcal{A}_{Buch;ISBN\rightarrow Titel}$ als Quellschema kodiert. Die Abbildung beschreibt eine Selektion auf dem Attribut $Jahr$, so dass gemäß Abschnitt 5.4.3 der Wertebereich des zugehörigen Zustands q_{Jahr} in zwei neue Zustände $q_{Jahr;Jahr\geq 2000}$ und $q_{Jahr;Jahr\ngeq 2000}$ aufgeteilt wird.

Werden gleichzeitig die Regeln mit den passenden Ausgabewerten und Komplementen nach Abschnitt 5.5 ergänzt, entsteht ein Baumautomat mit Ausgabe $\mathcal{T}_{\sigma_{Jahr\geq 2000}} = (Q, \Sigma, q_f, \Delta)$ und $\Delta = \{$

$\{\ Buch : q_{ISBN}(x_1)\ \}$
$\quad \longrightarrow q_f\ (\ \{\ Buch : x_1\ \}\ )$
$\{\ Buch : q_{ISBN;Jahr\geq 2000}\ (x_1)\ \}$
$\quad \longrightarrow q_f\ (\ \{\ Buch : x_1\ \}\ )$
$\{\ Buch : q_{ISBN;Jahr\not\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_f\ (\ \{\ Buch : x_1\ \}\ )$
$\{\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\geq 2000}\ (\ \{\ \}\ )$
$\{\ 0-201-53781-X : q_{Titel;Jahr\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\geq 2000}\ (\ \{\ 0-201-53781-X : x_1\ \}\ )$
$\{\ 0-201-53781-X : q_{Titel;Jahr\not\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\not\geq 2000}\ (\ \{\ 0-201-53781-X^C : x_1\ \}\ )$
$\{\ 3-544-685458-3 : q_{Titel;Jahr\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\geq 2000}\ (\ \{\ 3-544-685458-3 : x_1\ \}\ )$
$\{\ 3-544-685458-3 : q_{Titel;Jahr\not\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\not\geq 2000}\ (\ \{\ 3-544-685458-3^C : x_1\ \}\ )$
$\{\ 0-201-53781-X : q_{Titel;Jahr\geq 2000}(x_1),\ 3-544-685458-3 : q_{Titel;Jahr\geq 2000}(x_2)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\geq 2000}\ (\ \{\ 0-201-53781-X : x_1,\ 3-544-685458-3 : x_2\ \}\ )$
$\{\ 0-201-53781-X : q_{Titel;Jahr\geq 2000}(x_1),\ 3-544-685458-3 : q_{Titel;Jahr\not\geq 2000}(x_2)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\geq 2000}\ (\ \{\ 0-201-53781-X : x_1,\ 3-544-685458-3^C : x_2\ \}\ )$
$\{\ 0-201-53781-X : q_{Titel;Jahr\not\geq 2000}(x_1),\ 3-544-685458-3 : q_{Titel;Jahr\geq 2000}(x_2)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\geq 2000}\ (\ \{\ 0-201-53781-X^C : x_1,\ 3-544-685458-3 : x_2\ \}\ )$
$\{\ 0-201-53781-X : q_{Titel;Jahr\not\geq 2000}(x_1),\ 3-544-685458-3 : q_{Titel;Jahr\not\geq 2000}(x_2)\ \}$
$\quad \longrightarrow q_{ISBN;Jahr\not\geq 2000}\ (\ \{\ 0-201-53781-X^C : x_1,\ 3-544-685458-3^C : x_2\ \}\ )$
...
$\{\ Database\ theory : q_{Jahr;Jahr\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{Titel;Jahr\geq 2000}\ (\ \{\ Database\ theory : x_1\ \}\ )$
$\{\ Database\ theory : q_{Jahr;Jahr\not\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{Titel;Jahr\not\geq 2000}\ (\ \{\ Database\ theory^C : x_1\ \}\ )$
$\{\ Einsatz\ von\ Datenbanken : q_{Jahr;Jahr\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{Titel;Jahr\geq 2000}\ (\ \{\ Einsatz\ von\ Datenbanken : x_1\ \}\ )$
$\{\ Einsatz\ von\ Datenbanken : q_{Jahr;Jahr\not\geq 2000}(x_1)\ \}$
$\quad \longrightarrow q_{Titel;Jahr\not\geq 2000}\ (\ \{\ Einsatz\ von\ Datenbanken^C : x_1\ \}\ )$

...
$\{\ 1997\ \}$
$\quad \longrightarrow q_{Jahr;Jahr\not\geq 2000}\ (\ \{\ 1997^C\ \}\ )$
$\{\ 2001\ \}$
$\quad \longrightarrow q_{Jahr;Jahr\geq 2000}\ (\ \{\ 2001\ \}\ )$
$\{\ 1997,\ 2001\ \}$
$\quad \longrightarrow q_{Jahr;Jahr\geq 2000}\ (\ \{\ 1997^C,\ 2001\ \}\ )$
...

}.

Somit ist $enc(\sigma_{Jahr \geq 2000}(Buch)) = \mathcal{T}_{\sigma_{Jahr \geq 2000}}$. Der inverse Automat $\mathcal{T}^{-1}_{\sigma_{Jahr \geq 2000}}$ ergibt sich durch das Vertauschen von Ein- und Ausgabeseite bei jeder Regel.

Abbildung berechnen

Ist die Abbildung als Baumautomat kodiert, kann die Datenbankinstanz mit seiner Hilfe abgebildet werden. Es wird die Instanz $\mathbf{J} = f(\mathbf{I})$ berechnet, die Ausgangspunkt der Operationen ist. Eine Kodierung der gegebenen Relation *Buch* mit einem Tupel ergibt einen Baum als $enc(Buch) = \mathbf{I}$ mit

$$\mathbf{I} := \{\ Buch : \{\ 0 - 201 - 53781 - X : \{\ Database\ theory : \{\ 1997\ \}\ \}\ \}\ \}$$

Wird $\mathbf{I}$ vom Automaten $\mathcal{T}_{\sigma_{Jahr \geq 2000}}$ akzeptiert, wird als Ausgabe eine Anwendungsinstanz der Form

$$\mathbf{J} := \{\ Buch : \{\ 0 - 201 - 53781 - X^C : \{\ Database\ theory^C : \{\ 1997^C\ \}\ \}\ \}\ \}$$

erzeugt. Es gilt also $f_{\mathcal{T}_{\sigma_{Jahr \geq 2000}}}(\mathbf{I}) = \mathbf{J}$.

Da alle Annotationen bis auf den Relationsbezeichner mit einem C markiert sind, entspricht diese Ausgabe, wenn das Komplement wegfällt, der leeren Relation auf Sichtseite.

Einfügeoperation auf Durchführbarkeit testen

Als nächster Schritt werden die Einfügeoperationen auf Basis der Anwendungsinstanz $\mathbf{J}$ kodiert.

- Die erste Einfügeoperation erzeugt $\nu_1(\mathbf{J}) := \mathbf{J_1}$ mit Hilfe von $union(\mathbf{J}, \{\ Buch : \{\ 3 - 544 - 685458 - 3 : \{\ Einsatz\ von\ Datenbanken : \{\ 1997\ \}\ \}\ \}\ \}\ )$ mit

  $$\mathbf{J} = \{\ Buch : \{\ 0 - 201 - 53781 - X^C : \{\ Database\ theory^C : \{\ 1997^C\ \}\ \}\ \}\ \}$$

  als

  $$\mathbf{J_1} = \{\ Buch : \{\ 0 - 201 - 53781 - X^C : \{\ Database\ theory^C : \{\ 1997^C\ \}\ \}\ , 3 - 544 - 685458 - 3 : \{\ Einsatz\ von\ Datenbanken : \{\ 1997\ \}\ \}\ \}\ \}$$

  Es gilt zu testen, ob die sich ergebende Instanz $\mathbf{J_1}$ von $\mathcal{T}^{-1}_{\sigma_{Jahr \geq 2000}}$ akzeptiert wird. $\mathbf{J_1}$ wird abgelehnt, da es keine Regel gibt, die den Teilwert $\{\ 1997\ \}$ annimmt. Somit ist die Operation ν_1 nicht durchführbar. Ein solcher Wert wird von der Selektionsbedingung der Sicht herausgefiltert.

- Die zweite Einfügeoperation $\nu_2(\mathbf{J}) := \mathbf{J_2}$ erzeugt aus

  $$\mathbf{J} = \{\ Buch : \{\ 0 - 201 - 53781 - X^C : \{\ Database\ theory^C : \{\ 1997^C\ \}\ \}\ \}\ \}$$

mit Hilfe von $union(\mathbf{J}, \{\ Buch : \{\ 0-201-53781-X : \{\ Einsatz\ von\ Datenbanken : \{\ 2001\ \}\ \}\ \}\ \}\ )$ den Wert

$$\mathbf{J_2} = \{\ Buch : \{\ 0-201-53781-X : \{\ Database\ theory^C : \{\ 1997^C\ \}\ ,\ Einsatz\ von\ Datenbanken : \{\ 2001\ \}\ \}\ \}\ \}$$

Auch der Wert $\mathbf{J_2}$ wird vom inversen Automaten nicht akzeptiert, so dass dessen Durchführung nicht möglich ist. Es existiert keine Regel, die zwei Werte auf der Ebene des Titel-Attributs annimmt.

- Die dritte Operation erzeugt $\nu_3(\mathbf{J}) = \mathbf{J_3}$ mit

$$\mathbf{J} = \{\ Buch : \{\ 0-201-53781-X^C : \{\ Database\ theory^C : \{\ 1997^C\ \}\ \}\ \}\ \}$$

und $union(\mathbf{J},\ \{\ Buch : \{\ 3-544-685458-3 : \{\ Einsatz\ von\ Datenbanken : \{\ 2001\ \}\ \}\ \}\ \}\ )$

$$\mathbf{J_3} = \{\ Buch : \{\ 0-201-53781-X^C : \{\ Database\ theory^C : \{\ 1997^C\ \}\ \}\ ,3-544-685458-3 : \{\ Einsatz\ von\ Datenbanken : \{\ 2001\ \}\ \}\ \}\ \}$$

Der Wert $\mathbf{J_3}$ wird vom inversen Automat akzeptiert und generiert hierbei die Ausgabe

$$\mathbf{I_3} = \{\ Buch : \{\ 0-201-53781-X : \{\ Database\ theory : \{\ 1997\ \}\ \}\ ,\ 3-544-685458-3 : \{\ Einsatz\ von\ Datenbanken : \{\ 2001\ \}\ \}\ \}\ \}$$

als neue Datenbankinstanz.

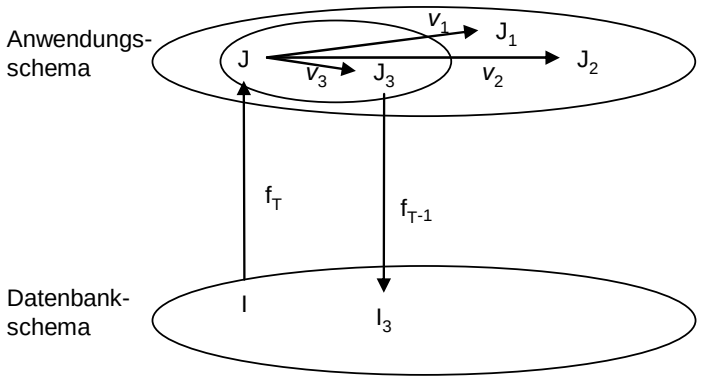


Abbildung 5.18: *Lösung des View-Update-Problems*

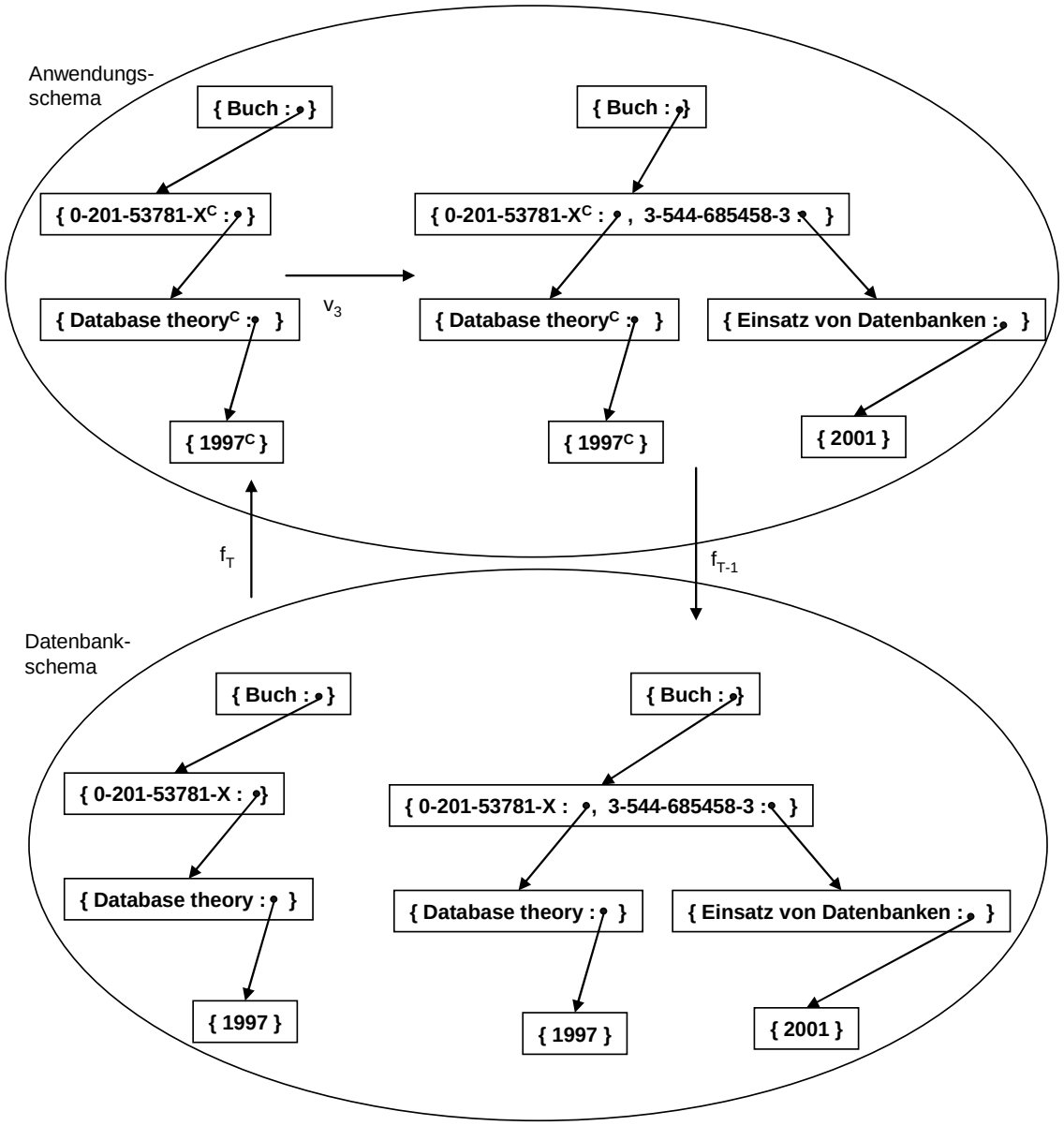


Abbildung 5.19: *Ausführung von* ν_3

Folglich können die ersten beiden Operationen ν_1 und ν_2 nicht durchgeführt werden und die dritte Operation ν_3 ist durchführbar und liefert die neue Datenbankinstanz $\mathbf{I_3}$ (Abb. 5.18 und 5.19).

□

5.7 Fazit

Sichten werden im relationalen Modell durch einen Ausdruck der relationalen Algebra definiert. Sollen Änderungsoperationen auf einer Sicht durchgeführt werden, gilt es geeignete Umsetzungen für diese Operationen auf dem Datenbankschema zu finden. Eine Umsetzung muss nicht immer existieren oder auch eine Vielzahl möglicher Umsetzungen kann vorhanden sein.

Es existiert eine Kodierung für eine durch die SPC-Algebra formulierte Sicht f als Baumautomat mit Ausgabe und die aus der Kodierung resultierende Abbildung $f_{\mathcal{T}}$ ist eine informationserhaltende Sicht, wenn neben der eigentlichen Ausgabe das Komplement ergänzt wird. Dies bedeutet, dass bei beliebigen Sichten, die mit Hilfe der SPC-Algebra formuliert werden, für jede Anwendungsoperation bestimmt werden kann, ob sie ausführbar ist, und dass, wenn sie ausführbar ist, eine eindeutige Umsetzung bestimmt werden kann. Dies gilt dann für alle Sichten über allen relationalen Schemata, auch beim Vorhandensein von Konsistenzbedingungen und ohne jegliche Vorbedingungen. Insgesamt betrachtet, zeigt sich, dass das View-Update-Problem – geeignet dargestellt – für alle Sichten der SPC-Algebra gelöst werden kann.

Da bei jeder Abbildung ein Zielschema vorhanden ist, kann nicht nur für einzelne Operationen überprüft werden, ob sie semantisch korrekt sind, sondern es kann auch eine Charakterisierung für die durch die Sicht implizierte Menge von semantisch korrekten Operationen gegeben werden. Dies sind all diejenigen Operationen, die von einer Instanz des Zielschemas zu einer anderen Instanz des Zielschemas führen.

6 Persistentes Speichern von Objekten

Sind objektorientierte Anwendungsdaten in einem relationalen Datenbanksystem gespeichert, wird eine Abbildung zwischen einem relationalen und einem objektorientierten Schema notwendig. Beim Object/Relational-Mapping-Problem liegt eine Schwierigkeit in der Überwindung der semantischen Heterogenität, da beide Datenmodelle unterschiedliche Ausdrucksmächtigkeiten besitzen. Ziel bei einer Abbildungsdefinition ist es, sicherzustellen, dass alle auftretenden Programmzustände auch gespeichert werden können, d.h. dass auf dem Anwendungsschema jede definierte Operation semantisch korrekt durchführbar ist.

Als ein zweiter Anwendungsfall wird in diesem Kapitel auf Basis einer formalen Beschreibung eines objektorientierten Modells das O/R-Mapping-Problem auf eine Abbildung zwischen Bäumen reduziert. Wird eine solche Abbildung als Baumabbildung formuliert, ist diese dann zur Speicherung der Programmzustände in einer Datenbank geeignet, wenn sie eine informationserhaltende Sicht beschreibt.

Abschnitt 6.1 legt die Notation für ein objektorientiertes Datenmodell fest, so dass in Abschnitt 6.2 das O/R-Mapping-Problem formuliert werden kann. Abschnitt 6.3 stellt einige existierende Verfahren zur Speicherung von Objektdaten vor. Die Formulierung einer Abbildung als Baumtransformation folgt in Abschnitt 6.4 und eine Bestimmung der Abbildungseigenschaften in Abschnitt 6.5. Im anschließenden Abschnitt 6.6 wird ein Beispiel für objektrelationale Abbildungen aufgezeigt, dem ein Fazit in 6.7 folgt.

6.1 Objektorientiertes Datenmodell

Bei einer Reduktion wird ein objektorientiertes Schema in einen Baumautomat überführt, so dass eine formale Beschreibung eines objektorientierten Modells benötigt wird, auf dessen Grundlage die Reduktion definiert wird.

Während im Bereich der Datenhaltung das relationale Modell eine weite Verbreitung gefunden hat, folgen heutige Anwendungsprogramme zumeist einem objektorientierten Programmiermodell, welches auch die Beschreibung der vom Programm verwendeten Daten mit umfasst. Im Gegensatz zum relationalen Modell existiert im objektorientierten Bereich kein Standardmodell. Vielmehr gibt es eine Vielzahl unterschiedlicher objektorientierter Datenmodelle [ABD+89, D+90, KMWZ91, SLR+92, A+95, AS95, DD95, KLW95, RS96, Clu98, CB00], die in ihren Grundmerkmalen übereinstimmen, sich aber doch in Details unterscheiden. Eine Reihe derartiger Merkmale, die ein objektorientiertes Modell ausmachen, werden in [ABD+89] zusammengefasst. Ruht das relationale

Modell auf einer mathematischen Beschreibung, fehlt mit einigen Ausnahmen wie etwa [KLW95] den meisten objektorientierten Modellen eine formale Semantik, so dass sie sich als Grundlage für eine Reduktion auf eine Baumdarstellung nicht eignen.

Das im Folgende beschriebene Modell orientiert sich an [ABD+89], [Clu98] und [AS95]. Allerdings finden nur die Zustandsdaten von Objekten Beachtung. Der funktionale Aspekt in Form von Objektmethoden wird nicht weiter berücksichtigt, da bei einer Abbildung des O/R-Mapping-Problems nur die Zustandsdaten von Objekten abgebildet werden, nicht aber deren Methoden. Das vorgestellte Modell verwendet eindeutige Objektbezeichner und erlaubt neben atomaren Datentypen auch zusammengesetzte Datentypen bei Objektattributen, wie etwa Listen oder Reihungen. Ein objektorientiertes Schema definiert eine Menge von Klassen, welche eine Klassenhierarchie bilden können, wobei jedes Objekt zu genau einer Klasse gehört.

6.1.1 Werte

Auf der Seite der Werte finden sich sowohl atomare als auch zusammengesetzte Werte sowie Objekte. Die Menge der Konstanten **const** wird durch die Vereinigung einer endlichen Menge von paarweise disjunkten Domänen **integer**, **string**, **float**, etc. gebildet. Des Weiteren gibt es eine Menge von *Objektbezeichnern* bzw. Objektidentifikatoren **oid**, eine Menge von *Klassenbezeichnern* **class** und eine Menge von *Attributbezeichnern* **att**. Zusätzlich existiert eine spezielle Konstante **nil**, die den „unbekannten“ Wert ausdrückt.

Ist eine Menge $O \subseteq \mathbf{oid}$ von Objektbezeichnern gegeben, dann lässt sich die Menge der Werte über O, $\mathbf{val}(O)$, durch die folgenden Regeln aufbauen:

- Jeder Wert $v \in \mathbf{const}$ ist ein atomarer Wert.
- Jeder Objektbezeichner $oid \in O$ und **nil** sind Werte.
- Sind $v_1, \ldots, v_n$ unterschiedliche Werte, dann ist die Menge $\{v_1, \ldots, v_n\}$ ein zusammengesetzter Wert.
- Sind $v_1, \ldots, v_n$ Werte und $\{a_1, \ldots, a_n\} \subseteq \mathbf{att}$, dann ist das Tupel $\{a_1 : v_1, \ldots, a_n : v_n\}$ ein zusammengesetzter Wert.

Ein *Objekt* ist ein Paar (oid, v), wobei $oid \in O$ ein Objektbezeichner und v ein Tupel $\{a_1 : v_1, \ldots, a_n : v_n\}$ aus $\mathbf{val}(O)$ ist. oid bildet den eindeutigen Bezeichner des Objekts und v repräsentiert die Zustandsdaten des Objekts. Die paarweise verschiedenen Attributbezeichner a_i, die innerhalb des Tupels v verwendet werden, sind die *Attribute* des Objekts und die durch sie adressierten Werte v_i die entsprechenden *Attributwerte*.

Beispiel 6.1 (*Objekte*)
Eine Menge von Objekten über den Objektbezeichnern $O = \{oid1, oid2, oid3, oid4, oid5, oid6, oid7\}$ ergibt sich etwa als

$(oid1, \{ISBN : \text{0-201-53781-X}, Titel : \text{'Database theory'}, Jahr : 1997\})$
$(oid2, \{ISBN : \text{1-55867-622-X}, Titel : \text{'Data and the Web'}, Jahr : 2001\})$
$(oid3, \{Autorenname : \text{'S. Maier'}, Institutsname : \{oid6, oid7\}, ISBN : \{oid1\}\})$
$(oid4, \{Autorenname : \text{'P. Hill'}, Institutsname : \{oid6, oid7\}, ISBN : \{oid2\}\})$
$(oid5, \{Autorenname : \text{'R. Scholl'}, Institutsname : \{oid8\}, ISBN : \{oid1\}\})$
$(oid6, \{Institutsname : \text{'I.N.R.I.A.'}, Ortsname : \text{'Rocquencourt'}\})$
$(oid7, \{Institutsname : \text{'I.N.R.I.A.'}, Ortsname : \text{'Rennes'}\})$
$(oid8, \{Institutsname : \text{'Bell Labs'}, Ortsname : \text{'Murray Hill'}\})$

□

6.1.2 Schema

Ein objektorientiertes Schema beschreibt durch die Angabe von Klassen und deren Struktur eine Menge von gültigen Schemainstanzen. Ist eine endliche Menge von Klassenbezeichnern C gegeben, dann wird die Menge der *Typbezeichner*, $\mathbf{types}(C)$, wie folgt gebildet:

- **integer**, **string**, **float** etc. sind Typbezeichner.
- Alle Klassenbezeichner aus C sind Typbezeichner.
- Ist τ ein Typbezeichner, dann ist $set\langle\tau\rangle$ ein Typbezeichner.
- Ist τ ein Typbezeichner und $n \geq 0$, dann ist $sequence\langle\tau, n\rangle$ ein Typbezeichner.
- Ist τ ein Typbezeichner und $n \geq 0$, dann ist $array\langle\tau, n\rangle$ ein Typbezeichner.
- Sind $\tau_1, \ldots, \tau_n$ Typbezeichner, $\{a_1, \ldots, a_n\} \subseteq \mathbf{att}$ und $n > 0$, dann ist $struct\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle$ ein Typbezeichner.
- Sind $\tau_1, \ldots, \tau_n$ Typbezeichner, $\{a_1, \ldots, a_n\} \subseteq \mathbf{att}$ und $n > 0$, dann ist $union\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle$ ein Typbezeichner.

Das Modell unterstützt neben den üblichen atomaren Typen und den Klassen auch zusammengesetzte Typen wie Mengen (*set*), die paarweise verschiedene Werte enthalten. Eine Folge (*sequence*) beschreibt eine geordnete Mehrfachmenge von maximal n Elementen, bei der sich die enthaltenen Werte nicht zwingend unterscheiden müssen. Eine geordnete Mehrfachmenge mit einer festen Elementanzahl n wird durch eine Reihung (*array*) modelliert, wobei die enthaltenen Werte anhand ihrer Position innerhalb der Menge angesprochen werden. Werden die Elemente einer Mehrfachmenge nicht über die Position, sondern über eindeutige Attributbezeichner adressiert, erhält man statt der Reihung einen Verbund (*struct*). Bei einem variablen Verbund (*union*) ist jeweils nur ein Wert gültig, der über einen der angegebenen Attributbezeichner ausgewählt wird.

Innerhalb einer Menge von Klassen C kann eine Beziehung zwischen den Klassen angegeben werden, die beschreibt, wann ein Objekt der einen Klasse anstelle eines Objekts der anderen Klasse verwendet werden darf. Eine *Klassenhierarchie* ist ein Tupel $(C, \sigma, \pi, \prec)$, wobei C eine endliche Menge von Klassenbezeichnern, σ eine Abbildung von C in die Menge der $struct\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle$ aus $\mathbf{types}(C)$, π eine Abbildung von C auf Teilmengen der Objektbezeichner und $\prec$ eine partielle Ordnung auf C ist. Die Menge C legt somit die in der Hierarchie vorhandenen Klassen fest, die Funktion σ weist jeder Klasse ihre Attribute zu, die Objektbezeichnerzuweisung π teilt die Objektbezeichner auf die Klassen auf und die Ordnung $\prec$ definiert eine Unterklassenbeziehung zwischen Klassen. Gilt $c \prec c'$, dann ist c eine *Oberklasse* von c' und entsprechend ist c' eine *Unterklasse* von c.

Definition 6.1 (*Untertypbeziehung*)
Sei $(C, \sigma, \pi, \prec)$ eine Klassenhierarchie. Eine *Untertypbeziehung* über $\mathbf{types}$(C) ist die kleinste partielle Ordnung $\leq$ über $\mathbf{types}$(C), so dass:

- wenn $c \prec c'$, dann $c \leq c'$,
- wenn $\tau_i = \tau_i'$ für alle $i \in \{1, \ldots, n\}$ und $n \leq m$, dann
  $struct\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle, \leq struct\langle a_1 : \tau_1', \ldots, a_n : \tau_n', \ldots, a_m : \tau_m'\rangle$

Eine Klassenhierarchie $(C, \sigma, \pi, \prec)$ ist *wohlgeformt*, falls für jedes Klassenpaar c und c' aus C aus der Beziehung $c \prec c'$ folgt, dass $\sigma(c) \leq \sigma(c')$ gilt, d.h. ist c' als eine Unterklasse einer anderen Klasse c festgelegt, so müssen die Attribute, die in der Oberklasse verwendet werden, auch in der Unterklasse vorhanden sein. Zudem muss $\pi(c_1) \cap \pi(c_2) = \emptyset$ gelten für alle $c_1 \neq c_2$ aus C. Somit wird jedes Objekt eindeutig über seinen Bezeichner einer Klasse zugeordnet. Eine wohlgeformte Klassenhierarchie $(C, \sigma, \pi, \prec)$ bildet ein objektorientiertes *Schema*.

Die Semantik eines Schemas bzw. die Semantik der Typbezeichner legt fest, welche Werte bei einer Angabe eines bestimmten Typbezeichners innerhalb eines Schemas bei einer Instanz des Schemas erlaubt sind. Die Interpretation der Domänen der Typbezeichner erfolgt basierend auf einem Schema $(C, \sigma, \pi, \prec)$:

- Für die atomaren Typen τ ist $dom(\tau)$ jeweils die gewöhnliche Interpretation als Menge von Zahlen, Zeichen, etc.
- Ein Klassenbezeichner $c \in C$ erlaubt Verweise auf Objekte dieser Klasse, auf Objekte aller Unterklassen der Klasse sowie die **nil**-Referenz und somit $dom(c) = \pi^*(c) \cup \{\mathbf{nil}\}$, wobei $\pi^*(c) = \{\pi(c') \mid c' \in c, c \prec c'\}$ neben den Bezeichnern der Klasse c auch die Bezeichner aller Unterklassen c' einschließt.
- Eine Menge $set\langle\tau\rangle$ erlaubt als Werte alle Teilmengen der Domäne des enthaltenen Typs und somit $dom(set\langle\tau\rangle) =$
  $$\{\{v_1, \ldots, v_n\} \mid n \geq 0 \wedge v_i \in dom(\tau), i \in \{1, \ldots, n\}\},$$

- Eine Folge $sequence\langle\tau, n\rangle$ beschreibt maximal n beliebige Werte aus der Domäne von τ, so dass $dom(sequence\langle\tau, n\rangle) =$
$$\{\{0 : v_1, \ldots, m : v_{m+1}\} \mid 0 \leq m < n \wedge v_i \in dom(\tau), i \in \{1, \ldots, m+1\}\},$$
- Eine Reihung $array\langle\tau, n\rangle$ beschreibt genau n beliebige Werte aus der Domäne von τ, so dass $dom(array\langle\tau, n\rangle) =$
$$\{\{0 : v_1, \ldots, m : v_{m+1}\} \mid m = n-1 \wedge v_i \in dom(\tau), i \in \{1, \ldots, m+1\}\},$$
- Ein Verbund $struct\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle$ adressiert eine Menge von Werten über die angegebenen Attributbezeichner und somit $dom(struct\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle) =$
$$\{\{a_1 : v_1, \ldots, a_n : v_n\} \mid v_i \in dom(\tau_i), i \in \{1, \ldots, n\}\},$$
- Bei einem variablen Verbund $union\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle)$ ist nur ein Wert über einen der Attributbezeichner zugreifbar und so $dom(union\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle) =$
$$\{\{a_i : v_i\} \mid v_i \in dom(\tau_i), i \in \{1, \ldots, n\}\}$$

Eine Instanz eines objektorientierten Schemas kombiniert Objektbezeichner und Werte zu Objekten, wobei die Werte diejenigen Attribute enthalten, welche durch die Klasse des Objekts vorgegeben sind.

Definition 6.2 (*Instanz eines Schemas*)
Eine *Instanz* eines Schemas $(C, \sigma, \pi, \prec)$ mit $O = \{\pi(c) \mid c \in C\}$ ist eine partielle Funktion ν, wobei ν ein $o \in O$ auf einen Wert aus $\mathbf{val}(O)$ des korrekten Typs abbildet, d.h. es gilt $\nu(o) \in dom(\sigma(c))$ für $c \in C$ und $o \in \pi(c)$.

Innerhalb einer Instanz bildet die Funktion ν Objektbezeichner aus O auf Werte ab. Werden nur die zu einer Klasse gehörigen Bezeichner betrachtet, die auch abgebildet werden, ergibt sich die *Extension* dieser Klasse als $\nu_c = \{o \mid \nu(o) \in \mathbf{val}(O) \wedge o \in \pi(c)\}$. Die Extension enthält diejenigen Bezeichner, zu denen auch tatsächlich Objekte existieren.

Beispiel 6.2 (*Objektorientiertes Schema*)
Ein objektorientiertes Schema beschreibt Bücher, Autoren und Institute jeweils als Objekte. Die zugehörigen Klassen geben die Objektstruktur in Form von Attributen vor, so dass ein Buch eine ISBN, einen Buchtitel und ein Erscheinungsjahr besitzt. Ein Autor enthält dessen Name und als Verweise auf andere Objekte jeweils die Institute, an denen er tätig ist, und die von ihm verfassten Bücher. Ein Institut wird durch den Name und den Standort beschrieben.

Das zugehörige objektorientierte Schema ergibt sich als $(C, \sigma, \pi, \prec)$ mit

$$C = \{Buch, Autor, Institut\}$$

$$\sigma(Buch) = struct\langle ISBN : string, Titel : string, Jahr : integer\rangle$$
$$\sigma(Autor) = struct\langle Autorenname : string, Institutsname : set\langle Institut\rangle,$$
$$ISBN : set\langle Buch\rangle\rangle$$

$\sigma(Institut) = struct\langle Institutsname : string, Ortsname : string\rangle$
$\pi(Buch) = \{oid1, oid2\}$
$\pi(Autor) = \{oid3, oid4, oid5\}$
$\pi(Institut) = \{oid6, oid7, oid8\}$

Eine gültige Instanz $\mathbf{I} = (\nu)$ für dieses Schemas definiert etwa die Objektmenge aus Beispiel 6.1 mit $(o, \nu(o))$ für jedes der dort angegebenen Objekte. □

Im Folgenden wird eine vereinfachte Notation verwendet, um Klassen und Objekte darzustellen. Für die Klasse `Buch` aus dem Beispiel erfolgt die Darstellung als

```
class Buch {
     attribute ISBN string;
     attribute Titel string;
     attribute Jahr integer;
};
```

wobei der Klassenname und die Klassenattribute sowie deren Typen angegeben werden. Ein Objekt dieser Klasse wird durch

```
object oid1 : Buch {
     ISBN = 0-201-53781-X;
     Titel = 'Database theory';
     Jahr = 1997;
};
```

repräsentiert mit Angabe des Objektbezeichners, der Klasse des Objekts, sowie den Klassenattributen und deren Werten.

6.1.3 Abbildungen

Für die Definition von Abbildungen innerhalb eines objektorientierten Modells wird häufig eine Erweiterung der relationalen Algebra eingesetzt, welche die Navigation zwischen Objekten über Verweise erlaubt bzw. den Aufruf von Objektmethoden in Anfragen gestattet. Beispiele für solche Sprachen sind O_2SQL [BDK92] oder OQL [CB00].

Beim O/R-Mapping-Problem werden allerdings Abbildungen zwischen einem relationalen und einem objektorientierten Schema untersucht, so dass für die weiteren Betrachtungen keine Abbildungsbeschreibung innerhalb des objektorientierten Modells benötigt wird.

6.1.4 Operationen

Um Änderungsoperationen in einem objektorientierten Modell auszudrücken, werden üblicherweise den Klassen entsprechende Methoden zugewiesen. Deren Implementierungen führen Berechnungen auf den Zustandsdaten von Objekten durch und greifen hierfür

auf die Attributwerte von Objekten lesend und schreibend zu. Um die Betrachtung zu vereinfachen, werden im Folgenden nur Schreibzugriffe auf einzelne Attributwerte als Änderungsoperationen auf einem objektorientierten Schema angesehen. Hierdurch erfolgt keine Einschränkung, da komplexere Methodenimplementierungen auf eine Folge von Lese- und Schreiboperationen auf Attributen herunter gebrochen werden können.

Neben dem Ändern von Attributwerten zählt auch das Anlegen oder Löschen von Objekten zu den Änderungsoperationen auf einem Schema. Wird ein Objekt gelöscht, stellt sich die Frage, wie mit existierenden Verweisen auf dieses Objekt umgegangen wird. Das vorgestellte objektorientierte Modell garantiert nicht, dass ein Objekt, auf das verwiesen wird, auch tatsächlich vorhanden ist. Verweise können ins Leere führen.

6.2 Object/Relational-Mapping-Problem

Eine Abbildungsbeschreibung umfasst beim Object/Relational-Mapping-Problem ein relationales Datenbankschema und ein objektorientiertes Anwendungsschema. Eine Instanz $\mathbf{I_1}$ des Datenbankschemas besteht aus einer Menge von Relationen und eine Instanz $\mathbf{J_1}$ des Anwendungsschemas enthält Mengen von Objekten. Zur Beschreibung einer Abbildung f zwischen beiden Schemata wird zumeist eine propritätere Sprache verwendet, die sich je nach verwendetem Ansatz oder eingesetztem Werkzeug unterscheidet. Die Operationen ν auf der objektorientierten Seite sind Lese- bzw. Schreibzugriffe auf einzelne Objektattribute oder gestatten das Anlegen und Löschen von Objekten. Auf Seite der Datenbank sind die Änderungsoperatoren *ins*, *del* und *mod* zum Einfügen, Löschen oder Ändern von Tupeln als Operationen gültig.

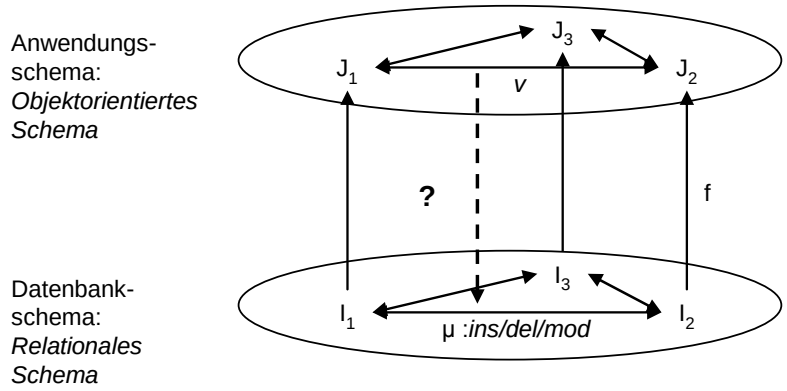


Abbildung 6.1: *O/R-Mapping-Problem: Finde für alle Anwendungsoperationen ν je eine geeignete Umsetzung in Form einer Operation μ*

Ist eine Abbildung f gegeben, so muss bei dieser sichergestellt sein, dass das Anwendungsprogramm jede auf dem Anwendungsschema definierte Operation tatsächlich

wie definiert durchführen kann. Die Anwendungsdaten können in jeden durch das Anwendungsschema erlaubten Zustand auch irgendwann einmal wechseln. Es muss folglich garantiert werden, dass für jede Änderungsoperation eine passende Umsetzung auf dem Datenbankschema vorhanden ist (Abb. 6.1). Jede der Anwendungsoperationen muss semantisch korrekt sein. Es ergibt sich das *O/R-Mapping-Problem*.

Definition 6.3 (*O/R-Mapping-Problem*)
Ist eine Abbildung $f : dom(\mathbf{S}) \rightarrow dom(\mathbf{T})$ gegeben. Für alle Änderungsoperationen ν auf allen $\mathbf{J} \in dom(\mathbf{T})$ finde je eine Umsetzung μ, so dass $\nu(f(\mathbf{I_1})) = f(\mu(\mathbf{I_1}))$ mit $\mathbf{J} = f(\mathbf{I_1})$ gilt.

Im Vergleich zum View-Update-Problem wird nicht nur für eine Operation auf dem Anwendungsschema die semantische Korrektheit gefordert, sondern für alle dort definierten Operationen. Eine Operation kann hierbei bei einer beliebigen Instanz des Anwendungsschemas starten. Eine notwendige Voraussetzung für die Korrektheit aller Operationen ist, dass sämtliche Instanzen des Anwendungsschemas auch im Bildbereich der Abbildung vorkommen, damit alle Operationen seiteneffektfrei sind. Folglich muss das Zielschema einer Abbildung f mit dem Anwendungsschema übereinstimmen, so dass f surjektiv sein muss. Somit existieren keine Instanzen des Anwendungsschemas außerhalb des Zielschemas. Da alle Operationen semantisch korrekt durchführbar sein sollen, bedeutet dies außerdem, dass alle Operationen eindeutig sein müssen und somit die Abbildung f injektiv sein muss. Es darf keine Anwendungsinstanz existieren, auf die zwei Datenbankinstanzen abgebildet werden. Insgesamt betrachtet muss f also bijektiv sein, so dass bei einer für die Problemstellung geeigneten Abbildung daher immer eine Umkehrabbildung f^{-1} existiert. Die Bijektivität von f kann auch so interpretiert werden, dass ein Programmzustand eindeutig als eine Datenbankinstanz gespeichert und der gespeicherte Zustand wieder eindeutig aus der Datenbank ausgelesen werden kann. Dies entspricht einem persistenten Speichern von Programmzuständen.

6.3 Verwandte Arbeiten

Wird eine Abbildung zwischen zwei Schemata unterschiedlicher Datenmodelle definiert, stellt sich zunächst die Frage, wie mit der unterschiedlichen Ausdrucksmächtigkeit der Modelle umgegangen wird. So besitzt ein objektorientiertes Modell das Konzept der Objektidentifikatoren, welches im relationalen Modell nicht vorhanden ist. Bei den meisten existierenden Ansätzen ist das Ziel, solche Konzepte der einen Seite auf ähnliche Konzepte auf der anderen Seite abzubilden, so dass sich etwa ein Objektbezeichner als Schlüsselwert in einer Relation wieder findet.

Bisherige Untersuchungen können in zwei Gruppen eingeteilt werden, je nachdem wie die Festlegung der Umkehrabbildung zur eigentlichen Abbildung erfolgt. Zum einen gibt es Ansätze, bei denen ein Anwender sowohl die Abbildung als auch die dazugehörige Umkehrabbildung beide getrennt spezifiziert, so dass zum Auslesen der Daten eine Ab-

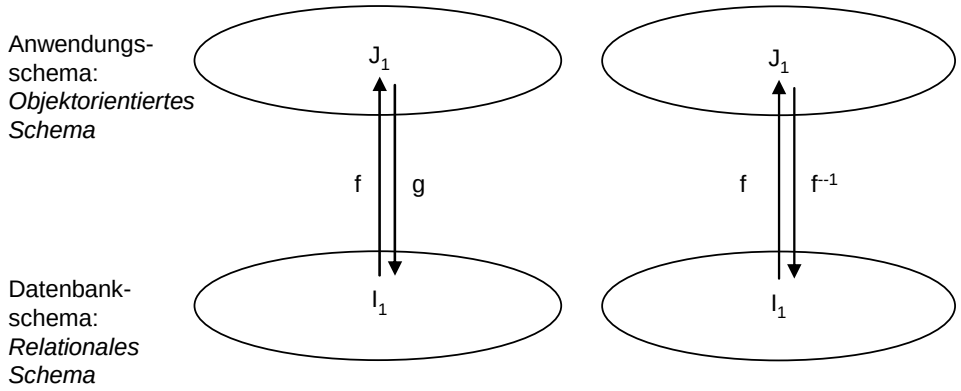


Abbildung 6.2: *Eine Umkehrabbildung g zu f wird entweder separat angegeben oder es wird automatisch ein f^{-1} bestimmt*

bildung f und zur Speicherung eine weitere Abbildung g verwendet wird (Abb. 6.2 links). Im zweiten Fall wird nur für eine der beiden Richtungen eine Abbildung f angegeben und die Gegenrichtung f^{-1} selbst durch das System bestimmt (Abb. 6.2 rechts).

6.3.1 Benutzerdefinierte Umkehrabbildungen

Ein Ansatz, welcher benutzerdefinierte Umkehrabbildungen verwendet, findet sich etwa im Produktdatenbereich innerhalb des EXPRESS-Datenmodells [SHT+77]. Ein Schema in EXPRESS beschreibt eine Menge von Objektklassen mit ihren Attributen, wobei Hierarchien von Objektklassen aufgebaut werden können. Neben dem eigentlichen Datenmodell wird eine imperative Programmiersprache zur Verfügung gestellt, mit der auf den Datenbestand eines Schemas zugegriffen werden kann. Hierdurch wird es möglich, Abbildungen mit Hilfe von benutzerdefinierten Funktionen anzulegen, wobei die Implementierungen dieser Funktionen Verzweigungs- und Schleifenanweisungen sowie vom System vordefinierte Funktionen verwenden können. Mittels der Sprachvariante EXPRESS-M (mapping) [VLA95] legt ein Anwender Funktionen an, welche Daten aus einem Datenbankschema auslesen und in Daten eines Anwendungsschemas umwandeln. Naturgemäß erfolgt hier die Abbildung nur in eine Richtung. Die Verwendung von bidirektionalen Abbildungen erlaubt die Sprache EXPRESS-C (conceptual) [VLA95], indem jeweils für die Hin- und die Rückrichtung eine separate, unidirektionale Abbildungsdefinition angegeben wird, so dass nach Änderungsoperationen auf einer Seite die Instanz auf der jeweils anderen Seite durch eine der Abbildungen berechnet werden kann.

Den Ansatz der Trennung von Hin- und Rückrichtung bei der Abbildungsbeschreibung verfolgen auch [HST99] bei der Abbildung von Daten zwischen einem zumeist objektorientierten Anwendungsschema und einem relationalen Datenbankschema. Die Aufteilung der Abbildungsdefinition ist hier allerdings noch weitergehend als beim Ansatz aus

[VLA95], da nicht nur zwischen den beiden Richtungen unterschieden wird, sondern auch je nach Art der Anwendungsoperation eine getrennte Abbildung festgelegt wird. Ein Abbildungsersteller definiert jeweils separat, welche Operationen auf dem Datenbankschema auszuführen sind, wenn auf ein Objekt im Anwendungsschema zum ersten Mal lesend zugegriffen wird, was geschieht, wenn ein Objekt im Anwendungsschema neu angelegt bzw. gelöscht wird, und welche Operationen bei einer Zustandsänderung eines Objekts des Anwendungsschemas auf der Datenbasis durchzuführen sind. Hierbei erlaubt die Abbildungsdefinitionssprache eine Zuordnung von Werten der Relationsattribute zu den Werten von Objektattributen, wobei über Referenzen bzw. Fremdschlüsselbeziehungen navigiert werden kann. Das Verfolgen von Referenzen ist sowohl in Richtung ihrer Definition als auch in die umgekehrte Richtung möglich. Zudem können Daten selektiert oder mengenwertige Objektattribute aufgebaut werden.

Erfolgen wie in den beiden Ansätzen die Definitionen der Abbildung und der Umkehrabbildung getrennt und unter Verwendung einer ausdrucksstarken Abbildungssprache, ist zwar eine Vielzahl von unterschiedlichen Abbildungen definierbar, doch über die Eigenschaften einer Abbildung können keinerlei Aussagen gemacht werden. So ist kein Korrektheitskriterium angegeben, mit dem sich überprüfen lässt, ob die Abbildung, die als Inverses einer Abbildung angegeben wird, tatsächlich die Umkehrabbildung ist. Ob eine Abbildung korrekt spezifiziert ist, d.h. ob sie eine Bijektion darstellt, liegt alleine bei den Fähigkeiten und Vorstellungen eines Abbildungserstellers, dem es auch freisteht, unsinnige Abbildungen zu definieren.

6.3.2 Automatische Umkehrabbildung

Um Objekte in Relationen zu speichern, existieren einige Abbildungsmuster, bei denen jeweils eine Umkehrabbildung vorhanden ist. Bei der Festlegung solcher Muster wird in den meisten Ansätzen bzw. Implementierungen von Abbildungswerkzeugen [CK85, BKSW91, Top93, AKK95, Kel97, BW97, KW00, Hib05, Top05, CC05] ähnlich vorgegangen. Das Hauptaugenmerk liegt darauf, wie objektorientierte Konzepte wie Objektidentifikatoren, Verweise zwischen Objekten und Klassenhierarchien in einem relationalen Schema ausgedrückt werden können.

Eines der Muster besteht darin, jeder Objektklasse einen Relationstyp zuzuweisen und ein Objekt als ein Tupel in der Relation unterzubringen. Der Objektbezeichner wird in einem Schlüsselattribut der Relation gespeichert, dessen Werte entweder künstlich generiert werden oder für das die Werte eines Objektattributs verwendet werden, welches Objekte eindeutig kennzeichnet. Die einzelnen Objektattribute spiegeln sich als Attribute des Relationstyps wider, so dass die Werte eines Tupels einem Objektzustand entsprechen. Für eine solche Abbildung ist eine Umkehrabbildung vorhanden. Abbildung 6.3 zeigt ein Beispiel einer Abbildung zwischen der Klasse `Buch` und dem Relationstyp *Buch*, wobei zwei Objekte der Klasse als Tupel gespeichert sind.

In einigen Ansätzen wie etwa [Top05] können die Attributwerte bei der Abbildung durch eine bijektive Funktion umgerechnet werden. Lauten die Werte der Domäne eines

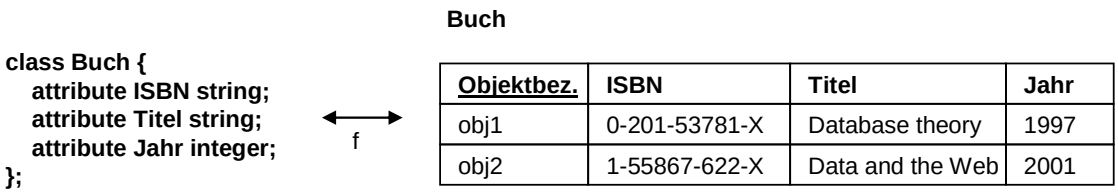


Abbildung 6.3: *Einer Objektklasse wird ein Relationstyp zugewiesen, der die Objekte in Form von Tupeln aufnimmt. Das Schlüsselattribut der Relation ist unterstrichen dargestellt.*

Objektattributes „F“ und „M“ und auf Seite des Relationsattributs „female“ und „male“, kann eine passende Umrechnung definiert werden. Ebenso ist das Serialisieren von Werten möglich, wobei für einen Wert eines Objektattributs eine Bit-Folge berechnet wird, welche als Binärdaten in einem Attribut der Relation abgelegt wird. Eine weitere Variante ist das Aufteilen von Werten eines Attributs auf Werte von zwei Attributen auf der Gegenseite. Sind etwa Datumswerte auf der Anwendungsseite durch eine Domäne `Datum` modelliert, deren Werte sowohl Tag als auch Uhrzeit umfassen, kann ein einzelner Wert beim Speichern in zwei Werte aufgeteilt werden, so dass die Speicherung von Tag und Uhrzeit in getrennten relationalen Attributen erfolgt.

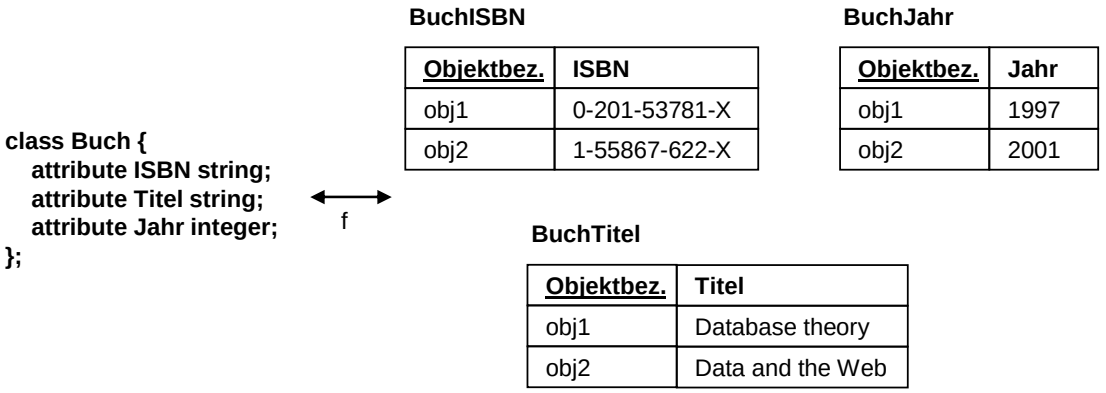


Abbildung 6.4: *Verteilung der Objektattributwerte auf mehrere Relationen*

Ein weiteres Abbildungsmuster besteht darin, den Objektzustand nicht nur in einem Tupel zu hinterlegen, sondern die verschiedenen Attributwerte auf mehrere Tupel zu verteilen. Der Objektbezeichner wird hierbei in jedes Tupel mit übernommen und spielt in jeder der zugewiesenen Relationen die Rolle eines Schlüssels. Im Extremfall kann somit jedes Objektattribut in einer eigenen Relation gespeichert werden (Abb. 6.4).

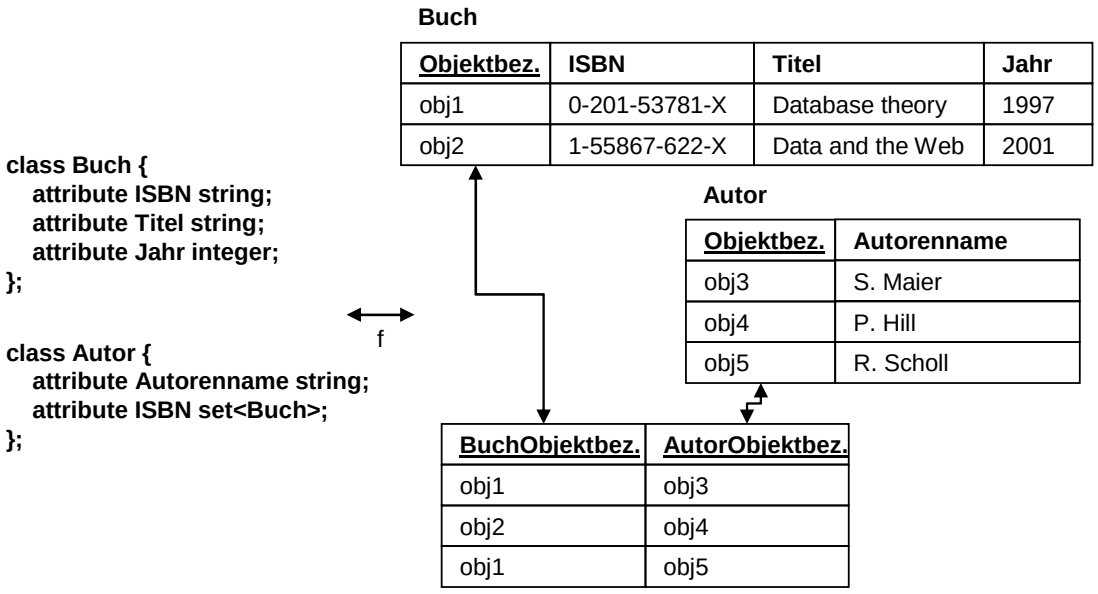


Abbildung 6.5: *Verweise werden über Hilfsrelationen modelliert*

Ein Muster, um Verweise zwischen Objekten zu speichern, besteht darin, im relationalen Schema eine Hilfsrelation anzulegen, welche die Schlüsselwerte der in Beziehung stehenden Objekte als Tupel aufnimmt (Abb. 6.5). Liegen nicht n:m-Beziehungen zwischen Objekten vor, können Vereinfachungen bei der Speicherung vorgenommen werden. Besitzt die Beziehung eine 1:n-Kardinalität, kann bei dem einen Relationstyp ein Fremdschlüsselattribut eingefügt werden, welches die Schlüssel der Objekte enthält, auf die jeweils verwiesen wird. Bei 1:1-Beziehungen können in Beziehung stehende Objekte in einem gemeinsamen Tupel innerhalb einer Relation gespeichert werden.

Bei der Übertragung von Klassenhierarchien auf Relationstypen innerhalb eines relationalen Schemas gibt es drei Vorgehensweisen, die sich darin unterscheiden, in welchem der für die Klassenhierarchie generierten Relationstypen die zu einem Objekt gehörenden Attributwerte untergebracht werden. Bei der horizontalen Partitionierung wird für jede Objektklasse aus der Hierarchie ein Relationstyp angelegt, dessen Attribute diejenigen Objektattribute enthalten, die bei der Objektklasse selbst definiert werden (Abb. 6.6). Im Gegensatz dazu werden bei einer vertikalen Partitionierung alle ererbten und die eigenen Attribute in der Relation der Objektklasse hinterlegt. Als dritten Möglichkeit wird für die komplette Klassenhierarchie nur ein Relationstyp angelegt, der für alle in der Hierarchie vorkommenden Objektattribute jeweils ein relationales Attribut enthält. Einem einzelnen Tupel wird neben dem Objektbezeichner auch der Klassenbezeichner mitgegeben, so dass die Instanzen unterschiedlicher Objektklassen auseinander gehalten werden können. Teilweise sind auch Kombinationen der drei Verfahren innerhalb einer

Hierarchie erlaubt wie etwa [CC05] aufzeigen.

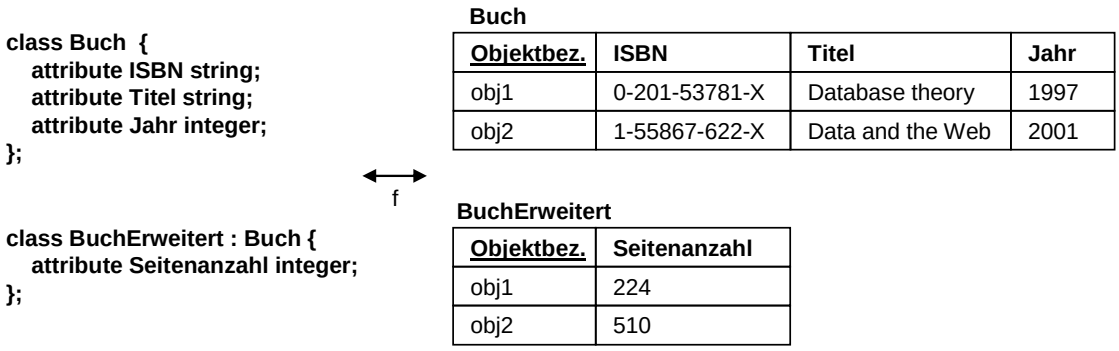


Abbildung 6.6: *Speicherung von Objekten einer Unterklasse* `BuchErweitert` *von* `Buch`

Für die drei Konzepte „Objektidentifikator“, „Verweis zwischen Objekten“ und „Klassenhierarchie“ werden bei den vorgestellten Ansätzen jeweils vorgegebene Abbildungsmuster verwendet, um ein relationales Schema anzugeben, welches die objektorientierten Daten aufnehmen kann. Bei diesen automatischen Abbildungsverfahren findet von vorneherein eine Festlegung auf bestimmte Klassen von Abbildungsdefinitionen statt, so dass etwa Objektklassen immer auf Relationstypen und Verweise zwischen Objekten immer auf Hilfsrelationen abgebildet werden. Andere Abbildungsmöglichkeiten, die von diesen Mustern abweichen, werden nicht betrachtet. Eine Behandlung komplexer Objektattribute wie Listen oder Mengen findet bei den beschriebenen Verfahren nicht statt. Die Werte solcher Attribute werden meist selbst wieder als Objekte aufgefasst und diese mit dem sie enthaltenden Objekt in Beziehung gesetzt. So sind letztendlich nur Objekte mit atomaren Attributwerten innerhalb einer objektorientierten Instanz vorhanden. Diese Vorgehensweise stellt sich aber etwa bei Attributwerten, die lange Listen oder Mengen mit vielen Elementen enthalten, als problematisch heraus, da für jedes Listen- bzw. Mengenelement in einer Relation ein Tupel anzulegen ist. Beim Speichern oder beim Lesen solcher Daten entstehen lange Zugriffszeiten, da dann auf eine große Anzahl von Tupeln zugegriffen werden muss.

6.4 Reduktion des O/R-Mapping-Problems

Bestehende Ansätze geben keinerlei Garantien, ob Abbildung und Umkehrabbildung zusammenpassen, wenn beide von einem Anwender modelliert werden, oder bieten zumeist nur in begrenztem Umfang eine Auswahl an unterschiedlichen Abbildungsmöglichkeiten an. Im Folgenden werden Abbildungsbeschreibungen des O/R-Mapping-Problems durch eine Reduktion auf Baumautomaten untersucht.

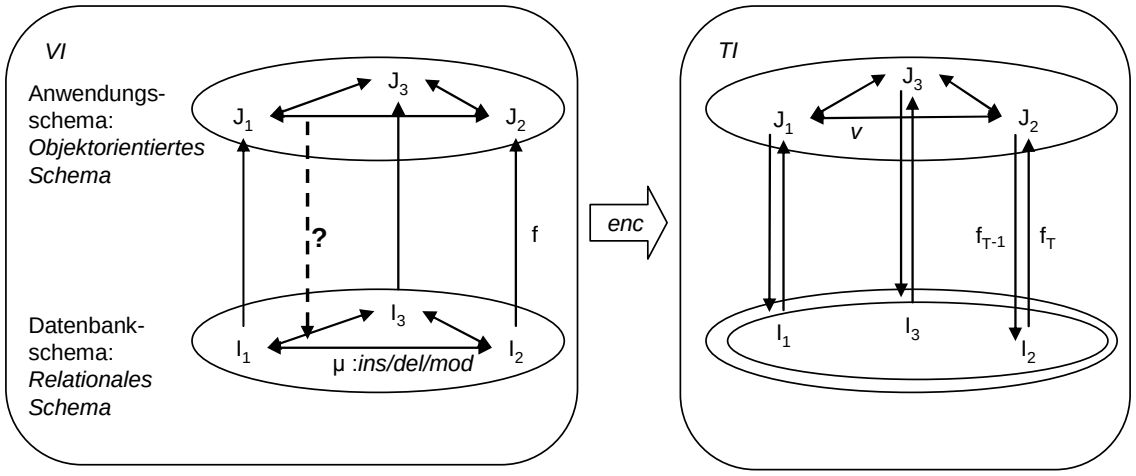


Abbildung 6.7: *Kodierung einer O/R-Abbildung als Baumautomat*

Abb. 6.7 zeigt die einzuschlagende Vorgehensweise. Ist eine Abbildungsbeschreibung VI bestehend aus einem relationalen Datenbankschema, einem objektorientierten Anwendungsschema und einer Abbildung f gegeben, so ist für die Abbildung zu überprüfen, ob sie die Eigenschaften einer Bijektion besitzt. Ist sie bijektiv, stimmen Anwendungs- und Zielschema überein und es ist immer eine Umkehrabbildung f^{-1} vorhanden. Anstatt nun die Abbildung f zu kodieren, kann folglich auch versucht werden, die Umkehrabbildung f^{-1} mit dem Anwendungsschema von f als Definitionsbereich zu kodieren. Die Umkehrabbildung muss hierbei alle Instanzen des objektorientierten Schemas auf Instanzen des relationalen Schemas abbilden. Ist die Umkehrabbildung f^{-1} eine informationserhaltende Sicht, ist sichergestellt, dass die Abbildung f selbst bijektiv ist, da dann alle Instanzen des objektorientierten Schemas erreicht werden und eine Eins-zu-eins-Zuordnung zwischen den Instanzen des Definitions- und des Bildbereichs stattfindet.

Die Abbildungsbeschreibung VI wird durch eine Kodierungsfunktion *enc* in eine Beschreibung mittels eines Baumautomaten überführt. Für das Anwendungsschema als Definitionsbereich einer Umkehrabbildung f^{-1} wird ein Baumautomat $\mathcal{A}_S$ erzeugt und f^{-1} wird als Abbildung basierend auf $\mathcal{A}_S$ kodiert (Abb. 6.8).

Gelingt also die Kodierung eines objektorientierten Schemas als Baumautomat und kann von diesem ausgehend die Umkehrabbildung als informationserhaltende Abbildung in die Kodierung eines relationalen Schemas beschrieben werden, dann ist die eigentliche Abbildung eine bijektive Abbildung vom Datenbank- zum Anwendungsschema. Die betrachteten Abbildungen unterliegen – außer dass sie als Baumabbildungen formulierbar sein müssen – keinen zusätzlich vorgegebenen Einschränkungen. In den folgenden Abschnitten muss also eine Kodierung für objektorientierte Schemata als Baumautomaten gefunden werden. Wird zusätzlich eine Dekodierung eines Baumautomaten gefunden,

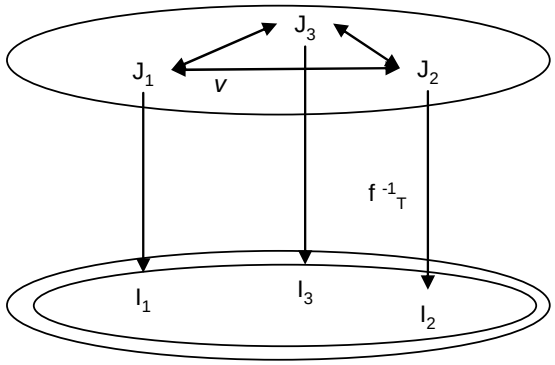


Abbildung 6.8: *Kodierung der Umkehrabbildung*

welche ein relationales Schema erzeugt, ist jede informationserhaltende Sicht zum Speichern geeignet.

Werden gegebene Abbildungen f untersucht, bedeutet dies auch, dass nur eine eingeschränkte Menge von möglichen Abbildungen betrachtet wird, da existierende Abbildungssprachen nur bestimmte Abbildungen zulassen; etwa nur solche, die bestimmte Abbildungsmuster verwenden. Es gibt aber eine Vielzahl weiterer Abbildungen, die dem Zweck des Speicherns von Objektdaten gerecht werden. Neben dem Überprüfen der Eigenschaften von einer gegebenen Abbildung f, besteht auch die Möglichkeit, die Abbildung erst auf Seite der Bäume zu definieren. Da sich jede informationserhaltende Abbildung auf Basis des kodierten objektorientierten Schemas eignet, die Objekte zu speichern, existiert eine große Vielfalt an verwendbaren Abbildungen. Jede solche Abbildung besitzt ein Zielschema. Kann dieses als ein relationales Schema dekodiert werden, ist ein Schema gefunden, welches mindestens alle Datenbankinstanzen, die durch f abgebildet werden, aufnehmen kann.

Als erster Schritt muss die Kodierungsfunktion *enc* für ein objektorientiertes Schema und dessen Instanzen definiert werden.

6.4.1 Objekte als Bäume

Das Kodierungsverfahren für Objekte muss wie im relationalen Fall gewährleisten, dass zum einen jede Instanz eines objektorientierten Schemas als Baum kodiert und dass zum anderen auf Basis dieser Kodierung eine Abbildung definiert werden kann.

Die gewählte Kodierung einer objektorientierten Instanz folgt der impliziten Schachtelungsstruktur der Instanz. Eine Instanz enthält Extensionen von Klassen, eine Extension besteht aus einer Menge von Objekten, ein Objekt enthält eine Menge von Attributen und ein Attribut besitzt einen atomaren oder einen zusammengesetzten Wert. Bei zusammengesetzten Werten wird der Schachtelungsstruktur des Werts gefolgt.

Sei eine Instanz $\mathbf{I} = (\nu)$ eines objektorientierten Schemas $(C, \sigma, \pi, \prec)$ gegeben, dann ist ihre Kodierung als Baum durch die folgende rekursive Definition von $enc(\mathbf{I})$ gegeben:

- Für eine Schemainstanz $\mathbf{I}$ gilt
  $enc(\mathbf{I}) = \{\ c_1 : enc(\nu_{c_1}),\ \ldots,\ c_n : enc(\nu_{c_n})\ \}$ mit $\{c_1, \ldots, c_n\} = C$

  Die Objekte einer Schemainstanz gehören zu Klassen c_i, welche durch das Schema vorgegeben sind. Da im betrachteten objektorientierten Modell ein Objekt nur zu einer Klasse gehört, stehen auf oberster Ebene der kodierten Instanz die Klassenbezeichner.

- Für die Extension ν_c einer Klasse c gilt
  $enc(\nu_c) = \{\ oid_1 : enc(\nu(oid_1)),\ \ldots,\ oid_n : enc(\nu(oid_n))\ \}$
  mit $\{oid_1, \ldots, oid_n\} = \nu_c$

  An die Klassenbezeichner auf oberster Ebene eines Werts schließt sich die Kodierung der Bezeichner der Objekte einer Extension auf der nächsten Ebene an.

- Für ein Tupel $\{a_1 : v_1, \ldots, a_n : v_n\}$ gilt
  $enc(\{a_1 : v_1, \ldots, a_n : v_n\}) = \{\ a_1 : enc(v_1),\ \ldots,\ a_n : enc(v_n)\ \}$

  Objektzustände und Tupelwert werden durch die Angabe ihrer Struktur in Form der Attributbezeichner kodiert. Hinter einem Attributbezeichner wird der zugehörige Attributwert kodiert.

- Für einen Verweis auf ein Objekt mit dem Bezeichner oid gilt
  $enc(oid) = \{\ oid\ \}$ und $enc(\mathbf{nil}) = \{\ nil\ \}$.

  Bei Verweisen auf ein anderes Objekt wird als atomarer Wert der Bezeichner des Objekts, auf das verwiesen wird, angegeben. Der **nil**-Verweis wird entsprechend durch die Annotation nil gekennzeichnet.

- Für eine Menge $\{v_1, \ldots, v_n\}$ gilt
  $enc(\{v_1, \ldots, v_n\}) = \{\ v_1,\ \ldots,\ v_n\ \}$

  Bei Mengen wird für je einen kompletten Wert eines Elements eine einzelne Annotation angelegt.

- Für einen atomaren Wert v gilt
  $enc(v) = \{\ v\ \}$

  Ein atomarer Wert ergibt sich als Annotation.

Da die Kodierungsfunktion für jeden im objektorientierten Modell vorkommenden Wert definiert ist, kann jede Instanz des objektorientierten Modells aus Abschnitt 6.1 kodiert werden. Ein aus einer objektorientierten Instanz erzeugter Baum wird wieder eindeutig dieser Instanz zugeordnet.

Beispiel 6.3 (*Objekte als Bäume*)
Die Instanz $\mathbf{I}$ aus Beispiel 6.1 stellt sich durch $enc(\mathbf{I})$ als Baum kodiert wie in Abb. 6.9

gezeigt dar. Auf oberster Ebene des Baums finden sich die Klassenbezeichner, die innerhalb des Schemas auftreten, woran sich auf der nächsten Ebene die Objektbezeichner anschließen, welche die Extensionen der Klassen modellieren. Die Struktur eines einzelnen Objekts wird durch die Ebene darunter beschrieben, welche die Attributbezeichner enthält. Darunter folgen die kodierten Werte der Attribute.

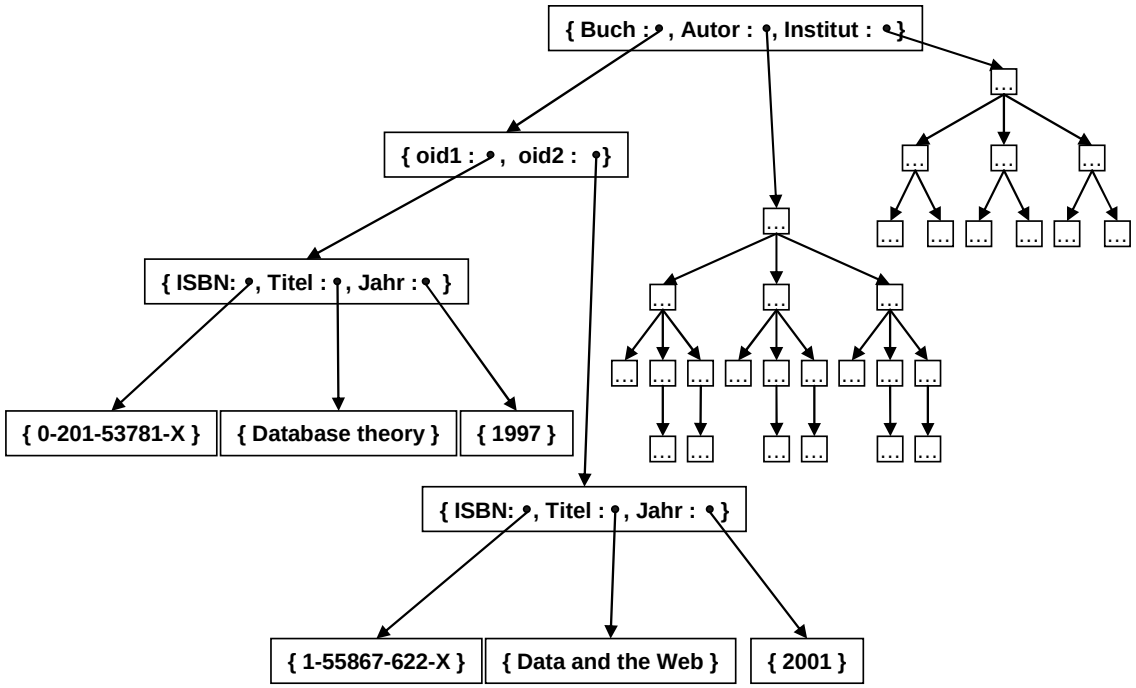


Abbildung 6.9: *Objekte als Baum*

Die Ebenen für die Objekte der Klassen `Autor` und `Institut` sind der besseren Übersichtlichkeit wegen nur verkleinert dargestellt. □

6.4.2 Atomare Typen und Klassen als Baumautomat

Die Kodierung für ein objektorientiertes Schema muss einen Automaten ohne Ausgabe $\mathcal{A}_S$ liefern, so dass einmal jedes Schema als Baumautomat kodiert werden kann und wenn eine Instanz des Schemas gegeben ist, dann die kodierte Instanz vom Automaten akzeptiert wird. Zudem muss das Verfahren einen kontextbezogen deterministischen Automaten liefern, damit nach der Ergänzung der Ausgabewerte eine Abbildung entsteht.

Das Kodierungsverfahren für Schemata orientiert sich an dem Verfahren für die Kodierung einer Instanz. Auf oberster Ebene einer Instanz sind die Klassenbezeichner des Schemas angegeben, so dass der Automat eine Schlussregel

$$\{\ c_1 : q_{\pi(c_1)},\ \ldots,\ c_n : q_{\pi(c_n)}\ \} \longrightarrow q_f$$

mit den Klassenbezeichnern c_i besitzen muss. In einen der Zustände $q_{\pi(c_i)}$ müssen dann alle erlaubten Objektextensionen zu einer Klasse c_i ableitbar sein:

$$enc(\nu_{c_i}) \quad \stackrel{*}{\longrightarrow}_{\mathcal{A}_S} \quad q_{\pi(c_i)}$$

Auf der nächsten Ebene eines Werts folgt eine Kombination aus den Objektbezeichnern der tatsächlich vorhandenen Objekte, welche eine Teilmenge der dieser Klasse zugewiesenen Objektbezeichner sein müssen. Es müssen folglich Regeln der Form:

$$\{\ oid_1 : q_\tau,\ \ldots,\ oid_n : q_\tau\ \} \quad \longrightarrow \quad q_{\pi(c_i)}$$

vorhanden sein, für jede beliebige Kombination von Objektbezeichner $oid_1, \ldots, oid_n$ aus der Menge der für diese Klasse c_i erlaubten Objektbezeichner $\pi(c_i)$. In den Zustand q_τ müssen alle kodierten Objekte der Klasse c_i ableitbar sein. Folgt man der weiteren Schachtelungsstruktur, ergeben sich durch analoge Überlegungen die Regeln für die Kodierung aller erlaubten Objektzustände und Attributwerte.

Ein objektorientiertes Schema wird als Baumautomat kodiert, indem für jeden Wert aus der Domäne eines Typs entsprechende Regeln angelegt werden. Ist ein objektorientiertes Schemas $(C, \sigma, \pi, \prec)$ gegeben, dann erfolgt die Kodierung als Baumautomat durch die folgenden rekursiven Vorschriften. Die Automaten für die einzelnen Typen werden zu einem Automaten für das Schema zusammengesetzt:

- Ist $(C, \sigma, \pi, \prec)$ ein Schema und $enc(\pi(c_i)) = (Q_{\pi(c_i)}, \Sigma_{\pi(c_i)}, q_{\pi(c_i)}, \Delta_{\pi(c_i)})$ für $c_i \in C$, dann $enc((C, \sigma, \pi, \prec)) = (Q, \Sigma, q_f, \Delta)$ mit

  - $Q = \bigcup_i Q_{\pi(c_i)} \cup \{q_f\}$
  - $\Sigma = \bigcup_i \Sigma_{\pi(c_i)} \cup C$
  - $\Delta = \bigcup_i \Delta_{\pi(c_i)} \cup \{\ \{\ c_1 : q_{\pi(c_1)},\ \ldots,\ c_n : q_{\pi(c_n)}\ \} \longrightarrow q_f \mid \{c_1, \ldots, c_n\} = C\}$

  wobei $q_f \notin Q_{\pi(c_i)}$ für alle $\pi(c_i)$.

  Sind $enc(\pi(c_i))$ jeweils die Automaten, welche die Menge von Objekten einer Klasse akzeptieren, dann wird als Schlussregel des Schemaautomaten eine Regel $\{\ c_1 : q_{\pi(c_1)},\ \ldots,\ c_n : q_{\pi(c_n)}\ \} \longrightarrow q_f$ mit den Klassenbezeichnern angelegt. Da immer alle Klassenbezeichner bei einer gültigen Instanz vorhanden sein müssen, ist nur eine Regel erforderlich.

- Ist $\pi(c)$ eine Objektbezeichnerzuweisung und $\sigma(c) = \tau$ mit $enc(\tau) = (Q_\tau, \Sigma_\tau, q_\tau, \Delta_\tau)$, dann $enc(\pi(c)) = (Q_{\pi(c)}, \Sigma_{\pi(c)}, q_{\pi(c)}, \Delta_{\pi(c)})$ mit

  - $Q_{\pi(c)} = Q_\tau \cup \{q_{\pi(c)}\}$
  - $\Sigma_{\pi(c)} = \Sigma_\tau \cup \pi(c)$

- $\Delta_{\pi(c)} = \Delta_\tau \cup \{ \{ oid_1 : q_\tau, \ldots, oid_n : q_\tau \} \longrightarrow q_{\pi(c)} \mid \{oid_1, \ldots, oid_n\} \in I\!P^{fin}(\pi(c))\}$

wobei $q_{\pi(c)} \notin Q_\tau$.

Ist $enc(\tau)$ der Automat, welcher ein Objekt der Klasse c akzeptiert, dann werden zur Akzeptanz einer Menge von Objekten in Form einer Extension alle erlaubten Kombinationen von Objektbezeichnern als Regeln angegeben. Da eine Extension auch leer sein kann, wird eine Regel mit dem leeren Wert angelegt.

- Ist $struct\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle$ ein Verbund und $enc(\tau_i) = (Q_{\tau_i}, \Sigma_{\tau_i}, q_{\tau_i}, \Delta_{\tau_i})$, dann $enc(struct\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle) = (Q_{str}, \Sigma_{str}, q_{str}, \Delta_{str})$ mit

  - $Q_{str} = \bigcup_i Q_{\tau_i} \cup \{q_{str}\}$
  - $\Sigma_{arr} = \bigcup_i \Sigma_{\tau_i} \cup \{a_1, \ldots, a_n\}$
  - $\Delta_{arr} = \bigcup_i \Delta_{\tau_i} \cup \{ \{ a_1 : q_{\tau_1}, \ldots, a_n : q_{\tau_n} \} \longrightarrow q_{str} \}$

  wobei $q_{str} \notin Q_{\tau_i}$ für alle i.

  Sind $enc(\tau_i)$ die Automaten, welche die Werte für die einzelnen in einem Verbund vorkommenden Typen akzeptieren, dann ergibt sich der Automat für den Verbund durch das Hinzufügen einer Regel, welche die Attributbezeichner des Verbunds enthält.

- Ist $union\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle$ ein variabler Verbund und $enc(\tau_i) = (Q_{\tau_i}, \Sigma_{\tau_i}, q_{\tau_i}, \Delta_{\tau_i})$, dann $enc(union\langle a_1 : \tau_1, \ldots, a_n : \tau_n\rangle) = (Q_{uni}, \Sigma_{uni}, q_{uni}, \Delta_{uni})$ mit

  - $Q_{uni} = \bigcup_i Q_{\tau_i} \cup \{q_{uni}\}$
  - $\Sigma_{uni} = \bigcup_i \Sigma_{\tau_i} \cup \{a_1, \ldots, a_n\}$
  - $\Delta_{uni} = \bigcup_i \Delta_{\tau_i} \cup \{ \{ a_i : q_{\tau_i} \} \longrightarrow q_{uni} \mid 1 \leq i \leq n\}$

  wobei $q_{uni} \notin Q_{\tau_i}$ für alle i.

  Sind $enc(\tau_i)$ die Automaten, welche die Werte für die einzelnen in einem variablen Verbund vorkommenden Typen akzeptieren, dann ergibt sich der Automat für den variablen Verbund durch das Hinzufügen je einer Regel für je ein Attribut. Bei einem variablen Verbund ist immer nur über einen Attributbezeichner gleichzeitig ein Wert zugreifbar. Ist ein Wert gegeben, dürfen für die anderen Attributbezeichner keine Werte vorhanden sein.

- Ist $sequence\langle \tau, n\rangle$ ein Folge und $enc(\tau) = (Q_\tau, \Sigma_\tau, q_\tau, \Delta_\tau)$, dann $enc(sequence\langle \tau, n\rangle)$ $= (Q_{seq}, \Sigma_{seq}, q_{seq}, \Delta_{seq})$ mit

  - $Q_{seq} = Q_\tau \cup \{q_{seq}\}$
  - $\Sigma_{seq} = \Sigma_\tau \cup \{0, \ldots, n\}$

- $\Delta_{seq} = \Delta_\tau \cup \{ \{ 0 : q_\tau, \ldots, m : q_\tau \} \longrightarrow q_{seq} \mid 0 \leq m < n\} \cup$
  $\{ \{ \} \longrightarrow q_{seq} \}$

  wobei $q_{seq} \notin Q_\tau$.

  Ist $enc(\tau)$ der Automat, der die Werte für den in der Folge enthaltenen Typ akzeptiert, dann ergibt sich der Automat für die Folge durch das Einfügen von Regeln, welche die Nummern von 0 bis zu maximal $n-1$ enthalten, da eine Folge bis zu n Elemente des angegebenen Typs enthalten darf. Da eine Folge auch keine Elemente enthalten kann, wird eine Regel mit dem leeren Wert auf der linken Seite definiert, um dies auszudrücken.

- Ist $array\langle\tau, n\rangle$ eine Reihung und $enc(\tau) = (Q_\tau, \Sigma_\tau, q_\tau, \Delta_\tau)$, dann $enc(array\langle\tau, n\rangle) = (Q_{arr}, \Sigma_{arr}, q_{arr}, \Delta_{arr})$ mit

  - $Q_{arr} = Q_\tau \cup \{q_{arr}\}$
  - $\Sigma_{arr} = \Sigma_\tau \cup \{0, \ldots, n\}$
  - $\Delta_{arr} = \Delta_\tau \cup \{ \{ 0 : q_\tau, \ldots, m : q_\tau \} \longrightarrow q_{arr} \mid m = n-1\}$

  wobei $q_{arr} \notin Q_\tau$.

  Ist $enc(\tau)$ der Automat, der die Werte für den in der Reihung enthaltenen Typ akzeptiert, dann ergibt sich der Automat für die Reihung durch das Einfügen einer Regel mit den Annotationen 0 bis $n-1$. Bei einer Reihung müssen immer n Elemente vorhanden sein.

- Ist $set\langle\tau\rangle$ eine Menge, dann $enc(set\langle\tau\rangle) = (Q_{set}, \Sigma_{set}, q_{set}, \Delta_{set})$ mit

  - $Q_{set} = \{q_{set}\}$
  - $\Sigma_{set} = dom(\tau)$
  - $\Delta_{set} = \{ \{ v_1, \ldots, v_n \} \longrightarrow q_{set} \mid \{v_1, \ldots, v_n\} \in I\!P^{fin}(dom(\tau)) \wedge n \geq 0\}$

  Bei einer Menge muss gewährleistet sein, dass die enthaltenen Elemente eindeutig sind. Dies wird auch innerhalb der Baumdarstellung garantiert, indem ein komplettes Element der Menge als einzelne Annotation kodiert wird. Da die Eindeutigkeit von Annotationen innerhalb einer Ebene eines Baums sichergestellt wird, ist die Mengeneigenschaft bei den kodierten Werten erfüllt.

- Ist τ ein atomarer Typbezeichner, dann $enc(\tau) = (Q_\tau, \Sigma_\tau, q_\tau, \Delta_\tau)$ mit

  - $Q_\tau = \{q_\tau\}$
  - $\Sigma_\tau = dom(\tau)$
  - $\Delta_\tau = \{ \{ v \} \longrightarrow q_\tau \mid v \in dom(\tau)\}$

  Es wird für jeden Wert der Domäne eine Regel angelegt, welche diesen Wert als Annotation enthält.

- Ist c ein Klassenbezeichner, dann $enc(c) = (Q_c, \Sigma_c, q_c, \Delta_c)$ mit
  - $Q_c = \{q_c\}$
  - $\Sigma_c = \pi^*(c) \cup \{nil\}$
  - $\Delta_c = \{\ \{\ nil\ \} \longrightarrow q_c\ \} \cup$
    $\{\ \{\ oid\ \} \longrightarrow q_c \mid oid \in \pi^*(c)\}$

  Bei Verweisen auf Objekte sind Objekte der angegebenen Klasse erlaubt, so wie Objekte aller Unterklassen und der **nil**-Zeiger.

Betrachtet man die Kodierung eines objektorientierten Schemas insgesamt, finden sich auf oberster Ebene einer Instanz die Klassenbezeichner wieder. Darunter folgen die Objektbezeichner für die vorhandenen Objekte und dann folgt auf der nächsten Ebene die Beschreibung der Struktur der Objekte in Form der Attributbezeichner, woran sich die Kodierung der Attributwerte und deren Struktur anschließt. Um komplexe Objektattribute zu modellieren, können die Ebenen entsprechend tief geschachtelt werden.

Ist eine Klassenhierarchie im zu kodierenden Schema vorhanden, besitzen die Unterklassen alle Attribute ihrer Oberklassen. Dies wird durch die Funktion σ sichergestellt. Bei der Kodierung des Schemas werden folglich bei den Regeln für die Attribute einer Unterklasse automatisch alle Attribute der Oberklassen mit kodiert, da sie im objektorientierten Schema bei der Unterklasse angegeben sind. Bei Verweisen drückt sich die Hierarchie bei der Kodierung des Schemas dadurch aus, dass durch den Baumautomaten neben Verweisen auf Objekte der angegebenen Klasse auch Verweise auf Objekte der Unterklassen erlaubt sind.

Das vorgestellte Kodierungsverfahren wird den gestellten Anforderungen gerecht. Mit Hilfe des Kodierungsverfahrens kann jedes objektorientierte Schema kodiert werden, da für alle Typkonstrukte des objektorientierten Modells die Kodierungsfunktion definiert ist.

Das Kodierungsverfahren erzeugt einen kontextbezogen deterministischen Automaten. Jeder Schritt des Kodierungsverfahrens erzeugt einen Zustand q, wobei nur in diesem Schritt Regeln eingefügt werden, welche diesen Zustand q auf der rechten Seite besitzen. Für alle q im resultierenden Automaten wird also Δ_q durch jeden Schritt vollständig beschrieben. Die Regeln Δ_q jedes Schritt müssen eine Partitionierung von $dom(q)$ liefert, damit der Automat kontextbezogen deterministisch ist. Dies ist bei jedem Schritt der Fall, da sich die Mengen der Annotationen der linken Seiten bei den eingefügten Regeln immer unterscheiden. Ein für ein objektorientiertes Schema erzeugter Automat ist kontextbezogen deterministisch.

Durch das beschriebene Kodierungsverfahren ist gewährleistet, dass jedes objektorientierte Schema als Baumautomat dargestellt werden kann, und, da die Kodierung der Schachtelungsstruktur von Instanzen folgt, gilt bei gegebenem objektorientierten Schema $(C, \sigma, \prec, \pi)$

$$\mathbf{I} \in dom((C, \sigma, \prec, \pi)) \Leftrightarrow enc(\mathbf{I}) \in dom(enc((C, \sigma, \prec, \pi)))$$

Beispiel 6.4 (*Schema als Baumautomat*)
Ist das objektorientierte Schema $(C, \sigma, \prec, \pi)$ aus Beispiel 6.2 gegeben, so erzeugt die Kodierung $enc((C, \sigma, \prec, \pi))$ einen Automaten, der unter anderem die folgenden Übergangsregeln besitzt:

$$
\begin{array}{lcl}
\{\ Buch : q_{BuchExt},\ Autor : q_{AutorExt},\ Institut : q_{InstitutExt}\ \} & \longrightarrow & q_f \\
 & \ldots & \\
\{\ oid1 : q_{Buch},\ oid2 : q_{Buch}\ \} & \longrightarrow & q_{BuchExt} \\
 & \ldots & \\
\{\ ISBN : q_{string},\ Titel : q_{string},\ Jahr : q_{integer}\ \} & \longrightarrow & q_{Buch} \\
 & \ldots & \\
\{\ 0-201-53781-X\ \} & \longrightarrow & q_{string} \\
\{\ 1-55867-622-X\ \} & \longrightarrow & q_{string} \\
 & \ldots & \\
\{\ Database\ theory\ \} & \longrightarrow & q_{string} \\
\{\ Data\ and\ the\ Web\ \} & \longrightarrow & q_{string} \\
 & \ldots & \\
\{\ 1997\ \} & \longrightarrow & q_{integer} \\
\{\ 2001\ \} & \longrightarrow & q_{integer} \\
 & \ldots & \\
\{\ oid3 : q_{Autor},\ oid4 : q_{Autor},\ oid5 : q_{Autor}\ \} & \longrightarrow & q_{AutorExt} \\
 & \ldots & \\
\{\ Autorenname : q_{string},\ Institutname : q_{Institut_{set}},\ ISBN : q_{Buch_{set}}\ \} & \longrightarrow & q_{Autor} \\
 & \ldots & \\
\{\ \} & \longrightarrow & q_{Buch_{set}} \\
\{\ nil\ \} & \longrightarrow & q_{Buch_{set}} \\
\{\ obj1\ \} & \longrightarrow & q_{Buch_{set}} \\
\{\ obj1,\ obj2\ \} & \longrightarrow & q_{Buch_{set}} \\
 & \ldots & \\
\{\ nil,\ obj1\ \} & \longrightarrow & q_{Buch_{set}} \\
 & \ldots &
\end{array}
$$

Die erste der angegebenen Regeln modelliert die im Schema vorhandenen Klassen. In den Zustand $q_{BuchExt}$ werden die in der Instanz vorhandenen kodierten Buchobjekte abgeleitet, so dass dieser Zustand die Objektextension der Klasse `Buch` repräsentiert. Für jede mögliche Kombination von Objektbezeichnern gibt es im Automaten je eine Regel, wobei im Beispiel nur die Regel für die beiden Objekte $oid1$ und $oid2$ angegeben ist. Auf der nächsten Ebene eines Werts sind die Bezeichner der Objektattribute $ISBN$, $Titel$ und $Jahr$ vertreten, die nach q_{Buch} abgeleitet werden. q_{string} und $q_{integer}$ stellen die atomaren Typen **string** und **integer** dar.

Analog erfolgt die Kodierung der Klasse `Autor`. Der Zustand $q_{Buch_{set}}$ steht für das ISBN-Attribut der Klasse `Autor`, das eine Menge von Verweisen auf Buchobjekte modelliert. Zu den Elementen der Menge gehört die leere Menge { } , wenn auf kein Buchobjekt verwiesen wird. Die **nil**-Referenz ist möglich, sowie alle Kombinationen von Objektbezeichnern der Buch-Extension. Da hier eine Menge von Objekten kodiert wird und **nil** ebenso wie alle anderen Objektbezeichner verwendet wird, ist auch eine Regel wie die letzte möglich, bei der explizit **nil** neben einem regulären Objektbezeichnern verwendet wird.

□

6.4.3 Abbildungen als Baumtransformationen

Kodierung der Umkehrabbildung

Ist eine Abbildung f vom relationalen zum objektorientierten Schema gegeben, wird deren Umkehrabbildung f^{-1} auf Basis des kodierten objektorientierten Schemas $\mathcal{A}_S$ beschrieben, indem dieses als Quellschema für einen Automaten mit Ausgabe verwendet wird. Im Gegensatz zum Fall von Abbildungen innerhalb des relationalen Modells, bei dem es mit der relationalen Algebra eine einheitliche Abbildungssprache gibt, existiert im Fall von Abbildungen von einem relationalen zu einem objektorientierten Schema keine einheitliche Abbildungssprache.

Wird eine spezielle Abbildungssprache betrachtet, können deren Operatoren als Baumabbildungen kodiert werden, indem ähnlich wie bei den Operatoren der SPC-Algebra in Abschnitt 5.4.3 vorgegangen wird. Da es ein Baumautomat zulässt, dass eine Annotation sowohl in Abhängigkeit ihres Werts als auch in Abhängigkeit von ihrer Verwendungsposition abgebildet werden kann, sind alle Möglichkeiten, eine Abbildung zu beschreiben, abgedeckt.

Da es zu einem Automaten, der f^{-1} kodiert, immer einen inversen Automaten gibt, kann überprüft werden, ob Abbildung f und Umkehrabbildung f^{-1} zusammenpassen. Für die kodierte Umkehrabbildung muss der inverse Automat bestimmt werden und bei diesem überprüft werden, ob er die Abbildung f beschreibt. Es ist also ein Korrektheitskriterium vorhanden, mit dessen Hilfe bestimmt werden kann, ob Abbildung und Umkehrabbildung zusammenpassen.

Dekodierung des Zielschemas als relationales Schema

Wird die Abbildung erst auf Seiten der Bäume festgelegt, ist jede informationserhaltende Abbildung geeignet, die Objekte zu speichern. Ist eine informationserhaltende Abbildung auf Basis des kodierten objektorientierten Schemas als Definitionsbereich gegeben, besitzt diese ein Zielschema. Das Zielschema beschreibt die Menge an Relationen, die zur Speicherung der Objekte notwendig ist. Kann das Zielschema als relationales Schema dekodiert werden, ergeben sich die Relationstypen, die mindestens zur Speicherung der Objekte benötigt werden. Die Dekodierung muss gewährleisten, dass aus jedem Ziel-

schema die Relationstypen berechnet werden können und dass die Bildinstanzen der Abbildung in die Relationen umgesetzt werden können. Ziel dieses Abschnitts ist es, ein geeignetes Dekodierungsverfahren festzulegen, das aus einem Automaten ohne Ausgabe Relationstypen erzeugt, so dass passend dazu aus Bäumen Relationen werden.

Um die Dekodierung eines Automaten ohne Ausgabe zu vereinfachen, wird davon ausgegangen, dass der zu dekodierende Automat nur Regeln mit flachen Werten enthält, d.h. es wird durch jede Regel nur eine Ebene eines Baums beschrieben. Automaten mit Regeln, die tiefer geschachtelte Eingabewerte besitzen, können durch Umformungen wie das Aufteilen von Regeln nach Abschnitt 4.3 in einen äquivalenten Automaten mit flachen Regeln umgebaut werden.

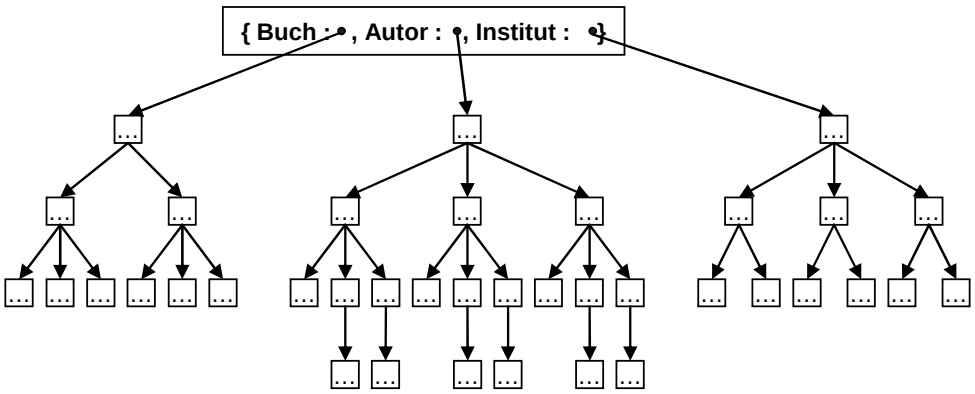


Abbildung 6.10: *Kodierung von Objekten als Baum*

Als ein erster Anhaltspunkt für ein Dekodierungsverfahren wird die Struktur von Bäumen betrachtet, die sich bei einem für ein objektorientiertes Schema kodierten Automaten ergeben können. Um den aus dem Kodierungsverfahren für Objekte generierten Baum in Abb. 6.10 in Relationen zu speichern, kann versucht werden, das Verfahren zur Kodierung von Relationen aus Kapitel 5 umzukehren und die Annotationen auf der obersten Ebene eines Werts als Relationsbezeichner und die jeweils nachfolgenden Teilbäume als zugehörige Relationen zu interpretieren. Diese einfache Umkehrung des Verfahrens funktioniert aber nicht, da solche Teilbäume nicht ausbalanciert sind, d.h. nicht alle Pfade in einem solchen Teilbaum besitzen dieselbe Länge wie dies bei kodierten Relationen der Fall ist. Der Teilbaum unter der Annotation *Autor* ist im Beispiel etwa nicht ausbalanciert. Würde für einen solchen Teilbaum trotzdem eine Relation erzeugt, würde dies bedeuten, dass in der Relation bei manchen Tupeln Attribute nicht mit Werten besetzt sind; die Relation also in dieser Hinsicht unvollständig ist. Um dies zu vermeiden, müssen die Pfade eines Werts passend sortiert werden, so dass nur gleich lange Pfade zu Tupeln einer Relation werden.

Ein Tupel einer Relation stellt sich in einem zugehörigen Baum als eine Folge von

Annotationen dar. Ist ein Automat für einen Relationstyp R [A, B, C] kodiert worden, so enthält dieser unter anderem Regeln der Form:

$$\begin{array}{lcl} \{\ R : q_A, \ldots\} & \longrightarrow & q_f \\ \{\ a : q_B, \ldots\} & \longrightarrow & q_A \\ \{\ b : q_C, \ldots\} & \longrightarrow & q_B \\ \{\ c, \ldots\} & \longrightarrow & q_C \end{array}$$

Ein Tupel ⟨ a, b, c ⟩ einer Relation von R findet sich im Automaten als eine Folge von Annotationssymbolen wieder. Die Folge beginnt beim Zustand q_f, passiert über die Annotation R den Zustand q_A, über a den Zustand q_B, über b den Zustand q_C und endet mit c. Die zum Tupel gehörigen Annotationen passieren somit eine Menge von Zuständen, wobei Tupel, die zur selben Relation gehören, an denselben Zuständen vorbeikommen. Unterschiedliche Annotationsfolgen, welche an denselben Zuständen vorbeilaufen, sind auch immer gleich lang. Eine geeignete Anordnung der Folgen, um Relationstypen zu generieren, kann daher aus dem Aufbau der Übergangsregeln gewonnen werden. Für jede Menge von Annotationsfolgen, welche dieselben Zustände passieren, wird ein Relationstyp erzeugt, der diese Folgen als Tupel aufnimmt.

Auf der Menge der Zustände Q und der Menge der Annotationen Σ eines Automaten $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ wird eine Beziehung $\xrightarrow{a} \subseteq Q \times \Sigma \times Q$ definiert mit

$$q \xrightarrow{a} q' \quad :\Leftrightarrow \quad \{\ \ldots, a : q', \ldots\} \longrightarrow q \in \Delta$$

die beschreibt, von welchem Zustand q welcher Zustand q' über eine Annotation a von oben nach unten erreichbar ist. Eine Verkettung von Beziehungen $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3$ wird als $q_1 \xrightarrow{a_1 a_2} q_3$ abgekürzt, indem die beiden Annotationen a_1 und a_2 als Folge angegeben werden. Sind mehrere Annotationsfolgen p_1 und p_2 zwischen zwei Zuständen vorhanden, wird dies als $q_1 \xrightarrow{p_1, p_2} q_3$ notiert. Erzeugt man für jeden Zustand eines Automaten einen Knoten in einem Graph und verbindet die Zustände über die $\xrightarrow{a}$-Beziehung, entsteht ein *Zustandsgraph* mit gerichteten Kanten.

Innerhalb eines Zustandsgraphen entsprechen die Kantenbezeichner den Annotationen von kodierten Tupeln bzw. Attributwerten von Tupeln. Folgt man einem Pfad in einem Zustandsgraph von der Wurzel zu einem Blatt, so besitzt dieser Pfad eine fixe Länge und somit die Tupel, die durch beliebige Kombinationen von Kantenbezeichnern entlang dieses Pfades gebildet werden, dieselbe Anzahl von Attributwerten. Ist eine Kante nur mit einem einzigen Bezeichner gekennzeichnet, bedeutet dies, dass alle Tupel auf diesem Pfad bei diesem Attribut denselben Wert besitzen. Es handelt sich dann um eine Konstante.

Beispiel 6.5 (*Zustandsgraph*)
Ist ein Automat als $\mathcal{A} = (\{q_f, q_A, q_B, q_C, q_D\}, \{a, b, c, d\}, q_f, \Delta)$ mit den Übergangsregeln

$$
\begin{array}{lcl}
\{\ R : q_A,\ S : q_D\ \} & \longrightarrow & q_f \\
\{\ \} & \longrightarrow & q_A \\
\{\ a : q_B\ \} & \longrightarrow & q_A \\
\{\ b : q_B\ \} & \longrightarrow & q_A \\
\{\ a : q_B,\ b : q_B\ \} & \longrightarrow & q_A \\
\{\ a : q_C\ \} & \longrightarrow & q_B \\
\{\ b : q_C\ \} & \longrightarrow & q_B \\
\{\ a : q_C,\ b : q_C\ \} & \longrightarrow & q_B \\
\{\ a\ \} & \longrightarrow & q_C \\
\{\ b\ \} & \longrightarrow & q_C \\
\{\ a,\ b\ \} & \longrightarrow & q_C \\
\{\ \} & \longrightarrow & q_D \\
\{\ c\ \} & \longrightarrow & q_D \\
\{\ d\ \} & \longrightarrow & q_D \\
\{\ c,\ d\ \} & \longrightarrow & q_D
\end{array}
$$

gegeben, kann für diesen der Zustandsgraph berechnet werden. Jeder Zustand ergibt einen Knoten des Graphen. Die Knoten sind über bezeichnete Kanten verbunden, wobei die Bezeichner aus den entsprechenden Annotationen der Regeln gewonnen werden. Für den Zustand q_A gibt die Regel $\{\ a : q_B,\ b : q_B\ \} \longrightarrow q_A$ vor, dass von q_A aus der Zustand q_B im Graph über eine Kante mit den Kantenbezeichnern a und b erreichbar ist. Für den Automaten entsteht ein Zustandsgraph, wie er in Abb. 6.11 zu sehen ist.

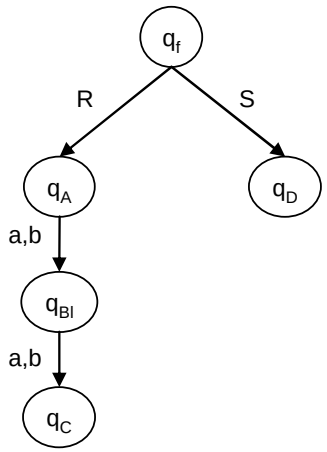


Abbildung 6.11: *Zustandsgraph*

Die Knotenfolge q_f, q_A, q_B, q_C im Zustandsgraph beschreibt einen Pfad von der Wurzel bis zu einem Blatt innerhalb des Graphen. Die Kanten auf diesem Weg sind mit den Bezeichnermengen „R“, „a, b“ und „a, b“ markiert, so dass dieser Pfad die mit den

Attributwerten ⟨ R, a, a ⟩, ⟨ R, a, b ⟩, ⟨ R, b, a ⟩ und ⟨ R, b, b ⟩ beginnenden Tupel beschreibt. Die jeweils letzten Werte für q_C ergeben sich nicht direkt aus dem Graph und müssen aus der Regelmenge des Automaten abgeleitet werden, um komplette Tupel zu erhalten. □

Ist ein Zustandsgraph für einen Automaten vorhanden, können aus dem Graphen die Relationstypen berechnet werden, welche die als Relationen dekodierten Instanzen des Automaten aufnehmen. Es wird für jeden Blattknoten des Zustandsgraphen ein Relationstyp angelegt, welcher die Tupel aufnimmt, die durch Kantenbezeichner auf einem Pfad von der Wurzel bis zu diesem Knoten gebildet werden können. Die notwendigen Attribute eines Relationstyps ergeben sich durch die Knoten, die auf einem Pfad von der Wurzel zum Blattknoten liegen. Gilt für einen Knoten auf einem Pfad, dass von ihm aus ein anderer Knoten über mehrere verschiedene Annotationsfolgen erreichbar ist, wird ein Attribut im Relationstyp angelegt. Ein solches Attribut unterscheidet dann die Tupel, welche die eine Annotationsfolge als Attributwerte enthälen, von den Tupeln, welche die andere Folge enthalten. Für Blattknoten wird ein Attribut benötigt, wenn der Wertebereich des zugehörigen Zustands mehr als einen Wert umfasst.

Die notwendigen Schritte, um aus einem Zustandsgraph Relationstypen zu erzeugen, werden in einem Algorithmus zusammengefasst (Abb. 6.12). Eine Funktion $att : Q \rightarrow$ **attr** ordnet dabei den Knoten des Zustandsgraphs Attributbezeichner zu und eine Funktion $rel : Q \rightarrow$ **rel** den Knoten Relationsbezeichner. Zuerst wird die Menge der Blattknoten des Zustandsgraphs als Q_{leaf} bestimmt. In dieser Menge sind alle Knoten q, die keine Ausgangskante besitzen und somit von ihnen aus kein anderer Knoten q' über eine Annotationsfolge p erreichbar ist. Als nächstes wird die Menge der Knoten Q_{attr} benötigt, für die Attribute in den Relationstypen anzulegen sind. Dies sind diejenigen Knoten q, von denen aus ein anderer Knoten q' über zwei unterschiedliche Annotationsfolgen a_1p_1 und a_2p_2 erreichbar ist, wobei die ersten Annotationen a_1 und a_2 der Folgen verschieden sein müssen. So wird vermieden, dass für Konstellationen wie $q \xrightarrow{a} q' \xrightarrow{p_1} q''$ und $q \xrightarrow{a} q' \xrightarrow{p_2} q''$ mit $p_1 \neq p_2$ sowohl für q als auch für q' ein Attribut angelegt wird. Hier ist das Attribut für q überflüssig und nur für q' wird ein Attribut benötigt. Q_{attr} beschreibt Knoten, für die zwingend ein Attribut angelegt werden muss, und Q_{leaf} die Knoten, für die es sich erst nach Betrachtung der Domäne entscheidet.

Die Domänen der Attribute werden in den folgenden beiden `for`-Schleifen ermittelt. Bei Blattknoten wird nur ein Attribut angelegt, wenn der Wertebereich des Zustands mehr als einen Wert enthält. Dann bilden diese Werte die Domäne des Attributs $attr(q)$, das für den Zustand erzeugt wird. Ansonsten wird für solche Zustände kein Attribut benötigt, da der Wert konstant ist. Für die Attribute, die für Knoten aus Q_{attr} angelegt werden, ergibt sich die Domäne durch die ausgehenden Annotationen a_i, über die ein anderer Knoten q' erreicht werden kann.

Ausgehend von der Menge der Blattknoten Q_{leaf} werden die Relationstypen erzeugt. Hierzu wird für jeden Knoten q aus Q_{leaf} ein Relationstyp $rel(q)$ angelegt und als Attribute, diejenigen ausgewählt, welche den Knoten zugewiesen sind, von denen aus q über

1: $\mathbf{R} = \emptyset$
2:
3: // — Zustände ermitteln, für die Attribut anzulegen sind
4: $Q_{leaf} \leftarrow \{q \in Q \mid \nexists q' \in Q : q \xrightarrow{p} q'\}$
5: $Q_{attr} \leftarrow \{q \in Q \mid \exists q' \in Q, \exists a_1p_1, a_2p_2 \in \Sigma^* : q \overset{a_1p_1}{\rightarrow} q' \wedge q \overset{a_2p_2}{\rightarrow} q' \text{ mit } a_1 \neq a_2\}$
6:
7: // — Domänen der Attribute bestimmen
8: **for all** $q \in Q_{leaf}$ **do**
9: **if** $|dom(q)| > 1$ **then**
10: $dom(attr(q)) \leftarrow dom(q)$
11: **end if**
12: **end for**
13: **for all** $q \in Q_{attr}$ **do**
14: $dom(attr(q)) \leftarrow \{a_1, a_2 \mid \exists q' \in Q, \exists a_1p_1, a_2p_2 \in \Sigma^* : q \overset{a_1p_1}{\rightarrow} q' \wedge q \overset{a_2p_2}{\rightarrow} q' \text{ mit } a_1 \neq a_2\}$
15: **end for**
16:
17: // — Relationstypen generieren
18: **for all** $q \in Q_{leaf}$ **do**
19: $\mathbf{R} \leftarrow \mathbf{R} \cup \{rel(q)\}$
20: $sort(rel(q)) = \{attr(q') \mid \exists p : q' \xrightarrow{p} q, q' \in Q_{attr}\}$
21: **if** $dom(attr(q))$ definiert **then**
22: $sort(rel(q)) = sort(rel(q)) \cup \{attr(q)\}$
23: **end if**
24: **end for**
25:
26: // — $\mathbf{R}$ enthält die notwendigen Relationstypen

Abbildung 6.12: *Algorithmus zur Erzeugung eines relationalen Schemas aus einem Automaten*

einen Pfad erreichbar ist.

Beispiel 6.6 (*Erzeugen von Relationstypen*)
Wendet man den Algorithmus auf den Graphen aus Abb. 6.11 an, gilt $Q_{leaf} = \{q_C, q_D\}$, da dies die einzigen beiden Blattknoten sind. Vom Knoten q_A aus ist der Knoten q_B über zwei unterschiedliche Annotationsfolgen a bzw. b erreichbar. Daher gehört q_A zur Menge Q_{attr}. Ebenso umfasst diese Menge q_B, da von hier aus q_C über zwei Annotationsfolgen erreichbar ist. Insgesamt ergibt sich $Q_{attr} = \{q_A, q_B\}$.

Die Domäne des Attributs für q_C enthält die beiden Werte a und b und die Domäne des Attributs für q_D die beiden Werte c und d. Für q_A wird die Domäne mit den zwei

Werte a und b errechnet und q_B besitzt ebenfalls die Domäne.

Die Menge Q_{leaf} enthält zwei Elemente, so dass zwei Relationstypen R_C und R_D angelegt werden. Die Attribute von R_C ergeben sich als Attribut C, für den Knoten q_C und als Attribute A und B, da von den Knoten q_A und q_B aus jeweils der Konten q_C erreichbar ist. Der zweite Relationstyp ergibt sich als R_D für q_D mit dem einen Attribut D für den Knoten q_D.

Insgesamt wird durch den Algorithmus das relationale Schema

$$R_C\ [\ A,\ B,\ C\ ] \text{ mit } dom(C) = dom(A) = dom(B) = \{a, b\}$$
$$R_D\ [\ D\ ] \text{ mit } dom(D) = \{c, d\}$$

generiert. □

Ein Blattknoten in einem Zustandsgraph kann auch über mehrere unterschiedliche Pfade erreichbar sein, wobei diese Pfade nicht die gleiche Länge besitzen müssen. In diesem Fall werden in einem Relationstyp Tupel unterschiedlicher Länge gespeichert, so dass fehlende Attributwerte etwa durch NULL-Werte aufgefüllt werden müssen.

Liegen rekursive Typen vor, gilt also $q \xrightarrow{p} q$ für einen Knoten q und eine Annotationsfolge p, kann der Algorithmus keine Relationstypen erzeugen, da rekursive Strukturen in einem relationalen Schema nicht modellierbar sind. Um dennoch solche Fälle abdecken zu können, kann bereits bei der Kodierung des objektorientierten Schemas als Baumautomat diese Rekursion aufgelöst werden, indem etwa Verweise zwischen den Elemente der Rekursion eingefügt werden und diese dann wie Verweise auf Objekte behandelt werden.

In einem Baum als Wert ergeben sich die Tupel als Annotationsfolgen entlang eines Pfads von der Wurzel bis zu einem Blatt. Jedem Tupel wird durch den Zustandsgraphen die passende Relation zugewiesen. Im Zustandsgraph entspricht ein Pfad einer Relation, so dass die Tupel der Relation sich als Kombinationen der Kantenbezeichner auf diesem Pfad ergeben. In welchem Attribut welche Annotation des Werts gespeichert wird, ergibt sich aus der Zuordnung der Domänen zu den Knoten. Konstante Werte, d.h. Werte für Kanten mit nur einem Bezeichner, fallen hierbei aus den Tupeln heraus. Ihre Werte ergeben sich bereits implizit durch die Abbildungsbeschreibung und müssen daher nicht gespeichert werden.

Beispiel 6.7 (*Objekte als Relationen*)
Ist der in Abb. 6.13 angegebene Wert als Instanz des Automaten aus Beispiel 6.5 gegeben, können für diesen die Tupel anhand des zugehörigen Zustandsgraphs in Abb. 6.11 berechnet werden.

Der Graph enthält zwei Pfade für zwei Relationstypen. Die Annotationsfolgen *Raaa*, *Raba* und *Rabb* im Baum führen im Zustandsgraph über die Knoten q_f, q_A, q_B und q_C und sind somit Tupel der Relation R_C, da über den Pfad q_C erreicht wird. Beim Wert R handelt es sich um eine Konstante, so dass diese bei der Speicherung wegfällt. Die Zuordnung zwischen Annotation und Wert eines Attributs entspricht der Zuordnung des Zustands im Zustandsgraph zum Attribut. Entsprechend führen die beiden Folgen *Sc*

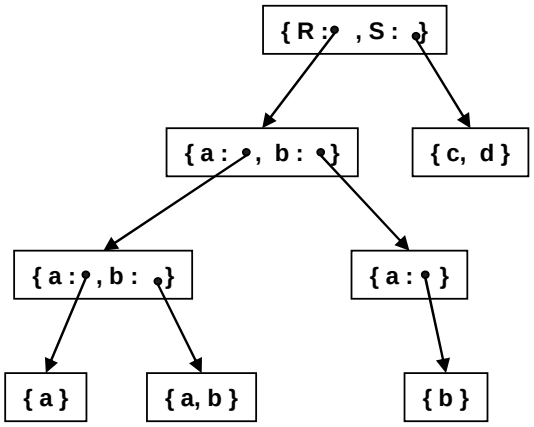


Abbildung 6.13: *Beispielwert*

und Sd zu q_D im Zustandsgraph, so dass diese die Tupel der zweiten Relation bilden. Der Wert wird folglich als die in Abb. 6.14 angegebenen Tupel gespeichert.

R_C

A	B	C
a	a	a
a	b	a
a	b	b
b	a	b

S_C

D
c
d

Abbildung 6.14: *Relationen zum Speichern des Beispielwerts*

□

Das Dekodierungsverfahren erzeugt für einen beliebigen Automaten eine Menge von Relationstypen, welche die als Relationen dekodierten Bäume aufnehmen können.

6.4.4 Änderungsoperationen

Änderungsoperationen auf einem objektorientierten Schema sind als das Ändern von Werten bei Objektattributen und das Anlegen bzw. Löschen von Objekten definiert. Auf Bäume übertragen werden diese Operationen zu einer Folge von Lösch- bzw. Einfügeoperationen von Pfaden auf einem gegebenen Baum. Hierzu können die beiden auf Bäumen definierten Operatoren *union* und $diff$ verwendet werden. Welche Pfade jeweils zu ändern sind, bestimmt das objektorientierte Schema.

Wird ein Objekt gelöscht, bedeutet dies nicht, dass die Verweise auf dieses Objekt ebenfalls gelöscht werden. Solche Verweise zeigen nach einem Löschen ins Leere.

6.5 Bestimmung semantisch korrekter Operationen

Ist eine Abbildung f zwischen einem relationalen und einem objektorientierten Schema gegeben, so wird die Abbildung nicht direkt untersucht, sondern deren Umkehrabbildung, die bei einer zur Speicherung von Objekten geeigneten Abbildung immer vorhanden sein muss. Für das objektorientierte Schema $(C, \sigma, \pi, \prec)$ wird ein Baumautomat ohne Ausgabe $enc((C, \sigma, \pi, \prec)) = \mathcal{A}_S$ erzeugt. Dieser Automat ist kontextbezogen deterministisch, so dass er als Grundlage für die Kodierung der Umkehrabbildung f^{-1} dienen kann, wenn bei seinen Regeln Ausgabewerte ergänzt werden. Optional können auf diesem Automat wieder die Umformungen aus 4.3 durchgeführt werden, bevor die Umkehrabbildung kodiert wird. Gelingt die Kodierung der Umkehrabbildung nicht, ist die Abbildung selbst nicht bijektiv und somit sind nicht alle Änderungsoperationen auf dem Anwendungsschema semantisch korrekt ausführbar. Ist die Kodierung erfolgreich, werden durch den für f^{-1} angelegten Automaten alle Instanzen von $\mathcal{A}_S$ abgebildet, so dass f surjektiv ist. Ergibt sich aus der Kodierung, dass der für die Umkehrabbildung vorhandene inverse Automat ebenfalls kontextbezogen deterministisch ist, liegt eine informationserhaltende Sicht vor (Abb. 6.15) und f ist injektiv. f ist dann eine Bijektion und alle Operationen auf dem objektorientierten Schema sind semantisch korrekt.

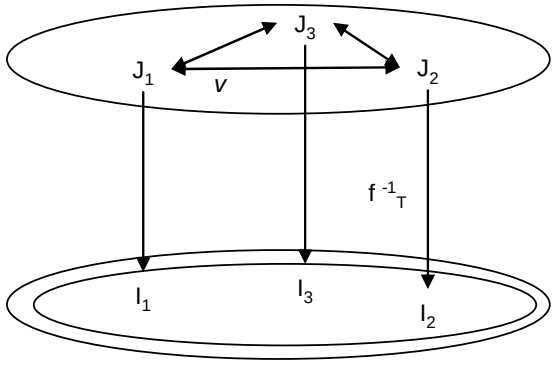


Abbildung 6.15: *Gelingt die Kodierung der Umkehrabbildung f^{-1} als informationserhaltende Abbildung, sind alle Operationen auf dem objektorientierten Schema semantisch korrekt.*

Ist ein objektorientiertes Schema gegeben, eignet sich jede informationserhaltende Abbildung, die auf Basis dieses Schemas definiert wird, zum Speichern der Daten. Jedes Schema kann als Baumautomat dargestellt werden und die Abbildungseigenschaften können anhand des Automaten bestimmt werden. Um das für eine Abbildung passende

relationale Schema zu erhalten, wird für das Zielschema ein Zustandsgraph erzeugt und basierend auf diesem die Relationstypen generiert.

6.6 Beispiel für O/R-Mapping

Diese Kapitel schließt mit einem Beispiel zum O/R-Mapping-Problem. In Beispiel 6.8 wird gezeigt, wie Objekte in Relationen gespeichert werden können.

Beispiel 6.8 (*Identitätsabbildung*)
Ist ein objektorientiertes Schema $(C, \sigma, \pi, \prec)$ mit einer Klasse `Buch` als

```
class Buch {
     attribute ISBN string;
     attribute Titel string;
     attribute Jahr integer;
};
```

gegeben, so sollen die Instanzen dieses Schemas in einer relationalen Datenbank gespeichert werden. Zur vereinfachten Darstellung wird in diesem Beispiel davon ausgegangen, dass $dom(string) = \{0-201-53781-X,\ 1-55867-622-X,\ Database\ theory,\ Data\ and\ the\ Web\}$ und $dom(integer) = \{1997,\ 2001\}$.

Im ersten Schritt wird für dieses Schema durch eine Kodierung $enc((C, \sigma, \pi, \prec))$ der zugehörige Automat $\mathcal{A}_S$ erzeugt. Dieser ergibt sich als $\mathcal{A}_S = (\{q_f, q_{BuchExt}, q_{Buch}, q_{string}, q_{integer}\}, \{Buch, oid1, \ldots, oidn, 0{-}201{-}53781{-}X, 1{-}55867{-}622{-}X, Database\,theory, Data\ and\ the\ Web, 1997, 2001\}, q_f, \Delta)$ mit den Regeln $\Delta = \{$

$$
\begin{array}{lcl}
\{\ Buch : q_{BuchExt}\ \} & \longrightarrow & q_f \\
\{\ oid1 : q_{Buch}, \ldots, oidn : q_{Buch}\ \} & \longrightarrow & q_{BuchExt} \\
 & \ldots & \\
\{\ ISBN : q_{string}, Titel : q_{string}, Jahr : q_{integer}\ \} & \longrightarrow & q_{Buch} \\
\{\ 0-201-53781-X\ \} & \longrightarrow & q_{string} \\
\{\ 1-55867-622-X\ \} & \longrightarrow & q_{string} \\
\{\ Database\ theory\ \} & \longrightarrow & q_{string} \\
\{\ Data\ and\ the\ Web\ \} & \longrightarrow & q_{string} \\
\{\ 1997\ \} & \longrightarrow & q_{integer} \\
\{\ 2001\ \} & \longrightarrow & q_{integer}
\end{array}
$$

$\}$.

Da das Schema nur eine Klasse `Buch` enthält, gibt es eine Schlussregel mit dem Bezeichner der Klasse als Annotation. An diese schließen sich die Regeln für die Extension dieser Klasse an, wobei es für jede Kombination der Objektbezeichner aus $\pi(\texttt{Buch})$ eine entsprechende Regel gibt, von der hier nur eine dargestellt ist. Die Werte, die nach q_{Buch} abgeleitet werden, entsprechen jeweils einem Objekt der Klasse `Buch`. q_{string} und $q_{integer}$

modellieren die atomaren Typen `integer` und `string` mit den jeweiligen Domänen. Der entstandene Automat ist kontextbezogen deterministisch, da das Kodierungsverfahren nur Automaten mit dieser Eigenschaft generiert.

Auf Basis dieses Automaten wird die Abbildung vom objektorientierten in das relationale Schema definiert. Im Beispiel wird die Identitätsabbildung gewählt, so dass sich die Regeln als

$$
\begin{aligned}
&\{\ Buch : q_{BuchExt}(x_1)\ \} \\
&\qquad\qquad \longrightarrow q_f\ (\ \{\ Buch : x_1\ \}\ ) \\
&\{\ oid1 : q_{Buch}(x_1),\ \ldots,\ oidn : q_{Buch}(x_n)\ \} \\
&\qquad\qquad \longrightarrow q_{BuchExt}\ (\ \{\ oid1 : q_{Buch}(x_1),\ \ldots,\ oidn : q_{Buch}(x_n)\ \}\ ) \\
&\ldots \\
&\{\ ISBN : q_{string}(x_1),\ Titel : q_{string}(x_2),\ Jahr : q_{integer}(x_3)\ \} \\
&\qquad\qquad \longrightarrow q_{Buch}\ (\ \{\ ISBN : x_1,\ Titel : x_2,\ Jahr : x_3\ \} \\
&\{\ 0-201-53781-X\ \} \\
&\qquad\qquad \longrightarrow q_{string}\ (\ \{\ 0-201-53781-X\ \}\ ) \\
&\{\ 1-55867-622-X\ \} \\
&\qquad\qquad \longrightarrow q_{string}\ (\ \{\ 1-55867-622-X\ \}\ ) \\
&\{\ Database\ theory\ \} \\
&\qquad\qquad \longrightarrow q_{string}\ (\ \{\ Database\ theory\ \}\ ) \\
&\{\ Data\ and\ the\ Web\ \} \\
&\qquad\qquad \longrightarrow q_{string}\ (\ \{\ Data\ and\ the\ Web\ \}\ ) \\
&\{\ 1997\ \} \\
&\qquad\qquad \longrightarrow q_{integer}\ (\ \{\ 1997\ \}\ ) \\
&\{\ 2001\ \} \\
&\qquad\qquad \longrightarrow q_{integer}\ (\ \{\ 2001\ \}\ )
\end{aligned}
$$

ergeben. Zum Automaten $\mathcal{T}$ entsteht der inverse Automat $\mathcal{T}^{-1}$, indem die Ein- und Ausgabeseiten der Regeln vertauscht werden. Bei der gewählten Identitätsabbildung ist $\mathcal{T}^{-1}$ mit dem Ausgangsautomaten identisch und somit ebenfalls kontextbezogen deterministisch. Die Abbildung beschreibt folglich eine informationserhaltende Sicht und ist somit zum Speichern von Objekten geeignet. Jede Operation auf dem objektorientierten Schema ist semantisch korrekt.

Im letzten Schritt muss ein für die Abbildung passendes relationales Schema erzeugt werden, welches eine Instanz des objektorientierten Schemas als Menge von Relationen aufnimmt. Hierfür wird zuerst für das Zielschema von $\mathcal{T}$ ein Zustandsgraph generiert. Da die Identitätsabbildung verwendet wird, entspricht das Zielschema dem Quellschema der Abbildung und somit $\mathcal{A}_S$. Der Zustandsgraph besitzt für jeden Zustand von $\mathcal{A}_S$ einen Knoten, so dass sich die Knoten q_f, $q_{BuchExt}$, q_{Buch}, q_{string} und $q_{integer}$ und die Kanten wie in Abb. 6.16 angegeben.

Zwischen q_f und $q_{BuchExt}$ gibt es eine Kante mit einem Kantenbezeichner $Buch$, da es eine Regel $\{\ Buch : q_{BuchExt}\ \} \longrightarrow q_f$ gibt. Zwischen $q_{BuchExt}$ und q_{Buch} ist die Kante mit

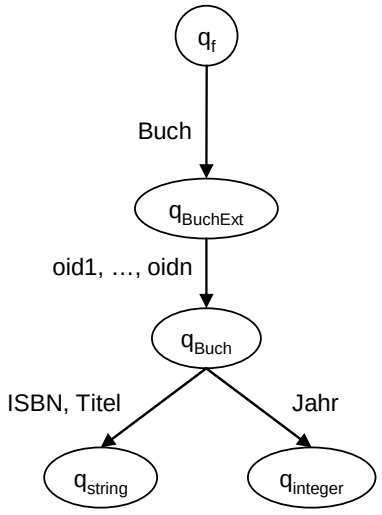


Abbildung 6.16: *Zustandsgraph*

den Objektbezeichnern oid_i markiert. Die Kanten für die anderen Beziehungen ergeben sich durch die entsprechenden anderen Regeln im Automaten.

Aus dem Zustandsgraph wird das relationale Schema mit Hilfe des Algorithmus generiert. Q_{leaf} beinhaltet die Zustände q_{string} und $q_{integer}$ und Q_{attr} die Zustände $q_{BuchExt}$ und q_{Buch}. $q_{BuchExt}$ ist enthalten, weil es über je einen Objektbezeichner mehrere unterschiedliche Wege nach q_{Buch} gibt, und q_{Buch} wegen der zwei Wege $ISBN$ und $Titel$ nach q_{string}.

Die Domäne für das Attribut für $q_{BuchExt}$ besteht aus den Objektbezeichnern und die Domäne für q_{Buch} enthält die zwei Werte $ISBN$ und $Titel$. Für q_{string} muss ein Attribut erzeugt werden, da $|dom(q_{string})| > 1$. Die zugehörige Domäne enthält die Werte $0-201-53781-X$, $1-55867-622-X$, *Database theory* und *Data and the Web*. Für $q_{integer}$ muss ebenfalls ein Attribut mit der Domäne $\{1997, 2001\}$ angelegt werden.

Wird aus dem Zustandsgraph ein relationales Schema abgeleitet, entstehen wegen der zwei Zustände in Q_{leaf} zwei Relationstypen. Diese ergeben sich als

$$Buch_{integer}[BuchExt, integer]$$
$$Buch_{string}[BuchExt, Buch, string]$$

mit $attr(q_{string}) = string$, $attr(q_{integer}) = integer$, $attr(q_{Buch}) = Buch$, $attr(q_{BuchExt}) = BuchExt$ und $rel(q_{string}) = Buch_{string}$, $rel(q_{integer}) = Buch_{integer}$.

Ist ein Beispielobjekt wie in Abb. 6.17 dargestellt zu speichern, ergeben sich die in der Abbildung angegebenen Tupel in den beiden Relationen.

□

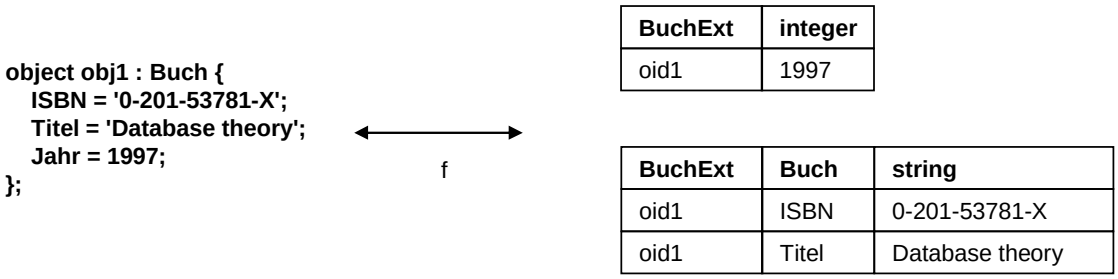


Abbildung 6.17: *Speicherung eines Objekts*

6.7 Fazit

Beim O/R-Mapping-Problem werden aus den in einer relationalen Datenbank hinterlegten Daten die Daten eines objektorientierten Anwendungsprogramms erzeugt. Auf der Anwendungsseite müssen alle dort definierten Operationen auch tatsächlich ausführbar sein, da ein Programm in jeden erlaubten Zustand wechseln kann. Somit ist eine bijektive Abbildung zwischen Datenbankschema und Anwendungsschema notwendig, so dass jeder Programmzustand gespeichert und wieder ausgelesen werden kann.

Um zu überprüfen, ob alle Anwendungsoperationen durchführbar sind, wird anstelle der Abbildung deren Umkehrabbildung kodiert und überprüft, ob sich aus der Kodierung einen informationserhaltende Abbildung ergibt. Ist dies der Fall, ist die Abbildung selbst eine Bijektion und alle Operationen auf dem Anwendungsschema sind semantisch korrekt.

Bei der Betrachtung von O/R-Abbildungen mittels annotierter Bäume ergeben sich im Vergleich zu bestehenden Ansätzen eine Reihe von Vorteilen. Objekte können beliebig geschachtelte Attributwerte besitzen, die in die Abbildungsberechnung mit einbezogen werden und nicht als eigenständige Objekte betrachtet werden. Es ist möglich, Daten auf Metadaten abzubilden und umgekehrt. Dadurch, dass die Abbildung mit Hilfe eines Baumautomaten beschrieben wird, existiert ein Korrektheitsbeweis, dass die Abbildung tatsächlich die gewünschten Eigenschaften besitzt; dass sie zum einen bijektiv ist und das zum anderen jede Anwendungsoperation semantisch korrekt ist.

7 Zusammenfassen von Regelmengen

Wird für ein relationales oder ein objektorientiertes Schema ein Baumautomat erzeugt, generiert die Kodierungsfunktion eine große Anzahl an Regeln, was zum einen den Vorteil bietet, sehr flexibel Abbildungen beschreiben zu können, zum anderen aber einen solchen Automaten unhandlich macht. Da die Regelmenge in diesen Fällen einen strukturierten Aufbau besitzt, kann die Anzahl der Regeln reduziert werden, wenn eine Regel nicht nur einen, sondern eine Reihe möglicher Übergänge zusammenfasst.

Abschnitt 7.1 gibt die Größenordnung für die Anzahl von Regeln in einem kodierten Schema an. Abschnitt 7.2 betrachtet eine Möglichkeit der Verkleinerung der Regelmenge, bei der nur die zur Beschreibung der konkreten Abbildung benötigten Instanzen kodiert werden. Da hier dann nicht komplette Domänen, sondern nur Teile davon kodiert werden, sinkt die Regelanzahl. In Abschnitt 7.3 werden mehrere Übergangsregeln durch eine einzelne ersetzt, wobei Mengen von Annotationen durch reguläre Ausdrücke beschrieben werden.

7.1 Anzahl von Regeln in einem kodierten Schema

Die Kodierungsverfahren aus Kapitel 5 und 6 erzeugen für ein gegebenes relationales oder objektorientiertes Schema einen Baumautomaten. Da die Kodierung auf Einzelwertebene stattfindet, d.h. jeder erlaubte Wert wird einzeln durch ein oder mehrere Regeln dargestellt, ist die Anzahl der Regeln innerhalb eines generierten Automaten groß.

Ist eine Domäne $dom(A)$ eines relationalen Attributs gegeben, so wird diese durch $2^{|dom(A)|} - 1$ Regeln kodiert. Dies entspricht der Anzahl an möglichen Kombinationen von Werten aus dieser Domäne, wobei für die leere Menge keine Regel erzeugt wird. Für einen Relationstyp mit den Attributen $A_1, \ldots, A_n$ bedeutet dies, dass der resultierende Automat dann $2 + \sum_{i=1}^{n}(2^{|dom(A_i)|} - 1)$ Regeln besitzt. Dieser besteht aus der Regel für den Relationsbezeichner, der Regel für die leere Relation und den Regeln für die Attributwerte.

Analog kann eine Betrachtung für einen aus einem objektorientierten Schema generierten Automaten erfolgen. Für die atomaren Domänen gelten hier dieselben Überlegungen wie im relationalen Fall. Zur Beschreibung der erlaubten Objektbezeichner pro Klasse wird für jede mögliche Kombination von Bezeichnern eine Regel angelegt, so dass hier die Regelanzahl ebenfalls der Elementanzahl der Potenzmenge der Bezeichner entspricht.

7.2 Teilmengen von Domänen

Die Menge an Regeln eines Automaten kann reduziert werden, indem nicht alle möglichen Werte aus Domänen betrachtet werden, sondern nur diejenigen, die für eine konkrete Problemstellung notwendig sind. Wird eine Sicht beim View-Update-Problem untersucht, ist bei dieser Fragestellung eine Instanz des Datenbankschemas, eine Instanz des Anwendungsschemas und eine Operation auf der letzteren Instanz gegeben. Die Instanzen umfassen nicht alle durch die Domänen erlaubten Attributwerte, sondern nur Teilmengen davon. Erfolgt die Kodierung dieser Teilmengen, werden weniger Regeln bei einem Automaten erzeugt. Wird nur eine Teilmenge $B \subset dom(A)$ der Domäne eines Attributs A betrachtet anstatt der kompletten Domäne, reduziert sich die generierte Regelmenge um $2^{|dom(A)|} - 2^{|B|}$ Regeln.

Beispiel 7.1 (*Verkleinerung von Domänen*)
Sei das Schema **R** mit den Relationstypen

R [A, B, C] mit $dom(A) = dom(B) = dom(C) = \{a, b\}$
S [D] mit $dom(D) = \{c, d\}$

und die Projektion $R_V := \pi_B(R)$ aus Beispiel 5.6 gegeben. Die Datenbankinstanz sei kodiert als $\{ R : \{ a : \{ a : \{ a \} \} \}$, $S : \{ c \} \}$ und auf der Sicht soll ein Löschen des dort vorhandenen Tupels $\langle$ a $\rangle$ aus der Relation R_V durchgeführt werden.

Da die Datenbankinstanz bei den Attributen von R nur den Wert a und bei S nur den Wert c enthält, die durchzuführende Änderungsoperation ebenfalls keine weiteren davon verschiedenen Attributwerte besitzt, müssen auch nur diese Werte im Automaten kodiert werden. Es ergibt sich ein Automat mit den Regeln

$$
\begin{array}{lcl}
\{ R : q_A(x_1),\, S : q_D(x_2) \} & \longrightarrow & q_f \; (\{ R : x_1,\, S^C : x_2 \}) \\
\{ \} & \longrightarrow & q_A \; (\{ \}) \\
\{ a : q_B(x_1) \} & \longrightarrow & q_A \; (\{ a^C : x_1 \}) \\
\{ a : q_C(x_1) \} & \longrightarrow & q_B \; (\{ a : x_1 \}) \\
\{ a \} & \longrightarrow & q_C \; (\{ a^C \}) \\
\{ c \} & \longrightarrow & q_D \; (\{ c^C \})
\end{array}
$$

welcher die Projektion auf der gegebenen Datenbankinstanz beschreibt. Wird die Datenbankinstanz abgebildet, entsteht der Wert $\{ R : \{ a^C : \{ a : \{ a^C \} \} \}$, $S^C : \{ c^C \} \}$. Wird hier das einzige Tupel $\langle$ a $\rangle$ gelöscht, also ein Wechsel nach $\{ R : \{ \}$, $S^C : \{ c^C \} \}$ vollzogen, kann dieser Wert durch den inversen Automaten auf die passende Datenbankinstanz abgebildet werden.

Im Vergleich zum Original-Baumautomaten aus Beispiel 5.6 hat sich die Regelmenge um neun Regeln verkleinert. Bei Domänen mit mehr Elementen fällt die Reduktion entsprechend größer aus. □

Abbildungseigenschaften gehen bei dieser Vorgehensweise nicht verloren, wenn die kodierte Abbildung auf den vollständigen Domänen informationserhaltend ist, da dann

zwischen Instanzen des Definitions- und Bildbereichs eine Eins-zu-eins-Zuordnung besteht. In diesem Fall spielt es keine Rolle, wenn Instanzen auf einer der Seiten oder auf beiden Seiten der Abbildung bei der Kodierung nicht berücksichtigt werden. Besitzt eine Abbildung diese Eigenschaft nicht, können keine verkleinerten Domänen betrachtet werden, da sich dann die Abbildungseigenschaften ändern können. Bildet eine Abbildung etwa die einzigen zwei Datenbankinstanzen auf eine einzelne Anwendungsinstanz ab und wird die eine Datenbankinstanz nicht kodiert, wird aus einer vormals nicht bijektiven Abbildung eine Bijektion.

Eine Einschränkung der Domänen bei einer Kodierung hat auch zur Folge, dass nicht mehr alle Möglichkeiten, eine Operation auszuführen, zur Verfügung stehen. Beim Automaten in Beispiel 7.1 ist der gewählte Wechsel zwischen den Bildinstanzen der einzig überhaupt mögliche – es gibt nur zwei Bildinstanzen –, um das Tupel zu löschen. Bei einem Automaten, der aus einer Kodierung der vollständigen Domänen gewonnen wird, stehen hier mehr Auswahlmöglichkeiten zur Verfügung, die Operatoren zu kodieren.

7.3 Annotationen als reguläre Ausdrücke

Annotationen können wie bisher angenommen einzelne Konstanten aus Σ eines Automaten $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ darstellen. Ändert man die Betrachtungsweise, können sie aber auch als reguläre Ausdrücke [Yu97] über der Menge der Annotationssymbole Σ aufgefasst werden, die jeweils eine Sprache beschreiben, welche genau die angegebene Konstante umfasst. Ist ein Alphabet Σ gegeben, so ist

- $e = a$ mit $a \in \Sigma$ ein regulärer Ausdruck, der die Sprache $L(e) = \{a\}$ beschreibt,
- $e = (e_1)(e_2)$ ein regulärer Ausdruck, der die Sprache $L(e) = L(e_1)L(e_2)$ beschreibt, die durch das Verketten der Wörter aus $L(e_1)$ und $L(e_2)$ entsteht,
- $e = (e_1) \mid (e_2)$ ein regulärer Ausdruck, der die Sprache $L(e) = L(e_1) \cup L(e_2)$ beschreibt, welche durch die Vereinigung der beiden Sprachen entsteht,
- $e = (e_1)^*$ ein regulärer Ausdruck, der die Sprache $L(e) = L(e_1)^*$ beschreibt, welche durch das Verketten von Wörtern aus $L(e_1)$ entsteht.

Eine einzelne Annotation a ist somit ein regulärer Ausdruck a, welcher die Sprache $L(a) = \{a\}$ beschreibt. Werden bei der Formulierung von Übergangsregeln anstatt einfacher Annotationen komplexere reguläre Ausdrücke erlaubt, wird es möglich, mehrere Regeln durch Regeln mit regulären Ausdrücken zusammenzufassen. Ein Ausdruck beschreibt dann eine Menge von an einer bestimmten Stelle erlaubten Annotationen.

Für die Übergangsregeln eines Automaten ohne Ausgabe $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ wird die folgende Schreibweise

$$\{\ e_1 : q_1, \ldots, e_n : q_n\ \} \quad \longrightarrow \quad q$$

eingeführt, wobei e_i für $1 \leq i \leq n$ jeweils einen regulären Ausdruck über den Annotationssymbolen Σ darstellt mit $L(e_i) \neq \emptyset$ und $L(e_i) \cap L(e_j) = \emptyset$ für alle $i \neq j$. Dadurch, dass die Schnittmenge von zwei Sprache leer sein muss, wird erreicht, dass innerhalb eines Werts die Annotationen einer Ebene wieder eindeutig sind. Die regulären Ausdrücke werden hierbei als Platzhalter für eine individuelle Annotation verstanden, d.h. in einem Baum kann der Platz des Ausdrucks von einem beliebigen Wort aus der durch den Ausdruck beschriebenen Sprache eingenommen werden.

Verwendet man Übergangsregeln mit regulären Ausdrücken, ergibt sich die Übergangsrelation zwischen zwei Werten als

$$v \longrightarrow_{\mathcal{A}} v' :\Leftrightarrow \begin{cases} \exists w \in \mathcal{V}(\Sigma, \{x\} \cup Q) \\ \exists \{\ e_1 : q_1, \ldots e_n : q_n\} \ \longrightarrow q \in \Delta, \\ \forall i \in \{1, \ldots, n\} : a_i \in L(e_i), \\ v = w[x \ / \ \{\ a_1 : q_1, \ldots, a_n : q_n\}\ ], \\ v' = w[x \ / \ q] \end{cases}$$

Ein regulärer Ausdruck fungiert als Platzhalter für eine Menge von vor einem bestimmten Zustand erlaubten Annotationen. Regeln mit tiefer geschachtelten Werten auf Eingabeseite sind dann entsprechend zu interpretieren.

Die Eigenschaften von Automaten mit festen Annotationen können auf Automaten mit regulären Ausdrücken übertragen werden. Beschreiben die Regeln aus Δ_q für alle Zustände q eines Automaten eine Partitionierung, dann ist der Automat kontextbezogen deterministisch. Für eine Überprüfung dieser Eigenschaft müssen die Instanzenmengen der entsprechend konstruierten Automaten wieder disjunkt sein.

Beispiel 7.2 (*Reguläre Ausdrücke*)
Betrachtet man einen Automaten $\mathcal{A} = (Q, \Sigma, q_f, \Delta)$ mit den Übergangsregeln

$$\begin{array}{lcl} \{\ R : q_A\ \} & \longrightarrow & q_f \\ \{\ a : q_B\ \} & \longrightarrow & q_A \\ \{\ b : q_B\ \} & \longrightarrow & q_A \\ \{\ a : q_B, b : q_B\ \} & \longrightarrow & q_A \\ \{\ a\ \} & \longrightarrow & q_B \\ \{\ b\ \} & \longrightarrow & q_B \\ \{\ a, b\ \} & \longrightarrow & q_B \end{array}$$

so sind dort die beiden Regeln $\{\ a : q_B\ \} \longrightarrow q_A$ und $\{\ b : q_B\ \} \longrightarrow q_A$ vorhanden. Diese stimmen bis auf die Annotationen überein, so dass sie zusammengefasst werden können zu einer Regel $\{\ (a \mid b) : q_B\ \} \longrightarrow q_A$ mit dem regulären Ausdruck $(a \mid b)$ anstatt den einfachen Annotationen. Verfährt man ebenso mit dem anderen Regelpaar, das zu q_B führt, ergibt sich ein zum ersten Automat äquivalenter Automat mit der Regelmenge

$$
\begin{array}{lcl}
\{\ R : q_A\ \} & \longrightarrow & q_f \\
\{\ (a \mid b) : q_B\ \} & \longrightarrow & q_A \\
\{\ a : q_B, b : q_B\ \} & \longrightarrow & q_A \\
\{\ (a \mid b)\ \} & \longrightarrow & q_B \\
\{\ a, b\ \} & \longrightarrow & q_B
\end{array}
$$

Durch die Schreibweise mit regulären Ausdrücken wird aus einem Automaten mit sieben Regeln ein Automat mit nur noch fünf Regeln. Bei entsprechend größeren Domänen reduziert sich die Zahl der Regeln stärker. □

Reguläre Ausdrücke erlauben es, Regeln zusammenzufassen, die an einer bestimmten Stelle unterschiedliche Annotationen erlauben und ansonsten übereinstimmen.

Beim Ergebnis einer Kodierung eines Schemas treten häufig auch Regeln auf, die sich darin unterscheiden, dass auf den linken Seiten eine unterschiedliche Anzahl von selben Zuständen vorkommt, denen jeweils ein anderes Annotationssymbol vorangestellt ist. Die Kodierung eines relationalen Attributs erzeugt Regeln der Form

$$
\begin{array}{lcl}
\{\ a_1 : q_{A_{i+1}}\ \} & \longrightarrow & q_{A_i} \\
\{\ a_1 : q_{A_{i+1}}, a_2 : q_{A_{i+1}}\ \} & \longrightarrow & q_{A_i} \\
\{\ a_1 : q_{A_{i+1}}, \ldots, a_n : q_{A_{i+1}}\ \} & \longrightarrow & q_{A_i}
\end{array}
$$

wobei alle a_i aus derselben Menge – der Domäne von A_i – stammen und nur der Zustand $q_{A_{i+1}}$ auf den linken Seiten der Regeln vorkommt. Durch den regelmäßigen Aufbau einer solchen Regelmenge kann diese ebenfalls zusammengefasst beschrieben werden.

Die Modellierungsmöglichkeiten von Regeln werden erweitert, um derartige Wiederholungen zu beschreiben. Eine hochgestellte Zahl m bei einem regulären Ausdruck gibt an, wie oft Annotationen aus der durch den Ausdruck beschriebenen Sprache gefolgt von dem angegebenen Zustand innerhalb einer Ebene eines Werts auftreten dürfen:

$$
\{\ e_1^{m_1} : q_1, \ldots, e_n^{m_n} : q_n\ \} \quad \longrightarrow \quad q
$$

mit $m_i \geq 1$ und $L(e_i) \neq \emptyset$ für alle $1 \leq i \leq n$. Es gilt wieder $L(e_i) \cap L(e_j) = \emptyset$ für alle $i \neq j$.

Entsprechend ergibt sich die Übergangsrelation als

$$
v \longrightarrow_{\mathcal{A}} v' :\Leftrightarrow \left\{
\begin{array}{l}
\exists w \in \mathcal{V}(\Sigma, \{x\}) \cup Q \\
\exists \{\ e_1^{m_1} : q_1, \ldots e_n^{m_n} : q_n\} \longrightarrow q \in \Delta, \\
\forall i \in \{1, \ldots, n\} : a_i^j \in L(expr_i), \\
a_i^j \text{ paarweise verschieden für unterschiedliche } j, \\
v = w[x \ / \ \{\ a_1^1 : q_1, \ldots, a_1^{m_1} : q_1, \ldots, a_n^1 : q_n, \ldots, a_n^{m_n} : q_n\}\ ], \\
v' = w[x \ / \ q]
\end{array}
\right.
$$

Eine hochgestellte Zahl m_i beschreibt eine fixe Anzahl von Wiederholungen eines Zustands. Wird anstelle einer festen Zahl ein Wertebereich angeben, variiert die Anzahl der Wiederholungen innerhalb dieses Wertebereichs.

Beispiel 7.3 (*Reguläre Ausdrücke*)
Im Automaten aus Beispiel 7.2 gibt es die Regeln

$$\begin{array}{lcl} \{\ a : q_B\ \} & \longrightarrow & q_A \\ \{\ b : q_B\ \} & \longrightarrow & q_A \\ \{\ a : q_B, b : q_B\ \} & \longrightarrow & q_A \end{array}$$

bei denen jeweils Annotationssymbole aus der Menge $\{a, b\}$ vorkommen und vom Zustand q_B ein bis zwei Wiederholungen auf den linken Seiten erlaubt sind.

Verwendet man reguläre Ausdrücke und hochgestellte Wertebereiche, können die drei Regeln zu einer Regel

$$\{\ (a \mid b)^{1 \leq n \leq 2} : q_B\ \} \quad \longrightarrow \quad q_A$$

zusammengefasst werden, wobei diese Regel ausdrückt, dass bei einem Wert eine Ebene mit einem oder zwei Annotationen aus der Sprache $L(a \mid b)$ vorkommen muss und der Annotation jeweils ein q_B folgen muss.

Entsprechend ergibt sich ein zum für das relationale Schema kodierten Automaten aus Beispiel 7.2 äquivalenter Automat mit den Regeln

$$\begin{array}{lcl} \{\ R : q_A\ \} & \longrightarrow & q_f \\ \{\ (a \mid b)^{1 \leq n \leq 2} : q_B\ \} & \longrightarrow & q_A \\ \{\ (a \mid b)^{1 \leq n \leq 2}\ \} & \longrightarrow & q_B \end{array}$$

Die Zahl der Regeln, um das relationale Schema zu beschreiben, ist auf drei gesunken. □

Das Zusammenfassen von Regeln zu Regeln mit regulären Ausdrücken kann anhand der Regelmenge eines gegebenen Automaten erfolgen, indem die Regeln mit einander verglichen werden und bei passenden Übereinstimungen die Regeln zusammengefasst werden. Eine weitere Alternative ist, dass das Kodierungsverfahren für ein Schema bereits Regeln mit regulären Ausdrücken erzeugt. Ist ein Relationstyp $R\ [\ A_1, \ldots, A_i, \ldots, A_n\ ]$ gegeben, wird eine Domäne eines Attributs A_i durch eine einzelne Regel

$$\{\ dom(A_i)^{1 \leq n \leq |dom(A_i)|} : q_{A_{i+1}}\ \} \quad \longrightarrow \quad q_{A_i}$$

beschrieben, die aussagt, dass eine beliebige Teilmenge von Werten aus der Domäne von A_i in einer Ebene vorkommen muss, wobei jede Annotation von $q_{A_{i+1}}$ gefolgt wird.

Werden reguläre Ausdrücke statt einzelner Annotationen innerhalb von Regeln verwendet, wandert die Komplexität von einer Vielzahl von Regeln und einer einfachen Übergangsrelation hin zu wenigen Regeln und komplexeren Übergangsrelationen. Da die eine Variante Regel darzustellen nur eine abkürzende Schreibweise der anderen darstellt, können mit beiden Varianten dieselben Wertemengen beschrieben werden.

Die Definition von Abbildungen kann bei der Verwendung von regulären Ausdrücken allerdings nicht mehr auf Einzelwertebene stattfinden, da über eine einzelne Regel nicht mehr nur ein einzelner Teilwert der Eingabe erfasst wird, sondern eine Eingabeseite einer Regel immer eine Menge von Teilwerten beschreibt. Wird anhand von Regeln mit regulären Ausdrücken eine Abbildung definiert, kann hier immer nur davon ausgegangen

werden, dass Teile der Eingabe in die Ausgabe kopiert werden und eine Abbildung nicht mehr individuell anhand einzelner Werte erfolgen kann. Ist eine Abbildung mit einfachen Regeln als

$$\begin{array}{lcl} \{\ a\ \} & \longrightarrow & q_f\ (\ \{\ 0\ \}\ ) \\ \{\ b\ \} & \longrightarrow & q_f\ (\ \{\ 1\ \}\ ) \end{array}$$

möglich, so kann mit nur noch einer Regel mit einem regulären Ausdruck diese Abbildung nicht mehr beschrieben werden

$$\{\ (a \mid b)\ \} \quad \longrightarrow \quad q_f\ (\ ?\ )$$

da der Ausgabewert nicht mehr anhand der linken Seite der Regel festgemacht werden kann. Eine mögliche Annahme wäre, dass die Annotationen der Eingabe in die Ausgabe kopiert werden und somit durch die angegebene Regel die Identitätsabbildung beschrieben wird.

7.4 Fazit

Die Kodierungsverfahren für relationale bzw. objektorientierte Schemata erzeugen Automaten, die zumeist eine große Anzahl an Regeln besitzen. Um diese Anzahl zu reduzieren, kann die Menge der bei einer konkreten Problemstellung betrachteten Schemainstanzen reduziert werden, indem nicht die vollständigen Domänen von Attributen kodiert werden, sondern nur eine Kodierung von Teilmengen erfolgt. Allerdings ist hierbei darauf zu achten, dass sich die Abbildungseigenschaften nicht ändern.

Eine Menge von Regeln wird zusammengefasst, indem anstatt einzelner Annotationen reguläre Ausdrücke innerhalb von Regeln verwendet werden. Somit wird durch eine Regel eine Menge von Regeln ersetzt, die einer gewissen Regelmäßigkeit folgen. Unterscheiden sich Regeln nur durch verschiedene Annotationen oder treten Wiederholungen von Zuständen auf den linken Seiten auf, kann eine abkürzende Schreibweise durch Regeln mit regulären Ausdrücken verwendet werden. Die Komplexität verschiebt sich hierbei von einer großen Anzahl von Regeln und einer einfachen Übergangsrelation zu einer reduzierten Regelmenge und komplexeren Übergangsrelationen.

8 Zusammenfassung und Ausblick

Komplexe Anwendungsprogramme wie Applikationsserver folgen in ihrem Aufbau häufig einer mehrschichtigen Architektur, wobei jede Schicht eine ihr zugewiesene Funktionalität erbringt. So sind in einer Datenbankschicht die Anwendungsdaten persistent hinterlegt, wohingegen die Verarbeitung der Daten in einer Anwendungsschicht stattfindet. Zwischen den Schichten erfolgt ein Datenaustausch, wobei sich die je Schicht verwendeten Datenmodelle unterscheiden können, wenn etwa ein objektorientiertes Anwendungsprogramm lesend und schreibend auf die in einer relationalen Datenbank gespeicherten Daten zugreift, oder übereinstimmen können, wenn eine relationale Datenbank über eine relationale Sicht der Anwendung einen Ausschnitt der hinterlegten Daten zur Verfügung stellt. In beiden Fällen beschreibt eine Abbildung von den Daten der Datenbank auf die Daten der Anwendungsschicht, wie die Daten zu transformieren sind.

Werden auf Seiten der Anwendungsschicht Änderungsoperationen auf den Daten durchgeführt, muss der Zustand der Datenbank passend nachgeführt werden, um wieder einen konsistenten Systemzustand zu erhalten. Die Ausführung einer Operation führt aber nicht immer zu der in ihrer Definition angegebenen Ergebnisinstanz, wenn die Ausführung etwa Seiteneffekte bewirkt oder aus einer Menge von möglichen Umsetzungen keine ausgewählt werden kann. Um Umsetzungen angeben zu können, schränken existierende Ansätze und Untersuchungen die Menge der betrachteten Abbildungen künstlich ein und bieten folglich nur für einen Ausschnitt der erlaubten Abbildungen und Änderungsoperationen eine Lösung an.

Diese Arbeit zeigt, dass für jede Abbildung bei jeder Operation bestimmt werden kann, ob diese durchführbar ist oder nicht und, wenn sie durchführbar ist, wie die Umsetzung aussieht. Es kann anhand eines formalen Berechnungsverfahrens auf Basis der auf dem Anwendungsschema durchzuführenden Änderungsoperation, der aktuellen Anwendungsinstanz und letztendlich der Abbildung selbst entschieden werden, ob eine Änderungsoperation gemäß ihrer Definition durchführbar ist und wie eine Umsetzung aussieht.

Existierende Darstellungsformen von Abbildungen sind unhandlich, um die korrekt durchführbaren Operationen zu bestimmen, da bei ihnen kein Zielschema zur Beschreibung des Bildbereichs vorhanden ist und da die Abbildung zumeist nicht anhand der Abbildungsdefinition auf Injektivität überprüfbar ist. Ein Zielschema ist notwendig, um die Seiteneffektfreiheit einer Operation nachzuweisen, d.h. zu zeigen, dass eine mögliche Umsetzung vorhanden ist. Die Injektivität der Abbildung garantiert die Eindeutigkeit von Operationen, da jeder Anwendungsinstanz höchstens eine Datenbankinstanz zugewiesen ist, d.h. es steht eindeutig fest, welches Ergebnis eine Umsetzung erreichen muss. Die Arbeit setzt eine andere Repräsentation einer Abbildungsbeschreibung ein, bei der

bei jeder formulierbaren Abbildung ein Zielschema vorhanden ist und die Injektivität bestimmt werden kann, und reduziert hierauf zu untersuchende Abbildungen.

Die Arbeit zeigt, dass ein Abbildungsmechanismus existiert, bei dem bei jeder mit diesem Mechanismus formulierten Abbildung die durchführbaren Änderungsoperationen bestimmt werden können. Werte werden in Form von Bäumen repräsentiert, wobei die Kindknoten über Annotationen eindeutig ansprechbar sind. Eine Menge von Bäumen wird durch einen Baumautomaten beschrieben, der Werte akzeptieren oder ablehnen kann. Die Menge der akzeptierten Bäume bildet seine Instanzen. Wird bei der Akzeptanz eines Werts durch den Automaten eine Ausgabe generiert, definiert der Automat eine Relation zwischen zwei Wertemengen. Gibt es für jeden akzeptierten Wert nur gleiche Ableitungswege, dann definiert der Automat eine Abbildung. Für jede durch einen Automaten definierte Abbildung ist ein Zielschema vorhanden und für jeden Automat ist ein inverser Automat definiert. Beschreibt der inverse Automat ebenfalls eine Abbildung, ist eine informationserhaltende Abbildung zwischen zwei Wertemengen gegeben.

Für eine Abbildungsbeschreibung innerhalb des View-Update-Problems und des O/R-Mapping-Problems wird eine Kodierungsfunktion festgelegt, mit deren Hilfe diese Abbildungen als Baumabbildungen dargestellt werden und so die Abbildungseigenschaften bestimmt werden können:

- Im Fall von relationalen Sichten zeigt diese Arbeit, dass eine Kodierung existiert, bei der bei jeder Sicht, die mit Hilfe von Selektionen, Projektionen oder kartesischem Produkt definiert wird, entschieden werden kann, welche Änderungsoperationen auf der Sicht wie auf der Datenbank durchgeführt werden können. Hierbei wird die Menge der überhaupt betrachteten Sichten nicht eingeschränkt, da die untersuchten Sichten keine Voraussetzungen erfüllen müssen.

- Im Fall des Speicherns von objektorientierten Anwendungsdaten in einer relationalen Datenbank wird gezeigt, dass eine gegebene Abbildung auf Bijektivität überprüft werden kann, da eine formale Definition für eine Abbildung angegeben wird. Jede bijektive Abbildung zwischen den Schemainstanzen kann verwendet werden, um die Daten zu speichern.

Die Arbeit stellt nur zwei Anwendungsfälle vor, doch ist der Ansatz ebenfalls zur Untersuchungen von Abbildungen zwischen Schemata anderer Datenmodelle wie XML einsetzbar. Hierfür wird eine Kodierungsfunktion benötigt, welche die entsprechenden Werte und Schemata in eine Baumdarstellung überführt. Da die Werte von Datenmodellen häufig einer geschachtelten Struktur folgen, ist eine Übertragung der Werte und Schemata durch eine Kodierungsfunktion in eine Baumdarstellung möglich.

Wird für ein Schema ein Automat generiert, ist die Zahl der Übergangsregeln des Automaten hoch. Da die Regeln bei Automaten, die kodierte Schemata darstellen, aber gewisse Strukturen aufweisen, kann die Regelmenge durch Verlagerung von Semantik in die Übergangsrelation verkleinert werden, indem Übergangsregeln mit Hilfe von regulären Ausdrücken nicht nur einen Übergang, sondern eine Menge von Übergängen beschreiben.

Um die Tragfähigkeit des Ansatzes zu zeigen, wurde eine prototypische Implementierung einer Baumabbildung im Fall des O/R-Mapping-Problems realisiert, bei der Objekte durch eine Identitätsabbildung auf Bäumen in einem relationalen Schema gespeichert werden. Beim praktischen Einsatz des Ansatzes zeigt sich allerdings, dass bei benutzerdefinierten Abbildungen die Möglichkeiten, eine Abbildung zu formulieren, nicht sehr anwenderfreundlich sind, da bei den Regeln einzeln die Ausgabewerte festgelegt werden müssen. Hier kann über eine Abbildungssprache nachgedacht werden, durch welche für die Regeln nicht einzeln die Ausgabewerte festgelegt werden, sondern eine Definition einer Abbildung auf höherer Ebene stattfinden kann, die dann auf Regelebene herunter gebrochen wird.

Insgesamt betrachtet, liegt der Fokus der Arbeit auf dem Nachweis, dass Operationen auf ihre Durchführbarkeit untersucht werden können. Es wird also ein Rahmen abgesteckt und aufgezeigt, welche Aspekte bei einer Untersuchung zu beachten sind. Existierende Ansätze bieten zwar teilweise eine einfachere Handhabung der Problemstellung an, können hierfür aber nicht über alle erlaubten Abbildungen Aussagen machen.

Literaturverzeichnis

[A+95] Abiteboul, Serge et al.: *The GOODSTEP Project — Final Report.* Technical Report, Insitut Natinal de Recherche en Informatique et en Automatique, 1995.

[ABD+89] Atkinson, Malcolm P., François Bancilhon, David J. DeWitt, Klaus R. Dittrich, David Maier und Stanley B. Zdonik: *The Object-Oriented Database System Manifesto.* In: Kim, Won, Jean-Marie Nicolas und Shojiro Nishio (Herausgeber): *Deductive and Object-Oriented Databases, Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), Kyoto research Park, Kyoto, Japan, 4-6 December, 1989*, Seiten 223–240. North-Holland/Elsevier Science Publishers, 1989.

[ACM97] Abiteboul, Serge, Sophie Cluet und Tova Milo: *Correspondence and Translation for Heterogeneous Data.* In: Afrati, Foto N. und Phokion Kolaitis (Herausgeber): *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, Band 1186 der Reihe *Lecture Notes in Computer Science*, Seiten 351–363, New York, NY, USA, 1997. Springer Verlag.

[ACM98] Abiteboul, Serge, Sophie Cluet und Tova Milo: *A logical view of structured files.* The VLDB journal, 7(2):96–114, 1998.

[AHV95] Abiteboul, Serge, Richard Hull und Victor Vianu: *Foundations of Databases.* Addison-Wesley, Reading, Mass., 1995.

[AKK95] Agarwal, Shailesh, Christopher Keene und Arthur M. Keller: *Architecting Object Applications for High Performance with Relational Databases.* In: *OOPSLA Workshop on Object Database Behavior, Benchmarks, and Performance, Austin, TX, October 1995*, 1995.

[AMN+01] Alon, Noga, Tova Milo, Frank Neven, Dan Suciu und Victor Vianu: *Typechecking XML Views of Relational Databases.* In: *6th Annual IEEE Symposium on Logic in Computer Science, 16-19 June 2001, Boston, Massachusetts, USA, Proceedings*, Seiten 421–430. IEEE Computer Society, 2001.

[AS95] Abiteboul, Serge und Cassio Souza Dos Santos: *IQL(2) : A Model with Ubiquitous Objects*. Verso Report 73, INRIA, 1995.

[BCF+05] Boag, Scott, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie und Jérôme Siméon: *XQuery 1.0: An XML Query Language*. Technischer Bericht, W3C, 2005.

[BDH03a] Braganholo, Vanessa, Susan Davidson und Carlos Heuser: *Reasoning about the updatability of XML views over relational databases*. Technical Report MS-CIS-03-13, PENN Database Research Group, 2003.

[BDH03b] Braganholo, Vanessa P., Susan B. Davidson und Carlos A. Heuser: *Using XQuery to build updatable XML views over relational databases*. Technical Report MS-CIS-03-18, Dept. of Computer and Information Sciences, University of Pennsylvania, 2003.

[BDK92] Bancilhon, François, Claude Delobel und Paris Kanellakis: *Building an Object-Oriented Database System — The Story of O_2*. Morgan Kaufmann Publishers, 1992.

[BFS00] Buneman, Peter, Mary Fernández und Dan Suciu: *UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion*. VLDB Journal, 9(1):76–110, 2000.

[BKSW91] Barsalou, Thierry, Arthur M. Keller, Niki Siambela und Gio Wiederhold: *Updating Relational Databases through Object-Based Views*. In: Clifford, James und Roger King (Herausgeber): *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, May 29-31, 1991, Denver, Colorado*, Seiten 248–257, New York, NY, USA, 1991. ACM Press Books.

[BL98] Bentayeb, Fadila und Dominique Laurent: *View Updates Translations in Relational Databases*. In: Quirchmayr, Gerald, Erich Schweighofer und Trevor J. M. Bench-Capon (Herausgeber): *Database and Expert Systems Applications, 9th International Conference, DEXA '98, Vienna, Austria, August 24-28, 1998, Proceedings*, Band 1460 der Reihe *Lecture Notes in Computer Science*, Seiten 322–331. Springer Verlag, 1998.

[BM99] Beeri, Catriel und Tova Milo: *Schemas for Integration and Translation of Structed and Semi-Structured Data*. In: *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, Nummer 1540 in *Lecture Notes in Computer Science*, Seiten 296–313. Springer Verlag, 1999.

[BPSM+04] Bray, Tim, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler und François Yergeau: *Extensible Markup Language (XML) 1.0 (Third Edition)*. Technischer Bericht, W3C, 2004.

[BS81] Bancilhon, François und Nicolas Spyratos: *Update Semantics of Relational Views*. ACM Transactions on Database Systems (TODS), 6(4):557–575, 1981.

[BW97] Brown, Kyle und Bruce Whitenack: *Crossing Chasms - A Pattern Language for Object-RDBMS Integration*. In: *OOPSLA'97 course notes*, 1997.

[CB00] Cattell, Rick G. G. und Douglas Barry: *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers, 2000.

[CC05] Cabibbo, Luca und Antonio Carosi: *Managing Inheritance Hierarchies in Object/Relational Mapping Tools*. In: Oscar Pastor, João Falcão e Cunha (Herausgeber): *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, Nummer 3520 in *Lecture Notes in Computer Science*, Seiten 135–150. Springer, 2005.

[CDG+99] Comon, Hubert, Mac Dauchet, Rémi Gilleron, Florent Jacquemard, Danis Lugiez, Sophie Tison und Marc Tommasi: *Tree Automata Techniques and Applications*. Université Charles de Gaulle, Lille, 1999.

[CDSS98] Cluet, Sophie, Claude Delobel, Jérôme Siméon und Katarzyna Smaga: *Your Mediators Need Data Conversion!* In: Laura M. Haas, Ashutosh Tiwary (Herausgeber): *Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, Seiten 177–188, New York, NY, USA, 1998. ACM Press Books.

[CK85] Copeland, George P. und Setrag Khoshafian: *A Decomposition Storage Model*. In: Navathe, Shamkant B. (Herausgeber): *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, 1985*, Band 14 der Reihe *SIGMOD Record*, Seiten 268–279. ACM, 1985.

[Clu98] Cluet, Sophie: *Designing OQL: Allowing Objects to be Queried*. Verso Report 144, INRIA, 1998.

[Cod70] Codd, E. F.: *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13(6):377–387, 1970.

[Cod74] Codd, E.F.: *Recent Investigations in Relational Data Base Systems.* In: Rosenfeld, Jack L. (Herausgeber): *Information Processing 74, Proceedings of IFIP Congress 74, Stockholm, Sweden, August 5-10, 1974. North-Holland, 1974*, Seiten 1017–1021, 1974.

[CP84] Cosmadakis, Stavros S. und Christos H. Papadimiriou: *Updates of Relational Views.* Journal of the ACM, 31(4):742–760, 1984.

[CR03] Claypool, Kajal T. und Elke A. Rundensteiner: *Sangam: A Framework for Modeling Heterogeneous Database Transformations.* In: *Fifth International Conference on Enterprise Information Systems (ICEIS-2003), Angers, France, April 23-26, 2003*, 2003.

[D+90] Deux, O. et al.: *The Story of O_2.* IEEE Transactions on Knowledge and Data Engineering, 2(1):91–108, 1990.

[Dat00] Date, Christopher J.: *An Introduction to Database Systems.* Addison-Wesley, 2000.

[DB78] Dayal, Umeshwar und Philip A. Bernstein: *On the Updatability of Relational Views.* In: *Fourth International Conference on Very Large Data Bases, September 13-15, 1978, West Berlin, Germany*, Seiten 368–377. IEEE Computer Society, 1978.

[DB82] Dayal, Umeshwar und Philip A. Bernstein: *On the Correct Translation of Update Operations on Relational Views.* ACM Transactions on Database Systems (TODS), 7(3):381–416, 1982.

[DD95] Darwen, Hugh und C. J. Date: *The Third Manifesto.* SIGMOD Record, 24(1):39–49, 1995.

[DFF+05] Draper, Denise, Peter Fankhauser, Mary F. Fernández, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon und Philip Wadler: *XQuery 1.0 and XPath 2.0 Formal Semantics.* Technischer Bericht, W3C, 2005.

[FC85] Furtado, Anthony L. und Marco A. Casanova: *Updating Relational Views.* In: Kim, Won, David S. Reiner und Don S. Batory (Herausgeber): *Query Processing in Database Systems*, Seiten 127–142. Springer, 1985.

[FKMP03] Fagin, Ronald, Phokion G. Kolaitis, Renée J. Miller und Lucian Popa: *Data Exchange: Semantics and Query Answering.* In: *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings*, Nummer 2572 in *Lecture Notes in Computer Science*, Seiten 207–224. Springer Verlag, 2003.

[GMPS03] Greenwald, Michael B., Jonathan T. Moore, Benjamin C. Pierce und Alan Schmitt: *A Language for Bi-Directional Tree Transformations*. Technischer Bericht MS-CIS-03-08, Department of Computer and Information Science, University of Pennsylvania, 2003.

[GMQP+95] Garcia-Molina, Hector, Dallan Quass, Yannis Papakonstantinou, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman und Jennifer Widom: *The TSIMMIS Approach to Mediation: Data Models and Languages*. In: *The Second International Workshop on Next Generation Information Technologies and Systems (NGITS '95), 27 - 29 June 1995, Hotel Carlton, Naharia, Israel*, 1995.

[GMUW02] Garcia-Molina, Hector, Jeffery D. Ullman und Jennifer D. Widom: *Database Systems — The Complete Book*. Prentice Hall, 2002.

[GPZ88] Gottlob, Georg, Paolo Paolini und Roberto Zicari: *Properties and Update Semantics of Consistent Views*. ACM Transactions on Database Systems (TODS), 13(4):486–524, 1988.

[GS97] Gécseg, F. und M. Steinby: *Tree languages*. In: *Handbook of Formal Languages — Volume 3. Beyond words, Background and Application*. Springer Verlag, Berlin, 1997.

[Heg84] Hegner, Stephen J.: *Canonical View Update Support through Boolean Algebras of Components*. In: *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, April 2-4, 1984, Waterloo, Ontario, Canada*, Seiten 163–172, New York, NY, USA, 1984. ACM Press Books.

[Hib05] *Hibernate 3.0 http://www.hibernate.org*, 2005.

[HST99] Härder, Theo, Günter Sauter und Joachim Thomas: *The Instrinsic Problems of Structural Heterogeneity and an Approach to their Solution*. The VLDB journal, 8(1):25–43, Mai 1999.

[Kay05] Kay, Michael: *XSL Transformations (XSLT) Version 2.0*. Technischer Bericht, W3C, 2005.

[Kel86] Keller, Arthur M.: *Choosing a View Update Translator by Dialog at View Definition Time*. In: Chu, Wesley W., Georges Gardarin, Setsuo Ohsuga und Yahiko Kambayashi (Herausgeber): *Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*, Seiten 467–474. Morgan Kaufmann Publishers, 1986.

[Kel87] Keller, Arthur M.: *Comments on Bancilhon and Spyratos' Update Semantics and Relational Views*. ACM Transactions on Database Systems (TODS), 12(3):521–523, 1987.

[Kel95] Keller, Arthur M.: *Updating Relational Databases Through Views*. Doktorarbeit, Department of Computer Science, Stanford University, 1995.

[Kel97] Keller, Wolfgang: *Mapping Objects To Tables - A Pattern Language*. In: *Second European Conference on Pattern Languages of Programming and Computing*, 1997.

[KLW95] Kifer, Michael, Georg Lausen und James Wu: *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM, 42(4):741–843, Juli 1995.

[KMWZ91] Kemper, Alfons, Guido Moerkotte, Hans-Dirk Walter und Andreas Zachmann: *GOM: A Strongly Typed Persistent Object Model With Polymorphism*. In: Appelrath, Hans-Jürgen (Herausgeber): *Datenbanksysteme in Büro, Technik und Wissenschaft, GI-Fachtagung, Kaiserslautern, 6.-8. März 1991, Proceedings*, Band 270 der Reihe *Informatik-Fachberichte*, Seiten 198–217. Springer Verlag, 1991.

[KU84] Keller, Arthur M. und Jeffrey D. Ullman: *On Complementary and Independent Mappings on Databases*. In: Yormark, Beatrice (Herausgeber): *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, Band 14 der Reihe *SIGMOD Record*, Seiten 143–148, Washington, DC, USA, 1984. ACM Press Books.

[KW00] Keller, Arthur M. und Gio Wiederhold: *Penguin: Objects for Programs, Relations for Persistence*. In: Chaudhri, Akmal (Herausgeber): *Object Data Management*. Wiley, 2000.

[KW01] Keller, Arthur M. und Gio Wiederhold: *Penguin: Objects for Programs, Relations for Persistence*. In: Chaudhri, Akmal B. und Roberto Zicari (Herausgeber): *Succeeding with Object Databases*. Wiley, 2001.

[Lec03] Lechtenörger, Jens: *The impact of the constant complement approach towards view updating*. In: *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-11, San Diego, California, USA*, Seiten 49–55. ACM Press, 2003.

[LL95] Lang, Stefan M. und Peter C. Lockemann: *Datenbankeinsatz*. Springer Verlag, 1995.

[LV02] Lechtenbörger, Jens und Gottfried Vossen: *On the Computation of Relational View Complements*. In: Popa, Lucian (Herausgeber): *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, Seiten 142–149. ACM, 2002.

[LV03] Lechtenbörger, Jens und Gottfried Vossen: *On the Computation of Relational View Complements*. ACM Transactions on Database Systems (TODS), 28(2):175–208, 2003.

[Mas84] Masunaga, Yoshifumi: *A Relational Database View Update Translation Mechanism*. In: Dayal, Umeshwar, Gunter Schlageter und Lim Huat Seng (Herausgeber): *Tenth International Conference on Very Large Data Bases, August 27-31, 1984, Singapore, Proceedings*, Seiten 309–320. Morgan Kaufmann Publishers, 1984.

[MN04] Martens, Wim und Frank Neven: *Frontiers of Tractability for Typechecking Simple XML Transformations*. In: Nascimento, Mario A., M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley und K. Bernhard Schiefer (Herausgeber): *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, Seiten 23–34. Morgan Kaufmann, 2004.

[MS99] Milo, Tova und Dan Suciu: *Type Inference for Queries on Semistructed Data*. In: *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, Seiten 215–226, New York, NY, USA, 1999. ACM Press Books.

[MSV00] Milo, Tova, Dan Suciu und Victor Vianu: *Typechecking for XML Transformers*. In: *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, Seiten 11–22. ACM, 2000.

[MZ98] Milo, Tova und Sagit Zohar: *Using Schema Matching to Simplify Heterogeneous Data Translation*. In: Gupta, Ashish, Oded Shmueli und Jennifer Widom (Herausgeber): *Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Seiten 122–133, San Fransisco, CA, USA, 1998. Morgan Kaufmann Publishers.

[PGMW95] Papakonstantinou, Yannis, Hector Garcia-Molina und Jennifer Widom: *Object Exchange Across Heterogeneous Information Sources*.

In: Yu, Philip S. und Arbee L. P. Chen (Herausgeber): *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, Seiten 251–260. IEEE Computer Society, 1995.

[PV00] Papakonstantinou, Yannis und Victor Vianu: *DTD Inference for Views of XML Data.* In: *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, Seiten 35–46. ACM, 2000.

[PVM+02] Popa, Lucian, Yannis Velegrakis, René J. Miller, Mauricio A. Hernández und Roland Fagin: *Translating Web Data.* In: ? (Herausgeber): *VLDB 2002*, Seite ? ?, 2002.

[QRS+95] Quass, Dallan, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman und Jennifer Widom: *Querying Semistructured Heterogeneuos Information.* In: Ling, Tok Wang, Alberto O. Mendelzon und Laurent Vieille (Herausgeber): *Deductive and Object-Oriented Databases, Fourth International Conference, DOOD'95, Singapore, December 4-7, 1995, Proceedings*, Band 1013 der Reihe *Lecture Notes in Computer Science*, Seiten 319–344. Springer, 1995.

[RF03] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa: *Data Exchange: Getting to the Core.* In: *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-11, San Diego, California, USA*, Seiten 90–101. ACM Press, 2003.

[RS79] Rowe, Lawrence A. und Kurt A. Shoens: *Data Abstraction, Views and Updates in RIGEL.* In: Bernstein, Philip A. (Herausgeber): *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June 1*, Seiten 71–81, New York, NY, USA, 1979. ACM Press Books.

[RS96] Riedel, Holger und Marc H. Scholl: *The CROQUE-Model — Formalization of the Data Model and Query Language.* Konstanzer Schriften in Mathematik und Informatik 23, University of Konstanz, Dept. of Mathematics and Computer Science, Konstanz, Germany, 1996.

[SHT+77] Shu, Nan C., Barron C. Housel, R. W. Taylor, Sakti P. Ghosh und Vincent Y. Lum: *EXPRESS: A Data EXtraction, Processing, amd REStructuring System.* ACM Transactions on Database Systems (TODS), 2(2):134–174, 1977.

[Shu98] Shu, Hua: *Using Constraint Satisfaction for View Update Translation.* In: Prade, Henri (Herausgeber): *13th European Conference on Artificial*

Intelligence, Brighton, UK, August 23-28 1998, Proceedings, Seiten 33–37. John Wiley and Sons, Chichester, 1998.

[SLR+92] Scholl, Marc H., Christian Laasch, Christian Rich, Markus Tresch und Hans-Jörg Schek: *The COCOON Object Model*. Technical Report 193, ETH Zürich, Dept. of Computer Science, Zürich, Switzerland, Dezember 1992.

[Suc01] Suciu, Dan: *On Database Theory and XML*. SIGMOD Record, 30(3):39–45, 2001.

[Suc02] Suciu, Dan: *The XML Typechecking Problem*. SIGMOD Record, 31(1), March 2002.

[Top93] Topaloglou, Thodoros: *Storage Management for Knowledge Bases*. In: Bhargava, Bharat K., Timothy W. Finin und Yelena Yesha (Herausgeber): *CIKM 93, Proceedings of the Second International Conference on Information and Knowledge Management, Washington, DC, USA, November 1-5, 1993*, Seiten 95–104. ACM, 1993.

[Top05] *Oracle Application Server 10g - TopLink http://www.oracle.com*, 2005.

[Toz01] Tozawa, Akihiko: *Towards Static Type Checking for XSLT*. In: *Proceedings of the 2001 ACM Symposium on Document Engineering, Atlanta, Georgia, USA, November 9-10, 2001*. ACM, 2001.

[Ull89] Ullman, Jeffrey D.: *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.

[VLA95] Verhoef, Marcel, Thomas Liebich und Robert Amor: *A Multi-Paradigm Mapping Method Survey*. In: Fisher, Law und Luiten (Herausgeber): *Modeling Of Building Through Their Life-Cycle*, CIB/W78-TG10 publication 180, Seiten 233–247. Stanford University, 1995.

[Vos00] Vossen, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagement-Systeme*. Oldenbourg Verlag, 2000.

[WG85] Waite, William M. und Gerhard Goos: *Compiler construction*. Springer Verlag, New York, 1985.

[Wie92] Wiederhold, Gio: *Mediators in the Architecture of Future Information Systems*. IEEE Computer, 25(3):38–49, March 1992.

[WK02] Wassilios Kazakos, Andreas Schmidt, Peter Tomczyk: *Datenbanken und XML: Konzepte, Anwendungen, Systeme*. Springer, 2002.

[Yu97] Yu, Shen: *Regular Languages*. In: *Handbook of Formal Languages — Volume 1. Word, Language, Grammar*. Springer Verlag, Berlin, 1997.

Anwendungsprogramme wie Applikationsserver folgen in ihrem Aufbau häufig einer mehrschichtigen Architektur, wobei eine Sicht beschreibt, wie die Daten aus der Datenbank in die Anwendung ausgelesen werden. Wird auf der Sicht eine Änderungsoperation durchgeführt, muss der Zustand der Datenbank entsprechend nachgeführt werden, so dass insgesamt wieder ein konsistenter Systemzustand erreicht wird. Im Fall von Sichten innerhalb des relationalen Datenmodells zeigt die Arbeit, dass für eine weitaus größere Klasse von Sichten bei Änderungsoperationen eine entsprechende Nachführung bestimmbar ist, als dies bei bisherigen Ansätzen der Fall ist. Des Weiteren wird gezeigt, dass bei Sichten zwischen einem relationalen Datenbankschema und einem objektorientierten Anwendungsschema zur Speicherung der Daten mehr Abbildungen zur Verfügung stehen, als sie von existierenden Mechanismen angeboten werden.

ISBN 3-86644-041-3

www.uvka.de