Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

Diplomarbeit

Analyse des Einflusses der Mächtigkeit von relationalen Anfragesprachen auf die Komplexität bei der Auswertung von Anfrageindexen

Verfasser:

Stephan Schosser

25. Oktober 2004

Betreuer:

Dipl.-Inf. Hagen Höpfner

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg
Germany

Schosser, Stephan:
Analyse des Einflusses der Mächtigkeit von relationalen Anfragesprachen auf die Komplexität bei der Auswertung von Anfrageindexen
Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 2004.

Danksagung

Mein Dank gilt an erster Stelle Hagen Höpfner, der mich bei der Erstellung dieser Arbeit mit hilfreichen Diskussionen unterstützte und die Arbeit betreute.

Weiterer Dank gilt all den Menschen, die mich während der Diplomarbeit, aber auch schon während des gesamten Studiums unterstützten. Hierzu zählen meine Eltern Peter und Marianne Schosser, die mich nicht nur finanziell, sondern auch durch Telefonate und Gespräche während des gesamten Studiums unterstützten. Meiner Freundin Kerstin sei besonders für ihr Verständnis für meinen Zeitmangel am Ende der Diplomarbeit, wie während der Prüfungsphasen, gedankt. Kai-Uwe Sattler und Hagen Höpfner sei dafür gedankt, dass sie in mir das Interesse für das Thema Datenbanken geweckt und gesteigert haben, indem sie mir die Möglichkeit gaben an verschiedenen ihrer Projekte mitzuarbeiten. Hierdurch konnte ich mir grundlegende Kenntnisse aneignen, die bei der Bearbeitung dieser Diplomarbeit von großer Hilfe waren.

Schließlich sei den Menschen und Institutionen gedankt, die mein Studium finanzierten, ohne sie hätte diese Arbeit nicht entstehen können. Dazu gehört zunächst das Europäische Patentamt, dessen großzügiges Stipendium den Hauptteil meiner monatlichen Einkünfte ausmachte. Weiterhin sei meinen Eltern gedankt, die diesen Betrag weiter aufstockten. Der Otto-von-Guericke Universität, der Robert Bosch GmbH und der Metop GmbH sei gedankt, da sie mir die Möglichkeit von bezahlter Beschäftigung gaben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einleitung

1975 gründeten Bill Gates und Paul Allen das Unternehmen Microsoft mit der Vision jeden Schreibtisch und jeden Haushalt mit einem PC auszustatten [Ded01]. Diese Vision prägte lange Zeit die Entwicklungen im Computerbereich. Noch bevor alle Haushalte mit einem Computer ausgestattet werden können, Anfang 2003 kamen in Deutschland auf ca. 39 Millionen Haushalte ca. 29 Millionen PCs [GKK03, Sta03], löst Joseph Weizenbaum diese Vision ab. Er glaubt, dass der Computer in den nächsten Jahren aus unserem Bewusstsein verschwindet [HBB+03].

Dabei wird nicht davon ausgegangen, dass die Computer durch eine neue Technologie abgelöst werden, sondern es wird erwartet, dass die heute üblichen Personal Computer (PCs) an Bedeutung verlieren. Dafür werden Geräte in der alltäglichen Umgebung des Menschen intelligenter, indem Computer, für den Benutzer unsichtbar, als Embedded Devices in sie eingebettet werden [HBB+03]. So ausgerüstete Geräte können über drahtlose Verbindungen Daten miteinander austauschen und diese Daten mittels Sensoren aufwerten. So bietet BMW bereits einen Service an, der über das Navigationssystem im PKW Zugriff auf aktuelle Verkehrsinformationen und Auskünfte wie den Standort der nächsten Tankstelle bietet[1]. Siemens stellte Ende 2003 die ersten Haushaltgeräte vor, die über Handys oder TabletPCs gesteuert werden können[2].

Szenarien, in welchen Embedded Devices, Personal Digital Assistants (PDAs) und Laptops dem Benutzer zu jeder Zeit und an jedem Ort den Zugriff auf die verschiedensten Informationen ermöglichen, werden unter dem Begriff Mobile Computing zusammengefasst [IK97]. Der Benutzer ist hier nicht mehr gezwungen eine feste Position im Netzwerk einzuhalten und verfügt so über fast grenzenlose Mobilität [IK97].

Mobile Computing stellt neue technische Anforderungen an die benutzten Endgeräte. So müssen mobile Computer, da sie leichter als stationäre Computer in die Hände Dritter gelangen, gut gegen unbefugten Zugriff geschützt werden [Fed01]. Durch die gesteigerte Mobilität der Teilnehmer entstehen zusätzliche Herausforderungen: Die Bandbreite kabelloser Verbindungen ist knapp und störanfälliger als Leitungen des Festnetzes. Phasen,

[1] siehe http://www.bmw-telematik.de
[2] siehe http://www.servehome.de

in welchen keine Verbindung zu anderen Rechnern besteht, müssen durch Verfahren wie das Speichern eventuell später benötigter Daten (Cachen) ausgeglichen werden [Fed01]. Im Bereich der Datenbanken wird hierfür das semantische Cachen vorgeschlagen [CSL98], das auch in Client-Server Umgebungen [DFJ+96], OLAP Systemen [DRSN98] und heterogenen Systemen [GG97] Anwendung findet.

Konventionelle Cache-Schemata in relationalen Datenbanken sind entweder seiten- oder tupelbasiert. Diese Ansätze besitzen einen Nachteil: Es werden keine semantischen Informationen mit den Daten übertragen, so dass der Client nicht selbst entscheiden kann, ob er eine neue Anfrage oder Teile einer neuen Anfrage aus dem Cache beantworten kann. Dies führt zu einem hohen Maß an Abhängigkeit und zusätzlichen Anforderungen an die Bandbreite des Netzwerkes [LLS99]. Eine Alternative bietet das semantische Cachen. Hierbei werden neben den Tupeln, die eine Datenbankanfrage beantworten, noch Informationen zur entsprechenden Anfrage gespeichert. So können dann genau die Tupel bestimmt werden, die zusätzlich notwendig sind, um eine neue Datenbankanfrage lokal zu beantworten.

Bei der Anfragebearbeitung entscheidet der Client, wie die Anfrage beantwortet wird. Ist das Anfrageergebnis bereits vollständig im Cache enthalten (Inklusion), so wird das Anfrageergebnis ohne Datenaustausch mit dem Server ermittelt. Ist die Anfrage nur teilweise durch den Cache beantwortbar (Überlappung), so können Teile des Anfrageergebnisses direkt aus dem Cache zurückgegeben werden. Die Anfrage wird dann so angepasst, dass vom Server nur noch die fehlenden Daten übermittelt werden. Damit kann das Datentransfervolumen deutlich reduziert werden. Datenbankanfragen, für die keine Informationen im Cache enthalten sind (Disjunktheit), werden vollständig auf dem Server ausgeführt und das Ergebnis wird an den Client übermittelt. Ist die Verbindung zum Server unterbrochen, kann im Fall von Inklusion die Anfrage beantwortet werden. Im Fall der Überlappung können bereits die ersten Tupel als Ergebnis übergeben werden. Durch die zusätzlichen semantischen Informationen kann der Benutzer darüber informiert werden, dass das Ergebnis nicht vollständig ist.

Es ist bekannt, dass das Problem festzustellen, ob ein Anfrageergebnis in dem Ergebnis einer anderen Anfrage enthalten ist oder nicht, im allgemeinen Fall unerfüllbar ist. Daher schränken existierende Ansätze zum semantischen Cachen die Mächtigkeit der Anfragesprache drastisch ein. Oft werden nur Selektionen und Projektionen auf einzelne Relationen zugelassen. Keiner der existierenden Ansätze betrachtet Mengenoperationen. Ziel dieser Arbeit ist es, den Einfluss der Mächtigkeit von relationalen Anfragesprachen auf die Komplexität bei der Auswertung von Anfrageindexen zu ermitteln.

Die Arbeit gliedert sich dazu wie folgt: Zunächst wird in Kapitel 2 ein Beispielszenario vorgestellt. Die theoretischen Grundlagen für die weitere Arbeit werden in Kapitel 3 erläutert. In Kapitel 4 werden Algorithmen für das Anfrageenthaltenseinproblem und das Anfrageerfüllbarkeitsproblem vorgestellt. In Kapitel 5 werden die für das semantische Cachen notwendigen Algorithmen und Cacheersetzungsstrategien vorgestellt. Kapitel 6 beschreibt das Evaluationsszenario, sowie die Evaluationsergebnisse. Kapitel 7 fasst die Arbeit zusammen und gibt einen kurzen Ausblick.

Kapitel 2

Beispielszenario

In diesem Kapitel wird ein Beispielszenario vorgestellt, das, soweit möglich, in der weiteren Arbeit für Beispiele herangezogen wird.

Ein Manager hat Zugriff auf eine Datenbank mit Daten bezüglich seiner Mitarbeiter. Die Daten sind in einer Datenbank gespeichert, die dem ER-Schema in Abbildung 2.1 (Seite 3) entspricht.

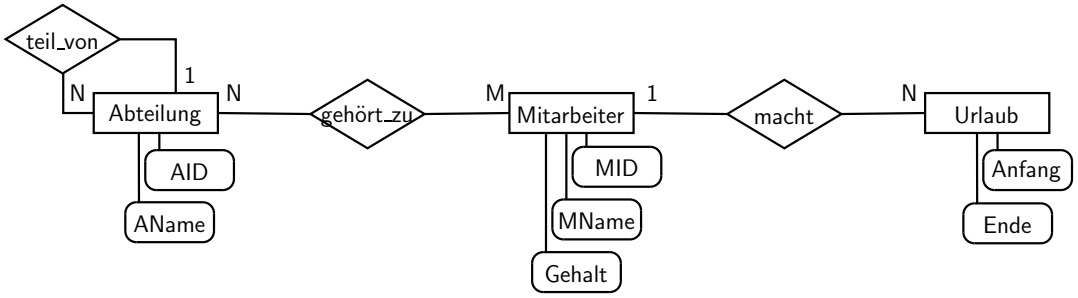

Abbildung 2.1: ER-Schema des Szenarios

Die Datenbank enthält die vier Relationen $Urlaub$, $Mitarbeiter$, $Mitarbeiter\_Abteilung$ und $Abteilung$ mit den folgenden Relationenschemata:

$Urlaub(MID, Anfang, Ende)$

$Mitarbeiter(MID, MName, Gehalt)$

$Mitarbeiter\_Abteilung(MID, AID)$

$Abteilung(AID, AName, MAID)$

In der Relation $Urlaub$ werden die Ferien der Mitarbeiter gespeichert. Sie besitzt die Attribute MID für die ID des entsprechenden Mitarbeiters, $Anfang$ als Zeitpunkt des Urlaubanfangs und $Ende$ als Urlaubsende. Die Relation $Mitarbeiter$ enthält die Mitarbeiternummer MID, den Namen des Mitarbeiters $MName$ sowie die Höhe seines Gehalts $Gehalt$. Jeder Mitarbeiter kann einer oder mehreren Abteilungen zugeordnet werden und jede Abteilung besitzt mehrere Mitarbeiter. Dieser Zusammenhang wird in

der Relation *Mitarbeiter_Abteilung* gespeichert, die sowohl die Nummer des Mitarbeiters MID, als auch die Nummer der Abteilung AID enthält. Die Relation *Abteilung* besitzt neben einer ID AID und einem Namen $AName$ noch eine ID $MAID$, in welcher die ID der Abteilung gespeichert ist, der diese Abteilung organisatorisch unterstellt ist. Abbildung 2.2 (Seite 4) zeigt entsprechende Relationen.

Mitarbeiter

MID	MName	Gehalt
123	Mayer	1234.50
124	Maler	2234.50
1	Grossmann	10000.00
127	Schmidt	1334.50

Urlaub

MID	Anfang	Ende
1	01.08.2004	14.08.2004
1	05.12.2004	12.12.2004
123	02.09.2004	09.09.2004
123	15.10.2004	19.10.2004

Abteilung

AID	AName	MAID
1	Management	null
2	Controlling	1
3	IT	1

Mitarbeiter_Abteilung

MID	AID
1	1
1	2
123	2
124	2
127	3

Abbildung 2.2: Relationen der Beispieldatenbank

Kapitel 3

Grundlagen

Ziel dieses Kapitels ist die Definition von relationalen Datenbanken und einer relational vollständigen Algebra. Weiterhin werden unterschiedliche Darstellungsweisen von Datenbankanfragen und damit verbundene grundlegende Algorithmen erläutert. Die hier eingeführten Algorithmen und Definitionen bilden die Grundlage der weiteren Arbeit.

3.1 Definitionen

Zunächst werden grundlegende Begriffe relationaler Datenbanken informal erläutert und schließlich formal präzisiert. Die formale Darstellung folgt, soweit nicht anders spezifiziert, den Definitionen in [HS00].

Informal kann eine Relation als eine zweidimensionale Tabelle betrachtet werden. Jede Zeile dieser Tabelle entspricht einem Tupel und jede Spalte einem Attribut. Eine Domäne beschreibt eine Menge von Werten, welche die Attribute von Relationen annehmen können [Dat00]. In der Praxis sind Domänen (auch Wertebereiche) meist Standard-Datentypen wie *integer*, *string*, *real* oder *boolean* [HS00]. In dieser Arbeit wird zwischen den Wertebereichen *real* und *integer* unterschieden. Soweit nicht anders vermerkt, wird davon ausgegangen, dass die dargestellten Ergebnisse für den Wertebereich *real* für alle geordneten, dichten Wertebereiche gelten. Bei Ergebnissen für den Wertebereich *integer* wird davon ausgegangen, dass diese für alle diskreten Wertebereiche gelten. So lässt sich beispielsweise der Datentyp *string* auf den Datentyp *real* [Klu88] abbilden. Da alle anderen in relationalen Datenbanken vorkommenden Datentypen auf diese Datentypen abgebildet werden können [SKN89], werden im Weiteren nur die Datentypen *real* und *integer* betrachtet. Werden Ergebnisse vorgestellt, die nicht explizit auf einen dieser Datentypen verweisen, so gelten die Ergebnisse unabhängig vom gewählten Datentyp.

Die nicht-leere, endliche Menge $\mathcal{U}$ heißt das Universum der Attribute. Jedes Element $A \in \mathcal{U}$ wird als Attribut bezeichnet. $\mathcal{W} = \{W_1, ..., W_m\}$ ist eine Menge endlicher, nicht-leerer Mengen mit $m \in \mathbb{N}$. Jede Menge W_i wird Wertebereich oder Domäne genannt. Es existiert eine Funktion $dom : \mathcal{U} \rightarrow \mathcal{W}$, die jedem Attribut A einen Wertebereich $dom(A)$ zuordnet. Ein Wert $w \in dom(A)$ wird Attributwert für A genannt. Jedes Rela-

tionenschema R ist Teilmenge des Universums der Attribute $\mathcal{U}$. Ein Relationenschema $R = \{A_1, ..., A_n\}$ mit $n \in \mathbb{N}$ kann folglich als eine Menge von Attributen A_j betrachtet werden. Die Abbildung $t : R \rightarrow \bigcup_{i=1}^{n} W_i$ mit $t(A) \in dom(A)$ wird als Tupel bezeichnet. Ein Tupel ist demnach eine Abbildung, die jedem Attribut des zu Grunde liegenden Relationenschemas einen konkreten Attributwert aus dem Wertebereich des Attributs zuordnet. Eine endliche Menge von Tupeln über dem Relationenschema R wird als Relation r über R (kurz: $r(R)$) bezeichnet. Ein Datenbankschema $D = \{R_1, ..., R_m\}$ ist definiert als Menge von Relationenschemata R_k mit $m \in \mathbb{N}$. Eine Datenbank d über einem Datenbankschema D (kurz: $d(D)$) ist eine Menge von Relationen $d = \{r_1, ..., r_m\}$. Dabei besitzt jede Relation r_i mit $i \in \{1, ..., m\}$ das Relationenschema R_i. Eine Datenbank kann demnach als eine Menge von Instanzen von Relationenschemata betrachtet werden, die durch das zu Grunde liegende Datenbankschema definiert sind.

Auf das obige Beispiel übertragen (siehe Kapitel 2) besteht das Universum $\mathcal{U}$ aus der Menge der Attribute $\mathcal{U} = \{AID$, $AName$, $Anfang$, $Ende$, $Gehalt$, $MAID$, MID, $MName\}$. Eine sinnvolle Menge von Domänen sei $\mathcal{W} = \{datum, integer, real, string\}$. Die Funktion *dom* ordnet jedem Element aus $\mathcal{U}$ ein Element aus der Menge $\mathcal{W}$ zu. So könnten folgende Beziehungen gelten: $dom(Gehalt) = real$, $dom(MID) = integer$, $dom(MName) = string$ usw. Das Attribut Gehalt besitzt folglich den Wertebereich *real*, das Attribut MID den Wertebereich *integer* usw. Ein Attributwert w für das Attribut Gehalt wäre demnach der Wert 1234.50, da dieser Wert in der Domäne *real* liegt. Auf der Basis der Attribute wurden im Beispiel Relationenschemata definiert. Die benutzten Relationenschemata sind $Urlaub = \{MID, Anfang, Ende\}$, $Mitarbeiter = \{MID, MName, Gehalt\}$ usw. Das Tupel $t(Mitarbeiter) = \{MID$: 123, $MName$: '$Mayer$', $Gehalt$: 1234.50$\}$ ist ein Tupel auf dem Relationenschema $Mitarbeiter$, da die Attributwerte aus dem Wertebereich der entsprechenden Attribute stammen. Eine endliche Menge solcher Tupel auf dem Relationenschema $Mitarbeiter$ bildet die Relation $r(Mitarbeiter)$. Das Datenbankschema D der im Beispiel eingeführten Datenbank ist $D = \{Urlaub, Mitarbeiter, Abteilung, Mitarbeiter\_Abteilung\}$ und die Datenbank ist $d(D) = \{r(Urlaub), r(Mitarbeiter), r(Abteilung), r(Mitarbeiter\_Abteilung)\}$.

3.2 Relationale Vollständigkeit

Eine Sprache, die es erlaubt auf der Basis von Relationen einer Datenbank neue Relationen zu bestimmen, wird als relationale Anfragesprache bezeichnet [HS00]. Ein typischer Vertreter von relationalen Anfragesprachen ist die Structured Query Language (SQL) [Dat00]. Unterschiedliche Anfragesprachen unterstützen unterschiedliche Operationen auf Relationen. Abhängig von der Anzahl der unterstützten Operationen spricht man von unterschiedlich „mächtigen“ Anfragesprachen [Dat00].

Die Mächtigkeit einer Anfragesprache kann über die relationale Vollständigkeit ausgedrückt werden [HS00]. Eine Anfragesprache $\mathcal{L}$ ist genau dann relational vollständig, wenn jede der Basisoperationen der Relationenalgebra in $\mathcal{L}$ abgebildet werden kann und die Operanden der Basisoperationen in $\mathcal{L}$ beliebige Ausdrücke von $\mathcal{L}$ sein können [Dat00].

Dabei werden unter Basisoperationen Operationen verstanden, die innerhalb von $\mathcal{L}$ nicht mit Hilfe anderer Operationen ausgedrückt werden können [Dat00]. Eine minimale Menge von Basisoperationen sind die fünf Operationen Selektion (σ), Projektion (π), Kreuzprodukt ($\times$), Vereinigung ($\cup$) und Differenz ($-$) [Dat00]. Neben dieser minimalen Menge von Basisoperationen werden in der Literatur auch andere Mengen von Operationen als minimale Menge der Basisoperationen betrachtet. So wird das Kreuzprodukt oft durch die Operationen Verbund ($\bowtie$) und Umbenennung (β) ersetzt. Die daraus resultierende Menge der Basisoperationen σ, π, $\cup$, $-$, $\bowtie$ und β ist auch minimal [HS00].

Im Weiteren wird die aus den fünf Operationen Selektion, Projektion, Kreuzprodukt, Vereinigung und Differenz bestehende Menge von Basisoperationen als minimale Menge bezeichnet. Die Komplexität einer Anfragesprache wird an diesen Operationen gemessen. Hierfür werden zunächst die einzelnen Operationen definiert. Die Relationenalgebra wurde zuerst in [Cod72] festgelegt. Grundlegende Annahmen bezüglich der Relationen haben sich seither gewandelt, beispielsweise nummerierte Codd die Spalten einer Relation noch und versah sie nicht mit Attributnamen wie heute üblich. Daher wird im Folgenden, soweit nicht anders vermerkt, von den in aktuellen Lehrbüchern ([Dat00, HS00]) üblichen Definitionen ausgegangen.

3.2.1 Selektion

Die Selektion wählt aus einer Relation genau die Tupel aus, die eine gegebene Selektionsbedingung erfüllen. Hierbei ist folgende Syntax üblich:

$$\sigma_{Selektionsbedingung}(Relation)$$

Gegeben sei die Relation $r(R)$ mit dem Relationenschema $R = \{A_1, ..., A_n\}$. $\theta \in \{=, \neq, <, \leq, \geq, >\}$ sei die Menge der Vergleichssymbole. Eine atomare Bedingung p kann eine Konstantenselektion oder eine Attributselektion sein. Konstantenselektionen haben die Form $p = A_i\ \theta\ w$ mit $A_i \in \{A_1, ..., A_n\}$ und $w \in dom(A_i)$, während Attributselektionen die Form $p = A_i\ \theta\ A_j$ mit $A_i, A_j \in \{A_1, ..., A_n\}$ besitzen. Weiterhin kann eine Bedingung p aus mehreren verschiedenen mit den booleschen Operatoren $\wedge$, $\vee$ und $\neg$ verknüpften einfachen Vergleichen bestehen. Die Selektion ist definiert als:

$$\sigma_p(r(R)) = \{t | t \in r(R) \wedge p(t) = true\}$$

Die Menge der Mitarbeiter, die eine Mitarbeiternummer größer 100 und ein Gehalt größer 2000 besitzen, kann mit der Selektion $\sigma_{(MID>100)\wedge(Gehalt>2000)}(r(Mitarbeiter))$ auf der Beispielrelation $r(Mitarbeiter)$ (siehe Kapitel 2) bestimmt werden. Die Anfrage liefert eine Relation mit dem Relationenschema $R = \{MID, MName, Gehalt\}$. Die Relation besitzt das Tupel $t(R) = \{MID$: 124, $MName$: '$Maler$', $Gehalt$: $2234,50\}$.

3.2.2 Projektion

Wie die Selektion „verkleinert“ die Projektion eine gegebene Relation. Während bei der Selektion einzelne Tupel, also „Zeilen“, aus der Relation entfernt werden, werden hier Attribute, also „Spalten“, aus der Relation genommen.

Folgende Notation ist üblich:

$$\pi_{Attributmenge}(Relation)$$

Gegeben sei die Relation aus Abschnitt 3.2.1 und eine Menge von Attributen $\mathcal{A} \subseteq R$. Die Projektion ist definiert als

$$\pi_{\mathcal{A}}(r(R)) = \{t(\mathcal{A}) | t \in r(R)\}$$

Alle Abteilungsnummern und Abteilungsnamen, der in der Datenbank gespeicherten Abteilungen, werden durch die Projektion $\pi_{AID,AName}(r(Abteilung))$ (siehe Kapitel 2) bestimmt. Sie liefert eine Relation mit dem Relationenschema $R = \{AID, AName\}$ und den Tupeln $t_1(R) = \{AID$: 1, $AName$: '$Management$'$\}$, $t_2(R) = \{AID$: 3, $AName$: 'IT'$\}$ und $t_3(R) = \{AID$: 2, $AName$: '$Controlling$'$\}$.

3.2.3 Kreuzprodukt

Das Kreuzprodukt (oder kartesische Produkt) zweier Relationen kombiniert zwei Relationen. Dabei wird jedes Tupel der ersten Relation mit jedem Tupel der zweiten Relation zu einem neuen Tupel vereinigt. Hierbei ist folgende Syntax üblich:

$$(Relation1) \times (Relation2)$$

Gegeben seien die Relationen $r(R_1)$ und $r(R_2)$. [Cod72] definiert das Kreuzprodukt als

$$r(R_1) \times r(R_2) = \{t_1 \cup t_2 | t_1 \in r(R_1) \wedge t_2 \in r(R_2)\}.$$

Das Kreuzprodukt $(r(Mitarbeiter)) \times (r(Urlaub))$ der Relationen aus dem Beispiel (siehe Kapitel 2) bestimmt die Kombination aller möglichen Tupelkombinationen der Relationen $r(Mitarbeiter)$ und $r(Urlaub)$. Die Anfrage veranschaulicht ein Problem, das beim Kreuzprodukt auftreten kann: Das Relationenschema des Ergebnisses würde zwei Attribute mit dem Namen MID besitzen. Somit könnte nicht eindeutig auf eines der beiden Attribute zugegriffen werden. Daher gilt hier als Konvention, dass bei Kreuzprodukten mit gleichnamigen Attributen beiden Attributen der Name der Ausgangsrelation vorangestellt wird [Cod91, Ull88]. Somit ist das Relationenschema der Ergebnisrelation $R = \{Mitarbeiter.MID, MName, Gehalt, Urlaub.MID, Anfang, Ende\}$. Die Ergebnisrelation besitzt für jedes der vier Tupel der Relation $r(Mitarbeiter)$ vier Tupel von $r(Urlaub)$, also 16 Tupel. Zwei dieser Tupel sind: $t_1(R) = \{Mitarbeiter.MID$: 1, $MName$: '$Grossmann$', $Gehalt$: 10000.00, $Urlaub.MID$: 123, $Anfang$: 15.10.2004, $Ende$: 19.10.2004$\}$ sowie $t_2(R) = \{Mitarbeiter.MID$: 1, $MName$: '$Grossmann$', $Gehalt$: 10000.00, $Urlaub.MID$: 1, $Anfang$: 05.12.2004, $Ende$: 12.12.2004$\}$.

Im Folgenden werden Kreuzprodukte nur auf Relationen mit unterschiedlichen Namen zugelassen. Auf diese Weise kann ein Problem der Namensgebung bei Attributen umgangen werden: Bei Kreuzprodukten, die zwei gleichnamige Relationen enthalten, führt auch das Voranstellen der Relationennamen vor den Attributnamen zu einer Relation mit uneindeutigen Attributbenennungen.

3.2.4 Vereinigung

Die Vereinigung zweier Relationen enthält alle Tupel der ersten und der zweiten Relation. Da das Ergebnis einer Vereinigung zweier Relationen wieder eine Relation bilden muss, können Vereinigungen nur aus Relationen erstellt werden, die vereinigungskompatibel sind [Cod91]. Zwei Relationen sind genau dann vereinigungskompatibel, wenn die Relationenschemata beider Relationen gleich viele Attribute besitzen und jedem Attribut der einen Relation ein Attribut der anderen Relation mit derselben Domäne zugeordnet werden kann [Cod91]. Für den Vereinigungsoperator ist folgende Notation üblich:

$$(Relation1) \cup (Relation2)$$

Gegeben seien zwei Relationen $r(R_1)$ und $r(R_2)$. Die Vereinigung der beiden Relationen ist definiert als

$$r(R_1) \cup r(R_2) = \{t | t \in r(R_1) \vee t \in r(R_2)\}.$$

Im Beispiel (siehe Kapitel 2) existieren keine zwei vereinigungskompatiblen Relationen. Derartige Relationen können jedoch durch Anwendung des Projektionsoperators auf die vorhandenen Relationen abgeleitet werden. Die Menge aller Kombinationen aus Mitarbeiternummern und Mitarbeiternamen zusammen mit der Menge aller Abteilungsnummer/Abteilungsname-Kombinationen wird durch die Vereinigung $(\pi_{MID,MName} r(Mitarbeiter)) \cup (\pi_{AID,AName} r(Abteilung))$ bestimmt. Hierbei sei dahingestellt, ob das Ergebnis sinnvoll ist. Die Anfrage ist vereinigungskompatibel, da $dom(MID) = dom(AID)$ und $dom(MName) = dom(AName)$ gilt. Hier tritt erneut ein Problem mit der Benennung der Attribute auf: Zwei Relationen können auch dann vereinigt werden, wenn ihre Attribute, wie hier, keine identischen Namen besitzen. Aufgrund der Kommutativität des Vereinigungsoperators ist jeder Attributname des resultierenden Relationenschemas, der basierend auf der Reihenfolge der Operanden gewählt wird, inakzeptabel [Cod91]. Es lässt sich keine offensichtliche Regel zum Bestimmen der Attributnamen der Ergebnisrelation finden. Damit kann das Relationenschema der Ergebnisrelation nicht eindeutig definiert werden [Ull88]. In [Ull88] wird daher vorgeschlagen, für das Ergebnis der Vereinigung keine Attributnamen anzugeben. Die Ergebnisrelation besitzt sieben Tupel: $t_1(R) = \{AID + MID$: 123, $AName + MName$: '$Mayer$'$\}$, $t_2(R) = \{AID + MID$: 1, $AName + MName$: '$Management$'$\}$, usw.

3.2.5 Differenz

Die Differenz zweier Relationen enthält alle Tupel der ersten Relationen, die nicht in der zweiten Relation enthalten sind. Wie auch die Vereinigung ist die Differenz eine Mengenoperation, bei der beide beteiligten Relationen vereinigungskompatibel sein müssen. Folgende Syntax ist für diese Operation üblich:

$$(Relation1) - (Relation2)$$

Gegeben seien die Relationen wie in Abschnitt 3.2.4, so ist die Differenz zweier Relationen definiert als

$$r(R_1) - r(R_2) = \{t | t \in r(R_1) \wedge t \notin r(R_2)\}$$

Die Mitarbeiternummern der Mitarbeiter, die nicht in Urlaub fahren werden, wird durch die Differenz $(\pi_{MID}r(Mitarbeiter)) - (\pi_{MID}r(Urlaub))$ bestimmt. Auch hier ist kein eindeutiges Relationenschema definiert, da die Differenz auch auf zwei Relationen mit unterschiedlichen Relationenschemata ermittelt werden kann. Die Tupel der Ergebnisrelation sind $t_1(R) = \{MID\text{: } 124\}$ und $t_2(R) = \{MID\text{: } 127\}$.

3.2.6 Attributnamen

Wie in Abschnitt 3.2.3 beschrieben, können als Ergebnis von Kreuzprodukten Relationen entstehen, deren Attributnamen nicht eindeutig sind. Um dies zu umgehen, wird im Folgenden angenommen, dass alle Attribute mit *Relationenname_Attributname* benannt sind.

3.3 Relationenkalkül

Neben der Relationenalgebra, die Anfragen als Operationen auf Datenbankinstanzen betrachtet, kann auch der Relationenkalkül genutzt werden, um Anfragen zu beschreiben [HS00]. Im Gegensatz zur Relationenalgebra, die beschreibt wie ein Anfrageergebnis berechnet werden kann, beschreibt der Relationenkalkül, was zum Anfrageergebnis gehört [HS00, Dat00].
Eine allgemeine Anfrage im Relationenkalkül hat die Form

$$\{f(\{x_1, ..., x_n\})|p(\{x_1, ..., x_n\})\}.$$

Dabei bezeichnen die x_i eine Menge von Variablen. Über diesen Variablen wird die Bedingung $p(\{x_1, ..., x_n\})$ ausgewertet [HS00]. Jede dieser Variablen x_i besitzt einen bestimmten Bereich W_i, an den sie gebunden ist. W_i kann der Wertebereich eines Datentyps oder eine Kollektion von Datenbankobjekten sein. f ist eine Ergebnisfunktion und p ein Selektionsprädikat über den Variablen x_i [HS00]. Ein Selektionsprädikat besteht aus Attributselektionen, Konstantenselektionen (siehe Abschnitt 3.2.1) und Relationenprädikaten [HS00]. Ein Relationenprädikat hat die Form $r(x_1, ..., x_m)$ wobei r der Name einer Relation $r(R)$ und m die Anzahl der Attribute des Relationenschemas ist. Je eine Variable oder Konstante x_i des Relationenprädikats muss auf den Wertebereich je eines Attributs abgebildet werden können [HS00]. Attributselektionen, Konstantenselektionen und Relationenprädikate können durch die Operatoren $\wedge, \vee, \neg$ und die Quantoren $\forall, \exists$ zu komplexeren Selektionsprädikaten verknüpft werden. Relationenprädikate seien im Folgenden mit r und die arithmetischen Vergleiche, Attributselektion und Konstantenselektion, mit c bezeichnet.

Anfragen im Relationenkalkül werden in zwei Schritten abgearbeitet. Zunächst werden alle Belegungen für die Menge der freien Variablen $\{x_1, ..., x_n\}$ bestimmt, für die das Selektionsprädikat $p(\{x_1, ..., x_n\})$ wahr wird. In einem zweiten Schritt wird die Funktion f auf die Menge der Belegungen angewandt. Das Ergebnis dieser Funktion ist gleich dem Ergebnis der Anfrage [HS00].

3.3.1 Bereichskalkül

Im Folgenden wird der Bereichskalkül (englisch: domain relational calculus) als ein Relationenkalkül weiter konkretisiert. Daneben gibt es noch andere Kalküle, wie den Tupelkalkül, auf die hier nicht weiter eingegangen wird. Hierfür sei auf entsprechende Grundlagenliteratur verwiesen [Ull88, HS00, Dat00].

Im Bereichskalkül sind nur Variablen elementarer Datentypen, so genannte Bereichsvariablen, zulässig. Anfragen haben in diesem Kalkül die Form

$$\{a_1, ..., a_n | \Phi(a_1, ..., a_n)\}.$$

Hierbei ist als Ergebnisfunktion eine Liste aller freien Variablen $a_1, ..., a_n$ zulässig [Ull88]. Φ ist eine Formel über den Variablen $a_1, ..., a_n$. Für die Formel Φ gelten dieselben Regeln wie für das Selektionenprädikat im allgemeinen Relationenkalkül (siehe Abschnitt 3.3). In der Formel Φ können neben den freien Variablen a_i der Ergebnisfunktion noch weitere Variablen auftreten. Diese werden als nicht freie Variablen b_i bezeichnet. Sie sind im Bedingungsteil Φ mit dem Existenzquantor $\exists$ gebunden [HS00].

Im gegebenen Beispiel (siehe Kapitel 2) lautet eine Bereichskalkülanfrage, welche die Mitarbeiternummern aller Mitarbeiter bestimmt, deren Name Grossmann ist: $\{a_1 | \exists b_1 \exists b_2 Mitarbeiter(a_1, b_1, b_2) \wedge b_1 = 'Grossmann'\}$. Die nicht freien Variablen, hier b_1 und b_2, sind im Bedingungsteil immer mit Existenzquantoren ausgezeichnet. Daher kann neben dieser Schreibweise die vereinfachte Schreibweise $\{a_1 | Mitarbeiter(a_1, b_1, b_2) \wedge b_1 = 'Grossmann'\}$ genutzt werden [HS00]. Die Bereichskalkülanfrage besitzt zwei atomare Formeln, die Konstantenselektion $b_1 = 'Grossmann'$ und das Relationenprädikat $Mitarbeiter(a_1, b_1, b_2)$. Das Relationenprädikat trägt dabei den Namen der Relation $Mitarbeiter$. Die Variablen a_1, b_1 und b_2 repräsentieren je ein Attribut dieser Relation. Diese Anfrage kann noch weiter vereinfacht werden, indem die Konstante $'Grossmann'$ als Parameter des Literals $Mitarbeiter(a_1, b_1, b_2)$ eingesetzt wird [HS00]. Die so weiter vereinfachte Anfrage lautet $\{a_1 | Mitarbeiter(a_1, 'Grossmann', b_2)\}$.

Auf diese Weise können Anfragen, die nur konjunktiv verknüpfte Relationenprädikate und Selektionsprädikate der Form $x_i = w$ enthalten, in Anfragen, die ausschließlich aus Relationenprädikaten bestehen, überführt werden.

3.3.2 Sichere Bereichskalkülanfragen

Relationen, die durch den Bereichskalkül beschrieben werden, sind nicht immer endlich. So liefert die Anfrage

$$\{a_1, a_2 | \neg R(a_1, a_2)\}$$

eine unendlich große Menge von Paaren (a_1, a_2), die nicht in der Relation R enthalten sind [Ull88]. Um derartige Anfragen, so genannte unsichere Anfragen, zu verhindern, wird die syntaktische Mächtigkeit der Bereichskalkülanfragen eingeschränkt [HS00].

Die Einschränkungen folgen dabei alle der Grundidee, dass jede freie Variable a_i in $\Phi(a_1, ..., a_n)$ durch positives Auftreten in einer Konstanteselektion $a_i = w$ oder in einem Relationenprädikat $R(..., a_i, ...)$ an endliche Bereiche gebunden werden muss [HS00].

Die Einschränkungen sind im Einzelnen [Ull88]:

1. Der $\forall$ Quantor darf nicht benutzt werden. Diese Regel schränkt die Ausdrucksmächtigkeit des Bereichskalküls nicht ein, da jede Formel $(\forall x)p$ äquivalent der Formel $\neg(\exists x)\neg p$ ist. Der $\forall$ Quantor kann somit durch Ersetzung durch obigen Ausdruck aus allen Formeln eliminiert werden.

2. Immer wenn zwei Ausdrücke p_1 und p_2 durch den $\vee$ Operator verknüpft sind, müssen die beiden Formeln die gleiche Menge an freien Variablen besitzen. Solche Verknüpfungen haben demnach immer die Form $p_1(x_1, ..., x_n) \vee p_2(x_1, ..., x_n)$.

3. Enthält eine Formel eine Subformel $p_1 \wedge ... \wedge p_m$, bestehend aus einer Konjunktion einer oder mehrerer Formeln p_i, so müssen alle Variablen, die in einer Formel p_i frei auftreten, begrenzt sein. Eine Variable x_j ist dann begrenzt, wenn sie mindestens in einem Ausdruck p_i auftritt, der eine der folgenden Bedingungen erfüllt:

   - x_j ist eine freie Variable eines Ausdrucks p_i, wobei p_i weder negiert noch ein arithmetischer Vergleich ist.
   - p_i hat die Form $x_j = w$ oder $w = x_j$, wobei w eine Konstante ist.
   - p_i hat die Form $x_j = x_k$ oder $x_k = x_j$ und x_k ist eine begrenzte Variable.

4. Weiterhin ist eine Anfrage nur dann sicher, wenn der $\neg$-Operator ausschließlich auf Subformeln der Form $p_1 \wedge ... \wedge p_m$ angewandt wird, die den oben beschriebenen Bedingungen genügen.

Sichere Bereichskalkülanfragen sind nicht so ausdrucksmächtig wie unsichere Bereichskalkülanfragen. Dennoch kann gezeigt werden, dass auch sichere Bereichskalkülanfragen relational vollständig sind [Ull88]. Im Folgenden werden, soweit nicht anders spezifiziert, unter Bereichskalkülanfragen stets sichere Bereichskalkülanfragen verstanden.

3.3.3 Entsprechungen von Operationen im Bereichskalkül

Die in Abschnitt 3.2 definierten Operationen lassen sich alle auf den sicheren Bereichskalkül abbilden. Seien die zwei Relationenschemata $R_1(A_1, ..., A_n)$ und $R_2(B_1, ..., B_n)$ gegeben, so gelten die folgenden Äquivalenzbeziehungen zwischen Ausdrücken der Relationenalgebra und Ausdrücken des Bereichskalküls [HS00]. Um die Äquivalenzbeziehung zu veranschaulichen, werden die zur Illustration der Operationen der relationalen Algebra genutzten Anfragen (siehe Abschnitt 3.2) entsprechenden Bereichskalkülanfragen gegenübergestellt.

- Selektion

  $$\sigma_\Phi(r(R_1)) \hat{=} \{a_1, ..., a_n | R_1(a_1, ..., a_n) \wedge \Phi'\}$$

  Der Ausdruck Φ' wird aus Φ gewonnen, indem alle $A_i \in \Phi$ durch a_i ersetzt werden.

Die Anfrage $\sigma_{(MID>100)\wedge(Gehalt>2000)}(r(Mitarbeiter))$ in der Relationenalgebra entspricht der Anfrage $\{a_1, a_2, a_3 | Mitarbeiter(a_1, a_2, a_3) \wedge a_1 > 100 \wedge a_3 > 2000\}$ im Bereichskalkül.

- Projektion
  $\pi_{\overline{A}}(r(R_1)) \hat{=} \{a_1, ..., a_k | \exists b_1 ... \exists b_n (R_1(b_1, ..., b_n) \wedge a_1 = b_{i_1} \wedge ... \wedge a_k = b_{i_k})\}$
  Dabei setzt sich die Attributliste wie folgt zusammen: $\overline{A} = (A_{i_1}, ..., A_{i_k})$
  Die Anfrage $\pi_{AID,AName}(r(Abteilung))$ in der Relationenalgebra entspricht der Anfrage $\{a_1, a_2 | \exists b_1 \exists b_2 \exists b_3 (Abteilung(a_1, a_2, a_3) \wedge b_1 = a_1 \wedge b_2 = a_2\}$ im Bereichskalkül.

- Kreuzprodukt
  $r(R_1) \times r(R_2) \hat{=} \{a_1, ..., a_n, a_{n+1}, ..., a_{n+m} | R_1(a_1, ..., a_n) \wedge R_2(a_{n+1}, ..., a_{n+m})\}$
  Das Kreuzprodukt $(r(Mitarbeiter)) \times (r(Urlaub))$ entspricht der sicheren Bereichskalkülanfrage $\{a_1, a_2, a_3, a_4, a_5, a_6 | Mitarbeiter(a_1, a_2, a_3) \wedge Urlaub(a_4, a_5, a_6)\}$.

- Vereinigung $(n = m)$
  $$r(R_1) \cup r(R_2) \hat{=} \{a_1, ..., a_n | R_1(a_1, ..., a_n) \vee R_2(a_1, ..., a_n)\}$$
  Die Anfrage $(\pi_{MID,MName} r(Mitarbeiter)) \cup (\pi AID, ANamer(Abteilung))$ der relationalen Algebra kann somit in die Anfrage $\{a_1, a_2 | Mitarbeiter(a_1, a_2, b_1) \vee Abteilung(a_1, a_2, b_2)\}$ des Bereichskalküls überführt werden. Dabei sei darauf hingewiesen, dass für die in der Anfrage enthaltene Projektion die vereinfachte Schreibweise genutzt wurde (siehe Abschnitt 3.3.1).

- Differenz $(n = m)$
  $$r(R_1) - r(R_2) \hat{=} \{a_1, ..., a_n | R_1(a_1, ..., a_n) \wedge \neg R_2(a_1, ..., a_n)\}$$
  Die Relationenalgebraanfrage $(\pi_{MID} r(Mitarbeiter)) - (\pi_{MID} r(Urlaub))$ entspricht der Bereichskalkülanfrage $\{a_1 | Mitarbeiter(a_1, b_1, b_2) \wedge \neg Urlaub(a_1, b_3, b_4)\}$. Auch hier wird für die Projektionen die vereinfachte Schreibweise genutzt.

Durch rekursive Anwendung dieser Äquivalenzbeziehungen können die Operationen einer relational vollständigen Algebra in Anfragen eines sicheren Bereichskalküls überführt werden [HS00].

3.3.4 Variablennamen

Die Variablen von Bereichskalkülanfragen werden entsprechend der Attributnamen der von ihnen repräsentierten Attribute gewählt. So kann die Abbildung von Anfragen der Relationenalgebra auf Bereichskalkülanfragen und damit auf die im Weiteren vorgestellten graphischen Repräsentationen (siehe Abschnitt 3.5) erleichtert werden, da keine Abbildungsfunktion der Attributnamen auf die Variablennamen eingeführt werden muss. Weiterhin garantiert dieses Vorgehen, dass alle Variablen begrenzt sind, wie für sichere Bereichskalkülanfragen gefordert (siehe Abschnitt 3.3.2).

3.4 Konjunktive Anfragen

Konjunktive Anfragen sind Datenbankanfragen mit einer Ausdrucksmächtigkeit, die nicht relational vollständig ist. Solche Anfragen sind von großer Bedeutung in relationalen Datenbanken, da fast jede Datenbankanfrage eine konjunktive Anfrage ist oder konjunktive Anfragen beinhaltet [LY85].

In der relationalen Algebra kann eine konjunktive Anfrage durch Selektionen, Projektionen und Kreuzprodukte ausgedrückt werden. Wobei Selektionsprädikate nur konjunktiv verknüpft werden dürfen. Durch Äquivalenzumformungen [SH99] kann aus jeder Anfrage bestehend aus Selektionen, Projektionen und Kreuzprodukten eine äquivalente Anfrage der Form

$$\pi_{A_1,...,A_n}(\sigma_\Phi(r(R_1) \times ... \times r(R_q))).$$

bestimmt werden [YL87]. Dabei besteht Φ aus konjunktiv verknüpften Attribut- und Konstanten-Selektionen. Mit Hilfe der in Abschnitt 3.3.3 eingeführten Entsprechungsregeln für relationale Operationen und Bereichskalkülanfragen kann diese Anfrage auf eine konjunktive Bereichskalkülanfrage zurückgeführt werden.

Somit haben konjunktive Anfragen im Bereichskalkül die Form

$$\{a_1, ..., a_n | \exists b_1 ... \exists b_m (p_1 \wedge ... \wedge p_o)\}.$$

Eine konjunktive Anfrage besitzt also n freie Variablen a_i und m mit dem Existenzquantor ausgezeichnete, gebundene Variablen b_j. Die konjunktiv Verknüpften p_k sind o atomare Formeln (englisch: atomic formula). Dabei kann jede atomare Formel entweder ein Relationenprädikat oder eine Ungleichung der Form $x_i \theta x_j$ oder $x_j \theta w$ mit $\theta \in \{<, \leq, =, \geq, >, \neq\}$ sein [Klu88].

In der Literatur wird die Mächtigkeit von konjunktiven Anfragen teilweise weiter eingeschränkt. Ein Ansatz lässt als Operator für die Selektion nur $\{=\}$ zu [CM77]. Derartige konjunktive Anfragen werden im Folgenden als konjunktive $\theta_=$-Anfragen bezeichnet. Andere Ansätze beschränken die Menge der in Selektionen zugelassenen Operatoren auf alle Vergleichsoperatoren ohne den $\neq$-Operator [GSW96a]. Diese Anfragen werden im Folgenden unter konjunktive $\theta_{\neg\neq}$-Anfragen zusammengefasst. Anfragen, die Selektionen mit allen Vergleichsoperatoren zulassen, heißen konjunktive θ_{ALL}-Anfragen. Die in dieser Arbeit vorgestellten Algorithmen können meist auf genau eine dieser Klassen konjunktiver Anfragen angewandt werden.

3.5 Graphische Repräsentation von Anfragen

Datenbankanfragen können nicht nur im Relationenkalkül oder in der Relationenalgebra abgebildet werden. Gerade Algorithmen zur Erfüllbarkeit und zum Enthaltensein von Anfragen, die eine wichtige Voraussetzung für das semantische Cachen bilden, greifen auf graphische Repräsentationen von Anfragen zurück. Die wichtigsten Abbildungen sowie einige ihrer Eigenschaften und auf ihnen basierende grundlegende Algorithmen werden in diesem Abschnitt vorgestellt.

3.5.1 Hypergraphen

Eine konjunktive $\theta_{=}$-Anfrage Q kann als Hypergraph $H_Q = (V, E)$ visualisiert werden. Hierfür werden entsprechende Bereichskalkülanfragen, wie in Abschnitt 3.3.1 geschildert, so vereinfacht, dass sie keine Existenzquantoren und keine arithmetischen Vergleiche enthalten. Jede in der Anfrage enthaltene Konstante w_i und jede Variable x_j wird durch einen Knoten $V = \{x_1, ..., x_n, w_1, ..., w_m\}$ und jedes Prädikat als Hyperkante $(x_1, ..., x_o, w_1, ..., w_p) \in E$ über die im Prädikat enthaltenen Variablen $x_1, ..., x_o$ und Konstanten $w_1, ..., w_p$ ausgedrückt [CR00].

Eine Anfrage, die alle Mitarbeiter in den Unterabteilungen der Abteilung '*Management*' bestimmt (siehe Kapitel 2), wird durch die Bereichskalkülanfrage $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7 | Mitarbeiter(a_1, a_2, a_3) \wedge Mitarbeiter\_Abteilung(a_1, a_4) \wedge Abteilung(a_4, a_5, a_6) \wedge Abteilung(a_6, 'Management', a_7)\}$ repräsentiert. Diese Anfrage kann in den in Abbildung 3.1 (Seite 15) dargestellten Hypergraphen überführt werden.

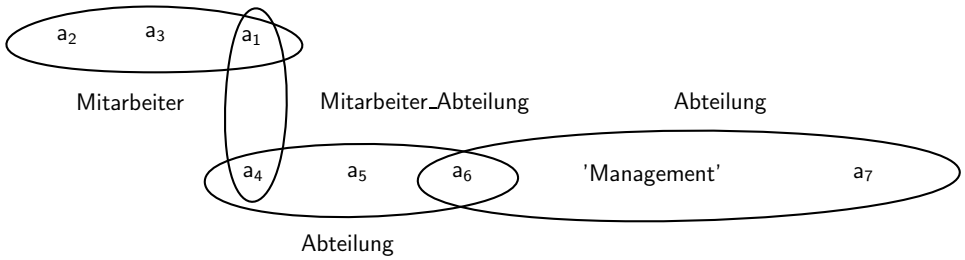

Abbildung 3.1: Beispiel eines Hypergraphen

Zyklenfreiheit

Eine Eigenschaft von Hypergraphen ist die Zyklenfreiheit (englisch: acyclicity). Ein Hypergraph besitzt einen Zyklus, wenn zwei Knoten des Hypergraphen über unterschiedliche Pfade erreichbar sind. So besitzt der Graph in Abbildung 3.1 (Seite 15) keinen Zyklus. Würde zusätzlich eine Kante $Zyklus(a_1, a_6)$ durch die Knoten a_1 und a_6 eingefügt werden, so besäße der Graph einen Zyklus, da a_6 von a_1 sowohl über die Kante $Zyklus(a_1, a_6)$, als auch über die Kanten $Mitarbeiter\_Abteilung(a_1, a_4)$ und $Abteilung(a_4, a_5, a_6)$ erreicht werden könnte. Graphen, die keine Zyklen enthalten, werden auch azyklisch, Graphen mit Zyklen zyklisch genannt [CR00]. Entsprechend werden auch Anfragen, die durch zyklische bzw. azyklische Hypergraphen repräsentiert werden, zyklisch bzw. azyklisch genannt [CR00].

Zur Prüfung eines Hypergraphen auf Zyklenfreiheit, wird die GYO-Reduktion angewandt: Seien E und F zwei Hyperkanten eines Hypergraphen H_Q, so dass in $E - F$ liegende Knoten in keiner anderen Hyperkante des Graphen H_Q liegen, dann wird E als Ohr (englisch: ear) bezeichnet. Die Entfernung des Ohrs E aus dem Hypergraphen H_Q wird als Ohrentfernung (englisch: ear removal) bezeichnet. Nun werden so lange alle Ohren aus dem Hypergraphen entfernt, bis der Hypergraph H_Q keine Ohren mehr besitzt.

Ist das Ergebnis einer GYO-Reduktion ein leerer Graph, so heißt der Ausgangsgraph H_Q azyklisch, andernfalls zyklisch [CR00].

Für azyklische Anfragen kann bei der GYO-Reduktion ein Verbundbaum (englisch: elimination tree oder join tree) aufgebaut werden. Dabei wird jeder Kante des Hypergraphen ein Knoten im Verbundbaum zugeordnet. Eine Kante zwischen zwei Knoten (v_1, v_2) des Verbundbaums existiert dann, wenn bei der GYO-Reduktion die vom Knoten v_1 repräsentierte Hyperkante vor ihrer Entfernung aus dem Hypergraphen nur eine Überlappung mit der Hyperkante, die durch den Knoten v_2 repräsentiert wurde, besaß [CR00].

Eine mögliche Reihenfolge bei der GYO-Reduktion des Hypergraphen aus Abbildung 3.1 (Seite 15) wäre: Zunächst werden die Kanten $Mitarbeiter(a_1, a_2, a_3)$ und $Abteilung(a_6, 'Management', a_7)$ entfernt. Dann wird die Kante $Abteilung(a_4, a_5, a_6)$ und schließlich die Kante $Mitarbeiter\_Abteilung(a_1, a_4)$ entfernt. Abbildung 3.2 (Seite 16) zeigt den entsprechenden Verbundbaum. Da der Hypergraph nach der GYO-Reduktion leer ist, sind sowohl die oben beschriebene Anfrage, als auch der Hypergraph azyklisch.

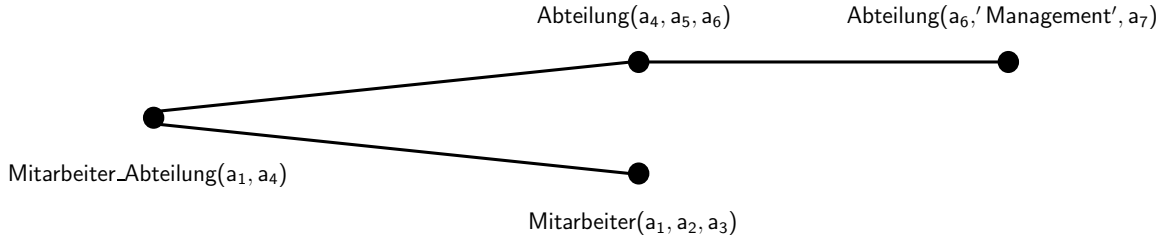

Abbildung 3.2: Verbundbaum des Hypergraphen aus Abbildung 3.1

In [TY84] wird gezeigt, dass ein Hypergraph in $O(|V| + |E|)$ auf Zyklenfreiheit überprüft werden kann. Dabei steht $|V|$ für die Anzahl der Knoten und $|E|$ für die Anzahl der Kanten des Hypergraphen. Der dort beschriebene Algorithmus gibt für azyklische Hypergraphen zusätzlich einen Verbundbaum aus.

Anfragebreite

Die Eigenschaft der Zyklenfreiheit kann auch auf zyklische Anfragen erweitert werden. Dabei wird versucht, den „Grad der Zyklizität“ eines Hypergraphen anzugeben [CR00]. Als Maß für die Zyklizität eines Graphen wird die Anfragebreite (englisch: query width) eingeführt [CR00].

Um die Anfragebreite zu bestimmen, werden zunächst für eine Anfrage alle Zerlegungsbäume (englisch: query decomposition trees) bestimmt.

Ein Zerlegungsbaum ist ein Baum $T = (I, F)$, bei dem jeder Knoten $i \in I$ eine Menge $X(i)$ aus Prädikaten und Attributen repräsentiert. Zunächst wird für jedes Prädikat s ein Knoten $i \in I$ mit $s \in X(i)$ erstellt. Die Menge der Knoten $\{i \in I | s \in i\}$, die dasselbe Prädikat beschreiben, bilden einen Teilbaum des Zerlegungsbaums. Zusätzlich

ergeben alle Knoten i, welche dasselbe Attribut $x_i \in X(i)$ enthalten, einen Teilbaum. Die maximale Anzahl von Elementen in $X(i)$ ist dann die Breite des Zerlegungsbaums [CR00].

Für jeden Hypergraphen können mehrere Zerlegungsbäume gefunden werden. Die Anfragebreite des Hypergraphen ist dann das Minimum aller Breiten der Zerlegungsbäume [CR00].

Als Beispiel sei die Anfrage $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7 | Mitarbeiter\_Abteilung(x_1, x_2) \wedge Abteilung(x_2, x_3, x_4) \wedge Abteilung(x_4, x_5, x_6) \wedge Abteilung(x_6, x_7, x_2)\}$ gegeben. Einer der Zerlegungsbäume ist in Abbildung 3.3 (Seite 17) gegeben. Der Knoten mit den meisten Elementen ist der Knoten mit $X(i) = \{Abteilung(x_4, x_5, x_6), x_2\}$. Er besitzt zwei Elemente $Abteilung(x_4, x_5, x_6)$ und x_2. Damit ist die Breite dieses Zerlegungsbaums 2. Unter der Annahme, dass es keinen Zerlegungsbaum für den gegebenen Hypergraphen gibt, der eine Breite kleiner 2 besitzt, ist auch die Anfragebreite 2.

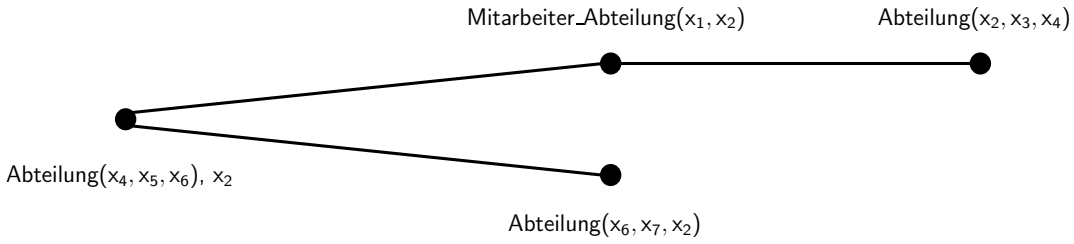

Abbildung 3.3: Beispiel eines Zerlegungsbaums

In [GLS99] wird gezeigt, dass das Problem der Feststellung, ob die Anfragebreite einer konjunktiven Anfrage maximal 4 ist, bereits NP-vollständig ist. Es wurde noch nicht bewiesen, dass die Berechnung der Anfragebreite NP-hart ist [CR00].

3.5.2 Graphen

Konjunktive $\theta_{\neg\neq}$-Anfragen Q, die ausschließlich Variablen desselben linear geordneten Wertebereichs enthalten, können durch gerichtete, benannte Graphen $G_Q(V, E)$ repräsentiert werden. Hierbei wird die Anfrage Q als logische Formel interpretiert. Eine konjunktive $\theta_{\neg\neq}$-Anfrage der Form $\{a_1, ..., a_n | \exists b_1 ... \exists b_m (p_1 \wedge ... \wedge p_o)\}$, deren atomare Formeln p_i entweder arithmetische Vergleiche oder Relationenprädikate sind, entspricht dann einer logischen Formel F_Q bestehend aus der Konjunktion der arithmetischen Vergleiche. Jede in der Formel F_Q enthaltene Variable x_i wird durch einen Knoten des Graphen G_Q repräsentiert. Somit ist die Knotenmenge $V = \{x_1, ..., x_n\}$, wobei n die Anzahl der unterschiedlichen in F_Q enthaltenen Variablen angibt. Jedem Knoten x_i wird weiterhin ein Wertebereich W_i zugeordnet. Die Grenzen des Wertebereichs W_i werden durch die in der Formel F_Q enthaltenen Konstantenselektionen festgelegt. Dabei führen Ungleichungen der Form $x_i < w$ bzw. $x_i \leq w$ zu oberen Intervallgrenzen und $x_i > w$ bzw. $x_i \geq w$ zu unteren Intervallgrenzen. Besitzt eine Gleichung die Form $x_i = w$ so wird die obere und die

untere Intervallgrenze gleich w gesetzt. Für jede in der Formel F_Q enthaltene Attributselektion der Form $x_j < x_k$ bzw. $x_j \leq x_k$ besitzt der Graph G_Q eine gerichtete Kante von x_j nach x_k, die mit $<$ bzw. $\leq$ bezeichnet ist. Eine solche Kante wird als $(x_j, x_k, <) \in E$ bzw. $(x_j, x_k, \leq) \in E$ notiert. Ungleichungen der Form $x_j > x_k$ und $x_j \geq x_k$ werden als Ungleichung $x_k < x_j$ bzw. $x_k \leq x_j$ behandelt. Hat eine Attributselektion die Form $x_j = x_k$, entspricht dies der Formel $x_j \leq x_k \wedge x_j \geq x_k$. Folglich sind in diesem Fall die zwei Kanten $(x_j, x_k, \leq)$ und $(x_k, x_j, \leq)$ Element der Kantenmenge E [GSW96b].

Als Beispiel dient die Anfrage $Q = \{a_1, a_2 | \exists b_1 \exists b_2 (a_1 \geq a_2 \wedge a_1 < b_2 \wedge a_2 \leq b_1 \wedge a_2 < 5 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 > 2 \wedge b_2 \leq 4)\}$. Diese Anfrage wird in einem ersten Schritt in die logische Formel $F_Q = (a_1 \geq a_2 \wedge a_1 < b_2 \wedge a_2 \leq b_1 \wedge a_2 < 5 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 > 2 \wedge b_1 \leq 4)$ überführt. Diese Formel wird mit Hilfe der obigen Vorschriften in den in Abbildung 3.4 (Seite 18) visualisierten Graph G_Q übertragen. Für jede Variable wird ein Knoten im Graphen erstellt. Aus den Ungleichungen der Form $x_i \ \theta \ w$ werden die zulässigen Wertebereiche der einzelnen Variablen abgeleitet. Hierbei ist darauf zu achten, dass Beziehungen mit $\{\leq\}$- oder $\{\geq\}$-Vergleichen zu geschlossenen Intervallen und $\{<\}$- oder $\{>\}$-Vergleiche zu geöffneten Intervallen führen. So besitzt b_1 den Wertebereich $]2, 4]$, da die 4 durch Ungleichung $b_1 \leq 4$ zugelassen wird, die 2 aber durch die Ungleichung $b_1 > 2$ aus dem Intervall ausgeschlossen wird. Die Kanten des Graphen werden aus den Ungleichungen der Form $x_i \ \theta \ x_j$ abgeleitet.

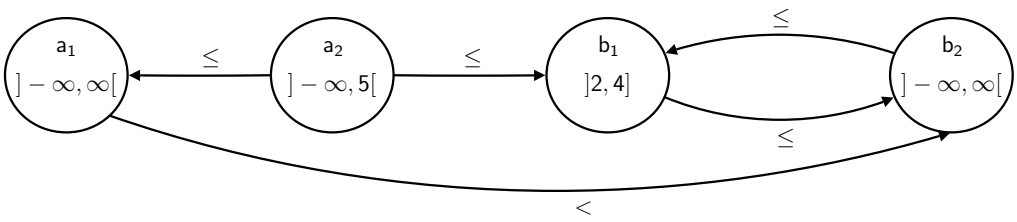

Abbildung 3.4: Beispiel eines Anfragegraphen G_Q

Implikation von Attributselektionen

Für jede Kombination zweier Knoten $x_i, x_j \in V$ gilt die Beziehung $x_i < x_j$ oder $x_i \leq x_j$, wenn im Graph G_Q ein Pfad vom Knoten x_i zum Knoten x_j existiert. Wenn alle Kanten des Pfades mit $\leq$ beschriftet sind, gilt $x_i \leq x_j$, sonst gilt $x_i < x_j$ [GSW96b]. Alle so implizierten Ungleichungen können mit Hilfe der transitiven Hülle des Graphen G_Q abgeleitet werden [GSW96b]. Die transitive Hülle G_Q^+ eines Graphen G_Q besteht aus den Kanten E und Knoten V des Ausgangsgraphen, sowie einer Kante von x_i nach x_j für jeden Pfad innerhalb G_Q mit einer Länge größer 1. Die zusätzlichen Kanten der transitiven Hülle eines Graphen werden ermittelt, indem für einen Knoten x_i alle Knoten x_j, die über eine eingehende Kante (x_j, x_i, θ_1) mit x_i, und alle Knoten x_k, die über eine ausgehende Kante (x_i, x_k, θ_2) mit x_i verbunden sind, ermittelt werden. θ_1 und θ_2 seien dabei die Beschriftung der Kanten. Die einzufügende Kante hat dann die Form (x_j, x_k, θ)

[OW90]. Wobei θ gleich $\leq$ ist, wenn sowohl θ_1 als auch θ_2 gleich $\leq$ sind. Sonst ist θ gleich $<$.

Abbildung 3.5 (Seite 19) zeigt die transitive Hülle des Graphen G_Q aus Abbildung 3.4 (Seite 18). Die neue Kante vom Knoten a_2 zum Knoten b_2 besitzt die Kantenbeschriftung $<$, obwohl ein Pfad über den Knoten b_1 existiert, dessen Kantenbeschriftungen alle $\leq$ sind. Allerdings existiert auch ein Pfad über den Knoten a_1, der eine Kante mit Beschriftung $<$ besitzt. Diese Kante ist für die Beschriftung der neuen Kante verantwortlich. Weiterhin wurden zwei Kanten eingefügt, die über die Knoten b_1 und b_2 auf sich selbst verweisen.

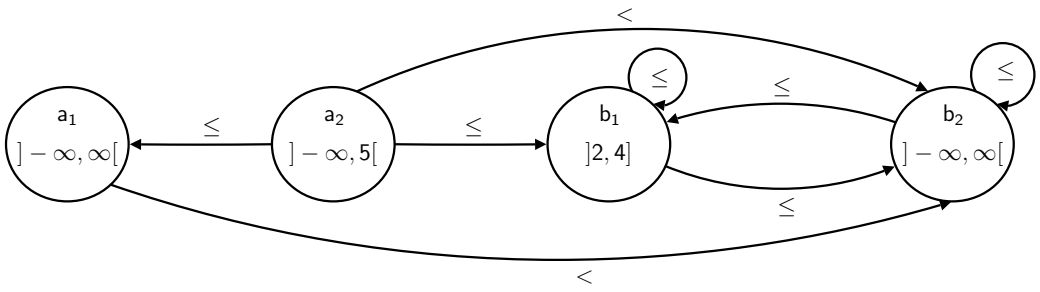

Abbildung 3.5: Transitive Hülle des Anfragegraphen G_Q

Der Algorithmus zur Berechnung der transitiven Hülle eines Graphen besitzt die Komplexität $O(|V| + |E*| + k^3)$. Dabei ist $|V|$ die Anzahl der Knoten des Graphen G_Q, $|E*|$ die Zahl der Kanten der transitiven Hülle und k die Anzahl der starken Zusammenhangskomponenten im Graph (siehe Abschnitt 3.5.2) [OW90]. Somit können in $O(|V| + |E*| + k^3)$ alle implizierten Ungleichungen einer Anfrage bestimmt werden.

Zyklen

Die Kanten $(b_1, b_1, \leq)$ und $(b_2, b_2, \leq)$ des Hypergraphen in Abbildung 3.5 (Seite 19) besitzen denselben Ein- und Ausgangsknoten. Solche Kanten treten immer dann auf, wenn der Graph einen Zyklus besitzt. Die Knoten eines Zyklus werden als stark zusammenhängend bezeichnet, da es von jedem Knoten im Zyklus einen Pfad zu jedem anderen Knoten im Graph gibt. Eine starke Zusammenhangskomponente (englisch: strongly connected component, kurz SCC) eines Graphen ist ein maximaler stark zusammenhängender Untergraph von G_Q [OW90]. Für jedes Paar von Knoten x_i, x_j einer starken Zusammenhangskomponente, gilt demnach $x_i\ \theta_1\ x_j$ und $x_j\ \theta_2\ x_i$, mit $\theta_1, \theta_2 \in \{<, \leq\}$. Sind alle θ_k von Kanten, die zwei Knoten einer starken Zusammenhangskomponente verbinden mit $\leq$ beschriftet, so sind alle Knoten, die Teil der Zusammenhangskomponente sind, gleich. In diesem Fall können diese Knoten zu einem Knoten zusammengefasst werden. Der Wertebereich des neuen Knotens ist dann durch das Maximum der unteren Intervallgrenzen und durch das Minimum der oberen Intervallgrenzen aller beteiligten Knoten festgelegt [GSW96b]. Der neue Knoten besitzt je eine Kante, für eine Verbindung eines Ursprungsknotens mit einem Knoten des Graphen, der nicht Teil der starken

Zusammenhangskomponente ist [GSW96b]. Enthält die starke Zusammenhangskomponente nur eine Kante, die mit $<$ beschriftet ist, so impliziert dies, dass die Anfrage Q eine Kontradiktion enthält. In diesem Fall ist die Anfrage nicht erfüllbar, es kann also keine Datenbankinstanz gefunden werden, so dass das Ergebnis der Anfrage mindestens ein Tupel enthält [GSW96b].

Der in Abbildung 3.4 (Seite 18) dargestellte Graph enthält eine starke Zusammenhangskomponente, bestehend aus b_1 und b_2. Diese beiden Knoten können folglich zu einem gemeinsamen Knoten $b_1\_b_2$ zusammengefasst werden, der den Wertebereich $W =]2, 4]$ besitzt. Der neue Knoten enthält die eingehenden Kanten $(a_1, b_1\_b_2, <)$ und $(a_2, b_1\_b_2, \leq)$.

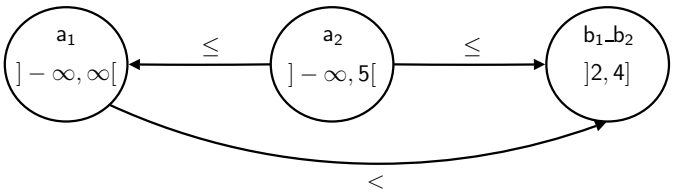

Abbildung 3.6: Anfragegraph G_Q ohne starke Zusammenhangskomponenten

Die Laufzeit des Algorithmus zur Bestimmung aller starken Zusammenhangskomponenten ist $O(|V|+|E|)$, wobei $|V|$ die Anzahl der Knoten und $|E|$ die Anzahl der Kanten des untersuchten Graphen ist [OW90].

Implikation von Konstantenselektionen

Neben zusätzlichen Attributselektionen können aus dem Graphen einer Anfrage G_Q Implikationen auf die Wertebereiche der Attribute abgelesen werden [GSW96b]. Existiert eine Kante (x_i, x_j, θ) von einem Knoten x_i, dessen Wertebereich unten durch w_{i_1} und oben durch w_{i_2} beschränkt ist, zu einem Knoten x_j mit den Wertebereichsgrenzen w_{j_1} und w_{j_2}, so impliziert dies die Ungleichungen $w_{i_2}\ \theta\ w_{j_2}$ und $w_{j_1}\ \theta\ w_{i_1}$. Mit Hilfe solcher Ungleichungen kann der tatsächliche Wertebereich jedes Attributs eingegrenzt werden. Hierfür werden die Knoten des Hypergraphen zunächst topologisch sortiert. Anschließend wird die untere Grenze des Wertebereichsintervalls w_i jedes einzelnen Knotens gesetzt, indem die Knoten in topologischer Reihenfolge durchlaufen werden. w_i ist dann das Maximum aller unteren Grenzen der Vorgängerknoten. Die obere Grenze wird anschließend analog gesetzt, indem die topologisch sortierten Knoten absteigend durchlaufen werden [GSW96b].

Abbildung 3.7 (Seite 21) zeigt den Graphen aus Abbildung 3.6 (Seite 20) mit angepassten Wertebereichsgrenzen. Für dieses Beispiel wurde der um die starken Zusammenhangskomponenten bereinigte Graph und nicht der Ausgangsgraph gewählt, da sich nur azyklische Graphen topologisch sortieren lassen [OW90]. Der Knoten a_1 besitzt den Wertebereich $W_1 =] - \infty, 4[$, da er über eine Kante mit der Beschriftung $<$ mit dem Knoten $b_1\_b_2$ verbunden ist. Im Wertebereich von a_2 dagegen ist die 4 noch enthalten, da

dieser Knoten über eine Kante mit der Beschriftung $\leq$ mit dem Knoten $b_1\_b_2$ verbunden wird.

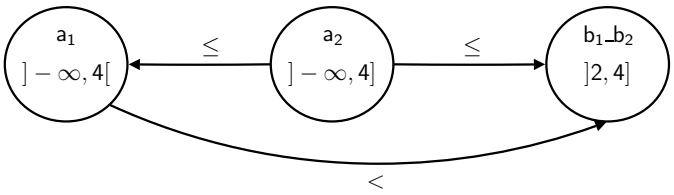

Abbildung 3.7: Anfragegraph G_Q mit angepassten Intervallgrenzen

Die Laufzeit des Algorithmus zum topologischen Sortieren und damit die Laufzeit zum Anpassen der Werterbereichsgrenzen ist $O(|V| + |E|)$ mit $|V|$ gleich der Anzahl der Knoten des Graphen und $|E|$ gleich der Anzahl der Kanten des Graphen [OW90].

3.5.3 Tableaus

Wie Hypergraphen ermöglichen Tableaus die Repräsentation konjunktiver $\theta_=$-Anfragen [SH99]. Ein Tableau ist eine Matrix bestehend aus einer Übersichtszeile (englisch: summary) z_0 und einer Menge von n Zeilen z_i mit $i = 1, ..., n$ [SY80]. Die Spalten des Tableaus entsprechen allen in der Anfrage auftretenden Attributen in fester Reihenfolge [SH99]. Die Symbole des Tableaus sind entweder ausgezeichnete Variablen a_i, nicht ausgezeichnete Variablen b_i, Konstanten w oder Leerzeichen (englisch: blanks). Während in der Übersichtszeile nur Konstanten, ausgezeichnete Variablen und Leerzeichen auftreten, sind in den übrigen Zeilen alle Symbole außer Leerzeichen zulässig [SY80].

So entspricht die Anfrage $\{a_1, a_2 | (Mitarbeiter(b_1, a_1, a_2) \wedge Urlaub(b_1, b_2, b_3)\}$ im Bereichskalkül dem Tableau in Abbildung 3.8 (Seite 21). Die ausgezeichneten Variablen des Tableaus entsprechen den freien Variablen in der Bereichskalküldarstellung [SH99], während die existentiell gebundenen Variablen des Bereichskalküls nicht ausgezeichneten Variablen entsprechen.

MID	MName	Gehalt	Anfang	Ende
	a_1	a_2		
b_1	a_1	a_2	b_4	b_5
b_1	b_6	b_7	b_2	b_3

Abbildung 3.8: Beispiel einer Tableauanfrage

Tableaus basieren auf der Annahme, dass zu jeder Zeit eine einzelne Universalrelation existiert, die alle Relationen der Datenbank enthält [ASU79b]. Das Relationenschema der Universalrelation besteht aus allen Attributen $A \in \mathcal{U}$ im Universum der Datenbank. Der Inhalt der Universalrelation wird durch Verbund aller Relationen in der Datenbank erstellt [SH99]. Alle zulässigen Anfragen können mittels Selektionen und Projektionen

auf der Universalrelation bestimmt werden [SH99]. Um eine eindeutige Universalrelation definieren zu können muss das zu Grunde liegende Datenbankschema azyklisch sein [HS00]. Da das Ergebnis jeder Anfrage wieder eine zulässige Relation sein muss, sind hier nur azyklische Anfragen zulässig.

Gekennzeichnete Tableaus (englisch: tagged tableaus) sind eine Erweiterung der Tableaus, so dass keine Universalrelation vorausgesetzt werden muss [ASU79b]. Damit können durch gekennzeichnete Tableaus auch zyklische Anfragen betrachtet werden. Hier besitzt jede Zeile, mit Ausnahme der Übersichtszeile, den Namen einer Relation als Kennung (englisch: tag) [SH99]. Die Attribute, die Teil des Relationenschemas dieser Relation sind, werden markiert. Nicht markierte Felder des Tableaus, in welchen nicht ausgezeichnete Variablen stehen, werden durch Leerzeichen ersetzt [ASU79b].

Für Tableauanfragen existieren einige effiziente Algorithmen zur Überprüfung auf Enthaltensein [ASU79a], allerdings sind hierfür weniger „natürliche“ Einschränkungen der Mächtigkeit der Anfragesprache als bei Hypergraphen nötig [Qia96]. Da Anfragen, die durch Tableaus repräsentiert werden können, auch durch Hypergraphen abbildbar sind, wird in der weiteren Arbeit auf die nähere Untersuchung von Tableaus verzichtet.

3.6 Erweiterung der Anfragemächtigkeit

Während Optimierungsmöglichkeiten konjunktiver Anfragen in der Literatur detailliert untersucht wurden, gibt es nur wenige Ansätze, die sich mit Anfragen mit größerer Mächtigkeit beschäftigen. Daher wird nun versucht, mächtigere Anfragen auf Kombinationen von konjunktiven Anfragen abzubilden.

Anfragen, die Selektionen, Projektionen und Kreuzprodukte enthalten, deren Selektionsprädikate nicht ausschließlich konjunktiv verknüpft sind, können wie jede boolesche Formel in die disjunktive Normalform überführt werden. Da die Äquivalenzbeziehung $\sigma_{p_1 \vee p_2}(r(R)) = \sigma_{p_1}(r(R)) \cup \sigma_{p_2}(r(R))$ gilt, kann jede Disjunktion mehrerer konjunktiver Anfragen als Vereinigung mehrerer konjunktiver Anfragen betrachtet werden. Somit kann ohne die Mächtigkeit der relationalen Anfragesprache einzuschränken auf den $\{\vee\}$-Operator in Selektionsbedingungen verzichtet werden, wenn die Anfragesprache Vereinigungen erlaubt. Somit ist eine Anfragesprache relational vollständig, wenn sie neben konjunktiven Anfragen auch die Mengenoperationen Vereinigung und Differenz unterstützt [ASU79b].

Mit Hilfe der algebraischen Optimierungsregeln für Ausdrücke der Relationenalgebra ist es möglich, jede beliebige konjunktive Anfrage in die Form $\pi_{\overline{A}}\sigma_p(r(R_1) \times ... \times r(R_n))$ zu transformieren [YL87]. Wobei $\overline{A}$ eine Menge von Attributen, p eine Selektionbedingung und $r(R_i)$ mit $i \in \{1, ..., n\}$ Relationen sind. Es ist wünschenswert, eine ähnlich Form für relational vollständige Anfragen zu finden. In [SY80] wird eine entsprechende Form für Anfragen vorgeschlagen, welche die Operatoren π, $\sigma_{x_i=w}$, $\sigma_{x_i=x_j}$, $\cup$ und $-$ enthalten. Dabei müssen allerdings Kombinationen der Operatoren in der Form $\pi...(... - ...)$ untersagt werden. Es wird gezeigt, dass derartige Anfragen in eine standardisierte Form transfor-

miert werden können [SY80]. Analog lassen sich auch Anfragen, deren Konstanten- und Attributselektionen andere Operatoren enthalten, in eine solche Form bringen. Hierfür wird eine Anfrage mit Hilfe der Optimierungsregeln der Relationenalgebra zunächst so umgeformt, dass Selektionen, Projektionen oder Kreuzprodukte keinen Operanden besitzen, der eine Vereinigung oder Differenz enthält [SY80]. So kann die Anfrage als Vereinigungen und Differenzen konjunktiver Anfragen betrachtet werden. Eine so umgeschriebene Anfrage Q' kann in eine äquivalente, disjunktive Normalform überführt werden [SY80]. Q' kann dann geschrieben werden als $Q' = Q_1 \cup ... \cup Q_n$, wobei jedes Q_i die Form $Q_i = Q_{i_1} \cap ... \cap Q_{i_j} \cap \overline{Q}_{i_{j+1}} \cap ... \cap \overline{Q}_{i_m}$ besitzt. Jedes Q_i entspricht also der Schnittmenge konjunktiver Anfragen und Komplementen von konjunktiven Anfragen. Da der Schnittmengenoperator ein Sonderfall eines Verbundes ist [SY80], entspricht $Q_{i_1} \cap ... \cap Q_{i_j}$ einer einzelnen konjunktiven Anfrage Q'_i. Der Ausdruck $\overline{Q}_{i_{j+1}} \cap ... \cap \overline{Q}_{i_m}$ kann mit Hilfe des Gesetzes von de Morgan in die Form $(1 - \cup_{k=j+1}^{m} Q_{i_k})$ überführt werden. Damit ist jeder Ausdruck Q_i äquivalent zu $Q'_i \cap (1 - \cup_{k=j+1}^{m} Q_{i_k})$ und damit äquivalent $Q' - \cup_{k=j+1}^{m} Q_{i_k}$. Eine Anfrage der Form $\cup_{k=j+1}^{m} Q_{i_k}$ ist eine Vereinigung konjunktiver Anfragen und eine Anfrage der Form $Q' - \cup_{k=j+1}^{m} Q_{i_k}$ wird Elementardifferenz (englisch: elementary difference) genannt [SY80]. Jede beliebige Anfrage kann folglich als Vereinigung von Elemtardifferenzen (englisch: union of elementary differences), $\cup_{i=1}^{n} Q'_i$, geschrieben werden [SY80], solange sie keine Kombinationen von Operatoren der Form $\pi...(... - ...)$ enthält. Die im Weiteren betrachteten Anfragen haben maximal die Mächtigkeit einer Vereinigung von Elementardifferenzen.

Kapitel 4

Algorithmen auf Anfragen

In diesem Kapitel werden die grundlegenden Algorithmen auf Datenbankanfragen vorgestellt, die für die Beantwortung von Anfragen beim semantischen Cachen vorausgesetzt werden. Dabei wird zunächst auf Algorithmen eingegangen, mit deren Hilfe überprüft werden kann, ob eine Anfrage erfüllbar ist. Anschließend werden Algorithmen betrachtet, die das Enthaltenseinproblem zweier Anfragen lösen. Diese Algorithmen können entscheiden, ob eine Datenbankanfrage in einer anderen Anfrage enthalten ist. Schließlich wird kurz auf das Anfragefaltungsproblem eingegangen.

4.1 Test auf Erfüllbarkeit

Dieser Abschnitt beschäftigt sich mit dem Test von Anfragen auf Erfüllbarkeit. Eine Datenbankanfrage Q ist genau dann erfüllbar, wenn eine Datenbank d existiert, so dass das Ergebnis der Anfrage auf dieser Datenbank $Q(d)$ nicht leer ist. Anfragen, die nur aus Projektionen oder Kreuzprodukten bestehen, sind genau dann erfüllbar, wenn die Projektion mindestens ein Attribut besitzt und am Kreuzprodukt mindestens eine Relation beteiligt ist. Dieser triviale Erfüllbarkeitstest soll im Weiteren nicht näher untersucht werden. Erst wenn Selektionen zugelassen werden, sind komplexere Algorithmen notwendig, um eine Anfrage auf Erfüllbarkeit zu testen. Daher werden in diesem Kapitel zunächst Algorithmen vorgestellt, die konjunktive $\theta_{=}$-Anfragen auf Erfüllbarkeit testen, anschließend werden entsprechende Algorithmen für konjunktive $\theta_{\neg\neq}$ Anfragen und θ_{ALL} Anfragen beschrieben. Abschließend erfolgt eine Zusammenfassung der vorgestellten Ergebnisse.

4.1.1 Konjunktive Anfragen mit $\theta_{=}$

Eine konjunktive $\theta_{=}$-Anfrage Q kann auf Erfüllbarkeit getestet werden, indem überprüft wird, ob die Anfrage Selektionen enthält, die eine Kontradiktion implizieren. Dies ist genau dann der Fall, wenn eine Beziehung der Form $w_1 = w_2$ impliziert wird und für die Konstanten w_1 und w_2 $w_1 \neq w_2$ gilt. Hierfür wird ein Feld a angelegt, indem für jedes Attribut der Anfrage ein Verweis auf einen Container existiert. In einem ersten Schritt

werden die Konstantenselektionen $x_i = w_i$ der Anfrage bearbeitet. Ist der Zeiger des Feldes von Element x_i in a noch nicht gesetzt, so wird er auf einen Container mit Inhalt w_i gesetzt. Verweist das Element x_i im Feld a bereits auf einen Container, so wird der darin enthaltene Wert w_j mit w_i verglichen. Sind beide Werte unterschiedlich, gilt also $w_i \neq w_j$, so wird der Algorithmus abgebrochen, die Anfrage ist nicht erfüllbar. Nachdem das Feld a so initialisiert wurde, werden die Attributselektionen abgearbeitet. Für jede Attributselektion werden die beteiligten Elemente x_i, x_j im Feld miteinander verglichen. Verweisen beide Elemente auf unterschiedliche Container, die mit den Werten w_i bzw. w_j initialisiert wurden und gilt $w_i \neq w_j$, so ist die Anfrage nicht erfüllbar, der Algorithmus wird abgebrochen. Ist nur einer der beiden Zeiger initialisiert, so wird der Zeiger des anderen Elements auf den bereits initialisierten Container gesetzt. Sind beide Zeiger nicht initialisiert, so wird ein neuer, leerer Container eingefügt, auf den beide Elemente durch Zeiger verweisen. Wurden alle Attributselektionen abgearbeitet, ohne dass ein Widerspruch gefunden wurde, so ist die Anfrage erfüllbar. Algorithmus 1 (Seite 26) zeigt das hier beschriebene Vorgehen im Pseudocode.

Algorithmus 1 : Erfüllbarkeitsproblem bei $\theta_=$-Anfragen

Eingabe : Anfrage Q
Ausgabe : TRUE oder FALSE

```
1  Erstellen eines Feldes a der Länge |A|
2  foreach Konstantenselektion x_i = w_i in Q do
3      if Element x_i in a nicht initialisiert then
4          Zeiger von x_i in a auf Container mit Inhalt w_i
5      else if Element x_i in a verweist auf Container mit Inhalt ≠ w_i then
6          return FALSE
7  end
8  foreach Attributselektion x_i = x_j in Q do
9      if Element x_i und x_j in a nicht initialisiert then
10         Erstelle leeren Container
11         Erstelle Verweis von x_i und x_j auf leeren Container
12     else if Element x_k in a mit k ∈ {i, j} nicht initialisiert then
13         Sei k̄ = {i, j} − k
14         Setze Container von x_k gleich Container von x_k̄
15     else if Elemente x_i und x_j in a verweisen auf Container mit w_i ≠ w_j then
16         return FALSE
17 end
18 return TRUE
```

Als Beispiel seien alle Mitarbeiter gesucht, die am 01.08.2004 genau einen Tag Urlaub machen. Im Bereichskalkül besitzt diese Anfrage die Form $\{a_1, a_2 | Mitarbeiter(a_1, a_2, b_1) \wedge Urlaub(a_1, b_2, b_3) \wedge b_2 = b_3 \wedge b_2 = 01.08.2004\}$. Die gegebene Anfrage enthält nur

die Konstantenselektion $b_2 = 01.08.2004$. Daher beinhaltet das Feld nach der Initialisierung nur einen Verweis (siehe (a) in Abbildung 4.1, Seite 27). Bei der Überprüfung der Attributselektion $b_2 = b_3$ wird festgestellt, dass der Eintrag des Feldes für b_2 bereits initialisiert wurde, b_3 aber noch nicht. Daher wird nun der Zeiger von b_3 auf den Container mit 01.08.2004 gesetzt (siehe (b) in Abbildung 4.1, Seite 27). Da die Anfrage keine weiteren Attributselektionen enthält, wird nun die Ausführung des Algorithmus abgebrochen. Die Anfrage ist erfüllbar.

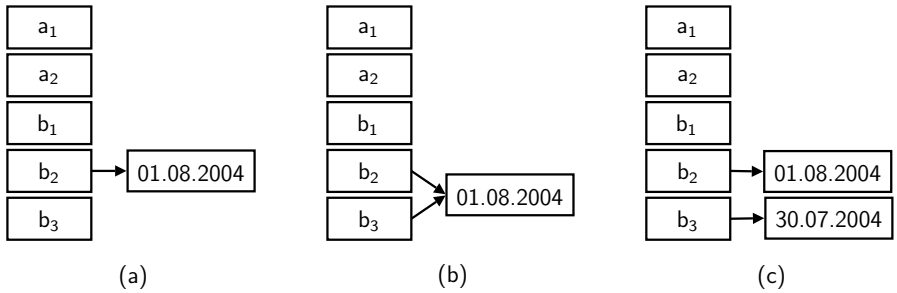

Abbildung 4.1: Beispiel für das Erfüllbarkeitsproblem bei $\theta_{=}$-Anfragen

Würde die Anfrage noch eine weitere Attributselektion $b_3 = 30.07.2004$ enthalten, so würde das Feld nach der Bearbeitung der Konstantenselektionen zwei initialisierte Zeiger enthalten (siehe (c) in Abbildung 4.1, Seite 27). Bei der Überprüfung der Attributselektion $b_2 = b_3$ würde nun festgestellt werden, dass die entsprechenden Container für b_2 und b_3 unterschiedliche Inhalte besitzen. Die Anfrage wäre nicht erfüllbar.

Für das Erstellen des Feldes müssen alle Attribute gezählt werden, dieser Schritt kann in $O(n)$ durchlaufen werden, wobei n die Anzahl der Variablen widerspiegelt. Da n durch die Länge der Anfrage Q beschränkt ist, kann $O(|Q|)$ als obere Grenze der Laufzeit betrachtet werden. Anschließend werden alle Konstanten und alle Attributselektionen durchlaufen. Beide Schritte sind in ebenfalls in $O(|Q|)$ abzuarbeiten. Damit ist die Gesamtlaufzeit des Algorithmus $O(|Q|)$.

4.1.2 Konjunktive Anfragen mit $\theta_{\neg\neq}$

Für die Wertebereiche *integer* und *real* existiert je ein Algorithmus zur Überprüfung, ob eine konjunktive $\theta_{\neg\neq}$ Anfrage in einer anderen derartigen Anfrage enthalten ist. Die beiden Algorithmen sind ähnlich aber nicht identisch [GSW96b]. Um Redundanzen zu vermeiden, werden beide Algorithmen zu einem Algorithmus zusammengefasst und bei der Erläuterung des Algorithmus auf eventuelle Besonderheiten abhängig vom benutzten Datentyp eingegangen. Der Algorithmus bestimmt alle Attribut- und Konstantenselektionen, die durch die Anfrage Q impliziert werden (siehe Abschnitt 3.5.2). Anschließend wird die Anfrage auf Widersprüche untersucht. Algorithmus 2 (Seite 28) zeigt das grobe Vorgehen im Pseudocode.

Bei diesem Algorithmus wird die Datenbankanfrage Q als logische Formel F_Q interpretiert (siehe Abschnitt 3.5.2). Die Zahl der Vergleichsoperatoren in F_Q wird reduziert,

Algorithmus 2 : Erfüllbarkeitsproblem bei $\theta_{\neg\neq}$-Anfragen

Eingabe : Anfrageformel F_Q
Ausgabe : Zyklenfreier Graph $G_{Q_{collapsed}}$ oder FALSE

```
1  Reduktion der Vergleichsoperatoren in F_Q
2  while ∃ Triviale Ungleichung tu in F_Q do
3      if tu ist Kontradiktion then
4          return FALSE
5      else
6          Eliminiere tu
7  end
8  Bestimmung des minimalen Wertebereichs W_i für alle x_i in Q
9  Überführung von F_Q in G_Q
10 Ermittlung von G_{Q_collapsed} durch Eliminierung der starken
   Zusammenhangskomponenten in G_Q
11 Bestimmung der wahren minimalen Wertebereiche W_i' durch Berechnung der
   implizierten Konstanteselektionen
12 if W_i' = ∅ then
13     return FALSE
14 else
15     return G_{Q_collapsed}
```

indem die Axiome $x_i = x_j \equiv (x_i \leq x_j) \wedge (x_i \geq x_j)$, $x_i = w \equiv (x_i \leq w) \wedge (x_i \geq w)$, $x_i \geq x_j \equiv x_j \leq x_i$ und $x_i > x_j \equiv x_j < x_i$ auf die Formel angewendet werden. So kann F_Q in eine Formel überführt werden, welche ausschließlich Konjunkte der Form $x_i\{<, \leq\}x_j$ oder $x_i\{<, \leq, \geq, >\}w$ enthält. Da für Formeln, die auf dem Wertebereich *integer* definiert sind, noch die Axiome $x_i < w \equiv x_i \leq (w-1)$ und $x_i > w \equiv x_i \geq (w+1)$ gelten, kann hier die Menge der in F_Q enthaltenen Ungleichungen mit Konstanten weiter auf die Form $x_i\{\leq, \geq\}w$ reduziert werden.

Im nächsten Schritt wird F_Q auf triviale Ungleichungen untersucht. Triviale Ungleichungen sind genau die Konjunkte in F_Q, welche die Form $x_i\ \theta\ x_i$ oder $w_j\ \theta\ w_k$ besitzen. Diese können entweder Tautologien oder Kontradiktionen sein. Tautologien werden eliminiert, da sie immer wahr sind und damit keinen Einfluss auf das Anfrageergebnis besitzen. Enthält F_Q eine Kontradiktion, also eine Ungleichung der Form $x_i\{<, >, \neq\}x_i$, $w\{<, >, \neq\}w$, $w_j\{<, \leq, =\}w_k$ mit $w_j > w_k$ oder $w_j\{>, \geq, =\}w_k$ mit $w_j < w_k$, so wird die Ausführung des Algorithmus abgebrochen. Die Anfrage ist nicht erfüllbar. Anschließend wird für jede Variable x_i in F_Q der minimale Wertebereich $W_i = \{w^i_{low}...w^i_{up}\}$ bestimmt. Dabei wird W_i zunächst das Intervall $]-\infty, +\infty[$ zugewiesen. Für jede Ungleichung der Form $x_i\ \theta\ w$ wird dieses Intervall nun weiter eingeschränkt. Dabei führen Ungleichungen der Form $x_i\{<, >\}w$ zu offenen Intervallgrenzen und Ungleichungen der Form $x_i\{\leq, \geq\}w$ zu geschlossenen Intervallgrenzen. Anschließend wird, wie in Abschnitt 3.5.2

beschrieben, die Formel F_Q in den Graph G_Q überführt.

Im nächsten Schritt wird der Graph G_Q in einen azyklischen Graph transformiert, indem die starken Zusammenhangskomponenten zu jeweils einem Knoten zusammengefasst werden (siehe Abschnitt 3.5.2). Tritt hierbei eine Kontradiktion auf, so wird die Ausführung des Algorithmus abgebrochen, die Anfrage ist nicht erfüllbar. Der durch Eliminierung der starken Zusammenhangskomponenten von G_Q abgeleitete, azyklische Graph heißt $G_{Q_{collapsed}}$.

Anschließend wird der *wahre* Wertebereich der Attribute W_i' bestimmt, indem alle implizierten Konstantenselektionen ermittelt werden (siehe Abschnitt 3.5.2). Für jeden Knoten x_i ist die neue untere Intervallgrenze $A^i_{low} = max(w^i_{low}, w_1, ..., w_n)$, wobei n die Anzahl der Kindknoten ist. $w_j \in \{w_1, ..., w_n\}$ wird dabei wie folgt aus dem Kindknoten j des Knotens x_i bestimmt: Ist die Domäne der in Q enthaltenen Attribute *integer*, so ist $w_j = A^j_{low}$, wenn die Kante zwischen x_i und x_j mit $\leq$ bezeichnet ist, und $w_j = A^j_{low} + 1$, wenn diese Kante mit $<$ bezeichnet ist. A^i_{low} ist dabei stets eine geschlossene Intervallgrenze. Ist die Domäne der Attribute *real*, so gilt $w_j = A^j_{low}$. Ist die Kante zwischen x_i und x_j mit $<$ bezeichnet, so ist dies eine offene, andernfalls eine geschlossene Intervallgrenze. Für die oberen Intervallgrenzen gilt $A^i_{up} = min(w^i_{up}, w_1, ..., w_n)$ mit n gleich der Anzahl der Vaterknoten. Im Wertebereich *integer* gilt für $w_j \in \{w_1, ..., w_n\}$ $w_j = A^j_{up}$, wenn die Kante zwischen x_i und x_j mit $\leq$ bezeichnet ist, und $w_j = A^j_{up} - 1$, wenn die Kante mit $<$ bezeichnet ist. In beiden Fällen ist die Intervallgrenze geschlossen. Ist die Domäne der Attribute *real*, so ist der Wert von $w_j = A^j_{up}$. Die entsprechende Intervallgrenze ist geschlossen, wenn die Kante zwischen x_i und x_j mit $\leq$ bezeichnet ist und sonst offen. Führen die so bestimmten Intervallgrenzen zu leeren Intervallen, ist also $A^i_{low} > A^i_{up}$ oder $A^i_{low} = A^i_{up}$ mit jeweils offenen Intervallgrenzen, so ist die Anfrage nicht erfüllbar. Sonst ist die Anfrage erfüllbar und es wird der Graph $G_{Q_{collapsed}}$ mit den neuen Intervallen W_i' zurückgegeben. Der Graph G_Q und der Graph $G_{Q_{collapsed}}$ sind äquivalent. Ebenso sind die Formeln F_Q und $F_{Q_{collapsed}}$ äquivalent [GSW96b].

Sei die Anfrage $Q = \{a_1, a_2 | \exists b_1 \exists b_2 (a_1 \geq a_2 \wedge a_1 < b_2 \wedge a_2 \leq b_1 \wedge a_2 < 5 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 > 2 \wedge b_2 \leq 4)\}$ aus Abschnitt 3.5.2 gegeben. Da diese Anfrage weder Tautologien noch Kontradiktionen enthält, liefert der Algorithmus zum Test auf Erfüllbarkeit mit dem Wertebereich *real* den in Abbildung 3.7 (Seite 21) dargestellten Graph. Die Anfrage Q ist folglich im Wertebereich *real* erfüllbar.

Für den Wertebereich *integer* ergeben sich einige Änderungen. Q wird zunächst in die Anfrageformel $F_Q = (a_1 \geq a_2 \wedge a_1 < b_2 \wedge a_2 \leq b_1 \wedge a_2 < 5 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 > 2 \wedge b_2 \leq 4)$ übertragen. Die Reduktion der Vergleichsoperatoren führt dazu, dass die Ungleichung $b_2 < 5$ durch die Ungleichung $b_2 \leq 4$ und $b_1 > 2$ durch die Ungleichung $b_1 \geq 3$ ersetzt werden kann. Somit kann F_Q in den Graphen in Abbildung 4.2 (Seite 30) überführt werden. Im Gegensatz zu dem Graph für den Wertebereich *real* enthält dieser Graph nur Wertebereiche mit geschlossenen Grenzen.

Anschließend werden die starken Zusammenhangskomponenten des Graphen G_Q eliminiert. Das Vorgehen ist dabei gleich dem Vorgehen für den Wertebereich *real*. Schließlich werden die Wertebereiche angepasst. Abbildung 4.3 (Seite 30) zeigt den resultie-

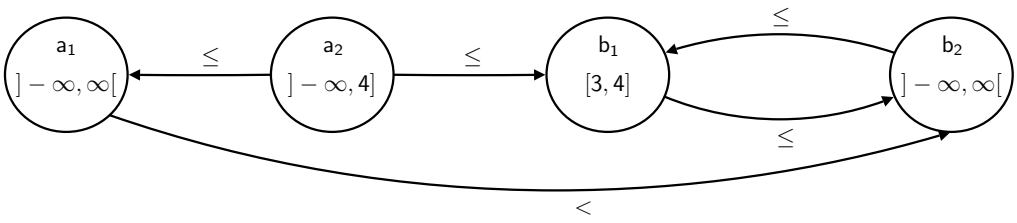

Abbildung 4.2: Anfragegraph G_Q für den Wertebereich *integer*

renden Graphen. Der durch Eliminierung der starken Zusammenhangskomponenten entstandene Knoten $b_1\_b_2$ besitzt die Wertebereichsgrenzen des Ausgangsknotens b_1, da hier sowohl die untere Grenze größer als auch die obere Grenze kleiner ist als bei b_2. Die obere Schranke des Wertebereichs von a_2 ist 4, da die Kante zwischen a_2 und $b_1\_b_2$ mit $\leq$ beschriftet ist. Die obere Grenze des Wertebereichs von a_1 hingegen ist 3, da hier die entsprechende Kante mit $<$ bezeichnet ist. Diese Anfrage ist auch für den Wertebereich *integer* erfüllbar, da $G_{Q_{collapsed}}$ keine Kontradiktionen impliziert.

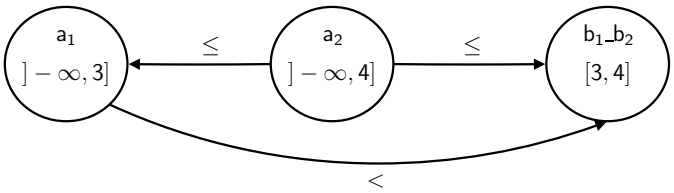

Abbildung 4.3: Anfragegraph $G_{Q_{collapsed}}$ für den Wertebereich *integer*

Für die Eliminierung der trivialen Ungleichungen der Formel F_Q genügt es, nacheinander alle Selektionsbedingungen der Anfrage zu durchlaufen. Somit ist dieser Schritt in einer Laufzeit von $O(|Q|)$ lösbar, wobei $|Q|$ die Länge der Anfrage wiedergibt. Die Bestimmung der minimalen Wertebereiche der x_i ist auch in $O(|Q|)$ möglich, da hierbei nacheinander alle Konstantenselektionen bearbeitet werden müssen. Der Graph $G_{Q_{collapsed}}$ kann ebenfalls in $O(|Q|)$ bestimmt werden, da alle starken Zusammenhangskomponenten in $O(|V| + |E|)$ ermittelt werden können (siehe Kapitel 3.5.2) und $|V| + |E|$ durch $|Q|$ begrenzt wird. Auch der Algorithmus zur Bestimmung der *wahren* Wertebereiche W_i' kann in $O(|Q|)$ gelöst werden, da alle implizierten Konstantenselektionen in $O(|V|+|E|)$ ermittelbar sind (siehe Abschnitt 3.5.2) und auch hier $|V| + |E|$ durch $|Q|$ begrenzt ist. Somit besitzt der Algorithmus zur Überprüfung einer konjunktiven $\theta_{\neg\neq}$-Anfrage Q auf Erfüllbarkeit eine Laufzeit von $O(|Q|)$ [GSW96b].

4.1.3 Konjunktive Anfragen mit θ_{ALL}

Für konjunktive θ_{ALL}-Anfragen wird hier zunächst ein Algorithmus zum Testen der Erfüllbarkeit einer Anfrage mit Attributen des Wertebereichs *real* vorgestellt. Für den Wertebereich *integer* ist das Erfüllbarkeitsproblem NP-hart [RH80]. Der hier vorgestellte Algorithmus für den Wertebereich *real* lässt sich aber auf einen Algorithmus für den

Wertebereich *integer* erweitern. Am Ende des Kapitels wird diese Erweiterung vorgenommen.

Das Vorgehen beim Test, ob eine konjunktive θ_{ALL}-Anfrage, die ausschließlich Attribute des Wertebereichs *real* enthält, erfüllbar ist, gestaltet sich ähnlich dem in Abschnitt 4.1.2 vorgestellten Algorithmus für die Wertebereiche *real* und *integer* [GSW96b]. Das Vorgehen im Pseudocode ist identisch dem in Algorithmus 2 (Seite 28) beschriebenen Vorgehen.

Nachdem die Anfrage Q in eine logische Formel F_Q übertragen wurde, folgt die Reduzierung der in F_Q enthaltenen Vergleichsoperatoren. Dabei werden neben den Axiomen $x_i = x_j \equiv (x_i \leq x_j) \wedge (x_i \geq x_j)$, $x_i = w \equiv (x_i \leq w) \wedge (x_i \geq w)$, $x_i \geq x_j \equiv x_j \leq x_i$ und $x_i > x_j \equiv x_j < x_i$, die auch beim entsprechenden Algorithmus für konjunktive $\theta_{\neg\neq}$-Anfragen Anwendung finden, das Axiom $x_i < x_j \equiv (x_i \neq x_j) \wedge (x_i \leq x_j)$ genutzt. Wird der Wertebereich *real* angenommen, so werden zusätzlich die Axiome $x_i < w \equiv (x_i \neq w) \wedge (x_i \leq w)$ und $x_i > w \equiv (x_i \neq w) \wedge (x_i \geq w)$ genutzt. Für den Wertebereich *integer* ist die Anwendung der letzten beiden Axiome nicht notwendig. Hier finden die Axiome $x_i < w \equiv (x_i \leq w - 1)$ und $x_i > w \equiv (x_i \geq w + 1)$ wie schon beim Algorithmus zur Überprüfung konjunktiver $\theta_{\neg\neq}$-Anfragen Anwendung. So kann F_Q in eine Form überführt werden, die ausschließlich Konjunkte der Form $x_i\{\leq, \neq\}x_j$ und $x_i\{\leq, \geq, \neq\}w$ enthält. In den nächsten Schritten werden triviale Ungleichungen eliminiert und für jedes x_i der minimale Wertebereich bestimmt.

Anschließend wird F_Q in G_Q überführt. Das Vorgehen hierbei ist identisch dem in Abschnitt 4.1.2 beschriebenen Vorgehen zur Eliminierung trivialer Ungleichungen. Bei der Eliminierung starker Zusammenhangskomponenten muss das Vorgehen leicht angepasst werden: Da eine konjunktive θ_{ALL}-Anfrage auch Ungleichungen der Form $x_i \neq x_j$ enthalten kann, muss vor dem Eliminieren überprüft werden, ob eine dieser Ungleichungen die Gleichheit zweier Elemente der Zusammenhangskomponente verbietet. In diesem Fall ist die Anfrage nicht erfüllbar. Die Ausführung des Algorithmus kann abgebrochen werden. Der aus G_Q nach Eliminierung aller starken Zusammenhangskomponenten entstandene Graph heißt $G_{Q_{collapsed}}$. Anschließend wird, wie in Abschnitt 4.1.2 beschrieben, für alle x_i der *wahre* Wertebereich W_i' bestimmt. Ist der *wahre* Wertebereich W_i' für ein $x_i \in F_Q$ leer, gilt also $A^i_{low} > A^i_{up}$, oder $A^i_{low} = A^i_{up}$ und $x_i \neq A^i_{low} \notin Q$, so ist F_Q nicht erfüllbar. Andernfalls wird $G_{Q_{collapsed}}$ zurückgegeben, Anfrage Q ist erfüllbar.

Dieser Algorithmus ist weitestgehend identisch dem Algorithmus aus Abschnitt 4.1.2, daher wird hier auf ein ausführliches Beispiel verzichtet. Ebenso ist die Laufzeit des Algorithmus mit $O(|Q|)$ gleich der Laufzeit des Algorithmus aus Abschnitt 4.1.2 [GSW96b].

Eine analoge Erweiterung des Algorithmus auf konjunktive θ_{ALL}-Anfragen mit Wertebereich *integer* ist nicht möglich. Um dies zu veranschaulichen, sei der Graph aus Abbildung 4.4 (Seite 32) gegeben. Zusätzlich zu den vom Graph implizierten Selektionsbedingungen sollen die Ungleichungen $x_1 \neq x_2$, $x_1 \neq x_3$ und $x_2 \neq x_3$ gelten. Im Wertebereich *real* ist eine durch diesen Graphen repräsentierte Anfrage erfüllbar, da alle Attribute eine unendliche Zahl von Werten zwischen 1 und 2 annehmen können. Für den Wertebereich *integer* ist die Anfrage dagegen nicht erfüllbar: Wenn x_1 den Wert 1 an-

nimmt, müssen x_2 und x_3 gleich 2 sein, dies widerspricht aber der Ungleichung $x_2 \neq x_3$. Gleiches gilt für den Fall, wenn x_1 den Wert 2 annimmt. Allerdings kann dieses Ergebnis nicht, wie bei den anderen Erfüllbarkeitsalgorithmen, ermittelt werden, indem nur der Knoten x_1 und die Beziehung zu seinen Nachbarn untersucht werden.

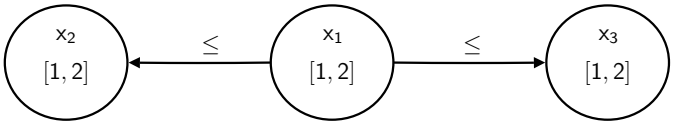

Abbildung 4.4: Problem bei konjunktiven θ_{ALL}-Anfragen im Wertebereich *integer*

Dennoch entspricht jede Ungleichung $x_i \neq x_j$ der Formel $x_i < x_j \vee x_i > x_j$. Auf diese Weise kann eine Anfrageformel F_Q in zwei neue Anfrageformeln F_{Q_1} und F_{Q_2} unterteilt werden. Dabei entspricht F_{Q_1} der Ausgangsformel F_Q, nur $x_i \neq x_j$ wird durch $x_i < x_j$ ersetzt und F_{Q_2} ist gleich der Ausgangsformel mit der Ungleichung $x_i > x_j$ anstelle von $x_i \neq x_j$. Die Ausgangsformel ist erfüllbar, wenn F_{Q_1} oder F_{Q_2} erfüllbar ist. Enthalten diese beiden Formeln weitere Ungleichungen der Form $x_i \neq x_j$, so werden sie rekursiv in weitere Anfrageformeln unterteilt. Analog kann dann für Konstantenselektionen mit Operator $\neq$ verfahren werden. So werden maximal 2^n Formeln ermittelt, wobei n für die Zahl der beteiligten Ungleichungen steht. Diese Formeln werden dann mit Hilfe des in Abschnitt 4.1.2 vorgestellten Algorithmus zur Überprüfung auf Erfüllbarkeit konjunktiver $\theta_{\neg\neq}$- Anfragen untersucht. Sobald eine Anfrage gefunden wurde, die erfüllbar ist, kann gefolgert werden, dass die gesamte Anfrage erfüllbar ist. So müssen maximal exponentiell viele Anfragen untersucht werden.

Das Problem ist wie bereits eingangs erwähnt NP-hart [RH80].

4.1.4 Zusammenfassung

In diesem Abschnitt sollen die Ergebnisse zum Erfüllbarkeitsproblem zusammengefasst werden. Zunächst wurde auf den trivialen Fall von Anfragen bestehend aus Projektionen und Kreuzprodukten eingegangen. In beiden Fällen kann in $O(1)$ überprüft werden, ob eine Anfrage mindestens ein Projektionsattribut bzw. mindestens eine Relation im Kreuzprodukt enthält. Weiterhin wurde ein Algorithmus beschrieben, mit dessen Hilfe konjunktive $\theta_{=}$-Anfragen in $O(|Q|)$ auf Erfüllbarkeit getestet werden können. Werden für konjunktive Anfragen weitere Operatoren zugelassen, so sind die Algorithmen je nach zugelassenem Wertebereich unterschiedlich. Dennoch lassen sich konjunktive $\theta_{\neg\neq}$-Anfragen unabhängig vom benutzten Wertebereich in $O(|Q|)$ auf Erfüllbarkeit prüfen. Auch konjunktive θ_{ALL}-Anfragen auf dem Wertebereich *real* lassen sich mit diesem Aufwand prüfen [GSW96b]. Lediglich das Testen von konjunktiven θ_{ALL}-Anfragen mit dem Wertebereich *integer* ist problematisch. Dieses Problem ist NP-hart [RH80]. Tabelle 4.1 (Seite 33) fasst diese Ergebnisse zusammen.

Op.	Einschränkungen	Ergebnis	Quelle
$\theta_{=}$	keine Einschränkung	$O(\|Q\|)$	-
$\theta_{\neg\neq}$	Wertebereich *integer*	$O(\|Q\|)$	[GSW96b]
	Wertebereich *real*	$O(\|Q\|)$	[GSW96b]
θ_{ALL}	Wertebereich *integer*	NP-hart	[RH80]
	Wertebereich *real*	$O(\|Q\|)$	[GSW96b]

Tabelle 4.1: Komplexität des Erfüllbarkeitsproblems

4.2 Problem des Anfrageenthaltenseins

Algorithmen zum Anfrageenthaltensein (englisch: query containment) beschäftigen sich mit dem Problem, ob das Ergebnis einer Anfrage im Ergebnis einer anderen Anfrage enthalten ist. Eine Datenbankanfrage Q' ist genau dann in einer Anfrage Q enthalten, geschrieben als $Q' \sqsubseteq Q$, wenn für jede Instanz $d(D)$ eines Datenbankschemas D das Ergebnis der Anfrage $Q'(d)$ auf der Datenbankinstanz $d(D)$ im Ergebnis der Anfrage $Q(d)$ enthalten ist, also $Q'(d) \subseteq Q(d)$ gilt. Zwei Anfragen sind äquivalent $Q' \equiv Q$, wenn sowohl $Q \sqsubseteq Q$ als auch $Q \sqsubseteq Q'$ gilt [CR97].

Das Problem, ob zwei Anfragen äquivalent sind (englisch: equivalence problem), ist im allgemeinen Fall nicht lösbar [Sol79]. Da zwei Relationen genau dann äquivalent sind, wenn sie in einander enthalten sind [CM77], impliziert dies, dass auch das Enthaltenseinproblem im allgemeinen Fall nicht lösbar ist. Daher gibt es verschiedene Ansätze, die Anfragemächtigkeit einzuschränken, um berechenbare Teilprobleme für das Enthaltenseinproblem zu bestimmen.

Bei den hier beschriebenen Algorithmen wird davon ausgegangen, dass die beteiligten Anfragen erfüllbar sind. Ist eine Anfrage Q' nicht erfüllbar, so kann sie niemals Tupel enthalten. Damit ist sie in jeder beliebigen Anfrage Q enthalten. Umgekehrt gilt: Wenn eine Anfrage Q nicht erfüllbar aber die Anfrage Q' erfüllbar ist, so kann Q' nicht in Q enthalten sein. Diese Sonderfälle können ausschließlich mit den in Abschnitt 4.1 beschriebenen Algorithmen zur Erfüllbarkeit von Anfragen gelöst werden und erfordern daher keine gesonderte Betrachtung.

4.2.1 Projektion

Zwei Anfragen Q' und Q, die nur aus Projektionen bestehen, sind genau dann ineinander enthalten, wenn die Menge Projektionsattribute P' der Anfrage Q' in der Menge der Projektionsattribute P von Q enthalten sind. Die Anfrage Q' ist demnach in Q enthalten, wenn $P' \subseteq P$ gilt. Um dies nachzuprüfen, muss folglich für jedes Attribut $x_i \in P'$ überprüft werden, ob $x_i \in P$ gilt.

Da in dieser Arbeit davon ausgegangen wird, dass ein Attribut in $O(1)$ in einer Anfrage gefunden werden kann, ist der Aufwand für einen entsprechenden Algorithmus

$O(|Q'|)$.

4.2.2 Kreuzprodukt

Das Problem des Anfrageenthaltenseins zweier Anfragen, $Q' \sqsubseteq Q$, bestehend aus Kreuzprodukten, ist im allgemeinen Fall nicht lösbar. Folgendes Beispiel soll diesen Fall illustrieren: Seien die Kreuzprodukte $Q' = r(R_1) \times r(R_2)$ und $Q = r(R_1) \times r(R_2) \times r(R_3)$ auf dem Datenbankschema D gegeben. Unabhängig vom Kreuzprodukt $r(R_1) \times r(R_2)$ gilt $Q(d) = \{\}$, wenn die Relation $r(R_3)$ in der Datenbankinstanz $d(D)$ leer ist. Ist $r(R_1) \times r(R_2)$ selbst nicht leer, so gilt $Q' \sqsubseteq Q$ nicht. Da für jedes Datenbankschema D eine Instanz $d(D)$ gefunden werden kann, für die $r(R_3) = \emptyset$ gilt, gilt im allgemeinen Fall $Q' \sqsubseteq Q$ nicht. Sobald $r(R_3)$ mindestens ein Tupel enthält, gilt $Q' \sqsubseteq Q$, da Q dann aus der Vereinigung der Tupel in $r(R_3)$ mit den Tupeln der Relation $r(R_1) \times r(R_2)$ besteht (siehe Abschnitt 3.2.3). Daher werden im Folgenden als Kreuzprodukte nur Anfragen zugelassen, deren Ergebnis mindestens ein Tupel besitzt. In diesem Fall ist Q' in einer Anfrage Q genau dann enthalten, wenn die Menge der Relationen $\mathbb{R}'$ der Anfrage Q' in der Menge der Relationen $\mathbb{R}$ von Q enthalten ist.

In diesem Fall können zwei Anfragen Q' und Q mit der Komplexität $O(|Q'|)$ auf Enthaltensein geprüft werden, wenn davon ausgegangen wird, dass in $O(1)$ eine Relation $r(R_i)$ in Q gefunden werden kann.

Die Algorithmen auf konjunktiven Anfragen aus der Literatur, die im Folgenden vorgestellt werden, betrachten ausschließlich Anfragen, die aus Kreuzprodukten $r(R_i) \times r(R_j)$ bestehen, die über Attributvergleiche der Form $x_k\ \theta\ x_l$ mit $x_k \in r(R_i)$ und $x_l \in r(R_j)$ miteinander verbunden sind. In diesem Fall tritt die oben beschriebene Besonderheit, wenn eine beteiligte Relation leer ist, nicht auf. Die Algorithmen können aber alle so erweitert werden, dass sie auch Kreuzprodukte ohne entsprechende Attributvergleiche zulassen, wenn die Anfrage Q, die in der Anfrage Q' enthalten sein soll, nicht leer ist. Entsprechende Erweiterungen werden am Ende jeden Abschnitts vorgenommen.

4.2.3 Konjunktive Anfragen mit $\theta_=$

In diesem Abschnitt soll zunächst ein Algorithmus beschrieben werden, mit dem azyklische, konjunktive $\theta_=$-Anfragen auf Enthaltensein überprüft werden können. Anschließend werden die Probleme bei zyklischen $\theta_=$-Anfragen dargestellt und eine Lösungsmöglichkeit angeboten.

Um zu testen, ob die konjunktive $\theta_=$-Anfrage Q' in der azyklischen, konjunktiven $\theta_=$-Anfrage Q enthalten ist, wird die Anfrage Q durch einen Hypergraphen repräsentiert. Die Eigenschaften des Hypergraphen können dann genutzt werden, um das Enthaltenseinproblem zu lösen [CR00]. Hierfür wird der Verbundbaum der konjunktiven Anfrage bestimmt (siehe Abschnitt 3.5.1). Dabei entspricht jeder Knoten des Verbundbaums einem Relationenprädikat r_i der Anfrage Q. Die in r_i enthaltenen Attribute seien mit $\overline{A_i}$ bezeichnet. Anschließend werden alle partiellen Abbildungsfunktionen ϕ bestimmt,

Algorithmus 3 : Enthaltenseinproblem bei $\theta_=$-Anfragen

Eingabe : Anfragen Q' und Q
Ausgabe : TRUE oder FALSE

```
1  Bestimmung des Verbundbaums von Q
2  Bestimmen aller partiellen Abbildungen φ, welche die Relationenprädikate von Q
   auf Q' abbilden
3  foreach partielle Abbildung φ, die r_i auf ein Relationenprädikat von Q' abbildet
   do
4      Map_i = Map_i ∪ φ(Ā_i)
5  end
6  foreach Knoten i in postorder Reihenfolge do
7      foreach Knoten j mit j ist Kindknoten von i do
8          Map_i = Map_i ⋉ Map_j
9      end
10 end
11 if Map_i des Wurzelknotens des Verbundbaums von Q ist leer then
12     return FALSE
13 else
14     return TRUE
```

die ein Relationenprädikat der Anfrage Q auf ein Relationenprädikat r_i der Anfrage Q' abbilden. Jedem Knoten des Verbundbaums wird dann eine Relation Map_i mit den Attributen $\overline{A_i}$ angelegt, die genau die Relationenprädikate der Anfrage Q' enthält, auf die ϕ das Relationenprädikat r_i des Verbundbaums abbildet. Danach werden alle Knoten i in postorder Reihenfolge, also einer Reihenfolge, in der ein Vaterknoten stets nach seinen Kindknoten auftritt, durchlaufen. Für jeden Kindknoten j des Knotens i wird nun der natürliche Semiverbund der Relation Map_j mit der Relation Map_i ausgeführt. Ist nach dieser Operation die Relation Map_i des Wurzelknotens nicht leer, so ist die Anfrage Q' in der Anfrage Q enthalten [CR00]. Algorithmus 3 (Seite 3) zeigt das Vorgehen im Pseudocode.

Sei die konjunktive $\theta_=$-Anfrage $Q = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7 |\ Mitarbeiter(a_1, a_2, a_3) \wedge Mitarbeiter\_Abteilung(a_1, a_4) \wedge Abteilung(a_4, a_5, a_6) \wedge Abteilung(a_6, 'Management', a_7)\}$ aus Abschnitt 3.5.1 und die konjunktive Anfrage $Q' = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 |\ Mitarbeiter(a_1, 'Meyer', a_3) \wedge Mitarbeiter\_Abteilung(a_1, a_4) \wedge Abteilung(a_4, a_5, a_6) \wedge Abteilung(a_6, 'Management', a_7) \wedge Abteilung(a_7, a_8, a_9)\}$ gegeben. Q bestimmt alle Mitarbeiter, die in einer Abteilung arbeiten, die der Abteilung '*Management*' untergeordnet ist. Q' hingegen bestimmt alle Mitarbeiter mit dem Namen '*Meyer*', die in einer Abteilung, die '*Management*' untergeordnet ist, arbeiten. Die Abteilung '*Management*' ist wiederum einer beliebigen anderen Abteilung untergeordnet. Um zu überprüfen, ob die Anfrage Q' in der Anfrage Q enthalten ist, wird zunächst der Verbundbaum der Anfrage Q ermittelt (siehe Abbildung 3.2, Seite 16). Im nächsten Schritt werden jedem

Knoten des Verbundbaums die Relationen Map_i zugeordnet. Die Kästen mit durchgezogener Linie in Abbildung 4.5 (Seite 36) stellen diese Relationen dar. Im nächsten Schritt, werden die natürlichen Semiverbunde berechnet. Das Ergebnis dieses Schritts ist in den Kästen mit gestrichelter Linie in Abbildung 4.5 (Seite 36) zu sehen. Die Relation des Wurzelknotens im Beispiel ist nicht leer. Daher gilt $Q' \sqsubseteq Q$.

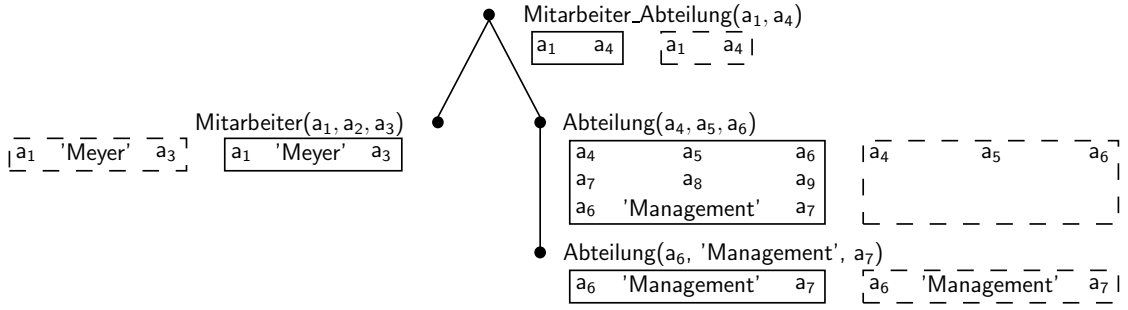

Abbildung 4.5: Verbundbaum des Hypergraphen aus Abbildung 3.1

Dieser Algorithmus entscheidet in $O(|Q| \cdot |Q'| \cdot log(|Q'|))$, ob die Anfrage Q' in Q enthalten ist [CR00]. Wobei $|Q|$ und $|Q'|$ jeweils die Länge der entsprechenden Anfrage bezeichnen.

Der beschriebene Algorithmus erlaubt nur die Betrachtung von Anfragen, deren Repräsentation als Hypergraph verbunden ist [CR00]. Anfragen, die aus dem Kreuzprodukt mehrerer Relationen bestehen, die nicht durch Attributselektionen miteinander in Verbindung gesetzt sind, werden durch mehrere Hypergraphen repräsentiert. Um solche Anfragen bearbeiten zu können, wird der Algorithmus erweitert: Anstelle eines Verbundbaums für die Anfrage Q wird nun ein Verbundwald (englisch: join forest) in Schritt 1 des Algorithmus 3 (Seite 35) aufgebaut. Dabei besteht der Verbundwald aus je einem Verbundbaum pro Hypergraph. Die partiellen Abbildungen ϕ und die Relationen Map_i werden wie oben beschrieben bestimmt. Dabei wird jeder Verbundbaum des Verbundwaldes gekennzeichnet, der mindestens einen Knoten mit einer nicht leeren Relation Map_i besitzt. Nur für die so gekennzeichneten Verbundbäume wird der natürliche Semiverbund berechnet. Die Anfrage Q' ist dann in der Anfrage Q enthalten, wenn all diese Verbundbäume eine nicht leere Relation Map_i in ihrem Wurzelknoten besitzen.

Die Laufzeit des erweiterten Algorithmus bleibt gleich der Laufzeit $O(|Q| \cdot |Q'| \cdot log(|Q'|))$ des oben beschriebenen Algorithmus. Ausgehend vom Verbundwald kann der Algorithmus jeweils auf einem Verbundbaum ausgeführt werden. Hierfür wird die Anfrage Q in n Teile zerlegt, die je durch einen Verbundbaum repräsentiert werden können. Jeder Verbundbaum beschreibt dann einen Teil der gesamten Anfrage Q. Die Länge jeder Teilanfrage sei mit $a_i \cdot |Q|$ gegeben. Da die Teilanfragen disjunkt sind, ist die Länge der gesamten Anfrage $|Q| = \sum_{i=1}^{n} a_i \cdot |Q|$ und damit $\sum_{i=1}^{n} a_i = 1$. Jeder einzelne Verbundbaum des Verbundwalds beschreibt nur den Teil a_i der Anfrage Q. Zudem ist die Laufzeit des Algorithmus auf einem einzelnen Verbundbaum $O(a_i \cdot |Q| \cdot |Q'| \cdot log(|Q'|))$. Zur Lösung des Gesamtproblems muss der Algorithmus auf allen n Verbundbäumen ausgeführt wer-

den. Für die Laufzeit gilt damit $O(\sum_{i=1}^{n} a_i \cdot |Q| \cdot |Q'| \cdot log(|Q'|))$. Da für $\sum_{i=1}^{n} a_i = 1$ gilt, ist die Laufzeit des gesamten Algorithmus $O(|Q| \cdot |Q'| \cdot log(|Q'|))$.

Der oben beschriebene Algorithmus kann so erweitert werden, dass er auch auf azyklischen Hypergraphen ausführbar ist. Hierbei wird anstelle eines Verbundbaums, der nur für azyklische Anfragen bestimmt werden kann, der Zerlegungsbaum der Anfrage Q genutzt. Die Schwierigkeit dieses Algorithmus liegt in der Bestimmung des Zerlegungsbaums [CR00]. Für eine gegebene Anfragebreite k kann zwar in linearer Zeit getestet werden, ob ein gegebener Hypergraph H_Q eine Anfragebreite von maximal k besitzt und ein passender Zerlegungsbaum mit Anfragebreite k ausgegeben werden [Bod93]. Allerdings wurde noch kein in polynomialer Zeit berechenbarer Algorithmus zum Bestimmen der Anfragebreite k gefunden [CR00]. Da das Anfrageenthaltenseinproblem für konjunktive $\theta_{=}$-Anfragen auch mit den Algorithmen des entsprechenden Problems für konjunktive $\theta_{\neg\neq}$-Anfragen gelöst werden kann und da in Abschnitt 4.2.4 gezeigt wird, dass dieses Problem mit geringerem Aufwand lösbar ist, werden zyklische, konjunktive $\theta_{=}$-Anfragen mit dem Algorithmus für konjunktive $\theta_{\neg\neq}$-Anfragen bearbeitet.

4.2.4 Konjunktive Anfragen mit $\theta_{\neg\neq}$

Der Test auf Implikation, also ob die Formel $F_{Q'}$ die Formel F_Q impliziert, ist äquivalent zu dem Problem des Anfrageenthaltenseins der Anfrage Q' in der Anfrage Q. Der Algorithmus für das Implikationsproblem ist für Attribute mit den Wertebereichen *integer* und *real* wie schon beim Erfüllbarkeitsproblem ähnlich, aber nicht identisch [GSW96b]. Daher wird hier für den Test auf Implikation nur ein Algorithmus betrachtet und an den entsprechenden Stellen auf die Unterschiede eingegangen. Der Algorithmus bestimmt, wie in Abschnitt 3.5.2 beschrieben, alle durch die Anfrage Q' implizierten Konstanten- und Attributselektionen. Anschließend werden die Variablen der Anfrage Q' auf die Variablen der Anfrageformel F_Q abgebildet. Schließlich wird für jedes Konjunkt der Anfrage F_Q überprüft, ob dieses auf Q' abgebildet werden kann. Ist dies der Fall, ist die Anfrage Q' in der Anfrage Q enthalten. Algorithmus 4 (Seite 38) zeigt das Vorgehen im Pseudocode.

In einem ersten Schritt werden die Anfragen Q und Q' in die Anfrageformeln F_Q und $F_{Q'}$ übertragen. Anschließend wird mit Hilfe des in Abschnitt 4.1.2 beschriebenen Algorithmus die Anfrageformel $F_{Q'}$ auf Erfüllbarkeit geprüft. Wie bereits eingangs erwähnt (siehe Abschnitt 4.2), gilt $Q' \sqsubseteq Q$ immer, wenn Q' nicht erfüllbar ist. Sonst liefert der Erfüllbarkeitsalgorithmus den Graphen $G_{Q'_{collapsed}}$. Anschließend wird, wie in Abschnitt 3.5.2 beschrieben, die transitive Hülle von $G_{Q'_{collapsed}}$ bestimmt. Aus der so bestimmten transitiven Hülle können nun alle von Q' implizierten Konstanten- und Attributselektionen abgelesen werden.

Im nächsten Schritt wird die Anfrageformel F_Q so bearbeitet, dass die in ihr enthaltenen Konjunkte mit den Konjunkten der Anfrage Q' verglichen werden können. Hierfür werden die Variablen in F_Q durch die entsprechenden repräsentativen Variablen in $F_{Q'}$ ersetzt. Unter repräsentativen Variablen werden dabei die Variablen verstanden, die beim Zusammenfassen von Knoten des Baums bei der Ausführung des Erfüllbarkeit-

Algorithmus 4 : Implikationsproblem $Q' \rightarrow Q$ für $\theta_{\neg\neq}$-Anfragen

Eingabe : Anfrageformeln F_Q und $F_{Q'}$
Ausgabe : TRUE oder FALSE

```
1 Bestimmung von G_{Q'_collapsed} durch Prüfen von F_{Q'} auf Erfüllbarkeit
2 Bestimmung der transitiven Hülle von G_{Q'_collapsed}
3 Ersetzen der Variablen in F_Q durch die repräsentativen Variablen in F_{Q'}
4 Bestimmung von F_{Q_reduced}
5 foreach Konjunkt k ∈ F_{Q_reduced} do
6     if k besitzt keine Entsprechung in der transitiven Hülle von G_{Q'_collapsed} then
7         return FALSE
8 end
9 return TRUE
```

salgorithmus entstanden sind. Anschließend werden auch in F_Q, wie in Abschnitt 4.1.2 beschrieben, Tautologien entfernt und die Anzahl der Vergleichsoperatoren reduziert. Das Ergebnis dieser Operation sei mit $F_{Q_{reduced}}$ bezeichnet.

Um das Implikationsproblem $Q' \rightarrow Q$ zu lösen, wird jedes in $F_{Q_{reduced}}$ enthaltene Konjunkt k mit der transitiven Hülle von $G_{Q'_{collapsed}}$ verglichen. Kann jedes Konjunkt der Anfrage Q auf die transitive Hülle von $G_{Q'_{collapsed}}$ abgebildet werden, so ist die Anfrage Q' in der Anfrage Q enthalten. Tabelle 4.2 (Seite 39) fasst die Überprüfungen, die hierbei durchzuführen sind, zusammen. Diese Tabelle kann dabei wie folgt gelesen werden: Für das zu untersuchende Konjunkt $k \in F_{Q_{reduced}}$ wird zunächst der Typ ermittelt. Hierbei wird zwischen den in Tabelle 4.2 (Seite 39) Spalte *Konjunkt* enthaltenen Typen unterschieden. Wird für den Typ des Konjunkts k in der Spalte *Wertebereich* ein Wertebereich angegeben, so gilt die Eigenschaft der dritten Spalte der Tabelle nur für Graphen dieses Wertebereichs. Die Variablen der Tabelle haben dabei die Bedeutung wie in Abschnitt 3.5.2: x_i, x_j sind Variablen und w ist eine Konstante einer Selektionsbedingung. A^i_{up} und A^i_{low} sind die unteren bzw. oberen Wertebereichsgrenzen des Wertebereichs des Knotens in $G_{Q'_{collapsed}}$, der x_i repräsentiert.

Hat das zu untersuchende Konjunkt k beispielsweise die Form $x_i \leq x_j$, so muss die transitive Hülle von $G_{Q'_{collapsed}}$ entweder eine Kante zwischen den Knoten x_i und x_j mit beliebiger Kantenbeschriftung besitzen oder es muss $A^i_{up} \leq A^j_{low}$ gelten. Besitzt $G_{Q'_{collapsed}}$ weder eine entsprechende Kante noch gilt die obige Ungleichung, so impliziert die Anfrage Q' nicht Q. Damit ist Q' nicht in Q enthalten. Analog wird für andere Konjunkte vorgegangen.

Sei als Anfrage Q' die Anfrage $\{a_1, a_2 | \exists b_1 \exists b_2 (a_1 \leq a_2 \wedge a_1 < b_2 \wedge a_2 \leq b_1 \wedge a_2 < 5 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 > 2 \wedge b_2 \leq 4)\}$ aus Abschnitt 3.5.2 gegeben. Weiterhin sei $Q = \{a_1, a_2 | \exists b_1 \exists b_2 (a_1 \leq 3 \wedge a_1 \leq a_2 \wedge a_2 \leq b_2 \wedge b_1 > 0)\}$ gegeben. Nun soll geprüft werden, ob $Q' \sqsubseteq Q$ gilt.

Konjunkt	**Wertebereich**	**Eigenschaft transitive Hülle von $G_{Q'_{collapsed}}$**
$x_i \leq x_j$	*integer* und *real*	$\exists$ eine Kante (x_i, x_j, θ) in $G_{Q'_{collapsed}}$ oder $A^i_{up} \leq A^j_{low}$
$x_i < x_j$	*integer*	$\exists$ eine Kante $(x_i, x_j, <)$ in$G_{Q'_{collapsed}}$ oder $A^i_{up} < A^j_{low}$
$x_i < x_j$	*real*	$\exists(x_i, x_j, <)$ oder $A^i_{up} < A^j_{low}$ oder $A^i_{up} = A^j_{low}$ und eine Grenze ist offen
$x_i \leq w$	*integer* und *real*	$w \geq A^i_{up}$
$x_i \geq w$	*integer* und *real*	$w \leq A^i_{low}$
$x_i < w$	*real*	$w > A^x_{up}$ oder $w = A^x_{up}$ und A^x_{up} ist eine offene Grenze
$x_i > w$	*real*	$w < A^x_{up}$ oder $w = A^x_{low}$ und A^x_{low} ist eine offene Grenze

Tabelle 4.2: Entsprechungen von k und der transitiven Hülle von $G_{Q'_{collapsed}}$ für konjunktive $\theta_{\neq\neg}$-Anfragen

Für den Wertebereich *real* liefert die Ausführung des Erfüllbarkeitsalgorithmus einen Graphen $G_{Q'_{collapsed}}$, der dem Graphen in Abbildung 3.7 (Seite 21) entspricht. Die Berechnung der transitiven Hülle liefert keine zusätzlichen Kanten, so dass dieser Graph gleich der transitiven Hülle von $G_{Q'_{collapsed}}$ ist. Nun wird für jedes Konjunkt von Q geprüft, ob es auf die transitive Hülle von $G_{Q'_{collapsed}}$ abgebildet werden kann. Bereits $a_1 \leq 3$ lässt sich nicht auf $G_{Q'_{collapsed}}$ abbilden, da die obere Grenze des Wertebereichs des Knotens a_1 gleich 4 ist. Somit ist Q' für den Wertebereich *real* nicht in Q enthalten. Es gilt $Q' \not\sqsubseteq Q$.

Für den Wertebereich *integer* ist das Ergebnis $G_{Q'_{collapsed}}$ des Algorithmus zur Prüfung auf Enthaltensein der Graph aus Abbildung 4.3 (Seite 30). Auch hier liefert die transitive Hülle keine weiteren Kanten, sodass dieser Graph gleich seiner transitiven Hülle ist. In diesem Fall wird die Ungleichung $a_1 \leq 3$ durch den Graphen impliziert, da hier die obere Schranke des Knotens a_1 gleich 3 ist. Für die Konjunkte $a_1 \geq a_2$ und $a_2 \leq b_2$ besitzt der Graph $G_{Q'_{collapsed}}$ je eine eine Kante. Weiterhin wird die Ungleichung $b_1 > 0$ durch den Graphen impliziert, da die untere Grenze des Knotens b_1 gleich 3 und damit größer 0 ist. Somit gilt in diesem Wertebereich $Q' \sqsubseteq Q$.

Dieser Algorithmus kann so erweitert werden, dass auch Anfragen, die Kreuzprodukte enthalten, auf Erfüllbarkeit getestet werden können. Hierfür muss nach Schritt 4, in dem $F_{Q_{reduced}}$ bestimmt wurde, ein weiterer Schritt eingefügt werden. In diesem Schritt werden alle Ungleichungen in $F_{Q_{reduced}}$ eliminiert, deren Attribute sich nur auf Relationen beziehen, die nicht in Q enthalten sind. Solche Ungleichungen können keine Entsprechung in der transitiven Hülle von $G_{Q'_{collapsed}}$ besitzen, da ihre Attribute in diesem Graph keine Entsprechung besitzen. Dennoch kann $Q' \sqsubseteq Q$ gelten.

Die Komplexität des Algorithmus lässt sich aus den einzelnen Schritten ableiten.

Wie bereits in Abschnitt 4.1.2 gezeigt, kann eine Anfrage Q' in $O(|Q'|)$ auf Erfüllbarkeit geprüft werden. Damit ist auch die Berechnung von $G_{Q'_{collapsed}}$ in $O(|Q'|)$ möglich. Die Variablen in F_Q können in $O(|Q'|)$ durch repräsentative Variablen ersetzt werden. Die Reduzierung von F_Q um Tautologien und Operatoren ist in $O(|Q|)$ möglich. Wie in Abschnitt 3.5.2 beschrieben, kann die transitive Hülle eines Graphen in $O(|V| + |E*| + k^3)$ ermittelt werden. Dabei steht $|V|$ für die Anzahl der Knoten des Graphen $G_{Q'_{collapsed}}$, $|E*|$ für die Anzahl der Kanten der transitiven Hülle des Graphen $G_{Q'_{collapsed}}$ und k gibt die Anzahl der starken Zusammenhangskomponenten von $G_{Q'_{collapsed}}$ wieder. Da der betrachtete Graph $G_{Q'_{collapsed}}$ keine starken Zusammenhangskomponenten mehr enthält, ist $k = 0$. Im schlechtesten Fall ist jeder Knoten der transitiven Hülle mit jedem anderen Knoten verbunden. Damit gilt $|E*| = |V^2|$. Folglich kann die transitive Hülle in $O(|V| + |V|^2)$ und damit in $O(|V|^2)$ bestimmt werden. Da die Anzahl der Knoten im Graphen von der Länge der Anfrage $|Q'|$ abhängt, lässt sich dieser Schritt in $O(|Q'|^2)$ bearbeiten. Der letzte Schritt, indem alle Konjunkte von F_Q mit dem Graph verglichen werden, kann in $O(|Q|)$ ausgeführt werden. Damit ist die Komplexität des gesamten Algorithmus $O(|Q'|^2+|Q|)$ [GSW96b]. Ungleichungen in $F_{Q_{reduced}}$, die ausschließlich Attribute beschreiben, die nicht in Q' enthalten sind, können ebenfalls in $O(|Q|)$ eliminiert werden: Alle Variablen besitzen als Präfix den Namen der Relation (siehe Abschnitt 3.3.4), der sie zugeordnet werden. Dieser kann in $O(1)$ auf Enthaltensein in der Anfrage Q geprüft werden. Somit ändert sich die Komplexität des Gesamtalgorithmus durch Einfügen dieses Schritts nicht.

4.2.5 Konjunktive Anfragen mit θ_{ALL}

Der folgende Algorithmus zur Entscheidung, ob eine konjunktive θ_{ALL} Anfrage in einer anderen konjunktiven θ_{ALL} Anfrage enthalten ist, wurde nur für Anfragen entwickelt, deren Attribute die Domäne *real* besitzen [GSW96b]. In [GSW96b] wird darauf verzichtet, den Algorithmus für den Wertebereich *integer* zu erweitern, da bereits das Erfüllbarkeitsproblem für diesen Wertebereich NP-hart ist [RH80]. Für die Entscheidung, ob eine Anfrage Q' in einer Anfrage Q enthalten ist, muss zunächst das Erfüllbarkeitsproblem gelöst werden. Somit ist auch das Enthaltenseinproblem NP-hart [SKN89]. Dennoch wird hier der Algorithmus für den Wertebereich *real* auch auf den Wertebereich *integer* erweitert.

Die Idee dieses Algorithmus ist gleich der Idee, die dem Algorithmus zur Lösung des Enthaltenseinproblems für konjunktive $\theta_{\neg\neq}$-Anfragen zu Grunde liegt. Zunächst werden mit Hilfe der Graphenrepräsentation alle Konjunkte bestimmt, die von der Anfrage Q' impliziert werden. Anschließend wird für jedes Konjunkt der Anfrage Q eine Entsprechung in Q' gesucht. Wird für alle Konjunkte von Q eine Entsprechung in Q' gefunden, so ist Q' in Q enthalten. Algorithmus 5 (Seite 41) zeigt das benutzte Vorgehen im Pseudocode.

In einem ersten Schritt werden die Anfragen Q' und Q in die logischen Formeln $F_{Q'}$ und F_Q überführt. Auf diesen Formeln werden dann die Axiome aus Abschnitt 4.1.3

Algorithmus 5 : Implikationsproblem $Q' \rightarrow Q$ für θ_{ALL}-Anfragen

Eingabe : Anfrageformeln F'_Q und F_Q
Ausgabe : TRUE oder FALSE

```
1  Bestimmung von G_{Q'_collapsed} aus F'_Q
2  Bestimmung der transitiven Hülle von G_{Q'_collapsed}
3  Bestimmung zusätzlicher ≠-Ungleichungen
4  Ersetzen der Variablen in Q durch die repräsentativen Variablen in Q'
5  Bestimmung von F_{Q_reduced}
6  foreach Konjunkt k ∈ F_{Q_reduced} do
7      if k besitzt keine Entsprechung in der transitiven Hülle von G_{Q'_collapsed} then
8          return FALSE
9  end
10 return TRUE
```

angewandt um die Menge der Vergleichsoperatoren zu reduzieren. Ungleichungen der Form $x_i \neq x_j$ und $x_i \neq w$ werden getrennt von den anderen Ungleichungen betrachtet. Hierfür werden diese Formeln zunächst aus $F_{Q'}$ entfernt. Die verbleibende Formel wird dann mit dem Algorithmus zur Überprüfung konjunktiver $\theta_{\neg\neq}$-Anfragen auf Erfüllbarkeit in $G_{Q'_{collapsed}}$ überführt. Anschließend wird die transitive Hülle von $G_{Q'_{collapsed}}$ bestimmt.

Nun werden die Ungleichungen der Form $x_i \neq x_j$ und $x_i \neq w$ betrachtet. Um alle von Q' implizierten $\neq$-Ungleichungen zu ermitteln, wird ein weiterer Algorithmus benötigt, der am Ende dieses Abschnitts vorgestellt wird. Die Menge der Ungleichungen, die durch Q' impliziert werden, sei mit $U_{Q'}$ bezeichnet.

Danach werden die in Q enthaltenen Variablen, wie bei der Überprüfung konjunktiver $\theta_{\neg\neq}$-Anfragen, durch repräsentative Variablen ersetzt. Anschließend werden, wie in Abschnitt 4.1.3 beschrieben, Tautologien eliminiert und die Menge der benutzten Vergleichsoperatoren reduziert. Die so bestimmte Formel sei $F_{Q_{reduced}}$.

Nun wird jedes Konjunkt $k \in F_{Q_{reduced}}$ mit der transitiven Hülle von $G_{Q'_{collapsed}}$ verglichen. Auch hier wird zur Übersichtlichkeit auf eine textuelle Beschreibung der einzelnen Entsprechungen verzichtet. Diese werden in Tabelle 4.3 (Seite 42) zusammengefasst. Diese Tabelle wird wie Tabelle 4.2 (Seite 39) genutzt, um für die Konjunkte k der Anfrage Q Entsprechungen in $G_{Q'_{collapsed}}$ zu finden. Die Variablen dieser Tabelle haben dabei folgende Bedeutung: A^i_{up} und A^i_{low} sind die Wertebereichsgrenzen des wahren Wertebereichs W'_i, W^i_{up} und W^i_{low} sind die Grenzen des Wertebereichs W_i der Variable x_i in $F_{Q'}$. (x_i, x_j, θ) beschreibt eine Kante in der transitiven Hülle von $G_{Q'_{collapsed}}$.

Wird hierbei nur ein Konjunkt gefunden, das keine Entsprechung in $G_{Q'_{collapsed}}$ besitzt, so gilt $Q' \sqsubseteq Q$ nicht.

Um den Algorithmus auch auf Anfragen mit Kreuzprodukten anwendbar zu machen, muss nach der Bestimmung der Formel $F_{Q_{reduced}}$, ein zusätzlicher Schritt eingefügt wer-

Konjunkt	**Wertebereich**	**Eigenschaft transitive Hülle von $G_{Q'_{collapsed}}$**
$x_i \leq x_j$	*integer* und *real*	$\exists$ eine Kante (x_i, x_j, θ) in $G_{Q'_{collapsed}}$ oder $A^i_{up} \leq A^j_{low}$
$x_i \neq x_j$	*integer* und *real*	$x_i \neq x_j \in U_{Q'}$
$x_i \neq w$	*integer* und *real*	$x_i \neq w \in U_{Q'}$ oder $w \notin W_i$
$x_i \leq w$	*real*	$w \geq A^i_{up}$
$x_i \geq w$	*real*	$w \leq A^i_{low}$
$x_i \leq w$	*integer*	$w \geq A^i_{up}$ oder $\forall w_j \in [w+1, A^i_{up}]$, gilt $x_i \neq w_j \in U_{Q'}$
$x_i \geq w$	*integer*	$w \leq A^i_{low}$ oder $\forall w_j \in [A^i_{low}, w-1]$, gilt $x_i \neq w_j \in U_{Q'}$

Tabelle 4.3: Entsprechungen von k und der transitiven Hülle von $G_{Q'_{collapsed}}$ für konjunktive θ_{ALL}-Anfragen

den, der die Selektionsbedingungen eliminiert, die in $G_{Q'_{reduced}}$ keine Entsprechung haben können (siehe Abschnitt 4.2.4).

Der Algorithmus soll nun an einem Beispiel für den Wertebereich *integer* veranschaulicht werden. Sei die Anfrage $Q' = \{a_1, a_2 | \exists b_1 b_2 (a_1 \geq a_2 \wedge a_1 < b_2 \wedge a_2 \leq b_1 \wedge a_2 < 5 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 > 2 \wedge b_1 \leq 4 \wedge b_1 \neq 4 \wedge a_2 \neq a_1)\}$ und die Anfrage $Q = \{a_1, a_2, a_3 | \exists b_1 b_2 b_3 (5 \leq 6 \wedge a_1 \geq a_2 \wedge a_1 < b_1 \wedge b_1 \leq 3 \wedge a_3 = b_3 \wedge b_3 \neq 6 \wedge b_3 < 0 \wedge b_1 \neq 4)\}$ gegeben. Nun soll geprüft werden ob $Q' \sqsubseteq Q$ gilt.

In einem vorbereitenden Schritt werden beide Anfragen in die zwei Anfrageformeln $F_{Q'} = (a_1 \geq a_2 \wedge a_1 < b_2 \wedge a_2 \leq b_1 \wedge a_2 < 5 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 > 2 \wedge b_1 \leq 4 \wedge b_1 \leq 4 \wedge a_2 \neq a_1 \wedge b_1 \neq 4)$ und $F_Q = (5 \leq 6 \wedge a_1 \geq a_2 \wedge a_1 < b_1 \wedge b_1 \leq 3 \wedge b_1 \neq 4 \wedge a_3 = b_3 \wedge b_3 \neq 6 \wedge b_3 < 0)$ überführt.

Dann wird der Anfragegraph $G_{Q'_{collapsed}}$ mit Hilfe des Algorithmus aus Abschnitt 4.1.3 für die Anfrage Q gebildet. Dabei wird zunächst die Menge der Vergleichsoperationen reduziert. Hierbei wird $a_1 < b_2$ durch $a_1 \leq b_2 \wedge a_1 \neq b_2$, $a_2 < 5$ durch $a_2 \leq 4$ und $b_1 > 2$ durch $b_1 \geq 3$ ersetzt. Die resultiere Anfrageformel $F_{Q'}$ lautet dann $F_{Q'} = (a_1 \leq a_2 \wedge a_1 \leq b_2 \wedge a_1 \neq b_2 \wedge a_2 \leq b_1 \wedge a_2 \leq 4 \wedge b_1 \leq b_2 \wedge b_1 \geq b_2 \wedge b_1 \geq 3 \wedge b_1 \leq 4 \wedge a_2 \neq a_1 \wedge b_1 \neq 4)$. Da $F_{Q'}$ keine trivialen Ungleichungen besitzt, verändert sich $F_{Q'}$ durch Eliminierung dieser Ungleichungen nicht. Anschließend werden die minimalen Wertebereiche der Variablen bestimmt und die Anfrageformel $F_{Q'}$ ohne die Ungleichungen der Form $x_i \neq x_j$ und $x_i \neq w$ in den Graphen $G_{Q'}$ überführt. Abbildung 4.6 (Seite 43) zeigt diesen Graphen. Die Wertebereichgrenzen der Variablen a_2 bzw. b_1 werden durch die Ungleichungen $a_2 \leq 4$ bzw. $b_1 \geq 3$ und $b_1 \leq 4$ festgelegt.

Der Graph $G_{Q'}$ besitzt eine starke Zusammenhangskomponente. Diese wird eliminiert. Anschließend werden, wie in Abschnitt 4.1.3 beschrieben, die wahren Wertebereiche der Variablen ermittelt. Abbildung 4.7 (Seite 43) zeigt den resultierenden Graphen

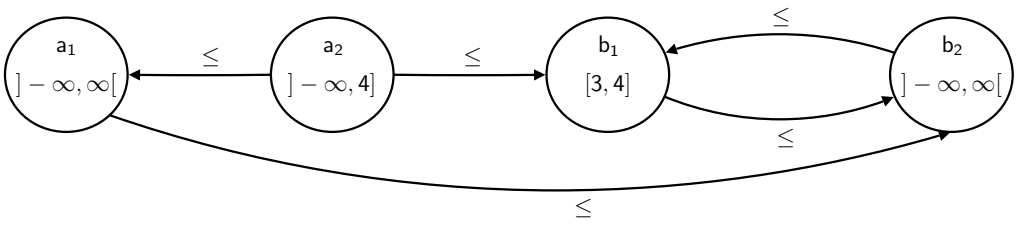

Abbildung 4.6: Anfragegraph G'_Q für den Wertebereich *integer*

$G_{Q'_{collapsed}}$.

Im nächsten Schritt wird die transitive Hülle für $G_{Q'_{collapsed}}$ bestimmt. Die transitive Hülle ist dabei gleich dem Graphen aus Abbildung 4.7 (Seite 43), da in diesem Schritt keine zusätzlichen Kanten gefunden werden können.

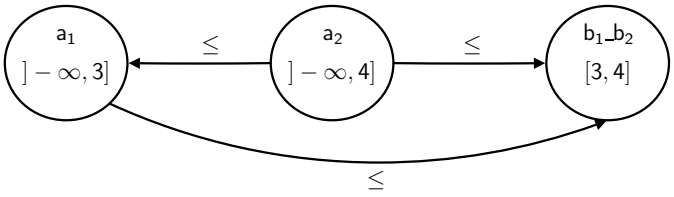

Abbildung 4.7: Anfragegraph $G_{Q'_{collapsed}}$ für den Wertebereich *integer*

Nun werden alle von $F_{Q'}$ implizierten $\neq$-Ungleichungen bestimmt. Hierfür wird der Algorithmus am Ende dieses Abschnitts angewandt. Dieser Algorithmus findet die Ungleichungen $a_1 \neq a_2$, $a_1 \neq b_1$, $a_1 \neq b_2$, $a_1 \neq 4$, $b_1 \neq 4$, $b_2 \neq 4$ und $3 \neq 4$. Somit ist die Menge der von $F_{Q'}$ ableitbaren $\neq$-Ungleichungen $U_{Q'} = \{(a_1 \neq b_1), (a_1 \neq b_2), (a_1 \neq 4), (3 \neq 4), (a_1 \neq a_2), (b_1 \neq 4), (b_2 \neq 4)\}$. Im nächsten Schritt werden die Variablen in F_Q durch repräsentative Variablen aus $F_{Q'}$ ersetzt. Somit ergibt sich für $F_Q = (5 \leq 6 \wedge a_1 \leq a_2 \wedge a_1 < b_1\_b_2 \wedge b_1\_b_2 \leq 3 \wedge a_3 = b_3 \wedge b_3 \neq 6 \wedge b_3 < 0)$. Wie in Abschnitt 4.1.3 beschrieben, wird nun die Menge der Operatoren in F_Q reduziert und triviale Ungleichungen eliminiert. Das Ergebnis dieser Umformungen ist $F_{Q_{reduced}} = (a_1 \leq a_2 \wedge a_1 \leq b_1\_b_2 \wedge a_1 \neq b_1\_b_2 \wedge b_1\_b_2 \leq 3 \wedge a_3 = b_3 \wedge b_3 \neq 6 \wedge b_3 \leq -1)$. Seien a_3 und b_3 zwei Variablen, die Attribute einer Relation, die nicht in der Anfrage Q' enthalten ist, repräsentieren. Somit können die Ungleichungen, die nur die Variablen a_3, b_3 und Konstanten beinhalten aus $F_{Q_{reduced}}$ entfernt werden. Nach diesem Schritt ist $F_{Q_{reduced}} = (a_1 \leq a_2 \wedge a_1 \leq b_1\_b_2 \wedge a_1 \neq b_1\_b_2 \wedge b_1\_b_2 \leq 3)$. Im letzten Schritt werden nun die Konjunkte der Formel $F_{Q_{reduced}}$ mit der transitiven Hülle von $G_{Q'_{collapsed}}$ (siehe Abbildung 4.7, Seite 43) verglichen. Die Ungleichungen $a_1 \leq a_2$ bzw. $a_1 \leq b_1\_b_2$ besitzen eine Entsprechung in diesem Graphen, da der Graph eine Kante von a_1 nach a_2 bzw. von a_1 nach $b_1\_b_2$ besitzt. Die Menge $U_{Q'}$ besitzt sowohl die Ungleichung $a_1 \neq b_1$ als auch $a_1 \neq b_2$, somit besitzt auch $a_1 \neq b_1\_b_2$ eine Entsprechung in Q'. Die Gleichung $b_1\_b_2 \leq 3$ wird nun mit Q' verglichen. Dabei wird festgestellt, dass 3 nicht kleiner oder gleich der oberen Grenze des Wertebereichs von $b_1\_b_2$ in Q' ist. Somit wird überprüft, ob für alle Konstanten c_j im Intervall zwischen

$3+1$ und der oberen Intervallgrenze des Wertebereichs von $b_1\_b_2$ eine Ungleichung der Form $b_1\_b_2 \neq c_j$ in der Menge $U_{Q'}$ existiert. Das zu untersuchende Intervall enthält nur den Wert 4. Somit wird überprüft, ob $b_1 \neq 4 \in U_{Q'}$ und $b_2 \neq 4 \in U_{Q'}$ gilt. Dies ist erfüllt. Somit ist die Anfrage Q' in Q enthalten.

Der Graph $G_{Q'_{collapsed}}$ kann in $O(|Q'|)$ ermittelt werden. Die transitive Hülle wird in $O(|Q'|^2)$ ermittelt. Für die Bestimmung der implizierten $\neq$-Ungleichungen wird $O(|Q'|^{2,376})$ benötigt [GSW96b]. Die Formel $F_{Q_{reduced}}$ kann in $O(|Q|)$ ermittelt werden. Für den Wertebereich *integer* können alle Konjunkte in $O(|Q|)$ auf Implikation überprüft werden. Da im Wertebereich *integer* für Konjunkte der Form $x_i \neq w$ im schlechtesten Fall alle implizierten Ungleichungen $x_i \neq w_j$ in G^*_Q geprüft werden müssen, ist hier die Komplexität höher. Es existieren maximal so viele Ungleichungen $x_i \neq w_j$ wie Konstanten in der Anfrage Q' enthalten sind. Die Anzahl der Konstanten ist durch $O(|Q'|)$ beschränkt. Daher ist die Laufzeit für diesen Schritt $O(|Q|\cdot|Q'|)$. Somit ergibt sich für den Algorithmus im Wertebereich *real* eine Laufzeit von $O(|Q'|^{2,376}+|Q|)$ [GSW96b], für den Wertebereich *integer* ist die Komplexität $O(|Q'|^{2,376}+|Q|\cdot|Q'|)$. Der zusätzliche Schritt zur Eliminierung von Ungleichungen, deren Attribute, sich nicht auf Relationen in Q' beziehen, benötigt $O(|Q|)$. Er erhöht damit die Komplexität des gesamten Algorithmus nicht.

Bestimmung implizierter $\neq$-Ungleichungen

Die von einer Formel $F_{Q'}$ implizierten $\neq$-Ungleichungen werden mit folgendem Vorgehen ermittelt [GSW96b]: In einem ersten Schritt wird die Formel $F_{Q'}$ erweitert. Hierbei werden alle Konstanten durch Variablen ersetzt und neue, durch die Konstanten implizierte Ungleichungen eingefügt. Dabei wird im Einzelnen folgendes Vorgehen gewählt: Zunächst werden alle Konstanten aufsteigend sortiert. Alle unterschiedlichen Konstanten werden dann durch je eine Variable d_i repräsentiert. Dabei wird der Index i entsprechend der Position der Konstanten in der Sortierung gewählt. So wird die kleinste Variable durch d_1 repräsentiert, die nächstgrößere durch d_2 usw. Danach werden die Konstanten in $F_{Q'}$ durch die so ermittelten Variablen ersetzt. Anschließend wird die Formel $F_{Q'}$ für jedes neue Variablenpaar d_i, d_j, für das $j = i+1$ gilt, mit den Ungleichungen $d_i \leq d_j$ und $d_i \neq d_j$ konjunktiv verknüpft. Die so ermittelte Formel heißt $F^+_{Q'}$. Sie enthält keine Konstanten mehr und impliziert alle Ungleichheitsbeziehungen, die auch von den Konstanten impliziert wurden. Die Formel $F^+_{Q'}$ wird dann, wie in Abschnitt 3.5.2 beschrieben, in einen Graphen überführt. Anschließend wird für diesen Graphen die transitive Hülle $G^+_{Q'}$ bestimmt. Die transitive Hülle enthält schließlich alle von $F^+_{Q'}$ implizierten Ungleichungen der Form $x_i < x_j$ und $x_i \leq x_j$, wobei x_i und x_j Variablen von $F^+_{Q'}$ sind.

Anschließend wird ein zweiter Graph bestimmt, der alle Ungleichungen der Form $x_i \neq x_j$ abbildet. Dieser Graph $G^*_{Q'} = (V, E)$ besitzt die Knotenmenge $V = A \cup B \cup C$ und die Kantenmenge $E = E_{AB} \cup E_{AC}$. Ein Knoten v_{x_i} ist genau dann in A und C enthalten, wenn x_i eine Variable in $F^+_{Q'}$ ist. Ein Knoten $v_{x_j x_k}$ ist in B enthalten, wenn die Ungleichung $x_j \neq x_k$ in $F^+_{Q'}$ enthalten ist. Zwischen den Knoten v_{x_i} und $v_{x_j x_k}$ besteht

eine Kante $(v_{x_i}, v_{x_j x_k}) \in E_{AB}$ genau dann, wenn $x_i \leq x_j$ und $x_i \leq x_k$ in $F'_{Q'}$ enthalten ist. Analog existiert eine Kante $(v_{x_j x_k}, v_{x_i}) \in E_{BC}$ dann, wenn $x_j \leq x_i$ und $x_k \leq x_i$ in $F^+_{Q'}$ enthalten ist. Die Menge der Kanten von $G^*_{Q'}$ wird nun mit Hilfe des Graphen $G^+_{Q'}$ erweitert. Hierfür werden nacheinander alle Knoten in B untersucht. Für die beiden von einem Knoten $v_{x_j x_k}$ repräsentierten Variablen werden die entsprechenden Knoten x_j und x_k im Graph $G^+_{Q'}$ gesucht. Für alle Knoten x_i, die über eingehende Kanten von den Knoten x_j und x_k erreichbar sind, wird dann die Kantenmenge E_{AB} um eine Kante $(v_{x_i}, v_{x_j x_k})$ erweitert. Analog wird die Kantenmenge E_{BC} um Kanten erweitert, die in $G^+_{Q'}$ über ausgehende Kanten von den Knoten x_j und x_k zu erreichen sind. Nun können in $G^*_{Q'}$ alle von $F^+_{Q'}$ implizierten Ungleichungen der Form $x_i \neq x_j$ abgelesen werden. Eine Ungleichung $x_i \neq x_j$ gilt genau dann, wenn $x_i \neq x_j$ in $F^+_{Q'}$ enthalten ist oder wenn x_i und x_j in $G^*_{Q'}$ durch einen zweistelligen Pfad verbunden sind.

Das Vorgehen zur Bestimmung der von einer Formel F_Q implizierten $\neq$-Ungleichungen soll nun an einem Beispiel erläutert werden. Hierfür wird ein Beispiel aus [GSW96b] genutzt. Sei die Anfrageformel $F_Q = x_1 \geq x_2 \wedge x_2 \geq x_3 \wedge x_1 \geq 5 \wedge x_1 \geq x_3 \wedge x_2 \leq 3 \wedge x_2 \neq x_3$ gegeben. Diese Formel wird wie oben beschrieben zur Formel $F^+_Q = x_1 \geq x_2 \wedge x_2 \geq x_3 \wedge x_1 \geq d_2 \wedge x_1 \geq x_3 \wedge x_2 \leq d_1 \wedge x_2 \neq x_3 \wedge d_1 \leq d_2 \wedge d_1 \neq d_2$ erweitert. Hierfür wurde die Konstante 5 auf d_2 und 3 auf d_1 abgebildet. Die neuen Ungleichungen sind $d_1 \leq d_2$ und $d_1 \neq d_2$.

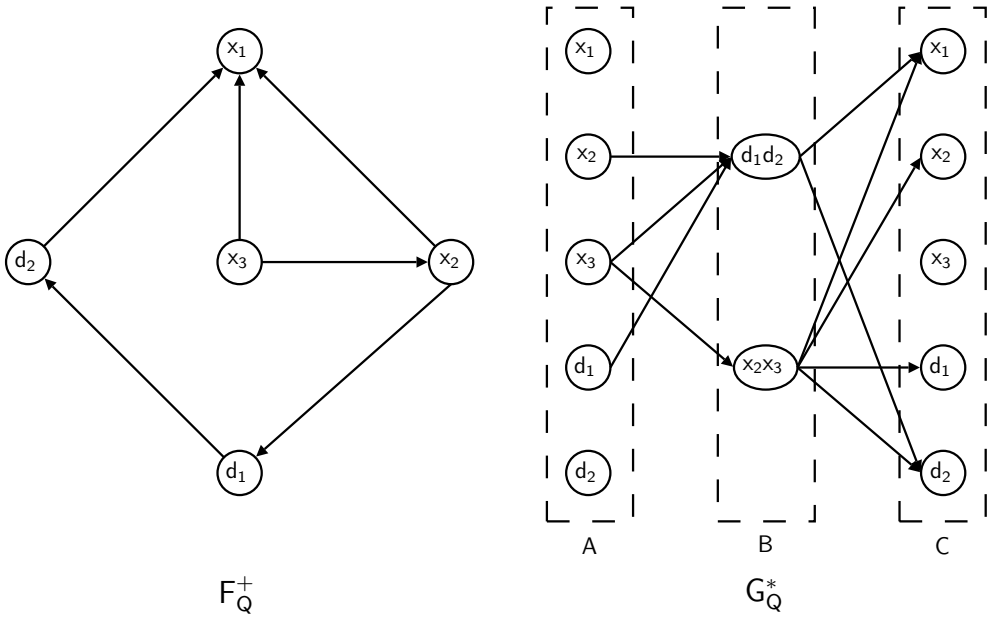

Abbildung 4.8: Deduktion von $\neq$-Ungleichungen

Die Formel F^+_Q wird nun in den entsprechend gekennzeichneten Graphen in Abbildung 4.8 (Seite 45) überführt. Die transitive Hülle G^+_Q dieses Graphen impliziert dann die zusätzlichen Ungleichungen $x_2 \leq d_2$, $x_3 \leq d_1$, $x_3 \leq d_2$. Der Graph G^*_Q besitzt, wie im Algorithmus beschrieben, in den Knotenmengen A und C jeweils einen Knoten für jede

Variable der Formel F_Q^+. B besitzt genau je einen Knoten für jede der Ungleichungen $x_2 \neq x_3$ und $d_1 \neq d_2$. Die Kanten wurden entsprechend der Ungleichungen der Form $x_i \leq x_j$ der Formel F_Q^+ und aus der transitiven Hülle G_Q^+ eingefügt. Aus den Pfaden mit Länge 2 im Graphen G_Q^* lassen sich nun die Ungleichungen $x_3 \neq x_1$, $x_3 \neq x_2$, $x_3 \neq d_1$, $x_3 \neq d_2$, $x_2 \neq x_1$, $x_2 \neq d_2$, $d_1 \neq x_1$ und $d_1 \neq d_2$ ablesen. Die Anzahl der implizierten $\neq$-Ungleichungen ist dabei kleiner als die Zahl der Pfade mit Länge 2, da für $x_3 \neq x_1$ und $x_3 \neq d_2$ zwei Pfade existieren.

4.2.6 Erweiterung um Mengenoperationen

In [SY80] werden Enthaltenseinalgorithmen für Mengenoperationen auf Tableauanfragen vorgestellt. Die dort benutzten Tableauanfragen sind keine gekennzeichneten Tableauanfragen, die äquivalent den hier vorgestellten $\theta_=$-Anfragen sind [SH99], sondern einfache Tableauanfragen. Diese Anfragen sind somit azyklisch (siehe Abschnitt 3.5.3).

Zunächst wird auf das Enthaltenseinproblem zweier Vereinigungen von $\theta_=$-Anfragen eingegangen: Seien zwei Anfragen $Q' = \cup_{i=1}^{n} Q'_i$ und $Q = \cup_{j=1}^{m} Q_j$ gegeben, die beide eine Vereinigung konjunktiver $\theta_=$-Anfragen beschreiben. So ist Q' in Q enthalten, wenn beide Anfragen das gleiche Relationenschema beschreiben und jede Teilanfrage Q'_i von Q' in einer Teilanfrage Q_i von Q enthalten ist [SY80].

Da jede azyklische, konjunktive $\theta_=$-Anfrage Q'_i in $O(|Q'_i| \cdot |Q_j| \cdot log(|Q_j|))$ auf Enthaltensein in einer Anfrage Q_j geprüft werden kann (siehe Abschnitt 4.2.3), wird maximal $O(\sum_{j=1}^{m}(|Q'_i| \cdot |Q_j| \cdot log(|Q_j|)))$ für die Prüfung benötigt, ob Q'_i in einer der Anfragen Q_j enthalten ist. Somit ist zur Prüfung, ob alle Anfragen Q'_i in den Anfragen Q_j enthalten sind, ein Algorithmus mit Laufzeit $O(\sum_{i=1}^{n}\sum_{j=1}^{m}(|Q'_i| \cdot |Q_j| \cdot log(|Q_j|))$ nötig. Diese Formel kann wie folgt vereinfacht werden:

$$
\begin{aligned}
& O(\sum_{i=1}^{n}\sum_{j=1}^{m}(|Q'_i| \cdot |Q_j| \cdot log(|Q_j|)) \\
= \; & O(\sum_{i=1}^{n}(|Q'_i| \cdot \sum_{j=1}^{m}(|Q_j| \cdot log(|Q_j|)))) \\
= \; & O(\sum_{i=1}^{n}(|Q'_i|) \cdot \sum_{j=1}^{m}(|Q_j| \cdot log(|Q_j|))) \\
= \; & O(|Q'| \cdot \sum_{j=1}^{m}(|Q_j| \cdot log(|Q_j|)))
\end{aligned}
$$

Da die Längen der Anfragen $|Q_j|$ durch $|Q|$ beschränkt sind, ergibt sich für die Laufzeit des Enthaltenseinproblems zweier Vereinigungen azyklischer, konjunktiver $\theta_=$-Anfragen die obere Grenze $O(m \cdot |Q'| \cdot (|Q| \cdot log(|Q|)))$.

Für zwei Elemenentardifferenzen $Q' = Q'_1 - Q'_2$ und $Q = Q_1 - Q_2$ gilt $Q' \sqsubseteq Q$ genau dann, wenn beide dasselbe Relationenschema besitzen, $Q'_1 \sqsubseteq Q_1$ und $Q'_1 \cap Q_2 \sqsubseteq Q'_2$

gilt [SY80]. Hierbei sind Q'_2 und Q_2 Vereinigungen konjunktiver $\theta_=$-Anfragen, Q'_1 und Q_1 sind jeweils konjunktive $\theta_=$-Anfragen. Da der Schnittmengenoperator ein Sonderfall des Verbundes ist [SY80], bildet $Q'_1 \cap Q_2$ eine Vereinigung einer konjunktiven Anfrage mit einem Verbund konjunktiver Anfragen. Da Vereinigung und Verbund distributiv sind [SH99], kann $Q'_1 \cap Q_2$ wieder in eine Vereinigung konjunktiver Anfragen übertragen werden.

Die Komplexität des Algorithmus hängt von der Laufzeit zur Überprüfung der Enthaltenseinbeziehungen $Q'_1 \sqsubseteq Q_1$ und $Q'_1 \cap Q_2 \sqsubseteq Q'_2$ ab. Die Länge der Anfragen Q'_1, Q_1, Q_2 und Q'_2 ist von $|Q'|$ bzw. $|Q|$ beschränkt. $|Q'_1 \cap Q_2|$ hingegegen ist gleich $m \cdot |Q'_1| + |Q_2|$, wobei m die Anzahl der konjunktiven Anfragen in Q_2 angibt, da jede konjunktive Anfrage in Q_2 mit der Anfrage Q'_1 verbunden werden muss. Da Q'_1 und Q_1 jeweils konjunktive Anfragen sind, $Q'_1 \cap Q_2$ und Q'_2 hingegen Vereinigungen konjunktiver Anfragen, ist die Laufzeit eines entsprechenden Algorithmus für azyklische, konjunktive $\theta_=$-Anfragen $O(|Q'| \cdot |Q| \cdot log(|Q|) + m \cdot (m \cdot |Q'_1| + |Q_2|) \cdot |Q'| \cdot log(|Q'|))$ und damit $O(|Q'| \cdot |Q| \cdot log(|Q|) + m \cdot |Q'| \cdot log(|Q'|) \cdot (m \cdot |Q'| + |Q|))$.

Diese Ergebnisse für Vereinigungen und Differenzen azyklischer, konjunktiver $\theta_=$ - Anfragen lassen sich nicht direkt auf Mengenoperationen mächtigerer konjunktiver Anfragen übertragen [Klu88]. In [Klu88] werden daher zwei neue Klassen konjunktiver Anfragen eingeführt. Die Anfragen dieser Klassen dürfen ausschließlich atomare Selektionsbedingungen der Form $x_i\ \theta\ w$ bzw. $w\ \theta\ x_i$ mit $\theta \in \{<, \leq, =\}$ enthalten. Die Klassen dieser Anfragen heißen linke Halbintervallanfragen (englisch: left semiinterval queries) bzw. rechte Halbintervallanfragen (englisch: right semiinterval queries). Ist $Q = \cup_{i=1}^{n} Q_i$ eine Vereinigung linker bzw. rechter Halbintervallanfragen und Q' eine linke bzw. rechte Halbintervallanfrage, so gilt $Q' \sqsubseteq Q$ genau dann, wenn ein Q_i existiert, sodass $Q' \in Q_i$ gilt [Klu88]. Somit kann eine Vereinigung von Halbintervallanfragen $Q' = \cup_{i=1}^{n} Q'_i$ auf Enthaltensein in einer anderen derartigen Anfrage $Q = \cup_{j=1}^{m} Q_j$ überprüft werden, indem für alle Q'_i ein Q_j gesucht wird, sodass $Q'_i \sqsubseteq Q_j$ gilt. Wird für ein Q'_i kein Q_j gefunden, sodass diese Beziehung gilt, so gilt $Q' \not\sqsubseteq Q$.

Zur Prüfung, ob eine Halbintervallanfrage in einer anderen Halbintervallanfrage enthalten ist, kann der Algorithmus zur Prüfung des Anfrageenthaltenseins für konjunktive $\theta_{\neg\neq}$-Anfragen genutzt werden, da die Menge der Halbintervallanfragen in der Menge der konjunktiven $\theta_{\neg\neq}$-Anfragen enthalten ist. Somit können sowohl für den Wertebereich *integer* als auch für den Wertebereich *real* zwei Halbintervallanfragen in $O(|Q'|^2 + |Q|)$ auf Enthaltensein geprüft werden. Da im schlechtesten Fall jede Anfrage Q'_i mit jeder Anfrage Q_j auf Enthaltensein geprüft werden muss, ist die Laufzeit des gesamten Algorithmus $O(\sum_{i=1}^{n} \sum_{j=1}^{m} |Q'_i|^2 + |Q_j|)$. Dieses Ergebnis lässt sich weiter vereinfachen zu $O(m \cdot \sum_{i=1}^{n} |Q'_i|^2 + \sum_{j=1}^{m} |Q_j|)$. Da $\sum_{j=1}^{m} |Q_j| = |Q|$ gilt und $|Q'_i|$ durch $|Q'|$ beschränkt wird, ist dieses Ergebnis äquivalent zu $O(m \cdot n \cdot |Q'|^2 + |Q|)$. Somit lassen sich zwei Halbintervallanfragen in $O(m \cdot n \cdot |Q'|^2 + |Q|)$ auf Enthaltensein prüfen.

Um Vereinigungen und Differenzen von Anfragen auf Enthaltensein zu prüfen, wird folglich versucht, die einzelnen Teilanfragen miteinander zu vergleichen. Dieses Vorgehen scheitert aber bei Teilanfragen, die mächtiger sind als Halbintervallanfragen [Klu88].

Daher soll im Weiteren auf die Betrachtung von Mengenoperationen auf konjunktiven Anfragen, die mächtiger sind als die hier vorgestellten konjunktiven Halbintervallanfragen, verzichtet werden.

4.2.7 Zusammenfassung

Dieser Abschnitt fasst die bisherigen Ergebnisse zusammen und gibt einen Überblick über weitere komplexitätstheoretische Ergebnisse von Anfrageenthaltenseinalgorithmen. Für die Grundlagen der Komplexitätstheorie sei auf entsprechende Grundlagenliteratur verwiesen [Weg03, Rei99]. In [SU02] wird ein Überblick über verschiedene Probleme der Komplexitätsklasse Π_2^P gegeben, auf die in der Grundlagenliteratur nicht eingegangen wird.

Zunächst wird auf die oben beschriebenen Algorithmen zur Lösung des Enthaltenseinproblems eingegangen. Besitzen Anfragen nur Projektionen oder Kreuzprodukte, so ist in $O(|Q'|)$ prüfbar, ob die Anfragen in einer anderen Anfrage enthalten sind.

Die Lösung des Problems für konjunktive Anfragen, die ausschließlich Selektionsbedingungen der Form $x_i = x_j$ oder $x_i = w$ zulassen, sind NP-vollständig [CM77]. Werden konjunktive Anfragen mit Selektionsprädikaten der Form $x_i\ \theta\ x_j$ und $x_i\ \theta\ w$ mit $\theta \in \{\leq, <, =\}$ zugelassen, so ist das Enthaltenseinproblem Π_2^P-vollständig [Klu88]. Wird als Vergleichsoperator θ auch noch $\neq$ zugelassen, so ist das zugehörige Problem Π_2^P-hart [vdM97].

Bei konjunktiven Anfragen hat demnach die Menge der zugelassenen Vergleichsoperatoren in Selektionsprädikaten großen Einfluss auf die Komplexität des Algorithmus zur Lösung des Anfrageenthaltenseinproblems. Daher wurden Algorithmen abhängig von den zugelassenen Operatoren vorgestellt.

Konjunktive Anfragen, die nur den =-Operator zulassen, können auf Hypergraphen abgebildet werden. Für diese Klasse von Anfragen ist das Enthaltenseinproblem in polynomialer Zeit lösbar, wenn der Hypergraph azyklisch ist oder wenn der Hypergraph eine Anfragebreite von maximal k besitzt und ein Zerlegungsbaum für den Hypergraphen gegeben ist. Für azyklische Anfragen ist der Aufwand zur Lösung des Enthaltenseinproblems $O(|Q'| * |Q| * log(Q))$ und für zyklische Anfragen mit Anfragebreite $\leq k$ und gegebenem Zerlegungsbaum $O(k * |Q'| * |Q|^k * log(|Q|)$ [CR00]. Alternativ kann die Klasse von azyklischen Anfragen auch auf Tableaus abgebildet werden. Für Sonderfälle von Tableauanfragen existieren Algorithmen, welche mit Aufwand $O((|Q'| + |Q|)^2)$ bzw. $O((|Q| + |Q'|)^3)$ gelöst werden können [SY80]. Die Benutzung dieser Algorithmen wird dann sinnvoll, wenn gezeigt werden kann, dass diese Algorithmen auch auf gekennzeichnete Tableaus, also Tableaus welche auch zyklische Anfragen zulassen, übertragen werden können. Entsprechende Beweise wurden im Rahmen dieser Arbeit nicht gefunden.

Für konjunktive $\theta_{\neg\neq}$-Anfragen existieren in polynomialer Zeit lösbare Algorithmen, wenn die Wertebereiche der Attribute nur *integer* oder *real* zugelassen sind. Die Anfragen werden hierfür auf Graphen abgebildet. Bei Anfragen, die nur Attribute mit dem Wertebereich *integer* zulassen, wird zwischen Anfragen mit Selektionsprädikaten der Form

$x_i\ \theta\ x_j$ bzw. $x_i\ \theta\ w$ und Selektionsprädikaten der Form $x_i\ \theta\ x_j + w$ unterschieden. Für den ersten Fall existieren Algorithmen mit Laufzeit $O(|Q'|^2 + |Q|)$ [GSW96b] und für den zweiten Fall mit Laufzeit $O(n^3 + |Q|)$ [SKN89]. Hierbei beschreibt n die Anzahl der Attribute in Anfrage Q'. Ist der Wertebereich *real* zugelassen, so kann das Enthaltenseinproblem für Selektionsprädikate der Form $x_i\ \theta\ x_j$ und $x_i\ \theta\ w$ in $O(|Q'|^2 + |Q|)$ gelöst werden [GSW96b]. In dieser Arbeit wird auf die Algorithmen welche Selektionsprädikate der Form $x_i\ \theta\ x_j + w$ zulassen nicht weiter eingegangen, da diese Form von Selektionsprädikaten in der relational vollständigen Algebra nicht vorgesehen ist.

Wird auch der Operator $\neq$ zugelassen, so ist das Enthaltenseinproblem für Attribute im Wertebereich *integer* NP-hart [SKN89]. Wird aber davon ausgegangen, dass alle beteiligten Anfragen erfüllbar sind, so kann ein Algorithmus gefunden werden, der das Enthaltenseinproblem für Anfragen mit Selektionsprädikaten der Form $x_i\ \theta\ x_j$ bzw. $x_i\ \theta\ w$ in $O(min(|Q'|^{2,376} + |Q'| \cdot |Q|)$ löst. Für Attribute mit dem Wertebereich *real* wird wieder zwischen Anfragen mit Selektionsprädikaten der Form $x_i\ \theta\ x_j$ bzw. $x_i\ \theta\ w$ und Selektionsprädikaten der Form $x_i\ \theta\ x_j + w$ unterschieden. Probleme des ersten Falls können nach Abbildung auf Graphen in $O(min(|Q'|^{2,376} + |Q|, |Q'| * |Q|)$ gelöst werden [GSW96b]. Probleme mit Selektionsprädikaten der Form $x_i\ \theta\ x_j + w$ können mit $O(|Q'| * n^2 + |Q|)$ gelöst werden [GSW96a]. Dabei gibt n die Anzahl der beteiligten Variablen wieder. Auch hier sei das Ergebnis für Selektionsprädikate der Form $x_i\ \theta\ x_j + w$ nur der Vollständigkeit halber angegeben.

Op.	**Einschränkungen**	**Ergebnis**	**Quelle**
$\theta_=$	keine Einschränkungen	NP-vollständig	[CM77]
	Hypergraph der Anfrage T azyklisch	$O(\|Q'\| \cdot \|Q\| \cdot log(\|Q\|))$	[CR97]
	Zerlegungsbaum für T mit Breite k	$O(k \cdot \|Q'\| \cdot \|Q\|^k \cdot log(\|Q\|))$	[CR97]
$\theta_{\neg\neq}$	keine Einschränkungen	Π_2^P-vollständig	[Klu88]
	$dom(x_i)$ = integer	$O(\|Q'\|^2 + \|Q\|)$	[GSW96b]
	$dom(x_i)$ = real	$O(\|Q'\|^2 + \|Q\|)$	[GSW96b]
θ_{ALL}	keine Einschränkungen	Π_2^P-hart	[vdM97]
	$dom(x_i)$ = integer	NP-hart	[GSW96b]
	$dom(x_i)$ = integer, Q' erfüllbar	$O(\|Q'\|^{2,376} + \|Q'\| \cdot \|Q\|)$	-
	$dom(x_i)$ = real	$O(min(\|Q'\|^{2,376} + \|Q\|, \|Q'\| \cdot \|Q\|))$	[GSW96b]

Tabelle 4.4: Komplexität von Enthaltenseinalgorithmen für konjunktive Anfragen

Die hier vorgestellten Ergebnisse für konjunktive Anfragen werden in Tabelle 4.4 (Seite 49) übersichtlich zusammengefasst. Sie können in Spezialfällen auf Anfragen mit Mengenoperationen erweitert werden. In [SY80] wird gezeigt, dass eine Vereinigung aus konjunktiven $\theta_=$-Anfragen Q' genau dann in einer anderen Vereinigung konjunktiver $\theta_=$-Anfragen Q enthalten ist, wenn jede konjunktive Anfrage aus Q' in einer konjunktiven

Anfrage von Q enthalten ist. Der entsprechende Enthaltenseinalgorithmus ist in NP [Klu88]. Sind alle beteiligten konjunktiven $\theta_{=}$-Anfragen azyklisch, so kann dieses Problem in $O(m \cdot |Q'| \cdot (|Q| \cdot log(|Q|)))$ gelöst werden. Weiterhin können Elementardifferenzen konjunktiver $\theta_{=}$-Anfragen in $O(|Q'| \cdot |Q| \cdot log(|Q|) + m \cdot |Q'| \cdot log(|Q'|) \cdot (m \cdot |Q'| + |Q|))$ auf Enthaltensein geprüft werden. Vereinigungen linker bzw. rechter Halbintervallanfragen können in $O(m \cdot n \cdot |Q'|^2 + |Q|)$ auf Enthaltensein getestet werden.

4.3 Faltung von Anfragen

Das Problem der Faltung von Anfragen (englisch: query folding) erweitert das Problem des Anfrageenthaltenseins. Hierbei wird nicht mehr nur untersucht, ob eine Anfrage Q' in einer anderen Anfrage Q enthalten ist, sondern, ob eine Anfrage Q' mit einer Menge von Ressourcen $\mathcal{V} = \{V_1, ..., V_n\}$ beantwortet werden kann [Qia96]. Dabei können Ressourcen materialisierte Sichten, Anfragen, die auf anderen Datenbanken ausgeführt werden können, oder, wie im vorliegenden Fall, gecachte Anfrageergebnisse sein [Qia96]. Bei der Anfragefaltung (in der Literatur auch: problem of answering queries using views oder rewriting queries using views [Hal01]) wird zwischen drei Arten von Anfragefaltungen unterschieden: äquivalente Faltungen (englisch: equivalent rewritings), maximal enthaltene Faltungen (englisch: maximally-contained rewritings) und enthaltene Faltungen (englisch: contained rewritings). Die Anfrage Q wird als äquivalente Faltung bezeichnet, wenn sie nur Ressourcen aus $\mathcal{V}$ enthält und äquivalent der Anfrage Q' ist. Betrachtet eine Anfrage Q in einer Anfragesprache $\mathcal{L}$ nur Ressourcen aus $\mathcal{V}$, ist Q in der ursprünglichen Anfrage Q' enthalten und gibt es keine andere Anfrage $Q_i \in \mathcal{L}$, so dass $Q \subseteq Q_i \subseteq Q'$ mit Q_i äquivalent Q gilt, so wird Q als maximal enthaltene Faltung von Q' bezeichnet. Ist eine Faltung Q' in Q enthalten, aber keine maximal enthaltene Faltung, so wird sie als enthaltene Faltung bezeichnet [Hal01].

Da mit Hilfe von Faltungsalgorithmen nur eine Menge von Ressourcen bestimmt werden kann, die für die Beantwortung einer Anfrage notwendig ist, nicht aber welchen Beitrag die einzelne Ressource für die Beantwortung einer Anfrage leistet, ist sie für das semantische Cachen nur bedingt geeignet. Wie später gezeigt wird, ist es beim semantischen Cachen notwendig, die an den Cache gestellte Anfrage zu unterteilen: Ein Teil beschreibt die Teilanfrage, die durch den semantischen Cache beantwortet werden kann. Der andere Teil beschreibt die Teilanfrage, die von der zu Grunde liegenden Datenbank bearbeitet wird. Diese Unterteilung kann durch die Algorithmen zur Anfragefaltung nicht realisiert werden.

Kapitel 5

Semantisches Cachen

Traditionelle Caching Ansätze bei relationalen Datenbanken beschäftigen sich mit dem Puffern von auf der Festplatte abgelegten Datenbankinhalten im Hauptspeicher des Servers, um die Zugriffslücke abzumildern und damit die Performanz der Datenbank zu steigern [SH99]. Das Puffern von Daten auf dem Client ist dabei nicht vorgesehen. In diesem Szenario wird der Server zum Flaschenhals, da viele komplexe Clientanfragen auf einem Server ausgeführt werden [SH99]. Deshalb wurde bei objektorientierten Datenbanken dazu übergegangen, das Puffern der Datenbankinhalte in Form von Seiten oder Objekten auf den Client zu verlagern. In diesem Fall ist der Server so schwach belastet, dass er auch bei komplexen Anfragen auf dem Client keinen Flaschenhals darstellen dürfte. Allerdings steigt der Kommunikationsaufwand zwischen Client und Server [SH99].

Besonders wenn logisch zusammenhängende Datensätze auf unterschiedlichen Pufferseiten verteilt sind, steigt neben dem Kommunikationsaufwand zwischen Client und Server auch die Menge der auf dem Client unnötig gespeicherten Tupel [RDK03]. Da keine semantischen Informationen über die Datenseiten gespeichert werden, muss vor jeder Anfragebearbeitung der Server kontaktiert werden. Daher bietet es sich an, in Szenarien, in denen keine permanente, leistungsfähige Datenleitung zwischen Client und Server besteht, mit dem semantischen Cachen einen anderen Ansatz zu wählen [RDK03].

Beim semantischen Cachen speichert der Client sowohl die Ergebnisse vorangegangener Anfragen als auch deren semantische Beschreibung [DFJ+96, KB96]. Bei neuen Anfragen, werden dann aus den semantischen Beschreibungen zwei neue Anfragen abgeleitet, eine Filteranfrage (englisch: probe query), welche die im Cache enthaltenen Tupel beschreibt, und eine Kompensationsanfrage (englisch: remainder query), welche die Daten beschreibt, die vom Server abgerufen werden müssen [DFJ+96]. Das Vorgehen wird im folgenden Beispiel illustriert:

Gegeben seien die Relationen des Beispielszenarios (siehe Kapitel 2). An einem Client wird zunächst die Anfrage Q_1 gestellt, um die Urlaubsanfangs- und Enddaten der Mitarbeiter zu ermitteln, die ein Gehalt kleiner 2000.00 € erhalten. Das Ergebnis dieser Anfrage wird im Cache gehalten.

Anfrage Q_1:

$\pi_{MName,Anfang,Ende}(\sigma_{Mitarbeiter.MID=Urlaub.MID \wedge Gehalt<2000.00}(r(Mitarbeiter) \times r(Urlaub)))$

Daraufhin wird eine ergänzende Anfrage Q_2 gestellt, die alle Urlaubsanfangs- und Enddaten der Mitarbeiter abfragt, die ein Gehalt kleiner 10000.00 € besitzen.

Anfrage Q_2:

$\pi_{MName,Anfang,Ende}(\sigma_{Mitarbeiter.MID=Urlaub.MID \wedge Gehalt<3000.00}(r(Mitarbeiter) \times r(Urlaub)))$

Das Anfrageergebnis der Anfrage Q_1 ist komplett in dieser neuen Anfrage enthalten. Die Filteranfrage ist in diesem Fall also Q_1. Da die Anfrage Q_1 nicht alle abgefragten Daten enthält, wird die Anfrage Q_2 um die Anfrage Q_1 verringert. Es wird also eine Anfrage Q_3 ermittelt, welche die fehlenden Mitarbeiter mit einem Gehalt zwischen 2000.00 € und 3000.00 € enthält.

Anfrage Q_3:

$\pi_{MName,Anfang,Ende}(\sigma_{Mitarbeiter.MID=Urlaub.MID \wedge Gehalt \geq 2000.00 \wedge Gehalt<3000.00}(r(Mitarbeiter) \times r(Urlaub)))$

Somit ist die Kompensationsanfrage die Anfrage Q_3, sie wird auf dem Server ausgeführt. Die Vereinigung aus der Kompensationsanfrage und der Filteranfrage bildet das gesuchte Ergebnis.

Neben der Reduktion der Serverlast und des Kommunkiationsaufwandes zwischen Client und Server, bietet dieses Vorgehen noch weitere Vorteile. So kann auf das wiederholte versenden sensitiver Daten verzichtet werden, indem diese permanent im Cache des Clients gehalten werden. Manche Datenbanken können vom Client nicht immer erreicht werden. Wenn eine Anfrage aber zum Teil vom Client aus beantwortet werden kann, ist wenigstens die Rückgabe eines Teils des Ergebnisses möglich. In manchen Fällen kann eine gute Näherung einer Aggregatfunktion, wie dem Durchschnitt, auf der Basis der im Cache enthaltenen Datensätze ermittelt werden. Schließlich sind mit dem semantischen Cachen noch einige Vorteile für die Benutzerinteraktion verbunden [GG97]: Der Teil des Anfrageergebnisses der im Cache enthalten ist, kann sofort zurückgegeben werden, während der übrige Teil auf dem Server bestimmt wird. So kann die Antwortzeit reduziert werden. Manchmal interessiert sich der Benutzer nicht für alle Ergebnisse, welche die Anfrage liefert. Wenn beispielsweise das Ergebnis einer Anfrage wesentlich größer ist, als vom Benutzer erwartet. In diesen Fällen können bereits die im Cache enthaltenen Tupel als Antwort ausreichen.

Neben Situationen, in welchen das semantische Cachen Vorteile gegenüber anderen Caching-Ansätzen bietet, gibt es einige Szenarien in welchen das Benutzen eines semantischen Caches Nachteile hat. Besonders dann, wenn das Bestimmen von Filteranfragen zu komplexen Kompensationsanfragen führt, ist es nicht mehr sinnvoll diese Anfragen zu bestimmen.

Im Folgenden wird zunächst auf die entsprechenden Gebiete des semantischen Cachen näher eingegangen. Dabei wird beschrieben, welcher Teil der Anfrage vom Cache beantwortet werden kann und welcher Teil der Anfrage zusätzlich ermittelt werden muss. Dann wird dargestellt, wie der semantische Cache strukturiert ist und schließlich wird

auf die Bearbeitung konjunktiver Anfragen näher eingegangen. Abschließend wird die Erweiterung des Bearbeitungsalgorithmus um Mengenoperationen betrachtet.

Ist der Cache leer, so wird stets das komplette Anfrageergebnis im Cache abgelegt [RD98]. Für jede weitere vom Client gestellte Anfrage (Clientanfrage) wird entschieden, welche Teile des Anfrageergebnisses im Cache gespeichert werden [RD98]. Hierbei sind die in Abbildung 5.1 (Seite 53) illustrierten Szenarien denkbar.

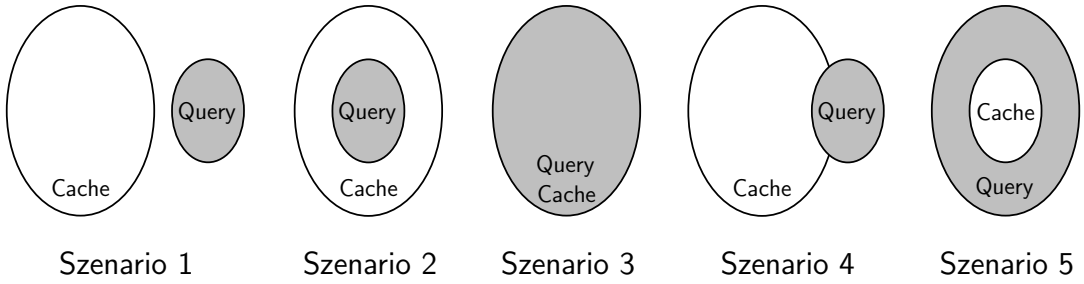

Abbildung 5.1: Mögliche Beziehungen zwischen Anfrage und Cache

Sei C die Beschreibung aller im Cache enthaltenen Tupel und Q eine Datenbankanfrage auf der Datenbank d. $\mathcal{C}$ seien die Daten im semantischen Cache.

Wenn $C \wedge Q$ nicht erfüllbar ist, enthält der Cache das Ergebnis der Anfrage nicht. In diesem Fall muss die Antwort komplett auf dem Server abgearbeitet werden. Die zurückgegebenen Tupel werden dann vollständig dem Cache hinzugefügt. Dieser Fall entspricht Szenario 1 [RD98].

Ist das Ergebnis der Anfrage komplett im Cache enthalten, so kann die Anfrage ohne Kommunikation mit dem Server aus dem Cache beantwortet werden. In diesem Fall gilt $Q \sqsubseteq C$. Das Anfrageergebnis wird nicht im Cache gespeichert. Lediglich Metainformationen, wie der Zeitpunkt des letzten Zugriffs auf die entsprechenden Tupel, werden geschrieben [LC01]. Meist gilt in diesem Fall $C \not\sqsubseteq Q$ wie in Szenario 2. Dann muss die Clientanfrage auf dem Cache ausgeführt werden, das Ergebnis ist $Q(\mathcal{C})$. Szenario 3 illustriert einen Sonderfall: Stimmt der Inhalt des Caches exakt mit der Anfrage überein, gilt $C \equiv Q$. Hier wird der komplette Inhalt des Caches $\mathcal{C}$ zurückgegeben [LC01].

Trifft keiner dieser Fälle zu, so ist das Ergebnis der Anfrage nur teilweise im Cache enthalten. In diesem Fall muss ein Teil der Anfrage vom Server und ein Teil aus dem Cache beantwortet werden. Es gilt $Q \not\sqsubseteq C$. Hierbei überlappen meist Cache und Anfrage, also $C \not\sqsubseteq Q$, wie in Szenario 4. In diesem Fall bildet die Vereinigung aus $Q(\mathcal{C})$ vom Client und $Q(d) - \mathcal{C}$ vom Server die gesuchte Antwort. In Ausnahmefällen ist der Cache in der Anfrage enthalten, es gilt $C \sqsubseteq Q$. In diesem in Szenario 5 dargestellten Fall, bildet die Vereinigung des Caches $\mathcal{C}$ und $Q(d) - \mathcal{C}$ aus vom Server das gesuchte Anfrageergebnis [LC01]. Die vom Server gelieferten Ergebnisse werden dann dem Cache zugefügt. Tabelle 5.1 (Seite 54) fasst die Ergebnisse zusammen.

Szenario	Eigenschaften	Ergebnis von	
		Cache	Server
Szenario 1	$Q \wedge C$ nicht erfüllbar	$\emptyset$	$Q(d)$
Szenario 2	$Q \sqsubseteq C \wedge C \not\sqsubseteq Q$	$Q(\mathcal{C})$	$\emptyset$
Szenario 3	$Q \equiv C$	$\mathcal{C}$	$\emptyset$
Szenario 4	$Q \not\sqsubseteq C \wedge C \not\sqsubseteq Q$	$Q(\mathcal{C})$	$Q(d) - \mathcal{C}$
Szenario 5	$Q \not\sqsubseteq C \wedge C \sqsubseteq Q$	$\mathcal{C}$	$Q(d) - \mathcal{C}$

Tabelle 5.1: Mögliche Beziehung zwischen Anfrage und Cache [LC01]

5.1 Semantische Segmente

Um das Vorgehen bei neuen Anfragen zu vereinfachen, werden gestellte Anfragen nicht mit dem gesamten Cache verglichen [RD98]. Vielmehr wird der semantische Cache $C = \{S_1, ..., S_k\}$ als eine Menge von k semantischen Segmenten S_i (englisch: semantic segments [RD98], semantic regions [DFJ+96] oder fragments [LLS99]) betrachtet. Jedes semantische Segment S_i repräsentiert das Ergebnis oder Teile von Ergebnissen vorangegangener Clientanfragen Q_j an die Datenbank [RD98]. Dabei wird jedes im Cache enthaltene Tupel mit genau einem dieser Segmente assoziiert [DFJ+96].

Wenn eine Anfrage Q nur teilweise von einem semantischen Segment S beantwortet werden kann, so werden die verbleibenden Tupel mit Hilfe einer Kompensationsanfrage Q^* vom Server bezogen. Im Ergebnis werden die in Abbildung 5.2 (Seite 54) dargestellten Fragmente bestimmt.

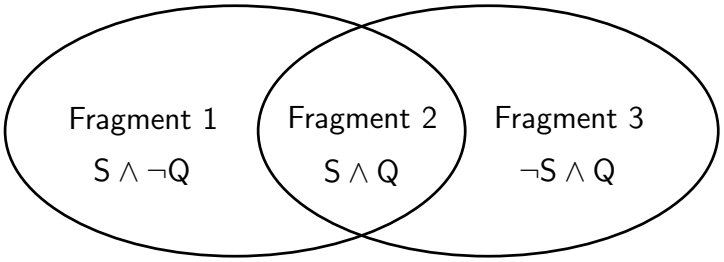

Abbildung 5.2: Beziehungen zwischen Fragmenten [RD98]

Fragment 1 entspricht dem Teil des semantischen Segments S, der nicht zur Beantwortung der Anfrage Q benötigt wird. Fragment 2 enthält die im semantischen Segment S vorhandenen Tupel, die für die Beantwortung der Anfrage benutzt werden. Fragment 3 enthält alle Informationen, die mittels der Kompensationsanfrage Q^* vom Server bezogen wurden.

Aus diesen drei Fragmenten können nun auf unterschiedliche Weise neue semantische Segmente gebildet werden. Die drei Fragmente können zu einem neuen Segment vereinigt werden (englisch: complete coalescence). Dann besteht das neue semantische Segment aus

dem alten Segment und der Kompensationsanfrage. Alternativ kann die gestellte Anfrage ein neues semantisches Segment bilden und das alte Segment um den mit dem neuen Segment gemeinsamen Teil verringert werden. Dieser Fall wird als teilweise Vereinigung (englisch: partial coalescence) bezeichnet. In diesem Fall ist Fragment 1 ein Semantisches Segment und die Vereinigung von Fragment 2 und Fragment 3 ein anderes Segment. Eine dritte Möglichkeit ist die Speicherung aller drei Fragmente als eigene Segmente [RD98].

Der Vorteil der vollständigen Vereinigung besteht darin, dass die Anzahl der im Cache enthaltenen Segmente reduziert und damit die Länge des Cache-Indexes reduziert werden kann. Zudem wird die Beantwortung von neuen Anfragen einfacher, da nur wenige semantische Segmente untersucht werden müssen [RD98]. Dieses Vorgehen kann aber auch zu sehr großen semantischen Segmenten führen, so dass die Ersetzung von semantischen Segmenten bei vollem Cache zur Leerung eines bedeutenden Teils des Caches führen kann [DFJ+96]. Der Verzicht auf Vereinigung von semantischen Segmenten führt zu einem fein granular aufgebauten semantischen Cache, bläht aber den Index und damit den Aufwand bei der Anfragebearbeitung auf [RD98]. Die teilweise Vereinigung bietet einen Mittelweg aus den zwei vorgestellten Ansätzen, der ihre Vorteile vereinigt [RD98]. Alternativ zur teilweisen Vereinigung wird in [DFJ+96] für das Vorgehen ein heuristisches Verfahren vorgeschlagen, das vollständige Vereinigung ausführt, wenn mindestens eines der Fragmente kleiner als 1% der Cachegröße ist, ansonsten wird auf die Vereinigung semantischer Segmente verzichtet.

5.2 Partitionierung

Während bei naiven Ansätzen der Cache physisch partitioniert wird, indem die neuen semantischen Segmente physisch getrennt im semantischen Cache gespeichert werden, wird in [LLS99] ein Ansatz vorgestellt, der den semantischen Cache logisch partitioniert.

Hierbei werden die bestehenden semantischen Segmente physisch nicht geändert. In Abbildung 5.2 (Seite 54) bleiben Fragment 1 und Fragment 2 ein gemeinsames semantisches Segment wie vor dem Stellen der Anfrage und Fragment 3 wird als neues semantisches Segment gespeichert. So können unnötige Schreib- und Lesezugriffe vermieden werden. Um die zusätzlichen Informationen nicht zu verlieren, wird schließlich ein Baum aufgebaut, der die Beziehung zwischen den Fragmenten 1 und 2 sowie dem gemeinsamen semantischen Segment enthält. Dabei ist die Beschreibung des gesamten Segments im Wurzelknoten enthalten. Dieser verweist auf die physische Instanz des semantischen Segments. Die Kindknoten enthalten die Beschreibung der Fragmente 1 und 2, aber keinen Verweis auf die physische Instanz der Segmente. Die den Fragmenten 1 und 2 entsprechenden semantischen Segmente werden erst bei Bedarf aus dem semantischen Segment S_0 generiert.

Abbildung 5.3 (Seite 56) verdeutlicht diese Art der Partitionierung am Beispiel eines ohne Vereinigung von Segmenten generierten, physisch partitionierten und eines logisch partitionierten semantischen Caches. Zum Zeitpunkt t_0 befindet sich das semantische Segment S_0 im Cache. Logisch und physisch partitionierter Cache enthalten beide ei-

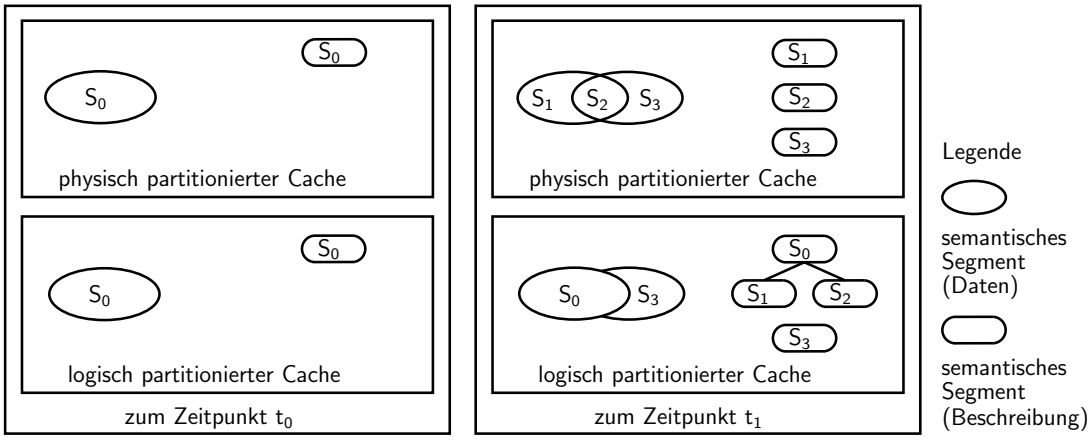

Abbildung 5.3: Physische und logische Partitionierung

ne Beschreibung und die Daten des Segments S_0. Zum Zeitpunkt t_1 wird eine Anfrage an den semantischen Cache gestellt, die nur zum Teil aus den Daten des semantischen Segments S_0 beantwortet werden kann. Daher werden weitere Daten vom Server abgerufen. Je nach Wahl der Granularität der Vereinigung der Datenfragmente werden bei der physischen Partitionierung unterschiedliche neue Fragmente gebildet. Alle Fragmente können zu einem semantischen Segment zusammengefasst werden (vollständige Vereinigung). Die gestellte Anfrage und die Inhalte von S_0 die nicht Teil des Anfrageergebnisses sind, können zu jeweils einem semantischen Segment unterteilt werden (teilweise Vereinigung). Alternativ werden, wie in Abbildung 5.3 (Seite 56) dargestellt, alle drei Fragmente einzeln abgespeichert (keine Vereinigung). Die Daten sind bei physischer und logischer Partitionierung im Segment S_3 enthalten. Die Beschreibung des semantischen Segments S_3 verweist in beiden Fällen direkt auf die Daten des Segments S_3. Neben diesem semantischen Segment enthält der physisch partitionierte Cache noch zwei weitere semantische Segmente S_1 und S_2. Dabei sind in S_2 die Daten der Schnittmenge der gestellten Anfrage und dem semantischen Segment S_0 enthalten und S_1 enthält die übrigen Tupel von S_0. Der logisch partitionierte Cache enthält zwei physische Segmente S_0 mit den Daten des Ausgangssegments und S_3 mit den zur Beantwortung der Anfrage zusätzlich vom Server angeforderten Daten. Logisch ist das Segment S_0 weiter unterteilt in die Segmente S_1 und S_2, diese Segmente werden erst bei Bedarf tatsächlich bestimmt. Die Beschreibung des Segments S_0 ist mit den Daten verknüpft.

Auf diese Weise kann die Anzahl der Lese- und Schreibzugriffe deutlich reduziert werden, da der bestehende Cacheinhalt bei Eingang neuer Anfragen nicht umgeschrieben werden muss. Außerdem kann die Performanz in einem stark fragmentierten Cache erhöht werden. Dafür muss zusätzlicher Speicher für die Beschreibungen der semantischen Segmente bereitgestellt werden und die Anfragebearbeitung wird komplizierter. Um keine zu großen Partitionierungsbäume entstehen zu lassen, kann ein Grenzwert (englisch: partitioning treshold) festgelegt werden. Wenn die Größe eines wiederbenutzten Fragments kleiner als dieser Grenzwert ist, wird auf die weitere Partitionierung verzichtet [LLS99].

5.3 Anfrageindex

Je nach Wahl der Cachepartitionierung sind zwei Arten von Anfrageindexen möglich. Bei der physischen Partitionierung besteht der Anfrageindex aus einer Liste von Beschreibungen einzelner semantischer Segmente. Dabei enthält jede Beschreibung den Namen des semantischen Segments, eine Beschreibung der gespeicherten Daten, Verweise auf die Speicherorte der durch das Segment verwalteten Datentupel und eine Zeitmarke, die den letzten Zugriff auf das entsprechende Segment wiedergibt [RD98]. Bei der logischen Partitionierung kann jedes Segment noch Verweise auf andere Segmente enthalten, die im aktuellen Segment enthalten sind.

Durch das Zulassen des Kreuzproduktes bei Anfragen tritt ein Problem auf, das bei den hier beschriebenen Anfrageindexen keine Beachtung findet. In diesem Fall müssen viele Daten redundant übertragen werden: Sei das Kreuzprodukt $Q = r(R_1) \times r(R_2)$ gegeben, n bezeichnet die Anzahl der Tupel in $r(R_1)$ und m bezeichnet die Anzahl der Tupel der Relation $r(R_2)$. Das Ergebnis der Anfrage Q besitzt für jedes Tupel der Relation $r(R_1)$ m Tupel. Da jedes Tupel von $r(R_1)$ mit jedem Tupel von $r(R_2)$ vereinigt wird. Folglich ist die Anzahl aller Tupel im Ergebnis der Anfrage Q $n \cdot m$. Verglichen mit der Anzahl der Tupel der Ausgangsrelationen $(n + m)$ ist dieses Ergebnis sehr groß. Daher empfiehlt es sich, um die Menge der übertragenen Daten zu verringern, vom Server nur einzelne Relationen zu übertragen und diese erst auf dem Client zu einem gesamten Anfrageergebnis zusammenzuführen. Hierbei können Selektionsprädikate genutzt werden, um das Anfrageergebnis weiter zu verkleinern. Selektionsbedingungen, die sich nur auf eine der beteiligten Relationen beziehen, können dieser direkt zugeordnet werden. Sei Q' die um die Attributselektion $\sigma_{x_i \theta x_j}$erweiterte Anfrage Q, wobei x_1 ein Attribut des Relationenschemas R_1 und x_j ein Attribut des Relationenschemas R_2 bezeichnet, so muss je nach Art des Operators θ unterschiedlich vorgegangen werden. Ist θ gleich $\neq$, so kann diese Ungleichung ignoriert werden, da zum Bestimmen aller Tupel im Ergebnis der Anfrage Q' alle Tupel der Relationen $r(R_1)$ und $r(R_2)$ benötigt werden. Ist θ einer der Operatoren in $\{<, \leq\}$, so kann die Menge der Tupel beschränkt werden, indem von der Relation $r(R_1)$ nur die Tupel bezogen werden, für die $x_i\ \theta\ max(x_j)$ gilt. Umgekehrt werden von der Relation $r(R_2)$ nur die Tupel benötigt, für welche $min(x_i)\ \theta\ x_j$ gilt. Besitzt die Attributselektion den Operator $=$, so können die Tupel, die von den Relationen $r(R_1)$ und $r(R_2)$ bezogen werden müssen, nicht einfach eingeschränkt werden. In diesem Fall muss das Ergebnis der Anfrage Q' auf dem Server bestimmt werden. Anschließend muss für jede der beteiligten Relationen eine Projektion des Anfrageergebnisses auf die Attribute der entsprechenden Relation durchgeführt werden. Auf diesen Teilergebnissen müssen schließlich alle Duplikate eliminiert werden.

Um dem Problem der Redundanz bei Kreuzprodukten zu begegnen, wird im Rahmen dieser Arbeit folgender Anfrageindex benutzt: Im semantischen Cache sind zum einen Relationensegmente, also Segmente, die sich nur auf eine Relation beziehen und zum anderen Kreuzproduktsegmente, die ein Kreuzprodukt mehrerer Relationensegmente beschreiben, zugelassen. Auf der Basis dieser zwei Segmenttypen wird nun eine hybride Form aus logischer und physischer Partitionierung als Anfrageindex gewählt. Kreuzpro-

duktsegmente enthalten nie eigene Tupel, sondern verweisen immer auf Relationensegmente. Relationensegmente hingegegen besitzen keine Verweise auf andere semantische Segmente.

5.4 Anfragebeschneidung

Das Problem der Anfragebearbeitung auf semantischen Caches besteht darin, bei gegebenem semantischem Cache und gegebener Anfrage die Anfrage in eine Filteranfrage und eine Kompensationsanfrage umzuschreiben. Damit kann die Filteranfrage auf den Daten des semantischen Caches ausgeführt werden und die Kompensationsanfrage die fehlenden Daten vom Server beziehen. Auf der Basis der in Kapitel 4 vorgestellten Algorithmen lassen sich hierfür unterschiedliche Vorgehensweisen finden. Zum einen kann die Filteranfrage durch Anwendung eines Algorithmus zur Anfragefaltung bestimmt werden. Dieser Algorithmus versucht jedem Teil der vom Benutzer gestellten Anfrage eine Ressource, also hier ein semantisches Segment, zuzuordnen. Die noch fehlenden Daten können durch Ausführung der Differenz aus Anfrage und Filteranfrage vom Server bezogen werden. Allerdings ist dieses Vorgehen nicht optimal, da die Filteranfrage zum Bestimmen der Differenz auch auf dem Server ausgeführt werden muss. Da die semantischen Segmente auf dem Server im Normalfall nicht in Form von materialisierten Sichten gespeichert sind, müssen diese zusätzlich bestimmt werden, um die Kompensationsanfrage auf dem Server abzuarbeiten. Daher wird in der Literatur [RDK03] ein alternatives Vorgehen vorgeschlagen.

Statt eine Anfrage mit dem gesamten Cache zu vergleichen, wird sie nur einzelnen semantischen Segmenten gegenübergestellt. Da jedes Segment und jede Anfrage eine Relation der Datenbank beschreibt, können sowohl semantische Segmente als auch Anfragen vereinfacht als Rechtecke visualisiert werden. Die Höhe des Rechtecks repräsentiert dabei die in der Relation enthaltenen Tupel und die Breite des Rechtecks repräsentiert die enthaltenen Attribute [GG97]. Die eingangs dieses Kapitels beschriebenen Beziehungen zwischen einer Anfrage und dem semantischen Cache können nun in eine Menge von Beziehungen zwischen der Anfrage und einzelnen semantischen Segmenten überführt werden. Die in Abbildung 5.1 (Seite 53) beschriebenen Szenarien können so auf die in Abbildung 5.4 (Seite 59) visualisierten Beziehungen abgebildet werden. Der Teil der Anfrage, der mit dem semantischen Segment überlappt, kann durch eine Filteranfrage vom semantischen Segment beantwortet werden, während der übrige Teil durch Ausführen einer Kompensationsanfrage auf der Datenbank beantwortet werden muss [RD98]. Das Aufteilen der Anfrage in eine Filteranfrage und eine Kompensationsanfrage wird als Anfragebeschneidung (englisch: query trimming) bezeichnet [KB96].

Um genauer auf die Anfragebearbeitung eingehen zu können, werden zunächst die verschiedenen Fälle der Anfragebeschneidung erklärt. Abbildung 5.4 (Seite 59) gibt über die verschiedenen Szenarien einen Überblick. Fall 1 stellt das vollständige Enthaltensein dar [RDK03]. In diesem Fall ist die Anfrage vollständig durch die im semantischen Segment enthaltenen Tupel beantwortbar. Die Filteranfrage ist in diesem Fall eine Anfrage

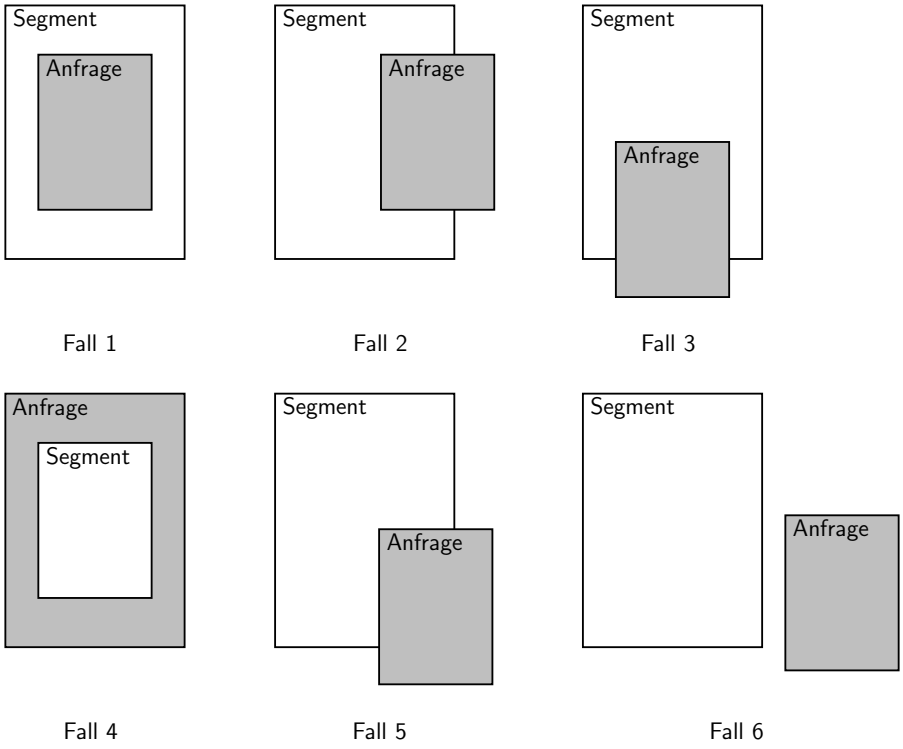

Abbildung 5.4: Beziehung zwischen Anfrage und semantischem Segment

auf dem semantischen Segment, während die Kompensationsanfrage leer ist. Im Fall 2 spricht man von vertikaler Partitionierung [RDK03]. In diesem Fall enthält der semantische Cache alle Tupel, die von der Anfrage abgefragt werden, allerdings müssen fehlende Spalten mit ihren Attributwerten vom Server bezogen werden. Enthält das semantische Segment die Schlüsselattribute, so können die fehlenden Daten in diesem Fall vom Server bezogen werden und lokal mit dem semantischen Segment verknüpft werden [GG97]. Sind die entsprechenden Primärschlüssel nicht vorhanden, so muss die Datenbankanfrage vollständig auf dem Server ausgeführt werden. Um dies zu verhindern, kann vor jeder Ausführung einer Kompensationsanfrage auf dem Server diese so erweitert werden, dass sie neben den gesuchten Attributen alle Schlüsselattribute mitabfragt [KB96]. Da in dieser Arbeit Relationen als Mengen und nicht als Multimengen betrachtet werden, kann die Menge der Projektionsattribute stets um einen Primärschlüssel erweitert werden. Dieser wird ermittelt, indem alle Attribute zu einem Schlüsselattribut zusammengefasst werden. In Fall 3 wird die horizontale Partitionierung dargestellt [RDK03]. In diesem Fall enthält das semantische Segment zwar alle gesuchten Attribute, allerdings fehlen einige Tupel, um die Anfrage vollständig beantworten zu können. Diese können durch Ausführung einer Kompensationsanfrage auf dem Server bezogen werden. Hierbei muss die Anfrage so angepasst werden, dass sie nur die Tupel enthält, die nicht im semantischen Cache enthalten sind. Hierfür wird aus dem Bedingungsteil der Anfrage C_Q und dem Bedingungsteil des semantischen Segments C_S ein neuer Bedingungsteil der Form $C_R = C_Q \wedge \neg C_S$ bestimmt [RDK03]. Die Kompensationsanfrage wird ermittelt, indem

der bestehende Bedingungsteil der Anfrage durch diesen Teil ersetzt wird. Fall 4 zeigt einen Sonderfall von Fall 5. Dieser Fall kann leicht auf den Fall 5 zurückgeführt werden. Man spricht hier von hybrider Partitionierung [RDK03]. Im Fall der hybriden Partitionierung wird die Anfrage zunächst horizontal getrimmt, anschließend wird auf der resultierenden Filteranfrage eine vertikale Partitionierung durchgeführt. In Fall 6 sind keine Daten des semantischen Caches für die Anfrage von Bedeutung. In diesem Fall kann die Anfrage direkt auf dem Server ausgeführt werden. Allerdings sollten in diesem Fall, wie bei der vertikalen Partitionierung, nach Möglichkeit die Primärschlüssel der beteiligten Relationen mit bestimmt werden [RDK03].

Um die Anfragebeschneidung auszuführen, muss entschieden werden, ob vollständiges Enthaltensein, horizontale, vertikale oder hybride Partitionierung vorliegt. Bei konjunktiven θ_{ALL}-Anfragen ist dies möglich, indem zunächst überprüft wird, ob die von der Anfrage abgefragten Attribute A_Q in den Attributen des semantischen Segments A_S enthalten sind. In diesem Fall ist die Anfrage Q vollständig im semantischen Segment S enthalten, wenn ein Algorithmus zur Überprüfung des Enthaltenseins wahr zurückgibt. Der Fall der horizontalen Partitionierung liegt vor, wenn die Anfrage $Q \wedge S$ erfüllbar ist. Wenn A_Q nicht in A_S enthalten ist, so liegt vertikale Partitionierung vor, wenn die Anfrageprädikate von Q die Anfrageprädikate von S implizieren. Ist dies nicht der Fall, aber ist $Q \wedge S$ erfüllbar, so liegt der Fall der hybriden Partitionierung vor. Wenn keiner der oben genannten Fälle eingetreten ist, enthält das semantische Segment keinen Teil der Anfrage [RDK03].

Während jedes einzelne semantische Segment nur einen Teil des Anfrageergebnisses enthalten kann, können mehrere Segmente zu einem größeren Teil kombiniert werden. Betrachtet man den gesamten semantischen Cache, wird die Anfrage nach und nach durch jedes semantische Segment getrimmt. Die so bestimmten Kompensationsanfragen werden dann zum Server geschickt und dort abgearbeitet [RDK03].

Da bei der Bearbeitung von hybrid partitionierten Anfragen zwei Kompensationsanfragen bestimmt werden, kann nicht linear vorgegangen werden. In diesem Fall kann ein Anfrageplanungsbaum (englisch: query plan tree) bestimmt werden [RDK03]. Dieser binäre Baum drückt die Beziehung zwischen jedem Teil der Anfrage und dem semantischen Segment, das einen Teil des Anfrageergebnisses enthält, aus. Jeder Knoten des Baums besitzt drei Attribute: Dazu zählen der Typ des Knotens, also ob der Knoten ein Blatt ist oder ob er durch Vereinigung oder Verbund seiner Kindknoten bestimmt werden kann, die Unteranfrage, die den Knoten beschreibt und das semantische Segment, das durch den Knoten bearbeitet wird. Ist das Attribut, das auf ein semantisches Segment verweist, nicht belegt, so muss die entsprechende Teilanfrage auf dem Server ausgeführt werden. Dieser Baum wird während der Anfragebeschneidung aufgebaut. Sobald der Baum vollständig ist, kann die Anfrage auf zwei Arten abgearbeitet werden. Zum einen können alle Knoten sequentiell berechnet werden, zum anderen ist eine parallele Ausführung möglich. Bei der sequentiellen Berechnung werden alle Knoten nacheinander abgearbeitet. Bei der parallelen Abarbeitung können gleichzeitig Anfragen auf dem Server und auf dem Client ausgeführt werden. Hierfür wird eine zusätzliche Datenstruktur

eingeführt. Diese Datenstruktur ist eine verkettete Liste, welche die Attribute Unteranfrage, Segment und Knoten im Anfrageplanungsbaum enthält. Jedes Element der Liste verweist auf das nächste abzuarbeitende Element.

Das Vorgehen bei der Anfragebeschneidung sei im Folgenden an einem Beispiel näher erläutert. Sei das Beispiel aus Abbildung 5.5 (Seite 61) gegeben.

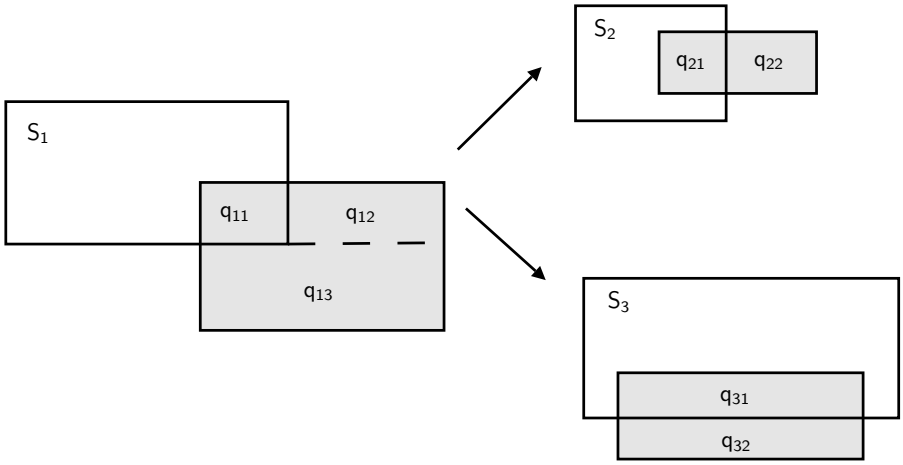

Abbildung 5.5: Anfragebearbeitung auf einem semantischen Cache

Die mit S_i bezeichneten Objekte seien semantische Segmente im semantischen Cache, während die q_{jk} Fragmente einer Anfrage an den Cache darstellen. Die ursprüngliche Anfrage ist die Vereinigung der Objekte q_{11}, q_{12} und q_{13}. Im ersten Schritt wird die ursprüngliche Anfrage mit dem Segment S_1 verglichen. Hierbei wird eine hybride Partitionierung festgestellt. Das Anfragefragment q_{11} kann direkt durch eine Filteranfrage auf dem semantischen Segment S_1 beantwortet werden. Die Kompensationsanfragen q_{12} und q_{13} werden anschließend mit weiteren Segmenten verglichen. Das Fragment q_{12} wird durch das Segment S_2 vertikal partitioniert. Während der Teil q_{21} vom semantischen Segment beantwortet werden kann, muss q_{22} weiter bearbeitet werden. Für q_{22} wird im semantischen Cache keine Entsprechung gefunden. Es wird daher als Kompensationsanfrage an den Server geschickt und dort beantwortet. Das Fragment q_{13} besitzt gemeinsame Tupel mit dem Segment S_3. In diesem Fall liegt horizontale Partitionierung vor. Das Anfragefragment q_{31} wird durch eine Filteranfrage vom semantischen Cache beantwortet, während q_{32} als Kompensationsanfrage vom Server beantwortet werden muss.

Während dieser Bearbeitungsschritte wurde der Anfrageplanungsbaum aus Abbildung 5.6 (Seite 62) angelegt. Die mit U bezeichneten Knoten im Baum, deuten an, dass die Kindknoten bei der Ermittlung des Anfrageergebnisses vereinigt werden müssen, während bei mit J bezeichneten Knoten ein Verbund über die Schlüsselattribute der Kindknoten durchgeführt werden muss.

Dieser Baum wird schließlich von den Blattknoten zum Wurzelknoten durchlaufen. Für jeden Blattknoten wird das Anfrageergebnis entweder auf einem semantischen Segment bestimmt (wie bei den Knoten (q_{11}, S_1), (q_{21}, S_2) und (q_{31}, S_3) in Abbildung 5.6,

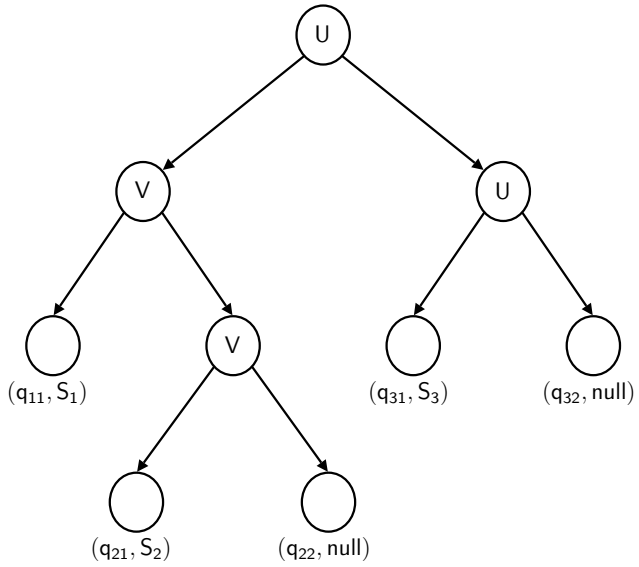

Abbildung 5.6: Anfrageplanungsbaum

Seite 62) oder, wenn der Zeiger auf das dazugehörige semantische Segment *null* ist, vom Server bezogen (wie für die Knoten $(q_{22}, null)$, $(q_{32}, null)$ in Abbildung 5.6, Seite 62). Bei Knoten, die keine Blattknoten sind, wird dann abhängig von der Knotenbezeichnung eine Vereinigung oder Differenz der Kindknoten durchgeführt. Der Wurzelknoten enthält schließlich das Ergebnis der Ausgangsanfrage, das in einem letzten Schritt an den Benutzer zurückgegeben werden kann.

5.5 Erweiterung um Mengenoperationen

In Abschnitt 4.2.6 wurde gezeigt, dass Elementardifferenzen azyklischer Anfragen und Vereinigungen von linken bzw. rechten Halbintervallanfragen auf Enthaltensein geprüft werden können. Um Vereinigungen derartiger konjunktiver Anfragen mit Hilfe eines semantischen Caches beantworten zu können, muss das Ergebnis jeder einzelnen konjunktiven Anfrage auf dem semantischen Cache beantwortet werden. Die Ergebnisse können dann in einem abschließenden Schritt zum gesuchten Anfrageergebnis vereinigt werden. Die gesonderte Speicherung von Vereinigungen konjunktiver Anfragen als eigene semantische Segmente im semantischen Cache ist nicht sinnvoll. In diesem Fall müssten Anfragen, die vom Cache beantwortet werden sollen, mit jeder konjunktiven Anfrage der Vereinigung verglichen werden. Der Aufwand für dieses Vorgehen ist gleich dem Aufwand beim Vergleich der einzelnen konjunktiven Anfragen mit dem Cache. Damit wäre dieses Vorgehen lediglich mit zusätzlichem Speicherbedarf verbunden. Da der Aufwand für das Beantworten einer Anfrage aus einer Vereinigung von n konjunktiven Anfragen n mal der Aufwand beim Vorgehen entsprechender konjunktiver Anfragen ist und da sich der grundlegende Algorithmus für das semantische Cachen in diesem Fall nicht ändert, wird

in der Evaluation von einer gesonderten Betrachtung abgesehen.

Die Berücksichtigung von Elementardifferenzen azyklischer Anfragen im semantischen Cache ist mit weiteren Problemen verbunden. Seien die Elementardifferenzen $Q' = Q'_1 - Q'_2$ und $Q = Q_1 - Q_2$, wobei Q'_1 und Q_1 konjunktive Anfragen und Q'_2 und Q_2 Vereinigungen konjunktiver Anfragen sind, gegeben. So ist die Elementardifferenz Q' genau dann in Q enthalten, wenn beide Elementardifferenzen dasselbe Relationenschema besitzen, $Q'_1 \sqsubseteq Q_1$ und $Q'_1 \cap Q_2 \sqsubseteq Q'_2$ gilt (siehe Abschnitt 4.2.6). Sei nun angenommen, dass im semantischen Cache sowie als Anfragen an den semantischen Cache Elementardifferenzen zugelassen sind. Sei die Elementardifferenz $Q = Q_1 - Q_2$ im semantischen Cache enthalten und sei die Anfrage $Q' = Q'_1 - Q'_2$ an den semantischen Cache gegeben. Schon die Voraussetzung, dass beide Elementardifferenzen dasselbe Schema besitzen müssen, ist eine gravierende Einschränkung. Diese Bedingung schließt vertikale und damit auch hybride Partitionierung aus, da über Anfragen, die unterschiedliche Relationenschemata besitzen, keine Aussage getroffen werden kann. Der Fall der horizontalen Partitionierung kann auftreten, wenn Q nicht alle Tupel besitzt, die von Q' benötigt werden. Diese können aber nicht einfach bestimmt werden, da die Tupel in Q enthalten sein müssen, aber nicht in Q' enthalten sein dürfen. Dies führt zu einer Kompensationsanfrage $Q' - Q$. Diese Anfrage entspricht $Q'_1 - Q'_2 - (Q_1 - Q_2)$, da Q'_2 und Q_2 Vereinigungen konjunktiver Anfragen sind, kann diese Anfrage nicht auf eine Elementardifferenz zurückgeführt werden. Die Anfrage müsste vom Server beantwortet werden, wobei der Aufwand zur Berechnung zweier Elementardifferenzen benötigt wird, unabhängig davon, ob weitere Tupel für die Anfrage Q' im semantischen Cache enthalten sind. Damit ist auch die horizontale Partitionierung zweier Vereinigungen von Elementardifferenzen nicht sinnvoll. Nur wenn Q' vollständig in Q enthalten oder äquivalent Q ist, kann die Anfrage beantwortet werden. Da dies den Nutzen des semantischen Cachens beträchtlich einschränkt, wird in der Evaluation auf die Betrachtung von Elementardifferenzen verzichtet. Das semantische Cachen von Elementardifferenzen ist nicht sinnvoll.

Kapitel 6

Evaluierung

Dieses Kapitel beschreibt die Evaluierung der in den vorangegangenen Kapiteln eingeführten Algorithmen. Hierfür wird zunächst kurz auf die Testumgebung eingegangen. Anschließend werden die Ergebnisse der Evaluation vorgestellt.

6.1 Testumgebung

Testrechner ist ein Fujitsu Lifebook E 7010 mit einem 1.7 GHz Pentium 4 Mobile Prozessor und 768 Megabyte RAM. Das genutzte Betriebsystem ist Linux.

Als Backend für die Implementierung dient das Datenbankmanagementsystem PostgreSQL Version 7.4.5[1]. Hierin wurden drei Datenbanken angelegt. Zwei enthalten jeweils zehn Relationen mit zwischen einem und fünf Attributen und zwischen einhundert und zweihundert Tupeln. Eine Datenbankinstanz dient als Server für die Evaluierung der Algorithmen auf dem Wertebereich *integer*. Alle Attribute in dieser Datenbank besitzen den PostgreSQL Datentyp *integer*. Die zweite Datenbankinstanz dient als Server für die Evaluierung der Algorithmen auf dem Wertebereich *real*. Alle Attribute in dieser Datenbank werden auf den PostgreSQL Datentyp *float(6)* abgebildet. Eine dritte Datenbank dient als Client. In dieser Datenbank werden sowohl die Inhalte des semantischen Caches, als auch Zwischenergebnisse für die Berechnung von Anfrageergebnissen gespeichert.

Die Algorithmen aus den vorangegangenen Kapiteln wurden auf der Java 2 Plattform Edition 5.0 implementiert. Hierfür wurde das entsprechende Java Development Kit von Sun Microsystems[2] genutzt.

Zunächst wurden die Algorithmen zum Test von Anfragen auf Erfüllbarkeit und auf Enthaltensein implementiert. Diese wurden gemäß der Beschreibungen in Kapitel 4 umgesetzt. Anschließend wurde der semantische Cache realisiert. Wie in Kapitel 5 beschrieben sind die Hauptfunktionen der Datenstrukturen des semantischen Caches das Verwalten der semantischen Segmente sowie die Anfragebearbeitung. Bei der Implemen-

[1] siehe http://www.postgresql.org
[2] siehe http://java.sun.com

tierung der Funktionen zur Verwaltung der semantischen Segmente wurde davon ausgegangen, dass sich die Daten der Serverdatenbank nicht ändern. Damit müssen auch keine Änderungen dieser Daten an den Client weitergegeben werden.

Das Beantworten von Anfragen, das im Rahmen dieser Arbeit untersucht werden soll, teilt sich in zwei Schritte. Zunächst wird die gestellte Anfrage beschnitten und ein Anfrageplanungsbaum aufgebaut. Dieser wird in einem zweiten Schritt abgearbeitet. Das Aufbauen des Anfrageplanungsbaums wurde vollständig in Java realisiert, während das Abarbeiten des Baums soweit wie möglich durch die Datenbank realisiert wird. Hierfür wird für jedes semantische Segment der Name der Relation, in welcher die zugehörigen Daten gespeichert sind, abgelegt. Werden zusätzliche Daten vom Server bezogen, so wird die Anfrage zunächst auf dem Server ausgeführt. Das Anfrageergebnis könnte dann mittels JDBC[3] in die Clientdatenbank kopiert werden. Dabei können Anfrageergebnisse allerdings so groß werden, dass sie nicht mehr im Hauptspeicher verwaltet werden können. Auch das Zuweisen des gesamten Hauptspeichers des Testrechners an den entsprechenden Java Prozess brachte hierbei keine Abhilfe. Daher werden Anfrageergebnisse auf dem Server weiter zerlegt und als Fragmente auf die Clientdatenbank übertragen, um dort wieder zum Gesamtergebnis kombiniert zu werden. Um keine Daten redundant zu übertragen wird hierbei eine Projektion des Anfrageergebnisses auf die Attribute je einer beteiligten Relation ausgeführt. Anschließend werden in diesem Ergebnis alle Duplikate eliminiert. Das so bestimmte Ergebnis wird dann mit JDBC in die Datenbank des Clients übertragen. Nachdem alle beteiligten Relationen so übermittelt wurden, wird die Anfrage auf den temporären Relationen ausgeführt und das Ergebnis in einer materialisierten Sicht gespeichert. Die Zwischenergebnisse bei der Bestimmung des Anfrageergebnisses werden als materialisierte Sichten auf diesen Ergebnissen gespeichert.

Der Quellcode dieser Anwendung kann von folgender Adresse heruntergeladen werden:

http://www.uni-magdeburg.de/schosser/sourcen.zip

6.2 Evaluationsergebnisse

Es wurde erwartet, dass das Erstellen des Anfrageplanungsbaums den größten Teil der Bearbeitungszeit benötigt. Das Abarbeiten dieses Baums sollte nur in Ausnahmefällen, wenn die Kompensationsanfrage sehr komplex wird (siehe Kapitel 5), ineffizient werden. Um dies zu belegen, wurden für jeden Anfragetyp einhundert zufällige Anfragen mit einem Kreuzprodukt über zwei Relationen, fünf Projektionsattributen und zwischen einem und 15 Selektionsbedingungen generiert. Aus diesen Anfragen wurden zehn ausgewählt. Sie bildeten den semantischen Cache, während die übrigen Anfragen mit ihrer Hilfe beantwortet wurden. Dieser Vorgang wurde mehrmals wiederholt, um die Ergebnisse unabhängig vom gewählten semantischen Cache zu machen. Abbildung 6.1 (Seite 67) zeigt die durchschnittlichen Bearbeitungszeiten.

In der Abbildung ist die Dauer der Erstellung des Anfrageplanungsbaums gegenüber

[3] siehe http://java.sun.com/products/jdbc/

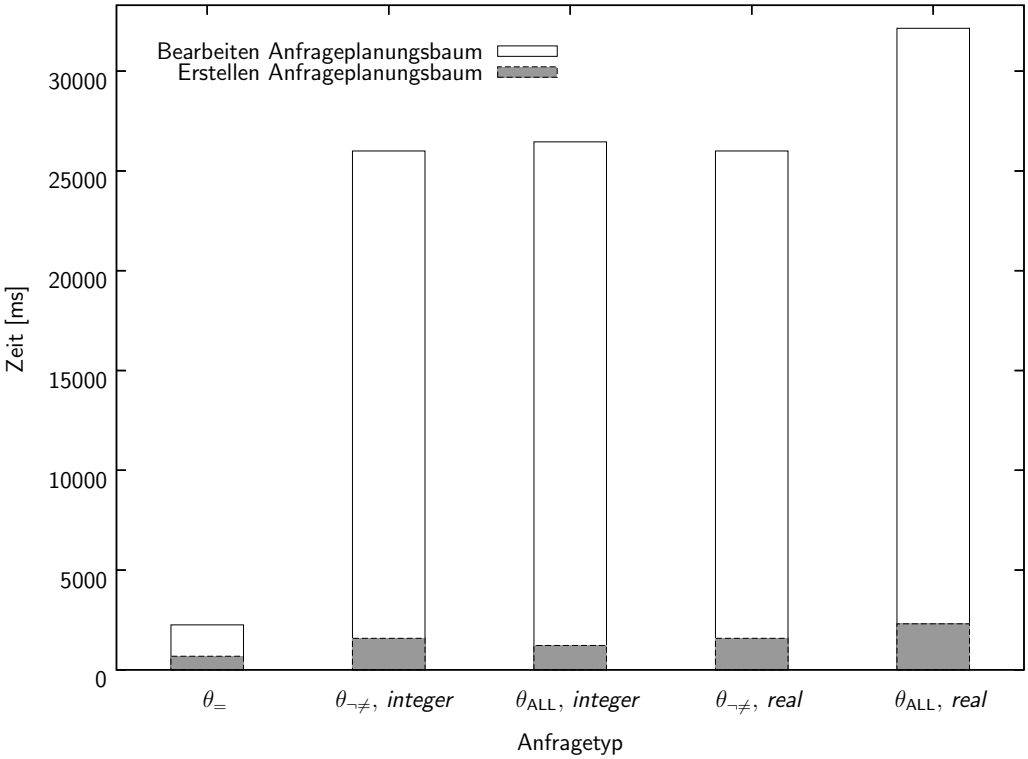

Abbildung 6.1: Dauer der Anfragebearbeitung bei konjunktiven Anfragen

der Zeit zur Berechnung des Anfrageergebnisses farblich abgehoben. Die Höhe der Balken gibt die Summe aus Erstellung und Bearbeitung des Baums wieder. Entgegen der Erwartungen dauert selbst für $\theta_{=}$-Anfragen die Abarbeitung des Anfrageplanungsbaums im Durchschnitt doppelt so lange, wie das Erstellen des Anfrageplanungsbaums. Auffällig ist, dass das Bestimmen des Anfrageplanungsbaums für konjunktive $\theta_{=}$-Anfragen, trotz des nicht polynomialen Erfüllbarkeitsalgorithmus (siehe Abschnitt 4.1.3), nicht länger dauert als bei den anderen Anfragetypen. Dies ist darauf zurückzuführen, dass in der Praxis nie alle 2^n möglichen Teilanfragen getestet werden müssen (siehe Abschnitt 4.1.3), sondern meist früher eine erfüllbare Teilanfrage gefunden wird.

Wünschenswert wäre eine nähere Untersuchung der Erstellungsdauer und der Abarbeitungsdauer der Anfrageplanungsbaums. Allerdings tritt hierbei das Problem auf, dass Struktur und Tiefe des Anfrageplanungsbaums unabhängig von der gestellten Anfrage sind. D.h. unabhängig von der Zahl der angefragten Tupel, der Länge der Anfrage und anderer Parameter, ist es möglich, dass das Anfrageergebnis vom ersten betrachteten Segment vollständig beantwortet werden kann. Ebenso ist es möglich, dass die Anfrage nur mit Hilfe eines Anfrageplanungsbaums zu beantworten ist, der auf alle semantischen Segmente und auf Daten auf dem Server verweist. Die Bearbeitungszeiten beider Szenarien sind sehr unterschiedlich. So wurden beispielsweise für einen semantischen Cache mit konjunktiven θ_{ALL}-Anfragen im Wertebereich *real* in einem Durchlauf zwei fast identische Testdatensätze gefunden. Beide bestehen aus Anfragen mit fünf Pro-

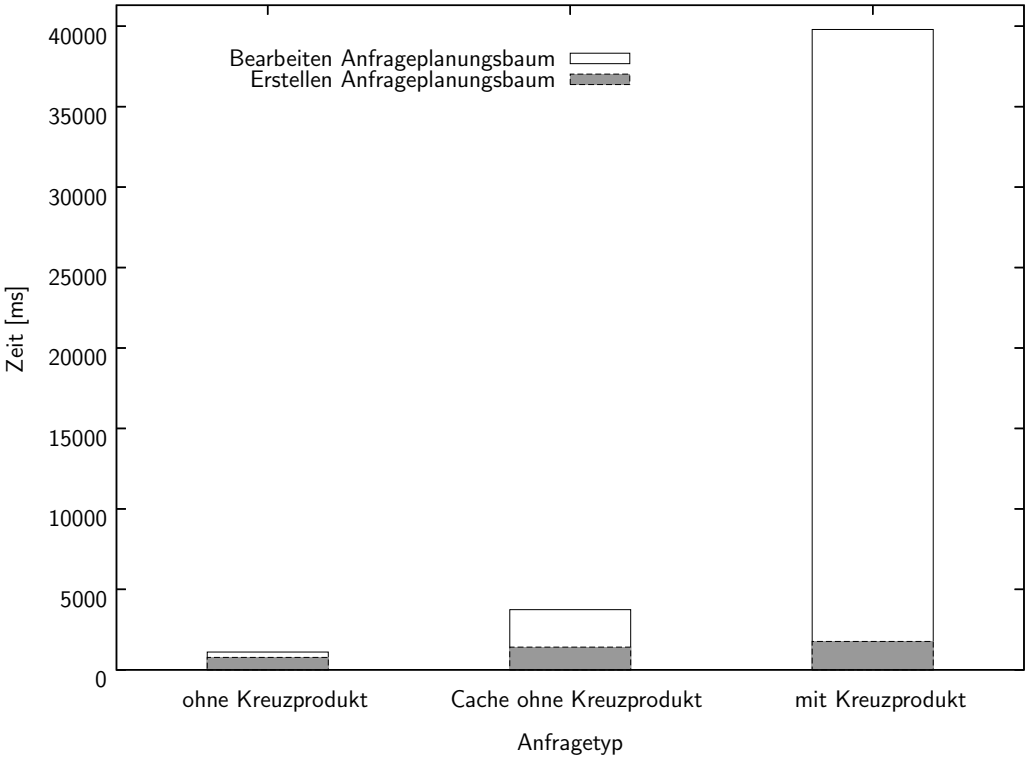

Abbildung 6.2: Dauer der Anfragebearbeitung nach Anfragetyp im Cache

jektionsattributen, zwei Selektionsbedingungen und zwei am Kreuzprodukt beteiligten Relationen. Die Länge der Anfrage ist im einen Fall 60 und im anderen Fall 68. Als Ergebnis lieferten beide Anfragen ca. 17 000 Tupel. Die Anfrage mit der Länge 68 konnte in nur 179 Sekunden beantwortet werden, während das Ergebnis der anderen Anfrage erst nach mehr als 24 Minuten zurückgegeben wurde. Dies ist darauf zurückzuführen, dass der Anfrageplanungsbaum der einen Anfrage 60 Knoten besaß, während die andere Anfrage zu einem Planungsbaum mit nur 13 Knoten führte. Mit steigender Größe des Anfrageplanungsbaums treten zwei Probleme auf: Zum einen müssen bei einem kleinen Anfrageplanungsbaum weniger Zwischenergebnisse berechnet werden, zum anderen werden die Kompensationsanfragen immer komplexer, je öfter sie partitioniert werden. Somit steigt auch der Zeitbedarf bei der Berechnung des Ergebnisses einer Kompensationsanfrage auf der Serverdatenbank. Es ist folglich unmöglich, Abhängigkeiten zwischen der Anfrage und der Laufzeit einzelner Bearbeitungsschritte zu finden.

Allerdings deutet Abbildung 6.1 (Seite 67) an, dass die Performanz des semantischen Caches von der Art der zugelassenen Anfragen abhängt. Konjunktive $\theta_{=}$-Anfragen wurden hier wesentlich schneller beantwortet, als vergleichbare andere Anfragen. Da bei den Testdaten, die dieser Abbildung zu Grunde liegen, die Mächtigkeit der Anfrage mit der Mächtigkeit der in semantischen Segmenten abgelegten Anfragen übereinstimmt, kann die Ursache für den Performanzunterschied in der Mächtigkeit der Anfragen im Cache gesucht werden. Dies erklärt auch, warum $\theta_{=}$-Anfragen schneller als andere konjunktive

Anfragen abgearbeitet werden: Derartige Anfragen betrachten im Normalfall deutlich weniger Tupel als andere konjunktive Anfragen. Ist beispielsweise eine $\theta_=$-Anfrage mit einer Selektionsbedingung $A_1 = 5$ gegeben, so liefert die Anfrage durchschnittlich weniger Tupel als eine Anfrage mit Selektionsbedingung $A_1 \geq 5$. Da diese sowohl alle Tupel mit $A_1 = 5$ und zusätzliche Tupel mit $A_1 > 5$ beschreibt. Entsprechend sind konjunktive $\theta_=$-Anfragen seltener in anderen Anfragen enthalten, als andere konjunktive Anfragen. Analog führt eine Attributselektion mit Gleichoperator zu einem kleineren Ergebnis als eine Attributselektion mit anderem Vergleichsoperator.

Abbildung 6.2 (Seite 68) beschreibt, welchen Einfluss die Mächtigkeit der im semantischen Cache gespeicherten Anfragen auf das Anfrageergebnis haben. Hierfür wurde ein semantischer Cache angelegt, der nur Anfragen enthielt, die keine Kreuzprodukte besaßen. Auf diesem semantischen Cache wurden Anfragen ohne Kreuzprodukt (siehe „*ohne Kreuzprodukt*“ in Abbildung 6.2, Seite 68) und Anfragen mit Kreuzprodukt (siehe „*Cache ohne Kreuzprodukt*“ in Abbildung 6.2, Seite 68) ausgeführt. Dieselben Anfragen mit Kreuzprodukt wurden schließlich auf einem semantischen Cache, der Anfragen mit Kreuzprodukten enthielt abgearbeitet (siehe „*mit Kreuzprodukt* “ in Abbildung 6.2, Seite 68). Dies wurde für alle Typen von konjunktiven Anfragen wiederholt und die Ergebnisse schließlich aggregiert. Während die Dauer des Erstellens des Anfrageplanungsbaums in allen drei Fällen ähnlich ist, bestehen große Unterschiede bei der Laufzeit der Abarbeitung des Anfrageplanungsbaums. Während diese im Fall von Anfragen ohne Kreuzprodukt auf einem semantischen Cache ohne Kreuzprodukt vernachlässigbar gering ist, steigt sie bei Anfragen mit Kreuzprodukt auf demselben Cache auf ähnliche Höhe, wie die Laufzeit der Anfrageplanungsbaumgenerierung. Jedoch ist sie in beiden Fällen wesentlich geringer als im Fall des semantischen Caches mit Segmenten mit konjunktiven Anfragen.

Daher wird im Folgenden weiter auf die Dauer der Bearbeitung des Anfrageplanungsbaums eingegangen. Dieser Schritt dauert für Anfragen mit Kreuzprodukt auf einem semantischen Cache mit Kreuzprodukten im Durchschnitt ca. 38 Sekunden (siehe Abbildung 6.2, Seite 68). Ein vernachlässigbar geringer Teil (285 Millisekunden) wird davon für das Bestimmen von Tupeln aus dem semantischen Cache aufgebraucht, während der Großteil der Bearbeitung des Anfrageplanungsbaums durch das Beziehen von Informationen vom Server benötigt wird (ca. 28 Sekunden). Für das Bearbeiten von Knoten des Anfrageplanungsbaums, die über Verbund bestimmt werden, werden ca. 6 Sekunden benötigt, während entsprechende Knoten für Vereinigung in nur 4 Sekunden ermittelt werden. Den größten Einfluss auf die Performanz hat damit das Bestimmen von Daten auf der Serverdatenbank. Dieser Effekt hat zwei Ursachen: Zum einen müssen die einzelnen Tupel mit der Hilfe von JDBC von der einen Datenbank zur anderen transferiert werden, zum anderen sind die Anfragen, die auf dem Server ausgeführt werden müssen, komplexer, als die im semantischen Cache. Daher sollten im semantischen Cache nur Anfragen auf je einer Relation zugelassen werden.

Abschließend lässt sich festhalten, dass sich beliebige Anfragen bestehend aus Selektion und Projektion gut durch einen semantischen Cache verwalten lassen. Sobald

jedoch Kreuzprodukte zugelassen werden, steigt die Bearbeitungszeit. Diese kann reduziert werden, indem im semantischen Cache nur Anfragen auf einzelnen Relationen zugelassen werden. Ausnahme bildet hierbei die Menge der konjunktiven $\theta_{=}$-Anfragen. Diese lassen sich auch effizient beantworten.

Kapitel 7

Zusammenfassung und Ausblick

Dieses Kapitel fasst die vorangegangenen Kapitel zusammen und gibt einige Ansatzpunkte für weitere Untersuchungen.

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde der Einfluss der Mächtigkeit einer relationalen Anfragesprache auf die Auswertung von Anfrageindexen am Beispiel des semantischen Cachens untersucht. Hierfür wurde mit der relationalen Algebra eine relational vollständige Anfragesprache vorgestellt. Diese wurde in den sicheren Bereichskalkül überführt. Daneben wurden weitere graphische Repräsentationen für Datenbankanfragen eingeführt.

Auf der Basis dieser Repräsentationen wurden dann Algorithmen zur Überprüfung von Anfragen auf Erfüllbarkeit und Enthaltensein vorgestellt. Beide Klassen von Algorithmen werden beim semantischen Cachen für die Beantwortung von Anfragen benötigt. Anschließend wurden Klassen von Anfragen identifiziert, die mit Hilfe der vorgestellten Algorithmen in polynomialer Zeit auf Enthaltensein geprüft werden können. Hier konnten einige Typen von Anfragen für die weitere Betrachtung ausgeschlossen werden: Diese Anfragen konnten nicht effizient auf Enthaltensein geprüft werden. Da das Enthaltenseinproblem ein Teilproblem der Auswertung von Anfrageindexen beim semantischen Cachen ist, können diese Anfragen auch nicht effizient vom semantischen Cache verwaltet werden.

Danach wurden die Grundlagen des semantischen Cachens eingeführt und auf das Beantworten von Anfragen mit semantischen Segmenten eingegangen. Hierbei konnte gezeigt werden, dass Anfragesprachen, die Mengenoperationen zulassen, nicht für die Bearbeitung in einem semantischen Cache geeignet sind.

Abschließend wurden die vorgestellten Algorithmen evaluiert. Hierfür wurden die in den vorangegangenen Kapiteln bestimmten Klassen von Anfragen näher untersucht. Dabei konnte gezeigt werden, dass die Beantwortung von Anfragen mit Kreuzprodukten auf einem semantischen Cache deutlich länger dauert, wenn im zu Grunde liegenden

Cache auch Anfragen mit Kreuzprodukten zugelassen sind.

7.2 Ausblick

Ein Problem der Evaluierung ist, dass die gemessenen Antwortzeiten nur zu einem Teil der Laufzeit der Erfüllbarkeits- und Enthaltenseinalgorithmen geschuldet sind. Dieser Teil ist in allen untersuchten Fällen ähnlich. Allerdings schwankt die Laufzeit der Anfragebeantwortung in Abhängigkeit von der Mächtigkeit der gewählten Anfragesprache sehr. Auf diesen Punkt hat die hier entwickelte Implementierung nur wenig Einfluss. Hier sollte die Komplexität der Kompensationsanfragen weiter betrachtet werden. Eventuell könnte die Laufzeit des Algorithmus verbessert werden, indem ein geeignetes Maß für die Komplexität von Kompensationsanfragen ermittelt wird. Mit Hilfe dieses Maßes könnte dann entschieden werden, ob eine Kompensationsanfrage weiter unterteilt oder vom Server beantwortet werden sollte. Diese Entscheidung könnte unabhängig von den weiteren im semantischen Cache enthaltenen Segmenten getroffen werden.

In [RDK03] wurde ein formales Modell eingeführt, das semantische Segmente auf eine Menge von Metadaten und einen Verweis auf die beschriebenen Daten abbildet. Allerdings wurde in diesem Modell von einer weniger mächtigen Anfragesprache als den hier betrachteten ausgegangen. Dieses Modell könnte so erweitert werden, dass es die hier beschriebenen Anfragen auch abbilden kann.

In dieser Arbeit wurden nur Anfragen betrachtet, die ausschließlich Attribute einer Domäne besitzen. So wurden keine Relationen über Attribute unterschiedlicher Domänen zugelassen. Da derartige Anfragen in der Praxis eher die Ausnahme bilden dürften, muss untersucht werden, welchen Einfluss die Nutzung unterschiedlicher Datentypen in einer Anfrage auf die beschriebenen Algorithmen hat.

Wird der semantische Cache nicht auf demselben Rechner wie die gecachte Datenbank gehalten, so treten bei der Übertragung der Daten vom Server zum Client weitere Verzögerungen auf. Dieses Problem blieb in dieser Arbeit unberücksichtigt. Auch auf Szenarien, in welchen zeitweise keine Verbindung zum Server besteht, wurden nicht eingegangen. Hier könnte beispielsweise untersucht werden, wann es lohnt, Daten aus dem semantischen Cache an den Benutzer zu übergeben, ohne auf die Antwort vom Server zu warten.

Literaturverzeichnis

[ASU79a] Aho, A. V.; Sagiv, Y.; Ullman, J. D.: Efficient optimization of a class of relational expressions. *ACM Transactions on Database Systems*, Band 4, Nr. 4, S. 435–454, Dezember 1979.

[ASU79b] Aho, A. V.; Sagiv, Y.; Ullman, J. D.: Equivalences among relational expressions. *SIAM Journal on Computing*, Band 8, Nr. 2, S. 218–246, Mai 1979.

[Bod93] Bodlaender, H. L.: A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, S. 226–234, Mai 1993.

[CM77] Chandra, A. K.; Merlin, P. M.: Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, S. 77–90, Mai 1977.

[Cod72] Codd, E. F.: Relational completeness of data base sublanguages. In *Proceedings of the Courant Computer Science Symposium 6*, S. 65–98, Mai 1972.

[Cod91] Codd, E. F.: *The Relational Model for Database Management; Version 2*. Addison-Wesley Publishing Company, Massachusetts, 1991.

[CR97] Chekuri, C.; Rajarman, A.: Conjunctive query containment revisited. In *Proceedings of the 6th International Conference on Database Theory*, S. 56–70, Januar 1997.

[CR00] Chekuri, C.; Rajaraman, A.: Conjunctive query containment revisited. *Theoretical Computer Science*, Band 239, Nr. 2, S. 211–229, Mai 2000.

[CSL98] Chan, B. Y.; Si, A.; Leong, H. V.: Cache management for mobile databases: Design and evaluation. In *Proceedings of the Fourteenth International Conference on Data Engineering*, S. 54–63, Februar 1998.

[Dat00] Date, C. J.: *An Introduction to Database Systems*. Addison-Wesley Publishing Company, Massachusetts, 7. Auflage, 2000.

[Ded01] Dedo, D.: Mobile devices in the enterprise. *http://www.microsoft.com/windowsmobile/resources/whitepapers/devicesinenterprise.mspx*, Oktober 2001.

[DFJ+96] Dar, S.; Franklin, M. J.; Jónsson, B. T.; Srivastava, D.; Tan, M.: Semantic data caching and replacement. In *Proceedings of the 22th International Conference on Very Large Data Bases*, S. 330–341, September 1996.

[DRSN98] Deshpande, P. M.; Ramasamy, K.; Shukla, A.; Naughton, J. F.: Caching multidimensional queries using chunks. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, S. 259–270, Juni 1998.

[Fed01] Federrath, H.: Mobile computing. In Bäumler, H.; Breinlinger, A.; Schrader, H.-H. (Hrsg.): *Datenschutz von A-Z*. Luchterhand, Neuwied, 2001.

[GG97] Godfrey, P.; Gryz, J.: Semantic query caching for heterogeneous databases. In *Proceedings of the 4th International Workshop: Knowledge Representation Meets Databases*, S. 6.1–6.6, August 1997.

[GKK03] Graumann, S.; Köhne, B.; Kahre, S.: Monitoring Informationswirtschaft - 6. Faktenbericht 2003. Technischer Bericht Nr. 6, NFO Infratest GmbH & Co. KG, München, März 2003.

[GLS99] Gottlob, G.; Leone, N.; Scarcello, F.: Hypertree decompositions and tractable queries. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, S. 21–32, Mai/Juni 1999.

[GSW96a] Guo, S.; Sun, W.; Weiss, M. A.: On satisfiability, equivalence, and implication problems involving conjunctive queries in database systems. *IEEE Transactions on Knowledge and Data Engineering*, Band 8, Nr. 4, S. 604–616, August 1996.

[GSW96b] Guo, S.; Sun, W.; Weiss, M. A.: Solving satisfiability and implication problems in database systems. *ACM Transactions on Database Systems*, Band 21, Nr. 2, S. 270–293, Juni 1996.

[Hal01] Halevy, A. Y.: Answering queries using views: A survey. *The International Journal on Very Large Data Bases*, Band 10, Nr. 4, S. 270–294, Dezember 2001.

[HBB+03] Hilty, L.; Behrendt, S.; Binswanger, M.; Bruinink, A.; Erdmann, L.; Fröhlich, J.; Köhler, A.; Kuster, N.; Som, C.; Würtemberger, F.: Das Vorsorgeprinzip in der Informationsgesellschaft. Technischer Bericht, TA-SWISS, 2003.

[HS00] Heuer, A.; Saake, G.: *Datenbanken: Konzepte und Sprachen*. MITP GmbH, Bonn, 2. Auflage, 2000.

[IK97] Imielinski, T.; Korth, H. F.: *Mobile Computing*. Kluwer Academic Publishers, Boston, 1997.

[KB96] Keller, A. M.; Basu, J.: A predicate-based caching scheme for client-server database architectures. *The International Journal on Very Large Data Bases*, Band 5, Nr. 1, S. 35–47, Januar 1996.

[Klu88] Klug, A.: On conjunctive queries containing inequalities. *Jounal of the ACM*, Band 35, Nr. 1, S. 146–160, Januar 1988.

[LC01] Lee, D.; Chu, W. W.: Towards intelligent semantic caching for web sources. *Journal of Intelligent Information Systems*, Band 17, Nr. 1, S. 23–45, November 2001.

[LLS99] Lee, K. C. K.; Leong, H. V.; Si, A.: Semantic query caching in a mobile environment. *ACM SIGMOBILE Mobile Computing and Communications Review*, Band 3, Nr. 2, S. 28–36, April 1999.

[LY85] Larson, P.-A.; Yang, H. Z.: Computing queries from derived relations. In *Proceedings of 11th International Conference on Very Large Data Bases*, S. 259–269, August 1985.

[OW90] Ottmnann, T.; Widmayer, P.: *Algorithmen und Datenstrukturen*. BI Wissenschaftsverlag, Mannheim, 1990.

[Qia96] Qian, X.: Query folding. In *Proceedings of the Twelfth International Conference on Data Engineering*, S. 48–55, Februar/März 1996.

[RD98] Ren, Q.; Dunham, M. H.: Semantic caching and query processing. Technischer Bericht Nr. 98-CSE-04, Southern Methodist University, Department of Computer Science and Engineering, Mai 1998.

[RDK03] Ren, Q.; Dunham, M. H.; Kumar, V.: Semantic caching and query processing. *IEEE Transactions on Knowledge and Data Engineering*, Band 15, Nr. 1, S. 192–210, Januar/Februar 2003.

[Rei99] Reischuk, K. R.: *Komplexitätstheorie Band I: Grundlagen*. B.G. Teubner, Stuttgart, 2. Auflage, 1999.

[RH80] Rosenkrantz, D. J.; Hunt, H. B.: Processing conjunctive predicates and queries. In *Proceedings of the Sixth International Conference on Very Large Databases*, S. 64–72, Oktober 1980.

[SH99] Saake, G.; Heuer, A.: *Datenbanken: Implementierungstechniken*. MITP-Verlag GmbH, Bonn, 1999.

[SKN89] Sun, X.-H.; Kamel, N. N.; Ni, L. M.: Processing implication on queries. *IEEE Transactions on Software Engineering*, S. 1168–1175, Oktober 1989.

[Sol79] Solomon, M. K.: Some properties of relational expressions. In *Proceedings of the 17th annual Southeast regional conference*, S. 111–116, April 1979.

[Sta03] Statistische Ämter des Bundes und der Länder: Gebiet und Bevölkerung - Haushalte. *http://www.statistik-portal.de/Statistik-Portal/de_jb01_jahrtab4.asp*, 03. Juni 2003.

[SU02] Schaefer, M.; Umans, C.: Completeness in the polynomial-time hierarchy: A compendium. *Sigact News*, Band 33, Nr. 3, September 2002.

[SY80] Sagiv, Y.; Yannakakis, M.: Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, Band 27, Nr. 4, S. 633–655, Oktober 1980.

[TY84] Tarjan, R. E.; Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *Siam Journal on Computing*, Band 13, S. 566 – 579, August 1984.

[Ull88] Ullman, J. D.: *Principles of Database and Knowledge-Base Systems: Volume I: Classical Database Systems*. Computer Science Press, 1988.

[vdM97] Meyden, R. v. d.: The complexity of querying indefinite data about linearly ordered domains. *Journal of Computer and System Sciences*, Band 54, S. 113–135, Februar 1997.

[Weg03] Wegener, I.: *Komplexitätstheorie - Grenzen der Effizienz von Algorithmen*. Springer-Verlag, Berlin, 2003.

[YL87] Yang, H. Z.; Larson, P. A.: Query transformation for PSJ-queries. In *Proceedings of the 13th International VLDB Conference*, S. 245–254, September 1987.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 25. Oktober 2004

Stephan Schosser