Einführung in die Datenbanktheorie

Vorlesung im Wintersemester

Prof. Dr. Nicole Schweikardt

Lehrstuhl Logik in der Informatik
Institut für Informatik
Humboldt-Universität zu Berlin

Version vom 17. Februar 2021

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Einführung ins Thema

Folie 1

„Was“ statt „Wie“ — am Beispiel von „Tiramisu“

Tiramisu — Deklarativ
Aus Eigelb, Mascarpone und in Likör und Kaffee getränkten Biskuits hergestellte cremige Süßspeise

(aus: DUDEN, Fremdwörterbuch, 6. Auflage)

Tiramisu — Operationell
1/4 l Milch mit 2 EL Kakao und 2 EL Zucker aufkochen. 1/4 l starken Kaffee und 4 EL Amaretto dazugeben.

5 Eigelb mit 75 g Zucker weißschaumig rühren, dann 500 g Mascarpone dazumischen.

ca 200 g Löffelbiskuit.

Eine Lage Löffelbiskuit in eine Auflaufform legen, mit der Flüssigkeit tränken und mit der Creme überziehen. Dann wieder Löffelbiskuit darauflegen, mit der restlichen Flüssigkeit tränken und mit der restlichen Creme überziehen.

Über Nacht im Kühlschrank durchziehen lassen und vor dem Servieren mit Kakao bestäuben.

(aus: Gisela Schweikardt, handschriftliche Kochrezepte)

Folie 2

Der große Traum der Informatik

Imperative Vorgehensweise:
Beschreibung, wie das gewünschte Ergebnis erzeugt wird ....*„Wie“*

Deklarative Vorgehensweise:
Beschreibung der Eigenschaften des gewünschten Ergebnisses ....*„Was“*

Traum der Informatik:
Möglichst wenig „wie“, möglichst viel „was“

D.h.: Automatische Generierung eines Ergebnisses aus seiner Spezifikation

Realität:
Software-Entwicklung: Generierungs-Tools

Programmiersprachen: Logik-Programmierung, insbes. Prolog

ABER: Imperativer Ansatz überwiegt in der Praxis

Datenbanken: *Deklarative Anfragesprache ist Industriestandard! (SQL)*

Folie 3

Datenbanksysteme

Datenbank (DB)

- zu speichernde Daten
- Beschreibung der gespeicherten Daten (Metadaten)

Datenbankmanagementsystem (DBMS)

- Softwarekomponente zum Zugriff auf die Datenbank
- Eigenschaften / Kennzeichen:
  - *Sichere* Verwaltung von Daten: langlebig, große Menge von Daten ... im Sekundärspeicher
  - *Effizienter* Zugriff auf (große) Datenmengen in der DB

Datenbanksystem (DBS)

- DB + DBMS

Folie 4

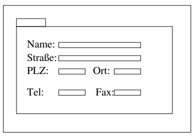
Wünschenswerte Eigenschaften eines DBS

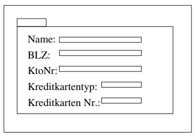
- *Unterstützung eines Datenmodells:* „Darstellung“ der Daten für den Zugriff
- *Bereitstellung einer DB-Sprache:*
  - zur Datendefinition: Data Definition Language (DDL)
  - zur Datenmanipulation und zum Datenzugriff:
    Data Manipulation Language (DML)
- *Zugangskontrolle:* Wer darf wann auf welche Daten zugreifen bzw. verändern?
- *Datenintegrität:* Wahrung der Datenkonsistenz und -korrektheit
- *Robustheit:* Wahrung eines konsistenten Zustands der DB trotz ...
  - Datenverlusts bei Systemfehlern (CPU Fehler, Plattencrash)
  - fehlerhafter Beendigung eines DB-Programms oder einer DB-Interaktion
  - Verletzung der Datenintegrität oder von Zugriffsrechten
- *Zugriffskoordination bei mehreren DB-Benutzern:*
  Synchronisation, korrekter Zugriff, korrektes Ergebnis bzw. korrekter DB-Zustand
- *Effizienter Datenzugriff und Datenmanipulation:*
  schnelle Bearbeitung der Benutzeranfragen

Folie 5

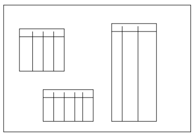
3-Schichten-Modell

Externe Schicht: ***Verschiedene Ansichten der Daten***

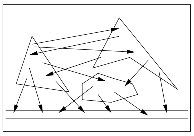
Ansicht 1: 

Ansicht 2: 

Logische Schicht: ***Daten = Tabellen***



Physische Schicht: ***Datenstrukturen, Speicherorganisation***



Folie 6

Anfragesprachen

Wünschenswerte Eigenschaften:

- *Möglichst viel „Was“*
  Beschreiben der Eigenschaften des gewünschten Ergebnisses (deklarativ)
- *Möglichst wenig „Wie“*
  Beschreiben, wie das gewünschte Ergebnis erzeugt werden soll (operationell)
- *Möglichst unabhängig von den Details der Datenorganisation*
  Bezug auf logische Schicht oder externe Schicht, <u>nicht</u> auf physische Schicht

Der Preis der Bequemlichkeit und Unabhängigkeit:

- deklarative Anfragen verschieben die Arbeit vom Benutzer zum System
- System muss Anfrage in eine Folge von Operationen umwandeln

$\rightsquigarrow$ *Gefahr der Ineffizienz*

$\rightsquigarrow$ *Geht das überhaupt? Was ist die Auswertungskomplexität?*

- Andererseits: System hat große Freiheit in der Umsetzung, da kein Lösungsweg vorgeschrieben ist

$\rightsquigarrow$ *Potenzial für Optimierung*

Folie 7

Hauptthema dieser Vorlesung: Anfragesprachen

Typische Fragestellungen für diese Vorlesung:

- Wie lassen sich deklarative Anfragen in ausführbare Operationen umsetzen?

$\leadsto$ *Äquivalenz von „Kalkül“ und „Algebra“*

- Welche Anfragen können in einer Anfragesprache gestellt werden, welche nicht?

$\leadsto$ *Ausdrucksstärke von Anfragesprachen*

- Wie aufwändig ist die Auswertung von Anfragen prinzipiell?

$\leadsto$ *Auswertungskomplexität*

- Wie lässt sich eine gegebene Anfrage möglichst <u>effizient</u> auswerten?

$\leadsto$ *Anfrageoptimierung, statische Analyse (Erfüllbarkeit, Äquivalenz, ...)*

Folie 8

Inhaltsübersicht

1. *Einleitung*
2. *Das Relationale Modell*
   - Datenmodell
   - Anfragen
   - Datenkomplexität und kombinierte Komplexität
3. *Konjunktive Anfragen*
   - Deskriptiver Ansatz: regelbasiert, graphisch und logikbasiert
   - Auswertungskomplexität
   - Algebraischer Ansatz: SPC-Algebra und SPJR-Algebra
   - Anfrageminimierung, statische Analyse und der Homomorphismus-Satz
   - Azyklische Anfragen

- Mengen-Semantik vs. Multimengen-Semantik

4. *Anfragesprachen mit Rekursion — Datalog*
   - Syntax und Semantik
   - Auswertung von Datalog-Anfragen, Statische Analyse
   - Datalog mit Negation

Folie 9

5. *Funktionale Abhängigkeiten*
   - „The Chase“
   - Anfrage-Optimierung unter Berücksichtigung funktionaler Abhängigkeiten
   - der Armstrong-Kalkül
6. *Relationale Algebra*
   - Definition und Beispiele
   - Anfrageauswertung und Heuristische Optimierung
   - Das Semijoin-Fragment der Relationalen Algebra
7. *Relationenkalkül*
   - Syntax und Semantik
   - Bereichsunabhängige Anfragen
   - Äquivalenz zur Relationalen Algebra
   - Auswertungskomplexität
   - Statische Analyse

Folie 10

Literatur

[AHV] Abiteboul, Hull, Vianu: Foundations of Databases, *Addison-Wesley, 1995*

[M] Maier: *The Theory of Relational Databases*, Computer Science Press, 1983

[AD] Atzeni, de Antonellis: *Relational Database Theory*, Benjamin Cummings, 1992

[SSS] Schweikardt, Schwentick, Segoufin: Database Theory: Query Languages. *Kapitel 19 in* Algorithms and Theory of Computation Handbook, *2nd Edition, Volume 2: Special Topics and Techniques, Mikhail J. Atallah and Marina Blanton (editors), CRC Press, 2009.*

1.2 Organisatorisches

Folie 11

Organisatorisches

- *Webseite der Vorlesung:*
  `www2.informatik.hu-berlin.de/logik/lehre/`

1.3 Grundlegende Schreibweisen

Folie 12

- $\mathbb{N} := \{0, 1, 2, 3, \ldots\}$
- $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$
- Für $n \in \mathbb{N}_{\geqslant 1}$ ist $[n] := \{1, \ldots, n\} = \{x \in \mathbb{N} \ : \ 1 \leqslant x \leqslant n\}$.
- Die Potenzmenge einer Menge M bezeichnen wir mit $\mathcal{P}(M)$.
  D.h.: $\mathcal{P}(M) = \{X \ : \ X \subseteq M\}$.
- Ist M eine Menge, so schreiben wir $X \subseteq_e M$ um auszudrücken, dass X eine *endliche* Teilmenge von M ist.
- Wir benutzen folgende Abkürzungen:
  - „f.a.“ steht für „für alle“

- „ex.“ steht für „es existiert“ bzw. „es gibt“
- „s.d.“ steht für „so dass“

Kapitel 2

Das Relationale Modell

2.1 Datenmodell

Folie 13

Datenmodell

Der Begriff „Datenmodell“ umfasst:

- einen Rahmen zur Repräsentation bzw. Speicherung von Daten
- Operationen zum Zugriff auf Daten
- Mechanismen zur Beschreibung von erwünschten Eigenschaften (Integritätsbedingungen)

Der Begriff „Datenmodell“ ist nicht präzise definiert (im mathematischen Sinn).

Im Folgenden wird eine präzise Definition des „Relationalen Modells“ gegeben.

Folie 14

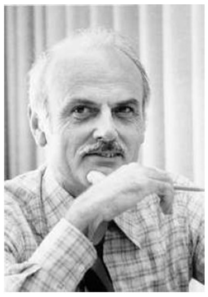
Das Relationale Modell

- Daten werden in Relationen („Tabellen“) organisiert
- Mengen-orientierte Operationen
- deklarative Anfragespezifikation

- effiziente Anfragebearbeitung
- 1970 eingeführt von Edgar F. Codd
- seit Ende der 80er Jahre „Industriestandard“

Beispiel-Relation:

A	B	C	D
a	1	z	9
b	2	z	8
c	1	y	8
d	4	x	7



Edgar F. Codd (1923-2003)

Folie 15

Grundbegriff: Relationsschema

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von *Attributnamen.* Diese Menge sei *geordnet* via $\leqslant_{\mathbf{att}}$.
- Eine abzählbar unendliche Menge **dom** von „potentiellen Datenbankeinträgen“ (*„Konstanten“*) (**dom** steht für „Domäne“ bzw. „Domain“)
- Eine abzählbar unendliche Menge **rel** von *Relationsnamen.* Die Mengen **att**, **dom**, **rel** seien disjunkt.
- Eine Funktion $\mathit{sorte} : \mathbf{rel} \to \mathcal{P}_e(\mathbf{att})$, die jedem Relationsnamen eine endliche Menge von Attributnamen zuordnet,
  und zwar so, dass für alle $U \in \mathcal{P}_e(\mathbf{att})$ gilt: $\mathit{sorte}^{-1}(U)$ ist unendlich.
  Notation: *$\mathcal{P}_e(M)$ ist die Menge aller* endlichen *Teilmengen von M.*
  D.h.: Für jede endliche Menge U von Attributnamen gibt es unendlich viele verschiedene Relationsnamen R der Sorte U.
- Die *Stelligkeit* eines Relationsnamens R ist $\mathrm{ar}(R) := |\mathit{sorte}(R)|$.
- Ein *Relationsschema* ist einfach ein Relationsname R.
- Manchmal schreiben wir kurz $R[U]$ für $\mathit{sorte}(R) = U$ und $R[k]$ für $\mathrm{ar}(R) = k$.

Beispiel.

A	B	C	D
a	1	z	9
b	2	z	8
c	1	y	8
d	4	x	7

ist eine Relation vom Schema R mit $\mathit{sorte}(R) = \{A, B, C, D\}$ und $\mathrm{ar}(R) = 4$.

Folie 16

Relationsschema vs. Relation

Relation $\widehat{=}$ Tabelle

Tupel $\widehat{=}$ Zeile in der Tabelle

Schreibweise: $t.A$ an Stelle von $t(A)$
für „Eintrag in Zeile t und Spalte A"

Beachte: *Mengensemantik*, d.h.:
Relation $\widehat{=}$ die Menge aller Tabellenzeilen

Definition 2.1. Sei R ein Relationsschema.

- Ein *R-Tupel* ist eine Abbildung $t : \mathit{sorte}(R) \to \mathbf{dom}$.
- Eine *R-Relation* ist eine <u>endliche</u> Menge von R-Tupeln.
- *inst*(R) bezeichnet die Menge aller R-Relationen.
  (*inst* steht für „Instanzen" bzw. „instances")

Folie 17

Grundbegriffe: Datenbankschema und Datenbank

Definition 2.2.

- Ein *Datenbankschema* (kurz: *DB-Schema*) $\mathbf{S}$ ist eine endliche, nicht-leere Menge von Relationsschemata.

  Manchmal schreiben wir $\mathbf{S} = \{\, R_1[U_1], \ldots, R_n[U_n] \,\}$ um die Relationsschemata anzugeben, die zu $\mathbf{S}$ gehören.

- Eine *Datenbank* (bzw. *Datenbankinstanz*) $\mathbf{I}$ vom Schema $\mathbf{S}$ ist eine Funktion, die jedem Relationsschema $R \in \mathbf{S}$ eine R-Relation zuordnet.
- $inst(\mathbf{S})$ bezeichnet die Menge aller Datenbanken vom Schema $\mathbf{S}$.
- $schema(\mathbf{I}) := \mathbf{S}$ bezeichnet das Schema der Datenbank $\mathbf{I}$.

Folie 18

Beispieldatenbank mit Kinodaten

Zur Illustration von Anfragen verwenden wir eine kleine Datenbank mit Kinodaten, bestehend aus

- einer Relation *Kinos*, die Informationen über Kinos (Name, Adresse, Stadtteil, Telefon) enthält.
- einer Relation *Filme*, die Informationen über Filme enthält (Titel, Regie, Schauspieler)
- einer Relation *Programm*, die Informationen zum aktuellen Kinoprogramm enthält (Kino, Titel, Zeit)

Folie 19

Kinos			
Name	Adresse	Stadtteil	Telefon
Babylon	Dresdner Str. 126	Kreuzberg	030 61 60 96 93
Casablanca	Friedenstr. 12-13	Adlershof	030 67 75 75 2
Filmtheater am Friedrichshain	Bötzowstr. 1-5	Prenzlauer Berg	030 42 84 51 88
Kino International	Karl-Marx-Allee 33	Mitte	030 24 75 60 11
Moviemento	Kotbusser Damm 22	Kreuzberg	030 692 47 85
Urania	An der Urania 17	Schöneberg	030 21 89 09 1

Filme		
Titel	Regie	Schauspieler
Alien	Ridley Scott	Sigourney Weaver
Blade Runner	Ridley Scott	Harrison Ford
Blade Runner	Ridley Scott	Sean Young
Brazil	Terry Gilliam	Jonathan Pryce
Brazil	Terry Gilliam	Kim Greist
Casablanca	Michael Curtiz	Humphrey Bogart
Casablanca	Michael Curtiz	Ingrid Bergmann
Gravity	Alfonso Cuaron	Sandra Bullock
Gravity	Alfonso Cuaron	George Clooney
Monuments Men	George Clooney	George Clooney
Monuments Men	George Clooney	Matt Damon
Resident Evil	Paul Anderson	Milla Jovovich
Terminator	James Cameron	Arnold Schwarzenegger
Terminator	James Cameron	Linda Hamilton
Terminator	James Cameron	Michael Biehn
...	...	...

Folie 20

Programm		
Kino	Titel	Zeit
Babylon	Casablanca	17:30
Babylon	Gravity	20:15
Casablanca	Blade Runner	15:30
Casablanca	Alien	18:15
Casablanca	Blade Runner	20:30
Casablanca	Resident Evil	20:30
Filmtheater am Friedrichshain	Resident Evil	20:00
Filmtheater am Friedrichshain	Resident Evil	21:30
Filmtheater am Friedrichshain	Resident Evil	23:00
Kino International	Casablanca	18:00
Kino International	Brazil	20:00
Kino International	Brazil	22:00
Moviemento	Gravity	17:00
Moviemento	Gravity	19:30
Moviemento	Alien	22:00
Urania	Monuments Men	17:00
Urania	Monuments Men	20:00

Folie 21

Datenbankschema der Kinodatenbank:

- Datenbankschema $\mathbf{KINO} = \{\mathit{Kinos}, \mathit{Filme}, \mathit{Programm}\}$
- $\mathit{sorte}(\mathit{Kinos}) = \{\text{Name}, \text{Adresse}, \text{Stadtteil}, \text{Telefon}\}$
- $\mathit{sorte}(\mathit{Filme}) = \{\text{Titel}, \text{Regie}, \text{Schauspieler}\}$
- $\mathit{sorte}(\mathit{Programm}) = \{\text{Kino}, \text{Titel}, \text{Zeit}\}$

Wir schreiben $\mathbf{I_{KINO}}$, um unsere konkrete Datenbank vom Schema **KINO** zu bezeichnen. Analog schreiben wir I_{Filme}, I_{Kinos} *und* I_{Programm} für die konkreten Relationen, die zur Datenbank $\mathbf{I_{KINO}}$ gehören.

Folie 22

Attribute: Benannte vs. Unbenannte Perspektive

Sind die Attributnamen Teil des expliziten Datenbankschemas?

In SQL: ja! Beispiel:

```
SELECT Titel FROM Filme WHERE Schauspieler = 'Sigourney Weaver'
```

Aber werden die Namen vom System nicht „weg-compiliert“?

- *Benannte Perspektive:*
  Ein Tupel über Relationsschema $R[U]$ ist eine Abbildung von U nach **dom**.
  Schreibweise: $t = (\,\text{A} : \text{a}, \quad \text{B} : 1, \quad \text{C} : \text{z}, \quad \text{D} : 9\,)$

- *Unbenannte Perspektive:*
  Ein Tupel über Relationsschema $R[k]$ ist ein Element aus $\mathbf{dom}^k$ (Kartesisches Produkt aus k Kopien von **dom**).
  Schreibweise: $t \;=\; (\text{a}, 1, \text{z}, 9)$

Folie 23

Wir nutzen folgende Schreibweisen für

Konstanten (d.h. Elemente aus **dom**)	a, b, c, "Ingrid Bergmann", ...
Attributnamen	A, B, C, ...
Mengen von Attributnamen	U, V, ...
Relationsnamen (bzw. -schemata)	R, R', $R[U]$, $R'[V]$, ...
Datenbankschemata	$\mathbf{S}$, $\mathbf{S'}$
Tupel	t, t', s
Relationen (d.h. Relations-Instanzen)	I, J
Datenbanken (Datenbank-Instanzen) .	$\mathbf{I}$, $\mathbf{J}$

2.2 Anfragen

Folie 24

Beispiel-Anfragen

(1) Wer führt Regie in "Blade Runner"?

(2) Wo läuft "Gravity"?

(3) Gib Adresse und Telefonnummer des "Kino International" aus!

(4) Welche Kinos spielen einen Film mit "Matt Damon"?

(5) Läuft zur Zeit ein Film von "James Cameron"?

(6) Welche (je 2) Schauspieler haben schon in Filmen gespielt, in denen der jeweils andere Regie geführt hat?

(7) Welche Regisseure haben in einem ihrer eigenen Filme mitgespielt?

(8) Gib die 2-Tupel von Schauspielern an, die gemeinsam in einem Film gespielt haben!

(9) Egal für welche Datenbank, gib als Antwort das Tupel (*“Terminator”*, *“Linda Hamilton”*) aus!

(10) Wo wird “Alien” oder “Brazil” gespielt?

(11) Welche Filme laufen in mindestens 2 Kinos?

(12) In welchen Filmen spielt “George Clooney” mit oder führt Regie?

(13) Gib alle Filme aus, die im “Moviemento” laufen oder in denen “Sandra Bullock” mitspielt!

(14) Liste alle Schauspieler und den Regisseur von “Monuments Men” auf!

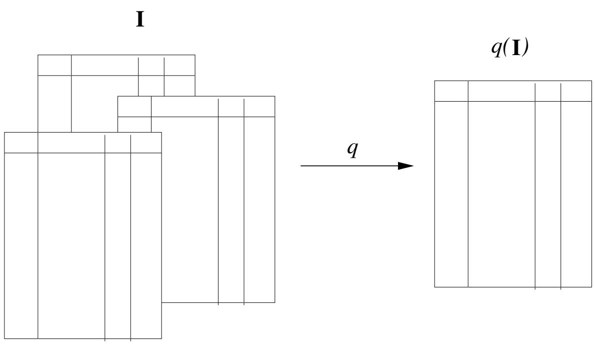
(15) Welche Filme laufen nur zu 1 Uhrzeit?

(16) In welchen Filmen spielt “George Clooney” mit, ohne Regie zu führen?

(17) Welche Filme haben nur Schauspieler, die schon mal in einem Film von “James Cameron” mitgespielt haben?

Folie 25

Anfragen und Anfragefunktionen



Definition 2.3.

- Eine *Anfragefunktion* ist eine Abbildung q, die, für ein Datenbankschema $\mathbf{S}$ und ein Relationsschema R, jeder Datenbank $\mathbf{I}$ vom Schema $\mathbf{S}$, eine Relation, $q(\mathbf{I})$ vom Schema R zuordnet. (q steht für „query“)

- Eine *Anfrage* ist eine Zeichenkette, die eine Anfragefunktion in einer bestimmten Syntax beschreibt.

Folie 26

Wünschenswerte Eigenschaften von Anfragefunktionen

Forderungen an eine Anfragefunktion q:

(a) Das Ergebnis sollte nicht von Details der Speicherung abhängen, sondern nur von der logischen Sicht auf die Daten

Dies wird dadurch gewährleistet, dass q eine Funktion $inst(\mathbf{S}) \to inst(R)$ *ist, für ein DB-Schema* $\mathbf{S}$ *und ein Rel.schema* R

(b) Sie sollte berechenbar sein.

D.h. es sollte einen Algorithmus geben, der bei Eingabe einer Datenbank $\mathbf{I}$ *das Anfrageergebnis* $q(\mathbf{I})$ *berechnet.*

(c) Das Ergebnis sollte möglichst wenig von den einzelnen Datenwerten und möglichst viel von den Beziehungen der Daten untereinander abhängen

... *siehe nächste Folie; Stichwort: „generisch“*

Folie 27

Erläuterungen zu Eigenschaft (c)

Beispiel-Anfrage:

(3) Gib Adresse und Telefonnummer des “Kino International” aus!

Eigenschaft (c):

- Wenn sich die Telefonnummer vom “Kino International” in der DB ändert, soll sich das Ergebnis der Anfrage entsprechend ändern.
- Aber wenn der Name “Kino International” sich in der DB ändert, soll das Ergebnis leer sein.
- *Allgemein:*

Werden Elemente der Datenmenge **dom**, die in der Anfrage <u>nicht</u> explizit als "Konstanten" vorkommen, umbenannt, so sollen sie im Ergebnis auch umbenannt werden.

- *Mathematische Präzisierung:* Begriff der *generischen Anfragefunktion*

Folie 28

Generische Anfragefunktionen

Definition 2.4.
Sei C eine endliche Menge von Datenwerten (kurz: $C \subseteq_e \mathbf{dom}$).

Eine Anfragefunktion q heißt *C-generisch*, falls für jede Datenbank $\mathbf{I}$ (vom zu q passenden DB-Schema) und jede Permutation π von **dom** mit $\pi|_C = id$ (d.h. $\pi(c) = c$ für alle $c \in C$) gilt:

$$q\big(\pi(\mathbf{I})\big) \quad = \quad \pi\big(q(\mathbf{I})\big).$$

q heißt *generisch*, falls q $\emptyset$-generisch ist.

Illustration:

$$\begin{array}{ccc} \mathbf{I} & \xrightarrow{q} & q(\mathbf{I}) \\ \pi\big\downarrow & & \pi\big\downarrow \\ \pi(\mathbf{I}) & \xrightarrow{q} & q\big(\pi(\mathbf{I})\big) \end{array}$$

Beispiel:

(3) Gib Adresse und Telefonnummer des "Kino International" aus!
ist {"Kino International"}-generisch.

(7) Welche Regisseure haben in einem ihrer eigenen Filme mitgespielt?
ist generisch.

Folie 29

Boolesche Anfragen

Manche Anfragen lassen sich nur mit „ja“ oder „nein“ beantworten.

Beispiel: *(5)* Läuft zur Zeit ein Film von "James Cameron"?

Konvention:

- Das Ergebnis ist eine 0-stellige Relation.
- Davon gibt es genau zwei Stück: $\emptyset$ *und* $\{\,()\,\}$.
  () steht für das „Tupel der Stelligkeit 0".
- Vereinbarung:
  - $\emptyset$ steht für „nein"
  - $\{\,()\,\}$ steht für „ja"

Folie 30

Anfragesprachen

Dieselbe Anfragefunktion kann in verschiedenen Anfragesprachen beschrieben werden.

Beispiel: *(4)* Welche Kinos spielen einen Film mit "Matt Damon"?

- *SQL:*

```
SELECT  Kinos.Name, Kinos.Adresse
FROM    Filme, Programm, Kinos
WHERE   Filme.Schauspieler = ‘‘Matt Damon’’ AND
        Filme.Titel = Programm.Titel AND
        Programm.Kino = Kinos.Name
```

- *Regelbasierte Anfrage:*

$$\begin{aligned} \mathit{Ans}(x_{\mathit{Kino}}, x_{\mathit{Adr}}) \ \leftarrow \ & \mathit{Filme}(x_{\mathit{Titel}}, x_{\mathit{Regie}}, \text{“Matt Damon”}), \\ & \mathit{Programm}(x_{\mathit{Kino}}, x_{\mathit{Titel}}, x_{\mathit{Zeit}}), \\ & \mathit{Kinos}(x_{\mathit{Kino}}, x_{\mathit{Adr}}, x_{\mathit{St}}, x_{\mathit{Tel}}) \end{aligned}$$

- *Relationenkalkül:*

$$\left\{ \begin{array}{lll} \big(x_{\mathit{Kino}}, x_{\mathit{Adr}}\big) & : & \exists x_T\ \exists x_R\ \exists x_Z\ \exists x_{St}\ \exists x_{\mathit{Tel}}\ \big(\ \mathit{Filme}(x_T, x_R, \text{“Matt Damon”}) \wedge \\ & & \qquad \mathit{Programm}(x_{\mathit{Kino}}, x_T, x_Z)\ \wedge\ \mathit{Kinos}(x_{\mathit{Kino}}, x_{\mathit{Adr}}, x_{St}, x_{\mathit{Tel}})\ \big) \end{array} \right\}$$

- *Relationale Algebra:*

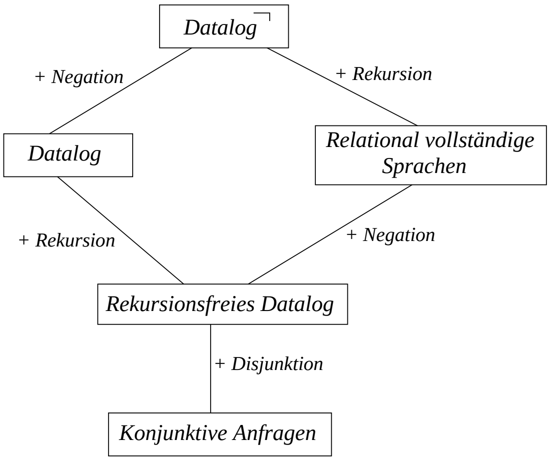
$$\pi_{\mathit{Kino},\mathit{Adresse}}\Big(\sigma_{\substack{\mathit{Schauspieler}\,=\\ \text{“Matt Damon”}}}\ \big(\mathit{Filme}\ \bowtie\ \mathit{Programm}\ \bowtie\ \delta_{\mathit{Name}\,\mapsto\,\mathit{Kino}}(\mathit{Kinos})\big)\Big)$$

Folie 31

Hierarchie der Anfragesprachen

Bemerkung: Anfragesprachen unterscheiden sich in ihrer Ausdrucksstärke.

Übersicht:



2.3 Datenkomplexität und kombinierte Komplexität

Typische Problemstellungen bzgl. Anfrageauswertung Folie 32

Sei $\mathcal{A}$ eine Anfragesprache.
Für eine Anfrage $Q \in \mathcal{A}$ schreiben wir $[\![Q]\!]$, um die von Q beschriebene *Anfragefunktion* zu bezeichnen.

AUSWERTUNGSPROBLEM FÜR $\mathcal{A}$: *Eingabe:* Anfrage $Q \in \mathcal{A}$, Datenbank $\mathbf{I}$ (vom zu Q passenden Schema) *Aufgabe:* Berechne $[\![Q]\!](\mathbf{I})$

Variante:

NICHT-LEERHEITS-PROBLEM FÜR $\mathcal{A}$: *Eingabe:* Anfrage $Q \in \mathcal{A}$, Datenbank $\mathbf{I}$ (vom zu Q passenden Schema) *Aufgabe:* Ist $[\![Q]\!](\mathbf{I}) \neq \emptyset$?

Wichtige Fragestellung:
Welche Ressourcen (etwa Zeit, Platz) sind nötig, um diese Probleme zu lösen?

Folie 33

Datenkomplexität und Kombinierte Komplexität

Die Komplexität der Anfrageauswertung kann unter zwei Blickwinkeln betrachtet werden:

1. Anfrage und Datenbank sind Eingabe $\rightsquigarrow$ *Kombinierte Komplexität*
   gemessen in n und k, wobei $n = \|\mathbf{I}\|$ und $k = \|Q\|$
2. Anfrage fest, Datenbank ist Eingabe: $\rightsquigarrow$ *Datenkomplexität*
   gemessen nur in $n = \|\mathbf{I}\|$

Rechtfertigung für „Datenkomplexität“:
i.d.R. ist die Anfrage kurz, die Datenbank aber sehr groß.

Hierbei sind $\|\mathbf{I}\|$ und $\|Q\|$ geeignete Maße für die Größe von Datenbanken **I** und die Länge von Anfragen Q.

Typische Form von Ergebnissen, die im Laufe der Vorlesung bewiesen werden:

- Die Datenkomplexität des Auswertungsproblems der Relationalen Algebra ist in LOGSPACE.
- Die kombinierte Komplexität des Auswertungsproblems der Relationalen Algebra ist PSPACE-*vollständig*.

Kapitel 3

Konjunktive Anfragen

3.1 Deskriptiver Ansatz: regelbasiert, graphisch und logikbasiert

Folie 34

Regelbasierte Konjunktive Anfragen — Informell

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?

Andere Formulierung:

> **Wenn** es in *Filme* ein Tupel (x_{Titel}, x_{Regie}, "Matt Damon") und
> in *Programm* ein Tupel ($x_{Kino}, x_{Titel}, x_{Zeit}$) und
> in *Kinos* ein Tupel ($x_{Kino}, x_{Adr}, x_{St}, x_{Tel}$) gibt,
> **dann** nimm das Tupel ($x_{Kino},\ x_{Adr}$) in die Antwort auf

Als *regelbasierte konjunktive Anfrage*:

$$\begin{aligned} \mathit{Ans}(x_{Kino}, x_{Adr}) \ \leftarrow \ & \mathit{Filme}(x_{Titel}, x_{Regie}, \text{“Matt Damon”}), \\ & \mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}), \\ & \mathit{Kinos}(x_{Kino}, x_{Adr}, x_{St}, x_{Tel}) \end{aligned}$$

Folie 35

Regelbasierte Konjunktive Anfragen — Präzise

Definition 3.1.

- **var** sei eine abzählbar unendliche Menge von *Variablen(symbolen)*, die disjunkt zu den Mengen **att**, **dom**, **rel** ist.

  Einzelne Variablen bezeichnen wir i.d.R. mit x, y, x_1, x_2, ...

- Ein *Term* ist ein Element aus $\mathbf{var} \cup \mathbf{dom}$.

- Ein *freies Tupel* der Stelligkeit k ist ein Element aus $(\mathbf{var} \cup \mathbf{dom})^k$.

Definition 3.2. Sei $\mathbf{S}$ ein Datenbankschema.
Eine *regelbasierte konjunktive Anfrage* über $\mathbf{S}$ ist ein Ausdruck der Form

$$Ans(u) \leftarrow R_1(u_1), \dots, R_\ell(u_\ell)$$

wobei $\ell \geqslant 0$, $R_1, \dots, R_\ell \in \mathbf{S}$, $Ans \in \mathbf{rel} \backslash \mathbf{S}$ und $u, u_1, \dots, u_\ell$ freie Tupel der Stelligkeiten $\mathrm{ar}(Ans)$, $\mathrm{ar}(R_1), \dots, \mathrm{ar}(R_\ell)$, so dass jede Variable, die in u vorkommt, auch in mindestens einem der Tupel $u_1, \dots, u_\ell$ vorkommt.

Folie 36

Semantik regelbasierter konjunktiver Anfragen

Sei Q eine regelbasierte konjunktive Anfrage (über einem DB-Schema $\mathbf{S}$) der Form

$$Ans(u) \leftarrow R_1(u_1), \dots, R_\ell(u_\ell)$$

- Mit $\mathrm{var}(Q)$ bezeichnen wir die Menge aller Variablen, die in Q vorkommen.

- Eine *Belegung* für Q ist eine Abbildung $\beta : \mathrm{var}(Q) \to \mathbf{dom}$.

  Wir setzen β auf natürliche Weise fort zu einer Abbildung von $\mathrm{var}(Q) \cup \mathbf{dom}$ nach $\mathbf{dom}$, so dass $\beta|_{\mathbf{dom}} = \mathrm{id}$. Für ein freies Tupel $u = (e_1, \dots, e_k)$ setzen wir $\beta(u) := (\beta(e_1), \dots, \beta(e_k))$.

- Der Anfrage Q ordnen wir die folgende Anfragefunktion $[\![Q]\!]$ zu:

$$[\![Q]\!](\mathbf{I}) \ := \ \left\{ \beta(u) \ : \ \begin{array}{l} \beta \textit{ ist eine Belegung für } Q, \textit{ so} \\ \textit{dass } \beta(u_i) \in \mathbf{I}(R_i), \textit{ für alle } i \in \\ \{1, \dots, \ell\} \end{array} \right\}$$

  für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$.

Folie 37

Beispiele

- Die Anfrage *(6)* Welche (je 2) Regisseure haben schon in Filmen gespielt, in denen der jeweils andere Regie geführt hat?

  lässt sich durch folgende regelbasierte konjunktive Anfrage ausdrücken:

  $$Antworten(x,y) \leftarrow Filme(z_1,x,y), Filme(z_2,y,x)$$

- Die Anfrage *(5)* Läuft zur Zeit ein Film von "James Cameron"?

  lässt sich durch folgende regelbasierte konjunktive Anfrage ausdrücken:

  $$Ans() \leftarrow Filme(x, \text{"James Cameron"}, y),\ \ Programm(z,x,w)$$

  Ans ist hier also ein Relationsname der Stelligkeit 0.
  Erinnern Sie sich an unsere Konvention, dass die Ausgabe „$\emptyset$" der Antwort „nein" entspricht, und die Ausgabe der Menge $\{()\}$, die aus dem „Tupel der Stelligkeit 0" besteht, der Antwort „ja" entspricht.

Folie 38

Noch ein Beispiel

Betrachte die Datenbank $\mathbf{I} := \{\mathbf{I}(R), \mathbf{I}(S)\}$ mit
$\mathbf{I}(R) := \{(a,a)\,, (a,b)\,, (b,c)\,, (c,b)\}$ und $\mathbf{I}(S) := \{(d,a,b)\,, (a,c,e)\,, (b,a,c)\}$.

- Die Anfrage $Q_1 :=$

  $$Ans_1(x_1,x_2,x_3) \leftarrow R(x_1,y), S(y,x_2,x_3)$$

  liefert auf $\mathbf{I}$ das Ergebnis $[\![Q_1]\!](\mathbf{I}) = \{\,(a,c,e)\,, (a,a,c)\,, (c,a,c)\,\}$.

- Die Anfrage $Q_2 :=$

  $$Ans_2(x,y) \leftarrow R(x,z_1),\ \ S(z_1,a,z_2),\ \ R(y,z_2)$$

  liefert auf $\mathbf{I}$ das Ergebnis $[\![Q_2]\!](\mathbf{I}) = \{\,(a,b)\,, (c,b)\,\}$.

Folie 39

Bezeichnungen

Oft sagen wir kurz *Regel*, um eine regelbasierte konjunktive Anfrage zu bezeichnen.

Sei Q eine Regel der Form $\quad Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$

- $Ans(u)$ wird als *Kopf* der Regel bezeichnet.
- $R_1(u_1), \ldots, R_\ell(u_\ell)$ wird als *Rumpf* der Regel bezeichnet.
- Die Relationsnamen aus $\mathbf{S}$ werden *extensionale Datenbankprädikate* (kurz: *edb*-Prädikate) genannt.
  Wir schreiben $edb(Q)$, um die Menge aller edb-Prädikate zu bezeichnen, die in Q vorkommen.
- Der Relationsname, der im Kopf von Q vorkommt, wird als *intensionales Datenbankprädikat* (kurz: *idb*-Prädikat) bezeichnet.
- Mit $\text{adom}(Q)$ bezeichnen wir die Menge aller *Konstanten* (also Elemente aus **dom**), die in Q vorkommen.

  „adom“ *steht für „aktive Domäne“ bzw. „active domain“.*

Folie 40

Der „Active Domain“ einer Datenbank

Definition 3.3.
Sei $\mathbf{S}$ ein Datenbankschema und sei $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$.
Der *Active Domain von* $\mathbf{I}$, kurz: $\text{adom}(\mathbf{I})$, ist die *Menge aller Elemente aus* **dom***, die in* $\mathbf{I}$ *vorkommen.* D.h.:

$$\text{adom}(\mathbf{I}) \;=\; \bigcup_{R \in \mathbf{S}} \text{adom}\big(\mathbf{I}(R)\big)$$

wobei für jedes R aus $\mathbf{S}$ gilt: $\text{adom}\big(\mathbf{I}(R)\big)$ ist die kleinste Teilmenge von **dom**, so dass jedes Tupel $t \in \mathbf{I}(R)$ eine Funktion von $sorte(R)$ nach $\text{adom}\big(\mathbf{I}(R)\big)$ ist.

Ist Q eine Anfrage und $\mathbf{I}$ eine Datenbank, so setzen wir

$$\text{adom}(Q, \mathbf{I}) \;:=\; \text{adom}(Q) \cup \text{adom}(\mathbf{I})$$

Proposition 3.4.
Für jede regelbasierte konjunktive Anfrage Q und jede Datenbank $\mathbf{I}$ (vom passenden DB-Schema) gilt: $\operatorname{adom}\big([\![Q]\!](\mathbf{I})\big) \subseteq \operatorname{adom}(Q, \mathbf{I})$.

Beweis. Sei Q von der Form

$$Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell).$$

Sei $r := \operatorname{ar}(Ans)$ und $u = (x_1, \ldots, x_r)$. Insbesondere ist jeder Eintrag x_j von u ein Element aus $\mathbf{var} \cup \mathbf{dom}$.
Sei $t := (t_1, \ldots, t_r)$ ein beliebiges Element aus $[\![Q]\!](\mathbf{I})$. Wir müssen zeigen, dass jeder Eintrag t_j von t ein Element in $\operatorname{adom}(Q, \mathbf{I})$ ist.
Wegen $t \in [\![Q]\!](\mathbf{I})$ gibt es gemäß der Definition der Semantik regelbasierter konjunktiver Anfragen eine Belegung β für Q, so dass gilt:

(a) $t = \beta(u)$ und

(b) $\beta(u_i) \in \mathbf{I}(R_i)$, für jedes $i \in [\ell] := \{1, \ldots, \ell\}$.

Wegen (a) gilt also für jedes $j \in [r]$, dass $t_j = \beta(x_j)$.
Fall 1: $x_j \in \mathbf{dom}$.
Dann gilt: $t_j = \beta(x_j) = x_j \in \operatorname{adom}(Q) \subseteq \operatorname{adom}(Q, \mathbf{I})$.
Fall 2: $x_j \in \mathbf{var}$.
Gemäß der Definition der Syntax regelbasierter konjunktiver Anfragen (Definition 3.2) gibt es ein $i \in [\ell]$, so dass x_j im freien Tupel u_i vorkommt. Dann kommt also $\beta(x_j)$ im Tupel $\beta(u_i)$ vor. Gemäß (b) wissen wir, dass $\beta(u_i) \in \mathbf{I}(R_i)$ ist. Somit ist $\beta(x_j) \in \operatorname{adom}(\mathbf{I}) \subseteq \operatorname{adom}(Q, \mathbf{I})$. $\square$

Folie 41

Monotonie und Erfüllbarkeit

Sind $\mathbf{I}$ und $\mathbf{J}$ zwei Datenbanken vom gleichen Schema $\mathbf{S}$, so sagen wir „$\mathbf{J}$ *ist eine Erweiterung von* $\mathbf{I}$" und schreiben kurz „$\mathbf{J} \supseteq \mathbf{I}$" (bzw. „$\mathbf{I} \subseteq \mathbf{J}$"), falls für alle $R \in \mathbf{S}$ gilt: $\mathbf{I}(R) \subseteq \mathbf{J}(R)$ (d.h. jedes Tupel, das in einer Relation von $\mathbf{I}$ vorkommt, kommt auch in der entsprechenden Relation von $\mathbf{J}$ vor).

Definition 3.5.
Sei $\mathbf{S}$ ein DB-Schema und sei q eine Anfragefunktion über $\mathbf{S}$.

(a) q heißt *monoton*, falls für alle Datenbanken $\mathbf{I}$ und $\mathbf{J}$ über $\mathbf{S}$ gilt:
Falls $\mathbf{I} \subseteq \mathbf{J}$, so ist $q(\mathbf{I}) \subseteq q(\mathbf{J})$.

(b) q heißt *erfüllbar*, falls es eine Datenbank $\mathbf{I}$ gibt mit $q(\mathbf{I}) \neq \emptyset$.

Satz 3.6. *Jede regelbasierte konjunktive Anfrage ist monoton und erfüllbar.*

Beweis. Sei $\mathbf{S}$ ein DB-Schema und sei Q eine Anfrage über $\mathbf{S}$ der Form

$$Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell).$$

Nachweis der Monotonie von Q:
Seien $\mathbf{I}$ und $\mathbf{J}$ Datenbanken mit $\mathbf{I} \subseteq \mathbf{J}$.
Wir müssen zeigen, dass $[\![Q]\!](\mathbf{I}) \subseteq [\![Q]\!](\mathbf{J})$ ist.
Sei also $t \in [\![Q]\!](\mathbf{I})$ beliebig gewählt. Zu zeigen: $t \in [\![Q]\!](\mathbf{J})$.

Wegen $t \in [\![Q]\!](\mathbf{I})$ gibt es eine Belegung β für Q, so dass $t = \beta(u)$ ist und für alle $i \in [\ell]$ gilt $\beta(u_i) \in \mathbf{I}(R_i)$.
Wegen $\mathbf{I} \subseteq \mathbf{J}$ gilt $\mathbf{I}(R_i) \subseteq \mathbf{J}(R_i)$ für jedes $i \in [\ell]$.
Somit gilt für jedes $i \in [\ell]$, dass $\beta(u_i) \in \mathbf{I}(R_i) \subseteq \mathbf{J}(R_i)$. Daher ist β eine Belegung, die gemäß der Definition der Semantik regelbasierter konjunktiver Anfragen bezeugt, dass $\beta(u) \in [\![Q]\!](\mathbf{J})$ ist. Also ist $t \in [\![Q]\!](\mathbf{J})$.

Nachweis der Erfüllbarkeit von Q:
Wir müssen eine Datenbank $\mathbf{I}$ vom Schema $\mathbf{S}$ finden, für die $[\![Q]\!](\mathbf{I}) \neq \emptyset$ ist.

Dazu wählen wir eine Konstante $a \in \mathbf{dom} \setminus \text{adom}(Q)$ (die gibt es, da $\text{adom}(Q)$ endlich und $\mathbf{dom}$ unendlich ist).
Für jedes $R \in \mathbf{S}$ setze

$$\mathbf{I}(R) \quad := \quad \big(\ \text{adom}(Q) \cup \{a\}\ \big)^{\text{ar}(R)}.$$

Sei β die Belegung für Q, die jede Variable aus Q auf das Element a abbildet.
Dann ist $\beta(u) \in [\![Q]\!](\mathbf{I})$, denn für jedes $i \in [\ell]$ ist

$$\beta(u_i) \quad \in \quad \underbrace{\big(\ \text{adom}(Q) \cup \{a\}\ \big)^{\text{ar}(R_i)}}_{=\,\mathbf{I}(R_i)}.$$

Insbesondere ist $[\![Q]\!](\mathbf{I}) \neq \emptyset$, d.h. Q ist erfüllbar. □

Folie 42

Anwendung von Satz 3.6

Satz 3.6 liefert ein einfaches Kriterium, um zu zeigen, dass bestimmte Anfragefunktionen nicht durch eine regelbasierte konjunktive Anfrage beschrieben werden können:

> Wenn eine Anfragefunktion q *nicht monoton* ist, *dann* kann sie auch *nicht durch eine regelbasierte konjunktive Aussage beschrieben* werden.

Vorsicht: Dies heißt nicht, dass jede monotone Anfragefunktion durch eine regelbasierte konjunktive Anfrage beschrieben werden kann!

Beispiel: Die Anfrage

(15) Welche Filme laufen nur zu 1 Uhrzeit?

ist nicht monoton, kann also nicht durch eine regelbasierte konjunktive Anfrage beschrieben werden.

Dass die Anfrage *(15)* nicht monoton ist, sieht man z.B. dadurch, dass man die Datenbank $\mathbf{I}$ betrachtet, die aus unserer Beispiel-Datenbank $\mathbf{I_{KINO}}$ entsteht, indem das Tupel (Babylon, Casablanca, 17:30) aus der *Programm*-Relation gelöscht wird. Offensichtlicherweise ist $\mathbf{I} \subseteq \mathbf{I_{KINO}}$. Aber das Ergebnis der Anfrage *(15)* ist bei Auswertung über der Datenbank $\mathbf{I_{KINO}}$ leer, während es bei Auswertung über der Datenbank $\mathbf{I}$ den Film "Casablanca" liefert (der gemäß der Datenbank $\mathbf{I}$ nur um 18:00 Uhr läuft).

Folie 43

„Graphische" Variante: Tableau-Anfragen

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?

Darstellung als Tableau T *(ähnlich zu „Query by Example" (QBE) von IBM):*

Filme

Titel	Regie	Schauspieler
x_{Titel}	x_{Regie}	"Matt Damon"

Programm

Kino	Titel	Zeit
x_{Kino}	x_{Titel}	x_{Zeit}

Kinos

Kino	Adresse	Stadtteil	Telefon
x_{Kino}	x_{Adr}	x_{St}	x_{Tel}

Zugehörige Tableau-Anfrage: $\big(\mathsf{T},\ (x_{Kino},\ x_{Adr})\big)$

Folie 44

Tableaus — Präzise

Definition 3.7. Sei $\mathbf{S}$ ein Datenbankschema und R ein Relationsschema.

- Ein *Tableau über* R (auch: Einzel-Tableau) ist eine endliche Menge von freien Tupeln (also Tupeln über $\mathbf{dom} \cup \mathbf{var}$) der Stelligkeit $\mathrm{ar}(R)$.

  (D.h. ein Tableau über R ist eine „Relation vom Schema R, die als Einträge nicht nur Elemente aus **dom**, sondern auch Variablen aus **var** haben kann".)

- Ein *Tableau* T *über* $\mathbf{S}$ ist eine Abbildung, die jedem $R \in \mathbf{S}$ ein Tableau über R zuordnet.

  (D.h. ein Tableau über $\mathbf{S}$ ist eine „Datenbank vom Schema $\mathbf{S}$, die als Einträge auch Variablen enthalten kann".)

- Eine *Tableau-Anfrage über* $\mathbf{S}$ (bzw. R) ist von der Form (T, u), wobei T ein Tableau über $\mathbf{S}$ (bzw. R) und u ein freies Tupel ist, so dass jede Variable, die in u vorkommt, auch in T vorkommt.

  u heißt Zusammenfassung der Anfrage (T, u).

Folie 45

Semantik von Tableau-Anfragen

Sei $Q = (\mathsf{T}, u)$ *eine Tableau-Anfrage.*

- $\mathrm{var}(Q)$ bezeichnet die Menge aller Variablen, die in u oder T vorkommen. $\mathrm{adom}(Q)$ bezeichnet die Menge aller Konstanten, die in u oder T vorkommen.
- Eine *Belegung für* Q ist eine Abbildung $\beta : \mathrm{var}(Q) \to \mathbf{dom}$.

  Wir schreiben $\beta(\mathsf{T})$, um die Datenbank zu bezeichnen, die aus T entsteht, indem man jedes Variablensymbol x in T durch die Konstante $\beta(x)$ ersetzt.

- Sei $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$.

  Eine Belegung β für Q heißt *Einbettung von* T *in* $\mathbf{I}$, falls $\beta(\mathsf{T}) \subseteq \mathbf{I}$ gilt (d.h. die Datenbank $\mathbf{I}$ ist eine Erweiterung der Datenbank $\beta(\mathsf{T})$).

- Der Tableau-Anfrage Q ordnen wir die folgende Anfragefunktion $[\![Q]\!]$ zu:

$$[\![Q]\!](\mathbf{I}) \ := \ \{\, \beta(u) \ : \ \beta \text{ ist eine Einbettung von } \mathsf{T} \text{ in } \mathbf{I} \,\}$$

für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$.

Folie 46

Logikbasierte Variante: Konjunktiver Kalkül

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?

Formulierung als Anfrage des konjunktiven Kalküls:

$$\left\{ \begin{array}{lll} \big(x_{\mathit{Kino}}, x_{\mathit{Adr}}\big) & : & \exists x_{\mathit{Titel}}\, \exists x_{\mathit{Regie}}\, \exists x_{\mathit{Zeit}}\, \exists x_{\mathit{St}}\, \exists x_{\mathit{Tel}}\ \big(\\ & & \qquad \mathit{Filme}(x_{\mathit{Titel}}, x_{\mathit{Regie}}, \text{“Matt Damon”}) \wedge \\ & & \qquad \mathit{Programm}(x_{\mathit{Kino}}, x_{\mathit{Titel}}, x_{\mathit{Zeit}}) \wedge \\ & & \qquad \mathit{Kinos}(x_{\mathit{Kino}}, x_{\mathit{Adr}}, x_{\mathit{St}}, x_{\mathit{Tel}})\ \big) \end{array} \right\}$$

Auf der rechten Seite des „:“
werden Varianten von Formeln der *Logik erster Stufe* verwendet.
Hier: eingeschränkte Variante, in der es nur $\exists$-Quantoren und Konjunktionen ($\wedge$) gibt.

Folie 47

Konjunktiver Kalkül (CQ) — Präzise

Definition 3.8. Sei $\mathbf{S}$ ein Datenbankschema.
Die Menge $\mathsf{CQ}[\mathbf{S}]$ aller Formeln des *konjunktiven Kalküls über* $\mathbf{S}$ ist induktiv wie folgt definiert: *(CQ steht für „conjunctive queries“)*

(A) $R(v_1, \ldots, v_r)$ gehört zu $\mathsf{CQ}[\mathbf{S}]$,
für alle $R \in \mathbf{S}$, $r := \operatorname{ar}(R)$ und $v_1, \ldots, v_r \in \mathbf{var} \cup \mathbf{dom}$.

(K) $(\varphi \wedge \psi)$ gehört zu $\mathsf{CQ}[\mathbf{S}]$,
für alle $\varphi \in \mathsf{CQ}[\mathbf{S}]$ und $\psi \in \mathsf{CQ}[\mathbf{S}]$.

(E) $\exists x\, \varphi$ gehört zu $\mathsf{CQ}[\mathbf{S}]$,
für alle $x \in \mathbf{var}$ und $\varphi \in \mathsf{CQ}[\mathbf{S}]$.

Insbesondere: Jede Formel in $\mathsf{CQ}[\mathbf{S}]$ ist eine Formel der *Logik erster Stufe* über der Signatur $\mathbf{S} \cup \mathbf{dom}$ (wobei jedes Element aus **dom** als „Konstanten-Symbol“ aufgefasst wird, das stets „mit sich selbst“ interpretiert wird).

Folie 48

Semantik von $\mathsf{CQ}[\mathbf{S}]$

Sei φ eine $\mathsf{CQ}[\mathbf{S}]$-Formel.

- adom(φ) bezeichnet die Menge aller Konstanten (also Elemente aus **dom**), die in φ vorkommen.

  var(φ) bezeichnet die Menge aller Variablen (also Elemente aus **var**), die in φ vorkommen.

- frei(φ) bezeichnet die Menge aller Variablen, die *frei* in φ vorkommen.
  D.h.: $\text{frei}(R(v_1,\ldots,v_r)) = \{v_1,\ldots,v_r\} \cap \mathbf{var}$;
  $\text{frei}((\varphi \wedge \psi)) = \text{frei}(\varphi) \cup \text{frei}(\psi)$;
  $\text{frei}(\exists x\varphi) = \text{frei}(\varphi) \setminus \{x\}$.

- Eine *Belegung für* φ ist eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{dom}$.

- Einer *Datenbank* $\mathbf{I}$ vom Schema $\mathbf{S}$ ordnen wir die *logische Struktur*

$$\mathcal{A}_{\mathbf{I}} := \Big(\mathbf{dom},\ \big(\mathbf{I}(R)\big)_{R\in\mathbf{S}},\ (c)_{c\in\mathbf{dom}}\Big)$$

  zu. *Insbesondere ist* $\mathcal{A}_{\mathbf{I}}$ *eine* σ*-Struktur über der Signatur* $\sigma := \mathbf{S} \cup \mathbf{dom}$.

- Ist $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$ und β eine Belegung für φ, so sagen wir „$\mathbf{I}$ *erfüllt* φ *unter* β“ und schreiben $\mathbf{I} \models \varphi[\beta]$ bzw. $(\mathbf{I},\beta) \models \varphi$, um auszudrücken, dass $(\mathcal{A}_{\mathbf{I}},\beta) \models \varphi$ gilt.

Bemerkung. Wir verwenden hier Notationen, die in der Veranstaltung *Logik in der Informatik* eingeführt wurden. Der Vollständigkeit halber geben wir hier die Definition rekursiv für alle $\mathsf{CQ}[\mathbf{S}]$-Formeln φ an:

- Falls φ von der Form $R(v_1,\ldots,v_r)$ ist, für $R \in \mathbf{S}$, $r = \text{ar}(R)$ und $v_1,\ldots,v_r \in \text{var} \cup \mathbf{dom}$, so gilt

$$(\mathcal{A}_{\mathbf{I}},\beta) \models \varphi \quad :\Longleftrightarrow \quad \big(\beta(v_1),\ldots,\beta(v_r)\big) \in \mathbf{I}(R)$$

- Falls φ von der Form $(\psi_1 \wedge \psi_2)$ ist, für $\psi_1 \in \mathsf{CQ}[\mathbf{S}]$ und $\psi_2 \in \mathsf{CQ}[\mathbf{S}]$, so gilt

$$(\mathcal{A}_{\mathbf{I}}, \beta) \models \varphi \quad :\Longleftrightarrow \quad \big(\ (\mathcal{A}_{\mathbf{I}}, \beta) \models \psi_1 \text{ und } (\mathcal{A}_{\mathbf{I}}, \beta) \models \psi_2\ \big)$$

- Falls φ von der Form $\exists x\, \psi$ ist, für $x \in \text{var}$ und $\psi \in \mathsf{CQ}[\mathbf{S}]$, so gilt

$(\mathcal{A}_{\mathbf{I}}, \beta) \models \varphi \quad :\Longleftrightarrow$ es gibt ein $a \in \mathbf{dom}$, so dass für die Belegung $\beta\frac{a}{x} : \text{frei}(\varphi) \to \mathbf{dom}$ mit $\beta\frac{a}{x}(x) := a$ und $\beta\frac{a}{x}(y) := \beta(y)$ für alle $y \neq x$ gilt: $(\mathcal{A}_{\mathbf{I}}, \beta\frac{a}{x}) \models \psi$

Folie 49

Notation

- Mit CQ bezeichnen wir die Klasse aller $\mathsf{CQ}[\mathbf{S}]$-Formeln für alle Datenbankschemata $\mathbf{S}$.

- Manchmal schreiben wir $\{x_1/a_1, \ldots, x_r/a_r\}$ um die *Belegung* $\beta : \{x_1, \ldots, x_r\} \to \mathbf{dom}$ zu bezeichnen mit $\beta(x_i) = a_i$, f.a. $i \in \{1, \ldots, r\}$.

- Für eine CQ-Formel φ schreiben wir oft $\varphi(x_1, \ldots, x_r)$, um anzudeuten, dass $\text{frei}(\varphi) \subseteq \{x_1, \ldots, x_r\}$.

- Ist $a_1, \ldots, a_r \in \mathbf{dom}$, so schreiben wir vereinfachend $\mathbf{I} \models \varphi[a_1, \ldots, a_r]$ an Stelle von $\mathbf{I} \models \varphi[\{x_1/a_1, \ldots, x_r/a_r\}]$.

- Ist $y \in \mathbf{dom} \cup \mathbf{var}$, so bezeichnet $\varphi(x_1/y, x_2, \ldots, x_r)$ die Formel, die aus φ entsteht, indem jedes Vorkommen der *Variablen x_1 durch y ersetzt* wird.

- Beim Schreiben von Formeln lassen wir Klammern „(“, „)“ oft weg und schreiben „$\exists x_1, \ldots, x_n$“ als Abkürzung für „$\exists x_1\, \exists x_2 \cdots \exists x_n$“.

- Zwei $\mathsf{CQ}[\mathbf{S}]$-Formeln φ *und* ψ heißen *äquivalent*, falls $\text{frei}(\varphi) = \text{frei}(\psi)$ und für jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und jede Belegung β für φ (und ψ) gilt: $\mathbf{I} \models \varphi[\beta] \iff \mathbf{I} \models \psi[\beta]$.

Folie 50

Konjunktiver Kalkül: Syntax und Semantik

Definition 3.9. Sei $\mathbf{S}$ ein Datenbankschema.
Eine *Anfrage des konjunktiven Kalküls* ist von der Form

$$\{\,(e_1,\ldots,e_r)\ :\ \varphi\}$$

wobei $\varphi \in \mathsf{CQ}[\mathbf{S}]$, $r \geqslant 0$ und $(e_1,\ldots,e_r)$ *ein freies Tupel* ist, so dass $\mathrm{frei}(\varphi) = \{e_1,\ldots,e_r\} \cap \mathbf{var}$.

Semantik:
Einer Anfrage Q der Form $\{(e_1,\ldots,e_r)\ :\ \varphi\}$ ordnen wir die folgende Anfragefunktion $[\![Q]\!]$ zu:

$$[\![Q]\!](\mathbf{I}) \ := \ \left\{ \ \beta\big(\,(e_1,\ldots,e_r)\big) \ : \ \begin{array}{l} \beta \textit{ ist eine Belegung für } \varphi \\ \textit{mit } \mathbf{I} \models \varphi[\beta] \end{array} \right\}$$

für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$.

Folie 51

Wertebereich von Anfragen des konjunktiven Kalküls

Für eine Anfrage Q der Form $\{(e_1,\ldots,e_r)\ :\ \varphi\}$ setzen wir

$$\mathrm{adom}(Q) \ := \ \mathrm{adom}(\varphi) \cup \big(\{e_1,\ldots,e_r\} \cap \mathbf{dom}\big).$$

Ist Q eine Anfrage und $\mathbf{I}$ eine Datenbank, so setzen wir – wie üblich –

$$\mathrm{adom}(Q,\mathbf{I}) \ := \ \mathrm{adom}(Q) \cup \mathrm{adom}(\mathbf{I})$$

Analog zu Proposition 3.4 gilt auch für Anfragen des konjunktiven Kalküls:

Proposition 3.10.
Für jede Anfrage Q des konjunktiven Kalküls und jede Datenbank $\mathbf{I}$ (vom passenden DB-Schema) gilt: $\mathrm{adom}\big([\![Q]\!](\mathbf{I})\big) \ \subseteq \ \mathrm{adom}(Q,\mathbf{I})$.

Beweis: Induktion über den Formelaufbau. Details: Übung.

Folie 52

Beispiel

Die Anfrage

> Gibt es einen Schauspieler, der sowohl in einem Film von "Paul Anderson" als auch in einem Film von "Ridley Scott" mitgespielt hat?

wird durch die folgende Anfrage des konjunktiven Kalküls beschrieben:

$$\Big\{ () \ : \ \exists x_{\textit{Schauspieler}} \Big(\exists x_{\textit{Titel1}} \ \textit{Filme}(x_{\textit{Titel1}}, \textit{``Paul Anderson''}, x_{\textit{Schauspieler}}) \ \wedge \ \exists x_{\textit{Titel2}} \ \textit{Filme}(x_{\textit{Titel2}}, \textit{``Ridley Scott''}, x_{\textit{Schauspieler}}) \Big) \Big\}$$

und durch die dazu äquivalente Anfrage

$$\Big\{ () \ : \ \exists x_{\textit{Schauspieler}} \ \exists x_{\textit{Titel1}} \ \exists x_{\textit{Titel2}} \Big(\textit{Filme}(x_{\textit{Titel1}}, \textit{``Paul Anderson''}, x_{\textit{Schauspieler}}) \ \wedge \ \textit{Filme}(x_{\textit{Titel2}}, \textit{``Ridley Scott''}, x_{\textit{Schauspieler}}) \Big) \Big\}$$

Folie 53

Eine Normalform für CQ

Definition 3.11.
Eine $\mathsf{CQ}[\mathbf{S}]$-*Formel* $\varphi(x_1, \ldots, x_r)$ ist *in Normalform*, falls sie von der Form

$$\exists x_{r+1} \cdots \exists x_{r+s} \ \Big(R_1(u_1) \wedge \cdots \wedge R_\ell(u_\ell) \Big)$$

ist, wobei gilt: $r, s, \ell \geqslant 0$, $R_1, \ldots, R_\ell \in \mathbf{S}$, $u_1, \ldots, u_\ell$ sind freie Tupel über $\{x_1, \ldots, x_{r+s}\} \cup \mathbf{dom}$, $\text{frei}(\varphi) = \{x_1, \ldots, x_r\}$ und $x_1, \ldots, x_{r+s}$ sind paarweise verschiedene Elemente aus **var**.

Eine Anfrage Q des konjunktiven Kalküls ist in Normalform, falls sie von der Form $\{(e_1, \ldots, e_r) \ : \ \varphi\}$ ist, wobei φ eine CQ-Formel in Normalform ist.

Lemma 3.12.
Jede CQ*-Formel ist äquivalent zu einer* CQ*-Formel in Normalform.*
Und es gibt einen Polynomialzeit-Algorithmus, der bei Eingabe einer CQ*-Formel eine äquivalente* CQ*-Formel in Normalform konstruiert.*

Beweis: Übung.

Folie 54

Beispiel

Die CQ-Formel

$$\begin{array}{l} \exists z \exists z' \ \mathit{Programm}(z, x, z') \ \wedge \\ \exists z \ \mathit{Filme}(x, y, z) \ \wedge \\ \exists z \ \mathit{Filme}(z, y, \text{“George Clooney”}) \ \wedge \\ \exists z \ \mathit{Filme}(z, \text{“George Clooney”}, y) \end{array}$$

ist <u>nicht</u> in Normalform, aber äquivalent zur Normalform-Formel

$$\exists z_1 \cdots \exists z_5 \ \big(\ \begin{array}{l} \mathit{Programm}(z_1, x, z_2) \wedge \\ \mathit{Filme}(x, y, z_3) \wedge \\ \mathit{Filme}(z_4, y, \text{“George Clooney”}) \wedge \\ \mathit{Filme}(z_5, \text{“George Clooney”}, y) \end{array}$$

wobei $z_1, \ldots, z_5$ paarweise verschiedene Elemente aus **var** sind.

Folie 55

Äquivalenz von Anfragesprachen

Definition 3.13. Seien $\mathcal{Q}_1$ und $\mathcal{Q}_2$ zwei Anfragesprachen. Wir schreiben

- $\mathcal{Q}_1 \leqslant \mathcal{Q}_2$ (bzw. $\mathcal{Q}_2 \geqslant \mathcal{Q}_1$; „*$\mathcal{Q}_2$ ist mindestens so ausdrucksstark wie $\mathcal{Q}_1$*“), falls jede Anfragefunktion, die durch eine Anfrage in $\mathcal{Q}_1$ ausgedrückt werden kann, auch durch eine Anfrage in $\mathcal{Q}_2$ ausgedrückt werden kann.
- $\mathcal{Q}_1 \equiv \mathcal{Q}_2$ („*$\mathcal{Q}_1$ und $\mathcal{Q}_2$ haben dieselbe Ausdrucksstärke*“) falls $\mathcal{Q}_1 \leqslant \mathcal{Q}_2$ und $\mathcal{Q}_2 \leqslant \mathcal{Q}_1$
- $\mathcal{Q}_1 < \mathcal{Q}_2$ (bzw. $\mathcal{Q}_2 > \mathcal{Q}_1$; „*$\mathcal{Q}_2$ ist ausdrucksstärker als $\mathcal{Q}_1$*“) falls $\mathcal{Q}_1 \leqslant \mathcal{Q}_2$ und <u>nicht</u> $\mathcal{Q}_2 \leqslant \mathcal{Q}_1$.

Folie 56

Äquivalenz der bisher eingeführten Anfragesprachen

Lemma 3.14.
Die Klassen der regelbasierten konjunktiven Anfragen, der Tableau-Anfragen und der Anfragen des konjunktiven Kalküls haben dieselbe Ausdrucksstärke.

Es gilt sogar: Jede Anfrage aus einer dieser drei Anfragesprachen kann in polynomieller Zeit in äquivalente Anfragen der beiden anderen Anfragesprachen übersetzt werden.

Beweis.

Regelbasiert $\rightsquigarrow$ Tableau:
Gegeben sei eine regelbasierte konjunktive Anfrage Q der Form

$$Ans(u) \;\leftarrow\; R_1(u_1), \ldots, R_\ell(u_\ell)$$

Sei $Q' = (\mathsf{T}', u')$ die Tableau-Anfrage mit $u' := u$ und

$$\mathsf{T}'(R) \;:=\; \quad \{u_i \;:\; i \in \{1, \ldots, \ell\} \text{ mit } R_i = R\},$$

für jedes $R \in \mathbf{S}$.
Einfaches Nachrechnen liefert für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$, dass $[\![Q']\!](\mathbf{I}) = [\![Q]\!](\mathbf{I})$ ist:

Für die Richtung „$\subseteq$" sei t ein beliebiges Tupel aus $[\![Q']\!](\mathbf{I})$. Ziel ist, zu zeigen, dass $t \in [\![Q]\!](\mathbf{I})$ ist.
Wegen $t \in [\![Q']\!](\mathbf{I})$ gibt es laut Definition der Semantik von Tableau-Anfragen eine Einbettung β von Q' in $\mathbf{I}$, so dass $t = \beta(u')$ ist.
Gemäß Definition des Begriffs einer „Einbettung" ist β eine Abbildung $\beta : \text{var}(Q') \to \mathbf{dom}$, für die gilt: $\beta(\mathsf{T}') \subseteq \mathbf{I}$, d.h. für jedes $R \in \mathbf{S}$ ist $\beta(\mathsf{T}'(R)) \subseteq \mathbf{I}(R)$.
Gemäß Konstruktion von Q' ist $\text{var}(Q') = \text{var}(Q)$ und $u' = u$. Wir wollen nun zeigen, dass für jedes Atom $R_i(u_i)$ im Rumpf von Q gilt: $\beta(u_i) \in \mathbf{I}(R_i)$.
Beachte: Gemäß der Definition der Semantik regelbasierter konjunktiver Anfragen bezeugt β dann, dass $\beta(u) \in [\![Q]\!](\mathbf{I})$ ist; und wegen $\beta(u) = \beta(u') = t$ sind wir dann fertig mit dem Beweis der Richtung „$\subseteq$".
Betrachte nun ein beliebiges Atom $R_i(u_i)$ im Rumpf von Q. Für $R := R_i$ gilt gemäß unserer Wahl des Tableaus T', dass $u_i \in \mathsf{T}'(R)$ ist. Da β eine Einbettung in $\mathbf{I}$ ist, gilt: $\beta(\mathsf{T}'(R)) \subseteq \mathbf{I}(R)$. Insbesondere ist also $\beta(u_i) \in \mathbf{I}(R)$ für $R := R_i$. Dies beendet den Beweis der Richtung „$\subseteq$".

Für die Richtung „$\supseteq$" sei t ein beliebiges Tupel aus $[\![Q]\!](\mathbf{I})$. Ziel ist, zu zeigen, dass $t \in [\![Q']\!](\mathbf{I})$ ist.
Wegen $t \in [\![Q]\!](\mathbf{I})$ gibt es laut der Definition der Semantik regelbasierter konjunktiver Anfragen eine Belegung $\beta : \text{var}(Q) \to \mathbf{dom}$, so dass $t = \beta(u)$ ist und $\beta(u_i) \in \mathbf{I}(R_i)$ für jedes $i \in \{1, \ldots, \ell\}$ gilt.
Gemäß der Konstruktion von Q' ist $\text{var}(Q') = \text{var}(Q)$ und $u' = u$, also $\beta(u') = \beta(u) = t$. Um den Beweis zu beenden, genügt es gemäß der Definition der Semantik von Tableau-Anfragen also nachzuweisen, dass β eine Einbettung von T' in $\mathbf{I}$ ist. Wir müssen daher nur noch zeigen, dass $\beta(\mathsf{T}'(R)) \subseteq \mathbf{I}(R)$ für jedes $R \in \mathbf{S}$ gilt. Betrachte dazu ein beliebiges Element $R \in \mathbf{S}$ und ein beliebiges Element $v \in \mathsf{T}'(R)$. Ziel ist, zu zeigen, dass $\beta(v) \in \mathbf{I}(R)$ ist.

Gemäß unserer Wahl des Tableaus T' gibt es für jedes $v \in \mathsf{T}'(R)$ ein $i \in \{1, \ldots, \ell\}$, so dass $v = u_i$ und $R = R_i$. Gemäß Voraussetzung wissen wir, dass gilt: $\beta(u_i) \in \mathbf{I}(R_i)$. Somit gilt also $\beta(v) \in \mathbf{I}(R)$. Dies beendet den Beweis der Richtung „$\supseteq$".

Tableau $\rightsquigarrow$ Konjunktiver Kalkül:
Gegeben sei eine Tableau-Anfrage $Q = (\mathsf{T}, u)$.
Sei $x_1, \ldots, x_k$ eine Aufzählung aller Variablen aus $\operatorname{var}(Q)$, die *nicht* im Zusammenfassungstupel u vorkommen.
Sei Q' die folgende Anfrage des Konjunktiven Kalküls:

$$\Big\{ \; (u) \; : \; \exists x_1 \cdots \exists x_k \; \bigwedge_{R \in \mathbf{S}} \; \bigwedge_{u_R \in \mathsf{T}(R)} R(u_R) \; \Big\}$$

Einfaches Nachrechnen liefert für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, dass $[\![Q']\!](\mathbf{I}) = [\![Q]\!](\mathbf{I})$ ist (Details: Übung).

Konjunktiver Kalkül $\rightsquigarrow$ Regelbasiert:
Gegeben sei eine Anfrage $Q = \{ (u) : \varphi \}$ des Konjunktiven Kalküls.
Wir nutzen Lemma 3.12, um φ in eine äquivalente CQ-Formel φ' in Normalform zu transformieren. Sei φ' von der Fom

$$\exists z_1 \cdots \exists z_s \; \big(\; R_1(u_1) \wedge \cdots \wedge R_\ell(u_\ell) \; \big)$$

Sei Q' die regelbasierte konjunktive Anfrage

$$\mathit{Ans}(u) \;\; \leftarrow \;\; R_1(u_1), \cdots, R_\ell(u_\ell)$$

Einfaches Nachrechnen liefert für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, dass $[\![Q']\!](\mathbf{I}) = [\![Q]\!](\mathbf{I})$ (Details: Übung). $\square$

Folie 57

Beispiel zur Übersetzung von Anfragen

Betrachte die Anfrage

$$\begin{aligned} \mathit{Ans}(x, y) \;\; \leftarrow \;\; & \mathit{Programm}(z_1, x, z_2), \\ & \mathit{Filme}(x, y, z_3), \\ & \mathit{Filme}(z_4, y, \text{“George Clooney”}), \\ & \mathit{Filme}(z_5, \text{“George Clooney”}, y) \end{aligned}$$

Unser Beweis von Lemma 3.14 übersetzt diese Anfrage in die Tableau-Anfrage $Q = (\mathsf{T}, u)$ mit Zusammenfassungstupel $u = (x, y)$ und folgendem Tableau T:

Programm

Kino	Titel	Zeit
z_1	x	z_2

Filme

Titel	Regie	Schauspieler
x	y	z_3
z_4	y	"George Clooney"
z_5	"George Clooney"	y

Diese Tableau-Anfrage wiederum wird durch unseren Beweis übersetzt in die folgende Anfrage des konjunktiven Kalküls:

$$\left\{ \begin{array}{lll} (x,y) \;:\; \exists z_1 \cdots \exists z_5 \; \big(& \mathit{Programm}(z_1, x, z_2) \wedge \\ & \mathit{Filme}(x, y, z_3) \wedge \\ & \mathit{Filme}(z_4, y, \text{“George Clooney”}) \wedge \\ & \mathit{Filme}(z_5, \text{“George Clooney”}, y) \; \big) \end{array} \right\}$$

Diese Anfrage ist in Normalform und wird durch unseren Beweis übersetzt in die regelbasierte Anfrage, mit der wir dieses Beispiel begonnen haben.

Folie 58

Die Kalkül-Anfrage

$$\left\{ \begin{array}{ll} (x,y) \;:\; & \exists z \exists z' \; \mathit{Programm}(z, x, z') \wedge \\ & \exists z \; \mathit{Filme}(x, y, z) \wedge \\ & \exists z \; \mathit{Filme}(z, y, \text{“George Clooney”}) \wedge \\ & \exists z \; \mathit{Filme}(z, \text{“George Clooney”}, y) \end{array} \right\}$$

die <u>nicht</u> in Normalform ist, wird durch unseren Beweis zunächst in die Normalform-Anfrage

$$\left\{ \begin{array}{lll} (x,y) \;:\; \exists z_1 \cdots \exists z_5 \; \big(& \mathit{Programm}(z_1, x, z_2) \wedge \\ & \mathit{Filme}(x, y, z_3) \wedge \\ & \mathit{Filme}(z_4, y, \text{“George Clooney”}) \wedge \\ & \mathit{Filme}(z_5, \text{“George Clooney”}, y) \; \big) \end{array} \right\}$$

übersetzt, die dann wiederum in die regelbasierte Anfrage übersetzt wird, mit der wir dieses Beispiel begonnen hatten.

Zur Info: Die im obigen Beispiel betrachtete Anfrage fragt nach Titel und Regisseur von zur Zeit laufenden Filmen, deren Regisseur sowohl als Schauspieler als auch als Regisseur schon mal mit George Clooney gearbeitet hat.

Folie 59

Einige Erweiterungen der Anfragesprachen

(1) *Test auf „Gleichheit"* von Variablen zulassen

(2) Hintereinanderausführung (*Komposition*) mehrerer Anfrage zulassen

Zu (1):

Zum Beispiel lässt sich die Anfrage

(6): Welche (je 2) Schauspieler haben schon in Filmen gespielt, in denen der jeweils andere Regie geführt hat?

ausdrücken durch

$$Ans(y_1, y_2) \leftarrow Filme(x_1, y_1, z_1),\ Filme(x_2, y_2, z_2),\ y_1{=}z_2,\ y_2{=}z_1$$

aber auch, äquivalent, durch

$$Ans(y_1, y_2) \leftarrow Filme(x_1, y_1, y_2),\ Filme(x_2, y_2, y_1)$$

Folie 60

Regelbasierte konjunktive Anfragen mit „="

Prinzipiell:
Im Rumpf von Regeln auch Atome der Form „$x{=}y$" und „$x{=}c$" zulassen, für beliebige Variablen $x, y \in \mathbf{var}$ und Konstanten $c \in \mathbf{dom}$.
$\leadsto$ *regelbasierte konjunktive Anfragen mit „="*
$\leadsto$ *Konjunktiver Kalkül mit „="*

Aber Vorsicht: Die Regel Q der Form

$$Ans(x, y) \ \leftarrow\ R(x),\ y{=}z$$

ausgewertet in einer Datenbank $\mathbf{I}$ mit $\mathbf{I}(R) = \{a\}$, liefert als Ergebnis

$$[\![Q]\!](\mathbf{I}) \ = \ \{\ (a, d)\ :\ d \in \mathbf{dom}\ \} \ = \ \{a\} \times \mathbf{dom}$$

Da **dom** *unendlich viele Elemente* hat, ist $[\![Q]\!](\mathbf{I})$ also eine „unendliche Relation"; per Definition, sind *als Relationen aber nur endliche Mengen erlaubt.*

Daher machen wir eine syntaktische Einschränkung auf *bereichsbeschränkte* Anfragen, um zu garantieren, dass das Ergebnis einer Anfrage stets eine endliche Relation ist.

Folie 61

Bereichsbeschränkte konjunktive Anfragen mit „=“

Präzise:
Jede Variable x, die im Rumpf der Regel vorkommt, muss auch in einem im Rumpf der Regel stehenden Atom der Form $R(u)$ oder $x{=}c$, für ein $c \in \mathbf{dom}$, vorkommen — oder es muss eine Kette von Gleichheits-Atomen der Form $x{=}y_1, y_1{=}y_2, \ldots, y_j{=}z$ und ein Atom der Form $z{=}c$ für eine Konstante $c \in \mathbf{dom}$ geben oder ein Atom der Form $R(u)$ im Rumpf der Regel geben, so dass die Variable z im freien Tupel u vorkommt.

Regelbasierte konjunktive Anfragen mit „=“, die diese Bedingung erfüllen, nennen wir *bereichsbeschränkt.*

Beobachtung 3.15. *Jede bereichsbeschränkte regelbasierte konjunktive Anfrage mit „=“ ist entweder unerfüllbar oder äquivalent zu einer regelbasierten konjunktiven Anfrage (ohne „=“). (Details siehe Übung.)*

Analog wird die Klasse $\mathsf{CQ}^=$ aller *bereichsbeschränkten Formeln des konjunktiven Kalküls mit „=“* definiert. *Beispiel für eine* $\mathsf{CQ}^=$*-Anfrage, die nicht erfüllbar ist:*

$$\{\ (x)\ :\ \ \big(\ R(x)\ \wedge\ x{=}a\ \wedge\ x{=}b\ \big)\ \}$$

wobei a und b zwei verschiedene Elemente aus **dom** sind.

Folie 62

Hintereinanderausführung mehrerer Anfragen

Im Folgenden widmen wir uns einer Erweiterung der konjunktiven Anfragen, die die Komposition mehrerer Anfragen ermöglicht, so dass eine Anfrage auf das Resultat einer (oder mehrerer) Anfragen angewendet werden kann.

Folie 63

Regelbasierte konjunktive Programme

Definition 3.16. Sei **S** ein Datenbankschema.

Ein *regelbasiertes konjunktives Programm* über **S** (mit oder ohne „=") hat die Form

$$\begin{array}{rcl} S_1(u_1) & \leftarrow & \mathit{Rumpf}_1 \\ S_2(u_2) & \leftarrow & \mathit{Rumpf}_2 \\ \vdots & \vdots & \vdots \\ S_m(u_m) & \leftarrow & \mathit{Rumpf}_m \end{array}$$

wobei $m \geqslant 1$, $S_1, \ldots, S_m$ sind paarweise verschiedene Relationsnamen aus $\mathbf{rel} \setminus \mathbf{S}$ und für jedes $i \in \{1, \ldots, m\}$ gilt:

$$Q_i \ := \quad S_i(u_i) \leftarrow \mathit{Rumpf}_i$$

ist eine regelbasierte konjunktive Anfrage über $\big(\mathbf{S} \cup \{S_j \ : \ 1 \leqslant j < i\}\big)$ (mit oder ohne „=").

Die Relationsnamen aus **S**, die im Programm vorkommen, heißen extensionale Prädikate (*edb-Prädikate*). Die Relationsnamen $S_1, \ldots, S_m$ heißen intensionale Prädikate (*idb-Prädikate*).

Folie 64

Semantik regelbasierter konjunktiver Programme

Ausgewertet über einer Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ beschreibt das obige Programm P die Relationen

$$[\![P(S_1)]\!](\mathbf{I}), \quad \ldots, \quad [\![P(S_m)]\!](\mathbf{I}),$$

die induktiv für alle $i \in \{1, \ldots, m\}$ wie folgt definiert sind:

$$[\![P(S_i)]\!](\mathbf{I}) \ := \ [\![Q_i]\!](\mathbf{J}_{i-1})$$

wobei $\mathbf{J}_{i-1}$ die Erweiterung der Datenbank **I** um die Relationen $\mathbf{J}(S_j) := [\![P(S_j)]\!](\mathbf{I})$, für alle j mit $1 \leqslant j < i$ ist.

Folie 65

Äquivalenz von Regeln und Programmen

Beobachtung 3.17. *Für jedes regelbasierte konjunktive Programm P über einem Relationsschema* **S** *(mit oder ohne „=") und jedes idb-Prädikat S von P gibt es eine regelbasierte konjunktive Anfrage Q (mit „=") über* **S**, *so dass*

$$[\![Q]\!](\mathbf{I}) \ = \ [\![P(S)]\!](\mathbf{I})$$

für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$.

Beispiel:
Sei $\mathbf{S} = \{Q, R\}$. Betrachte folgendes regelbasierte konjunktive Programm P:

$$\begin{array}{rcl} S_1(x,z) & \leftarrow & Q(x,y),\ R(y,z,w) \\ S_2(x,y,z) & \leftarrow & S_1(x,w),\ R(w,y,v),\ S_1(v,z) \end{array}$$

Die von P durch S_2 definierte Anfrage ist äquivalent zu

$$\begin{array}{rcl} S_2(x,y,z) & \leftarrow & x{=}x',\ w{=}z',\ Q(x',y'),\ R(y',z',w'), \\ & & R(w,y,v), \\ & & v{=}x'',\ z{=}z'',\ Q(x'',y''),\ R(y'',z'',w'') \end{array}$$

3.2 Auswertungskomplexität

Folie 66

Auswertungskomplexität konjunktiver Anfragen

Auswertungsproblem für CQ	*(kombinierte Komplexität)*
Eingabe:	Anfrage Q des konjunktiven Kalküls, Datenbank $\mathbf{I}$ (von einem zu Q passenden Schema $\mathbf{S}$)
Aufgabe:	Berechne $[\![Q]\!](\mathbf{I})$

Schön wäre: Algorithmus, der zur Lösung dieses Problems mit Zeit polynomiell in „Größe der Eingabe + Größe der Ausgabe“ auskommt.

Frage: **Gibt es einen solchen Algorithmus?**

Größe der Eingabe: $k + n$, wobei

- $k := \|Q\|$ die Länge der Anfrage (betrachtet als Wort über dem Alphabet $\mathbf{dom} \cup \mathbf{var} \cup \mathbf{S} \cup \{\,\exists,\ \wedge,\ (,\ ),\ \{,\ \},\ :,\ ,\ \}$)

  Die Länge von konjunktiven regelbasierten Anfragen bzw. von Anfragen anderer Anfragesprachen wird analog definiert.

- $n := \|\mathbf{I}\|$ die Größe der Datenbank, also

$$n \ := \ \|\mathbf{I}\| \ := \ \sum_{R \in \mathbf{S}} \|\mathbf{I}(R)\| \quad \text{wobei} \quad \|\mathbf{I}(R)\| \ := \ \mathrm{ar}(R) \cdot \underbrace{|\mathbf{I}(R)|}_{\text{Anzahl Tupel in } \mathbf{I}(R)}$$

Größe der Ausgabe:
$\| \, [\![Q]\!](\mathbf{I}) \, \| \ := $ „Stelligkeit“ · „Anzahl Tupel im Ergebnis“

Folie 67

Auswertung konjunktiver Anfragen

Proposition 3.18.
Das Auswertungsproblem für CQ *lässt sich in Zeit* $\mathcal{O}\big((k+n)^k\big)$ *lösen.*

Beweis: siehe unten.

Bemerkung: Das ist exponentiell in der Länge der Anfrage.
Frage: Geht das effizienter?

Beweis von Proposition 3.18:
Als Eingabe erhalten wir eine Datenbank $\mathbf{I}$ und eine Anfrage Q der Form

$$\{\ (e_1,\dots,e_r)\ \ :\ \ \varphi(x_1,\dots,x_m)\ \}\,,$$

wobei $(e_1,\dots,e_r)$ ein freies Tupel ist, dessen Variablen genau die Variablen aus $\{x_1,\dots,x_m\}$ sind, und φ eine CQ-Formel mit $\text{frei}(\varphi)=\{x_1,\dots,x_m\}$, die o.B.d.A. von der Form

$$\exists x_{m+1}\cdots\exists x_s\ \bigwedge_{i=1}^{\ell} R_i(u_i)$$

ist, wobei jedes u_i ein freies Tupel mit Variablen aus $\{x_1,\dots,x_s\}$ ist.
Unser Algorithmus geht bei Eingabe von Q und $\mathbf{I}$ wie folgt vor:

(1) Bestimme $\text{adom}(Q,\mathbf{I})$.

(2) Für jedes $\overline{b}=(b_1,\dots,b_s)\in\big(\text{adom}(Q,\mathbf{I})\big)^s$ tue Folgendes:

  (2.1) Sei $\beta:\{x_1,\dots,x_s\}\to\mathbf{dom}$ die Belegung mit $\beta(x_i)=b_i$ für jedes $i\in\{1,\dots,s\}$.

  (2.2) $\texttt{test} := \texttt{true}$

  (2.3) Für $i=1,\dots,\ell$ tue Folgendes:

    (2.3.1) Teste, ob $\beta(u_i)\in\mathbf{I}(R_i)$ gilt.

    (2.3.2) Falls nicht, so setze $\texttt{test} := \texttt{false}$.

  (2.4) Falls $\texttt{test} = \texttt{true}$, so gib das Tupel $(\beta(e_1),\dots,\beta(e_r))$ aus.

Unter Verwendung von Proposition 3.10 („$\text{adom}(\llbracket Q\rrbracket(\mathbf{I})\subseteq\text{adom}(Q,\mathbf{I})$“) sieht man leicht, dass dieser Algorithmus genau die Tupel aus $\llbracket Q\rrbracket(\mathbf{I})$ ausgibt.
Eine Laufzeitanalyse für $k:=\|Q\|$, $n:=\|I\|$ und $N:=k+n$ ergibt, dass

- für Zeile (1) nur $\mathcal{O}(N \log N)$ Schritte gemacht werden, um eine Liste aller beim Lesen von Q und $\mathbf{I}$ gefundenen Konstanten zu erstellen (der Faktor $\log N$ wird benutzt, um die Liste zu sortieren und Duplikate zu eliminieren),
- die Schleife in Zeile (2) höchstens N^s mal durchlaufen wird und
- bei jedem Schleifendurchlauf höchstens $s + 1 + \ \ell \cdot (n+1)\ + 1 + r$ Schritte gemacht werden.

Eine nähere Betrachtung von Q zeigt, dass

$$k \quad = \quad \|Q\| \quad \geqslant \quad 2 + 2r + 1 + \ 2(s{-}m) \ + \ \ell{\cdot}4$$

ist. Außerdem ist $r \geqslant m$, und daher ist $r + (s{-}m) \geqslant s$, und somit ist

$$k \quad \geqslant \quad 3 + r + (s{-}m) + s + 4\ell \quad \geqslant \quad s + 1 + \ell + 1 + r.$$

Die Gesamtlaufzeit unseres Algorithmus ist also

$$\begin{aligned}
\mathcal{O}(N \log N) \ + \ N^s \cdot \big(s + 1 + \ell + 1 + r + \ell{\cdot}n\big) \quad &\leqslant \quad \mathcal{O}(N \log N) \ + \ N^s \cdot \big(k + \ell{\cdot}n\big) \\
&\leqslant \quad \mathcal{O}(N^2) \ + \ N^s \cdot N^2 \\
&\leqslant \quad \mathcal{O}(N^{s+2}) \\
&\leqslant \quad \mathcal{O}(N^k) \\
&\leqslant \quad \mathcal{O}((k+n)^k).
\end{aligned}$$

Dies beendet den Beweis von Proposition 3.18. $\square$

Folie 68

Boolesche Anfragen

- *Zur Erinnerung:*

  Boolesche Anfragen sind „ja / nein“-Anfragen,
  d.h. Anfragen, deren Ergebnis die Stelligkeit 0 hat.

- *Klar:*

  Falls wir zeigen können, dass das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls schwierig ist, so ist es auch für allgemeine Anfragen des konjunktiven Kalküls schwierig.

- *Wir werden sehen, dass umgekehrt aber auch gilt:*

  Falls wir einen Algorithmus haben, der das Auswertungsproblem für *Boolesche* Anfragen des konjunktiven Kalküls löst, so können wir diesen Algorithmus verwenden, um das Auswertungsproblem für *beliebige* Anfragen des konjunktiven Kalküls zu lösen.

Folie 69

Algorithmen mit Taktung $f(k, n)$

Definition 3.19. Sei $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, sei $\mathcal{A}$ *eine Datenbankanfragesprache* und sei $\mathbb{A}$ *irgendein Algorithmus*, der bei allen Eingaben terminiert.

Wir sagen:

> *das Auswertungsproblem für $\mathcal{A}$ kann unter Rückgriff auf $\mathbb{A}$ mit Taktung $f(k, n)$ gelöst werden,*

falls es einen Algorithmus $\mathbb{B}$ gibt, der bei Eingabe einer Anfrage Q aus $\mathcal{A}$ und einer Datenbank $\mathbf{I}$ nach und nach genau die Tupel aus $[\![Q]\!](\mathbf{I})$ ausgibt (und zwar jedes nur einmal) und

- vor der Ausgabe des ersten Tupels,
- zwischen der Ausgabe von je zwei aufeinanderfolgenden Tupeln,
- nach der Ausgabe des letzten Tupels

je höchstens $f(\|Q\|, \|\mathbf{I}\|)$ viele Elementarschritte oder Schritte, in denen der Algorithmus $\mathbb{A}$ aufgerufen wird, macht.

Folie 70

Boolesche Anfragen $\rightsquigarrow$ beliebige Anfragen

Theorem 3.20. *Sei $\mathbb{A}$ ein Algorithmus, der das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls löst.*

Dann gibt es einen Algorithmus $\mathbb{B}$, der das Auswertungsproblem für (beliebige) Anfragen des konjunktiven Kalküls unter Rückgriff auf $\mathbb{A}$ mit Taktung $\mathcal{O}(k^3 \cdot n \cdot \log n)$ löst.

Beweis: siehe unten.

Folgerung: Falls wir das Auswertungsproblem für *Boolesche* Anfragen des konjunktiven Kalküls effizient lösen können, dann können wir es auch für *beliebige* Anfragen des konjunktiven Kalküls effizient lösen.

Beweis von Theorem 3.20:

Wir betrachten o.B.d.A. nur Anfragen der Form

$$\{\ (x_1,\dots,x_r,c_1,\dots,c_m)\ \ :\ \ \varphi(x_1,\dots,x_r)\ \}$$

wobei für $r,m,s,\ell \geqslant 0$ gilt:

- $x_1,\dots,x_r$ sind paarweise verschiedene Variablen.
  Die Zahl r wird im Folgenden *Stelligkeit der Anfrage* genannt.
- $c_1,\dots,c_m$ sind (nicht notwendigerweise unterschiedliche) Konstanten.
- $\text{frei}(\varphi) = \{x_1,\dots,x_r\}$.
- φ ist von der Form $\exists x_{r+1}\cdots\exists x_s\ \bigwedge_{i=1}^{\ell} R_i(u_i)$, wobei jedes u_i ein freies Tupel mit Variablen aus $\{x_1,\dots,x_s\}$ ist.

Bei Eingabe einer Datenbank $\mathbf{I}$ und einer Anfrage Q dieser Form geht unser Algorithmus $\mathbb{B}$ in 2 Phasen vor.

Phase 1: Ermittle die Stelligkeit r der Anfrage und erstelle eine Liste $a_1,\dots,a_N$ aller Elemente aus $\text{adom}(Q,\mathbf{I})$, für $N := |\,\text{adom}(Q,\mathbf{I})|$.

Für $k := \|Q\|$ und $n := \|\mathbf{I}\|$ gilt $N \leqslant k+n$. Und Phase 1 benötigt nur Zeit $\mathcal{O}((k+n)\log(k+n))$ (wir sortieren die beim Lesen von Q und $\mathbf{I}$ erstellte Liste von Konstanten, um Duplikate zu eliminieren).

Phase 2: Berechne $[\![Q]\!](\mathbf{I})$.
Unter Verwendung von Proposition 3.10 erhalten wir, dass Folgendes gilt:

$$[\![Q]\!](\mathbf{I})\ \subseteq\ \{a_1,\dots,a_N\}^r \times \{c_1\} \times \cdots \times \{c_m\}.$$

Eine erste und noch nicht ganz so gute Idee, $[\![Q]\!](\mathbf{I})$ zu berechnen, ist wie folgt vorzugehen.

Für alle $b_r \in \{a_1,\dots,a_N\}$ tue Folgendes:
  Für alle $b_{r-1} \in \{a_1,\dots,a_N\}$ tue Folgendes:
    $\ddots$
      Für alle $b_1 \in \{a_1,\dots,a_N\}$ tue Folgendes:
        (1) Starte Algorithmus $\mathbb{A}$ mit DB $\mathbf{I}$ und Anfrage $Q_{(b_1,\dots,b_r)}\ :=\ \{\,()\ :\ \varphi(x_1/b_1,\dots,x_r/b_r)\,\}$
        (2) Falls $\mathbb{A}$ „ja“ ausgibt, so gib das Tupel $(b_1,\dots,b_r,c_1,\dots,c_m)$ aus.

Man kann sich leicht davon überzeugen, dass dieser Algorithmus genau die Tupel aus $[\![Q]\!](\mathbf{I})$ ausgibt. Das Problem ist allerdings, dass vor der Ausgabe des ersten Tupels (und zwischen der Ausgabe zweier Tupel und nach der Ausgabe des letzten Tupels) evtl. zu viel Zeit vergeht. Wenn beispielsweise $(a_N, \ldots, a_N, c_1, \ldots, c_m)$ das *einzige* Tupel in $[\![Q]\!](\mathbf{I})$ ist, dann gibt der obige Algorithmus dieses erst im allerletzten Schleifendurchlauf aus, also nach etwa N^r Schritten. Das „r“, das hier im Exponenten steht, hängt von der Anfrage Q ab. Die hier erzielte Taktung von N^r ist u.U. also viel größer als die im Theorem behauptete Taktung von maximal $k^3 \cdot n \cdot \log n$.

Lösungsansatz: Die Abarbeitung der ineinandergeschachtelten Schleifen des obigen Algorithmus können wir uns als einen Tiefensuche-Durchlauf durch den folgenden *Suchbaum* vorstellen: Der Suchbaum ist ein vollständiger Baum der Tiefe r, bei dem jeder Knoten, der sich auf einer Tiefe $< r$ befindet, genau N Kinder hat (und diese sind mit $a_1, \ldots, a_N$ beschriftet). Die N Kinder $v_1, \ldots, v_N$ der Wurzel stellen alle möglichen Belegungen für b_r dar; jedes v_i hat wieder N Kinder, die alle möglichen Belegungen für b_{r-1} darstellen, usw.
Unser neuer Lösungsansatz soll nun sicherstellen, dass nur solche Kanten des Suchbaums durchlaufen werden, die tatsächlich zu einem Tupel aus $[\![Q]\!](\mathbf{I})$ führen. Dies kann man dadurch bewerkstelligen, dass man vor dem Durchlaufen einer Kante eine Boolesche Anfrage stellt, die nachfragt, ob es im Teilbaum unterhalb dieser Kante ein Tupel gibt, das zu $[\![Q]\!](\mathbf{I})$ führt. Diese Methode ist unter dem Namen *flashlight search* bekannt.
Dies wird erreicht, indem wir die folgende Prozedur $\mathbb{C}$ mit Eingabe r, Q starten.

Prozedur $\mathbb{C}(r, Q)$:

Falls $r = 0$:

(1) Starte Algorithmus $\mathbb{A}$ mit DB $\mathbf{I}$ und Boolescher Anfrage
$Q^{\text{Bool}} \;:=\; \{\, () \;:\; \varphi \,\}$

(2) Falls $\mathbb{A}$ „ja“ ausgibt, so gib das Tupel $(c_1, \ldots, c_m)$ aus und halte an.

Falls $r > 0$:

Für jedes $b_r \in \{a_1, \ldots, a_N\}$ tue Folgendes:

(3) Starte Algorithmus $\mathbb{A}$ mit DB $\mathbf{I}$ und Boolescher Anfrage
$Q^{\text{ex.Lsg}} \;:=\; \{\, () \;:\; \exists x_1 \cdots \exists x_{r-1}\, \varphi(x_1, \ldots, x_{r-1}, x_r/b_r) \,\}$

(4) Falls $\mathbb{A}$ „ja“ ausgibt, so starte einen rekursiven Aufruf der Prozedur $\mathbb{C}$ mit Eingabe r' und Q', für

- $r' := r{-}1$
- $Q' := Q^{\text{berechne Lsgen}}_{b_r} :=$
  $\{\, (x_1,\ldots,x_{r-1},b_r,c_1,\ldots,c_m) \ : \ \varphi(x_1,\ldots,x_{r-1},x_r/b_r)\,\}$

Per Induktion nach r zeigen wir, dass Folgendes gilt:

$(1)_r$: $\mathbb{C}(r,Q)$ gibt genau die Tupel aus $[\![Q]\!](\mathbf{I})$ aus.

$(2)_r$: $\mathbb{C}(r,Q)$ hat Taktung höchstens $(r{+}1)\cdot(2k{+}1)\cdot(N{+}1)$, d.h. vor der Ausgabe des ersten Tupels, zwischen der Ausgabe zweier Tupel und nach der Ausgabe des letzten Tupels führt die Prozedur jeweils höchstens $(r{+}1)\cdot(2k{+}1)\cdot(N{+}1)$ Elementarschritte oder Aufrufe von $\mathbb{A}$ durch.

Induktionsanfang für $r = 0$:
Aussage $(1)_0$ gilt offensichtlicherweise.
Außerdem macht die Prozedur maximal k Elementarschritte, um die Anfrage Q^{Bool} zu konstruieren, startet danach einen Aufruf von $\mathbb{A}$ und macht ggf. nochmal m Elementarschritte, um das Tupel $(c_1,\ldots,c_m)$ auszugeben. Die Gesamtzahl der Schritte ist also höchstens $k+1+m \leqslant 2k+1 \leqslant (r{+}1)(2k{+}1)(N{+}1)$. Somit ist Aussage $(2)_0$ erfüllt.

Induktionsschritt für $r > 0$:
Die Gültigkeit der Aussage $(1)_r$ folgt direkt aus der Konstruktion der Prozedur $\mathbb{C}$, der Induktionsannahme und der Tatsache, dass $\text{adom}([\![Q]\!](\mathbf{I})) \subseteq \text{adom}(Q,\mathbf{I})$ ist (Details: Übung).
Zum Nachweis der Gültigkeit der Aussage $(2)_r$ beachte Folgendes:
Vor der Ausgabe des ersten Tupels, zwischen der Ausgabe von zwei Tupeln und nach der Ausgabe des letzten Tupels macht die Prozedur je höchstens

(A) N Schritte, in denen Zeile (3) aufgerufen wird und die Ausgabe „nein“ liefert.

Bei jedem solchen Schritt fallen maximal k Elementarschritte zur Konstruktion der Anfrage $Q^{\text{ex.Lsg}}$ an, und 1 Schritt zum Aufruf von $\mathbb{A}$.

(B) 1 Schritt, in dem Zeile (3) aufgerufen wird und die Ausgabe „ja“ liefert, so dass danach Zeile (4) aufgerufen wird (und wir wissen, dass während der Abarbeitung dieses Aufrufs mindestens ein Tupel ausgegeben wird).

Dabei werden höchstens folgende Schritte gemacht:

- k Elementarschritte zur Konstruktion von $Q_{b_r}^{\text{ex.Lsg}}$
- 1 Schritt zum Aufruf von $\mathbb{A}$
- k Elementarschritte zur Konstruktion von $Q_{b_r}^{\text{berechne Lsgen}}$
- nach dem Start von $\mathbb{C}(r{-}1, Q_{b_r}^{\text{berechne Lsgen}})$ gemäß Induktionsannahme maximal $r{\cdot}(2k{+}1){\cdot}(N{+}1)$ Schritte bis zur Ausgabe des ersten Tupels, zwischen der Ausgabe von 2 Tupeln und nach der Ausgabe des letzten Tupels.

Bei (A) werden also insgesamt höchstens $(k{+}1) \cdot N$ Schritte gemacht, und bei (B) beträgt die „Taktung" höchstens $2k + 1 + r(2k{+}1)(N{+}1)$. Insgesamt haben wir also eine „Taktung" von höchstens

$$\begin{aligned} & (k{+}1)N + 2k + 1 + r{\cdot}(2k{+}1){\cdot}(N{+}1) \\ \leqslant\ & (2k{+}1){\cdot}(N{+}1) \ + \ r{\cdot}(2k{+}1){\cdot}(N{+}1) \\ \leqslant\ & (r{+}1) \cdot (2k{+}1) \cdot (N{+}1) \end{aligned}$$

Schritten. Somit ist also Aussage $(2)_r$ bewiesen.

Insgesamt erhalten wir, dass unser Algorithmus $\mathbb{B}$ das Auswertungsproblem unter Rückgriff auf $\mathbb{A}$ mit Taktung

$$\begin{aligned} \mathcal{O}(\underbrace{(k{+}n){\cdot}\log(k{+}n)}_{\text{Phase 1}} + \underbrace{r{\cdot}k{\cdot}N}_{\text{Phase 2}}) \ &\leqslant\ \mathcal{O}((k{+}n)(\log(k) + \log(n)) \ + \ k^2{\cdot}(k{+}n)) \\ &\leqslant\ \mathcal{O}(k^3{\cdot}n{\cdot}\log n) \end{aligned}$$

löst. Dies beendet den Beweis von Theorem 3.20. □

Folie 71

Die Komplexitätsklasse NP

Zur Erinnerung:

- Komplexitätsklassen bestehen aus *Entscheidungsproblemen*, d.h. Problemen mit *„ja/nein"-Ausgabe*

  $\rightsquigarrow$ das *Auswertungsproblem für Boolesche Anfragen passt gut* zum Konzept der klassischen Komplexitätstheorie;

  $\rightsquigarrow$ das Auswertungsproblem für beliebige Anfragen nicht

- Ein Entscheidungsproblem B gehört zur Klasse NP, falls es einen nichtdeterministischen Algorithmus $\mathbb{B}$ gibt, der eine *durch ein Polynom beschränkte worst-case Laufzeit* besitzt und der bei Eingabe jeder für B zulässigen Eingabe x Folgendes leistet:

- Falls x eine „ja"-Instanz von B ist, so besitzt $\mathbb{B}$ bei Eingabe x mindestens einen Berechnungspfad, der mit der Ausgabe „ja" endet.
- Falls x eine „nein"-Instanz von B ist, so endet *jeder* Berechnungspfad von $\mathbb{B}$ mit der Ausgabe „nein".

„Nichtdeterministische Algorithmen" werden in der Komplexitätstheorie i.d.R. durch nichtdeterministische Turingmaschinen modelliert; ein „Berechnungspfad" entspricht dann einem *Lauf* der Turingmaschine.

Folie 72

NP-Vollständigkeit

Zur Erinnerung:

Ein Entscheidungsproblem B heißt NP-*vollständig* (bzgl. Polynomialzeit-Reduktionen), falls gilt:

(1) $B \in \mathrm{NP}$ und

(2) B *ist* NP-*hart*, d.h. für jedes Problem $A \in \mathrm{NP}$ gibt es eine *Polynomialzeit-Reduktion* f von A auf B (kurz: $f : A \leqslant_p B$)

Folie 73

Exkurs: Reduktionen

- Wir wollen einen Algorithmus $\mathbb{A}$ finden, der ein Entscheidungsproblem A löst.

  Beispiel: *A ist das Problem, bei Eingabe eines Graphen G und einer Zahl k zu entscheiden, ob G eine Clique der Größe k besitzt.*[1]

  Mit M_A bezeichnen wir im Folgenden die Menge aller für das Problem A zulässigen Eingaben.

- Nehmen wir mal an, wir hätten bereits einen Algorithmus $\mathbb{B}$ konstruiert, der ein Entscheidungsproblem B löst.

[1] Eine Clique der Größe k in $G = (V, E)$ ist eine Menge $C \subseteq V$ mit $|C| = k$, so dass für alle $u, v \in C$ mit $u \neq v$ gilt: u und v sind in G durch eine Kante verbunden.

Beispiel: *B ist das Problem, bei Eingabe eines Graphen H und einer Zahl ℓ zu entscheiden, ob H eine unabhängige Menge der Größe ℓ besitzt.*[2]

Mit M_B bezeichnen wir die Menge aller für das Problem B zulässigen Eingaben.

Folie 74

- **Idee zur Konstruktion von $\mathbb{A}$:** Nutze den Algorithmus $\mathbb{B}$.

  Wir bekommen eine für A zulässige Eingabe x und wollen entscheiden, ob x eine „ja"-Instanz für A ist.

  Finde eine **effizient berechenbare** Funktion $f : M_A \to M_B$, so dass für alle Eingaben x für A gilt:

x ist eine „ja"-Instanz für das Problem A	$\iff$	$f(x)$ ist eine „ja"-Instanz für das Problem B

  Dann kann Algorithmus $\mathbb{A}$ bei Eingabe x wie folgt vorgehen:

  1. Berechne $y := f(x)$,
  2. starte Algorithmus $\mathbb{B}$ mit Eingabe y,
  3. gib das aus, was $\mathbb{B}$ ausgibt.

  Beispiel:
  $f(G,k) = \big(\overline{G}, k\big)$, *wobei $\overline{G}$ das **Komplement** des Graphen G ist.*

- *f heißt **Reduktion** von A auf B.* Kurz: $f : A \leqslant B$.

Folie 75

- Eine Reduktion f von A auf B heißt *Polynomialzeit-Reduktion* (kurz: $f : A \leqslant_p B$), falls es einen (deterministischen) Polynomialzeit-Algorithmus gibt, der bei Eingabe von $x \in M_A$ den Wert $f(x)$ berechnet.

[2] Eine unabhängige Menge der Größe ℓ in $H = (V,E)$ ist eine Menge $U \subseteq V$ mit $|U| = \ell$, so dass für alle $u,v \in U$ mit $u \neq v$ gilt: u und v sind in H *nicht* durch eine Kante verbunden.

- Wenn es eine Polynomialzeit-Reduktion f von A auf B gibt, wissen wir, dass Folgendes gilt:
  - Ein Algorithmus, der B löst, kann zum Lösen von A verwendet werden. Das Problem A ist also „höchstens so schwer“ wie das Problem B.
  - Das heißt umgekehrt aber auch, dass das Problem B „mindestens so schwer“ wie das Problem A ist.

Folie 76

NP-Vollständigkeit

Zur Erinnerung:

- Ein Entscheidungsproblem B heißt NP-*vollständig* (bzgl. Polynomialzeit-Reduktionen), falls gilt:
  (1) $B \in \mathrm{NP}$ und
  (2) B *ist* NP-*hart*, d.h. für jedes Problem $A \in \mathrm{NP}$ gibt es eine *Polynomialzeit-Reduktion* f von A auf B (kurz: $f : A \leqslant_p B$)
- Eine *Polynomialzeit-Reduktion* f von A auf B ist eine (deterministisch) in polynomieller Zeit berechenbare Funktion, die jede zum Problem A passende Eingabe x auf eine zum Problem B passende Eingaben $f(x)$ abbildet, so dass gilt

  x ist eine „ja“-Instanz für $A \iff f(x)$ ist eine „ja“-Instanz für B.

  Anschaulich bedeutet $A \leqslant_p B$, dass A „höchstens so schwer“ wie B ist. NP-vollständige Probleme sind also „die schwierigsten Probleme“ in NP.
- Der Begriff *Polynomialzeit-Reduktion* ist so definiert, dass Folgendes gilt, wobei P die Klasse aller Entscheidungsprobleme bezeichnet, die deterministisch in Polynomialzeit lösbar sind:
  - $A \leqslant_p B$ und $B \in \mathrm{P} \implies A \in \mathrm{P}$
  - $A \notin \mathrm{P}$ und $A \leqslant_p B \implies B \notin \mathrm{P}$
  - A NP-hart und $A \leqslant_p B \implies B$ NP-hart.

Folie 77

Auswertungskomplexität konjunktiver Anfragen

Theorem 3.21 (Chandra, Merlin, 1977). *Das Auswertungsproblem (kombinierte Komplexität) für Boolesche regelbasierte konjunktive Anfragen ist* NP*-vollständig.*

Beweis:
Zugehörigkeit zu NP*:* Bei Eingabe einer DB $\mathbf{I}$ und einer Booleschen Anfrage Q der Form

$$Ans() \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$$

mit $\text{Var}(Q) = \{x_1, \ldots, x_s\}$ kann ein nichtdeterministischer Algorithmus wie folgt vorgehen:

(1) Bestimme $\text{adom}(Q, \mathbf{I})$

(2) „Rate“ (nichtdeterministisch) Elemente $b_1, \ldots, b_s$ aus $\text{adom}(Q, \mathbf{I})$.
Sei $\beta : \text{Var}(Q) \to \mathbf{dom}$ die Belegung mit $\beta(x_j) := b_j$ f.a. $j \in \{1, \ldots, s\}$.

(3) Teste, ob für alle $i \in \{1, \ldots, \ell\}$ gilt: $\beta(u_i) \in \mathbf{I}(R_i)$.

(4) Falls „ja“, so STOPP mit Ausgabe „ja“;
ansonsten STOPP mit Ausgabe „nein“.

Man kann leicht nachprüfen, dass dies ein nichtdeterministischer Algorithmus ist, der das Auswertungsproblem (kombinierte Komplexität) für Boolesche regelbasierte konjunktive Anfragen in Zeit $\mathcal{O}(k \cdot n)$ löst. Insbesondere liegt das Problem also in NP.

NP*-Härte:* Zum Nachweis der NP-Härte benutzen wir das folgende Resultat, das Sie bereits aus der Veranstaltung *Einführung in die Theoretische Informatik* kennen:

Theorem: CLIQUE *ist* NP*-vollständig.*

Das Problem CLIQUE ist dabei folgendermaßen definiert:

CLIQUE
Eingabe: Ein endlicher ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$
Frage: Enthält G eine k-Clique?
(D.h. gibt es k verschiedene Knoten in G, von denen jeder mit jedem anderen durch eine Kante verbunden ist?)

Bei einem *endlichen ungerichteten Graphen* $G = (V, E)$ ist jede Kante in E ist eine 2-elementige Teilmenge von V.

Wir beweisen die NP-Härte des Auswertungsproblems (kombinierte Komplexität) für regelbasierte konjunktive Anfragen, indem wir eine Polynomialzeit-Reduktion f vom CLIQUE-Problem auf dieses Problem angeben.

Sei (G, k) eine Eingabe für's CLIQUE-Problem, und sei $G = (V, E)$. Unsere Reduktion f bildet die CLIQUE-Instanz (G, k) auf die wie folgt definierte Instanz $f(G, k) := \big(Q_{(G,k)}, \mathbf{I}_G\big)$ des Auswertungsproblems ab:

- Sei $\mathbf{S} := \{R\}$ das Datenbankschema, das aus einem Relationsnamen R der Stelligkeit 2 besteht.
- Sei $\mathbf{I}_G$ die Datenbank über dem Schema $\mathbf{S}$ mit

$$\mathbf{I}_G(R) \ := \quad \{\, (u, v) \ : \ \{u, v\} \in E \,\}$$

- Sei $Q_{(G,k)}$ die regelbasierte konjunktive Anfrage

$$Ans() \ \leftarrow \ \big(\, R(x_i, x_j), \,\big)_{1 \leqslant i < j \leqslant k'} \,,$$

  wobei $k' := \min\{k, \, |V|{+}1\}$. Mit $\big(\, R(x_i, x_j), \,\big)_{1 \leqslant i < j \leqslant k'}$ ist hier gemeint, dass der Rumpf der Anfrage aus einer durch Kommas getrennten Liste aller Atome $R(x_i, x_j)$, für alle i und j mit $1 \leqslant i < j \leqslant k'$, bestehen soll.

Beispiel:
Ist $G = (V, E)$ der Graph mit Knotenmenge $V = \{a, b, c, d\}$, dessen Kantenmenge aus den Kanten $\{a, b\}$ und $\{a, c\}$ besteht, dann ist $\mathbf{I}_G$ die Datenbank, deren Relation $\mathbf{I}_G(R)$ aus den Tupeln (a, b), (b, a), (a, c) und (c, a) besteht. Ist $k = 3$, so ist außerdem $k' = 3$, und $Q_{(G,k)}$ ist die Anfrage

$$Ans() \ \leftarrow \ R(x_1, x_2), \ R(x_1, x_3), \ R(x_2, x_3) \,.$$

Man sieht leicht, dass bei Eingabe von (G, k) die Datenbank $\mathbf{I}_G$ und die Anfrage $Q_{(G,k)}$ in Zeit polynomiell in der Größe von (G, k) erzeugt werden können. Dabei ist zu beachten, dass die Größe des Graphen $G = (V, E)$ definiert ist als $|V| + |E|$, während die „Größe der Zahl k“ definiert ist als die Anzahl der Bits, die nötig sind, um k zu repräsentieren — und das sind

$\mathcal{O}(\log k)$ viele Bits. Den Wert k' haben wir so gewählt, dass die Anfrage $Q_{(G,k)}$ in Zeit polynomiell in $|V| + |E| + \log k$ erzeugt werden kann.

Die Funktion f kann also in Polynomialzeit berechnet werden. Um den Beweis abzuschließen, müssen wir nur noch beweisen, dass Folgendes gilt:

$$G \text{ besitzt eine } k\text{-Clique} \quad \iff \quad [\![Q_{(G,k)}]\!](\mathbf{I}_G) \;=\; \{\,()\,\}\,.$$

Für die Richtung „$\Longrightarrow$" sei $v_1, \ldots, v_k$ eine Liste von k verschiedenen Knoten von G, die eine k-Clique bilden. Insbesondere gilt dann:

- $|V| \geqslant k$, und daher ist $k' = k$; und
- für alle $i, j \in \{1, \ldots, k\}$ mit $i < j$ gibt es in E die Kante $\{v_i, v_j\}$.

Sei β die Belegung für $Q_{(G,k)}$ mit $\beta(x_i) := v_i$ für jedes $i \in \{1, \ldots, k\}$. Gemäß unserer Wahl der Datenbank $\mathbf{I}_G$ gilt dann für alle $i, j \in [k]$ mit $i < j$, dass $\big(\beta(x_i), \beta(x_j)\big) \in \mathbf{I}_G(R)$ ist. Somit bezeugt β, dass $[\![Q_{(G,k)}]\!](\mathbf{I}_G) = \{\,()\,\}$ ist.

Für die Richtung „$\Longleftarrow$" sei $\beta : \{x_1, \ldots, x_{k'}\} \to \mathbf{dom}$ eine Belegung für $Q_{(G,k)}$, die bezeugt, dass $[\![Q_{(G,k)}]\!](\mathbf{I}_G) = \{\,()\,\}$ ist. Also gilt für alle $i, j \in [k']$ mit $i < j$, dass $\big(\beta(x_i), \beta(x_j)\big) \in \mathbf{I}_G(R)$.
Für alle $i \in \{1, \ldots, k'\}$ setze $v_i := \beta(x_i)$. Gemäß unserer Konstruktion der Datenbank $\mathbf{I}_G$ folgt aus $\big(\beta(x_i), \beta(x_j)\big) \in \mathbf{I}_G(R)$, dass $\{v_i, v_j\} \in E$ ist — und da G ungerichtet ist, gilt insbesondere auch $v_i \neq v_j$.
Somit ist $v_1, \ldots, v_{k'}$ eine Liste von k' verschiedenen Knoten, die in G eine k'-Clique bilden. Insbesondere ist $k' \leqslant |V|$, und daher ist $k' \stackrel{\text{Def}}{=} \min\{k,\ |V|{+}1\} = k$. Der Graph G besitzt also eine k-Clique.
Dies beendet den Beweis von Theorem 3.21. $\square$

Folie 78

Folgerung aus der NP-Vollständigkeit

Folgerung:
Falls P $\neq$ NP, so gibt es keinen Algorithmus, der das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls deterministisch in Zeit $(k{+}n)^{\mathcal{O}(1)}$ löst.

Bemerkung 3.22 (hier ohne Beweis)**.** Unter Verwendung einer stärkeren Annahme aus der *Parametrisierten Komplexitätstheorie* lässt sich Folgendes zeigen:

Theorem (Papadimitriou, Yannakakis, 1997)

Falls FPT $\neq$ W[1], *so gibt es keinen Algorithmus, der das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls deterministisch in Zeit $f(k){\cdot}n^c$ löst (wobei f irgendeine berechenbare Funktion und c irgendeine Konstante ist).*

FPT und W[1] sind Komplexitätsklassen, die in der parametrisierten Komplexitätstheorie Rollen spielen, die in etwa mit den Rollen von P und NP in der klassischen Komplexitätstheorie vergleichbar sind.

3.3 Algebraischer Ansatz: SPC-Algebra und SPJR-Algebra

Folie 79

Algebraische Anfragen — Informell

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?

Als *Anfrage in der SPJR-Algebra*:

$$\pi_{Kino,Adresse}\Big(\sigma_{\substack{Schauspieler = \\ \text{"Matt Damon"}}}\big(Filme \bowtie Programm \bowtie \delta_{\text{Name} \mapsto \text{Kino}} Kinos\big)\Big)$$

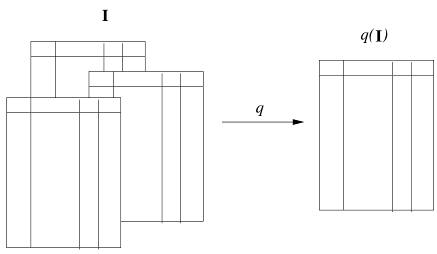
Als *Anfrage in der SPC-Algebra*:

$$\pi_{7,8}\Big(\sigma_{4=7}\big(\sigma_{1=5}\big(\sigma_{3=\text{"Matt Damon"}}\big(Filme \times Programm \times Kinos\big)\big)\big)\Big)$$

Folie 80

SPC-Algebra bzw. SPJR-Algebra

Zur Erinnerung:



- Mathematik:
  Algebraische Struktur $\widehat{=}$ Grundmenge + Operationen
- Hier:
  - Operationen auf (endlichen) Relationen
  - Speziell: Projektion, Selektion,
    Kartesisches Produkt bzw. Join und Umbenennung

Folie 81

Unbenannte Perspektive: Die SPC-Algebra

Selektion: Zwei Varianten:

- *Operator* $\sigma_{j=a}$, für eine Konstante $a \in \textbf{dom}$ und eine natürliche Zahl $j \geqslant 1$.
  Dieser Operator kann angewendet werden auf Relationen I der Stelligkeit $\geqslant j$ und liefert als Ausgabe die folgende Teilmenge der Relation I:
  $$\sigma_{j=a}(I) \ := \ \{t \in I \ : \ \text{in der } j\text{-ten Komponente von } t \text{ steht ein } a\}$$
- *Operator* $\sigma_{j=k}$, für zwei natürliche Zahlen $j, k \geqslant 1$.
  Dieser Operator kann angewendet werden auf Relationen I der Stelligkeit $\geqslant \max(j, k)$ und liefert als Ausgabe die folgende Teilmenge der Relation I:
  $$\sigma_{j=k}(I) \ := \ \left\{ t \in I \ : \ \begin{array}{l} \text{in der } j\text{-ten Komponente von } t \text{ steht derselbe} \\ \text{Eintrag wie in der } k\text{-ten Komponente von } t \end{array} \right\}$$

Selektion ist eine „horizontale“ Operation: sie wählt einzelne Tabellen-Zeilen aus.

„j=a“ und „j=k“ werden *Selektionsbedingungen* genannt.

Folie 82

Projektion:

Operator $\pi_{j_1,\ldots,j_k}$, für (nicht notwendigerweise paarweise verschiedene) natürliche Zahlen $j_1, \ldots, j_k \geqslant 1$ (und $k \geqslant 0$).

Dieser Operator kann angewendet werden auf Relationen I der Stelligkeit $\geqslant \max\{j_1,\dots,j_k\}$ und liefert als Ausgabe die folgende Relation der Stelligkeit k:

$$\pi_{j_1,\dots,j_k}(I) \ := \ \left\{ \ \big(t(j_1),\dots,t(j_k)\big) \ : \ t \in I \ \right\}$$

Projektion ist eine „vertikale“ Operation: sie wählt einzelne Tabellen-Spalten aus und arrangiert sie in möglicherweise anderer Reihenfolge

Folie 83

Kartesisches Produkt:

Operator $\times$

Dieser Operator kann angewendet werden auf Relationen I und J beliebiger Stelligkeiten m und n und liefert als Ausgabe die folgende Relation der Stelligkeit $m{+}n$:

$$I \times J \ := \ \left\{ \Big(t(1),\dots,t(m),\ s(1),\dots,s(n)\Big) \ : \ t \in I \text{ und } s \in J \right\}$$

Wir benutzen den Operator manchmal auch für einzelne Tupel:
Sind t und s Tupel der Stelligkeiten m und n, so schreiben wir $t \times s$, um das Tupel $\big(t(1),\dots,t(m),\ s(1),\dots,s(n)\big)$ zu bezeichnen.

Folie 84

Definition 3.23.
Sei $\mathbf{S}$ ein Datenbankschema. Die Klasse der Anfragen der *SPC-Algebra über* $\mathbf{S}$ (kurz: $\mathsf{SPC}[\mathbf{S}]$) ist induktiv wie folgt definiert:

- Für alle Relationsnamen $R \in \mathbf{S}$ ist R eine $\mathsf{SPC}[\mathbf{S}]$-Anfrage der Stelligkeit $\operatorname{ar}(R)$.
- Für alle Konstanten $c \in \mathbf{dom}$ ist $\{(c)\}$ eine $\mathsf{SPC}[\mathbf{S}]$-Anfrage der Stelligkeit 1.
- Ist Q eine $\mathsf{SPC}[\mathbf{S}]$-Anfrage der Stelligkeit m, sind j,k natürliche Zahlen aus $\{1,\dots,m\}$ und ist $a \in \mathbf{dom}$ eine Konstante, so sind auch $\sigma_{j=a}(Q)$ und $\sigma_{j=k}(Q)$ $\mathsf{SPC}[\mathbf{S}]$-Anfragen der Stelligkeit m.
- Ist Q eine $\mathsf{SPC}[\mathbf{S}]$-Anfrage der Stelligkeit m, ist $k \geqslant 0$ und sind $j_1,\dots,j_k$ natürliche Zahlen aus $\{1,\dots,m\}$, so ist $\pi_{j_1,\dots,j_k}(Q)$ eine $\mathsf{SPC}[\mathbf{S}]$-Anfrage der Stelligkeit k.
- Sind Q und P zwei $\mathsf{SPC}[\mathbf{S}]$-Anfragen der Stelligkeiten m und n, so ist $(Q \times P)$ eine $\mathsf{SPC}[\mathbf{S}]$-Anfrage der Stelligkeit $m{+}n$.

Die *Semantik* $[\![Q]\!]$ von $\mathsf{SPC}[\mathbf{S}]$-Anfragen Q ist induktiv auf die offensichtliche Art definiert. Insbesondere für die Anfrage $Q := \{(c)\}$ ist die Semantik so definiert, dass für *jede* Datenbank $\mathbf{I}$ gilt: $[\![Q]\!](\mathbf{I}) = \{\,(c)\,\}$.

Folie 85

Beispiele für Anfragen der SPC-Algebra

- *(4)* Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?

$$\pi_{7,8}\Big(\sigma_{4=7}\big(\sigma_{1=5}\big(\sigma_{3=\text{“Matt Damon”}}\big(\mathit{Filme} \times \mathit{Programm} \times \mathit{Kinos}\big)\big)\big)\Big)$$

- *(9)* Egal, wie die DB aussieht, gib immer (*"Terminator", "Linda Hamilton"*) aus!

$$\big(\ \{\,(\text{“Terminator”})\,\} \times \{\,(\text{“Linda Hamilton”})\,\}\ \big)$$

- *(3)* Gib Adresse und Telefonnummer des "Kino International" aus!

$$\pi_{2,4}\big(\,\sigma_{1=\text{“Kino International”}}(\mathit{Kinos})\,\big)$$

- *(5)* Läuft zur Zeit ein Film von "James Cameron"?

$$\pi\Big(\sigma_{1=5}\big(\,(\,\sigma_{2=\text{“James Cameron”}}(\mathit{Filme})\ \times\ \mathit{Programm}\,)\,\big)\Big)$$

  Der Operator $\pi(\cdot)$ bedeutet hier, dass auf „keine einzige" Spalte projiziert wird. Das Ergebnis ist daher entweder $\{()\}$ oder $\emptyset$ (also „ja" oder „nein").

- Sei $\mathbf{I}$ die Datenbank mit

$\mathbf{I}(R)$:

1	2
a	b
a	c
b	d

und $\mathbf{I}(S)$:

1	2	3
b	a	a
c	b	a
c	b	b

  und sei Q die Anfrage $\sigma_{2=5}\big(\,(\,R \times \sigma_{2=3}(S)\,)\,\big)$.

  Auswertung von Q in $\mathbf{I}$:

- $\sigma_{2=3}(S)$ liefert auf **I** die Relation

1	2	3
b	a	a
c	b	b

- $(\,R \times \sigma_{2=3}(S)\,)$ liefert auf **I** die Relations

1	2	3	4	5
a	b	b	a	a
a	b	c	b	b
a	c	b	a	a
a	c	c	b	b
b	d	b	a	a
b	d	c	b	b

- $\sigma_{2=5}\big(\,(\,R \times \sigma_{2=3}(S)\,)\,\big)$ liefert auf **I** die Relation

1	2	3	4	5
a	b	c	b	b

Somit ist $[\![Q]\!](\mathbf{I}) = \{\,(a,b,c,b,b)\,\}$.

Folie 86

Verallgemeinerung des Selektions-Operators

Eine *positive konjunktive Selektionsbedingung* ist eine Formel F der Form

$$\gamma_1 \wedge \cdots \wedge \gamma_n$$

wobei $n \geqslant 1$ und jedes γ_i eine Selektionsbedingung der Form $j_i{=}a_i$ oder $j_i{=}k_i$ für natürliche Zahlen $j_i, k_i \geqslant 1$ und Konstanten $a_i \in \mathbf{dom}$.

Der *Selektionsoperator* σ_F hat dieselbe Wirkung wie die Hintereinanderausführung der Selektionsoperatoren σ_{γ_i} für alle $i \in \{1,\ldots,n\}$.

Folie 87

Eine Normalform für SPC-Anfragen

Sei $\mathbf{S}$ ein Relationsschema.

Definition 3.24. Eine $\mathsf{SPC}[\mathbf{S}]$-Anfrage ist in *Normalform*, falls sie von der Form

$$\pi_{j_1,\ldots,j_k}\Big(\{(c_1)\} \times \cdots \times \{(c_m)\} \times \sigma_F\big(R_1 \times \cdots \times R_\ell\big)\Big)$$

ist, für $k, m, \ell \geqslant 0$, paarweise verschiedene Elemente $j_1, \ldots, j_k$, so dass $\{1, \ldots, m\} \subseteq \{j_1, \ldots, j_k\}$, Konstanten $c_1, \ldots, c_m \in \mathbf{dom}$, $R_1, \ldots, R_\ell \in \mathbf{S}$ und F eine positive konjunktive Selektionsbedingung.

Proposition 3.25. *Für jede* $\mathsf{SPC}[\mathbf{S}]$*-Anfrage* Q *gibt es eine* $\mathsf{SPC}[\mathbf{S}]$*-Anfrage* Q' *in Normalform, die dieselbe Anfragefunktion definiert; und es gibt einen Polynomialzeit-Algorithmus, der* Q' *bei Eingabe von* Q *erzeugt.*

Beweis: Übung.

Folie 88

Benannte Perspektive: Die SPJR-Algebra
Operationen:
Selektion σ, Projektion π, Join $\bowtie$, Umbenennung (Renaming) δ

Attributnamen an Stelle von Spaltennummern!

Selektion: Zwei Varianten:

- *Operator* $\sigma_{A=a}$,
  für eine Konstante $a \in \mathbf{dom}$ und einen *Attributnamen* A.
  Dieser Operator kann angewendet werden auf R-Relationen I mit $A \in \mathit{sorte}(R)$ und liefert als Ausgabe die folgende Teilmenge der Relation I:
  $$\sigma_{A=a}(I) \ := \ \{ \ t \in I \ \ : \ \ t(A) = a \ \}$$
- *Operator* $\sigma_{A=B}$, für zwei *Attributnamen A und B*.
  Dieser Operator kann angewendet werden auf R-Relationen I mit $A, B \in \mathit{sorte}(R)$ und liefert als Ausgabe die folgende Teilmenge der Relation I:
  $$\sigma_{A=B}(I) \ := \ \{ \ t \in I \ : \ t(A) = t(B) \ \}$$

„$A{=}a$“ und „$A{=}B$“ werden *Selektionsbedingungen* genannt.

Folie 89

Projektion:

Operator $\pi_{A_1,\dots,A_k}$, für paarweise verschiedene Attributnamen $A_1,\dots,A_k$ (und $k \geqslant 0$).

Dieser Operator kann angewendet werden auf R-Relationen I mit $sorte(R) \supseteq \{A_1,\dots,A_k\}$ und liefert als Ausgabe die folgende Relation der Sorte $\{A_1,\dots,A_k\}$:

$$\pi_{A_1,\dots,A_k}(I) \ := \ \Big\{ \ \big(A_1 : t(A_1), \ \dots, \ A_k : t(A_k) \big) \ : \ \ t \in I \ \Big\}$$

Folie 90

An Stelle des Kartesischen Produkts: Natürlicher Join

Beispiel: *Wenn* I und J zwei Relationen mit *disjunkten Attributmengen* (d.h. Spaltenbezeichnungen) sind, so bilde einfach das herkömmliche Kartesische Produkt.

$I \quad \bowtie \quad J \quad = \quad I \bowtie J$

A	B
1	a
2	b
3	c

C	D	E
e	f	g
h	i	j

A	B	C	D	E
1	a	e	f	g
1	a	h	i	j
2	b	e	f	g
2	b	h	i	j
3	c	e	f	g
3	c	h	i	j

Frage: Was soll passieren, wenn I und J *gemeinsame Attribute* haben?
Antwort: Zueinander passende Tupel werden verschmolzen.

Beispiel: $I \quad \bowtie \quad J \quad = \quad I \bowtie J$

A	B
1	a
2	b
3	c

B	C	D
a	f	g
a	i	j
c	l	m

A	B	C	D
1	a	f	g
1	a	i	j
3	c	l	m

Folie 91

Natürlicher Join:

Operator $\bowtie$

Dieser Operator kann angewendet werden auf eine R-Relation I und eine S-Relation J beliebiger Sorten und liefert als Ausgabe die folgende Relation der Sorte $\Sigma := \mathit{sorte}(R) \cup \mathit{sorte}(S)$:

$$I \bowtie J \;:=\; \Big\{ \text{Tupel } t \text{ der Sorte } \Sigma : \begin{array}{l} \text{es gibt Tupel } t' \in I \text{ und } t'' \in J \text{ so dass} \\ t|_{\mathit{sorte}(R)} = t' \;\text{ und }\; t|_{\mathit{sorte}(S)} = t'' \end{array} \Big\}$$

Wir benutzen den Operator manchmal auch für einzelne Tupel:
Sind t' und t'' Tupel der Sorten $\mathit{sorte}(R)$ und $\mathit{sorte}(S)$, so schreiben wir $t' \bowtie t''$, um das Tupel t der Sorte $\mathit{sorte}(R) \cup \mathit{sorte}(S)$ mit $t|_{\mathit{sorte}(R)} = t'$ und $t|_{\mathit{sorte}(S)} = t''$ zu bezeichnen (bzw. den Wert *„undefiniert“*, falls es kein solches Tupel gibt, d.h. falls $t'|_{\mathit{sorte}(R)\cap \mathit{sorte}(S)} \neq t''|_{\mathit{sorte}(R)\cap\mathit{sorte}(S)}$).

Folie 92

Umbenennung (Renaming):

Operator δ_f, für eine *Umbenennungsfunktion* f, d.h. eine injektive Funktion $f : U \to \textbf{att}$, für eine beliebige endliche Menge U von Attributnamen.
Dieser Operator kann angewendet werden auf R-Relationen I, für die gilt: $U \subseteq \mathit{sorte}(R)$ und $\big(\mathit{sorte}(R) \setminus U\big) \cap f(U) = \emptyset$ und liefert die folgende Relation der Sorte $f(U) \cup \big(\mathit{sorte}(R) \setminus U\big)$:

$$\delta_f(I) \;:=\; \left\{ \text{Tupel } t \;:\; \begin{array}{l} \text{ex } t' \in I \text{ so dass gilt:} \\ \text{f.a. } A \in U \text{ ist } t'(A) = t(f(A)) \\ \text{und f.a. } A \in \mathit{sorte}(R) \setminus U \text{ ist } t'(A) = t(A) \end{array} \right\}$$

Oft schreiben wir $A_1 \cdots A_k \mapsto B_1 \cdots B_k$ um die Umbenennungsfunktion f mit Definitionsbereich $U = \{A_1, \ldots, A_k\}$ und Werten $f(A_i) = B_i$, für alle $i \in \{1, \ldots, k\}$, zu bezeichnen.

Folie 93

Definition 3.26.
Sei $\mathbf{S}$ ein Datenbankschema. Die Klasse der Anfragen der *SPJR-Algebra über* $\mathbf{S}$ (kurz: $\mathsf{SPJR}[\mathbf{S}]$) ist induktiv wie folgt definiert:

- Für alle Relationsnamen $R \in \mathbf{S}$ ist R eine $\mathsf{SPJR}[\mathbf{S}]$-Anfrage der Sorte $\mathit{sorte}(R)$.
- Für alle Konstanten $c \in \textbf{dom}$ und alle Attributnamen $A \in \textbf{att}$ ist $\{(A : c)\}$ eine $\mathsf{SPJR}[\mathbf{S}]$-Anfrage der Sorte $\{A\}$.
- Ist Q eine $\mathsf{SPJR}[\mathbf{S}]$-Anfrage der Sorte Σ, sind $A, B \in \Sigma$ und ist $a \in \textbf{dom}$ eine Konstante, so sind auch $\sigma_{A=a}(Q)$ und $\sigma_{A=B}(Q)$ $\mathsf{SPJR}[\mathbf{S}]$-Anfragen der Sorte Σ.

- Ist Q eine SPJR[**S**]-Anfrage der Sorte Σ, ist $k \geqslant 0$ und sind $A_1, \ldots, A_k$ paarweise verschiedene Elemente aus Σ, so ist $\pi_{A_1,\ldots,A_k}(Q)$ eine SPJR[**S**]-Anfrage der Sorte $\{A_1, \ldots, A_k\}$.

- Sind Q und P zwei SPJR[**S**]-Anfragen der Sorten Σ und Π, so ist $(Q \bowtie P)$ eine SPJR[**S**]-Anfrage der Sorte $\Sigma \cup \Pi$.

- Ist Q eine SPJR[**S**]-Anfrage der Sorte Σ und ist $f : \Sigma \to \mathbf{att}$ eine Umbenennungsfunktion, so ist $\delta_f(Q)$ eine SPJR[**S**]-Anfrage der Sorte $f(\Sigma)$.

Die *Semantik* $[\![Q]\!]$ von SPJR[**S**]-Anfragen Q ist induktiv auf die offensichtliche Art definiert.
Insbesondere für die Anfrage $Q := \{(A : c)\}$ ist die Semantik so definiert, dass für *jede* Datenbank **I** gilt: $[\![Q]\!](\mathbf{I}) = \{\, (A : c) \,\}$.

Folie 94

Beispiele für Anfragen der SPJR-Algebra

- *(4)* Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?

$$\pi_{Name,Adresse}\Big(\sigma_{\substack{Schauspieler\,=\\ \text{“Matt Damon”}}}\big((Filme \bowtie (Kinos \bowtie \delta_{\text{Kino} \mapsto \text{Name}}(Programm)))\big)\Big)$$

- *(9)* Egal, wie die DB aussieht, gib immer (*"Terminator", "Linda Hamilton"*) aus!

$$\big(\ \{\,(\,1 : \text{“Terminator”}\,)\,\}\ \bowtie\ \{\,(\,2 : \text{“Linda Hamilton”}\,)\,\}\ \big)$$

- *(3)* Gib Adresse und Telefonnummer des "Kino International" aus!

$$\pi_{\text{Adresse, Telefon}}\big(\,\sigma_{\text{Name=“Kino International”}}(Kinos)\,\big)$$

- *(5)* Läuft zur Zeit ein Film von "James Cameron"?

$$\pi\big(\,(\,\sigma_{\text{Regie=“James Cameron”}}(Filme)\ \bowtie\ Programm\,)\,\big)$$

Der Operator $\pi(\cdot)$ bedeutet hier, dass auf „keine einzige" Spalte projiziert wird. Das Ergebnis ist daher entweder $\{()\}$ oder $\emptyset$ (also „ja" oder „nein").

- Sei $\mathbf{I}$ die Datenbank mit

$\mathbf{I}(R)$:

A	B
a	b
a	c
b	d

und $\mathbf{I}(S)$:

C	D	E
b	a	a
c	b	a
c	b	b

und sei Q die Anfrage $\sigma_{C=c}\big(\pi_{A,C}(\,(R \bowtie \delta_{E\mapsto A}(S)\,)\,)\big)$.

Auswertung von Q in $\mathbf{I}$:

- $\delta_{E\mapsto A}(S)$ liefert auf $\mathbf{I}$ die Relation

C	D	A
b	a	a
c	b	a
c	b	b

- $(\,R \bowtie \delta_{E\mapsto A}(S)\,)$ liefert auf $\mathbf{I}$ die Relations

A	B	C	D
a	b	b	a
a	b	c	b
a	c	b	a
a	c	c	b
b	d	c	b

- $\pi_{A,C}(\,(R \bowtie \delta_{E\mapsto A}(S)\,)\,)$ liefert auf $\mathbf{I}$ die Relation

A	C
a	b
a	c
a	b
a	c
b	c

=

A	C
a	b
a	c
b	c

- $\sigma_{C=c}\big(\pi_{A,C}(\,(R \bowtie \delta_{E\mapsto A}(S)\,)\,)\big)$ liefert auf $\mathbf{I}$ die Relation

A	C
a	c
b	c

Somit ist $[\![Q]\!](\mathbf{I}) = \{\,(\,A:a,\ C:c\,)\ ,\ (\,A:b,\ C:c\,)\,\}$.

Folie 95

Verallgemeinerung des Selektions-Operators

Wie bei der SPC-Algebra lassen wir wieder eine *Verallgemeinerung des Selektions-Operators* zu:

- Eine *positive konjunktive Selektionsbedingung* ist eine Formel F der Form
$$\gamma_1 \wedge \cdots \wedge \gamma_n$$
wobei $n \geqslant 1$ und jedes γ_i eine Selektionsbedingung der Form $A_i{=}a_i$ oder $A_i{=}B_i$ für Attributnamen A_i, B_i und Konstanten $a_i \in \mathbf{dom}$.

- Der *Selektionsoperator* σ_F hat dieselbe Wirkung wie die Hintereinanderausführung der Selektionsoperatoren σ_{γ_i} für alle $i \in \{1, \ldots, n\}$.

Folie 96

Eine Normalform für SPJR-Anfragen

Sei $\mathbf{S}$ ein Relationsschema.

Definition 3.27.
Eine $\mathsf{SPJR}[\mathbf{S}]$-Anfrage ist in *Normalform*, falls sie von der Form

$$\pi_{B_1,\ldots,B_k}\Big(\{(A_1 : c_1)\} \bowtie \cdots \bowtie \{(A_m : c_m)\} \bowtie \sigma_F\big(\delta_{f_1}(R_1) \bowtie \cdots \bowtie \delta_{f_\ell}(R_\ell)\big)\Big)$$

ist, für $k, m, \ell \geqslant 0$, $B_1, \ldots, B_k, A_1, \ldots, A_m \in \mathbf{att}$ so dass $\{A_1, \ldots, A_m\} \subseteq \{B_1, \ldots, B_k\}$ und die $A_1, \ldots, A_m$ paarweise verschieden, $c_1, \ldots, c_m \in \mathbf{dom}$, $R_1, \ldots, R_\ell \in \mathbf{S}$, eine positive konjunktive Selektionsbedingung F und Umbenennungsfunktionen $f_1, \ldots, f_\ell$, so dass die Sorten von $\delta_{f_1}(R_1)$,..., $\delta_{f_\ell}(R_\ell)$ paarweise disjunkt sind und keins der $A_1, \ldots, A_m$ als Attribut von einem der $\delta_{f_j}(R_j)$ vorkommt.

Proposition 3.28. *Für jede* $\mathsf{SPJR}[\mathbf{S}]$*-Anfrage Q gibt es eine* $\mathsf{SPJR}[\mathbf{S}]$*-Anfrage Q' in Normalform, die dieselbe Anfragefunktion definiert; und es gibt einen Polynomialzeit-Algorithmus, der Q' bei Eingabe von Q erzeugt.*

Beweis: Übung.

Folie 97

Nicht-Erfüllbare Anfragen

Bemerkung:
Sowohl in der SPC-Algebra als auch in der SPJR-Algebra lassen sich unerfüllbare Anfragen ausdrücken.

Beispiel: Die Anfrage $Q :=$

$$\sigma_{3=\text{“George Clooney”}}\Big(\sigma_{3=\text{“Matt Damon”}}\big(\mathit{Filme}\big)\Big)$$

wählt in einer Datenbank vom Schema **KINO** genau diejenigen Tupel $t = (a, b, c)$ aus der *Filme*-Relation aus, für deren dritte Komponente c gilt: $c =$ “Matt Damon” <u>und</u> $c =$ “George Clooney”.

Solche Tupel kann es aber nicht geben!

Für <u>jede</u> Datenbank $\mathbf{I}$ vom Schema **KINO** gilt also: $[\![Q]\!](\mathbf{I}) = \emptyset$.

Somit ist die Anfrage Q nicht erfüllbar.

Folie 98

Äquivalenz der Ausdrucksstärke der SPC-Algebra und der SPJR-Algebra

Lemma 3.29. *Die SPC-Algebra und die SPJR-Algebra können genau dieselben Anfragen ausdrücken.*

Es gilt sogar: Jede Anfrage aus einer dieser Anfragesprachen kann in polynomieller Zeit in eine äquivalente Anfrage der anderen Anfragesprache übersetzt werden.

Beweis. Wir nutzen die Normalformen für SPC- und für SPJR-Anfragen, die in den Propositionen 3.25 und 3.28 bereitgestellt wurden. Es reicht daher o.B.d.A. aus, jeweils nur solche Anfragen zu übersetzen, die in Normalform sind.

SPC $\rightsquigarrow$ SPJR:
Bei Eingabe einer SPC-Anfrage Q der Form

$$\pi_{j_1,\ldots,j_k}\Big(\{(c_1)\} \times \cdots \times \{(c_m)\} \times \sigma_F\big(R_1 \times \cdots \times R_\ell\big)\Big)$$

konstruieren wir die SPJR-Anfrage Q' der Form

$$\pi_{j_1,\ldots,j_k}\Big(\{(1:c_1)\} \bowtie \cdots \bowtie \{(m:c_m)\} \bowtie \sigma_{F'}\big(\delta_{f_1}(R_1) \bowtie \cdots \bowtie \delta_{f_\ell}(R_\ell)\big)\Big)\,,$$

wobei $f_1,\ldots,f_\ell$ Umbenennungsfunktionen sind, die Folgendes gewährleisten:

- $\delta_{f_1}(R_1)$ hat die Spaltenbezeichner $m{+}1,\ldots,m{+}\mathrm{ar}(R_1)$,
- $\delta_{f_2}(R_2)$ hat die Spaltenbezeichner $m{+}\mathrm{ar}(R_1){+}1,\ldots,m{+}\mathrm{ar}(R_1){+}\mathrm{ar}(R_2)$,
- $\vdots$
- $\delta_{f_\ell}(R_\ell)$ hat die Spaltenbezeichner $m_{\ell-1}{+}1,\ldots,m_{\ell-1}+\mathrm{ar}(R_\ell)$, wobei $m_{\ell-1} := m+\mathrm{ar}(R_1)+\cdots+\mathrm{ar}(R_{\ell-1})$ ist.

Die verallgemeinerte Selektionsbedingung F' entsteht dabei aus der verallgemeinerten Selektionsbedingung F, indem jede Selektionsbedingung in F der Form

- $\nu_i = a_i$ (für eine natürliche Zahl ν_i und eine Konstante $a_i \in \mathbf{dom}$) ersetzt wird durch die Selektionsbedingung $\nu_i' = a_i$, für $\nu_i' := m+\nu_i$, und
- $\nu_i = \mu_i$ (für natürliche Zahlen ν_i und μ_i) ersetzt wird durch die Selektionsbedingung $\nu_i' = \mu_i'$, für $\nu_i' := m{+}\nu_i$ und $\mu_i' := m{+}\mu_i$.

Einfaches Nachprüfen zeigt, dass die beiden Anfragen Q und Q' tatsächlich äquivalent sind, d.h. dass für jede Datenbank $\mathbf{I}$ vom zu Q passenden Schema gilt: $[\![Q]\!](\mathbf{I}) = [\![Q']\!](\mathbf{I})$. Details: Übung!

SPJR $\rightsquigarrow$ SPC:
Analog; Details: Übung! $\square$

Folie 99

Äquivalenz des deskriptiven und des algebraischen Ansatzes

Theorem 3.30 (Äquivalenz konjunktiver Anfragesprachen). *Die folgenden Anfragesprachen können genau dieselben erfüllbaren Anfragefunktionen ausdrücken:*

(a) die Klasse der regelbasierten konjunktiven Anfragen

(b) die Klasse der Tableau-Anfragen

(c) die Klasse der Anfragen des konjunktiven Kalküls

(d) die Anfragen der SPC*-Algebra*

(e) die Anfragen der SPJR*-Algebra.*

Es gilt sogar: Jede Anfrage aus einer dieser Anfragesprachen kann in polynomieller Zeit in äquivalente Anfragen der anderen Anfragesprachen übersetzt werden.

Beweis. Von Lemma 3.14 wissen wir bereits, dass die Anfragesprachen aus (a), (b) und (c) dieselbe Ausdrucksstärke besitzen, und dass Anfragen jeweils in Polynomialzeit in jede der drei Sprachen übersetzt werden können. Von Lemma 3.29 wissen wir, dass SPC-Algebra und SPJR-Algebra dieselbe Ausdrucksstärke besitzen, und dass Anfragen in Polynomialzeit in äquivalente Anfragen der beiden Algebren übersetzt werden können.
Die Einschränkung auf *erfüllbare* Anfragen müssen wir machen, weil gemäß Satz 3.6 alle regelbasierten konjunktiven Anfragen erfüllbar sind, während in der SPC- und der SPJR-Algebra auch nicht-erfüllbare Anfragen formuliert werden können.
Um den Beweis von Theorem 3.30 abzuschließen, genügt es, die Richtung von (a) nach (d) und die Richtung von (d) nach (a) zu beweisen.

(a) $\rightsquigarrow$ (d):
Sei Q eine regelbasierte konjunktive Anfrage der Form

$$Ans(u) \ \leftarrow \ R_1(u_1), \ldots, R_\ell(u_\ell) \ .$$

Sei $r := \operatorname{ar}(Ans)$, und für jedes $i \in \{1, \ldots, \ell\}$ sei $r_i := \operatorname{ar}(R_i)$.
Sei u von der Form $x_1, \ldots, x_r$, sei m die Anzahl der Konstanten in $x_1, \ldots, x_r$ und seien $\alpha_1 < \cdots < \alpha_m$ diejenigen Indizes in $\{1, \ldots, r\}$ mit $x_{\alpha_1}, \ldots, x_{\alpha_m} \in \mathbf{dom}$.
Außerdem sei u_1 von der Form $y_1, \ldots, y_{r_1}$, sei u_2 von der Form $y_{r_1+1}, \ldots y_{r_1+r_2}$, usw., und sei u_ℓ von der Form $y_{m_{\ell-1}+1}, \ldots, y_{m_{\ell-1}+r_\ell}$, wobei $m_{\ell-1} := r_1 + \cdots + r_{\ell-1}$ ist und jeweils $y_i \in \mathbf{var} \cup \mathbf{dom}$ für jedes $i \in \{1, \ldots, m_{\ell-1}{+}r_\ell\}$ gilt.

Wir wählen folgende SPC-Algebra-Anfrage Q':

$$\pi_{j_1,\ldots,j_r}\Big(\ \{(x_{\alpha_1})\} \times \cdots \times \{(x_{\alpha_m})\} \times \sigma_F\big(R_1 \times \cdots \times R_\ell\big) \ \Big) ,$$

wobei die verallgemeinerte Selektionsbedingung F und die Indizes $j_1, \ldots, j_r$ wie folgt gewählt sind. F ist die Konjunktion aller Bedingungen der Form

- $i = c$, für alle $i \in \{1, \ldots, m_{\ell-1}+r_\ell\}$ und alle $c \in \textbf{dom}$ mit $y_i = c$, und
- $i = j$, für alle $i, j \in \{1, \ldots, m_{\ell-1}+r_\ell\}$ mit $i \neq j$ und $y_i = y_j$.

Die Indizes $j_1, \ldots, j_r$ sind so gewählt, dass für jedes $i \in \{1, \ldots, r\}$ gilt:

$$j_i \ := \begin{cases} m+k & , \ \text{falls } x_i \in \textbf{var} \text{ und } k \text{ minimal, so dass } x_i = y_k \\ k & , \ \text{falls } x_i \in \textbf{dom} \text{ und } k \in \{1, \ldots, m\}, \text{ so dass } i = \alpha_k \end{cases}$$

Einfaches Nachprüfen zeigt, dass Q und Q' äquivalent sind. Details: Übung!

(d) $\rightsquigarrow$ (a):
Wir können hier rekursiv entlang dem Aufbau von Anfragen der SPC-Algebra vorgehen (und beachten dabei, dass bei einer *erfüllbaren* Anfrage Q der SPC-Algebra auch sämtliche Teilanfragen $\hat{Q}$, aus denen Q aufgebaut ist, erfüllbar sind). Details: Übung! $\square$

Folie 100

Beispiel zur Übersetzung von Anfragen

Betrachte die Anfrage $Q :=$

$$\begin{array}{rcl} \mathit{Ans}(x, \text{“Treffer”}) & \leftarrow & \mathit{Programm}(x, y, z_1), \\ & & \mathit{Filme}(z_2, y, \text{“George Clooney”}), \\ & & \mathit{Filme}(z_3, \text{“George Clooney”}, y) \end{array}$$

Unser Beweis der Richtung (a) $\rightsquigarrow$ (b) von Theorem 3.30 übersetzt diese Anfrage in die SPC-Algebra-Anfrage

$$\pi_{2,1}\Big(\{(\text{“Treffer”})\} \times \sigma_F\big(\mathit{Programm} \times \mathit{Filme} \times \mathit{Filme}\big)\Big)$$

mit der verallgemeinerten Selektionsbedingung $F :=$

$$6 = \text{“George Clooney”} \ \wedge \ 8 = \text{“George Clooney”} \ \wedge \ 2 = 5 \ \wedge \ 2 = 9 \ \wedge \ 5 = 9\,.$$

3.4 Homomorphismus-Satz, Statische Analyse und Anfrageminimierung

Folie 101

Vorbemerkung zum Thema „Statische Analyse, Optimierung“

Optimierung:
Finde zur gegebenen Anfrage eine „minimale“ äquivalente Anfrage

Definition 3.31 (Äquivalenz und Query Containment)**.**
Seien P und Q Anfragen einer Anfragesprache über einem DB-Schema $\mathbf{S}$.

(a) Wir schreiben $Q \equiv P$ und sagen *„Q ist äquivalent zu P“*, falls für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen *„Q ist in P enthalten“*, falls für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) \subseteq [\![P]\!](\mathbf{I})$.

Statische Analyse:

- *Äquivalenzproblem:*
  Bei Eingabe zweier Anfragen Q, P teste, ob $Q \equiv P$.
- *Query Containment Problem:*
  Bei Eingabe von Anfragen Q, P teste, ob $Q \sqsubseteq P$.
- *Erfüllbarkeitsproblem:*
  Bei Eingabe einer Anfrage Q teste, ob Q erfüllbar ist (d.h. ob es eine DB $\mathbf{I}$ gibt, so dass $[\![Q]\!](\mathbf{I}) \neq \emptyset$ ist).

Bekannt:

- Für regelbasierte konjunktive Anfragen ist das Erfüllbarkeitsproblem trivial (Satz 3.6).
- Für konjunktive Anfragen mit „=“ (bzw. für SPC- bzw. SPJR-Anfragen) ist das Erfüllbarkeitsproblem in polynomieller Zeit lösbar (dies wurde in den Übungsaufgaben gezeigt).
- *Jetzt:* Algorithmen für's Query Containment Problem und für's Äquivalenzproblem für konjunktive Anfragen.

Folie 102

Zusammenhänge:

- *Äquivalenz vs. Containment:*
  - $Q \equiv P \quad \Longleftrightarrow \quad Q \sqsubseteq P$ und $P \sqsubseteq Q$
  - $Q \sqsubseteq P \quad \Longleftrightarrow \quad (Q \vee P) \equiv P$
    Dies lässt sich für Optimierung nutzen!
- *Containment vs. Erfüllbarkeit:*
  - Q unerfüllbar $\quad \Longrightarrow \quad Q \sqsubseteq P$ für *alle* Anfragen P
  - $Q \sqsubseteq P \quad \Longleftrightarrow \quad (Q \wedge \neg P)$ ist unerfüllbar
    Dies lässt sich für Optimierung nutzen!
- *Erfüllbarkeit vs. Äquivalenz:*
  - Q unerfüllbar $\quad \Longrightarrow \quad (Q \vee P) \equiv P$ für *alle* Anfragen P
    Dies lässt sich für Optimierung nutzen!

Folie 103

Zur Erinnerung: Tableau-Anfragen — Beispiel

Beispiel-Anfrage:

Filmtitel + Regisseur aller z.Zt. laufenden Filme, deren Regisseur schon mal mit "Sandra Bullock" zusammengearbeitet hat.

Als regelbasierte konjunktive Anfrage:

$$\begin{array}{rcl} \mathit{Ans}(x_T, x_R) & \leftarrow & \mathit{Programm}(x_K, x_T, x_Z), \\ & & \mathit{Filme}(x_T, x_R, x_S), \\ & & \mathit{Filme}(y_T, x_R, \text{"Sandra Bullock"}) \end{array}$$

Als Tableau-Anfrage: $\big(\mathsf{T},\ (x_T,\ x_R)\big)$ mit folgendem Tableau T:

Programm

Kino	Titel	Zeit
x_K	x_T	x_Z

Filme

Titel	Regie	Schauspieler
x_T	x_R	x_S
y_T	x_R	"Sandra Bullock"

Folie 104

Homomorphismen

Definition 3.32. Seien $Q' = (\mathsf{T}', u')$ und $Q = (\mathsf{T}, u)$ zwei Tableau-Anfragen über einem DB-Schema $\mathbf{S}$.

(a) Eine *Substitution für* Q' ist eine Abbildung $h : \mathrm{var}(Q') \to \mathbf{var} \cup \mathbf{dom}$.

Wie üblich setzen wir h auf natürliche Weise fort zu einer Abbildung von $\mathrm{var}(Q') \cup \mathbf{dom}$ nach $\mathbf{var} \cup \mathbf{dom}$, so dass $h|_{\mathbf{dom}} = \mathrm{id}$.
Für ein freies Tupel $t = (e_1, \ldots, e_k)$ setzen wir $h(t) := (h(e_1), \ldots, h(e_k))$.
Für eine Menge M von freien Tupeln ist $h(M) := \{\, h(t) \ :\ t \in M \,\}$.

(b) Eine Substitution h für Q' heißt *Homomorphismus von* Q' *auf* Q, falls

- $h(u') = u$ und
- $h(\mathsf{T}') \subseteq \mathsf{T}$, d.h. für alle $R \in \mathbf{S}$ ist $h\big(\mathsf{T}'(R)\big) \subseteq \mathsf{T}(R)$.

Beachte: Dann gilt insbes.: $h\big(\,\mathrm{var}(Q')\big) \ \subseteq\ \mathrm{var}(Q) \cup \mathrm{adom}(Q)$.

Beispiel: $\mathbf{S} := \{R\}$, $Q' := (\mathsf{T}', (x, y))$, $Q := (\mathsf{T}, (x, y))$ mit

$\mathsf{T}'(R) :=$

A	B
x	y_1
x_1	y_1
x_1	y

$\mathsf{T}(R) :=$

A	B
x	y

Homomorphismus h von Q' auf Q: $h : x, y, x_1, y_1 \mapsto x, y, x, y$.
Es gibt keinen Homomorphismus von Q auf Q'.

Folie 105

Die kanonische Datenbank $\mathbf{I}_Q^{Q'}$

Idee: $\mathbf{I}_Q^{Q'}$ ist die Datenbank, die aus dem Tableau T von Q entsteht, indem jede Variable in T durch eine Konstante repräsentiert wird, die nicht in Q oder Q' vorkommt.

Wir legen ein für alle Mal für jedes endliche $C \subseteq \mathbf{dom}$ eine *injektive Abbildung* $\alpha_C : \mathbf{var} \to \mathbf{dom} \setminus C$ fest. Wie üblich setzen wir α_C fort zu einer Abbildung von $\mathbf{var} \cup \mathbf{dom}$ nach $\mathbf{dom}$ mit $\alpha|_{\mathbf{dom}} = \mathrm{id}$.

Für die folgendermaßen definierte „Umkehrfunktion“
$\alpha_C^{-1} : \mathbf{dom} \to \mathbf{var} \cup \mathbf{dom}$

$$\alpha_C^{-1}(a) \ := \ \begin{cases} a & \text{falls } a \notin \text{Bild}(\alpha_C) \\ y & \text{falls } a = \alpha_C(y) \text{ für } y \in \mathbf{var} \end{cases}$$

gilt für alle $b \in \mathbf{var} \cup C$, dass $\alpha_C^{-1}\big(\alpha_C(b)\big) = b$.

Definition 3.33 (*Idee: Repräsentiere* $Q = (\mathsf{T}, u)$ *durch eine Datenbank* $\mathbf{I}_Q^{Q'}$ *und ein Tupel* $u_Q^{Q'}$)**.**
$Q' = (\mathsf{T}', u')$ und $Q = (\mathsf{T}, u)$ seien Tableau-Anfragen über einem DB-Schema $\mathbf{S}$. Die *kanonische Datenbank* $\mathbf{I}_Q^{Q'} \in \mathit{inst}(\mathbf{S})$ und das *kanonische Tupel* $u_Q^{Q'}$ sind folgendermaßen definiert:
Für $C := \text{adom}(Q) \cup \text{adom}(Q')$ ist $u_Q^{Q'} := \alpha_C(u)$ und $\mathbf{I}_Q^{Q'} := \alpha_C(\mathsf{T})$, d.h. $\mathbf{I}_Q^{Q'}(R) = \alpha_C\big(\mathsf{T}(R)\big)$, für alle $R \in \mathbf{S}$.

Proposition 3.34. *$Q' = (\mathsf{T}', u')$ und $Q = (\mathsf{T}, u)$ seien Tableau-Anfragen über einem DB-Schema $\mathbf{S}$. Dann gilt:*
Es gibt einen Homomorphismus von Q' auf Q $\iff$ $u_Q^{Q'} \in [\![Q']\!](\mathbf{I}_Q^{Q'})$.

Beweis. Sei $C := \text{adom}(Q) \cup \text{adom}(Q')$.

„$\Longrightarrow$“: Sei $h : \text{Var}(Q') \to \mathbf{var} \cup \mathbf{dom}$ ein Homomorphismus von Q' auf Q.
D.h.: $h(u') = u$ und $h(\mathsf{T}') \subseteq \mathsf{T}$.
Unser Ziel ist, zu zeigen, dass $u_Q^{Q'} \in [\![Q']\!](\mathbf{I}_Q^{Q'})$. Dazu müssen wir eine Belegung $\beta : \text{Var}(Q') \to \mathbf{dom}$ finden, für die gilt:

(1) $\beta(u') = u_Q^{Q'}$ und

(2) β ist eine Einbettung von Q' in $\mathbf{I}_Q^{Q'}$, d.h. es gilt: $\beta(\mathsf{T}') \subseteq \mathbf{I}_Q^{Q'}$.

Dazu betrachten wir die Belegung $\beta : \text{Var}(Q') \to \mathbf{dom}$ mit $\beta := \alpha_C \circ h$, d.h. $\beta(x) = \alpha_C\big(h(x)\big)$ für alle $x \in \text{Var}(Q')$.
Offensichtlicherweise gilt dann:

(1) $\beta(u') \ = \ \alpha_C(h(u')) \ = \ \alpha_C(u) \stackrel{\text{Def.}}{=} u_Q^{Q'}$.

(2) $\beta(\mathsf{T}') \ = \ \alpha_C(h(\mathsf{T}')) \ \stackrel{\text{Hom.}}{\subseteq} \ \alpha_C(\mathsf{T}) \ \stackrel{\text{Def.}}{=} \ \mathbf{I}_Q^{Q'}$.

Also ist $u_Q^{Q'} \in [\![Q']\!](\mathbf{I}_Q^{Q'})$.

„$\Longleftarrow$": Laut Voraussetzung gilt $u_Q^{Q'} \in [\![Q']\!](\mathbf{I}_Q^{Q'})$. Es gibt also eine Einbettung $\beta : \text{Var}(Q') \to \textbf{dom}$ von T' in $\mathbf{I}_Q^{Q'}$ mit $\beta(u') = u_Q^{Q'}$.
Unser Ziel ist, einen Homomorphismus h von Q' auf Q zu finden.
Betrachte dazu die Substitution $h : \text{Var}(Q') \to \textbf{var} \cup \textbf{dom}$ mit
$h := \alpha_C^{-1} \circ \beta$, d.h. $\alpha_C(h(x)) = \beta(x)$ gilt für alle $x \in \text{Var}(Q')$.
Um nachzuweisen, dass h ein Homomorphismus von Q' auf Q ist, müssen wir zeigen, dass Folgendes gilt:

(1) $h(u') = u$ und

(2) $h(\mathsf{T}') \subseteq \mathsf{T}$.

zu (1): $h(u') \;=\; \alpha_C^{-1}(\beta(u')) \;=\; \alpha_C^{-1}(u_Q^{Q'}) \;=\; u$.

zu (2): $h(\mathsf{T}') \;=\; \alpha_C^{-1}(\beta(\mathsf{T}'))$. Da β eine Einbettung von Q' in $\mathbf{I}_Q^{Q'}$ ist, gilt: $\beta(\mathsf{T}') \subseteq \mathbf{I}_Q^{Q'}$. Somit ist $h(\mathsf{T}') \;=\; \alpha_C^{-1}(\beta(\mathsf{T}')) \;\subseteq\; \alpha_C^{-1}\big(\mathbf{I}_Q^{Q'}\big) \;=\; \mathsf{T}$.

Also ist h ein Homomorphismus von Q' auf Q. □

Folie 106

Beispiel:
Sei $\mathbf{S} := \{R\}$, $Q' := (\mathsf{T}', (x,y))$ und $Q := (\mathsf{T}, (x,y))$ mit

$\mathsf{T}'(R) =$

A	B
x	y_1
x_1	y_1
x_1	y

$\mathsf{T}(R) =$

A	B
x	y
y	d

wobei $d \in \textbf{dom}$ ist.

Um die kanonischen Datenbanken und die kanonischen Tupel zu Q und Q' zu bilden, betrachten wir die Menge $C := \text{adom}(Q) \cup \text{adom}(Q') = \{d\}$. Für jedes $z \in \textbf{var}$ sei $a_z := \alpha_C(z)$. D.h. a_z ist die Konstante, die gemäß α_C die Variable z repräsentiert. Dann ist $d \notin \{a_z \;:\; z \in \textbf{var}\}$ und es gilt:

Kanonische Tupel: $u_{Q'}^{Q} = (a_x, a_y)$ und $u_Q^{Q'} = (a_x, a_y)$.

Kanonische Datenbanken:

$\mathbf{I}_{Q'}^{Q}(R) =$

A	B
a_x	a_{y_1}
a_{x_1}	a_{y_1}
a_{x_1}	a_y

$\mathbf{I}_Q^{Q'}(R) =$

A	B
a_x	a_y
a_y	d

Um herauszufinden, ob $Q \sqsubseteq Q'$ ist, genügt es laut Theorem 3.35 zu testen, ob $u_Q^{Q'} \in [\![Q']\!](\mathbf{I}_Q^{Q'})$ ist.
Genaues Betrachten der Anfrage Q' und der Datenbank $\mathbf{I}_Q^{Q'}$ zeigt, dass die Abbildung β mit $\beta(x) := \beta(x_1) := a_x$ und $\beta(y) := \beta(y_1) := a_y$ eine Einbettung von T' in $\mathbf{I}_Q^{Q'}$ ist, für die gilt: $\beta((x,y)) = (a_x, a_y) = u_Q^{Q'}$. Somit ist also $u_Q^{Q'} \in [\![Q']\!](\mathbf{I}_Q^{Q'})$. Gemäß Theorem 3.35 gilt also: $Q \sqsubseteq Q'$.

Um herauszufinden, ob $Q' \sqsubseteq Q$ ist, genügt es laut Theorem 3.35 zu testen, ob $u_{Q'}^{Q} \in [\![Q]\!](\mathbf{I}_{Q'}^{Q})$ ist.
Genaues Betrachten der Anfrage Q und der Datenbank $\mathbf{I}_{Q'}^{Q}$ zeigt, dass es keine Einbettung von T in $\mathbf{I}_{Q'}^{Q}$ gibt (denn angenommen, β wäre eine solche Einbettung, dann müsste das Tupel $(y,d) \in \mathsf{T}(R)$ durch β auf ein Tupel in $\mathbf{I}_{Q'}^{Q}(R)$ abgebildet werden — aber $\beta\big((y,d)\big) = \big(\beta(y), d\big)$, und in $\mathbf{I}_{Q'}^{Q}(R)$ gibt es kein Tupel, das in der zweiten Komponente den Eintrag d hat).
Somit ist also $[\![Q]\!](\mathbf{I}_{Q'}^{Q}) = \emptyset$. Daher ist $u_{Q'}^{Q} \notin [\![Q]\!](\mathbf{I}_{Q'}^{Q})$. Gemäß Theorem 3.35 gilt also: $Q' \not\sqsubseteq Q$.

Folie 107

Der Homomorphismus-Satz

Theorem 3.35 (Chandra, Merlin, 1977)**.**
Sei $\mathbf{S}$ *ein Datenbankschema und seien* $Q' = (\mathsf{T}', u')$ *und* $Q = (\mathsf{T}, u)$ *zwei Tableau-Anfragen über* $\mathbf{S}$*. Dann gilt:*

$$\begin{aligned} & Q \sqsubseteq Q' \\ \iff & \text{es gibt einen Homomorphismus von } Q' \text{ auf } Q \\ \iff & u_Q^{Q'} \in [\![Q']\!](\mathbf{I}_Q^{Q'}). \end{aligned}$$

Beweis.
Die Äquivalenz der letzten beiden Zeilen gilt gemäß Proposition 3.34.

Für die Richtung von der zweiten zur ersten Zeile gibt es laut Voraussetzung einen Homomorphismus h von Q' auf Q.
Sei $\mathbf{I}$ eine beliebige Datenbank vom Schema $\mathbf{S}$. Wir müssen zeigen, dass $[\![Q]\!](\mathbf{I}) \subseteq [\![Q']\!](\mathbf{I})$ ist.
Sei dazu t ein beliebiges Tupel in $[\![Q]\!](\mathbf{I})$. Wir müssen zeigen, dass $t \in [\![Q']\!](\mathbf{I})$ ist. Gemäß der Definition der Semantik von Tableau-Anfragen reicht es, eine Einbettung β' von T' in $\mathbf{I}$ finden, so dass $t = \beta'(u')$ ist.
Laut Voraussetzung ist $t \in [\![Q]\!](\mathbf{I})$. Es gibt also eine Einbettung β von T in $\mathbf{I}$, so dass $t = \beta(u)$ ist.

Außerdem gibt es laut Voraussetzung einen Homomorphismus h von Q' auf Q, d.h. eine Abbildung $h : \mathrm{Var}(Q') \to \mathbf{var} \cup \mathbf{dom}$, so dass $h(u') = u$ und $h(\mathsf{T}') \subseteq \mathsf{T}$ ist.
Wir wählen $\beta' := \beta \circ h$. D.h. β' ist die Abbildung von $\mathrm{Var}(Q')$ nach **dom** mit $\beta'(x) := \beta(h(x))$ für alle $x \in \mathrm{Var}(Q')$. Für dieses β' gilt:

- $\beta'(u') \;=\; \beta(h(u')) \;\stackrel{\text{Hom.}}{=}\; \beta(u) \;=\; t$ und
- $\beta'(\mathsf{T}') \;=\; \beta(h(\mathsf{T}')) \;\stackrel{\text{Hom.}}{\subseteq}\; \beta(\mathsf{T}) \;\subseteq\; \mathbf{I}$.

Somit ist β' eine Einbettung von T' in $\mathbf{I}$ mit $\beta'(u') = t$. Daher ist $t \in [\![Q']\!](\mathbf{I})$.

Für die Richtung von der ersten Zeile zur dritten Zeile gilt laut Voraussetzung, dass $Q \sqsubseteq Q'$. Unser Ziel ist, zu zeigen, dass für $\mathbf{I} := \mathbf{I}_Q^{Q'}$ gilt: $u_Q^{Q'} \in [\![Q']\!](\mathbf{I})$.
Wegen $Q \sqsubseteq Q'$ gilt insbesondere für die Datenbank $\mathbf{I} = \mathbf{I}_Q^{Q'}$, dass $[\![Q]\!](\mathbf{I}) \subseteq [\![Q']\!](\mathbf{I})$. Daher genügt es, zu zeigen, dass $u_Q^{Q'} \in [\![Q]\!](\mathbf{I})$ ist.
Sei β die Belegung mit $\beta(x) := \alpha_C(x)$ für alle $x \in \mathrm{Var}(Q)$. Dann gilt offensichtlicherweise:

- $\beta(u) \;=\; \alpha_C(u) \;\stackrel{\text{Def.}}{=}\; u_Q^{Q'}$ und
- $\beta(\mathsf{T}) \;=\; \alpha_C(\mathsf{T}) \;\stackrel{\text{Def.}}{=}\; \mathbf{I}_Q^{Q'}$.

Also ist β eine Einbettung von T in $\mathbf{I}_Q^{Q'}$ mit $\beta(u) = u_Q^{Q'}$. Gemäß der Definition der Semantik von Tableau-Anfragen ist also $u_Q^{Q'} \in [\![Q]\!](\mathbf{I}_Q^{Q'})$. $\square$

Korollar 3.36. *Das*

QUERY CONTAINMENT PROBLEM FÜR TABLEAU-ANFRAGEN	
Eingabe:	Tableau-Anfragen Q und Q' über einem DB-Schema $\mathbf{S}$
Frage:	Ist $Q \sqsubseteq Q'$ (d.h. gilt für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, dass $[\![Q]\!](\mathbf{I}) \subseteq [\![Q']\!](\mathbf{I})$) ?

ist NP*-vollständig.*

Beweis.
Zugehörigkeit zu NP*:*
Bei Eingabe von Q und Q' (und der Frage, ob $Q \sqsubseteq Q'$ ist), gehen wir wie folgt vor:

(1) Konstruiere die kanonische Datenbank $\mathbf{I}_Q^{Q'}$ und das kanonische Tupel $u_Q^{Q'}$.
Das geht deterministisch in Zeit polynomiell in der Größe von Q und Q'.

(2) Löse das Auswertungsproblem für die Datenbank $\mathbf{I}_Q^{Q'}$ und die Boolesche konjunktive Anfrage

$$\text{„Liegt } u_Q^{Q'} \text{ in } [\![Q']\!](\mathbf{I}_Q^{Q'}) \text{ ?“}$$

Dies geht nichtdeterministisch in polynomieller Zeit (durch „Raten“ einer Belegung β und anschließendes Verifizieren, dass $\beta(u') = u_Q^{Q'}$ und $\beta(\mathsf{T}') \subseteq \mathbf{I}_Q^{Q'}$ gilt).

NP-*Härte:*
Wir wissen bereits, dass das Auswertungsproblem für Boolesche Tableau-Anfragen NP-vollständig ist (siehe Theorem 3.21 und Lemma 3.14). Es reicht also, eine Polynomialzeit-Reduktion f vom Auswertungsproblem für Boolesche Tableau-Anfragen (kurz: AWP) aufs Query Containment Problem für Tableau-Anfragen (kurz: QCP) zu finden.
Unsere Reduktion f bildet eine Instanz $(\mathbf{I}, Q)$ des AWP auf die QCP-Instanz

$$f\big(\mathbf{I}, Q\big) \quad := \quad \big(P_{\mathbf{I}},\ Q\big)$$

ab, wobei $P_{\mathbf{I}} := (\mathsf{T}_{\mathbf{I}}, v)$ mit $v := ()$ und $\mathsf{T}_{\mathbf{I}} := \mathbf{I}$.
Offensichtlicherweise sind $P_{\mathbf{I}}$ und Q Boolesche Tableau-Anfragen, die bei Eingabe von $\mathbf{I}$ und Q in polynomieller Zeit konstruiert werden können.
Wir müssen also nur noch zeigen, dass Folgendes gilt:

$$\begin{array}{l}(\mathbf{I}, Q) \text{ ist eine „ja“-Instanz} \\ \text{fürs AWP, d.h. } () \in [\![Q]\!](\mathbf{I})\end{array} \iff \begin{array}{l}(P_{\mathbf{I}}, Q) \text{ ist eine „ja“-Instanz} \\ \text{fürs QCP, d.h. } P_{\mathbf{I}} \sqsubseteq Q.\end{array}$$

Gemäß Homomorphismussatz gilt:

$$\begin{aligned} P_{\mathbf{I}} \sqsubseteq Q \quad &\iff \quad \text{es gibt einen Homomorphismus von } Q \text{ auf } P_{\mathbf{I}} \\ &\iff \quad u_{P_{\mathbf{I}}}^{Q} \in [\![Q]\!](\mathbf{I}_{P_{\mathbf{I}}}^{Q}). \end{aligned}$$

Hierbei ist $u_{P_{\mathbf{I}}}^{Q} = ()$. Da $\mathsf{T}_{\mathbf{I}}$ keine Variablen enthält und $\mathsf{T}_{\mathbf{I}} = \mathbf{I}$ ist, gilt außerdem: $\mathbf{I}_{P_{\mathbf{I}}}^{Q} = \mathsf{T}_{\mathbf{I}} = \mathbf{I}$. Also gilt:

$$P_{\mathbf{I}} \sqsubseteq Q \quad \iff \quad () \ \in \ [\![Q]\!](\mathbf{I}).$$

Dies beendet den Beweis der NP-Härte des QCP. □

Bemerkung:
Für die im obigen Beweis konstruierte Tableau-Anfrage $P_{\mathbf{I}}$ gilt übrigens für jede Datenbank $\mathbf{J}$ vom Schema $\mathbf{S}$: $() \in [\![P_{\mathbf{I}}]\!](\mathbf{J}) \iff \mathbf{I} \subseteq \mathbf{J}$. D.h. bei Eingabe einer Datenbank $\mathbf{J}$ liefert die Anfrage $P_{\mathbf{I}}$ genau dann die Antwort „ja", wenn $\mathbf{J}$ eine Erweiterung der Datenbank $\mathbf{I}$ ist.

Bemerkung: Wegen

$$Q \equiv Q' \quad \iff \quad (\ Q \sqsubseteq Q' \ \text{ und } \ Q' \sqsubseteq Q\ )$$

gibt es laut Korollar 3.36 insbes. auch einen Algorithmus, der bei Eingabe zweier Tableau-Anfragen Q und Q' entscheidet, ob die beiden Anfragen äquivalent sind.

Folie 108

Tableau-Minimierung

Definition 3.37. Sei $\mathbf{S}$ ein Datenbankschema.

(a) Eine Tableau-Anfrage (T, u) heißt *minimal*, falls es keine zu (T, u) äquivalente Tableau-Anfrage (T', u') mit $|\mathsf{T}'| < |\mathsf{T}|$ gibt (wobei $|\mathsf{T}|$ die Kardinalität, d.h. die Gesamtzahl von Tupeln in T bezeichnet).

(b) Zwei Tableau-Anfragen $Q' := (\mathsf{T}', u')$ und $Q := (\mathsf{T}, u)$ heißen *isomorph*, falls es eine Bijektion $\pi : \text{var}(Q') \to \text{var}(Q)$ gibt, so dass $\pi(\mathsf{T}') = \mathsf{T}$ und $\pi(u') = u$ ist.

Theorem 3.38 (Chandra, Merlin, 1977)**.**

(a) Zu jeder Tableau-Anfrage (T, u) gibt es ein $\mathsf{T}' \subseteq \mathsf{T}$ (d.h. für jedes $R \in \mathbf{S}$ ist $\mathsf{T}'(R) \subseteq \mathsf{T}(R)$), so dass die Anfrage (T', u) minimal und äquivalent zu (T, u) ist.

(b) Sind (T_1, u_1) und (T_2, u_2) zwei minimale äquivalente Tableau-Anfragen, so sind (T_1, u_1) und (T_2, u_2) isomorph.

Beweis.
(a): Sei $Q := (\mathsf{T}, u)$ die gegebene Tableau-Anfrage, und sei $Q_1 = (\mathsf{T}_1, u_1)$ eine *minimale* Tableau-Anfrage, die äquivalent zu Q ist. Dann ist $Q_1 \sqsubseteq Q$ und $Q \sqsubseteq Q_1$. Gemäß Homomorphismussatz gibt es also einen Homomorphismus h von Q auf Q_1 und einen Homomorphismus g von Q_1 auf Q.

Skizze:

$$\begin{array}{ccccc} Q & & Q_1 & & Q \\ \shortparallel & h & \shortparallel & g & \shortparallel \\ (\mathsf{T}, u) & \longrightarrow & (\mathsf{T}_1, u_1) & \longrightarrow & (\mathsf{T}, u) \end{array}$$

Betrachte die Abbildung $f := g \circ h$. Dann ist f eine Abbildung von $\text{Var}(Q)$ auf $\mathbf{var} \cup \mathbf{dom}$ mit

(1) $f(u) \;=\; g(h(u)) \overset{h \text{ Hom.}}{=} g(u_1) \overset{g \text{ Hom.}}{=} u \quad$ und

(2) $f(\mathsf{T}) \;=\; g(h(\mathsf{T})) \overset{h \text{ Hom.}}{\subseteq} g(\mathsf{T}_1) \overset{g \text{ Hom.}}{\subseteq} \mathsf{T}$.

Sei $\mathsf{T}' := f(\mathsf{T})$ und sei $Q' := (\mathsf{T}', u)$. Wegen (2) ist $\mathsf{T}' \subseteq \mathsf{T}$. Um den Beweis von (a) abzuschließen, müssen wir noch Folgendes zeigen:

(i) Q' ist eine Tableau-Anfrage,

(ii) $Q' \equiv Q$ und

(iii) Q' ist minimal.

Für (i) müssen wir zeigen, dass jede Variable, die in u vorkommt, auch in T' vorkommt. Sei dazu x eine beliebige Variable in u. Da Q eine Tableaux-Anfrage ist, kommt x in T vor. Wegen $\mathsf{T}' = f(\mathsf{T})$ kommt daher $f(x)$ in T' vor. Und wegen $f(u) = u$ gilt $f(x) = x$. Somit kommt x in T' vor.

Für (ii) beachte, dass die Abbildung f ein Homomorphismus von Q auf Q' ist. Gemäß Homomorphismussatz ist also $Q' \sqsubseteq Q$. Umgekehrt ist die Abbildung id (mit $id(x) := x$ für alle $x \in \mathbf{var}$) ein Homomorphismus von Q' auf Q. Gemäß Homomorphismussatz ist also $Q \sqsubseteq Q'$. Insgesamt ist daher $Q \equiv Q'$.

Für (iii) beachte, dass gemäß (2) gilt: $\mathsf{T}' = f(\mathsf{T}) \subseteq g(\mathsf{T}_1)$. Insbesondere ist also $|\mathsf{T}'| \leqslant |\mathsf{T}_1|$. Da Q_1 eine *minimale* zu Q äquivalente Tableau-Anfrage ist und da Q' zu Q äquivalent ist, muss $|\mathsf{T}'| \geqslant |\mathsf{T}_1|$ sein. Insgesamt ist also $|\mathsf{T}'| = |\mathsf{T}_1|$, und somit ist Q' eine zu Q äquivalente *minimale* Tableau-Anfrage. Dies beendet den Beweis von (a).

(b): Ähnlich (Details: Übung). □

Folie 109

Theorem 3.39.

(a) Es gibt einen Algorithmus, der bei Eingabe einer Tableau-Anfrage $Q = (\mathsf{T}, u)$ eine minimale zu Q äquivalente Tableau-Anfrage (T', u) mit $\mathsf{T}' \subseteq \mathsf{T}$ berechnet.

(b) Das Problem

Eingabe: Tableau-Anfrage (T, u) und Tableau $\mathsf{T}' \subseteq \mathsf{T}$
Frage: Ist $(\mathsf{T}, u) \equiv (\mathsf{T}', u)$?

ist NP*-vollständig.*

Beweis.
(a): Sei $Q = (\mathsf{T}, u)$ die eingegebene Tableau-Anfrage und sei $\mathbf{S}$ das DB-Schema, über dem Q formuliert ist.
Für jedes $R \in \mathbf{S}$ sei $n_R := |\mathsf{T}(R)|$ die Anzahl der Tupel im Einzel-Tableau $\mathsf{T}(R)$, und sei $u_{R,1}, \ldots, u_{R,n_R}$ eine Auflistung aller Tupel in $\mathsf{T}(R)$.
Sei $\ell := |\mathbf{S}|$, und sei $R_1, \ldots, R_\ell$ eine Auflistung aller Relationsnamen in $\mathbf{S}$.
Unser Minimierungs-Algorithmus geht bei Eingabe von $Q = (\mathsf{T}, u)$ wie folgt vor:

(1) Setze $\hat{\mathsf{T}} := \mathsf{T}$ und $\hat{Q} := (\hat{\mathsf{T}}, u)$.

(2) Für $j = 1, \ldots, \ell$ tue Folgendes:

(3) Für $i = 1, \ldots, n_{R_j}$ tue Folgendes:

(4)
- Sei $\hat{\mathsf{T}}'$ das Tableau, das aus $\hat{\mathsf{T}}$ entsteht, indem das Tupel $u_{R_j,i}$ aus $\hat{\mathsf{T}}(R_j)$ gelöscht wird.
- Teste, ob jede Variable in u auch in $\hat{T}'$ vorkommt und wenn ja, ob für $\hat{Q}' := (\hat{\mathsf{T}}', u)$ gilt: $\hat{Q}' \sqsubseteq \hat{Q}$.
- Falls „ja“, so setze $\hat{\mathsf{T}} := \hat{\mathsf{T}}'$ und $\hat{Q} := \hat{Q}'$.

(5) Gib $\hat{Q} := (\hat{\mathsf{T}}, u)$ aus.

Wir müssen zeigen, dass für die vom Algorithmus ausgegebene Anfrage $\hat{Q}$ gilt:

(i) $\hat{Q} \equiv Q$ und

(ii) $\hat{Q}$ ist minimal.

An Stelle von (i) zeigen wir sogar, dass während des gesamten Laufs des Algorithmus $\hat{Q}$ stets äquivalent zu Q ist. Zu Beginn des Algorithmus gilt dies, da $\hat{Q} = Q$ ist. Gemäß Induktionsannahme gelte nun zu Beginn des Schleifendurchlaufs für j und i, dass $\hat{Q} \equiv Q$ ist. Wir müssen zeigen, dass auch am Endes dieses Schleifendurchlaufs $\hat{Q}$ äquivalent zu Q ist. Sei dazu $\hat{Q}_0$ die Anfrage $\hat{Q}$ zu Beginn des Schleifendurchlaufs, und sei $\hat{Q}_1$ die Anfrage $\hat{Q}$ am Ende des Schleifendurchlaufs.
Fall 1: Der Test während des Scheifendurchlaufs liefert die Antwort „nein“. Dann ist $\hat{Q}_1 = \hat{Q}_0$ und gemäß Induktionsannahme äquivalent zu Q.
Fall 2: Der Test während des Schleifendurchlaufs liefert die Antwort „ja“. Dann ist $\hat{Q}' \sqsubseteq \hat{Q}_0$. Außerdem ist das Tableau von $\hat{Q}'$ im Tableau von $\hat{Q}_0$ enthalten. Daher ist die identische Abbildung *id* ein Homomorphismus von $\hat{Q}'$ auf $\hat{Q}_0$. Laut Homomorphismus-Satz ist also $\hat{Q}_0 \sqsubseteq \hat{Q}'$. Insgesamt erhalten wir: $\hat{Q}' \equiv \hat{Q}_0 \equiv Q$. Am Ende des Schleifendurchlaufs wird $\hat{Q}$ durch $\hat{Q}'$ ersetzt. Also ist am Ende des Schleifendurchlaufs $\hat{Q} \equiv Q$.

Für (ii) führen wir einen Beweis durch Widerspruch. Angenommen, $\hat{Q}$ ist *nicht* minimal. Laut Theorem 3.38 gibt es ein Tableau $\mathsf{T}_1 \subseteq \hat{\mathsf{T}}$, so dass die Anfrage (T_1, u) minimal und äquivalent zu $\hat{Q}$ ist.
Da laut Annahme $\hat{Q}$ *nicht* minimal ist, ist $\hat{\mathsf{T}} \neq \mathsf{T}_1$.
Sei j_0 das kleinste $j \in \{1, \dots, \ell\}$, für das $\hat{\mathsf{T}}(R_j) \neq \mathsf{T}_1(R_j)$ ist.
Sei i_0 das kleinste $i \in \{1, \dots, n_{R_{j_0}}\}$, für das gilt:

$$u_{R_{j_0},i} \;\in\; \hat{\mathsf{T}}(R_{j_0}) \qquad \text{und} \qquad u_{R_{j_0},i} \;\notin\; \mathsf{T}_1(R_{j_0}). \tag{3.1}$$

Bei unserem Algorithmus ist zu Beginn des Schleifendurchlaufs für $j = j_0$ und $i = i_0$ dann $\hat{\mathsf{T}}$ das Tableau, bei dem für jedes $j' \in \{1, \dots, \ell\}$ gilt:

$$\hat{\mathsf{T}}(R_{j'}) \quad = \quad \begin{cases} \mathsf{T}_1(R_{j'}) & , \ \text{falls } j' < j_0 \\ \mathsf{T}_1(R_{j_0}) \cup \{u_{R_{j_0},i'} : i' \geqslant i_0\} & , \ \text{falls } j' = j_0 \\ \mathsf{T}(R_{j'}) & , \ \text{falls } j' > j_0\,. \end{cases}$$

Während des Schleifendurchlaufs für $j = j_0$ und $i = i_0$ ist dann $\hat{\mathsf{T}}'$ das Tableau, das aus $\hat{\mathsf{T}}$ entsteht, indem das Tupel $u_{R_{j_0},i_0}$ aus $\hat{\mathsf{T}}(R_{j_0})$ entfernt wird. Und es ist $\hat{Q}' = (\hat{\mathsf{T}}', u)$.

Gemäß (3.1) ist $u_{R_{j_0},i_0} \notin \mathsf{T}_1(R_{j_0})$.
Somit ist $\mathsf{T}_1 \subseteq \hat{\mathsf{T}}'$. Daher ist die identische Abbildung *id* ein Homomorphismus von Q_1 auf $\hat{Q}'$. Gemäß Homomorphismus-Satz ist also

$\hat{Q}' \sqsubseteq Q_1$. Und da $Q_1 \equiv Q$ ist, gilt: $\hat{Q}' \sqsubseteq Q$. Der im Algorithmus während des Schleifendurchlaufs für $j = j_0$ und $i = i_0$ durchgeführte Test auf $\hat{Q}' \sqsubseteq \hat{Q}$ liefert also die Antwort „ja“ (da, wie wir bereits gezeigt haben, $\hat{Q} \equiv Q$ gilt). Somit wird am Ende des Scheifendurchlaufs $\hat{\mathsf{T}} := \hat{\mathsf{T}}'$ gesetzt. Daher gilt für die am Ende vom Algorithmus ausgegebene Anfrage $\hat{Q} = (\hat{\mathsf{T}}, u)$, dass $u_{R_{j_0},i_0} \not\in \hat{\mathsf{T}}(R_{j_0})$ ist. Gemäß (3.1) müsste aber gelten: $u_{R_{j_0},i_0} \in \hat{\mathsf{T}}(R_{j_0})$. Widerspruch!
Dies beendet den Beweis von (ii) und damit auch den Beweis von (a).

(b): Übung. □

Folie 110

Optimierung von SPJR-Anfragen

Vorgehensweise bei Eingabe einer SPJR-Anfrage Q:

(1) Übersetze Q in eine Tableau-Anfrage (T, u) (bzw. gib „unerfüllbar“ aus, falls Q nicht erfüllbar ist)

(2) Finde minimales Tableau $\mathsf{T}' \subseteq \mathsf{T}$ so dass (T', u) äquivalent zu (T, u) ist.

(3) Übersetze (T', u) in eine äquivalente SPJR-Anfrage Q'

(4) Wende heuristische Optimierung (siehe Kapitel 6) auf Q' an und werte Q' aus.

Die Minimierung des Tableaus entspricht der Minimierung der Anzahl der Joins, denn: Anzahl Zeilen im Tableau = 1 + Anzahl Join-Operationen bei der Auswertung der Anfrage.

Beispiel 3.40.

- $\mathbf{S} = \{R\}$, wobei R die Attribute A, B, C hat.
- $Q \;:=\; \pi_{A,B}\big(\sigma_{B=\text{“5”}}(R)\big) \bowtie \pi_{B,C}\Big(\pi_{A,B}(R) \bowtie \pi_{A,C}(\sigma_{B=\text{“5”}}(R))\Big)$
- zugehörige Tableau-Anfrage: (T, u) mit $u = (x_A, 5, z_C)$ und

$$\mathsf{T}(R) \;=\; \begin{array}{|ccc} \\ \hline x_A & 5 & x_C \\ y_A & 5 & y_C \\ y_A & 5 & z_C \end{array}$$

- minimales Tableau T': Wir nutzen den Minimierungsalgorithmus aus dem Beweis von Theorem 3.39, um $Q = (\mathsf{T}, u)$ zu minimieren.
  - Setze $\hat{\mathsf{T}} := \mathsf{T}$ und $\hat{Q} := (\hat{T}, u)$. Setze $j := 1$ und $R_1 := R$. Betrachte die Schleifendurchläufe für $i = 1, \dots, 3$.
  - Schleifendurchlauf für $i = 1$: Sei $\hat{\mathsf{T}}'$ das Tableau mit

    $$\hat{\mathsf{T}}'(R) \;=\; \begin{array}{|ccc} & & \\ \hline y_A & 5 & y_C \\ y_A & 5 & z_C \end{array}$$

    Teste, ob für $\hat{Q}' := (\hat{\mathsf{T}}', u)$ gilt: $\hat{Q}' \sqsubseteq \hat{Q}$:
    Gemäß Homomorphismus-Satz genügt es dafür, zu testen, ob es einen Homomorphismus von $\hat{Q}$ auf $\hat{Q}'$ gibt. Angenommen, h wäre ein solcher Homomorphismus. Wegen $h(u) = u$ muss dann $h(x_A) = x_A$ und $h(z_C) = z_C$ sein. Die erste Zeile von $\hat{\mathsf{T}}(R) = \mathsf{T}(R)$ wird durch h dann abgebildet auf $(x_A, 5, h(x_C))$. Aber $\hat{\mathsf{T}}'(R)$ enthält keine solche Zeile. Daher kann es keinen Homomorphismus von $\hat{Q}$ auf $\hat{Q}'$ geben. Es ist also $\hat{Q}' \not\sqsubseteq \hat{Q}$. Am Ende des Schleifendurchlaufs ist also immer noch $\hat{\mathsf{T}} = \mathsf{T}$.
  - Schleifendurchlauf für $i = 2$: Sei $\hat{\mathsf{T}}'$ das Tableau mit

    $$\hat{\mathsf{T}}'(R) \;=\; \begin{array}{|ccc} & & \\ \hline x_A & 5 & x_C \\ y_A & 5 & z_C \end{array}$$

    Teste, ob für $\hat{Q}' := (\hat{\mathsf{T}}', u)$ gilt: $\hat{Q}' \sqsubseteq \hat{Q}$:
    Gemäß Homomorphismus-Satz genügt es dafür, zu testen, ob es einen Homomorphismus von $\hat{Q}$ auf $\hat{Q}'$ gibt. Angenommen, h wäre ein solcher Homomorphismus. Wegen $h(u) = u$ muss dann $h(x_A) = x_A$ und $h(z_C) = z_C$ sein. Wir wählen h so, dass außerdem gilt: $h(x_C) = x_C$ und $h(y_C) = z_C$. Man kann leicht nachprüfen, dass diese Abbildung h tatsächlich ein Homomorphismus von $\hat{Q}$ auf $\hat{Q}'$ ist. Es gilt also $\hat{Q}' \sqsubseteq \hat{Q}$. Am Ende des Schleifendurchlaufs wird also $\hat{\mathsf{T}}$ durch $\hat{\mathsf{T}}'$ ersetzt.
  - Schleifendurchlauf für $i = 3$: Sei $\hat{\mathsf{T}}'$ das Tableau mit

    $$\hat{\mathsf{T}}'(R) \;=\; \begin{array}{|ccc} & & \\ \hline x_A & 5 & x_C \end{array}$$

    Teste, ob für $\hat{Q}' := (\hat{\mathsf{T}}', u)$ gilt: $\hat{Q}' \sqsubseteq \hat{Q}$:
    Gemäß Homomorphismus-Satz genügt es dafür, zu testen, ob es einen Homomorphismus von $\hat{Q}$ auf $\hat{Q}'$ gibt. Angenommen, h wäre ein solcher

Homomorphismus. Wegen $h(u) = u$ muss dann $h(x_A) = x_A$ und $h(z_C) = z_C$ sein. Zur Erinnerung: Jetzt ist

$$\hat{\mathsf{T}}(R) \; = \; \begin{array}{|ccc} \\ \hline x_A & 5 & x_C \\ y_A & 5 & z_C \end{array}$$

Die letzte Zeile von $\hat{\mathsf{T}}(R)$ wird durch h dann abgebildet auf $(h(y_A), 5, z_C)$. Aber $\hat{\mathsf{T}}'(R)$ enthält keine solche Zeile. Daher kann es keinen Homomorphismus von $\hat{Q}$ auf $\hat{Q}'$ geben. Es ist also $\hat{Q}' \not\sqsubseteq \hat{Q}$. Am Ende des Schleifendurchlaufs bleibt also $\hat{\mathsf{T}}$ unverändert.

– Am Ende gibt der Algorithmus die minimale Tableau-Anfrage (T', u) aus mit $u = (x_A, 5, z_C)$ und

$$\hat{\mathsf{T}}'(R) \; = \; \begin{array}{|ccc} \\ \hline x_A & 5 & x_C \\ y_A & 5 & z_C \end{array}$$

- zugehörige SPJR-Anfrage:
$Q' \; := \; \pi_{A,B}\big(\sigma_{B=“5”}(R)\big) \; \bowtie \; \pi_{B,C}\big(\sigma_{B=“5”}(R)\big)$.

3.5 Azyklische Anfragen

Folie 111

Motivation

- *Ziel jetzt:* Teilklasse der Klasse der konjunktiven Anfragen, für die das Auswertungsproblem in Polynomialzeit lösbar ist (kombinierte Komplexität)
- $\rightsquigarrow$ *azyklische konjunktive Anfragen*
- Wir werden in diesem Kapitel oft nur *Boolesche* Anfragen betrachten (... wenn wir die effizient auswerten können, dann können wir die Konstruktion aus dem Beweis von Theorem 3.20 benutzen, um auch Anfragen auszuwerten, deren Ergebnis die Stelligkeit $\geqslant 1$ hat)
- *In der Literatur:* Verschiedene äquivalente Definitionen (bzw. Charakterisierungen) der azyklischen Booleschen konjunktiven Anfragen, etwa:

- regelbasierte konjunktive Anfragen mit azyklischem Hypergraph
- regelbasierte konjunktive Anfragen der Hyperbaum-Weite 1
- *regelbasierte konjunktive Anfragen, die einen Join-Baum besitzen*
- *Boolesche Semijoin-Anfragen*
- *konjunktive Sätze des Guarded Fragment*

Folie 112

Beispiel

Beispiel-Datenbank mit Relationen

- T mit Attributen *Student*, *Kurs*, *Semester* ..... T steht für „Teilnehmer“
- D mit Attributen *Prof*, *Kurs*, *Semester* .... D steht für „Dozent“
- E mit Attributen *Person1*, *Person2* .... steht für „Person1 ist Elternteil von Person2“

Anfrage 1: Gibt es einen Dozenten, dessen Tochter/Sohn an irgendeinem Kurs teilnimmt?
Als regelbasierte konjunktive Anfrage $Q_1 :=$

$$Ans() \ \leftarrow \ \ E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, y_K, y_Z)$$

Auswertung als „Semijoin-Anfrage“:
$\pi\Big(\big(E(x_P, x_S) \ltimes D(x_P, x_K, x_Z)\big) \ltimes T(x_S, y_K, y_Z)\Big)$

Anfrage 2: Gibt es einen Studenten, der an einem Kurs teilnimmt, der von seinem/r Vater/Mutter veranstaltet wird?
Als regelbasierte konjunktive Anfrage $Q_2 :=$

$$Ans() \ \leftarrow \ \ E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, x_K, x_Z)$$

Auswertung: $\pi\Big(E(x_P, x_S) \bowtie D(x_P, x_K, x_Z) \bowtie T(x_S, x_K, x_Z)\Big)$
Auswertung durch eine „Semijoin-Anfrage“ ist nicht möglich.

Folie 113

Join-Bäume & Azyklische regelbasierte konj. Anfragen

Definition 3.41.

(a) Sei $Q := \ Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ eine regelbasierte konjunktive Anfrage. Ein *Join-Baum* von Q ist ein (ungerichteter) *Baum* mit *Knotenmenge* $\{R_1(u_1), \ldots, R_\ell(u_\ell)\}$, so dass für alle Knoten $R_i(u_i)$ und $R_j(u_j)$ die folgende „*Weg-Eigenschaft*“ gilt:

> Jede Variable x, die sowohl in u_i als auch in u_j vorkommt, kommt in *jedem* Knoten vor, der auf dem (eindeutig bestimmten) kürzesten Weg zwischen $R_i(u_i)$ und $R_j(u_j)$ liegt.

(b) Eine regelbasierte konjunktive Anfrage Q (beliebiger Stelligkeit) heißt *azyklisch*, falls es einen Join-Baum für Q gibt.

Beispiele:

- $Q_1 := \ Ans() \leftarrow E(x_P, x_S), \ D(x_P, x_K, x_Z), \ T(x_S, y_K, y_Z)$

  besitzt folgenden Join-Baum:

  $$D(x_P, x_K, x_Z) — E(x_P, x_S) — T(x_S, y_K, y_Z)$$

  Ein Baum, der zwar die richtige Knotenmenge hat, aber trotzdem kein Join-Baum für Q_1 ist:

  $$E(x_P, x_S) — D(x_P, x_K, x_Z) — T(x_S, y_K, y_Z)$$

  Hier ist die Weg-Eigenschaft nicht erfüllt, da die Variable x_S in den beiden äußeren aber nicht im mittleren Knoten des Baums liegt.

- Es gibt keinen Join-Baum für
  $Q_2 := \ Ans() \leftarrow E(x_P, x_S), \ D(x_P, x_K, x_Z), \ T(x_S, x_K, x_Z)$

  denn: Die Knotenmenge eines Join-Baums für Q_2 muss genau aus den 3 Knoten $E(x_P, x_S)$, $D(x_P, x_K, x_Z)$ und $T(x_S, x_K, x_Z)$ bestehen. Auf diesen 3 Knoten gibt es nur die folgenden drei verschiedenen (ungerichteten) Bäume:

  $$E(x_P, x_S) — D(x_P, x_K, x_Z) — T(x_S, x_K, x_Z)$$
  $$D(x_P, x_K, x_Z) — E(x_P, x_S) — T(x_S, x_K, x_Z)$$
  $$D(x_P, x_K, x_Z) — T(x_S, x_K, x_Z) — E(x_P, x_S)$$

Keiner dieser 3 Bäume erfüllt die Weg-Eigenschaft. Daher gibt es keinen Join-Baum für Q_2.

- $Q_3 := \ \mathit{Ans}(x,y) \leftarrow E_1(x,y),\ E_2(y,z),\ E_3(z,x),\ E_4(x,y,z)$

  besitzt einen Join-Baum, nämlich den Baum, bei dem der Knoten $E_4(x,y,z)$ zu jedem der Knoten $E_1(x,y)$, $E_2(y,z)$, $E_3(z,x)$ eine Kante hat.

- $Q_4 :=$
  $\mathit{Ans}() \leftarrow R(y,z),\ P(x,y),\ S(y,z,u),\ S(z,u,w),\ T(y,z),\ T(z,u),\ R(z',y')$

  besitzt einen Join-Baum, nämlich den Baum, bei dem der Knoten $R(y,z)$ eine Kante zu jedem der Knoten $P(x,y)$, $S(y,z,u)$, $R(z',y')$ hat, und der Knoten $S(y,z,u)$ eine Kante zu jedem der Knoten $S(z,u,w)$, $T(y,z)$, $T(z,u)$ hat.

Folie 114

Effiziente Auswertung von azyklischen Booleschen konjunktiven Anfragen

Vorgehensweise:

Eingabe: Boolesche regelbasierte konjunktive Anfrage Q, Datenbank $\mathbf{I}$

Ziel: Berechne $[\![Q]\!](\mathbf{I})$

(1) Teste, ob Q azyklisch ist und konstruiere ggf. einen Join-Baum T für Q.
(Details dazu: später)

(2) Nutze T zur Konstruktion einer Booleschen *Semijoin-Anfrage* Q', die äquivalent zu Q ist.
(Details dazu: gleich)

(3) Werte Q' in $\mathbf{I}$ aus
(das geht gemäß Proposition 3.43 in Zeit $\mathcal{O}\big(\|Q'\|\cdot\|\mathbf{I}\|\cdot\log(\|\mathbf{I}\|)\big)$)

Folie 115

Semijoin-Anfragen

Definition 3.42. Sei $\mathbf{S}$ ein Datenbankschema.
Die Klasse der *Semijoin-Anfragen über* $\mathbf{S}$ ist induktiv wie folgt definiert:

(A) Jedes Relations-Atom $R(v_1,\ldots,v_r)$, für $R \in \mathbf{S}$, $r := \mathrm{ar}(R)$ und $v_1,\ldots,v_r \in \mathbf{var} \cup \mathbf{dom}$ ist eine Semijoin-Anfrage der Sorte $(v_1,\ldots,v_r)$.

Semantik: Für jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ ist $[\![R(v_1,\ldots,v_r)]\!](\mathbf{I}) :=$

$$\left\{ (\beta(v_1),\ldots,\beta(v_r)) \ : \ \begin{array}{l} \beta : (\{v_1,\ldots,v_r\} \cap \mathbf{var}) \to \mathbf{dom} \text{ ist eine} \\ \text{Belegung, so dass} \\ (\beta(v_1),\ldots,\beta(v_r)) \in \mathbf{I}(R) \end{array} \right\}$$

(S) Sind Q_1 und Q_2 Semijoin-Anfragen der Sorten $(v_1,\ldots,v_r)$ und $(v'_1,\ldots,v'_s)$, so ist $Q := (Q_1 \ltimes Q_2)$ eine Semijoin-Anfrage der Sorte $(v_1,\ldots,v_r)$.

Semantik: Für jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ ist $[\![Q]\!](\mathbf{I}) :=$

$$\left\{ (a_1,\ldots,a_r) \in [\![Q_1]\!](\mathbf{I}) \ : \ \begin{array}{l} \text{es gibt ein } (b_1,\ldots,b_s) \in [\![Q_2]\!](\mathbf{I}), \\ \text{so dass} \\ \text{für alle } i,j \text{ mit } v_i = v'_j \in \mathbf{var} \\ \text{gilt: } a_i = b_j \end{array} \right\}$$

Eine *Boolesche Semijoin-Anfrage* über $\mathbf{S}$ ist von der Form $\pi(Q)$, wobei Q eine Semijoin-Anfrage über $\mathbf{S}$ ist.

Beispiele.

(a) Seien x,y,z drei verschiedene Variablen und seien c,d zwei verschiedene Konstanten. Betrachte die Semijoin-Anfragen $Q_1 := R(x,x,c)$ und $Q_2 := S(d,x,y,z)$ und $Q_3 := (Q_1 \ltimes Q_2)$. Für jede Datenbank $\mathbf{I}$ vom Schema $\{R,S\}$ gilt dann:

- $[\![Q_1]\!](\mathbf{I}) = \{t = (t_1,t_2,t_3) \in \mathbf{I}(R) \ : \ t_1 = t_2 \text{ und } t_3 = c\}$
- $[\![Q_2]\!](\mathbf{I}) = \{s = (s_1,s_2,s_3,s_4) \in \mathbf{I}(S) \ : \ s_1 = d\}$
- $[\![Q_3]\!](\mathbf{I}) = \{t = (t_1,t_2,t_3) \in [\![Q_1]\!](\mathbf{I}) \ : \ \text{ex.}$
  $s = (s_1,s_2,s_3,s_4) \in [\![Q_2]\!](\mathbf{I}) \text{ s.d. } t_1 = s_2 \text{ und } t_2 = s_2 \ \}$.

(b) Seien x, y, z, z' vier verschiedene Variablen. Die Semijoin-Anfrage

$$((R(x,y) \ltimes S(y,z)) \ltimes T(x,z))$$

ist äquivalent zur Semijoin-Anfrage

$$((R(x,y) \ltimes S(y,z)) \ltimes T(x,z'))$$

und sie ist *nicht* äquivalent zur Semijoin-Anfrage

$$(R(x,y) \ltimes (S(y,z) \ltimes T(x,z))$$

Folie 116

Auswertung von Semijoin-Anfragen

Proposition 3.43. *Das Auswertungsproblem für Semijoin-Anfragen bzw. Boolesche Semijoin-Anfragen ist in Zeit* $\mathcal{O}(k \cdot n \cdot \log n)$ *lösbar, wobei k die Größe der eingegebenen Anfrage und n die Größe der eingegebenen Datenbank ist.*

Beweis.
Bei Eingabe einer Semijoin-Anfrage Q und einer Datenbank $\mathbf{I}$ nutzen wir folgenden rekursiven Algorithmus zur Berechnung von $[\![Q]\!](\mathbf{I})$.

<u>AUSWERTUNG(Q,$\mathbf{I}$):</u>
Eingabe: Semijoin-Anfrage Q und Datenbank $\mathbf{I}$ über dem DB-Schema $\mathbf{S}$.
Ziel: Berechne $[\![Q]\!](\mathbf{I})$.

1. Falls Q von der Form $R(v_1, \ldots, v_r)$ für ein $R \in \mathbf{S}$ ist, $r = \mathrm{ar}(R)$ und $v_1, \ldots, v_r \in \mathbf{var} \cup \mathbf{dom}$ ist, so gib $[\![Q']\!](\mathbf{I})$ für die regelbasierte konjunktive Anfrage $Q' := \;\; Ans(v_1, \ldots, v_r) \leftarrow R(v_1, \ldots, v_r)$ aus.
2. Falls Q von der Form $(Q_1 \ltimes Q_2)$ ist, wobei Q_1 und Q_2 Semijoin-Anfragen der Sorten $(v_1, \ldots, v_r)$ und $(v'_1, \ldots, v'_s)$ sind, so tue Folgendes:
   (a) Berechne $J_1 := [\![Q_1]\!](\mathbf{I})$ und $J_2 := [\![Q_2]\!](\mathbf{I})$ (durch Aufruf von AUSWERTUNG($Q_i, \mathbf{I}$) für $i \in \{1,2\}$), wähle eine geeignete Reihenfolge der Spalten von J_1 und J_2 und sortiere J_1 und J_2 gemäß der entsprechenden lexikographischen Ordnung.

(b) Berechne $J_1 \ltimes J_2 := \{(a_1, \ldots, a_r) \in J_1 \ : \ \text{es gibt ein } (b_1, \ldots, b_s) \in J_2 \text{ so dass für alle } i \in \{1, \ldots, r\} \text{ und } j \in \{1, \ldots, s\} \text{ mit } v_i = v'_j \in \textbf{var} \text{ gilt: } a_i = b_j \ \}$.

Nutze dazu die „Merge-Methode“, die auch beim Merge-Sort-Algorithmus verwendet wird: Lass einen Zeiger entlang der sortierten Version von J_1 und einen Zeiger entlang der sortierten Version von J_2 laufen; starte mit beiden Zeigern auf dem ersten Tupel der jeweiligen sortierten Listen von Tupeln; usw. Details: Übung!

Wenn die Details dieses Algorithmus auf die richtige Art ausgeführt werden, berechnet der Algorithmus $\textsc{Auswertung}(Q, \mathbf{I})$ offensichtlicherweise das richtige Ergebnis $[\![Q]\!](\mathbf{I})$.

Laufzeitanalyse: Wir schreiben $A(k, n)$ um eine obere Schranke für die Laufzeit des Algorithmus bei Eingabe einer Anfrage der Größe k und einer Datenbank der Größe n zu bezeichnen. Als Maß für die Größe einer Anfrage nehmen wir hierbei die Anzahl der in der Anfrage vorkommenden Atome und $\ltimes$-Symbole.
Gemäß Punkt 1. des Algorithmus gilt:

$$A(1, n) \ \leqslant \ n \tag{3.2}$$

In Punkt 2. des Algorithmus gilt $k = k_1 + k_2 + 1$, wobei k, k_1, k_2 die Größen der Anfragen Q, Q_1, Q_2 sind. Und es gilt:

$$A(k, n) \ \leqslant \ A(k_1, n) + A(k_2, n) + (r{+}s) \cdot \log(r{+}s) + n \cdot \log(n) + n \cdot \log(n) + 2n$$

Hierbei ist

- $A(k_i, n)$ der Aufwand zur Berechnung von J_i (für $i \in \{1, 2\}$),
- $(r{+}s) \cdot \log(r{+}s)$ der Aufwand zum Finden einer geeigneten Reihenfolge der Spalten
- $n \cdot \log(n)$ der Aufwand zum Sortieren von J_i (für $i \in \{1, 2\}$); beachte dazu, dass $\|J_i\| \leqslant \|\mathbf{I}\| \leqslant n$ ist
- $2n$ der Aufwand für Punkt 2.(b) des Algorithmus.

Wegen $r{+}s \leqslant 2n$ gilt also:

$$A(k, n) \ \leqslant \ A(k_1, n) + A(k_2, n) + 6n \cdot \log(n). \tag{3.3}$$

Das Auflösen der Rekursions(un)gleichung, die sich aus (3.2) und (3.3) ergibt, liefert (für alle $k, n \in \mathbb{N}_{\geqslant 1}$):

$$A(k,n) \;\leqslant\; 6kn \cdot \log(n) \tag{3.4}$$

Beweis: Induktionsanfang $k = 1$: Aus (3.2) folgt: $A(1,n) \leqslant 6n \cdot \log(n)$.
Induktionsschritt $k > 1$: Seien $k_1, k_2 < k$ so dass $k = k_1 + k_2 + 1$. Aus (3.3) und der Induktionsannahme folgt:

$$\begin{aligned} A(k,n) &\leqslant A(k_1,n) + A(k_2,n) + 6n \cdot \log(n) \\ &\leqslant 6k_1 n \log(n) \;+\; 6k_2 n \log(n) \;+\; 6n \cdot \log(n) \\ &= 6(k_1{+}k_2{+}1)n \cdot \log(n) \end{aligned}$$

Dies beendet den Beweis von Proposition 3.43. □

Folie 117

Semijoin-Anfragen vs. Join-Bäume

Lemma 3.44.

(a) *Es gibt einen Algorithmus, der bei Eingabe einer Semijoin-Anfrage Q in Zeit $\mathcal{O}(\|Q\|)$ eine zu Q äquivalente azyklische regelbasierte konjunktive Anfrage Q' und einen Join-Baum für Q' berechnet.*

(b) *Es gibt einen Algorithmus, der bei Eingabe einer azyklischen Booleschen regelbasierten konjunktiven Anfrage Q und eines Join-Baums T für Q in Zeit $\mathcal{O}(\|Q\|)$ eine zu Q äquivalente Boolesche Semijoin-Anfrage Q' berechnet.*

Beweis: (a): Übung. (b): siehe unten. □

Folgerung: Mit azyklischen Booleschen regelbasierten konjunktiven Anfragen kann man genau dieselben Anfragefunktionen ausdrücken wie mit Booleschen Semijoin-Anfragen.

Vorsicht: Dies gilt nicht, wenn man an Stelle von *Booleschen Anfragen* Anfragen beliebiger Stelligkeit betrachtet.

Beweis von Lemma 3.44(b):
Seit T der gegebene Join-Baum von Q. Wähle einen beliebigen Knoten w von T aus und lege fest, dass dies ab jetzt die *Wurzel* von T sein soll.
Wir nutzen folgende Notationen für jeden Knoten v von T:

- $R_v(u_v)$ bezeichnet das Relations-Atom, aus dem v besteht
- $\text{var}(v)$ bezeichnet die Menge aller Variablen, die in u_v vorkommen
- T_v bezeichnet den Teilbaum von T (betrachtet als Baum mit Wurzel w), der in Knoten v gewurzelt ist
- wir schreiben $v' \in T_v$ um auszudrücken, dass v' ein Knoten in T_v ist.

Per Induktion über T (betrachtet als Baum mit Wurzel w), beginnend mit den Blättern von T, konstruieren wir für jeden Knoten v von T eine Semijoin-Anfrage Q'_v der Sorte u_v für die gilt:

$$(*) \quad Q'_v \text{ ist äquivalent zur Anfrage } Q_v := \ Ans(u_v) \ \leftarrow \ \underbrace{\Big(R_{v'}(u_{v'})\Big)_{v' \in T_v}}_{\text{Liste aller Atome in } T_v}$$

Insbesondere folgt dann für die Wurzel w von T, dass die Boolesche Semijoin-Anfrage $Q' := \pi(Q'_w)$ äquivalent ist zur Booleschen Anfrage $Ans() \ \leftarrow \ \big(R_v(u_v)\big)_{v \in T}$. Und dies ist ja genau die Anfrage Q.

Induktionsanfang: Für jedes Blatt v von T setzen wir $Q'_v := R_v(u_v)$. Mit dieser Wahl ist $(*)$ offensichtlicherweise erfüllt.

Induktionsschritt: Sei v ein Knoten von T (betrachtet als Baum mit Wurzel w) und seien $v_1, \dots, v_k$ die Kinder von v. Gemäß Induktionsannahme haben wir bereits für jedes $i \in \{1, \dots, k\}$ eine Semijoin-Anfrage Q'_{v_i} konstruiert, die $(*)$ erfüllt. Wir wählen

$$Q'_0 \ := \ R_v(u_v)$$

und für jedes $i \in \{1, \dots, k\}$ wählen wir

$$Q'_i \ := \ (Q'_{i-1} \ltimes Q'_{v_i}).$$

Und wir setzen

$$Q'_v \ := \ Q'_k.$$

Klar: Jede der Anfragen $Q'_0, \dots, Q'_k$ ist eine Semijoin-Anfrage der Sorte u_v.

Behauptung $(**)$*:* Für jedes $i \in \{0, \dots, k\}$ ist Q'_i äquivalent zur Anfrage $Q_i := \ Ans(u_v) \leftarrow Rumpf_i$ wobei $Rumpf_i$ die Liste ist, die aus dem Atom $R_v(u_v)$ und sämtlichen Atomen, die in einem der Bäume T_{v_j} für $j \in \{1, \dots, i\}$ vorkommen, besteht.

Beweis von Behauptung $(**)$*:* Per Induktion nach i.
Der Induktionsanfang $i = 0$ ist offensichtlich.

Für den Induktionsschritt $i \to i{+}1$ nutzen wir die Tatsache, dass der Join-Baum T die *Weg-Eigenschaft* erfüllt, d.h. jede Variable, die sowohl in einem Atom von $T_{v_{j_1}}$ als auch in einem Atom von $T_{v_{j_2}}$ für zwei verschiedene $j_1, j_2 \in \{1, \dots, k\}$ vorkommt, muss auch in u_v und in $u_{v_{j_1}}$ und in $u_{v_{j_2}}$ vorkommen. *Details: Übung!* $\square_{Beh.(**)}$

Aus Behauptung $(**)$ folgt, dass $(*)$ für die Semijoin-Anfrage Q'_v erfüllt ist. Insbesondere für den Knoten $v := w$ liefert dies, dass die Boolesche Semijoin-Anfrage $Q' = \pi(Q'_w)$ äquivalent zur gegebenen Booleschen Anfrage Q ist.
Ein Algorithmus zur Konstruktion von Q' kann während eines Bottom-Up-Durchlaufs durch den in w gewurzelten Baum T die Anfrage Q' in Zeit linear in der Anzahl der Knoten von T erzeugen. Dies beendet den Beweis von Lemma 3.44(b). $\square$

Beispiele. Seien T_1 und T_4 die im Beispiel nach Definition 3.41 beschriebenen Join-Bäume der Anfragen Q_1 und Q_4.

(a) Wenn wir für T_1 den Knoten $E(x_P, x_S)$ als Wurzel auswählen, erhalten wir aus dem Beweis von Lemma 3.44(b) die folgende zu Q_1 äquivalente Boolesche Semijoin-Anfrage:

$$\pi\Big(((E(x_P, x_S) \ltimes D(x_P, x_K, x_Z)) \ltimes T(x_S, y_K, y_Z))\Big)$$

Wenn wir stattdessen den Knoten $D(x_P, x_K, x_Z)$ als Wurzelknoten von T_1 auswählen, erhalten wir aus dem Beweis von Lemma 3.44(b) die folgende zu Q_1 äquivalente Boolesche Semijoin-Anfrage:

$$\pi\Big((D(x_P, x_K, x_Z) \ltimes (E(x_P, x_S) \ltimes T(x_S, y_K, y_Z)))\Big)$$

(b) Wenn wir für T_4 den Knoten $R(y, x)$ als Wurzel auswählen, erhalten wir aus dem Beweis von Lemma 3.44(b) die folgende zu Q_4 äquivalente Boolesche Semijoin-Anfrage:

$$\pi\Big((((R(y,z) \ltimes P(x,y)) \ltimes (((S(y,z,u) \ltimes S(z,u,w)) \ltimes T(y,z)) \ltimes T(z,u))) \ltimes R(z',y'))\Big)$$

Folie 118

Konstruktion eines Join-Baums

Lemma 3.45. *Es gibt einen Polynomialzeit-Algorithmus, der bei Eingabe einer regelbasierten konjunktiven Anfrage Q entscheidet, ob Q azyklisch ist und ggf. einen Join-Baum für Q konstruiert.*

Beweis:

Algorithmus: **Eingabe: Anfrage Q der Form $\boldsymbol{Ans}(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$**

(1) $V := \{R_1(u_1), \ldots, R_\ell(u_\ell)\}$ .... *Knotenmenge*

(2) $E := \emptyset$ .... *Kantenmenge*

(3) alle Elemente von V sind *unmarkiert*

(4) alle Variablen sind *unmarkiert*

(5) *Wiederhole* so lange, bis sich nichts mehr ändert:

- (5.1) Falls es *unmarkierte Knoten* $R_i(u_i)$ und $R_j(u_j)$ (mit $i \neq j$) gibt, so dass alle *unmarkierten Variablen* aus u_j in u_i vorkommen, so *markiere den Knoten* $R_j(u_j)$ und füge in E eine *Kante zwischen* $R_i(u_i)$ *und* $R_j(u_j)$ ein.
- (5.2) *Markiere* sämtliche *Variablen* x, für die gilt:
„Es gibt *genau einen unmarkierten Knoten*, in dem x vorkommt.“

(6) Falls es *nur noch einen unmarkierten Knoten* gibt, so gib (V, E) aus; sonst gib aus: „Q ist nicht azyklisch“.

Folie 119

Einige Details zum Beweis von Lemma 3.45

Beispiel: Probelauf des Algorithmus für die Anfragen

- $Q_1 := \ Ans() \leftarrow E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, y_K, y_Z)$
- $Q_2 := \ Ans() \leftarrow E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, x_K, x_Z)$
- $Q_3 := \ Ans() \leftarrow$
$R(y,z),\ P(x,y),\ S(y,z,u),\ S(z,u,w),\ T(y,z),\ T(z,u),\ R(z',y')$

Notation:

- *Zeitpunkt t* = Beginn des t-ten Durchlaufs durch Zeile (5)
- $w_1^t, \ldots, w_{r_t}^t$: die zu Zeitpunkt t noch unmarkierten Knoten
- MV^t : Menge der zum Zeitpunkt t bereits markierten Variablen
- E^t : die Kantenmenge zum Zeitpunkt t

Folie 120

Die Korrektheit des Algorithmus folgt direkt aus den folgenden Behauptungen 1 & 2:

Behauptung 1: Zu jedem Zeitpunkt t gilt:

$(1)_t$: E^t *ist ein Wald* aus Bäumen $T_1^t, \ldots, T_{r_t}^t$, deren Wurzeln die Knoten $w_1^t, \ldots, w_{r_t}^t$ sind.

$(2)_t$: Jeder dieser Bäume erfüllt die *Weg-Eigenschaft*, d.h. für alle $i \in \{1, \ldots, r_t\}$, alle Knoten $v, v' \in T_i^t$ und jede Variable x, die sowohl in v als auch in v' vorkommt, gilt: x kommt in jedem Knoten auf dem Weg zwischen v und v' vor.

$(3)_t$: Jede *unmarkierte Variable* (d.h. jede Variable, die nicht zu MV^t gehört), die in einem Baum T_k^t vorkommt, kommt auch in dessen Wurzel w_k^t vor.

$(4)_t$: Es gibt keine *markierte Variable* (d.h. aus MV^t), die in 2 verschiedenen Bäumen T_i^t und T_j^t vorkommt.

Beweis: Induktion nach t.
$t = 1$: klar. $t \mapsto t{+}1$: Nachrechnen (Übung).

Behauptung 2:
Wenn Q azyklisch ist, so endet der Algorithmus mit nur *einem* unmarkierten Knoten.
Beweis: Siehe Tafel. $\square$

Folie 121

Auswertungskomplexität azyklischer konjunktiver Anfragen

Theorem 3.46 (Yannakakis, 1981). *Das*

> AWP FÜR AZYKLISCHE REGELBASIERTE KONJ. ANFRAGEN
> *Eingabe:* Regelbasierte konjunktive Anfrage Q und Datenbank $\mathbf{I}$
> *Aufgabe:* Falls Q ayzklisch ist, so berechne $[\![Q]\!](\mathbf{I})$;
> ansonsten gib „Q ist nicht azyklisch“ aus.

kann in Zeit polynomiell in $\|Q\| + \|\mathbf{I}\| + \|[\![Q]\!](\mathbf{I})\|$ *gelöst werden.*

Beweis:

- ***Algorithmus für Boolesche Anfragen:*** Nutze Lemma 3.45, Lemma 3.44 (b) und Proposition 3.43.
- ***Algorithmus für Anfragen beliebiger Stelligkeit:*** Nutze den Algorithmus für Boolesche Anfragen und die Konstruktion aus dem Beweis von Theorem 3.20. Beachte dabei, dass sämtliche Boolesche Anfragen, die zur Auswertung einer azyklischen Anfrage Q gestellt werden, denselben Join-Baum besitzen wie Q und daher insbesondere azyklisch sind. $\square$

Bemerkung: Es ist bekannt, dass das Auswertungsproblem für Boolesche azyklische regelbasierte konjunktive Anfragen vollständig ist für die Komplexitätsklasse LOGCFL *(Gottlob, Leone, Scarcello, 1998).*

Folie 122

Konjunktives Guarded Fragment $\mathsf{GF}(\mathsf{CQ})$

Definition 3.47. Sei $\mathbf{S}$ ein Datenbankschema.

Mit $\mathsf{GF}(\mathsf{CQ})[\mathbf{S}]$ bezeichnen wir die Menge aller Formeln des konjunktiven Kalküls $\mathsf{CQ}[\mathbf{S}]$ (vgl. Definition 3.8), die zum Guarded Fragment $\mathsf{GF}[\mathbf{S}]$ gehören, d.h. ... (Details: siehe Tafel)

Konjunktive *Sätze* des Guarded Fragment sind Formeln aus $\mathsf{GF}(\mathsf{CQ})[\mathbf{S}]$, die keine freie(n) Variable(n) besitzen.

Folie 123

Azyklische Boolesche Konjunktive Anfragen

Satz 3.48. *Die folgenden Anfragesprachen können genau dieselben Booleschen Anfragefunktionen ausdrücken:*

(a) azyklische Boolesche regelbasierte konjunktive Anfragen,

(b) Boolesche Semijoin-Anfragen,

(c) konjunktive Sätze des Guarded Fragment.

Und jede Anfrage aus einer dieser Anfragesprachen kann in polynomieller Zeit in äquivalente Anfragen der anderen Sprachen übersetzt werden.

Gemäß Theorem 3.46 ist das Auswertungsproblem (kombinierte Komplexität) also für jede dieser Anfragesprachen in Polynomialzeit lösbar.

Beweis: (a) $\iff$ (b): Lemma 3.44. (a) $\iff$ (c): Übung. □

3.6 Mengen-Semantik vs. Multimengen-Semantik

Folie 124

Motivation

bisher: *Mengen-Semantik* (engl.: *set semantics*):

- DB-Relation = eine Menge von Tupeln
- Duplikate eines Tupels werden eliminiert

$$\{t\} \ = \ \{t,t\} \ = \ \{t,t,t\} \ = \ \cdots$$

in SQL:

- keine Duplikat-Elimination bei Anfragen der Form
  `SELECT * FROM ... WHERE ...`
- falls Duplikat-Elimination explizit gewünscht:
  `SELECT DISTINCT * FROM ... WHERE ...`

betrachte jetzt: *Multimengen-Semantik* (engl: *bag semantics*):

$$\{t\} \ \neq \ \{t,t\} \ \neq \ \{t,t,t\} \ \neq \ \cdots$$

Folie 125

Beispiel

Datenbankschema:

- 2-stellige Relation *Hersteller* mit Attributen *Name* und *Ort*
- 2-stellige Relation *Bauteil* mit Attributen *Teil* und *Lager*

„Datenbank" $\mathbf{I}_F$ mit

$\mathbf{I}_F(\boldsymbol{Hersteller})$

Name	*Ort*
Boeing	Seattle
Boeing	New York
Airbus	Hamburg

Notation:

- $|\,(\text{Boeing,Seattle})\,|_{\mathbf{I}_F(Hersteller)} \;=\; 1$
- $|\,(\text{Boeing,New York})\,|_{\mathbf{I}_F(Hersteller)} \;=\; 1$
- $|\,(\text{Airbus,Hamburg})\,|_{\mathbf{I}_F(Hersteller)} \;=\; 1$

$\mathbf{I}_F(\boldsymbol{Bauteil})$

Teil	*Lager*
Motor	Seattle
Motor	Seattle
Flügel	Portland
Cockpit	Seattle
Cockpit	Seattle
Cockpit	Seattle

Notation:

- $|\,(\text{Motor,Seattle})\,|_{\mathbf{I}_F(Bauteil)} \;=\; 2$
- $|\,(\text{Flügel,Portland})\,|_{\mathbf{I}_F(Bauteil)} \;=\; 1$
- $|\,(\text{Cockpit,Seattle})\,|_{\mathbf{I}_F(Bauteil)} \;=\; 3$

Folie 126

Multimengen (Bags) und Multimengen-Datenbanken

Sei M eine Menge.

- Eine *Multimenge B über M* ist eine Abbildung $B : M \to \mathbb{N}_{\geqslant 0}$
- Notation: Für $a \in M$ schreibe $|a|_B$ an Stelle von $B(a)$
  $|a|_B = i$ *bedeutet: das Element a kommt i-mal in der Multimenge B vor.*
- B heißt *endlich*, falls die Menge $\{a \in M \ : \ |a|_B \neq 0\}$ endlich ist.
- Für Multimengen B und B' über M gilt:
  - $B =_b B' \quad :\iff \quad |a|_B = |a|_{B'}$, für alle $a \in M$
  - $B \subseteq_b B' \quad :\iff \quad |a|_B \leqslant |a|_{B'}$, für alle $a \in M$
  - Insbesondere gilt: $B =_b B' \iff \Big(B \subseteq_b B' \text{ und } B' \subseteq_b B\Big)$
  - $B \cup_b B' \quad :=$
    die Multimenge B'' über M mit $|a|_{B''} := |a|_B + |a|_{B'}$, für alle $a \in M$

Definition 3.49. Sei $\mathbf{S}$ ein Datenbankschema.
Eine *Multimengen-Datenbank* $\mathbf{I} \in inst_b(\mathbf{S})$ ordnet jedem Relationssymbol $R \in \mathbf{S}$ eine endliche Multimenge $\mathbf{I}(R)$ über $\mathbf{dom}^{\mathrm{ar}(R)}$ zu.

Folie 127

Anfragen mit Mulitmengen-Semantik

Beispiel:

SQL-Anfrage:

```
SELECT B1.Teil, B2.Teil
FROM Bauteil B1, Bauteil B2
WHERE B1.Lager = B2.Lager
```

regelbasiert:
$Ans(x, y) \leftarrow Bauteil(x, z),\ Bauteil(y, z)$

Auswertung der SQL-Anfrage in Datenbank mit

$\mathbf{I}_F(\boldsymbol{Bauteil})$

Teil	*Lager*
Motor	Seattle
Motor	Seattle
Flügel	Portland
Cockpit	Seattle
Cockpit	Seattle
Cockpit	Seattle

- bilde Kreuzprodukt $\mathbf{I}_F(\mathit{Bauteil}) \times \mathbf{I}_F(\mathit{Bauteil})$ (ohne Duplikatelimination)
- wähle die Tupel, in denen die Lager-Komponenten gleich sind
- streiche die Spalten mit den Lager-Komponenten

Liefert als Ergebnis die Multimenge M mit

- $|\,(\text{Motor,Motor})\,|_M = 4 \quad |\,(\text{Flügel,Flügel})\,|_M = 1$
  $|\,(\text{Cockpit,Cockpit})\,|_M = 9$
- $|\,(\text{Motor,Cockpit})\,|_M = 6 = |\,(\text{Cockpit,Motor})\,|_M$

Folie 128

Konjunktive Anfragen mit Multimengen-Semantik

Multimengen-Semantik $[\![Q]\!]_b$***:***
Sei $Q := \quad \mathit{Ans}(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ eine regelbasierte konjunktive Anfrage der Stelligkeit r über einem Datenbankschema $\mathbf{S}$.

- Eine *Belegung* β *für* Q ist, wie bisher, eine Abbildung $\beta : \text{var}(Q) \to \mathbf{dom}$.
- Die Auswertung von Q in einer Multimengen-Datenbank $\mathbf{I} \in \mathit{inst}_b(\mathbf{S})$ liefert als Ergebnis die Multimenge $[\![Q]\!]_b(\mathbf{I})$, so dass für alle Tupel $t \in \mathbf{dom}^r$ gilt:

$$|\,t\,|_{[\![Q]\!]_b(\mathbf{I})} \quad := \quad \sum_{\substack{\beta\ :\ \beta \text{ Belegung} \\ \text{für } Q \text{ mit } \beta(u)=t}} \Big(\ |\beta(u_1)|_{\mathbf{I}(R_1)} \cdot |\beta(u_2)|_{\mathbf{I}(R_2)} \cdots |\beta(u_\ell)|_{\mathbf{I}(R_\ell)} \ \Big)$$

Äquivalenz und Query Containment von Anfragen:
Seien Q und Q' zwei regelbasierte konjunktive Anfragen derselben Stelligkeit r über einem Datenbankschema $\mathbf{S}$.

- $Q \equiv_b Q' \quad :\Longleftrightarrow \quad [\![Q]\!]_b(\mathbf{I}) =_b [\![Q']\!]_b(\mathbf{I}),\;$ für alle $\mathbf{I} \in \mathit{inst}_b(\mathbf{S})$
- $Q \sqsubseteq_b Q' \quad :\Longleftrightarrow \quad [\![Q]\!]_b(\mathbf{I}) \subseteq_b [\![Q']\!]_b(\mathbf{I}),\;$ für alle $\mathbf{I} \in \mathit{inst}_b(\mathbf{S})$
- Klar: $\quad Q \equiv_b Q' \iff \Big(Q \sqsubseteq_b Q' \text{ und } Q' \sqsubseteq_b Q\Big)$

  Insbes: Äquivalenz ist höchstens so schwer wie Query Containment.

Folie 129

Ergebnisse

Theorem 3.50 (Chaudhuri, Vardi, 1993 (hier ohne Beweis))**.**

(a) Seien Q und Q' regelbasierte konjunktive Anfragen derselben Stelligkeit über demselben Datenbankschema. Dann ist $Q \equiv_b Q'$ genau dann, wenn die zu Q und Q' gehörenden Tableau-Anfragen isomorph sind (im Sinne von Definition 3.37).

(b) Das Problem

ÄQUIVALENZ KONJ. ANFRAGEN BZGL. MULTIMENGEN-SEMANTIK
Eingabe: regelbasierte konjunktive Anfragen Q und Q'
Frage: Ist $Q \equiv_b Q'$?

ist genauso schwer wie das Graph-Isomorphie-Problem.

Folie 130

Folgerungen und eine offene Frage

- Das *Äquivalenzproblem bzgl. Multimengen-Semantik* liegt in NP und ist vermutlich nicht NP-vollständig (da vermutet wird, dass das Graph-Isomorphie-Problem nicht NP-hart ist).

  Somit ist Äquivalenz bzgl. Multimengen-Semantik vermutlich <u>einfacher</u> als Äquivalenz bzgl. Mengen-Semantik.
- Das *Query Containment Problem bzgl. Multimengen-Semantik* ist vermutlich <u>schwerer</u> als das Query Containment Problem bzgl. Mengen-Semantik.

Offene Forschungsfrage:
Bisher ist nicht bekannt, ob das Query Containment Problem für konjunktive Anfragen bzgl. Multimengen-Semantik überhaupt entscheidbar ist.

Folie 131

Konj. Anfragen mit $\neq$ in Multimengen-Semantik

Konjunktive regelbasierte Anfragen mit $\neq$ sind von der Form

$$Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell), U_1, \ldots, U_m$$

wobei $R_1(u_1), \ldots, R_\ell(u_\ell)$ Relations-Atome und $U_1, \ldots, U_m$ Ungleichungen der Form $x \neq y$ mit $x, y \in \mathbf{var} \cup \mathbf{dom}$ sind.

Theorem 3.51 (Jayram, Kolaitis, Vee, 2006 (hier ohne Beweis))**.**
Das Problem

QCP für konj. Anfragen mit $\neq$ bzgl. Multimengen-Semantik
Eingabe: Q und Q' : regelbasierte konjunktive Anfragen mit $\neq$
Frage: Ist $Q \sqsubseteq_b Q'$?

ist <u>*nicht*</u> *entscheidbar.*

Folie 132

Ab jetzt wieder

— und bis zum Ende des Semesters —

Mengen-Semantik

Kapitel 4

Datalog

4.1 Syntax, Semantik und Auswertungskomplexität

Folie 133

Beispiel: Frankfurter U-Bahn-Netz

Hier vereinfacht: Eine Relation *U-Bahn-Netz* mit Attributen *Linie*, *Halt*, *nächsterHalt*

U-Bahn-Netz

Linie	*Halt*	*nächsterHalt*
U4	Bockenheimer Warte	Festhalle/Messe
U4	Festhalle/Messe	Hauptbahnhof
U4	Hauptbahnhof	Willy-Brandt-Platz
U4	Willy-Brandt-Platz	Dom/Römer
...	...	...
U7	...	...
U7	Kirchplatz	Leipziger Str.
U7	Leipziger Str.	Bockenheimer Warte
U7	Bockenheimer Warte	Westend
...	...	...

Anfrage:
Gib alle Stationen aus, die von "Bockenheimer Warte" aus ohne Umsteigen zu erreichen sind.

(1) mit max. 1 Zwischenhalt:

$$\Big\{ (x_S) : \exists x_L \Big(\textit{U-Bahn-Netz}(x_L, \text{"Bockenheimer Warte"}, x_S) \vee$$

$$\exists x_Z \Big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_Z) \land \textit{U-Bahn-Netz}(x_L, x_Z, x_S) \Big) \Big) \Big\}$$

(2) mit max. 2 Zwischenhalten: *analog*

(3) mit *beliebig vielen* Zwischenhalten *???*
(In einem späteren Kapitel werden wir sehen: *Nicht ausdrückbar in Relationaler Algebra*)

Folie 134

Erreichbarkeits-Anfrage in SQL

Im SQL (ab SQL-99 Standard) kann die Anfrage *“Gib alle Stationen aus, die von “Bockenheimer Warte” aus ohne Umsteigen zu erreichen sind”* folgendermaßen ausgedrückt werden:

```
WITH RECURSIVE Erreichbar(Linie,Start,Ziel)
AS (
    SELECT Linie, Halt, NaechsterHalt
    FROM U-Bahn-Netz
 UNION ALL
    SELECT Erreichbar.Linie,
         Erreichbar.Start,
         U-Bahn-Netz.NaechsterHalt
    FROM Erreichbar, U-Bahn-Netz
    WHERE Erreichbar.Linie = U-Bahn-Netz.Linie AND
        Erreichbar.Ziel = U-Bahn-Netz.Halt
)
SELECT Ziel
FROM Erreichbar
WHERE Start=’’Bockenheimer Warte’’
```

Folie 135

Erreichbarkeits-Anfrage in Datalog

Die Anfrage

"Gib alle Stationen aus, die von "Bockenheimer Warte" aus ohne Umsteigen zu erreichen sind"

kann in *Datalog* folgendermaßen ausgedrückt werden:

$$\begin{aligned}
\textit{Erreichbar}(L,S,Z) &\leftarrow \textit{U-Bahn-Netz}(L,S,Z)\\
\textit{Erreichbar}(L,S,Z) &\leftarrow \textit{Erreichbar}(L,S,Z'),\ \textit{U-Bahn-Netz}(L,Z',Z)\\
\textit{Ans}(Z) &\leftarrow \textit{Erreichbar}(L,\text{“Bockenheimer Warte”},Z)
\end{aligned}$$

Folie 136

Datalog: Syntax

Definition 4.1.

(a) Eine *Datalog-Regel* ist ein Ausdruck der Form

$$R_0(u_0) \;\leftarrow\; R_1(u_1),\ldots,R_\ell(u_\ell)$$

wobei $\ell \geqslant 0$, $R_0,R_1,\ldots,R_\ell \in \mathbf{rel}$ und $u_0,u_1,\ldots,u_\ell$ freie Tupel der Stelligkeiten $\mathrm{ar}(R_0),\mathrm{ar}(R_1),\ldots,\mathrm{ar}(R_\ell)$ sind, so dass jede Variable, die in u_0 vorkommt, auch in mindestens einem der Tupel $u_1,\ldots,u_\ell$ vorkommt.

(b) Ein *Datalog-Programm* P ist eine *endliche Menge von Datalog-Regeln.*

(c) Eine *Datalog-Anfrage* (P,R) besteht aus einem Datalog-Programm P und einem Relationsnamen R, der in P vorkommt.

Folie 137

Notation

- Wie üblich bezeichnen wir mit $\mathrm{adom}(P)$ bzw. $\mathrm{adom}(Q)$ die Menge der Konstanten, die in einem Datalog-Programm P bzw. einer Datalog-Anfrage Q vorkommen.

  Für eine Datenbank $\mathbf{I}$ ist $\mathrm{adom}(P,\mathbf{I}) := \mathrm{adom}(P) \cup \mathrm{adom}(\mathbf{I})$ und $\mathrm{adom}(Q,\mathbf{I}) := \mathrm{adom}(Q) \cup \mathrm{adom}(\mathbf{I})$.

- $R_0(u_0)$ heißt *Kopf* der Regel $R_0(u_0) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$; $R_1(u_1), \ldots, R_\ell(u_\ell)$ heißt *Rumpf* der Regel.

- *edb*(P) bezeichnet die Menge der Relationsnamen, die ausschließlich im *Rumpf* von Regeln in P vorkommen (die sog. "extensionalen Prädikate von P").

- *idb*(P) bezeichnet die Menge der Relationsnamen, die im *Kopf* mindestens einer Regel in P vorkommen (die sog. "intensionalen Prädikate von P").

- $\mathit{sch}(P) \;:=\; \mathit{edb}(P) \,\dot{\cup}\, \mathit{idb}(P)$ heißt *Schema* von P.

Folie 138

Beispiel

$$P \;:=\; \left\{ \begin{array}{rcl} \mathit{Erreichbar}(L,S,Z) & \leftarrow & \mathit{U\text{-}Bahn\text{-}Netz}(L,S,Z)\ , \\ \mathit{Erreichbar}(L,S,Z) & \leftarrow & \mathit{Erreichbar}(L,S,Z'),\ \mathit{U\text{-}Bahn\text{-}Netz}(L,Z',Z)\ , \\ \mathit{Ans}(Z) & \leftarrow & \mathit{Erreichbar}(L,\text{``Bockenheimer Warte''},Z) \end{array} \right\}$$

ist ein Datalog-Programm mit

- $\mathit{edb}(P) = \{\ \mathit{U\text{-}Bahn\text{-}Netz}\ \}$
- $\mathit{idb}(P) = \{\ \mathit{Erreichbar},\ \mathit{Ans}\ \}$
- $\mathit{sch}(P) = \{\ \mathit{U\text{-}Bahn\text{-}Netz},\ \mathit{Erreichbar},\ \mathit{Ans}\ \}$

$Q_1 := (P, \mathit{Ans})$ ist eine Datalog-Anfrage;
$Q_2 := (P, \mathit{Erreichbar})$ ist noch eine Datalog-Anfrage;
$Q_3 := (P, \mathit{U\text{-}Bahn\text{-}Netz})$ ist noch eine Datalog-Anfrage.

Folie 139

Datalog: Semantik

Die Semantik von Datalog lässt sich auf verschiedene Weisen definieren:

- *Fixpunkt-Semantik:*
  "Regeln schrittweise anwenden, bis sich nichts mehr ändert"

- *Modellbasierte Semantik:*
  kleinste Datenbank über $sch(P)$, die alle "Regeln wahr macht"

- *Beweisbasierte Semantik:*
  "Fakten, die sich herleiten lassen, sind im Ergebnis"

Glücklicherweise kann man zeigen, dass alle drei Ansätze zum gleichen Resultat führen.
Wir betrachten im Folgenden hauptsächlich die Fixpunkt-Semantik, die folgendermaßen definiert ist:

Folie 140

Der "immediate consequence"-Operator T_P

Definition 4.2. Sei P ein Datalog-Programm.

(a) Für jedes $R \in idb(P)$ seien $Q_{R,1}, \ldots, Q_{R,k_R}$ diejenigen Regeln aus P, in deren Kopf das Relationssymbol R steht, und seien $Q'_{R,1}, \ldots, Q'_{R,k_R}$ die Regeln, die aus $Q_{R,1}, \ldots, Q_{R,k_R}$ entstehen, indem im *Kopf* jeweils R durch das Relationssymbol Ans' ersetzt wird (mit $Ans' \notin sch(P)$ und $\mathrm{ar}(Ans') = \mathrm{ar}(R)$).

(b) Der *"immediate consequence"-Operator* $T_P : inst(sch(P)) \to inst(sch(P))$ ist folgendermaßen definiert:
Für jedes $\mathbf{J} \in inst(sch(P))$ und jedes $R \in sch(P)$ ist

$$T_P(\mathbf{J})(R) \;:=\; \begin{cases} \mathbf{J}(R) & \text{falls } R \in edb(P) \\ [\![Q'_{R,1}]\!](\mathbf{J}) \cup \cdots \cup [\![Q'_{R,k_r}]\!](\mathbf{J}) & \text{falls } R \in idb(P) \end{cases}$$

Beispiel: Ist P das Datalog-Programm aus dem Beispiel der Erreichbarkeits-Anfrage, und besteht $\mathbf{J}(Erreichbar)$ aus allen Tupeln, die "mit max. i Zwischenhalten ohne Umsteigen zu erreichen sind", so besteht $T_P(\mathbf{J})(Erreichbar)$ aus allen Tupeln, die "mit max. $i+1$ Zwischenhalten ohne Umsteigen zu erreichen sind".

Folie 141

Monotonie von T_P

Bemerkung: Eine alternative (und äquivalente) Definition von T_P:

$$T_P(\mathbf{J})(R) \ := \{ \text{ unmittelbare } R\text{-Konsequenzen von } P \text{ über } \mathbf{J} \ \},$$

wobei ein Tupel t eine *unmittelbare R-Konsequenz von P über* $\mathbf{J}$ ist, falls

- $R \in \mathit{edb}(P)$ und $t \in \mathbf{J}(R)$ oder
- $R \in \mathit{idb}(P)$ und es eine Regel der Form $R(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ in P und eine Belegung β gibt, so dass $\beta(u) = t$ und $\beta(u_i) \in \mathbf{J}(R_i)$, für alle $i \in \{1, \ldots, \ell\}$.

Lemma 4.3. *Für jedes Datalog-Programm P gilt:*
Der Operator T_P ist monoton, d.h. für alle $\mathbf{J}, \mathbf{J}' \in \mathit{inst}(\mathit{sch}(P))$ mit $\mathbf{J} \subseteq \mathbf{J}'$ (d.h. $\mathbf{J}(R) \subseteq \mathbf{J}'(R)$ f.a. $R \in \mathit{sch}(P)$) gilt: $T_P(\mathbf{J}) \subseteq T_P(\mathbf{J}')$.

Beweis: leicht (*Übung*).

Folie 142

Der "Stufen"-Operator S_P

Definition 4.4. Sei P ein Datalog-Programm.
Der *Stufen-Operator* $S_P : \mathit{inst}(\mathit{sch}(P)) \to \mathit{inst}(\mathit{sch}(P))$ ist folgendermaßen definiert:
Für jedes $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$ und jedes $R \in \mathit{sch}(P)$ ist

$$S_P(\mathbf{J})(R) \quad := \quad \mathbf{J}(R) \ \cup \ T_P(\mathbf{J})(R).$$

Folie 143

Fixpunkte

Bemerkung 4.5.

(a) Man sieht leicht, dass für jedes $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$ gilt:

$$\mathbf{J} \ \subseteq \ S_P(\mathbf{J}) \ \subseteq \ S_P\big(S_P(\mathbf{J})\big) \ \subseteq \ \cdots \ \subseteq \ S_P^i(\mathbf{J}) \ \subseteq \ S_P^{i+1}(\mathbf{J}) \ \subseteq \ \cdots$$

wobei $S_P^0(\mathbf{J}) := \mathbf{J}$ und $S_P^{i+1}(\mathbf{J}) := S_P\big(S_P^i(\mathbf{J})\big)$.

(b) Man sieht leicht, dass $\mathrm{adom}(S_P(\mathbf{J})) \subseteq \mathrm{adom}(P, \mathbf{J})$ und $\mathrm{adom}(S_P^i(\mathbf{J})) \subseteq \mathrm{adom}(P, \mathbf{J})$, f.a. $i \in \mathbb{N}$.

(c) Da $\mathrm{adom}(P, \mathbf{J})$ nur endlich viele Elemente besitzt, muss es in der Inklusionskette aus *(a)* ein $i_0 \in \mathbb{N}$ mit $S_P^{i_0}(\mathbf{J}) = S_P^{i_0+1}(\mathbf{J})$ geben. Offensichtlicherweise gilt dann: $S_P^{i_0}(\mathbf{J}) = S_P^{j}(\mathbf{J})$ f.a. $j \geqslant i_0$.

Folie 144

Notation

- Ein $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$ heißt *Fixpunkt von S_P*, falls $S_P(\mathbf{J}) = \mathbf{J}$.

- $\textit{Ab-Stufe}(P, \mathbf{J}) := \min\{i_0 \ : \ S_P^{i_0}(\mathbf{J}) = S_P^{i_0+1}(\mathbf{J})\}$ heißt *"Abschluss-Stufe"* von P auf $\mathbf{J}$.

  Die einzelnen Instanzen $S_P^0(\mathbf{J})$, $S_P^1(\mathbf{J})$, $S_P^2(\mathbf{J})$ etc. werden *"Stufen des Fixpunktprozesses"* genannt.

  $\textit{Ab-Stufe}(P, \mathbf{J})$ gibt also diejenige Stufe an, ab der der Fixpunkt erreicht ist:

$$\mathbf{J} = S_P^0(\mathbf{J}) \subsetneqq S_P^1(\mathbf{J}) \subsetneqq \cdots \subsetneqq S_P^{\textit{Ab-Stufe}(P,\mathbf{J})}(\mathbf{J}) = S_P^{\textit{Ab-Stufe}(P,\mathbf{J})+1}(\mathbf{J}) = S_P^j(\mathbf{J}) \quad \text{f.a. } j \geqslant \textit{Ab-Stufe}(P, \mathbf{J})$$

- $S_P^\omega(\mathbf{J}) := S_P^{\textit{Ab-Stufe}(P,\mathbf{J})}(\mathbf{J})$. Klar: $S_P^\omega(\mathbf{J})$ ist ein Fixpunkt von S_P.

Folie 145

Fixpunkt-Semantik von Datalog

Definition 4.6.

(a) Sei P ein Datalog-Programm, $\mathbf{S} := \mathit{edb}(P)$.
Jeder Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ ordnen wir die Datenbank $\hat{\mathbf{I}} \in \mathit{inst}(\mathit{sch}(P))$ mit

$$\hat{\mathbf{I}}(R) \ := \ \begin{cases} \mathbf{I}(R) & \text{falls } R \in \mathit{edb}(P) \\ \emptyset & \text{falls } R \in \mathit{idb}(P) \end{cases}$$

zu. Ausgewertet in $\mathbf{I}$ liefert P die Datenbank

$$[\![P]\!](\mathbf{I}) \ := \ S_P^\omega(\hat{\mathbf{I}}) \ \in \ \mathit{inst}(\mathit{sch}(P))$$

(b) Eine Datalog-Anfrage $Q = (P, R)$ liefert auf einer Datenbank $\mathbf{I} \in \mathit{inst}(\mathit{edb}(P))$ die Relation

$$[\![Q]\!](\mathbf{I}) \ := \ \big([\![P]\!](\mathbf{I})\big)(R).$$

Folie 146

Auswertungskomplexität

Bemerkung:
Ein einfacher Algorithmus, der bei Eingabe von P und $\mathbf{I}$ das Resultat $[\![P]\!](\mathbf{I})$ berechnet, kann folgendermaßen vorgehen:

(1) $\mathbf{J} := \hat{\mathbf{I}}$
(2) Berechne $\mathbf{J}' := S_P(\mathbf{J})$
(3) Falls $\mathbf{J}' \neq \mathbf{J}$, so $\big(\ \mathbf{J} := \mathbf{J}',\ \text{GOTO } \mathit{(2)}\ \big)$
(4) Gib $\mathbf{J}$ aus

Datenkomplexität dieses Algorithmus: Polynomialzeit.

Folie 147

Fixpunkt-Semantik vs. Modellbasierte Semantik

Notation:
Für zwei Datenbanken $\mathbf{J}, \mathbf{J}' \in \mathit{inst}(\mathbf{S})$ bezeichnet $\mathbf{J} \cap \mathbf{J}'$ die Datenbank mit $(\mathbf{J} \cap \mathbf{J}')(R) := \mathbf{J}(R) \cap \mathbf{J}'(R)$, f.a. $R \in \mathbf{S}$.
Für eine Menge $M \subseteq \mathit{inst}(\mathbf{S})$ ist $\bigcap M := \bigcap_{\mathbf{J} \in M} \mathbf{J}$.

Satz 4.7 (Knaster und Tarski)**.**
Sei P ein Datalog-Programm und sei $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$. Dann gilt:

$$S_P^\omega(\mathbf{J}) \ = \ \bigcap \big\{\ \mathbf{J}' \in \mathit{inst}(\mathit{sch}(P)) \ : \ S_P(\mathbf{J}') = \mathbf{J}' \text{ und } \mathbf{J} \subseteq \mathbf{J}'\ \big\}.$$

D.h.: $S_P^\omega(\mathbf{J})$ ist die kleinste Erweiterung von $\mathbf{J}$, die ein Fixpunkt von S_P ist.

Beweis: Siehe Tafel.

Folie 148

Beweisbasierte Semantik von Datalog

Sichtweise:

- Ein *Faktum* ist ein Ausdruck der Form $R(t)$ mit $R \in \mathbf{rel}$ und $t \in \mathbf{dom}^{\mathrm{ar}(R)}$.

- Eine Datenbank $\mathbf{I} \in inst(\mathbf{S})$ wird mit der folgenden *Menge von Fakten* identifiziert:

$$Fakten(\mathbf{I}) \; := \; \{ \; R(t) \quad : \quad R \in \mathbf{S} \; und \; t \in \mathbf{I}(R) \; \}$$

- Für die beweisbasierte Semantik werden Datalog-Regeln als Schlussregel-Muster in Beweisen betrachtet.

Folie 149

Beweisbäume

Definition 4.8. Sei P ein Datalog-Programm und $\mathbf{J} \in inst(sch(P))$. Ein *Beweisbaum für ein Faktum $R(t)$ bzgl. $\mathbf{J}$ und P* ist ein gerichteter Baum mit den folgenden Eigenschaften:

(1) Jeder Knoten ist mit einem Faktum markiert.

(2) Die Wurzel enthält das Faktum $R(t)$.

(3) Die Fakten an den Blättern sind Elemente aus $Fakten(\mathbf{J})$

(4) Für jeden inneren Knoten v mit Kindern $v_1, \ldots, v_\ell$ gibt es eine Regel $R_0(u_0) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ in P und eine Belegung $\beta : \mathrm{var}(P) \to \mathbf{dom}$ so dass gilt:

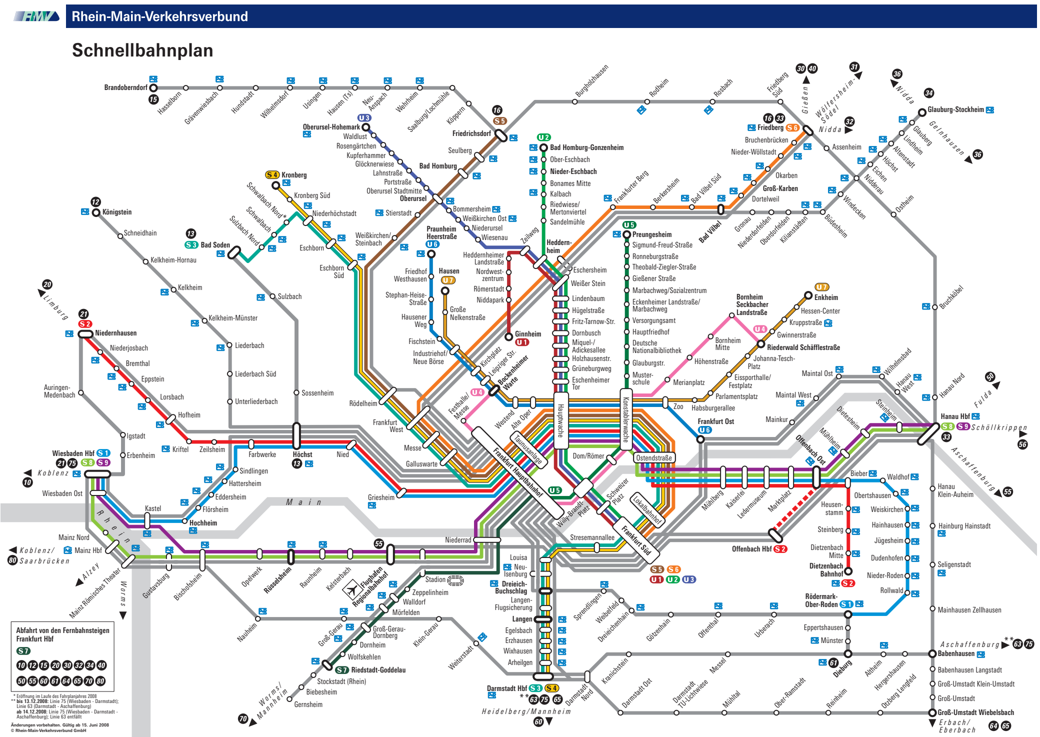
- der Knoten v enthält das Faktum $R_0(\beta(u_0))$
- das Kind v_i von v enthält das Faktum $R_i(\beta(u_i))$ (für alle $i \in \{1, \ldots, \ell\}$).

Beispiel: Beweisbäume für *Ans*(*"Frankfurt Süd"*) bzgl. dem Frankfurter U-Bahn-Netz und dem folgenden Datalog-Programm:

$$\begin{array}{rcl} E(L,S,Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\ E(L,S,Z) & \leftarrow & E(L,S,Y),\ \textit{U-Bahn-Netz}(L,Y,Z) \\ \textit{Ans}(Z) & \leftarrow & E(L,\text{“Hauptwache”},Z) \end{array}$$

(siehe Tafel)

Folie 150



Folie 151

Fixpunkt-Semantik vs. Beweisbasierte Semantik

Satz 4.9. *Für jedes Datalog-Programm P, alle $\mathbf{J} \in inst(sch(P))$ und alle Fakten $R(t)$ mit $R \in sch(P)$ und $t \in \mathbf{dom}^{\mathrm{ar}(R)}$ gilt:*

$$t \in S_P^{\omega}(\mathbf{J})(R) \iff \text{es gibt einen Beweisbaum für } R(t) \text{ bzgl. } \mathbf{J} \text{ und } P.$$

Beweis: Übung.

4.2 Grenzen der Ausdrucksstärke von Datalog

Folie 152

Monotonie und Abschluss unter $\mathrm{adom}(Q)$-Homomorphismen

Satz 4.10. *Für jede Datalog-Anfrage Q gilt: $[\![Q]\!]$ ist monoton.*

Beweis: Übung (folgt leicht aus der Monotonie des T_P-Operators).

Auf ähnliche Art lässt sich zeigen:

Satz 4.11. *Für jede Datalog-Anfrage Q gilt:*
$[\![Q]\!]$ *ist abgeschlossen unter* $\text{adom}(Q)$*-Homomorphismen.*

Zur Erinnerung: Auf Übungsblatt 1 wurde Folgendes definiert.

- Ein *C-Homomorphismus* (für $C \subseteq \mathbf{dom}$) ist eine Abbildung $h : \mathbf{dom} \to \mathbf{dom}$ mit $h_{|C} = \text{id}$.
- Eine Anfragefunktion q ist *abgeschlossen unter C-Homomorphismen*, falls für alle C-Homomorphismen h und alle Datenbanken $\mathbf{I}$ und $\mathbf{J}$ gilt:

$$\text{Falls } \; h(\mathbf{I}) \subseteq \mathbf{J}, \; \text{ so ist } \; h(q(\mathbf{I})) \subseteq q(\mathbf{J}).$$

Folie 153

Ausdrucksstärke von Datalog

Bemerkung 4.12. Die Ausdrucksstärken von Datalog-Anfragen und von Anfragen des Relationalen Algebra sind unvergleichbar:

- Beispiel für eine Datalog-Anfrage, die nicht in Relationaler Algebra formuliert werden kann:
  "Welche Stationen sind von "Bockenheimer Warte" aus ohne Umsteigen zu erreichen?"

- Beispiel für eine Relationale Algebra-Anfrage, die nicht in Datalog formuliert werden kann:
  "In welchen Kinos läuft kein Film um 17:00 Uhr?"

  Bzw. jede andere Anfrage, die nicht monoton ist (aber in Relationaler Algebra formuliert werden kann).

  Aus der Monotonie von Datalog-Anfragen folgt leicht, dass diese Anfrage nicht in Datalog ausgedrückt werden kann.

4.3 Datalog zur Simulation von Turingmaschinen

Folie 154

Repräsentation von Worten als Datenbanken

Wir repräsentieren Worte über einem Alphabet Σ durch Datenbanken über dem Schema

$$\mathbf{S}_\Sigma \ := \ \{\text{Succ}, \text{Min}, \text{Max}\} \cup \{P_a \ : \ a \in \Sigma\},$$

wobei Succ 2-stellig ist und alle anderen Relationsnamen 1-stellig sind.

Für ein Wort $w = w_0 \cdots w_{n-1} \in \Sigma^*$ mit $|w| = n \geqslant 1$ und $w_0, \dots, w_{n-1} \in \Sigma$ ist für jedes $a \in \Sigma$

$$\mathbf{I}_w(P_a) \ := \ \{p \in \{0, \dots, n{-}1\} \ : \ w_p = a\}$$

und

$$\begin{aligned} \mathbf{I}_w(\text{Min}) &:= \{0\} \\ \mathbf{I}_w(\text{Max}) &:= \{n{-}1\} \\ \mathbf{I}_w(\text{Succ}) &:= \{(p, p{+}1) \ : \ 0 \leqslant p < n{-}1\}. \end{aligned}$$

Folie 155

Datalog zur Simulation von Turingmaschinen

Satz 4.13. *Für jede deterministische Turingmaschine M mit Eingabealphabet Σ und jede Zahl $k \geqslant 1$ gibt es ein Datalog-Programm $P_{M,k}$ mit $edb(P_{M,k}) = \mathbf{S}_\Sigma$ und ein 0-stelliges idb-Prädikat Ans von $P_{M,k}$, so dass für die Anfrage $Q_{M,k} := (P_{M,k}, Ans)$ und für jedes Wort $w \in \Sigma^+$ gilt:*

$$[\![Q_{M,k}]\!](\mathbf{I}_w) = \text{“yes”} \iff \text{bei Eingabe } w \text{ hält } M \text{ nach höchstens } |w|^k{-}1 \text{ Schritten in einem akzeptierenden Zustand an.}$$

Beweis: Siehe Tafel.

Folie 156

Die Auswertungskomplexität von Datalog-Anfragen

Theorem 4.14 (Immerman und Vardi)**.**

(a) Die kombinierte Komplexität des Auswertungsproblems für Datalog-Anfragen ist EXPTIME-vollständig.

(b) Die Datenkomplexität des Auswertungsproblems für Datalog-Anfragen ist PTIME-vollständig.

Beweis: Übung (unter Verwendung von ähnlichen Methoden wie beim Beweis von Satz 4.13).

Mit Aussage (b) ist Folgendes gemeint:

(1) Für jede Datalog-Anfrage $Q = (P, R)$ ist das Problem

> AWP_Q
> *Eingabe:* Datenbank $\mathbf{I} \in \mathit{inst}(\mathit{edb}(P))$
> *Aufgabe:* Berechne $[\![Q]\!](\mathbf{I})$

in Zeit polynomiell in der Größe von $\mathbf{I}$ lösbar, und

(2) es gibt eine Boolesche Datalog-Anfrage Q, für die das Problem AWP_Q PTIME-vollständig ist (bzgl. logspace-Reduktionen).

4.4 Statische Analyse

Folie 157

Erfüllbarkeit

Theorem 4.15. *Das*

> ERFÜLLBARKEITSPROBLEM FÜR DATALOG-ANFRAGEN
> *Eingabe:* Datalog-Anfrage $Q = (P, R)$
> *Frage:* Gibt es ein $\mathbf{I} \in \mathit{inst}(\mathit{edb}(P))$ so dass $[\![Q]\!](\mathbf{I}) \neq \emptyset$?

ist entscheidbar.

Beweisidee:

Verallgemeinerung des Beweises der Erfüllbarkeit regelbasierter konjunktiver Anfragen (Satz 3.6).

Details: Übung.

Folie 158

Query Containment — Zwei Varianten

Herkömmliches *Query Containment:*

QUERY CONTAINMENT PROBLEM FÜR DATALOG-ANFRAGEN
Eingabe: Zwei Datalog-Anfragen $Q_1 = (P_1, R)$ und $Q_2 = (P_2, R)$ mit $edb(P_1) = edb(P_2)$ und $R \in idb(P_1) \cap idb(P_2)$
Frage: Gilt $[\![Q_1]\!](\mathbf{I}) \subseteq [\![Q_2]\!](\mathbf{I})$ für alle $\mathbf{I} \in inst(edb(P_1))$?

Uniformes Containment:

UNIFORMES CONTAINMENT PROBLEM FÜR DATALOG-PROGRAMME
Eingabe: Zwei Datalog-Programme P_1 und P_2 mit $edb(P_1) = edb(P_2)$ und $idb(P_1) = idb(P_2)$
Frage: Gilt $S^{\omega}_{P_1}(\mathbf{J}) \subseteq S^{\omega}_{P_2}(\mathbf{J})$ für alle $\mathbf{J} \in inst(sch(P_1))$?

Theorem 4.16.

(a) Das Query Containment Problem für Datalog-Anfragen ist unentscheidbar.

(b) Das uniforme Containment Problem für Datalog-Programme ist entscheidbar.

Beweis: Hier nur der Beweis von (a): siehe Tafel.

Folie 159

Beschränktheit (Boundedness)

- Wir haben gesehen: $[\![P]\!](\mathbf{I})$ kann berechnet werden, indem man nach und nach

$$\hat{\mathbf{I}},\ S_P(\hat{\mathbf{I}}),\ S_P(S_P(\hat{\mathbf{I}})),\ \ldots,\ S_P^{\textit{Ab-Stufe}(P,\mathbf{I})}(\hat{\mathbf{I}}),\ S_P^{\textit{Ab-Stufe}(P,\mathbf{I})+1}(\hat{\mathbf{I}})$$

berechnet.

- Anzahl der Iterationen, die dafür nötig sind: $\textit{Ab-Stufe}(P, \mathbf{I}) + 1$

- *Frage: Wie groß kann Ab-Stufe*$(P, \mathbf{I})$ *werden?*

- *Klar:* – in jeder Iteration kommt mindestens ein Faktum dazu
  – $\mathrm{adom}(\llbracket P \rrbracket(\mathbf{I})) \subseteq \mathrm{adom}(P, \mathbf{I})$

- *Daher:* *Ab-Stufe*$(P, \mathbf{I}) \leqslant \sum_{R \in idb(P)} |\,\mathrm{adom}(P, \mathbf{I})|^{\mathrm{ar}(R)}$

- Besonders "schön" sind solche Datalog-Programme P, bei denen *Ab-Stufe*$(P, \mathbf{I})$ gar nicht von $\mathbf{I}$ abhängt. Solche Programme heißen *beschränkt* (engl.: *bounded*).

- *Präzise:* Ein Datalog-Programm P heißt *beschränkt*, falls eine Zahl $d \in \mathbb{N}$ gibt, so dass für alle $\mathbf{I} \in inst(edb(P))$ gilt: *Ab-Stufe*$(P, \mathbf{I}) \leqslant d$.
  Den kleinsten solchen Wert d nennen wir *maximale Rekursionstiefe von* P.

- Wenn man weiß, dass ein Programm P bschränkt ist und maximale Rekursionstiefe d hat, so kann man leicht eine zu P äquivalente Anfrage in relationaler Algebra konstruieren, die nur die Operatoren der SPC-Algebra und den Vereinigungs-Operator benutzt (kurz: SPCU-Algebra).

Folie 160

Beispiel

Das Datalog-Programm

$$\begin{array}{rcl} E(L,S,Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\ E(L,S,Z) & \leftarrow & E(L,S,Y),\ \textit{U-Bahn-Netz}(L,Y,Z) \\ \textit{Ans}(Z) & \leftarrow & E(L, \text{“}\textit{Bockenheimer Warte}\text{”}, Z) \end{array}$$

ist nicht beschränkt.

Das Datalog-Programm

$$\begin{array}{rcl} \textit{Kauft}(x,y) & \leftarrow & \textit{Mag}(x,y) \\ \textit{Kauft}(x,y) & \leftarrow & \textit{Person}(x),\ \textit{Trendsetter}(z),\ \textit{Kauft}(z,y) \end{array}$$

ist beschränkt mit maximaler Rekursionstiefe 3, da für jede DB $\mathbf{I}$ gilt:
$T_P^{\omega}(\hat{\mathbf{I}})(\textit{Kauft}) = \mathbf{I}(\textit{Mag}) \cup \{(x,y) \ : \ x \in \mathbf{I}(\textit{Person}), \ \ \text{ex. } z \in$

$\mathbf{I}(\mathit{Trendsetter})$ s.d. $(z,y) \in \mathbf{I}(\mathit{Mag})\}$ und äquivalent zum nicht-rekursiven Programm

$$\begin{array}{rcl} \mathit{Kauft}'(x,y) & \leftarrow & \mathit{Mag}(x,y) \\ \mathit{Kauft}'(x,y) & \leftarrow & \mathit{Person}(x),\ \mathit{Trendsetter}(z),\ \mathit{Mag}(z,y) \\ \mathit{Kauft}(x,y) & \leftarrow & \mathit{Kauft}'(x,y) \\ \mathit{Kauft}(x,y) & \leftarrow & \mathit{Person}(x),\ \mathit{Trendsetter}(z),\ \mathit{Kauft}'(z,y) \end{array}$$

Folie 161

Beschränktheit (Boundedness)

Theorem 4.17. *Das*

Boundedness Problem für Datalog-Programme
Eingabe: Datalog-Programm P
Frage: Ist P beschränkt, d.h. gibt es ein $d \in \mathbb{N}$ so dass $\mathit{Ab\text{-}Stufe}(P,\mathbf{I}) \leqslant d$ für alle $\mathbf{I} \in \mathit{inst}(\mathit{edb}(P))$?

ist unentscheidbar.

Beweis: siehe Tafel.

4.5 Einschränkung und Erweiterungen: nr-Datalog und Datalog mit Negation

Folie 162

Nicht-Rekursives Datalog — Beispiel

Beispiel 4.18. (a) Das Datalog-Programm

$$\begin{array}{rcl} \mathit{Kauft}(x,y) & \leftarrow & \mathit{Mag}(x,y) \\ \mathit{Kauft}(x,y) & \leftarrow & \mathit{Person}(x),\ \mathit{Trendsetter}(z),\ \mathit{Mag}(z,y) \end{array}$$

ist nicht-rekursiv.

(b) Äquivalent dazu, aber NICHT nicht-rekursiv ist

$$\begin{array}{rcl} \mathit{Kauft}(x,y) & \leftarrow & \mathit{Mag}(x,y) \\ \mathit{Kauft}(x,y) & \leftarrow & \mathit{Person}(x),\ \mathit{Trendsetter}(z),\ \mathit{Kauft}(z,y) \end{array}$$

Folie 163

Nicht-Rekursives Datalog — Präzise

Definition 4.19. Sei P ein Datalog-Programm.

(a) Der *Abhängigkeitsgraph* G_P von P ist der gerichtete Graph mit Knotenmenge $V_P := sch(P)$ und Kantenmenge

$$E_P \;:=\; \left\{ (R,S) \;:\; \begin{array}{l} \text{es gibt eine Regel in } P\text{, in deren Rumpf} \\ R \text{ und in deren Kopf } S \text{ vorkommt} \end{array} \right\}$$

(b) Die Klasse *nr-Datalog* aller *nicht-rekursiven Datalog-Programme* besteht aus allen Datalog-Programmen P, deren *Abhängigkeitsgraph* G_P *azyklisch* ist, d.h. keinen einfachen Kreis enthält.

Beispiele für Abhängigkeitsgraphen: *siehe Tafel*

Folie 164

"Nicht-rekursiv" vs. "beschränkt" ("bounded")

Proposition 4.20. *(a) Jedes nr-Datalog-Programm ist beschränkt.*

(b) Jedes beschränkte Datalog-Programm ist äquivalent zu einem nr-Datalog-Programm.

Beweis: Einfache Übung.

Folie 165

Ausdrucksstärke von nr-Datalog

- Die *SPCU-Algebra* ist die Erweiterung der SPC-Algebra um den *Vereinigungsoperator* $\cup$, der es erlaubt, die Ergebnisse zweier Anfragen derselben Stelligkeit zu vereinigen. Semantik: $[\![(Q_1 \cup Q_2)]\!](\mathbf{I}) := [\![Q_1]\!](\mathbf{I}) \cup [\![Q_2]\!](\mathbf{I})$.

- Der *positive existentielle Kalkül PE-CALC$_{adom}$* ist die Klasse aller Anfragen Q der Form $\{(e_1,\ldots,e_r) : \varphi\}$, wobei φ eine Formel der Logik erster Stufe ist, in der keins der Symbole $\neg, \forall, \rightarrow, \leftrightarrow$ vorkommt. Semantik: $[\![Q]\!](\mathbf{I}) \;:=\; \{\,\beta((e_1,\ldots,e_r)) \;:\; \beta : \text{var} \to \text{adom}(Q,\mathbf{I}),\ \textit{so dass}\ \mathbf{I} \models \varphi[\beta]\}$

Satz 4.21. *Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen beschreiben:*

(a) nr-Datalog-Anfragen

(b) SPCU-Algebra

(c) positiver existentieller Kalkül PE-CALC$_{adom}$

Bemerkung: Die Übersetzung von nr-Datalog in eine der anderen Sprachen kann mehr als polynomiell viel Zeit beanspruchen, da nr-Datalog-Programme u.U. viel kürzer sind als äquivalente Anfragen der anderen Sprachen.

Folie 166

Beweis von Satz 4.21

(b) ⇒ (a): Induktion über den Aufbau von SPCU-Anfragen (leicht). Details: siehe Tafel.

(a) ⇒ (c): siehe Tafel.

(c) ⇒ (b): Bei gegebener Anfrage $Q = \{(e_1, \dots, e_r) \ : \ \varphi\}$ bringe die Formel φ zunächst in disjunktive Normalform, d.h. in eine Formel der Form $\varphi_1 \vee \cdots \vee \varphi_\ell$, wobei φ_i eine $\mathsf{CQ}^=$-Formel ist.

Dann gilt für jede Datenbank $\mathbf{I}$, dass $[\![Q]\!](\mathbf{I}) = [\![Q_1]\!](\mathbf{I}) \cup \cdots \cup [\![Q_\ell]\!](\mathbf{I})$, wobei

$$[\![Q_i]\!](\mathbf{I}) \quad = \quad \big\{ \ \beta((e_1, \dots, e_r)) \ : \ \beta : \text{var} \to \text{adom}(Q, \mathbf{I}), \ \text{so dass } \mathbf{I} \models \varphi_i \ \big\}.$$

Wir können nun ähnlich wie in Theorem 3.30 beim Beweis der Äquivalenz von Konjunkivem Kalkül und SPC-Algebra vorgehen, um eine SPCU-Anfrage Q_i' zu konstruieren, so dass für alle Datenbanken $\mathbf{I}$ gilt: $[\![Q_i']\!](\mathbf{I}) = [\![Q_i]\!](\mathbf{I})$.
Details: Übung. □

Folie 167

Datalog mit Negation

Ziel: Auch Negationszeichen "¬" in Datalog-Regeln zulassen.

Definition 4.22. (a) Ein *Literal* ist ein Relationsatom $R(u)$ oder ein negiertes Relationsatom $\neg R(u)$.
Ein Literal der Form $R(u)$ heißt *positiv*; ein Literal der Form $\neg R(u)$ heißt *negativ* bzw. *negiert*.

(b) Eine *Datalog*$^\neg$*-Regel* ist ein Ausdruck der Form

$$R_0(u_0) \ \leftarrow \ L_1(u_1), \ldots, L_\ell(u_\ell)$$

wobei $\ell \geqslant 0$, $R_0 \in \mathbf{rel}$, u_0 ein freies Tupel der Stelligkeit $\mathrm{ar}(R_0)$ und $L_1(u_1), \ldots, L_\ell(u_\ell)$ Literale, so dass jede Variable, die in u_0 vorkommt, auch in mindestens einem *positiven* Literal $L_i(u_i)$ vorkommt.

(c) Ein *Datalog*$^\neg$*-Programm* P ist eine *endliche Menge von Datalog*$^\neg$*-Regeln.*

(d) Eine *Datalog*$^\neg$*-Anfrage* (P, R) besteht aus einem Datalog$^\neg$-Programm P und einem Relationsnamen R, der in P vorkommt.

Folie 168

Frage: Was soll die Semantik von Datalog$^\neg$ sein?

Beispiel 4.23. *Anfrage:*
"Gib alle Stationen aus, die von "Bockenheimer Warte" aus <u>nicht</u> ohne Umsteigen zu erreichen sind."

Als Datalog$^\neg$*-Anfrage:*

$$\begin{array}{rcl}
E(L,S,Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
E(L,S,Z) & \leftarrow & E(L,S,Y),\ \textit{U-Bahn-Netz}(L,Y,Z) \\
\textit{Erreichbar_BW}(Z) & \leftarrow & E(L,\text{“Bockenheimer Warte”},Z) \\
\textit{Station}(S) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
\textit{Station}(Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
\textit{Ans}(Z) & \leftarrow & \textit{Station}(Z),\ \neg\textit{Erreichbar_BW}(Z)
\end{array}$$

Hier: Semantik intuitiv klar.

Folie 169

Probleme mit der Semantik von Datalog$^\neg$

Zur Erinnerung: In der Übung wurde gezeigt, dass für jedes Datalog-Programm P und jede Datenbank $I \in \mathit{inst}(\mathit{edb}(P))$ gilt: für jedes $i \in \mathbb{N}$ ist $S_p^i(\hat{\mathbf{I}}) = T_p^i(\hat{\mathbf{I}})$.
Also ist $[\![P]\!](\mathbf{I}) = S_P^\omega(\hat{\mathbf{I}}) = T_p^\omega(\hat{\mathbf{I}})$.

Für Datalog$^\neg$ gilt dies nicht:

Beispiel 4.24. (a) $R(x) \leftarrow A(x),\ \neg R(x)$

"Ausgewertet" über einer DB $\mathbf{I} \in \mathit{inst}(\{A\})$ gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \;=\; \mathbf{I}(A) \quad \text{für alle } i \geqslant 1$$

aber

$$T_P^i(\hat{\mathbf{I}})(R) \;=\; \begin{cases} \emptyset & \text{falls } i \text{ gerade} \\ \mathrm{adom}(\mathbf{I}) & \text{falls } i \text{ ungerade} \end{cases}$$

Somit: Die Folge $\left(T_P^i(\hat{\mathbf{I}})\right)_{i\geqslant 0}$ hat keinen Fixpunkt.
Außerdem: $T_P(\cdot)$ hat überhaupt keinen Fixpunkt.

Folie 170

(b) $R(x) \leftarrow A(x),\ \neg S(x)$
$S(x) \leftarrow A(x),\ \neg R(x)$

Hier gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \;=\; S_P^i(\hat{\mathbf{I}})(S) \;=\; \mathbf{I}(A) \quad \text{für alle } i \geqslant 1$$

aber $T_P(\cdot)$ hat zwei verschiedene minimale Fixpunkte *(minimal bzgl. $\subseteq$)*:

- FP_1 mit $\mathrm{FP}_1(R) = \emptyset$ und $\mathrm{FP}_1(S) = \mathrm{adom}(\mathbf{I})$
- FP_2 mit $\mathrm{FP}_2(R) = \mathrm{adom}(\mathbf{I})$ und $\mathrm{FP}_2(S) = \emptyset$

Folie 171

Datalog$^\neg$ und der Stufenoperator S_P

Um eine Semantik für Datalog$^\neg$ festzulegen, könnte man einfach wieder den Stufenoperator S_P betrachten. Natürlich gilt für jedes Datalog$^\neg$-Programm P und jede Datenbank $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$, dass

$$\mathbf{J} \ \subseteq \ S_P(\mathbf{J}) \ \subseteq \ S_P\big(S_P(\mathbf{J})\big) \ \subseteq \ \cdots \ \subseteq \ S_P^i(\mathbf{J}) \ \subseteq \ S_P^{i+1}(\mathbf{J}) \ \subseteq \ \cdots$$

Da adom$(P, \mathbf{J})$ endlich ist, wird irgendwann ein (eindeutig definierter) Fixpunkt von $S_P(\cdot)$ erreicht (dieser heißt übrigens *inflationärer Fixpunkt von P auf* $\mathbf{J}$, kurz: $S_P^\omega(\mathbf{J})$).

Wir könnten nun einfach festlegen, dass die Semantik von P auf $\mathbf{I} \in \mathit{inst}(\mathit{edb}(P))$ via $[\![P]\!](\mathbf{I}) = S_P^\omega(\hat{\mathbf{I}})$ definiert ist.

Folie 172

Problem mit der inflationären Fixpunkt-Semantik von Datalog$^\neg$

Diese über den Stufenoperator S_P definierte Semantik ist für viele Datalog$^\neg$-Programme *unnatürlich.*

Beispiel 4.25.

$$\begin{array}{rcl}
E(L,S,Z) & \leftarrow & \mathit{U\text{-}Bahn\text{-}Netz}(L,S,Z) \\
E(L,S,Z) & \leftarrow & E(L,S,Y),\ \mathit{U\text{-}Bahn\text{-}Netz}(L,Y,Z) \\
\mathit{Erreichbar_BW}(Z) & \leftarrow & E(L,\text{``}\mathit{Bockenheimer\ Warte}\text{''},Z) \\
\mathit{Station}(S) & \leftarrow & \mathit{U\text{-}Bahn\text{-}Netz}(L,S,Z) \\
\mathit{Station}(Z) & \leftarrow & \mathit{U\text{-}Bahn\text{-}Netz}(L,S,Z) \\
\mathit{Ans}(Z) & \leftarrow & \mathit{Station}(Z),\ \neg \mathit{Erreichbar_BW}(Z)
\end{array}$$

"Intuitive Semantik":
Alle Stationen, die von "Bockenheimer Warte" aus <u>nicht</u> ohne Umsteigen zu erreichen sind.

Inflationäre Fixpunkt-Semantik: <u>Alle</u> Stationen.

Folie 173

Frage: Was soll die Semantik von Datalog$^\neg$ sein?

Probleme:

- Inflationäre Fixpunkt-Semantik: unnatürlich (siehe Beispiel 4.25)
- Fixpunkt-Semantik via $T_P^\omega(\hat{\mathbf{I}})$:
  für manche Datalog$^\neg$-Programme undefiniert (siehe Beispiel 4.24)
- Semantik via "kleinster Fixpunkt" von $T_P(\cdot)$:
  ist für manche Datalog$^\neg$-Programme nicht eindeutig (siehe Beispiel 4.24)
- Beweisbasierte Semantik für Datalog$^\neg$: *???*

Aber für Programme wie in Beispiel 4.25 ist die Semantik "intuitiv klar".

$\rightsquigarrow$ Betrachte im Folgenden nur Datalog$^\neg$-Programme von eingeschränkter Form:

- Semipositives Datalog$^\neg$
- nr-Datalog$^\neg$
- Stratifiziertes Datalog$^\neg$

Folie 174

Semipositives Datalog$^\neg$

Negation ist *nur bei edb-Prädikaten* erlaubt.

Man kann leicht zeigen, dass für jedes semipositive Datalog$^\neg$-Programm P und
alle $\mathbf{I} \in \mathit{inst}(\mathit{edb}(P))$ gilt:

- $T_P(\cdot)$ hat einen eindeutig bestimmten kleinsten Fixpunkt $\mathbf{J}$ mit $\mathbf{J}|_{\mathit{edb}(P)} = \mathbf{I}$.
- Dieser wird von der Sequenz
  $$\hat{\mathbf{I}},\quad T_P(\hat{\mathbf{I}}),\quad T_P^2(\hat{\mathbf{I}}),\quad T_P^3(\hat{\mathbf{I}}),\ \ldots$$
  erreicht. Notation für diesen Fixpunkt: $T_P^\omega(\hat{\mathbf{I}})$.

Definition der Semantik von semipositiven Datalog$^\neg$-Programmen P:
Für alle $\mathbf{I} \in \mathit{inst}(\mathit{edb}(P))$ setze

$$[\![P]\!](\mathbf{I}) \ := \ T_P^\omega(\hat{\mathbf{I}})$$

Folie 175

Stratifiziertes Datalog$^\neg$ — Beispiel

$$\begin{array}{rcl}
E(L,S,Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
E(L,S,Z) & \leftarrow & E(L,S,Y),\ \textit{U-Bahn-Netz}(L,Y,Z) \\
\mathit{Erreichbar_BW}(Z) & \leftarrow & E(L,\text{“Bockenheimer Warte”},Z) \\
\mathit{Station}(S) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
\mathit{Station}(Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
\mathit{Ans}(Z) & \leftarrow & \mathit{Station}(Z),\ \neg\mathit{Erreichbar_BW}(Z)
\end{array}$$

- Die Negation ist hier nicht mit der Rekursion verschränkt.
- Sie kann angewendet werden, nachdem $\mathit{Erreichbar\_BW}(\cdot)$ vollständig berechnet ist.
- $\rightsquigarrow$ Grundidee für stratifiziertes Datalog$^\neg$.

Folie 176

Stratifiziertes Datalog$^\neg$ — Präzise

Definition 4.26. Sei P ein Datalog$^\neg$-Programm.
Eine *Stratifizierung von P* ist eine Folge $P^1,\ldots,P^m$ von Datalog$^\neg$-Programmen, so dass $m \geqslant 1$ ist und es eine Abbildung $\sigma : \mathit{idb}(P) \to \{1,\ldots,m\}$ gibt, so dass gilt:

(1) $P^1,\ldots,P^m$ ist eine *Partition von P* (d.h. $P = P^1 \,\dot{\cup}\, \cdots \,\dot{\cup}\, P^m$),

(2) für jedes idb-Prädikat R von P gilt: alle Regeln, in deren Kopf R vorkommt, gehören zu $P^{\sigma(R)}$,

(3) kommt ein $S \in \mathit{idb}(P)$ im Rumpf einer Regel mit Kopf R vor, so ist $\sigma(S) \leqslant \sigma(R)$, und

(4) kommt ein $S \in idb(P)$ *negiert* im Rumpf einer Regel mit Kopf R vor, so ist $\sigma(S) < \sigma(R)$.

Notation:

- P^i heißt *i-tes Stratum* bzw. *i-te Schicht* der Stratifizierung $P^1, \ldots, P^m$ *("Stratum": lat. für "Schicht"; Plural: Strata)*
- σ heißt *Stratifizierungs-Abbildung*
- Ein Datalog$^\neg$-Programm P heißt *stratifizierbar*, falls es eine Stratifizierung von P gibt. *Stratifiziertes Datalog*$^\neg$ bezeichnet die Menge aller stratifizierbaren Datalog$^\neg$-Programme.

Folie 177

Stratifiziertes Datalog$^\neg$ — Beispiele

$P^1 :=$

$$\begin{array}{rcl} E(L,S,Z) & \leftarrow & BVG(L,S,Z) \\ E(L,S,Z) & \leftarrow & E(L,S,Y),\ \textit{U-Bahn-Netz}(L,Y,Z) \\ \textit{Erreichbar\_BW}(Z) & \leftarrow & E(L,\text{“Bockenheimer Warte”},Z) \\ \textit{Station}(S) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\ \textit{Station}(Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \end{array}$$

$P^2 :=$

$$\textit{Ans}(Z) \ \leftarrow\ \textit{Station}(Z),\ \neg\textit{Erreichbar\_BW}(Z)$$

ist eine Stratifizierung des Datalog$^\neg$-Programms aus Beispiel 4.25.

Das Datalog$^\neg$-Programm $R(x) \leftarrow A(x),\ \neg R(x)$ ist *nicht stratifizierbar.*

Das Datalog$^\neg$-Programm

$$\begin{array}{rcl} R(x) & \leftarrow & A(x),\ \neg S(x) \\ S(x) & \leftarrow & A(x),\ \neg R(x) \end{array}$$

auch nicht.

Folie 178

Test auf Stratifizierbarkeit

Definition 4.27. Sei P ein Datalog$^\neg$-Programm. Der *Abhängigkeitsgraph* $G_P = (V_P, E_P^+, E_P^-)$ ist der gerichtete Graph mit

- Knotenmenge $V_P := sch(P)$
- Kantenmengen

$$E_P^+ := \left\{ (R,S) \ : \ \begin{array}{l} \text{es gibt eine Regel in } P\text{, in deren Kopf} \\ S \text{ vorkommt und in deren Rumpf } R \text{ \textit{po-}} \\ \text{\textit{sitiv} vorkommt} \end{array} \right\}$$

$$E_P^- := \left\{ (R,S) \ : \ \begin{array}{l} \text{es gibt eine Regel in } P\text{, in deren Kopf} \\ S \text{ vorkommt und in deren Rumpf } R \text{ \textit{ne-}} \\ \text{\textit{gativ} vorkommt} \end{array} \right\}$$

Beispiel: *Siehe Tafel:* Abhängigkeitsgraph für

$$\begin{array}{rcl}
E(L,S,Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
E(L,S,Z) & \leftarrow & E(L,S,Y),\ \textit{U-Bahn-Netz}(L,Y,Z) \\
\textit{Erreichbar_BW}(Z) & \leftarrow & E(L, \text{“\textit{Bockenheimer Warte}”}, Z) \\
\textit{Station}(S) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
\textit{Station}(Z) & \leftarrow & \textit{U-Bahn-Netz}(L,S,Z) \\
\textit{Ans}(Z) & \leftarrow & \textit{Station}(Z),\ \neg\textit{Erreichbar_BW}(Z)
\end{array}$$

Folie 179

Test auf Stratifizierbarkeit

Proposition 4.28. *Für jedes Datalog$^\neg$-Programm P gilt:*

$$P \text{ \textit{ist stratifizierbar}} \iff \begin{array}{l} \text{\textit{Im Abhängigkeitsgraph} } G_P \text{ \textit{gibt es}} \\ \text{\textit{keinen Kreis, in dem eine Kante}} \\ \text{\textit{aus} } E_P^- \text{ \textit{vorkommt.}} \end{array}$$

Beweis:

“$\Longrightarrow$”: Sei σ eine Stratifizierungs-Abbildung von P.
Angenommen, es gibt einen Kreis von R nach R, auf dem mindestens eine Kante aus E_P^- vorkommt.
Dann gilt: $\sigma(R) > \sigma(R)$. *Widerspruch!*

“$\Longleftarrow$”: Idee: Nutze eine *topologische Sortierung* der *starken Zusammenhangskomponenten von* $(E_P^+ \cup E_P^-)$, um die einzelnen Schichten zu definieren.

Details: *Übung.*

Folie 180

Stratifiziertes Datalog$^\neg$ — Semantik

- Sei $P^1, \ldots, P^m$ eine Stratifizierung eines Datalog$^\neg$-Programms P.
- Betrachte jede Schicht P^i als ein semipositives Datalog$^\neg$-Programm mit
  $edb(P^i) \ \subseteq \ edb(P) \cup idb(P^1) \cup \cdots \cup idb(P^{i-1})$
- Sei $\mathbf{I} \in edb(P)$.
  Die *Semantik* $[\![P]\!](\mathbf{I})$ *von* P *auf* $\mathbf{I}$ ist folgendermaßen definiert:
  $[\![P]\!](\mathbf{I}) := \mathbf{I}^m$,
  wobei
  $$\begin{array}{rcl} \mathbf{I}^1 & := & [\![P^1]\!](\mathbf{I}) \\ \mathbf{I}^2 & := & [\![P^2]\!](\mathbf{I}^1) \\ \vdots & \vdots & \vdots \\ \mathbf{I}^m & := & [\![P^m]\!](\mathbf{I}^{m-1}) \end{array}$$

Man kann leicht zeigen, dass Folgendes gilt:

(a) Obige Definition hängt nicht von der konkreten Wahl der Stratifizierung von P ab.
D.h. für je zwei verschiedene Stratifizierungen $P^1, \ldots, P^m$ und $Q^1, \ldots, Q^n$ von P gilt: $[\![P^m]\!](\mathbf{I}^{m-1}) \ = \ [\![Q^n]\!](\mathbf{I}^{n-1})$.

(b) $[\![P]\!](\mathbf{I})$ ist der (eindeutig definierte) kleinste Fixpunkt $\mathbf{J}$ von $T_P(\cdot)$ mit $\mathbf{J}|_{edb(P)} = \mathbf{I}$.

Folie 181

Spezialfall: nr-Datalog$^\neg$

$$\begin{array}{rcl} \text{nr-Datalog}^\neg & = & \text{nr-Datalog mit Negation} \\ & = & \text{stratifiziertes Datalog}^\neg\text{, eingeschränkt auf Programme } P\text{, in deren Abhängigkeitsgraph} \\ & & \text{es keinen gerichteten Kreis (über } (E_P^+ \cup E_P^-)) \text{ gibt.} \end{array}$$

Satz 4.29. *Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen beschreiben:*

(a) nr-Datalog$^{\neg}$-Anfragen

(b) Relationale Algebra

(c) Relationenkalkül.

Bemerkung: Die Übersetzung von nr-Datalog$^{\neg}$ in eine der anderen Sprachen kann mehr als polynomiell viel Zeit beanspruchen, da nr-Datalog$^{\neg}$-Programme u. U. viel kürzer sind als äquivalente Anfragen der anderen Sprachen.

Beweis: Der Beweis wird in den folgenden Kapiteln zur Relationalen Algebra (Kapitel 6) und zum Relationenkalkül (Kapitel 7) gegeben.

Kapitel 5

Funktionale Abhängigkeiten

5.1 Notationen

Folie 182

Zur Erinnerung — Benannte Perspektive

In Kapitel 2 haben wir festgelegt:

- Eine abzählbar unendliche Menge **att** von *Attributnamen*. Diese Menge ist *geordnet* via $\leqslant_{\mathbf{att}}$.
- Eine abzählbar unendliche Menge **rel** von *Relationsnamen*. Die Mengen **att**, **dom**, **rel** sind disjunkt.
- Eine Funktion $\mathit{sorte} : \mathbf{rel} \to \mathcal{P}_e(\mathbf{att})$, die jedem Relationsnamen eine endliche Menge von Attributen zuordnet ... und zwar so, dass f.a. $U \subseteq_e \mathbf{att}$ gilt: es gibt unendlich viele $R \in \mathbf{rel}$ mit $\mathit{sorte}(R) = U$.
- Ein *Relationsschema* ist einfach ein Relationsname R.
- Manchmal schreiben wir kurz $R[U]$ für $\mathit{sorte}(R) = U$.
- Ein R-*Tupel* ist eine Funktion $t : \mathit{sorte}(R) \to \mathbf{dom}$.
- Eine R-*Relation* ist eine endliche Menge von R-Tupeln.
- $\mathit{inst}(R)$ bezeichnet die Menge aller R-Relationen.
- $\mathit{inst}(U)$ bezeichnet die Menge aller Relationen über einem Relationsschema der Sorte U (für $U \subseteq_e \mathbf{att}$)

Folie 183

Motivation

Ziel beim Datenbank-Entwurf:
Ein DB-Schema entwickeln, so dass Informationen zum gewünschten Anwendungsbereich „sinnvoll“ gespeichert werden können.
Insbesondere: *Wenn möglich, Redundanzen und Inkonsistenzen vermeiden.*

Beispiel: Relation *Warenlager[Bauteil-Nr, Lager-Nr, Menge, Ort]*

Warenlager:

Bauteil-Nr	Lager-Nr	Menge	Ort
2411	2	200	Riedberg
2412	3	300	Bornheim
3001	1	100	Hanau
2415	2	100	Riedberg

Unschön: Redundanz — der Ort von Lager 2 ist mehrfach gespeichert.

Dadurch können Inkonsistenzen auftreten: *Update-Anomalien* = *Inkonsistenzen, die durch Aktualisierung der DB auftreten können:*

- *Änderungs-Anomalie:* den Ort in Zeile 1 durch "Westend" ersetzen
  $\rightsquigarrow$ Adresse von Lager 2 nicht mehr eindeutig
- *Lösch-Anomalie:* Löschen von Zeile 2
  $\rightsquigarrow$ Information über die Adresse von Lager 3 geht verloren
- *Einfüge-Anomalie:* Die Adresse eines neuen Lagers kann erst dann eingefügt werden, wenn mindestens ein Bauteil dort gelagert wird.

Folie 184

Zur Vermeidung dieser Update-Anomalien:
Informationen auf 2 Relationen *Adressen[Lager-Nr,Ort]* und *Lagerung[Bauteil-Nr,Lager-Nr,Menge]* aufteilen

Adressen:

Lager-Nr	Ort
2	Riedberg
3	Bornheim
1	Hanau

Abhängigkeit:
Lager-Nr $\rightarrow$ Ort

Lagerung:

Bauteil-Nr	Lager-Nr	Menge
2411	2	200
2412	3	300
3001	1	100
2415	2	100

Abhängigkeit:
Bauteil-Nr, Lager-Nr → Menge

Zur Anfrage-Optimierung:
Die „optimierte Anfrage“ muss nur auf solchen Datenbanken äquivalent zur Original-Anfrage sein, die die obigen Abhängigkeiten erfüllen.

$\leadsto$ die minimale, solchermaßen äquivalente Anfrage ist evtl. noch kleiner als die, die durch die Tableau-Minimierung aus Kapitel 3.4 gefunden wird.

Folie 185

Notation

Attributnamen :	$A,\ B,\ C,\ A_1,\ A_2,\ \ldots$	
Attributmengen :	$U,\ X,\ Y,\ Z,\ X_1,\ X_2,\ \ldots$	(endliche Mengen von Attributnamen)
Relationen :	$I,\ J$	
$\{A,B,C\}$:	ABC	
$X \cup Y$:	XY	

Folie 186

Funktionale Abhängigkeiten

Definition 5.1. Sei U eine endliche Menge von Attributnamen.

(a) Eine *funktionale Abhängigkeit* (kurz: *FD*) über U ist ein Ausdruck der Form $X \to Y$, wobei $X, Y \subseteq U$. *(„FD“ steht für „functional dependency“)*

(b) Eine Relation $I \in \mathit{inst}(U)$ *erfüllt* die FD $X \to Y$ (kurz: $I \models X \to Y$), falls
für alle Tupel t und s aus I gilt:

$$\pi_X(t) = \pi_X(s) \quad \Longrightarrow \quad \pi_Y(t) = \pi_Y(s)$$

D.h.: Wenn t und s in sämtlichen Spalten aus X übereinstimmen, dann stimmen sie auch in jeder Spalte aus Y überein.

(c) Ist $\mathcal{F}$ eine Menge von FDs über U, so gilt

$$I \models \mathcal{F} \quad :\Longleftrightarrow \qquad \text{für alle } f \in \mathcal{F} \text{ gilt } \; I \models f.$$

(d) Eine *Schlüsselbedingung* ist eine FD der Form $X \to U$.

Folie 187

Funktionale Abhängigkeiten und verlustfreie Joins

Proposition 5.2. *Sei $X \to Y$ eine FD über U und sei $Z := U \setminus (X \cup Y)$.*
Für jede Relation $I \in inst(U)$ gilt:
Falls $I \models X \to Y$, so ist $I = \pi_{XY}(I) \bowtie \pi_{XZ}(I)$.

Beweis: Siehe Tafel.

Folgerung: Die in Relation I gespeicherte Information kann „verlustfrei“ auf zwei Relationen aufgeteilt werden (eine mit den Spalten XY und eine mit den Spalten XZ), aus denen die Original-Relation rekonstruiert werden kann. *(Stichwort: „lossless join“)*

Beispiel für einen „verlustreichen Join“: Siehe Tafel.

5.2 The Chase — Die Verfolgungsjagd

Folie 188

Beispiel zu „The Chase — Die Verfolgungsjagd“

Beispiel 5.3. Tableau-Anfrage $Q = (T, t)$ mit

$$T = \begin{array}{|cccc|} A & B & C & D \\ \hline w & x & y & z' \\ w' & x & y' & z \\ \hline \end{array} \qquad \text{und} \qquad t \;=\; (w, x, y, z)$$

Klar: Die Anfrage $Q := (T, t)$ ist *minimal* im Sinne von Kapitel 3.4.

Situation jetzt:

- Gegeben sei die FD-Menge $\mathcal{F} := \{\ B \to D\ \}$
- Q soll nur auf solchen DBs ausgewertet werden, die $\mathcal{F}$ erfüllen
- Ziel: Vereinfache (minimiere) Q.

Details: siehe Tafel.

Folie 189

Äquivalenz bzw. Query Containment bzgl. $\mathcal{F}$

Für ein Relationsschema R und das Datenbankschema $\mathbf{S} := \{R\}$

- identizifieren wir eine R-Relation I mit der Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ mit $\mathbf{I}(R) = I$.
- sagen wir auch „Q ist eine Anfrage über R“ statt „Q ist eine Anfrage über $\mathbf{S}$“.
- schreiben wir $I \models Q$ an Stelle von $\mathbf{I} \models Q$, für eine R-Relation I und die zugehörige Datenbank $\mathbf{I}$ mit $\mathbf{I}(R) = I$. Analog schreiben wir $[\![Q]\!](I)$ statt $[\![Q]\!](\mathbf{I})$.

Definition 5.4. Sei R ein Relationsschema, sei $\mathcal{I} \subseteq \mathit{inst}(R)$, und seien Q_1 und Q_2 Anfragen über R.

- $Q_1 \sqsubseteq_{\mathcal{I}} Q_2 \quad :\iff \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) \ \subseteq \ [\![Q_2]\!](I)$
- $Q_1 \equiv_{\mathcal{I}} Q_2 \quad :\iff \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) \ = \ [\![Q_2]\!](I)$.

Sei $\mathcal{F}$ eine Menge von FDs über R.

- $\operatorname{Mod}(\mathcal{F}) \ := \ \operatorname{Mod}_R(\mathcal{F}) \ := \ \{\, I \in \mathit{inst}(R) \ : \ I \models \mathcal{F} \,\}$
- $Q_1 \sqsubseteq_{\mathcal{F}} Q_2 \quad :\iff \quad Q_1 \sqsubseteq_{\operatorname{Mod}(\mathcal{F})} Q_2$
- $Q_1 \equiv_{\mathcal{F}} Q_2 \quad :\iff \quad Q_1 \equiv_{\operatorname{Mod}(\mathcal{F})} Q_2$

Folie 190

Vereinbarungen für den Rest von Kapitel 5.2

Der Einfachheit halber betrachten wir im Folgenden

- ein festes Relationsschema R
- Mengen $\mathcal{F}$ von FDs über R, in denen o.B.d.A. jede FD von der Form $X \to A$ mit $X \subseteq \mathit{sorte}(R)$ und $A \in \mathit{sorte}(R)$ ist
- eine feste lineare Ordnung $<$ auf der Variablenmenge var
- nur Tableau-Anfragen $Q = (T, t)$ über R, in denen *keine Konstanten vorkommen*

Bemerkung: Die Ergebnisse aus Kapitel 5.2 können leicht verallgemeinert werden auf Anfragen mit Konstanten und auf Anfragen über einem Datenbankschema.

Folie 191

Regel für die Verfolgungsjagd

Definition 5.5. *FD-Regel:*

Sei $f := (X \to A)$ eine FD über R, sei (T,t) eine Tableau-Anfrage über R.
Seien u und v Zeilen von T mit $\pi_X(u) = \pi_X(v)$ und $u(A) \neq v(A)$.
Sei $\{x,y\} := \{u(A), v(A)\} \subseteq \text{var}$ und sei $x < y$.

Anwenden der FD f auf u,v in (T,t) liefert die Tableau-Anfrage $\big(h(T), h(t)\big)$, wobei
h die Substitution mit $h(y) := x$ und $h(z) := z$ für alle $z \in \text{Var}((T,t))$ mit $z \neq y$.

Folie 192

Anwenden der FD-Regel erhält Äquivalenz bzgl. $\mathcal{F}$

Proposition 5.6. *Sei $\mathcal{F}$ eine Menge von FDs über R,
sei $f := (X \to A) \in \mathcal{F}$,
sei $Q := (T,t)$ eine Tableau-Anfrage über R und
sei $Q' := (T',t')$ eine Tableau-Anfrage, die durch 1-maliges Anwenden der FD-Regel mit einer FD $f \in \mathcal{F}$ aus Q entsteht.*

Dann gilt: $Q \equiv_{\mathcal{F}} Q'$.

Beweis: Siehe Tafel.

Folie 193

Verfolgungssequenzen

Definition 5.7.

(a) Eine *Verfolgungssequenz für (T,t) mittels $\mathcal{F}$* ist eine Folge

$$(T_0,t_0),\ (T_1,t_1),\ (T_2,t_2),\ \dots$$

für die gilt:

- $(T_0,t_0) = (T,t)$ und
- für jedes $i \geqslant 0$ entsteht (T_{i+1},t_{i+1}) durch 1-maliges Anwenden der FD-Regel mit einer FD aus $\mathcal{F}$ auf (T_i,t_i).

(b) Die Verfolgungssequenz ist *terminiert*, falls sie endlich ist und auf ihr letztes Element (T_m, t_m) keine FD-Regel mit einer FD aus $\mathcal{F}$ mehr angewendet werden kann.
(T_m, t_m) heißt dann das *Resultat* der Sequenz.

Folie 194

Notation: $T \models \mathcal{F}$

Definition 5.8.

(a) Ein Tableau T über R *erfüllt* die FD $X \to A$ (kurz: $T \models X \to A$), falls für alle Zeilen u und v von T gilt:

$$\pi_X(u) = \pi_X(v) \quad \Longrightarrow \quad u(A) = v(A)$$

(D.h.: Wenn u und v in sämtlichen Spalten aus X übereinstimmen, dann stimmen sie auch in der Spalte A überein.)

(b) Ist $\mathcal{F}$ eine Menge von FDs über R, so ist

$$T \models \mathcal{F} \quad :\Longleftrightarrow \quad T \models f, \text{ für alle } f \in \mathcal{F}.$$

Folie 195

Eigenschaften des Resultats einer Verfolgungssequenz

Lemma 5.9. *Sei (T', t') das Resultat einer terminierten Verfolgungssequenz für (T, t) mittels $\mathcal{F}$.*
Dann gilt: $\quad (T', t') \equiv_{\mathcal{F}} (T, t) \quad$ *und* $\quad T' \models \mathcal{F}$.

Beweis: Übung.

Beobachtung: Jede Verfolgungssequenz ist endlich und kann zu einer terminierten Sequenz vervollständigt werden.

Bemerkenswert: Man kann zeigen, dass alle terminierten Verfolgungssequenzen für (T, t) mittels $\mathcal{F}$ dasselbe Resultat liefern. Dies wird auch *Church-Rosser-Eigenschaft* genannt.

Folie 196

Die Church-Rosser-Eigenschaft der Verfolgungsjagd

Theorem 5.10. *Sei (T, t) eine Tableau-Anfrage über R und sei $\mathcal{F}$ eine Menge von FDs über R. Dann gilt: Alle terminierten Verfolgungssequenzen für (T, t) mittels $\mathcal{F}$ liefern dasselbe Resultat.*

Hier ohne Beweis.
Ein Beweis findet sich am Ende von Kapitel 8.4 des Buchs *[AHV]*.

Definition 5.11. Ist (T, t) eine Tableau-Anfrage über R und $\mathcal{F}$ eine Menge von FDs über R, so bezeichnet $\mathit{chase}(T, t, \mathcal{F})$ das Resultat einer (bzw. sämtlicher) terminierter Verfolgungssequenzen für (T, t) mittels $\mathcal{F}$.

Bemerkung: Von Lemma 5.9 wissen wir, dass $\mathit{chase}(T, t, \mathcal{F}) \equiv_{\mathcal{F}} (T, t)$ und $\mathit{chase}(T, t, \mathcal{F}) \models \mathcal{F}$.

Folie 197

Berechnung von *chase*$(T, t, \mathcal{F})$

Korollar 5.12. *Es gibt einen Polynomialzeit-Algorithmus, der bei Eingabe einer Tableau-Anfrage (T, t) über R und einer Menge $\mathcal{F}$ von FDs über R die Tableau-Anfrage $\mathit{chase}(T, t, \mathcal{F})$ berechnet.*

Beweis:

Algorithmus:

(1) Wiederhole so lange, bis keine FD-Regel bzgl. $\mathcal{F}$ mehr auf (T, t) anwendbar ist:

  (1.1) (T', t') sei das Resultat der Anwendung einer FD-Regel bzgl. $\mathcal{F}$ auf (T, t).

  (1.2) Setze $(T, t) := (T', t')$.

(2) Gib (T, t) aus.

Korrektheit folgt direkt aus der Church-Rosser-Eigenschaft.
Polynomielle Laufzeit, da jede FD $f \in \mathcal{F}$ auf jedes Paar u, v von Zeilen von T höchstens 1-mal angewendet werden kann und da jede einzelne Anwendung der FD-Regel nur polynomiell viel Zeit benötigt. $\square$

Folie 198

Äquivalenz von Anfragen bzgl. $\mathcal{F}$

Theorem 5.13. *Seien $Q_1 := (T_1, t_1)$ und $Q_2 := (T_2, t_2)$ Tableau-Anfragen über R und sei $\mathcal{F}$ eine Menge von FDs über R. Dann gilt:*

(a) $Q_1 \sqsubseteq_{\mathcal{F}} Q_2 \iff chase(T_1, t_1, \mathcal{F}) \sqsubseteq chase(T_2, t_2, \mathcal{F})$ *und*

(b) $Q_1 \equiv_{\mathcal{F}} Q_2 \iff chase(T_1, t_1, \mathcal{F}) \equiv chase(T_2, t_2, \mathcal{F})$ *und*

Beweis: Siehe Tafel.

Bemerkung: Aus Theorem 5.13, Korollar 5.12 und Korollar 3.36 folgt insbesondere, dass „$Q_1 \sqsubseteq_{\mathcal{F}} Q_2$" bzw. „$Q_1 \equiv_{\mathcal{F}} Q_2$" entscheidbar ist und zur Komplexitätsklasse NP gehört.

Folie 199

Anfrage-Minimierung bzgl. $\mathcal{F}$

Vorgehensweise:

- *Eingabe:* Tableau-Anfrage $Q = (T, t)$ über R und Menge $\mathcal{F}$ von FDs über R
- *Schritt 1:* Berechne $Q' := (T', t') := chase(T, t, \mathcal{F})$. *Klar:* $Q' \equiv_{\mathcal{F}} Q$
- *Schritt 2:* Nutze den Algorithmus aus Theorem 3.39(a) um eine *minimale* Tableau-Anfrage $Q'' := (T'', t'')$ mit $Q'' \equiv Q'$ zu berechnen
  Insbesondere gilt: $Q'' \equiv_{\mathcal{F}} Q' \equiv_{\mathcal{F}} Q$

Notation: Ist $Q = (T, t)$ eine Tableau-Anfrage, so schreibe $min(Q) := min(T, t)$, um die gemäß Theorem 3.39(a) minimale zu Q äquivalente Tableau-Anfrage zu bezeichnen.

Lemma 5.14. *Sei $Q = (T, t)$ eine Tableau-Anfrage über R und sei $\mathcal{F}$ eine Menge von FDs über R.*
Dann gilt: $\big|min\big(chase(T, t, \mathcal{F})\big)\big| \leqslant |min(T, t)|$.

Hier ohne Beweis.
Eine Beweisskizze findet sich auf Seite 178 des Buchs *[AHV]*.

5.3 Der Armstrong-Kalkül

Folie 200

Implikation von Abhängigkeiten

Gegeben: Eine Menge $\mathcal{F}$ von FDs über U.

Frage: Welche anderen FDs folgen aus $\mathcal{F}$?

Beispiel 5.15. Sei $U := \{A, B, C, D, E\}$ und
$\mathcal{F} := \{ A \to C,\ B \to C,\ CD \to E \}$.
Dann gilt für jede Relation I mit $I \models \mathcal{F}$ auch, dass $I \models AD \to E$.

Definition 5.16. Seien $\mathcal{F}$ und $\mathcal{G}$ zwei Mengen von FDs über U.

(a) $\mathcal{F}$ *impliziert* $\mathcal{G}$ (kurz: $\mathcal{F} \models_U \mathcal{G}$ bzw. $\mathcal{F} \models \mathcal{G}$, falls U aus dem Kontext klar ist), falls für alle Relationen $I \in \mathit{inst}(U)$ gilt: Falls $I \models \mathcal{F}$, so auch $I \models \mathcal{G}$.
Besteht $\mathcal{G}$ aus einer einzigen FD f, so schreibe auch $\mathcal{F} \models f$ statt $\mathcal{F} \models \{f\}$.

(b) $\mathcal{F}$ und $\mathcal{G}$ sind *äquivalent* (kurz: $\mathcal{F} \equiv_U \mathcal{G}$ bzw. $\mathcal{F} \equiv \mathcal{G}$, falls U aus dem Kontext klar ist), falls $\mathcal{F} \models \mathcal{G}$ und $\mathcal{G} \models \mathcal{F}$.

(c) Die *Hülle* von $\mathcal{F}$ über U (kurz: $\mathcal{F}^{*,U}$ bzw. $\mathcal{F}^{*}$, falls U aus dem Kontext klar ist), ist definiert als
$\mathcal{F}^{*,U} := \{ X \to Y \ :\ X, Y \subseteq U \ \textit{und}\ \mathcal{F} \models X \to Y \}$.

Klar: Für alle $X \subseteq U$ und alle $Y \subseteq X$ gilt $X \to Y \in \mathcal{F}^{*,U}$.
Insbesondere: $2^{|U|} \leqslant |\mathcal{F}^{*,U}| \leqslant 2^{2 \cdot |U|}$.

Folie 201

Fragen: Wie kann man *(1)* testen, ob $\mathcal{F} \models_U X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models_U X \to Y$ spricht über *alle* Relationen $I \in \mathit{inst}(U)$. Das sind unendlich viele (da **dom** unendlich ist).

Lösung für (1): Reduziere das Problem „$\mathcal{F} \models_U X \to Y$?“ auf das Problem „$Q_X \sqsubseteq_{\mathcal{F}} Q_Y$?“ für geeignete Tableau-Anfragen Q_X und Q_Y.
Beachte: Einen Algorithmus zum Lösen des letzteren Problems haben wir bereits kennengelernt.

Satz 5.17.
Es gibt einen Linearzeit-Algorithmus, der bei Eingabe einer endlichen Attributmenge U, einer Menge $\mathcal{F}$ von FDs über U und einer FD $(X \to Y)$ über U zwei azyklische Tableau-Anfragen Q_X und Q_Y berechnet, für die gilt: $\mathcal{F} \models_U (X \to Y) \iff Q_X \sqsubseteq_{\mathcal{F}} Q_Y$.
Ob $Q_X \sqsubseteq_{\mathcal{F}} Q_Y$ gilt, kann in Zeit polynomiell in der Größe von U, $\mathcal{F}$, X und Y entschieden werden.

Beweis. Sei $r := |U|$ und sei $U = \{A_1, \ldots, A_r\}$. Es gilt:

$$\begin{aligned} & \mathcal{F} \models_U (X \to Y) \\ \iff \; & \text{für alle } I \in \mathit{inst}(U) \text{ mit } I \models \mathcal{F} \text{ gilt: } I \models (X \to Y) \\ \iff \; & \text{für alle } I \in \mathit{inst}(U) \text{ mit } I \models \mathcal{F} \text{ gilt:} \\ & \{(t,s) \,:\, t \in I,\ s \in I,\ \pi_X(t) = \pi_X(s)\} \subseteq \\ & \{(t,s) \,:\, t \in I,\ s \in I,\ \pi_Y(t) = \pi_Y(s)\} \end{aligned} \tag{5.1}$$

Wir konstruieren daher die Anfragen $Q_X = (T_X, u_X)$ und $Q_Y = (T_Y, u_Y)$ so, dass für alle $I \in \mathit{inst}(U)$ gilt:

- $[\![Q_X]\!](I) \;=\; \{(t,s) \,:\, t \in I,\ s \in I, \pi_X(t) = \pi_X(s)\}$ und
- $[\![Q_Y]\!](I) \;=\; \{(t,s) \,:\, t \in I,\ s \in I, \pi_Y(t) = \pi_Y(s)\}$

Zur Konstruktion von Q_X und Q_Y wählen wir paarweise verschiedene Variablen $x_1, \ldots, x_r, y_1, \ldots, y_r,\ z_{1,1}, \ldots, z_{1,r},\ z_{2,1}, \ldots, z_{2,r}$ und setzen für alle $i \in \{1,2\}$ und alle $j \in \{1, \ldots, r\}$

$$v_{i,j} := \begin{cases} x_j & \text{falls } A_j \in X \\ z_{i,j} & \text{sonst} \end{cases} \qquad\qquad w_{i,j} := \begin{cases} y_j & \text{falls } A_j \in Y \\ z_{i,j} & \text{sonst} \end{cases}$$

und wählen

$$T_X = \begin{array}{|ccc|} \hline A_1 & \cdots & A_r \\ \hline v_{1,1} & \cdots & v_{1,r} \\ v_{2,1} & \cdots & v_{2,r} \\ \hline \end{array} \qquad \text{und} \qquad u_X := (v_{1,1}, \ldots, v_{1,r},\ v_{2,1}, \ldots, v_{2,r}),$$

$$T_Y = \begin{array}{|ccc|} \hline A_1 & \cdots & A_r \\ \hline w_{1,1} & \cdots & w_{1,r} \\ w_{2,1} & \cdots & w_{2,r} \\ \hline \end{array} \qquad \text{und} \qquad u_Y := (w_{1,1}, \ldots, w_{1,r},\ w_{2,1}, \ldots, w_{2,r}).$$

Beispiel: Für $U = \{A_1, A_2, A_3, A_4\}$, $X = \{A_1, A_2\}$ und $Y = \{A_3\}$ ist

$$T_X = \begin{array}{|cccc|}\hline A_1 & A_2 & A_3 & A_4 \\ \hline x_1 & x_2 & z_{1,3} & z_{1,4} \\ x_1 & x_2 & z_{2,3} & z_{2,4} \\ \hline \end{array} \quad \text{und} \quad u_X := (x_1, x_2, z_{1,3}, z_{1,4}, x_1, x_2, z_{2,3}, z_{2,4}),$$

$$T_Y = \begin{array}{|cccc|}\hline A_1 & A_2 & A_3 & A_4 \\ \hline z_{1,1} & z_{1,2} & y_3 & z_{1,4} \\ z_{2,1} & z_{2,2} & y_3 & z_{2,4} \\ \hline \end{array} \quad \text{und} \quad u_Y := (z_{1,1}, z_{1,2}, y_3, z_{1,4}, z_{2,1}, z_{2,2}, y_3, z_{2,4}).$$

Mit dieser Wahl von $Q_X = (T_X, u_X)$ und $Q_Y = (T_Y, u_Y)$ gilt tatsächlich für jedes $I \in \mathit{inst}(U)$, dass

- $[\![Q_X]\!](I) \;=\; \{(t,s) \;:\; t \in I,\; s \in I, \pi_X(t) = \pi_X(s)\}$ und
- $[\![Q_Y]\!](I) \;=\; \{(t,s) \;:\; t \in I,\; s \in I, \pi_Y(t) = \pi_Y(s)\}$.

Mit (5.1) erhalten wir, dass Folgendes gilt:

$$\begin{aligned} & \mathcal{F} \models_U (X \to Y) \\ \iff\; & \text{für alle } I \in \mathit{inst}(U) \text{ mit } I \models \mathcal{F} \text{ gilt: } [\![Q_X]\!](I) \subseteq [\![Q_Y]\!](I) \\ \iff\; & Q_X \sqsubseteq_{\mathcal{F}} Q_Y\,. \end{aligned}$$

Da T_X und T_Y jeweils nur aus 2 Zeilen besteht, sind Q_X und Q_Y offensichtlicherweise azyklisch.
Gemäß Theorem 5.13 gilt $Q_X \sqsubseteq_{\mathcal{F}} Q_Y$ genau dann, wenn $\mathit{chase}(T_X, u_X, \mathcal{F}) \sqsubseteq \mathit{chase}(T_Y, u_Y, \mathcal{F})$. Da Q_X und Q_Y azyklisch sind, sind auch die Anfragen $Q'_X := \mathit{chase}(T_X, u_X, \mathcal{F})$ und $Q'_Y := \mathit{chase}(T_Y, u_Y, \mathcal{F})$ azyklisch. Aus der Kombination vom Homomorphismussatz und dem Polynomialzeit-Algorithmus zum Auswerten azyklischer Anfragen erhalten wir einen Algorithmus, der in Zeit polynomiell in der Größe von U, $\mathcal{F}$, X und Y entscheidet, ob $Q_X \sqsubseteq_{\mathcal{F}} Q_Y$ gilt. Dies beendet den Beweis von Satz 5.17. $\square$

Folie 202

Fragen: Wie kann man *(1)* testen, ob $\mathcal{F} \models_U X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models_U X \to Y$ spricht über *alle* Relationen $I \in \mathit{inst}(U)$. Das sind unendlich viele (da **dom** unendlich ist).

Lösung für (2): Finde eine endliche Regelmenge $\mathfrak{K}$, mit deren Hilfe aus $\mathcal{F}$ neue Abhängigkeiten hergeleitet werden können.
(Analog zum Begriff der „Kalküle" aus der Vorlesung *Logik in der Informatik*)

Ziel:

- Alle FDs, die sich mit $\mathfrak{K}$ aus $\mathcal{F}$ ableiten lassen, sind in $\mathcal{F}^*$ $\rightsquigarrow$ $\mathfrak{K}$ *ist korrekt*
- Alle FDs aus $\mathcal{F}^*$ lassen sich in $\mathfrak{K}$ aus $\mathcal{F}$ herleiten $\rightsquigarrow$ $\mathfrak{K}$ *ist vollständig*

Folie 203

Der Armstrong-Kalkül

Der *Armstrong-Kalkül* $\mathfrak{K}_A$ *über* U ist wie folgt definiert:

Axiome:

(A) Für alle $X \subseteq U$ und alle $Y \subseteq X$ ist

$$\frac{}{X \to Y}$$ ein Axiom des Armstrong-Kalküls über U.

Weitere Regeln:

(E) Für alle $X, Y, Z \subseteq U$ ist die *Erweiterungsregel*

$$\frac{X \to Y}{XZ \to YZ}$$ eine Regel des Armstrong-Kalküls über U.

(T) Für alle $X, Y, Z \subseteq U$ ist die *Transitivitätsregel*

$$\frac{X \to Y \qquad Y \to Z}{X \to Z}$$ eine Regel des Armstrong-Kalküls über U.

Folie 204

Ableitungen im Armstrong-Kalkül

Definition 5.18. Sei f eine FD über U und sei $\mathcal{F}$ eine Menge von FDs über U.

Eine *Ableitung von f aus $\mathcal{F}$ im Armstrong-Kalkül* ist eine endliche Folge $(f_1, \ldots, f_\ell)$ von FDs über U, so dass $\ell \geqslant 1$ und

- $f_\ell = f$ und
- für alle $i \in \{1, \ldots, \ell\}$ gilt:
  - $f_i \in \mathcal{F}$ oder
  - $\frac{}{f_i}$ ist ein Axiom (A) des Armstrong-Kalküls über U oder
  - es gibt ein $j < i$, so dass $\frac{f_j}{f_i}$ eine Erweiterungsregel (E) des Armstrong-Kalküls über U ist oder
  - es gibt $j, k < i$, so dass $\frac{f_j \quad f_k}{f_i}$ eine Transitivitätsregel (T) des Armstrong-Kalküls über U ist.

Wir schreiben $\mathcal{F} \vdash_U f$ (bzw. $\mathcal{F} \vdash f$, falls U aus dem Kontext klar ist), um auszudrücken, dass es eine Ableitung von f aus $\mathcal{F}$ im Armstrong-Kalkül über U gibt.

Mit $\mathrm{abl}_{\mathfrak{K}_A}(\mathcal{F})$ bezeichnen wir die Menge aller FDs f über U, für die gilt: $\mathcal{F} \vdash_U f$.

Folie 205

Beispiel für eine Ableitung im Armstrong-Kalkül

Sei $U := \{A, B, C, D, E\}$ und $\mathcal{F} := \{\ A \to C,\ B \to C,\ CD \to E\ \}$.

Eine Ableitung von $AD \to E$ aus $\mathcal{F}$ im Armstrong-Kalkül über U:

$$\Big(\ A \to C,\quad AD \to CD,\quad CD \to E,\quad AD \to E\ \Big)$$

Erläuterung:

(1)	$A \to C$	$\in \mathcal{F}$
(2)	$AD \to CD$	durch Anwenden der Erweiterungsregel (E) auf (1)
(3)	$CD \to E$	$\in \mathcal{F}$
(4)	$AD \to E$	durch Anwenden der Transitivitätsregel (T) auf (2) und (3).

Folie 206

Ein einfaches Lemma zum Armstrong-Kalkül

Lemma 5.19. *Sei U eine endliche Menge von Attributnamen und sei $\mathcal{F}$ eine Menge von FDs über U. Seien $X, Z \subseteq U$ so, dass für jedes $A \in Z$ gilt: $\mathcal{F} \vdash_U (X \to A)$. Dann gilt auch: $\mathcal{F} \vdash_U (X \to Z)$.*

Beweis.
Sei $Z = \{A_1, \ldots, A_k\}$. Laut Voraussetzung gilt $\mathcal{F} \vdash_U (X \to A_i)$ für jedes $i \in [k]$.
Per Induktion nach i zeigen wir, dass für alle $i \in [k]$ gilt:
$\mathcal{F} \vdash_U (X \to A_1 \cdots A_i)$.

Induktionsanfang $i = 1$: Laut Voraussetzung gilt $\mathcal{F} \vdash_U (X \to A_1)$.

Induktionsschritt $i \mapsto i{+}1$: Gemäß Induktionsannahme gilt:
$\mathcal{F} \vdash_U (X \to A_1 \cdots A_i)$.
Anwenden der Erweiterungsregel (E) liefert, dass $\mathcal{F} \vdash_U (X \to XA_1 \cdots A_i)$.
Außerdem gilt laut Voraussetzung: $\mathcal{F} \vdash_U (X \to A_{i+1})$.
Anwenden der Erweiterungsregel (E) liefert, dass
$\mathcal{F} \vdash_U (XA_1 \cdots A_i \to A_1 \cdots A_i A_{i+1})$.
Anwenden der Transitivitätsregel (T) liefert, dass
$\mathcal{F} \vdash_U (X \to A_1 \cdots A_i A_{i+1})$. □

Folie 207

Korrektheit und Vollständigkeit des Armstrong-Kalküls

Sei U eine endliche Menge von Attributnamen und sei $\mathcal{F}$ eine Menge von FDs über U.
Der Armstrong-Kalkül $\mathfrak{K}_A$ über U heißt

- *korrekt*, falls für jede Menge $\mathcal{F}$ von FDs über U und jede FD f über U gilt:
  Falls $\mathcal{F} \vdash_U f$, so $\mathcal{F} \models_U f$.
- *vollständig*, falls für jede Menge $\mathcal{F}$ von FDs über U und jede FD f über U gilt:
  Falls $\mathcal{F} \models_U f$, so $\mathcal{F} \vdash_U f$.

Theorem 5.20 (Armstrong, 1974)**.** *Der Armstrong-Kalkül $\mathfrak{K}_A$ ist korrekt und vollständig, d.h.:*
Für jede endliche Menge U von Attributnamen, jede FD f über U und jede Menge $\mathcal{F}$ von FDs über U gilt: $\mathcal{F} \vdash_U f \iff \mathcal{F} \models_U f$.

Beweis:
"$\Longrightarrow$": Übung.

"$\Longleftarrow$": Wir zeigen: $\mathcal{F} \not\vdash_U f \implies \mathcal{F} \not\models_U f$.
Laut Voraussetzung gilt also: $\mathcal{F} \not\vdash_U f$. Sei f von der Form $X \to Y$ (für $X, Y \subseteq U$).

Ziel: Konstruiere ein $I \in \mathit{inst}(U)$, so dass gilt:

$$I \models \mathcal{F} \qquad \text{und} \qquad I \not\models (X \to Y).$$

Sobald wir dies erreicht haben, bezeugt I, dass $\mathcal{F} \not\models_U f$, und wir sind fertig.

Zur Konstruktion von I betrachte die Attributmenge

$$X^* \quad := \quad \{A \in U \quad : \quad \mathcal{F} \vdash_U (X \to A)\}. \tag{5.2}$$

Auf Grund des Axioms (A) gilt insbesondere für jedes $A \in X$, dass $A \in X^*$. Es ist also $X \subseteq X^*$.

Behauptung 1: *Es gibt ein $B \in Y$ so dass gilt: $B \notin X^*$.*
Beweis: Angenommen, für jedes $B \in Y$ gilt $B \in X^*$, d.h., $\mathcal{F} \vdash_U (X \to B)$. Dann gilt gemäß Lemma 5.19 auch: $\mathcal{F} \vdash_U (X \to Y)$. Aber $(X \to Y)$ ist ja gerade die FD f, für die laut Voraussetzung gilt: $\mathcal{F} \not\vdash_U f$.
Widerspruch! $\square_{\text{Beh.1}}$

Setze $I := \{t_1, t_2\}$, wobei t_1 und t_2 die beiden folgenden Tupel über U sind:

- $t_1(A) := 1$ für alle $A \in U$
- $t_2(A) := \begin{cases} 1 & \text{für alle } A \in X^* \\ 0 & \text{für alle } A \notin X^*. \end{cases}$

Gemäß Behauptung 1 ist $t_1 \neq t_2$, und es gilt: $\pi_Y(t_1) \neq \pi_Y(t_2)$.
Wegen $X \subseteq X^*$, ist $\pi_X(t_1) = \pi_X(t_2)$. Somit gilt:

$$I \not\models (X \to Y).$$

Um den Beweis abzuschließen, müssen wir also nur noch zeigen, dass gilt: $I \models \mathcal{F}$. Sei dazu $(Z \to Z')$ eine beliebige FD in $\mathcal{F}$. Wir müssen zeigen, dass gilt: $I \models (Z \to Z')$, d.h. für alle $t, s \in I$ mit $\pi_Z(t) = \pi_Z(s)$ ist $\pi_{Z'}(t) = \pi_{Z'}(s)$. Wegen $I = \{t_1, t_2\}$ reicht es, die Situation zu betrachten, in der $t = t_1$ und $s = t_2$ ist.
Falls $\pi_Z(t_1) \neq \pi_Z(t_2)$ ist, so ist nichts zu zeigen. Falls $\pi_Z(t_1) = \pi_Z(t_2)$ ist, so muss gemäß der Definition der Tupel t_1 und t_2 gelten: $Z \subseteq X^*$.

Gemäß der Definition von X^* gilt also für alle $A \in Z$, dass $\mathcal{F} \vdash_U (X \to A)$.
Laut Lemma 5.19 gilt also auch: $\mathcal{F} \vdash_U (X \to Z)$.
Außerdem ist $(Z \to Z') \in \mathcal{F}$. Durch Anwenden der Transitivitätsregel (T) erhalten wir, dass $\mathcal{F} \vdash_U (X \to Z')$. Auf Grund des Axioms (A) gilt außerdem für jedes $A \in Z'$, dass $\mathcal{F} \vdash_U (Z' \to A)$. Anwenden der Transitivitätsregel (T) liefert, dass $\mathcal{F} \vdash_U (X \to A)$. Somit gilt also für alle $A \in Z'$, dass $A \in X^*$.
Gemäß der Definition der Tupel t_1 und t_2 ist daher $\pi_{Z'}(t_1) = \pi_{Z'}(t_2)$. Dies schließt den Beweis von Theorem 5.20 ab. □

Bemerkungen:

- Die Hülle $\mathcal{F}^{*,U}$ kann man also berechnen, indem man alle FDs f berechnet, für die gilt $\mathcal{F} \vdash_U f$.
- Wenn man nur für eine bestimmte FD f testen will, ob $\mathcal{F} \models_U f$, so ist es ziemlich aufwändig, erst ganz $\mathcal{F}^{*,U}$ zu berechnen, da die Hülle immer mindestens $2^{|U|}$ viele Elemente enthält.

Kapitel 6

Relationale Algebra

6.1 Definition und Beispiele

Folie 208

Grenzen der Ausdrucksstärke konjunktiver Anfragen

Wir haben gesehen:

- konjunktive Anfragen können nur *monotone* Anfragefunktionen beschreiben (Satz 3.6) $\rightsquigarrow$ Anfragen mit Negationen der Art

  Welche Regisseure haben noch nie mit "Matt Damon" gearbeitet?

  können nicht in SPC- bzw. SPJR-Algebra gestellt werden.

- konjunktive Anfragen können keine Ver-ODER-ungen der Art

  In welchen Kinos wird "Alien" oder "Brazil" gespielt?

  ausdrücken (vgl. Übungsblatt 1).

Jetzt:
Erweitere SPC- bzw. SPJR-Algebra um die Möglichkeit, auch solche Anfragen zu beschreiben.

Folie 209

Vereinigung und Differenz

Operatoren $\cup$ und $-$:

Diese Operatoren können angewendet werden auf Relationen I und J, die dieselbe Sorte bzw. Stelligkeit haben und liefern als Ausgabe die Relationen

$$I \cup J \ := \ \{\, t \ : \ t \in I \ \textit{oder} \ t \in J \,\}$$

bzw.

$$I - J \ := \ \{\, t \in I \ : \ t \notin J \,\}$$

Folie 210

SPJRU, SPCU und relationale Algebra

Definition 6.1. Sei $\mathbf{S}$ ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der $\mathsf{SPJRU}[\mathbf{S}]$ (bzw. der $\mathsf{SPCU}[\mathbf{S}]$) werden die Definitionen von $\mathsf{SPJR}[\mathbf{S}]$ (bzw. $\mathsf{SPC}[\mathbf{S}]$) um die folgende Regel erweitert:

- Sind Q und P zwei $\mathsf{SPJRU}[\mathbf{S}]$-Anfragen derselben Sorte Σ (bzw. $\mathsf{SPCU}[\mathbf{S}]$- Anfragen derselben Stelligkeit k), so ist $(Q \cup P)$ eine $\mathsf{SPJRU}[\mathbf{S}]$-Anfrage der Sorte Σ (bzw. eine $\mathsf{SPCU}[\mathbf{S}]$-Anfrage der Stelligkeit k).

(b) Zur Definition der Klasse der Anfragen der *relationalen Algebra* über $\mathbf{S}$ in der benannten (bzw. der unbenannten) Perspektive werden die Definitionen von $\mathsf{SPJRU}[\mathbf{S}]$ (bzw. $\mathsf{SPCU}[\mathbf{S}]$) um die folgende Regel erweitert:

- Sind Q und P zwei Anfragen der relationalen Algebra derselben Sorte Σ (bzw. derselben Stelligkeit k), so ist $(Q - P)$ eine Anfrage relationalen Algebra der Sorte Σ (bzw. der Stelligkeit k).

Die *Semantik* $[\![Q]\!]$ solcher Anfragen Q ist induktiv auf die offensichtliche Art definiert.
Mit SPJRU (bzw. SPCU) bezeichnen wir die Klasse aller $\mathsf{SPJRU}[\mathbf{S}]$-Anfragen (bzw. $\mathsf{SPCU}[\mathbf{S}]$-Anfragen) für alle Datenbankschemata $\mathbf{S}$.

Folie 211

Beispiele

- In welchen Kinos wird "Alien" oder "Brazil" gespielt?

$$\pi_{Kino}\Big(\sigma_{Titel=\text{"Alien"}}\big(Programm\big) \ \cup\ \sigma_{Titel=\text{"Brazil"}}\big(Programm\big) \Big)$$

- Welche Regisseure haben noch nie mit "Matt Damon" gearbeitet?

$$\pi_{Regie}\big(Filme\big) \ - \ \pi_{Regie}\Big(\sigma_{Schauspieler=\text{"Matt Damon"}}\big(Filme\big)\Big)$$

- Welche derzeit laufenden Filme haben nur Schauspieler, die schon mal in einem Film von "James Cameron" mitgespielt haben?

$$\pi_{Titel}\big(Programm\big) - \pi_{Titel}\Big(\underbrace{Filme \bowtie \underbrace{\Big(\pi_{\substack{Schau-\\spieler}}\big(Filme\big) \ - \ \pi_{\substack{Schau-\\spieler}}\big(\sigma_{\substack{Regie=\text{"James}\\ \text{Cameron"}}}(Filme)\big)\Big)}_{\textit{Schauspieler, die noch nie mit James Cameron gearbeitet haben}}}_{\textit{Filme mit mind. einem Schauspieler, der noch nie mit James Cameron gearbeitet hat}}\Big)$$

Folie 212

Ausdrucksstärke

Proposition 6.2.

(a) Jede SPCU*-Anfrage und jede* SPJRU*-Anfrage ist monoton.*

(b) Für jede Datenbank $\mathbf{I}$ *und jede Anfrage* Q *der relationalen Algebra gilt:*

$$\text{adom}\big([\![Q]\!](\mathbf{I})\big) \ \subseteq\ \text{adom}(Q,\mathbf{I}).$$

(c)

$$\begin{array}{ccccc} \mathsf{SPC} & < & \mathsf{SPCU} & < & \textit{relationale Algebra (unbenannte Perspektive)} \\ \equiv & & \equiv & & \equiv \\ \mathsf{SPJR} & < & \mathsf{SPJRU} & < & \textit{relationale Algebra (benannte Perspektive)} \end{array}$$

Beweis:
(a)+(b): Einfache Induktion über den Aufbau der Anfragen.
(c): Übung. □

Folie 213

Proposition 6.3. *(a) Benannte Perspektive:*
Keiner der Operatoren σ, π, $\cup$, $-$, $\bowtie$, δ ist redundant.

D.h.: Weglassen jedes einzelnen dieser Operatoren führt zu einer Algebra, die manche der in der relationalen Algebra ausdrückbaren Anfragefunktionen nicht beschreiben kann.

(b) Unbenannte Perspektive:

(i) Der Operator σ kann durch Kombination der Operatoren π, $-$, $\times$ ausgedrückt werden.

Beachte: Um dies zu zeigen, muss man nutzen, dass bei der Projektion $\pi_{j_1,\dots,j_k}$ die Indizes j_i nicht paarweise verschieden sein müssen.

(ii) Keiner der Operatoren π, $\cup$, $-$, $\times$ ist redundant.

Beweis: Übung.

Folie 214

Theta-Join und Semijoin

Eine *positive konjunktive Join-Bedingung* ist ein Ausdruck θ der Form $\bigwedge_{\ell=1}^{m} x_{i_\ell}{=}y_{j_\ell}$, für natürliche Zahlen $m \geqslant 0$ und $i_1,\dots,i_m,j_1,\dots,j_m \geqslant 1$.

Zwei Tupel $\overline{a} = (a_1,\dots,a_r) \in \mathbf{dom}^r$ und $\overline{b} = (b_1,\dots,b_s) \in \mathbf{dom}^s$ mit $r \geqslant \max\{i_1,\dots,i_m\}$ und $s \geqslant \max\{j_1,\dots,j_m\}$ *erfüllen* θ

$$(\text{kurz: } (\overline{a},\overline{b}) \models \theta, \text{ bzw. } \theta(\overline{a},\overline{b})),$$

falls für alle $\ell \in \{1,\dots,m\}$ gilt: $a_{i_\ell} = b_{j_\ell}$.

In der relationalen Algebra (unbenannte Perspektive) lassen sich u.a. die folgenden Operationen ausdrücken:

- *Theta-Join* $\bowtie_\theta$, wobei θ eine positive konjunktive Join-Bedingung ist.
  Semantik: $I \bowtie_\theta J \ := \ \big\{ (\overline{a},\overline{b}) \,:\, \overline{a} \in I,\ \overline{b} \in J,\ \textit{so dass } (\overline{a},\overline{b}) \models \theta \big\}$
- *Semijoin* $\ltimes_\theta$, wobei θ eine positive konjunktive Join-Bedingung ist.
  Semantik: $I \ltimes_\theta J \ := \ \big\{ \overline{a} \in I \,:\, \textit{ex. } \overline{b} \in J,\ \textit{so dass } (\overline{a},\overline{b}) \models \theta \big\}$

6.2 Anfrageauswertung und Heuristische Optimierung

Folie 215

Anfrageauswertung und Heuristische Optimierung

... hat viele Aspekte:

- Speicher- und Indexstrukturen
- Betriebssystem
- Seitenersetzungsstrategien
- Statistische Eigenschaften der Daten
- Statistische Informationen über Anfragen
- Implementierung der einzelnen Operatoren
- Ausdrucksstärke der Anfragesprache

Hier: Überblick und einige Teilaspekte.
Details: In den Vorlesungen DBS I und DBS II von Prof. Freytag

Folie 216

Anfrageauswertung allgemein

Vorbemerkung:

- Datenbanken sind *sehr* groß
- werden auf Sekundärspeicher (Festplatte) gespeichert
- *Aufwand wird dominiert durch die Anzahl der Plattenzugriffe* ("Seitenzugriffe")
  Denn: In derselben Zeit, die für einen "Random Access" auf der Festplatte benötigt wird, können zigtausende Operationen im Hauptspeicher durchgeführt werden.

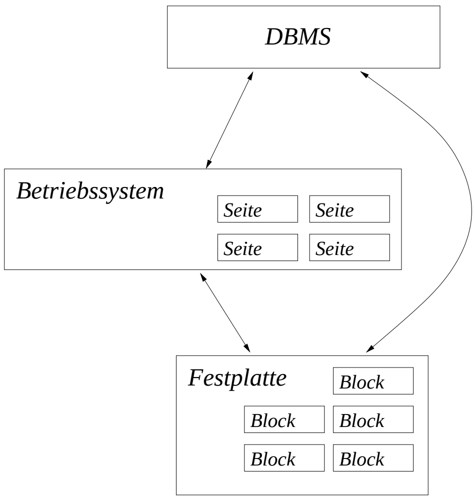
Allgemeines Vorgehen:

- durch Erzeugen, Filtern, Manipulieren und Kombinieren von *Tupelströmen*
- dabei evtl. Verwendung von Indexstrukturen ("Wegweiser"), Hashing und Sortier-Schritten

- wünschenswert: möglichst wenig auf Festplatte zwischenspeichern
- Operationen: Operationen der relationalen Algebra, Sortieren, Duplikatelimination, ...

Folie 217

Verwaltung des Sekundärspeichers



Hier (der Einfachheit halber): 1 Plattenzugriff $\widehat{=}$ Lesen eines Blocks bzw. einer Seite

Folie 218

Wichtige Parameter einer Datenbankrelation I

- n_I: Anzahl der Tupel in Relation I
- s_I: (mittlere) Größe eines Tupels aus I
- f_I: Blockungsfaktor ("Wie viele Tupel aus I passen in einen Block?")

$$f_I \;\approx\; \frac{\text{Blockgröße}}{s_I}$$

- b_I: Anzahl der Blöcke (Seiten) der Festplatte, die Tupel aus I beinhalten

$$b_I \;\approx\; \frac{n_I}{f_I}$$

Hier (der Einfachheit halber):
Lesen eines Blocks (bzw. einer Seite) $\widehat{=}$ 1 Zugriff auf Platte

Folie 219

Operationen der relationalen Algebra

Selektion $\sigma_F(I)$:

- Selektion meist als Filter auf einem Tupelstrom:
  $\mathcal{O}(b_I)$ Zugriffe auf Festplatte; $\mathcal{O}(n_I)$ Schritte insgesamt
- evtl. Verwendung eines Index;
  dann schneller, falls nur sehr wenige Tupel im Ergebnis

Folie 220

Projektion $\pi_{j_1,\dots,j_k}(I)$:

- 2 Komponenten:
  - Ändern der einzelnen Tupel (Auswahl und Reihenfolge von Spalten)
  - Duplikatelimination
- Tupeländerung: als Filter auf einem Tupelstrom
  $\mathcal{O}(b_I)$ Zugriffe auf Festplatte; $\mathcal{O}(n_I)$ Schritte insgesamt
  Dabei können Duplikate "entstehen"
- Duplikatelimination:
  - in SQL i.d.R. nicht verlangt (außer bei `SELECT DISTINCT`)
  - sind die Tupel sortiert, so können Duplikate durch einen Scan leicht erkannt werden
  - Sortieren: durch Merge-Sort möglich mit $\mathcal{O}(b_I \cdot \log b_I)$ Plattenzugriffen und $\mathcal{O}(n_I \cdot \log n_I)$ Schritten insgesamt
  - Alternative: Hashing
    * Abbilden der Tupel durch eine Hash-Funktion
    * $\rightsquigarrow$ Duplikate werden auf denselben Wert abgebildet und dadurch erkannt
    * bei idealer Hash-Funktion: lineare Zeit

Folie 221

Binäre Operationen auf zwei Relationen I und J: $\cup,\ -,\ \times,\ \bowtie_\theta,\ \ltimes_\theta$

- *Nested-Loops-Methode:* (Schleifeniteration)
  für jedes Tupel $t \in I$ (bzw. jede Seite) wird die gesamte Relation J durchlaufen
  $\mathcal{O}(b_I \cdot b_J)$ Plattenzugriffe; $\mathcal{O}(n_I \cdot n_J)$ Schritte insgesamt
- *Merge-Methode:* (weniger sinnvoll für $\times$)
  I und J sortiert $\leadsto$ schrittweise in der vorgegebenen Tupelreihenfolge durchlaufen;
  Für $\ltimes_\theta$: $\mathcal{O}(b_I + b_J)$ Plattenzugriffe; $\mathcal{O}(n_I + n_J)$ Gesamtschritte

  Evtl. vorher nötig: Sortieren von I und/oder J (durch Merge-Sort)
  $\mathcal{O}(b_I \cdot \log b_I)$ und/oder $\mathcal{O}(b_J \cdot \log b_J)$ Plattenzugriffe;
  $\mathcal{O}(n_I \cdot \log n_I)$ und/oder $\mathcal{O}(n_J \cdot \log n_J)$ Gesamtschritte
- *Hash-Methode:* (weniger sinnvoll für $\times$)
  die kleinere der beiden Relationen in Hash-Tabelle;
  Tupel der zweiten Relation finden ihren Vergleichspartner mittels Hash-Funktion;
  bei idealer Hash-Funktion: Aufwand $\mathcal{O}(n_I + n_J)$

Folie 222

Beispiel für Merge-Technik

Berechne $I \bowtie_\theta J$ *für Join-Bedingung* $\theta := x_1{=}y_4 \wedge x_2{=}y_1$

1. Sortiere I lexikographisch nach "1-te Spalte; 2-te Spalte"
2. Sortiere J lexikographisch nach "4-te Spalte; 1-te Spalte"
3. Seien t und s die ersten Tupel von I und J
4. Falls $(t_1, t_2) < (s_4, s_1)$, so lies nächstes Tupel t aus I.
5. Falls $(t_1, t_2) > (s_4, s_1)$, so lies nächstes Tupel s aus J.
6. Falls $(t_1, t_2) = (s_4, s_1)$, so gib die Tupel (t, s) und (t, s') für alle Nachfolger s' von s in J mit $(s'_4, s'_1) = (s_4, s_1)$ aus
7. Lies nächstes Tupel t aus I und gehe zu Zeile 4.

Aufwand für Zeilen 3–7:

- falls alle Tupel den gleichen Wert in den Spalten 1,2 bzw. 4,1 haben: ca. $n_I \cdot n_J$ Gesamtschritte

- falls alle Tupel aus J in (s_4, s_1) unterschiedliche Werte haben: ca. $n_I + n_J$ Gesamtschritte :-)

- bei Semijoin $\ltimes_\theta$ statt Theta-Join $\bowtie_\theta$ reichen immer $n_I + n_J$ Gesamtschritte

Folie 223

Anfrageauswertung

Proposition 6.4. *Das Auswertungsproblem für die relationale Algebra lässt sich in Zeit $(k{+}n)^{\mathcal{O}(k)}$ lösen.*

Beweis:
Zeige per Induktion nach dem Aufbau von Anfragen der relationalen Algebra, dass für jede Anfrage Q der Länge k und jede Datenbank $\mathbf{I}$ der Größe n gilt:

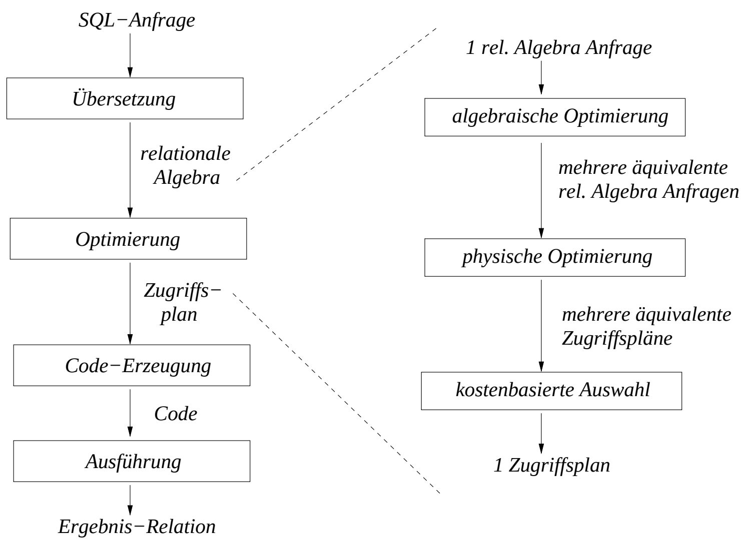
(1) $\|\,[\![Q]\!](\mathbf{I})\,\| \ \leqslant\ (k{+}n)^k$

(2) Q kann auf $\mathbf{I}$ in $\mathcal{O}\big((k{+}n)^{2k}\big)$ Elementarschritten ausgewertet werden.

Details: *Übung.*

Folie 224

Anfragebearbeitung durch ein DBMS



Folie 225

Ziel der Optimierung

- möglichst schnelle Auswertung der Anfrage
- möglichst wenige Zugriffe auf Festplatte

Grundregeln:

(1) Selektionen so früh wie möglich

(2) auch Projektionen früh, aber evtl. Duplikatelimination vermeiden

(3) Basisoperationen zusammenfassen und wenn möglich ohne Zwischenspeicherung realisieren
(Bsp: $\bowtie_\theta$ besser als $\times$; $\ltimes_\theta$ besser als $\bowtie_\theta$)

(4) Redundante Operationen oder leere Zwischenrelationen entfernen

(5) Zusammenfassung gleicher Teilausdrücke:
Wiederverwendung von Zwischenergebnissen

Folie 226

Anfrageauswertung an einem Beispiel

Anfrage:
Welche Kinos (Name + Adresse) spielen einen Film von "James Cameron"?
In SQL:

```
SELECT Kinos.Name, Kinos.Adresse
FROM Kinos, Filme, Programm
WHERE Kinos.Name = Programm.Kino and
      Programm.Titel = Filme.Titel and
      Filme.Regie = ‘‘James Cameron’’
```

Direkte Übersetzung in relationale Algebra:

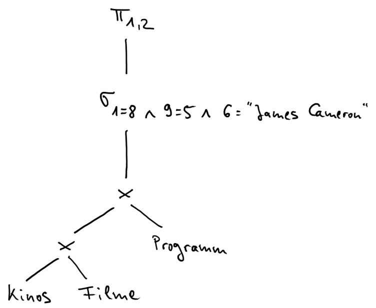
$$\pi_{1,2}\Big(\sigma_{\substack{1=8\wedge 9=5\wedge\\ 6=\text{“James Cameron”}}}\big(\mathit{Kinos}\times \mathit{Filme}\times \mathit{Programm}\big)\Big)$$

Folie 227

Original-Anfrage

$$\pi_{1,2}\Big(\sigma_{\substack{1=8\wedge 9=5\wedge\\ 6=\text{“James Cameron”}}}\big(\mathit{Kinos}\times \mathit{Filme}\times \mathit{Programm}\big)\Big)$$

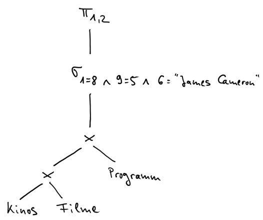
dargestellt als Anfrage-Baum:



Folie 228

Anfrage auswerten auf folgender Beispiel-Datenbank:

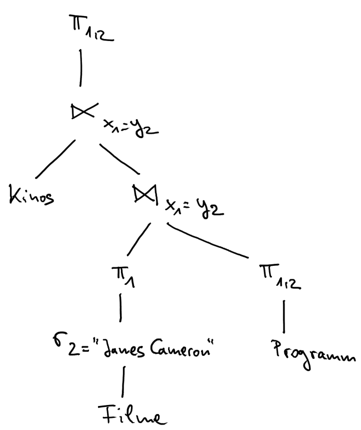
- *Filme*: 10.000 Tupel auf 200 Seiten (je 50 pro Seite); je 5 Tupel pro Film, 10 Filme von James Cameron
- *Programm*: 200 Tupel auf 4 Seiten (je 50 pro Seite); davon 3 Cameron-Filme in 4 Kinos
- *Kinos*: 100 Tupel auf 2 Seiten (je 50 pro Seite)

Anfrage-Baum: 

Direkte Auswertung dieser Anfrage führt zu über 10.000.000 Plattenzugriffen.

Folie 229

Viel besserer Plan:



Auswertung dieses Plans in unserer Beispiel-Datenbank führt zu weniger als 250 Plattenzugriffen.

Folie 230

Heuristische Optimierung

- Die heuristische Optimierung wendet allgemeine Regeln zur Umformung einer Anfrage der relationalen Algebra in eine äquivalente Anfrage an, die zu einem vermutlich effizienteren Auswertungsplan führt
- Grundregel: Selektionen so früh wie möglich
- Projektionen auch früh, aber evtl. Duplikatelimination vermeiden
- Anwendung von algebraischen Umformungsregeln
- Ziel: Operationen verschieben, um kleinere Zwischenergebnisse zu erhalten; wenn möglich Redundanzen erkennen

Folie 231

Einige algebraische Umformungsregeln:

(1) Kartesische Produkte und Joins sind kommutativ und assoziativ:
$Q_1 \bowtie_\theta Q_2 \quad \longleftrightarrow \quad Q_2 \bowtie_{\tilde{\theta}} Q_1$
$(Q_1 \bowtie_{\theta_1} Q_2) \bowtie_{\theta_2} Q_3 \quad \longleftrightarrow \quad Q_1 \bowtie_{\tilde{\theta}_1} (Q_2 \bowtie_{\tilde{\theta}_2} Q_3)$
($\tilde{\theta}$ entsteht aus θ durch "Zurückrechnen" der Spaltennummern)

(2) Ketten von Selektionen (bzw. Projektionen) zusammenfassen:
$\sigma_{F_1}(\sigma_{F_2}(Q)) \quad \longleftrightarrow \quad \sigma_{F_1 \wedge F_2}(Q) \quad \longleftrightarrow \quad \sigma_{F_2}(\sigma_{F_1}(Q))$
$\pi_X(\pi_Y(Q)) \quad \longleftrightarrow \quad \pi_{\tilde{X}}(Q)$
($\tilde{X}$ entsteht aus X durch "Zurückrechnen" der Spaltennummern)

(3) Vertauschen von Selektion und Join:
$\sigma_{F_1 \wedge F_2 \wedge F_3}(Q_1 \times Q_2) \quad \longleftrightarrow \quad \sigma_{F_1}(Q_1) \bowtie_{\theta_3} \sigma_{\tilde{F}_2}(Q_2)$,
wobei die Selektionsbed. F_1 (F_2) sich nur auf Spalten von Q_1 (Q_2) bezieht und F_3 Spalten von Q_1 mit Spalten von Q_2 vergleicht. $\tilde{F}_2$ und θ_3 entstehen aus F_2 und F_3 durch "Zurückrechnen" der Spaltennummern.

(4) Einführung von Semijoins, falls X nur Spalten von Q_1 beinhaltet:
$\pi_X(Q_1 \bowtie_\theta Q_2) \quad \longleftrightarrow \quad \pi_X(Q_1 \ltimes_\theta Q_2)$

Folie 232

(5) Vertauschen von Selektion und Vereinigung bzw. Differenz:
$\sigma_F(Q_1 \cup Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) \cup \sigma_F(Q_2)$
$\sigma_F(Q_1 - Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) - \sigma_F(Q_2)$

(6) Analog: Vertauschen von Projektion und Vereinigung bzw. Differenz.

(7) Vertauschen von Projektion und Selektion unter bestimmten Bedingungen

(8) Vertauschen von Projektion und Join unter bestimmten Bedingungen

(9) Löschen von Redundanzen:
$Q \cup Q \;\longrightarrow\; Q, \qquad Q \cap Q \;\longrightarrow\; Q, \qquad Q \bowtie Q \;\longrightarrow\; Q$

(10) Löschen leerer Zwischenergebnisse:
$Q - Q \;\longrightarrow\; \emptyset, \qquad Q \cap \emptyset \;\longrightarrow\; \emptyset, \qquad Q \cup \emptyset \;\longrightarrow\; Q,$
$Q \bowtie \emptyset \;\longrightarrow\; \emptyset,$
$Q - \emptyset \;\longrightarrow\; Q, \qquad \emptyset - Q \;\longrightarrow\; \emptyset$

(11) ... usw. ...

Folie 233

Wunschliste für bessere Optimierung:

- zum "Löschen leerer Zwischenergebnisse":
  Test, ob eine gegebene (Teil-)Anfrage Q nicht erfüllbar ist ("$Q \equiv \emptyset$")
- zum "Löschen von Redundanzen":
  Test, ob zwei (Teil-)Anfragen Q und P äquivalent sind ("$Q \equiv P$")

Wir kennen bereits:

- Algorithmen zum Lösen dieser Probleme für konjunktive Anfragen

Im nächsten Kapitel:

- Nicht-Entscheidbarkeit dieser Probleme für allgemeine Anfragen der relationalen Algebra

Folie 234

Vorgehensweise eines Optimierers in einem DBMS

- Erzeugung verschiedener rel. Algebra Ausdrücke: unter Verwendung heuristischer Optimierungsregeln
- Erzeugung von verschiedenen Auswertungsplänen: Anfrage-Bäume, erweitert um Informationen über zu verwendende Zugriffsstrukturen und Algorithmen für die einzelnen Operationen
- Abschätzung der Kosten für jeden erzeugten Auswertungsplan: unter Verwendung von statistischen Informationen über die Daten ($\leadsto$ kostenbasierte Optimierung)
- Auswahl des am günstigsten erscheinenden Plans

Generell:
Je häufiger die Anfrage ausgewertet werden soll, desto mehr Aufwand sollte für die Optimierung verwendet werden.

Folie 235

Join-Reihenfolge

- Joins sind kommutativ und assoziativ
  $\leadsto$ Änderung der Klammerung bzw. Reihenfolge von Join-Operationen ändert nicht das Ergebnis der Anfrage (modulo Ändern der Spalten-Reihenfolge)
- Aber: Die Größe der Zwischenergebnisse und somit der Auswertungs-Aufwand kann sich drastisch ändern.
- Unter Umständen lässt sich sogar die Anzahl der Joins verkleinern (das haben wir bereits bei der Minimierung von konjunktiven Anfragen gesehen)

- Klassische Vorgehensweise in DBMS: Nur Auswertungspläne betrachten, die Joins von links nach rechts klammern ("left-deep-trees")
  $\leadsto$ durch Umordnen sind immerhin alle Reihenfolgen möglich (immer noch exponentiell viele Möglichkeiten)
- (Es ist bekannt, dass diese Einschränkung nicht immer sinnvoll und nötig ist)

Jetzt: Heuristische Join-Optimierung bzgl. left-deep-trees

Folie 236

Beispiele

Beispiel 6.5. R : 2-stellige Relation mit Attributen A, B und 1.000 Tupeln
S : 2-stellige Relation mit Attributen C, D und 10.000 Tupeln
T : 2-stellige Relation mit Attributen B, C und 100 Tupeln
Pro Tupel $(b, c) \in T$ gibt es <u>ein</u> Tupel $(\cdot, b) \in R$ und <u>ein</u> Tupel $(c, \cdot) \in S$.

Anfrage: $Ans(x) \leftarrow R(x, x_1),\ S(x_2, x_3),\ T(x_1, x_2)$

Aufwand bei Links-nach-rechts-Auswertung:
10.000.000 Tupel nach erstem Join, 100 Tupel nach zweitem Join

Andere Join-Reihenfolge: $Ans(x) \leftarrow R(x, x_1),\ T(x_1, x_2),\ S(x_2, x_3)$

Aufwand bei Links-nach-rechts-Auswertung:
100 Tupel nach erstem Join, 100 Tupel nach zweitem Join

Folie 237

Beispiel 6.6. R : 2-stellige Relation mit Attributen A, B und 1.000 Tupeln
S : 2-stellige Relation mit Attributen C, D und 10.000 Tupeln
T : 2-stellige Relation mit Attributen B, C und 100 Tupeln
Pro Tupel $(b, c) \in T$ gibt es <u>ein</u> Tupel $(\cdot, b) \in R$ und <u>ein</u> Tupel $(c, \cdot) \in S$.
Für die Konstante d gibt es nur 1 Tupel $(\cdot, d)$ in S.

Anfrage: $Ans(x) \leftarrow R(x, x_1),\ T(x_1, x_2),\ S(x_2, d)$

Aufwand bei Links-nach-rechts-Auswertung:
100 Tupel nach erstem Join, max. 1 Tupel nach zweitem Join

Andere Join-Reihenfolge: $Ans(x) \leftarrow S(x_2, d),\ T(x_1, x_2),\ R(x, x_1)$

Aufwand bei Links-nach-rechts-Auswertung:
max. 1 Tupel nach erstem Join, max. 1 Tupel nach zweitem Join

Folie 238

Noch ein Beispiel: a und b seien jetzt Konstanten, d.h. $a, b \in \mathbf{dom}$

- Auswertung von links nach rechts
- Anfrage: $Ans(z) \leftarrow R(v,w,y),\ v \leqslant x,\ S(x,y,z),\ P(a,v),\ Q(b,w,x)$
- Besserer Auswertungsplan:
  $Ans(z) \leftarrow P(a,v),\ Q(b,w,x),\ R(v,w,y),\ S(x,y,z),\ v \leqslant x,$
- Denn:
  - wahrscheinlich wenige Tupel der Form $(a,\cdot)$ in P
  - wahrscheinlich wenige Tupel der Form $(b,\cdot,\cdot)$ in Q
  - wahrscheinlich wenige Tupel, die $P(a,v),\ Q(b,w,x),\ R(v,w,y)$ erfüllen

Heuristik ("Sideways-Information-Passing", kurz: SIP):

- Relations-Atome mit Konstanten zuerst auswerten
- Wenn möglich, Relations-Atome erst dann, wenn weiter links schon eine ihrer Variablen steht.
- Vergleichsoperatoren ($\leqslant$, $<$) möglichst erst dann verwenden, wenn beide Variablen schon verwendet wurden.

Folie 239

SIP-Graph und SIP-Strategie

Definitionen:

- Regel $Ans(u) \leftarrow R_1(u_1), \ldots, R_k(u_k), E_1, \ldots, E_\ell, C_1, \ldots, C_m$
  wobei E_i Vergleich mit $=$
  und C_i Vergleich mit $<$ oder $\leqslant$ (dafür sei $<$ eine lineare Ordnung auf **dom**).
- *SIP-Graph*
  - Knotenmenge: Rel.-Atome $R_1(u_1), \ldots, R_k(u_k)$ und "$=$"-Atome $E_1, \ldots, E_\ell$
  - Kante zwischen zwei Knoten, falls diese (mind.) eine Variable gemeinsam haben

  – Knoten ist *markiert*, falls er mind. eine Konstante enthält

- Falls der SIP-Graph zusammenhängend ist, so ist eine *SIP-Strategie* eine Anordnung $A_1, \ldots, A_{k+\ell+m}$ der Atome, so dass für jedes $j > 1$ gilt:
  - A_j ist ein *markierter* Knoten des SIP-Graphen, oder
  - A_j ist ein Knoten des SIP-Graphen und es gibt ein $j' < j$, so dass es im SIP-Graph eine Kante zwischen A_j und $A_{j'}$ gibt, oder
  - A_j ist ein C_i und für jede Variable x in C_i gibt es $j' < j$, so dass $A_{j'}$ ein Knoten des SIP-Graphen ist, in dem x vorkommt
- Außerdem: Falls ex. $R_i(u_i)$ od. E_i mit mind. einer Konstanten, so ist A_1 ein solches Atom
- Falls der SIP-Graph nicht zusammenhängend ist: SIP-Strategie für jede Zusammenhangskomponente.

Folie 240

Beispiele: a und b seien Konstanten, d.h. $a, b \in \mathbf{dom}$.

- $Ans(z) \leftarrow P(a,v),\ Q(b,w,x),\ R(v,w,y),\ S(x,y,z),\ v \leqslant x$ ist in der Reihenfolge einer SIP-Strategie
- $Ans(z) \leftarrow R(v,w,y),\ v \leqslant x,\ S(x,y,z),\ P(a,v),\ Q(b,w,x)$ ist nicht in der Reihenfolge einer SIP-Strategie

Bemerkungen:

- Zu jeder regelbasierten konjunktiven Anfrage (evtl. mit $=$ oder $\leqslant$; dann aber bereichsbeschränkt) gibt es eine SIP-Strategie.
- Eine SIP-Strategie für eine gegebene Anfrage lässt sich in Zeit polynomiell in der Größe der Anfrage erzeugen.
- Wird eine Variable weiter rechts nicht mehr benötigt, so kann sie beim Auswerten der Anfrage aus dem Zwischenergebnis "heraus projiziert" werden.

Kapitel 7

Relationenkalkül

7.1 CALC$_{nat}$, CALC$_{adom}$ und CALC$_{di}$

Folie 241

Einleitung

Algebraisch	**... äquivalent dazu ...**	**Logik-basiert**
SPC-Algebra		konjunktiver Kalkül
Boolesche Semijoin-Anfagen		GF(CQ)-Sätze
relationale Algebra		*Relationenkalkül*

Der *konjunktive Kalkül* basiert auf einem *Fragment der Logik erster Stufe*.

Der *Relationenkalkül* basiert auf der vollen *Logik erster Stufe* (kurz: FO).
("FO" steht für "*first-order logic*")

Folie 242

Die Logik erster Stufe — FO[S]

Definition 7.1. Sei $\mathbf{S}$ ein Datenbankschema.
Die Menge $\mathsf{FO}[\mathbf{S}]$ aller Formeln der *Logik erster Stufe* über $\mathbf{S}$ ist induktiv wie folgt definiert:

(A) *"Relations-Atome":* $R(v_1, \ldots, v_r)$ gehört zu $\mathsf{FO}[\mathbf{S}]$,
für alle $R \in \mathbf{S}$, $r := \mathrm{ar}(R)$ und $v_1, \ldots, v_r \in \mathrm{var} \cup \mathbf{dom}$.

(G) *"Gleichheits-Atome":* $v_1{=}v_2$ gehört zu $\mathsf{FO}[\mathbf{S}]$, für alle
$v_1, v_2 \in \mathrm{var} \cup \mathbf{dom}$.

(BK) *"Boolesche Kombinationen":* Falls $\varphi_1 \in \mathsf{FO}[\mathbf{S}]$ und $\varphi_2 \in \mathsf{FO}[\mathbf{S}]$, so gehört auch jede der folgenden fünf Formeln zu $\mathsf{FO}[\mathbf{S}]$:
$\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$ und $(\varphi_1 \leftrightarrow \varphi_2)$.

(Q) *"Quantoren":* Ist $\varphi_1 \in \mathsf{FO}[\mathbf{S}]$ und ist $x \in \text{var}$, so gehören auch die beiden folgenden Formeln zu $\mathsf{FO}[\mathbf{S}]$: $\exists x\, \varphi_1$ und $\forall x\, \varphi_1$.

Bemerkung: Benutzen wir die Notation aus der Vorlesung *„Logik in der Informatik"*, so ist $\mathsf{FO}[\mathbf{S}]$ genau die Menge aller Formeln aus $\mathsf{FO}[\sigma]$, für die Signatur $\sigma := \mathbf{S} \cup \mathbf{dom}$, wobei jedes Element aus **dom** als Konstantensymbol aufgefasst wird, das stets „mit sich selbst" interpretiert wird).

Folie 243

Notationen und Anmerkungen

- Manchmal werden wir Formeln der Form
  - $(\varphi_1 \vee \varphi_2)$ als Abkürzung für $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$,
  - $(\varphi_1 \rightarrow \varphi_2)$ als Abkürzung für $(\neg\varphi_1 \ \vee \ \varphi_2)$
  - $(\varphi_1 \leftrightarrow \varphi_2)$ als Abkürzung für $(\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \ \wedge \ \neg\varphi_2)$
  - $\forall x\, \varphi_1$ als Abkürzung für $\neg\, \exists x\, \neg\varphi_1$

  auffassen.

- $\text{adom}(\varphi)$ bezeichnet die Menge aller Konstanten (also Elemente aus **dom**), die in φ vorkommen.
  $\text{Var}(\varphi)$ bezeichnet die Menge aller Variablen (also Elemente aus var, die in φ vorkommen.
  $\text{frei}(\varphi)$ bezeichnet die Menge aller Variablen, die *frei* in φ vorkommen. D.h.
  - $\text{frei}\big(R(v_1, \ldots, v_r)\big) \ := \ \{v_1, \ldots, v_r\} \cap \text{var}$
  - $\text{frei}\big(v_1{=}v_2\big) \ := \ \{v_1, v_2\} \cap \text{var}$
  - $\text{frei}\big(\neg\varphi_1\big) \ := \ \text{frei}\big(\varphi_1\big)$
  - $\text{frei}\big((\varphi_1 * \varphi_2)\big) \ := \ \text{frei}\big(\varphi_1\big) \cup \text{frei}\big(\varphi_2\big)$ (für alle $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$)
  - $\text{frei}\big(\exists x\, \varphi_1\big) \ := \ \text{frei}\big(\forall x\, \varphi_1\big) \ := \ \text{frei}\big(\varphi_1\big) \setminus \{x\}$

- Eine *Belegung für* φ ist eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{dom}$.
- Sei $\mathbf{d} \subseteq \mathbf{dom}$. Eine *Belegung für* φ *in* $\mathbf{d}$ ist eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{d}$.

Folie 244

Natürliche Semantik von $\mathsf{FO}[\mathbf{S}]$ und CALC

Wir werden verschiedene Varianten der Semantik von $\mathsf{FO}[\mathbf{S}]$ betrachten. Hier zunächst die *natürliche Semantik*:

Definition 7.2 (natürliche Semantik von $\mathsf{FO}[\mathbf{S}]$)**.**

- *Zur Erinnerung:* Einer Datenbank $\mathbf{I}$ vom Schema $\mathbf{S}$ ordnen wir die logische Struktur $\mathcal{A}_{\mathbf{I}} := \big(\mathbf{dom},\ \big(\mathbf{I}(R)\big)_{R\in\mathbf{S}},\ (c)_{c\in\mathbf{dom}}\big)$ zu.
- Ist $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$ und β eine Belegung für φ (also eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{dom}$), so sagen wir "$\mathbf{I}$ *erfüllt* φ *unter* β" und schreiben $\mathbf{I} \models \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models \varphi$, um auszudrücken, dass $(\mathcal{A}_{\mathbf{I}}, \beta) \models \varphi$.

Definition 7.3 (Relationenkalkül CALC)**.** Sei $\mathbf{S}$ ein Datenbankschema. Eine Anfrage des *Relationenkalküls* CALC[$\mathbf{S}$] ist von der Form
$\{\ (e_1, \dots, e_r)\ :\ \varphi\ \}$,
wobei $\varphi \in \mathsf{FO}[\mathbf{S}]$, $r \geqslant 0$ und $(e_1, \dots, e_r)$ ein freies Tupel ist (d.h. $e_1, \dots, e_r \in \text{var} \cup \mathbf{dom}$), so dass $\text{frei}(\varphi) = \{e_1, \dots, e_r\} \cap \text{var}$.
Wir setzen $\text{adom}(Q) := \text{adom}(\varphi) \cup (\{e_1, \dots, e_r\} \cap \mathbf{dom})$.

Semantik: Einer CALC[$\mathbf{S}$]-Anfrage Q der obigen Form ordnen wir die folgende "Anfragefunktion" $[\![Q]\!]_{nat}$ zu (f.a. $\mathbf{I} \in \mathit{inst}(\mathbf{S})$):

$$[\![Q]\!]_{nat}(\mathbf{I}) \ :=\ \big\{\ \beta\big((e_1, \dots, e_r)\big)\ :\ \ \beta \textit{ ist eine Belegung für } \varphi \textit{ mit } \mathbf{I} \models \varphi[\beta]\ \big\}$$

Folie 245

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{\ (x_{\mathit{Titel}})\ :\ \exists x_{\mathit{Regie}}\ \Big(\mathit{Filme}(x_{\mathit{Titel}}, x_{\mathit{Regie}}, \text{"George Clooney"}) \land \neg\mathit{Filme}(x_{\mathit{Titel}}, \text{"George Clooney"}, \text{"George Clooney"})\ \Big)\ \Big\}$$

- Welche Filme laufen nur zu 1 Uhrzeit?

$$\Big\{ (x_{Titel}) \ : \ \exists x_{Kino}\, \exists x_{Zeit} \Big(\mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \forall y_{Kino} \forall y_{Zeit} \big(\mathit{Programm}(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit}{=}x_{Zeit} \ \big) \Big) \Big\}$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{ (x_T) \ : \ \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \ \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

Frage: Was ist mit ... ?

$$\Big\{ (x_T) \ : \ \forall y_S \Big(\exists x_R\, \mathit{Filme}(x_T, x_R, y_S) \ \rightarrow \ \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

Folie 246

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \big\{ (x_{Schausp}) \ : \ \neg \mathit{Filme}(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \big\}$

(2) $Q_2 := \big\{ (x_S, y_T) \ : \ \mathit{Filme}(\text{“Alien”}, \text{“Ridley Scott”}, x_S) \vee \mathit{Filme}(y_T, \text{“George Clooney”}, \text{“George Clooney”}) \big\}$

(3) $Q_3 :=$

$$\Big\{ (x_T) \ : \ \forall y_S \Big(\exists x_R\, \mathit{Filme}(x_T, x_R, y_S) \ \rightarrow \ \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

Auswertung mit $[\![\cdot]\!]_{nat}$***:*** *Q_1 liefert alle (unendlich vielen) $a \in$* **dom**, *die* nicht *als Schauspieler im Film "Alien" mitgespielt haben.*
Q_2 liefert alle (unendlich vielen) Tupel der Form (a, b), für die a ein Schaupieler in "Alien" und b ein beliebiges Element in **dom** *ist oder b ein Film von und mit "George Clooney" und a ein beliebiges Element aus* **dom**.
Q_3 liefert alle Filme, die nur solche Schauspieler haben, die schon mal mit "Stephen Spielberg" gearbeitet haben, aber auch ***alle Elemente aus* dom*, die nicht Titel eines Films*** *sind.*

Problem: Gemäß der bisherigen Definition der Semantik liefern die Anfragen Q_1, Q_2 und Q_3 stets "unendliche Ergebnisse" die gemäß unserer Definition *keine Datenbank-Relationen* sind (da die stets endlich sein müssen). Solche Anfrage heißen *unsicher.*

Folie 247

Eine weitere problematische Anfrage

$Q_4 := \big\{ (x) \ : \ \forall y \, R(x,y) \big\}$

Q_4 liefert als Ergebnis stets die leere Relation, d.h.
$[\![Q_4]\!]_{nat}(\mathbf{I}) = \emptyset$, für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$.

Dies ist unschön.

Daher (und weil manche Anfragen unsicher sind in der nat. Semantik), betrachten wir im Folgenden andere Varianten der Semantik.

Folie 248

Relativierte Semantik von $\mathsf{FO}[\mathbf{S}]$ und CALC$[\mathbf{S}]$

Definition 7.4. Sei $\mathbf{S}$ ein Datenbankschema.

(a) Eine *relativierte Datenbank* über $\mathbf{S}$ ist ein Paar $(\mathbf{d}, \mathbf{I})$, wobei $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$ ist und $\mathbf{d}$ eine Menge, so dass $\text{adom}(\mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$.

(b) Einer relativierten Datenbank $(\mathbf{d}, \mathbf{I})$ über $\mathbf{S}$ ordnen wir die logische Struktur $\mathcal{A}_{(\mathbf{d},\mathbf{I})} := \big(\mathbf{d}, \ \big(\mathbf{I}(R)\big)_{R \in \mathbf{S}}, \ (c)_{c \in \mathbf{d}}\big)$ zu.

(c) Eine $\mathsf{FO}[\mathbf{S}]$-Formel φ heißt *interpretierbar über* $(\mathbf{d}, \mathbf{I})$, falls $\text{adom}(\varphi) \subseteq \mathbf{d}$.
Eine CALC$[\mathbf{S}]$-Anfrage Q heißt *interpretierbar über* $(\mathbf{d}, \mathbf{I})$, falls $\text{adom}(Q) \subseteq \mathbf{d}$.

(d) Ist $(\mathbf{d}, \mathbf{I})$ eine relativierte Datenbank über $\mathbf{S}$, ist φ eine $\mathsf{FO}[\mathbf{S}]$-Formel, die interpretierbar ist über $(\mathbf{d}, \mathbf{I})$, und ist β eine Belegung für φ in $\mathbf{d}$ (also eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{d}$), so sagen wir "$\mathbf{I}$ *erfüllt* φ *unter* β *relativ zu* $\mathbf{d}$" und schreiben $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models_{\mathbf{d}} \varphi$, um auszudrücken, dass $(\mathcal{A}_{(\mathbf{d},\mathbf{I})}, \beta) \models \varphi$.

(e) Ist Q eine CALC$[\mathbf{S}]$-Anfrage der Form $\{(e_1, \ldots, e_r) \ : \ \varphi\}$ und $(\mathbf{d}, \mathbf{I})$ eine relativierte Datenbank über $\mathbf{S}$, über der Q interpretierbar ist, so ist

$$[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) \ := \ \Big\{ \ \beta\big((e_1, \ldots, e_r)\big) \ : \ \ \beta \textit{ ist eine Belegung für } \varphi \textit{ über } \mathbf{d} \textit{ mit } \mathbf{I} \models_{\mathbf{d}} \varphi[\beta] \ \Big\}$$

("Relativiert" soll andeuten, dass Quantifizierung relativiert wird auf Elemente aus $\mathbf{d}$*.)*

Folie 249

Ein Beispiel

Beispiel 7.5.

- $\mathbf{S}$ enthalte ein 1-stelliges Relationssymbol R
- $\mathbf{I}$ sei die Datenbank mit
  $\mathbf{I}(R) = \{(a), (b), (c)\}$ und $\mathbf{I}(S) = \emptyset$ für alle $S \in \mathbf{S} \setminus \{R\}$.
- Q sei die CALC[$\mathbf{S}$]-Anfrage

$$Q := \left\{ \ (x) \ : \ \Big(\ R(x) \ \land \ \exists y \left(\neg R(y) \ \land \ \forall z \left(R(z) \lor z{=}y\right)\right) \ \Big) \ \right\}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) = \emptyset$
  - $[\![Q]\!]_{\{a,b,c,d\}}(\mathbf{I}) = \{(a), (b), (c)\}$
  - $[\![Q]\!]_{\{a,b,c,d,e\}}(\mathbf{I}) = \emptyset$

Folie 250

Active Domain Semantik von $\mathsf{FO}[\mathbf{S}]$ und CALC[$\mathbf{S}$]

Definition 7.6.

(a) Ist $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und $\varphi \in \mathsf{FO}[\mathbf{S}]$, so ist $\operatorname{adom}(\varphi, \mathbf{I}) := \operatorname{adom}(\varphi) \cup \operatorname{adom}(\mathbf{I})$.
Ist $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und $Q \in \text{CALC}[\mathbf{S}]$, so ist
$\operatorname{adom}(Q, \mathbf{I}) := \operatorname{adom}(Q) \cup \operatorname{adom}(\mathbf{I})$.

(b) Ist $\varphi \in \mathsf{FO}[\mathbf{S}]$, $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$ und β eine Belegung für φ in $\operatorname{adom}(\varphi, \mathbf{I})$ (also eine Abbildung $\beta : \operatorname{frei}(\varphi) \to \operatorname{adom}(\varphi, \mathbf{I})$), so sagen wir "**I** *erfüllt φ unter β in der Active Domain Semantik*" und schreiben $\mathbf{I} \models_{adom} \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models_{adom} \varphi$, um auszudrücken, dass $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ *für* $\mathbf{d} := \operatorname{adom}(\varphi, \mathbf{I})$ gilt.

(c) Ist Q eine CALC[$\mathbf{S}$]-Anfrage der Form $\{(e_1, \ldots, e_r) \ : \ \varphi\}$, so definiert Q in der Active Domain Semantik die folgende Anfragefunktion

$$[\![Q]\!]_{adom}(\mathbf{I}) \ := \ [\![Q]\!]_{\operatorname{adom}(Q,\mathbf{I})}(\mathbf{I})$$

Folie 251

Bemerkungen zur natürlichen Semantik und zur Active Domain Semantik

- Die Semantik $[\![\cdot]\!]_{nat}$ ist vom Standpunkt der Logik aus "natürlich"; vom Standpunkt des Datenbanknutzers aber eher unnatürlich, da manche Anfragen unendliche Ergebnisse liefern (d.h. unsicher sind).
- Die Active Domain Semantik $[\![\cdot]\!]_{adom}$ liefert immer ein endliches Ergebnis; aber das Ergebnis kann sich (vom Standpunkt des Datenbanknutzers) auf unnatürliche Weise ändern, wenn ein neuer Wert in die Datenbank eingetragen wird (selbst wenn dieser scheinbar nichts mit der Anfrage zu tun hat).

  Beispiel: $Q := \left\{ \ (x) \ : \ \big(\ R(x) \ \land \ \exists y \big(\neg R(y) \ \land \ \forall z (R(z) \lor z{=}y) \big) \ \big) \ \right\}$
- Radikaler Schritt: Betrachte nur Anfragen, die *bereichsunabhängig* sind, d.h. bei denen es egal ist, ob sie in $[\![\cdot]\!]_{nat}$, $[\![\cdot]\!]_{adom}$ oder in $[\![\cdot]\!]_{\mathbf{d}}$ für irgendeine Menge $\mathbf{d}$ mit $\operatorname{adom}(Q, \mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$ auswertet werden.

Definition 7.7 (bereichunabhängige CALC[**S**]-Anfragen)**.**
Eine Anfrage $Q \in \text{CALC}[\mathbf{S}]$ heißt *bereichsunabhängig* (domain independent), falls für jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und alle $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\operatorname{adom}(Q, \mathbf{I}) \subseteq \mathbf{d}, \mathbf{d}'$ gilt: $[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) = [\![Q]\!]_{\mathbf{d}'}(\mathbf{I})$.

Folie 252

Beispiele 7.8. Beispiele für bereichsunabhängige Anfragen:

(a) Welche Filme laufen in mindestens 2 Kinos?

$$\Big\{ \ (x_T) \ : \ \exists x_K \, \exists x_Z \, \exists y_K \, \exists y_Z \big(\mathit{Programm}(x_K, x_T, x_Z) \land \mathit{Programm}(y_K, x_T, y_Z) \land \neg x_K{=}y_K \big) \Big\}$$

(b) In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ \ (x_{Titel}) \ : \ \exists x_{Regie} \ \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \land \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

(c) Welche Filme laufen nur zu 1 Uhrzeit?

$$\Big\{ \ (x_{Titel}) \ : \ \exists x_{Kino} \, \exists x_{Zeit} \ \Big(\ \mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}) \land \forall y_{Kino} \, \forall y_{Zeit} \big(\mathit{Programm}(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit}{=}x_{Zeit} \ \big) \Big) \Big\}$$

Folie 253

(d) Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{\ (x_T)\ :\ \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S)\ \wedge \\ \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \to \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

Nicht *bereichunabhängig ist ...*

$$\Big\{ (x_{Titel})\ :\ \forall y_S \Big(\exists x_R\, \mathit{Filme}(x_{Titel}, x_R, y_S)\ \to\ \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

Folie 254

$\mathbf{CALC}_{nat}$, $\mathbf{CALC}_{adom}$, $\mathbf{CALC}_{di}$

Definition 7.9.
$\mathrm{CALC}_{nat}[\mathbf{S}]$ bezeichnet die Klasse aller in der natürlichen Semantik $[\![\cdot]\!]_{nat}$ interpretierten $\mathrm{CALC}[\mathbf{S}]$-Anfragen.

$\mathrm{CALC}_{adom}[\mathbf{S}]$ bezeichnet die Klasse aller in der Active Domain Semantik $[\![\cdot]\!]_{adom}$ interpretierten $\mathrm{CALC}[\mathbf{S}]$-Anfragen.

$\mathrm{CALC}_{di}[\mathbf{S}]$ bezeichnet die Klasse aller bereichsunabhängigen $\mathrm{CALC}[\mathbf{S}]$-Anfragen (interpretiert in $[\![\cdot]\!]_{adom}$ oder, äquivalent dazu, in $[\![\cdot]\!]_{nat}$).
("di" steht für "domain independent")

Satz 7.10 (Äquivalenz von CALC_{adom}, CALC_{di} und relationaler Algebra)**.** *Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen ausdrücken:*

(a) CALC_{di}

(b) CALC_{adom}

(c) relationale Algebra (unbenannte oder benannte Perspektive).

Und für jedes feste Datenbankschema $\mathbf{S}$ *gilt: Jede Anfrage aus einer dieser Sprachen kann in polynomieller Zeit in äquivalente Anfragen der anderen Sprachen übersetzt werden.*

Beweis. *(a)$\Longrightarrow$(b):* Für jede CALC_{di}-Anfrage Q wähle die CALC_{adom}-Anfrage $Q' := Q$. Dann gilt für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und alle $\mathbf{d} \subseteq \mathbf{dom}$ mit $\mathbf{d} \supseteq \text{adom}(Q, \mathbf{I})$, dass $[\![Q']\!]_{adom}(\mathbf{I}) = [\![Q]\!]_{\mathbf{d}}(\mathbf{I})$.

(b)$\Longrightarrow$(c): Sei $\mathbf{S}$ ein beliebiges Datenbankschema. Für eine endliche Menge $C \subseteq \mathbf{dom}$ sei AD^C die folgende Anfrage der relationalen Algebra:

$$\text{AD}^C \ := \ \bigcup_{c \in C} \{(c)\} \ \cup \ \bigcup_{R \in \mathbf{S}} \bigcup_{i=1}^{\text{ar}(R)} \pi_i(R)\,,$$

und definiere $(\text{AD}^C)_1 := \text{AD}^C$ und $(\text{AD}^C)_{r+1} := \big((\text{AD}^C)_r \times \text{AD}^C \big)$ für jedes $r \in \mathbb{N}_{\geqslant 1}$. Dann gilt für jedes $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, dass $[\![\text{AD}^C]\!](\mathbf{I}) = C \cup \text{adom}(\mathbf{I})$ und $[\![(\text{AD}^C)_r]\!](\mathbf{I}) = \big(C \cup \text{adom}(\mathbf{I})\big)^r$ für alle $r \in \mathbb{N}_{\geqslant 1}$.

Für den Beweis von "(b)$\Longrightarrow$(c)" betrachten wir nur solche CALC_{adom}-Anfragen Q vom Schema $\mathbf{S}$, die von der Form $\{(x_1, \ldots, x_r) \ : \ \varphi\}$ sind, wobei $x_1, \ldots, x_r$ paarweise verschiedene Variablen sind und φ eine $\mathsf{FO}[\mathbf{S}]$-Formel ist, in der keins der Symbole $\forall, \wedge, \rightarrow, \leftrightarrow$ vorkommt (*Übungsaufgabe:* Warum folgt die Richtung "(b)$\Longrightarrow$(c)" dann auch für alle beliebigen CALC_{adom}-Anfragen?).

Per Induktion über den Aufbau von $\mathsf{FO}[\mathbf{S}]$ zeigen wir, dass es für jedes $\varphi \in \mathsf{FO}[\mathbf{S}]$, in dem keins der Symbole $\forall, \wedge, \rightarrow, \leftrightarrow$ vorkommt, und für jedes Tupel $(x_1, \ldots, x_r)$ von paarweise verschiedenen Variablen mit $\text{frei}(\varphi) \subseteq \{x_1, \ldots, x_r\}$ und für jede endliche Menge $C \subseteq \mathbf{dom}$ mit $C \supseteq \text{adom}(\varphi)$ eine Anfrage

$$Q^C_{\varphi,(x_1,\ldots,x_r)}$$

der relationalen Algebra (unbenannte Perspektive) gibt, so dass für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ gilt:

$$\begin{aligned} & [\![Q^C_{\varphi,(x_1,\ldots,x_r)}]\!](\mathbf{I}) \\ = \ & \Big\{ \ (a_1, \ldots, a_r) \in \big(C \cup \text{adom}(\mathbf{I})\big)^r \ : \ \mathbf{I} \models_{C \cup \text{adom}(\mathbf{I})} \varphi[a_1, \ldots, a_r] \ \Big\} \end{aligned} \tag{7.1}$$

Beachte: Daraus folgt dann insbesondere für $C := \text{adom}(\varphi)$, dass

$$[\![Q^C_{\varphi,(x_1,\ldots,x_r)}]\!](\mathbf{I}) \ = \ [\![\, \{(x_1, \ldots, x_r) : \varphi\} \,]\!]_{adom}(\mathbf{I})\,,$$

und dies schließt den Beweis der Richtung "(b)$\Longrightarrow$(c)" dann ab.

Induktionsanfang:

Fall 1: φ ist von der Form $x_i{=}c$ oder $c{=}x_i$ mit $c \in C$ und $i \in \{1, \ldots, r\}$. Wir wählen $Q^C_{\varphi,(x_1,\ldots,x_r)} := \sigma_{i=c}((\text{AD}^C)_r)$.

Fall 2: φ ist von der Form $x_i = x_j$ mit $i, j \in \{1, \ldots, r\}$.
Wir wählen $Q^C_{\varphi,(x_1,\ldots,x_r)} := \sigma_{i=j}\big((\mathrm{AD}^C)_r\big)$.

Fall 3: φ ist von der Form $R(v_1, \ldots, v_{\mathrm{ar}(R)})$ mit $v_1, \ldots, v_{\mathrm{ar}(R)} \in \{x_1, \ldots, x_r\} \cup C$. Wir wählen

$$Q^C_{\varphi,(x_1,\ldots,x_r)} \;:=\; \pi_{1,\ldots,r}\big(\,\sigma_\alpha\big((\mathrm{AD}^C)_r \times \sigma_\beta(R)\big)\,\big)\,,$$

wobei

- β die positive konjunktive Selektionsbedingung ist, die aus den Bedingungen
  - $i{=}c$ für alle $i \in \{1, \ldots, \mathrm{ar}(R)\}$ mit $v_i{=}c$ für ein $c \in C$, und
  - $i{=}j$ für alle $i, j \in \{1, \ldots, \mathrm{ar}(R)\}$ mit $i < j$ und $v_i = v_j \in \mathbf{var}$

  besteht, und
- α die positive konjunktive Selektionsbedingung ist, die aus den Bedingungen $j = r{+}i$ für alle $j \in \{1, \ldots, r\}$ und alle $i \in \{1, \ldots, \mathrm{ar}(R)\}$ besteht, für die gilt: $v_i = x_j$.

Man kann sich davon überzeugen (Details: Übung!), dass in allen drei Fällen (7.1) erfüllt ist.

Induktionsschritt:
Fall 1: φ ist von der Form $\neg\varphi_1$.
Wir wählen $Q^C_{\varphi,(x_1,\ldots,x_r)} := \big((\mathrm{AD}^C)_r - Q^C_{\varphi_1,(x_1,\ldots,x_r)}\big)$.

Fall 2: φ ist von der Form $(\varphi_1 \vee \varphi_2)$.
Wir wählen $Q^C_{\varphi,(x_1,\ldots,x_r)} := \big(Q^C_{\varphi_1,(x_1,\ldots,x_r)} \cup Q^C_{\varphi_2,(x_1,\ldots,x_r)}\big)$.

Fall 3: φ ist von der Form $\exists y \varphi_1$.
Sei x_{r+1} eine Variable, die nicht in $\{x_1, \ldots, x_r\}$ vorkommt und die nicht als quantifizierte Variable in φ_1 vorkommt. Sei φ_1' die Formel, die aus φ_1 entsteht, indem jedes freie Vorkommen der Variablen y ersetzt wird durch die Variable x_{r+1}. Dann ist die Formel φ äquivalent zur Formel $\exists x_{r+1}\varphi_1'$. Außerdem ist $\mathrm{frei}(\varphi_1') \subseteq \{x_1, \ldots, x_r, x_{r+1}\}$. Wir wenden die Induktionsannahme an auf die Formel φ_1' und das Tupel $(x_1, \ldots, x_r, x_{r+1})$. Sei die Anfrage $Q^C_{\varphi_1',(x_1,\ldots,x_r,x_{r+1})}$ gemäß der Induktionsannahme gewählt.
Wir wählen dann $Q^C_{\varphi,(x_1,\ldots,x_r)} := \pi_{1,\ldots,r}\big(Q^C_{\varphi_1',(x_1,\ldots,x_r,x_{r+1})}\big)$.

Man kann sich davon überzeugen (Details: Übung!), dass in allen drei Fällen (7.1) erfüllt ist. Dies beendet den Beweis der Richtung "(b)$\Longrightarrow$(c)".

(c)$\Longrightarrow$(a): Wir wählen eine feste unendliche Folge $x_1, x_2, x_3, \ldots$ von paarweise verschiedenen Variablen.
Per Induktion über den Aufbau der relationalen Algebra zeigen wir, dass es für jede Anfrage Q der relationaln Algebra (unbenannte Perspektive) eine $\mathsf{FO}[\mathbf{S}]$-Formel φ_Q gibt, so dass gilt:

($*$) $\operatorname{adom}(\varphi_Q) = \operatorname{adom}(Q)$, $\operatorname{frei}(\varphi_Q) = \{x_1, \ldots, x_r\}$ wobei r die Stelligkeit von Q ist, und für die CALC[$\mathbf{S}$]-Anfrage

$$Q' \ := \ \{\ (x_1, \ldots, x_r) \ : \ \varphi_Q \ \}$$

gilt $[\![Q]\!](\mathbf{I}) = [\![Q']\!]_{\mathbf{d}}(\mathbf{I})$ für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und alle $\mathbf{d} \subseteq \mathbf{dom}$ mit $\mathbf{d} \supseteq \operatorname{adom}(Q, \mathbf{I})$.

Induktionsanfang:

Fall 1: Q ist von der Form R für ein $R \in \mathbf{S}$.
Sei $r = \operatorname{ar}(R)$ und wähle $\varphi_Q := R(x_1, \ldots, x_r)$.

Fall 2: Q ist von der Form $\{\,(c)\,\}$ für ein $c \in \mathbf{dom}$.
Wähle $\varphi_Q := x_1{=}c$.

Man kann sich davon überzeugen (Details: Übung!), dass in beiden Fällen ($*$) erfüllt ist.

Induktionsschritt:

Fall 1: Q ist von der Form $\sigma_{j=c}(Q_1)$.
Sei r die Stelligkeit von Q (und von Q_1), und sei die Formel φ_{Q_1} gemäß Induktionsannahme gewählt. Wir wählen $\varphi_Q := (\varphi_{Q_1} \wedge\ x_j{=}c)$.

Fall 2: Q ist von der Form $\sigma_{j=j'}(Q_1)$.
Sei r die Stelligkeit von Q (und von Q_1), und sei die Formel φ_{Q_1} gemäß Induktionsannahme gewählt. Wir wählen $\varphi_Q := (\varphi_{Q_1} \wedge x_j{=}x_{j'})$.

Fall 3: Q ist von der Form $\pi_{j_1,\ldots,j_r}(Q_1)$.
Sei r die Stelligkeit von Q und sei r_1 die Stelligkeit von Q_1. Sei die Formel φ_{Q_1} gemäß Induktionsannahme gewählt.
Seien $z_1, \ldots, z_{r_1}$ neue Variablen, die weder in φ_{Q_1} noch in $\{x_1, \ldots, x_r\}$ vorkommen. Sei φ'_{Q_1} die Formel, die aus φ_{Q_1} entsteht, indem jedes freie Vorkommen der Variablen x_i für $i \in \{1, \ldots, r_1\}$ ersetzt wird durch die Variable z_i. Wir wählen

$$\varphi_Q \ := \quad \exists z_1 \cdots \exists z_{r_1} \big(\ \varphi'_{Q_1} \ \wedge\ \bigwedge_{i=1}^{r} x_i{=}z_{j_i}\ \big)\,.$$

Fall 4: Q ist von der Form $(Q_1 \times Q_2)$.
Für $i \in \{1,2\}$ sei r_i die Stelligkeit von Q_i, und sei die Formel φ_{Q_i} gemäß Induktionsannahme gewählt. Dann hat Q die Stelligkeit $r = r_1 + r_2$.
Sei φ'_{Q_2} die Formel, die aus φ_{Q_2} entsteht, indem (1) die quantifizierten Variablen so umbenannt werden, dass keine der Variablen $x_1, \ldots, x_r$ als quantifizierte Variable genutzt wird, und (2) jedes freie Vorkommen der Variablen x_i für $i \in \{1, \ldots, r_2\}$ ersetzt wird durch die Variable x_{r_1+i}. Wir wählen $\varphi_Q := (\varphi_{Q_1} \wedge \varphi'_{Q_2})$.

Fall 5: Q ist von der Form $(Q_1 \cup Q_2)$.
Sei r die Stelligkeit von Q (und von Q_1 und von Q_2), und seien die Formeln φ_{Q_1} und φ_{Q_2} gemäß Induktionsannahme gewählt. Wir wählen $\varphi_Q := (\varphi_{Q_1} \vee \varphi_{Q_2})$.

Fall 6: Q ist von der Form $(Q_1 - Q_2)$.
Sei r die Stelligkeit von Q (und von Q_1 und von Q_2), und seien die Formeln φ_{Q_1} und φ_{Q_2} gemäß Induktionsannahme gewählt. Wir wählen $\varphi_Q := (\varphi_{Q_1} \wedge \neg\varphi_{Q_2})$.

Man kann sich davon überzeugen (Details: Übung!), dass in allen sechs Fällen $(*)$ erfüllt ist.

Für jede der Richtungen "(a)$\Longrightarrow$(b)", "(b)$\Longrightarrow$(c)" und "(c)$\Longrightarrow$(a)" kann man sich leicht davon überzeugen, dass die im Beweis durchgeführte Übersetzung von einer Anfrage einer Sprache in die dazu äquivalente Anfrage der anderen Sprache in polynomieller Zeit durchgeführt werden kann. Dies beendet den Beweis von Satz 7.10. $\square$

Folie 255

Bereichsunabhängigkeit — Pro & Contra CALC_{di}

Vorteile:

- logik-basierte Anfragesprache, die die "richtige" Ausdrucksstärke hat (äquivalent zur relationalen Algebra; insbesondere sind alle CALC_{di}-Anfragen "sicher")
- keine unerwünschten Effekte, wie sie bei CALC_{adom} auftreten können (vgl. Beispiel 7.5).
- Ergebnis der Anfragen ist unabhängig davon, in welcher Semantik ($[\![\cdot]\!]_{adom}$ oder $[\![\cdot]\!]_{nat}$ oder $[\![\cdot]\!]_{\mathbf{d}}$) sie interpretiert werden

Frage:

- Wie kann man bei einer gegebenen CALC-Anfrage Q feststellen, ob Q zu CALC_{di} gehört, d.h. ob Q bereichsunabhängig ist?

Antwort — großer Nachteil von CALC_{di}*:*

- Das ist *unentscheidbar*, d.h. es gibt keinen Algorithmus, der bei Eingabe einer beliebigen CALC[**S**]-Anfrage Q nach endlich vielen Schritten anhält und entscheidet, ob Q bereichsabhängig ist oder nicht.
- Beweis: auf den nächsten Folien ...

Folie 256

Der Satz von Trakhtenbrot

Theorem 7.11 (Trakhtenbrot, 1950 (hier ohne Beweis))**.**
Sei **S** *ein Datenbankschema, das mindestens ein Relationssymbol enthält, dessen Stelligkeit* $\geqslant 2$ *ist. Dann ist das folgende Problem unentscheidbar:*

Erfüllbarkeitsproblem für FO[S] über Datenbanken	
Eingabe:	Ein FO[**S**]-Satz ψ
Frage:	Gibt es eine DB $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ mit $\mathbf{I} \models_{adom} \psi$?

Beweis: Siehe Vorlesung *„Logik und Komplexität“*.

Folie 257

Bereichsunabhängigkeit ist unentscheidbar

Satz 7.12. *Sei* **S** *ein Datenbankschema, das mindestens ein Relationssymbol R enthält, dessen Stelligkeit* $\geqslant 2$ *ist. Dann ist das folgende Problem unentscheidbar:*

Bereichsunabhängigkeit für CALC[S]	
Eingabe:	Eine CALC[**S**]-Anfrage Q
Frage:	Ist Q bereichsunabhängig (d.h., gehört Q zu $\text{CALC}_{di}[\mathbf{S}]$) ?

Beweis: Einfache Folgerung aus dem Satz von Trakhtenbrot; siehe Tafel.

7.2 Sicherer Relationenkalkül: CALC_{sr}

Folie 258

Motivation

- CALC_{di} ist eine logik-basierte Anfragesprache, die *die "richtige Ausdrucksstärke"* hat (nämlich dieselbe wie die relationale Algebra).
- Wegen der Unentscheidbarkeit (Satz 7.12) ist CALC_{di} als Anfragesprache *für praktische Zwecke aber ungeeignet*, da man bei Eingabe einer "Anfrage" aus $\mathrm{CALC}[\mathbf{S}]$ nicht feststellen kann, ob dies eine Anfrage in CALC_{di} ist.
- *Ziel jetzt:* Finde ein *syntaktisches Kriterium* für $\mathrm{CALC}[\mathbf{S}]$-Anfragen, das
  (a) algorithmisch leicht nachprüfbar ist,
  (b) Bereichsunabhängigkeit garantiert und
  (c) zu einer Anfragesprache führt, die immer noch dieselbe Ausdrucksstärke wie die relationale Algebra hat.

$\rightsquigarrow$ *Sicherer Relationenkalkül* CALC_{sr}*:* entscheidbare Teilklasse von CALC_{di}, die dieselbe Ausdrucksstärke wie CALC_{di} hat.

Folie 259

Safe-Range Normalform (SRNF)

Für jede $\mathsf{FO}[\mathbf{S}]$-Formel φ sei $\mathrm{SRNF}(\varphi)$ die $\mathsf{FO}[\mathbf{S}]$-Formel, die aus φ entsteht, indem man die folgenden Regeln so lange anwendet, bis keine Regel mehr anwendbar ist:

- Alle quantifizierten Variablen werden so umbenannt, dass sie paarweise verschieden sind und verschieden von den freien Variablen der Formel.
- Teilformeln der Form $\forall x \psi$ werden ersetzt durch $\neg\exists x \neg\psi$
- Implikationspfeile $\rightarrow$ und "genau-dann-wenn"-Pfeile $\leftrightarrow$ werden durch Kombinationen von $\wedge$, $\vee$, $\neg$ ersetzt:
  - Teilformeln der Form $(\psi_1 \rightarrow \psi_2)$ werden ersetzt durch $(\neg\psi_1 \;\vee\; \psi_2)$

  - Teilformeln der Form $(\psi_1 \leftrightarrow \psi_2)$ werden ersetzt durch $\big((\psi_1 \wedge \psi_2) \vee (\neg\psi_1 \ \wedge \ \neg\psi_2)\big)$

- *Negationszeichen werden "nach innen geschoben"*:
  - $\neg(\psi_1 \wedge \psi_2)$ wird ersetzt durch $(\neg\psi_1 \vee \neg\psi_2)$
  - $\neg(\psi_1 \vee \psi_2)$ wird ersetzt durch $(\neg\psi_1 \wedge \neg\psi_2)$
- "doppelte Negationszeichen" werden gelöscht: Teilformeln der Form $\neg\neg\psi$ werden ersetzt durch ψ

Offensichtlich gilt: Die Formel SRNF(φ) ist äquivalent zur Formel φ. Falls $\varphi = \text{SRNF}(\varphi)$, so sagen wir: *$\varphi$ ist in Safe-Range Normalform* (kurz: SRNf).

Folie 260

Beispiel für Transformation von φ zu SRNF(φ)

Beispiel 7.13.

$$\begin{aligned}\varphi \ := \quad & \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \\ & \qquad \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{"\textit{Stephen Spielberg}"}, y_S)\big)\Big)\end{aligned}$$

$$\begin{aligned}\rightsquigarrow \quad & \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \\ & \qquad \neg\exists y_S \neg \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{"Stephen Spielberg"}, y_S)\big)\Big)\end{aligned}$$

$$\begin{aligned}\rightsquigarrow \quad & \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \\ & \qquad \neg\exists y_S \neg \big(\neg\mathit{Filme}(x_T, x_R, y_S) \vee \exists y_T \, \mathit{Filme}(y_T, \text{"Stephen Spielberg"}, y_S)\big)\Big)\end{aligned}$$

$$\begin{aligned}\rightsquigarrow \quad & \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \\ & \qquad \neg\exists y_S \big(\neg\neg\mathit{Filme}(x_T, x_R, y_S) \wedge \neg\exists y_T \, \mathit{Filme}(y_T, \text{"Stephen Spielberg"}, y_S)\big)\Big)\end{aligned}$$

$$\begin{aligned}\rightsquigarrow \quad & \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \\ & \qquad \neg\exists y_S \big(\mathit{Filme}(x_T, x_R, y_S) \wedge \neg\exists y_T \, \mathit{Filme}(y_T, \text{"\textit{Stephen Spielberg}"}, y_S)\big)\Big) \\ =: \ & \text{SRNF}(\varphi)\end{aligned}$$

Folie 261

Bereichsbeschränkte Variablen einer Formel in SRNf

Definition 7.14. Per Induktion nach dem Aufbau von *SRNf-Formeln* φ definieren wir $rr(\varphi)$ folgendermaßen:
($rr(\varphi)$ steht für "range restricted variables of φ")

- $rr\big(R(v_1,\ldots,v_{\mathrm{ar}(R)})\big) := \{v_1,\ldots,v_{\mathrm{ar}(R)}\} \cap \mathbf{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathbf{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1,v_2\} \subseteq \mathbf{dom}$ oder $\{v_1,v_2\} \subseteq \mathbf{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$
- Für $x,y \in \mathbf{var}$ gilt: $rr\big(\psi \wedge x{=}y\big) := rr\big(x{=}y \wedge \psi\big) :=$
  $\begin{cases} rr(\psi) \cup \{x,y\} & \textit{falls } \{x,y\} \cap rr(\psi) \neq \emptyset \\ rr(\psi) & \textit{sonst} \end{cases}$
- Ist weder φ_1 noch φ_2 von der Form $x{=}y$ für $x,y \in \mathbf{var}$, so gilt:
  $rr\big(\varphi_1 \wedge \varphi_2\big) := rr(\varphi_1) \cup rr(\varphi_2)$
- $rr\big(\exists x \psi\big) := \begin{cases} \perp & \textit{falls } x \notin rr(\psi) \\ rr(\psi) \setminus \{x\} & \textit{sonst} \end{cases}$

Dabei gilt: *"Einmal $\perp \rightsquigarrow$ immer $\perp$"*, d.h. für alle Mengen X gilt:
$\perp \;=\; \perp\setminus X \;=\; \perp\cap X \;=\; X \cap \perp \;=\; \perp \cap \perp \;=\; \perp \cup X \;=\; X \cup \perp \;=\; \perp \cup \perp$

- $rr(\neg\psi) := \begin{cases} \perp & \textit{falls } rr(\psi) = \perp \\ \emptyset & \textit{sonst} \end{cases}$

Bemerkung: Insbesondere gilt $rr(\varphi) = \perp$ *oder* $rr(\varphi) \subseteq \mathrm{frei}(\varphi)$

Folie 262

Beispiele für $rr(\cdot)$

Beispiel 7.15. (a) Für die SRNf-Formel

$$\varphi \;:=\; \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \;\wedge\; \neg\exists y_S \big(\mathit{Filme}(x_T, x_R, y_S) \wedge \neg\exists y_T\, \mathit{Filme}(y_T, \text{"Stephen Spielberg"}, y_S) \big) \Big)$$

gilt $\quad rr(\varphi) \;=\; \{\, x_T \,\} \;=\; \mathrm{frei}(\varphi).$

(b) Für die SRNf-Formel

$$\psi \ := \ \neg\exists y_S\Big(\big(\exists x_R\ \mathit{Filme}(x_T, x_R, y_S)\big) \ \wedge \ \neg\big(\exists y_T\ \mathit{Filme}(y_T, \text{“}\mathit{Stephen\ Spielberg}\text{”}, y_S)\big)\Big)$$

gilt $\quad rr(\psi) \ = \ \emptyset \ \subsetneq \ \text{frei}(\psi) \ = \ \{x_T\}$.

($\psi = \text{SRNF}(\psi')$, wobei ψ' die allerletzte Formel von Beispiel 7.8 ist.

In der Active Domain Semantik ist ψ äquivalent zu φ)

(c) Für die SRNf-Formel

$$\chi \ := \quad R(x) \ \wedge \ \exists y\Big(\neg R(y) \ \wedge \ \neg\exists z\big(\neg R(z) \ \wedge \ \neg z{=}y\big)\Big)$$

gilt $\quad rr(\chi) \ = \ \bot$. *($\chi$ ist die Formel* SRNF*(”Formel aus Bsp. 7.5”))*

Folie 263

Der Sichere Relationenkalkül CALC$_{sr}$

Definition 7.16. Sei $\mathbf{S}$ ein Datenbankschema.
CALC$_{sr}$[$\mathbf{S}$] bezeichnet die Klasse aller CALC[$\mathbf{S}$]-Anfragen der Form $\{(e_1,\ldots,e_r) : \varphi\}$, für die gilt: $rr\big(\text{SRNF}(\varphi)\big) = \text{frei}(\varphi)$.

Bemerkung:
Aus den Definitionen von SRNF$(\cdot)$ und $rr(\cdot)$ erhält man direkt einen Algorithmus, der bei Eingabe einer FO[$\mathbf{S}$]-Formel φ überprüft, ob $rr\big(\text{SRNF}(\varphi)\big) = \text{frei}(\varphi)$.
Somit gibt es also auch einen Algorithmus, der bei Eingabe einer CALC[$\mathbf{S}$]-Anfrage Q entscheidet, ob Q zu CALC$_{sr}$[$\mathbf{S}$] gehört oder nicht.

Ziel jetzt:
Zeige, dass alle CALC$_{sr}$-Anfragen *bereichsunabhängig* sind und dass CALC$_{sr}$ dieselben Anfragefunktionen ausdrücken kann wie die relationale Algebra.

Folie 264

Beispiele für CALC$_{sr}$-Anfragen

Anfragen in CALC$_{sr}$*:*

- Welche Filme laufen in mindestens 2 Kinos?

$$\Big\{\ (x_T)\ :\ \exists x_K\,\exists x_Z\,\exists y_K\,\exists y_Z\big(\mathit{Programm}(x_K,x_T,x_Z)\wedge\mathit{Programm}(y_K,x_T,y_Z)\wedge\neg x_K{=}y_K\big)\Big\}$$

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ \ (x_{\mathit{Titel}}) \ : \ \exists x_{\mathit{Regie}} \ \Big(\mathit{Filme}(x_{\mathit{Titel}}, x_{\mathit{Regie}}, \text{“George Clooney”}) \ \land \ \neg \mathit{Filme}(x_{\mathit{Titel}}, \text{“George Clooney”}, \text{“George Clooney”}) \ \Big) \ \Big\}$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{ \ (x_T) \ : \ \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \land \ \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \to \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

Nicht *in* CALC$_{sr}$ *sind ...*

- $$\Big\{ (x_T) \ : \ \forall y_S \Big(\exists x_R \, \mathit{Filme}(x_T, x_R, y_S) \ \to \ \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$
- $$\Big\{ (x) \ : \ \ R(x) \ \land \ \exists y \Big(\neg R(y) \ \land \ \forall z \, \big(R(z) \ \lor \ z{=}y \big) \Big) \Big\}$$

Folie 265

Ein technisches Lemma

Lemma 7.17. *Sei* $\mathbf{S}$ *ein Datenbankschema.*
Für jede $\mathsf{FO}[\mathbf{S}]$*-Formel* φ *in SRNf mit* $rr(\varphi) \neq \bot$,
für jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$,
für jedes $\mathbf{d} \subseteq \mathbf{dom}$ *mit* $\mathrm{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d}$ *und*
für jede Belegung β *für* φ *in* $\mathbf{d}$ *gilt:*

(1) Falls $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, *so gilt für alle* $x \in rr(\varphi)$*:* $\quad \beta(x) \in \mathrm{adom}(\varphi, \mathbf{I})$.

und

(2) Für alle $\mathbf{d}' \subseteq \mathbf{dom}$ *mit* $\mathrm{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d}'$, *so dass* β *auch eine Belegung für* φ *in* $\mathbf{d}'$ *ist (d.h.: für alle* $x \in \mathrm{frei}(\varphi)$ *ist* $\beta(x) \in \mathbf{d}' \cap \mathbf{d}$*) gilt:*

$$\mathbf{I} \models_{\mathbf{d}} \varphi[\beta] \quad \Longleftrightarrow \quad \mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]$$

Folgerung 7.18. *Jede Anfrage Q in* $\text{CALC}_{sr}[\mathbf{S}]$ *ist bereichsunabhängig.*

Beweis von Lemma 7.17.
zu (1): Per Induktion nach dem Aufbau von $\mathsf{FO}[\mathbf{S}]$ zeigen wir, dass die Aussage (1) des Lemmas für alle $\varphi \in \mathsf{FO}[\mathbf{S}]$ in SRNf mit $rr(\varphi) \neq \bot$, jedes $\mathbf{I} \in inst(\mathbf{S})$, jedes $\mathbf{d} \subseteq \mathbf{dom}$ mit $\text{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d}$ und jede Belegung β für φ in $\mathbf{d}$ gilt.

Induktionsanfang:

Fall 1: φ ist von der Form $R(v_1, \ldots, v_{\text{ar}(R)})$.
Dann ist $rr(\varphi) = \{v_1, \ldots, v_{\text{ar}(R)}\} \cap \mathbf{var}$. Offensichtlicherweise ist dann (1) erfüllt, da aus $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ folgt, dass $\beta(x) \in \text{adom}(\mathbf{I}) \subseteq \text{adom}(\varphi, \mathbf{I})$ für alle $x \in rr(\varphi)$ gilt.

Fall 2: φ ist von der Form $x{=}a$ oder $a{=}x$ für $a \in \mathbf{dom}$ und $x \in \mathbf{var}$.
Dann ist $rr(\varphi) = \{x\}$. Offensichtlicherweise ist dann (1) erfüllt, da aus $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ folgt, dass $\beta(x) = a \in \text{adom}(\varphi) \subseteq \text{adom}(\varphi, \mathbf{I})$.

Fall 3: φ ist von der Form $x{=}y$ mit $x, y \in \mathbf{dom}$ oder $x, y \in \mathbf{var}$.
Dann ist $rr(\varphi) = \emptyset$ und (1) ist trivialerweise erfüllt.

Induktionsschritt:

Fall 1: φ ist von der Form $\neg\varphi_1$.
Dann ist $rr(\varphi) = \emptyset$ und (1) ist trivialerweise erfüllt.

Fall 2: φ ist von der Form $(\varphi_1 \vee \varphi_2)$.
Dann ist $rr(\varphi) = rr(\varphi_1) \cap rr(\varphi_2)$.
Falls $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, so gibt es ein $i \in \{1, 2\}$ so dass $\mathbf{I} \models_{\mathbf{d}} \varphi_i[\beta]$. Gemäß Induktionsannahme gilt (1) für die Formel φ_i (beachte dazu, dass $rr(\varphi_i) \neq \bot$ da $rr(\varphi) \neq \bot$; und φ_i ist in SNRf da φ in SNRf ist), und somit gilt für alle $x \in rr(\varphi_i)$, dass $\beta(x) \in \text{adom}(\varphi_i, \mathbf{I}) \subseteq \text{adom}(\varphi, \mathbf{I})$. Wegen $rr(\varphi) \subseteq rr(\varphi_i)$ ist Fall 2 also gezeigt.

Fall 3: φ ist von der Form $(\varphi_1 \wedge \varphi_2)$, wobei weder φ_1 noch φ_2 von der Form $x{=}y$ mit $x, y \in \mathbf{var}$ ist.
Dann ist $rr(\varphi) = rr(\varphi_1) \cup rr(\varphi_2)$.
Falls $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, so gilt für alle $i \in \{1, 2\}$, dass $\mathbf{I} \models_{\mathbf{d}} \varphi_i[\beta]$. Gemäß Induktionsannahme gilt (1) für die Formel φ_i (beachte dazu, dass $rr(\varphi_i) \neq \bot$ da $rr(\varphi) \neq \bot$; und φ_i ist in SNRf da φ in SNRf ist), und somit gilt für alle $x \in rr(\varphi_i)$, dass $\beta(x) \in \text{adom}(\varphi_i, \mathbf{I}) \subseteq \text{adom}(\varphi, \mathbf{I})$. Wegen $rr(\varphi) = rr(\varphi_1) \cup rr(\varphi_2)$ ist Fall 3 also gezeigt.

Fall 4: φ ist von der Form $(\varphi_1 \wedge x{=}y)$ oder von der Form $(x{=}y \wedge \varphi_1)$ mit $x, y \in \mathbf{var}$, und es gilt $rr(\varphi_1) \cap \{x, y\} = \emptyset$. Dann ist $rr(\varphi) = rr(\varphi_1)$.

Falls $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, so gilt insbesondere $\mathbf{I} \models_{\mathbf{d}} \varphi_1[\beta]$. Gemäß Induktionsannahme gilt (1) für die Formel φ_1 (beachte dazu, dass $rr(\varphi_1) \neq \bot$ da $rr(\varphi) \neq \bot$; und φ_1 ist in SNRf da φ in SNRf ist), und somit gilt für alle $z \in rr(\varphi_1)$, dass $\beta(z) \in \mathrm{adom}(\varphi_1, \mathbf{I}) = \mathrm{adom}(\varphi, \mathbf{I})$. Wegen $rr(\varphi) = rr(\varphi_1)$ ist Fall 4 also gezeigt.

Fall 5: φ ist von der Form $(\varphi_1 \wedge x{=}y)$ oder von der Form $(x{=}y \wedge \varphi_1)$ mit $x, y \in \mathbf{var}$, und es gilt $rr(\varphi_1) \cap \{x, y\} \neq \emptyset$.
O.B.d.A. sei $x \in rr(\varphi_1)$. Dann ist $rr(\varphi) = rr(\varphi_1) \cup \{y\}$.
Falls $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, so gilt $\mathbf{I} \models_{\mathbf{d}} \varphi_1[\beta]$ und $\beta(x) = \beta(y)$. Gemäß Induktionsannahme gilt (1) für die Formel φ_1 (beachte dazu, dass $rr(\varphi_1) \neq \bot$ da $rr(\varphi) \neq \bot$; und φ_1 ist in SNRf da φ in SNRf ist), und somit gilt für alle $z \in rr(\varphi_1)$, dass $\beta(z) \in \mathrm{adom}(\varphi_1, \mathbf{I}) = \mathrm{adom}(\varphi, \mathbf{I})$. Wegen $x \in rr(\varphi_1)$ und $\beta(x) = \beta(y)$ folgt, dass auch gilt: $\beta(y) \in \mathrm{adom}(\varphi, \mathbf{I})$. Wegen $rr(\varphi) = rr(\varphi_1) \cup \{y\}$ ist Fall 5 also gezeigt.

Fall 6: φ ist von der Form $\exists y\, \varphi_1$.
Wegen $rr(\varphi) \neq \bot$ ist auch $rr(\varphi_1) \neq \bot$ und es gilt: $y \in rr(\varphi_1)$ und $rr(\varphi) = rr(\varphi_1) \setminus \{y\}$.
Falls $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, so gibt es eine Belegung $a \in \mathbf{d}$ für y, so dass $\mathbf{I} \models_{\mathbf{d}} \varphi_1[\beta\frac{a}{y}]$. Gemäß Induktionsannahme gilt (1) für die Formel φ_1 und die Belegung $\beta\frac{a}{y}$. Somit gilt für alle $x \in rr(\varphi_1)$, dass $\beta\frac{a}{y}(x) \in \mathrm{adom}(\varphi_1, \mathbf{I}) \subseteq \mathrm{adom}(\varphi, \mathbf{I})$. Wegen $\beta(x) = \beta\frac{a}{y}(x)$ für alle $x \in rr(\varphi) = rr(\varphi_1) \setminus \{y\}$, ist Fall 6 also gezeigt.

Wir haben alle Fälle abgehandelt, die bei Formeln φ in SRNf auftreten können. Damit ist Teil (1) des Lemmas bewiesen.

zu (2): Per Induktion nach dem Aufbau von $\mathsf{FO}[\mathbf{S}]$ zeigen wir, dass für alle $\varphi \in \mathsf{FO}[\mathbf{S}]$ in SRNf mit $rr(\varphi) \neq \bot$, jedes $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, alle $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\mathrm{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ und jede Belegung β für φ in $\mathbf{d} \cap \mathbf{d}'$ gilt:

$$\mathbf{I} \models_{\mathbf{d}} \varphi[\beta] \quad\iff\quad \mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]\,.$$

Induktionsanfang:

Fall 1: φ ist von der Form $R(v_1, \ldots, v_{\mathrm{ar}(R)})$.
Sei $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, seien $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\mathrm{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ und sei β eine Belegung für φ in $\mathbf{d} \cap \mathbf{d}'$.
"$\Longrightarrow$": Wenn $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, dann ist $(\beta(v_1), \ldots, \beta(v_{\mathrm{ar}(R)})) \in \mathbf{I}(R)$, und somit auch $\mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]$.
Die Richtung "$\Longleftarrow$" folgt aus Symmetriegründen.

Fall 2: φ ist von der Form $x{=}y$ für $x, y \in \mathbf{dom} \cup \mathbf{var}$.
Sei $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, seien $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\mathrm{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ und sei β eine Belegung für φ in $\mathbf{d} \cap \mathbf{d}'$.

"$\Longrightarrow$": Wenn $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$, dann ist $\beta(x) = \beta(y)$, und somit auch $\mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]$. Die Richtung "$\Longleftarrow$" folgt aus Symmetriegründen.

Induktionsschritt:

Fall 1: φ ist von der Form $\neg\varphi_1$.
Wegen $rr(\varphi) \neq \bot$ ist auch $rr(\varphi_1) \neq \bot$; und da φ in SNRf ist, ist auch φ_1 in SNRf.
Sei $\mathbf{I} \in inst(\mathbf{S})$, seien $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\text{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ und sei β eine Belegung für φ in $\mathbf{d} \cap \mathbf{d}'$. Gemäß Induktionsannahme gilt

$$\mathbf{I} \models_{\mathbf{d}} \varphi_1[\beta] \quad \Longleftrightarrow \quad \mathbf{I} \models_{\mathbf{d}'} \varphi_1[\beta]\,.$$

Da φ von der Form $\neg\varphi_1$ ist, folgt daraus, dass auch gilt:

$$\mathbf{I} \models_{\mathbf{d}} \varphi[\beta] \quad \Longleftrightarrow \quad \mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]\,.$$

Fall 2: φ ist von der Form $(\varphi_1 * \varphi_2)$ mit $* \in \{\wedge, \vee\}$.
Wegen $rr(\varphi) \neq \bot$ ist auch $rr(\varphi_i) \neq \bot$ für alle $i \in \{1,2\}$; und da φ in SNRf ist, sind auch φ_1 und φ_2 in SNRf.
Sei $\mathbf{I} \in inst(\mathbf{S})$, seien $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\text{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ und sei β eine Belegung für φ in $\mathbf{d} \cap \mathbf{d}'$. Gemäß Induktionsannahme gilt für jedes $i \in \{1,2\}$:

$$\mathbf{I} \models_{\mathbf{d}} \varphi_i[\beta] \quad \Longleftrightarrow \quad \mathbf{I} \models_{\mathbf{d}'} \varphi_i[\beta]\,.$$

Für jedes $* \in \{\wedge, \vee\}$ folgt daraus dann direkt (aus der Semantik von "$\wedge$" und "$\vee$"), dass auch gilt:

$$\mathbf{I} \models_{\mathbf{d}} (\varphi_1 * \varphi_2)[\beta] \quad \Longleftrightarrow \quad \mathbf{I} \models_{\mathbf{d}'} (\varphi_1 * \varphi_2)[\beta]\,.$$

Fall 3: φ ist von der Form $\exists y\, \varphi_1$.
Wegen $rr(\varphi) \neq \bot$ ist auch $rr(\varphi_1) \neq \bot$ und es gilt: $y \in rr(\varphi_1)$. Und da φ in SNRf ist, ist auch φ_1 in SNRf.
Sei $\mathbf{I} \in inst(\mathbf{S})$, seien $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\text{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ und sei β eine Belegung für φ in $\mathbf{d} \cap \mathbf{d}'$.
Zum Beweis der Richtung "$\Longrightarrow$" gehen wir davon aus, dass $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ gilt. D.h. es gibt ein $a \in \mathbf{d}$, so dass $\mathbf{I} \models_{\mathbf{d}} \varphi_1[\beta\frac{a}{y}]$. Wegen $y \in rr(\varphi_1)$ folgt aus Aussage (1) des Lemmas für die Belegung $\beta' := \beta\frac{a}{y}$, dass $a = \beta'(y) \in \text{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$. Somit ist $\beta\frac{a}{y}$ eine Belegung für φ_1 in $\mathbf{d} \cap \mathbf{d}'$. Gemäß Induktionsannahme folgt daher aus $\mathbf{I} \models_{\mathbf{d}} \varphi_1[\beta\frac{a}{y}]$, dass auch $\mathbf{I} \models_{\mathbf{d}'} \varphi_1[\beta\frac{a}{y}]$. Wegen $a \in \mathbf{d}'$ gilt also auch $\mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]$. Somit ist die Richtung "$\Longrightarrow$" bewiesen.
Die Richung "$\Longleftarrow$" folgt aus Symmetriegründen.

Wir haben alle Fälle abgehandelt, die bei Formeln φ in SRNf auftreten können. Damit ist Teil (2) des Lemmas bewiesen. $\square$

Beweis von Folgerung 7.18.
Sei $Q = \{\ (e_1,\dots,e_r)\ :\ \varphi\ \}$ eine beliebige Anfrage aus $\text{CALC}_{sr}[\mathbf{S}]$.
O.B.d.A. ist φ in SRNf. Wegen $Q \in \text{CALC}_{sr}[\mathbf{S}]$ gilt: $rr(\varphi) = \text{frei}(\varphi)$; und insbes. ist $rr(\varphi) \neq \bot$ — wir können also Lemma 7.17 für die Formel φ anwenden.
Seien $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ mit $\text{adom}(Q,\mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ beliebig gewählt. Wir müssen zeigen, dass gilt: $[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) = [\![Q]\!]_{\mathbf{d}'}(\mathbf{I})$. Wir zeigen hier "$\subseteq$"; aus Symmetriegründen folgt dann auch "$\supseteq$".
Gemäß Definition der Semantik gilt:

$$[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) \;=\; \Big\{\ \big(\beta(e_1),\dots,\beta(e_r)\big)\ :\ \begin{array}{l}\beta \text{ ist eine Belegung für } \varphi \text{ in } \mathbf{d}\\ \text{mit } \mathbf{I} \models_{\mathbf{d}} \varphi[\beta]\ \Big\}\end{array}$$

Betrachte also eine beliebige Belegung β für φ in $\mathbf{d}$ mit $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ und das zugehörige Tupel $\big(\beta(e_1),\dots,\beta(e_r)\big) \in [\![Q]\!]_{\mathbf{d}}(\mathbf{I})$. Wir müssen zeigen, dass dieses Tupel auch in $[\![Q]\!]_{\mathbf{d}'}(\mathbf{I})$ liegt.
Gemäß Aussage (1) von Lemma 7.17 gilt für jede Variable $x \in rr(\varphi) = \text{frei}(\varphi)$, dass $\beta(x) \in \text{adom}(\varphi,\mathbf{I}) \subseteq \text{adom}(Q,\mathbf{I})$. Wegen $\text{adom}(Q,\mathbf{I}) \subseteq \mathbf{d} \cap \mathbf{d}'$ ist β also auch eine Belegung für φ in $\mathbf{d}'$. Gemäß Aussage (2) von Lemma 7.17 folgt aus $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ also, dass auch $\mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]$.
Somit liegt das Tupel $\big(\beta(e_1),\dots,\beta(e_r)\big)$ also in $[\![Q]\!]_{\mathbf{d}'}(\mathbf{I})$.
Damit haben wir gezeigt, dass $[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) \subseteq [\![Q]\!]_{\mathbf{d}'}(\mathbf{I})$ ist. Die Richtung "$\supseteq$" folgt aus Symmetriegründen. □

Folie 266

Relationenkalkül vs. Relationale Algebra

Korollar 7.19. *Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen ausdrücken:*

(a) CALC_{sr}

(b) CALC_{di}

(c) CALC_{adom}

(d) relationale Algebra (unbenannte oder benannte Perspektive)

Und für jedes feste Datenbankschema $\mathbf{S}$ *gilt: Jede Anfrage aus einer dieser Sprachen kann in polynomieller Zeit in äquivalente Anfragen der anderen Sprachen übersetzt werden.*

Im weiteren Verlauf der Vorlesung schreiben wir manchmal *"Relationenkalkül"* oder einfach CALC, um irgendeine der Varianten CALC_{sr}, CALC_{di} bzw. CALC_{adom} des Relationenkalküls zu bezeichnen.

In der Literatur wird oft der Begriff *"Relational vollständige Sprache"* benutzt, um Anfragesprachen zu bezeichnen, die mindestens die Ausdrucksstärke der relationalen Algebra (bzw., äquivalent dazu, des Relationenkalküls) besitzen.

Beweis von Korollar 7.19.
Die Richtung "(a)$\Longrightarrow$(b)" gilt gemäß Folgerung 7.18. Die Äquivalenz zwischen (b), (c) und (d) ist Aussage von Satz 7.10. Zum Beweis der Richung "(d)$\Longrightarrow$(a)" können wir genauso vorgehen wie im Beweis der Richtung "(c)$\Longrightarrow$(a)" und uns davon überzeugen, dass die dabei für eine Anfrage Q der relationalen Algebra erzeugte CALC_{di}-Anfrage Q' tatsächlich zu CALC_{sr} gehört. Details: Übung! — Insbes. für **Fall 6** muss dabei Folgendes gezeigt werden: Wenn φ_{Q_2} in SRNf ist und $rr(\varphi_{Q_2}) = \mathrm{frei}(\varphi_{Q_2})$, dann gilt für die Formel $\psi := \mathrm{SRNF}(\neg\varphi_{Q_2})$, dass $rr(\psi) \neq \bot$ ist. $\square$

7.3 Statische Analyse und Auswertungskomplexität

Folie 267

Zur Erinnerung

Wunschliste für bessere Optimierung:

- zum "Löschen leerer Zwischenergebnisse":
  Test, ob eine gegebene (Teil-)Anfrage Q unerfüllbar ist ("$Q \equiv \emptyset$")
- zum "Löschen von Redundanzen":
  Test, ob zwei (Teil-)Anfragen Q und P äquivalent sind ("$Q \equiv P$")
- Wir kennen bereits: Algorithmen zum Lösen dieser Probleme für konjunktive Anfragen
- Jetzt: *Unentscheidbarkeit dieser Probleme für relationale Algebra*

Folie 268

Statische Analyse für Relational vollständige Sprachen

Satz 7.20.
Sei $\mathbf{S}$ *ein Datenbankschema, das mindestens ein Relationssymbol* R *enthält, dessen Stelligkeit* $\geqslant 2$ *ist. Dann sind die folgenden Probleme unentscheidbar:*

> ERFÜLLBARKEITSPROBLEM FÜR RELATIONALE ALGEBRA ÜBER $\mathbf{S}$
> *Eingabe:* Anfrage Q der relationalen Algebra über dem Datenbankschema $\mathbf{S}$
> *Frage:* Ist Q erfüllbar (d.h. gibt es mind. ein $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ mit $[\![Q]\!](\mathbf{I}) \neq \emptyset$) ?

> ÄQUIVALENZPROBLEM FÜR RELATIONALE ALGEBRA ÜBER $\mathbf{S}$
> *Eingabe:* Anfragen Q und P der rel. Algebra über dem DB-Schema $\mathbf{S}$
> *Frage:* Ist $Q \equiv P$ (d.h. gilt für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, dass $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$) ?

> QUERY CONTAINMENT PROBLEM FÜR REL. ALGEBRA ÜBER $\mathbf{S}$
> *Eingabe:* Anfragen Q und P der rel. Algebra über dem DB-Schema $\mathbf{S}$
> *Frage:* Ist $Q \sqsubseteq P$ (d.h. gilt für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, dass $[\![Q]\!](\mathbf{I}) \subseteq [\![P]\!](\mathbf{I})$) ?

Beweis: Einfache Folgerung aus dem Satz von Trakhtenbrot
(Theorem 7.11) und der Äquivalenz zwischen relationaler Algebra und dem Relationenkalkül. Details: *Übung.*

Folie 269

Auswertungsproblem:
Boolesche Anfragen $\rightsquigarrow$ beliebige Anfragen

Theorem 7.21. *Sei* $\mathbb{A}$ *ein Algorithmus, der das Auswertungsproblem für Boolesche Anfragen des Relationenkalküls löst.*

Dann gibt es einen Algorithmus $\mathbb{B}$*, der das Auswertungsproblem für (beliebige) Anfragen des Relationenkalküls unter Rückgriff auf* $\mathbb{A}$ *mit Taktung* $\mathcal{O}(k^3{\cdot}n{\cdot}\log n)$ *löst.*

Beweis: Identisch zum Beweis von Theorem 3.20.

Folgerung: Falls wir das Auswertungsproblem für *Boolesche* Anfragen des Relationenkalküls effizient lösen können, dann können wir es auch für *beliebige* Anfragen des Relationenkalküls effizient lösen.

Zur Erinnerung: Wir wissen bereits, dass das Auswertungsproblem bereits für Boolesche Anfragen des *konjunktiven Kalküls* NP-*vollständig* ist (Satz von Chandra und Merlin).

Folie 270

Die Komplexitätsklasse PSPACE ("polynomieller Platz")

Zur Erinnerung:

- Ein Entscheidungsproblem B gehört zur Klasse PSPACE, falls es eine (deterministische) Turingmaschine T und eine Konstante c gibt, so dass für jede Zahl N und jede zum Problem B passende Eingabe w der Größe N gilt:
  Die Berechnung von T bei Eingabe w benutzt $\leqslant N^c$ Bandzellen und endet mit der Ausgabe "ja", falls w eine "ja"-Instanz für B ist, bzw. mit der Ausgabe "nein", falls w eine "nein"-Instanz für B ist.
- Es gilt: $\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME}$.
- Ein Entscheidungsproblem B heißt PSPACE-*vollständig* (bzgl. Polynomialzeit-Reduktionen), falls gilt:
  (1) $B \in \text{PSPACE}$ und
  (2) B *ist* PSPACE-*hart*, d.h. für jedes Problem $A \in \text{PSPACE}$ gibt es eine *Polynomialzeit-Reduktion* f von A auf B (kurz: $f : A \leqslant_p B$)
- Eine *Polynomialzeit-Reduktion* f von A auf B ist eine (deterministisch) in polynomieller Zeit berechenbare Funktion, die jede zum Problem A passende Eingabe w auf eine zum Problem B passende Eingaben $f(w)$ abbildet, so dass gilt

  $$w \text{ ist eine "ja"-Instanz für } A \iff f(w) \text{ ist eine "ja"-Instanz für } B.$$

- Es gilt: $\big(A \text{ PSPACE-}\textit{hart und } A \leqslant_p B \big) \implies B \text{ PSPACE-}\textit{hart}.$

Folie 271

Auswertungskomplexität des Relationenkalküls

Theorem 7.22 (Chandra, Merlin, 1977). *Das Auswertungsproblem für Boolesche Anfragen des Relationenkalküls ist* PSPACE*-vollständig. (kombinierte Komplexität)*

Beweis: Siehe Tafel ...

Zum Nachweis der PSPACE-Härte benutzen wir das folgende Resultat *(hier ohne Beweis)*:

Theorem (Stockmeyer, Meyer, 1973) QBF *ist* PSPACE*-vollständig.*

Das Problem QBF ist dabei folgendermaßen definiert: *(QBF steht für "Quantified Boolean Formulas")*

QBF	
Eingabe:	Quantifizierte Aussagenlogische Formel ψ
Frage:	Hat ψ den Wert 1 ?

"Quantifizierte Aussagenlogische Formel" bedeutet:
ψ ist von der Form $\exists X_1 \, \forall X_2 \, \exists X_3 \cdots Q_m X_m \, \chi$ wobei $m \geqslant 1$, $Q_m = \exists$, falls m ungerade und $Q_m = \forall$, falls m gerade, χ eine Aussagenlogische Formel über den Variablen $X_1, \ldots, X_m$.
Die Formel ψ *hat den Wert 1, falls* gilt: Es gibt eine Belegung von X_1 (mit 0 oder 1), so dass es für jede Belegung von X_2 eine Belegung von X_3 gibt, so dass ... χ gilt.
(Andernfalls hat ψ den Wert 0.)

Folie 272

Datenkomplexität des Relationenkalküls

Datenkomplexität:

- *Blickwinkel:* Anfrage ist fest; die Eingabe besteht "nur" aus der Datenbank
- *Rechtfertigung:* i.d.R. ist die Anfrage kurz, die Datenbank aber sehr groß.
  Bei DB-Anwendungen in der Praxis außerdem oft: einige (wenige) Standard-Anfragen, die immer wieder gestellt werden (d.h.: Anfagen fest, die Einträge in der Datenbank können sich mit der Zeit aber ändern).

- *Notation:* Für jede feste Anfrage Q des Relationenkalküls (bzw. der relationalen Algebra) bezeichnet EVAL_Q das folgende Auswertungsproblem:

EVAL_Q
Eingabe: Datenbank $\mathbf{I}$ (vom zu Q "passenden" Datenbankschema)
Aufgabe: Berechne $[\![Q]\!](\mathbf{I})$.

- Für jede feste Anfrage Q der relationalen Algebra kann EVAL_Q *in polynomieller Zeit* gelöst werden (denn gemäß Proposition 6.4 in Zeit $(k+n)^{\mathcal{O}(k)}$, wobei $k := \|Q\|$)

- Dies lässt sich noch deutlich verbessern zur Aussage: EVAL_Q kann *in konstanter Zeit* gelöst werden auf *Parallelrechnern mit polynomiell vielen Prozessoren.*
  Genauer: EVAL_Q gehört zur Komplexitätsklasse AC^0 ...

Folie 273

Schaltkreise:

- Wir betrachten Schaltkreise mit $\wedge$, $\vee$, $\neg$-*Gattern*;
  wobei $\wedge$ und $\vee$-Gatter beliebig viele Eingänge haben dürfen

- *Zum "Zugriff" auf die Eingabe-Datenbank* $\mathbf{I}$ außerdem:
  Für jedes $R \in \mathbf{S}$, $r := \text{ar}(R)$ und alle $i_1, \ldots, i_r \in \{1, \ldots, |\operatorname{adom}(\mathbf{I})|\}$ ein mögliches *"Eingabe-Gatter"* der Form *"$R(i_1, \ldots, i_r)$"* zum Testen, ob das aus den entsprechenden Werten gebildete Tupel zur Relation $\mathbf{I}(R)$ gehört oder nicht.
  Analog auch für jedes $c \in \mathbf{dom}$ Eingabe-Gatter der Form *"$i{=}c$"* zum Testen, ob das i-te Element in $\operatorname{adom}(\mathbf{I})$ die Konstante c ist.

Definition 7.23 (AC^0 $\widehat{=}$"Probleme, die mit Schaltkreisen konst. Tiefe und polyn. Größe lösbar sind")**.** Sei Q eine Boolesche Anfrage an Datenbanken vom Schema $\mathbf{S}$.
Das Problem EVAL_Q gehört genau dann zur *Komplexitätsklasse* AC^0, wenn es eine Familie $(C_m)_{m \geqslant 1}$ von Schaltkreisen (der obigen Art) gibt, so dass gilt:

- Es gibt eine Zahl $d \in \mathbb{N}$, so dass für jedes $m \geqslant 1$ gilt: C_m hat die Tiefe $\leqslant d$.

- Es gibt eine Zahl $d' \in \mathbb{N}$, so dass für jedes $m \geqslant 1$ gilt:
  C_m besteht aus maximal $m^{d'}$ vielen Gattern.

- Für jedes $m \geqslant 1$ und jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ mit $|\operatorname{adom}(\mathbf{I})| = m$ gilt:

$$C_m \text{ akzeptiert Eingabe } \mathbf{I} \iff [\![Q]\!](\mathbf{I}) = \text{“ja”}$$

Folie 274

Theorem 7.24 (Immerman, 1987)**.** *Für jede Boolesche Anfrage Q des Relationenkalküls gehört* $\textsc{Eval}_Q$ *zu* AC^0

Beweis: Hier nur die Beweisidee (Details: *Übung*) anhand der Booleschen Anfrage

$$Q \;:=\; \Big\{\; () \;:\; \underbrace{\exists x\, \forall y\, \exists z\, \big(R(x,y) \;\land\; \neg z{=}c\big)}_{=:\,\varphi} \;\Big\}$$

Ansatz zur Konstruktion von C_m (auszuwerten über einer Datenbank $\mathbf{I}$ mit $|\operatorname{adom}(\mathbf{I})| = m$) :

$$\varphi \quad \rightsquigarrow \quad \bigvee_{i=1}^{m} \bigwedge_{j=1}^{m} \bigvee_{\ell=1}^{m} \Big(R(i,j) \;\land\; \neg \ell{=}c\Big) \quad \rightsquigarrow$$

Schaltkreis C_m (siehe Tafel)
der Tiefe $\leqslant \|\varphi\|$

7.4 Grenzen der Ausdrucksstärke

Folie 275

Beispiel: Frankfurter U-Bahn-Netz

Hier vereinfacht: Eine Relation *U-Bahn-Netz* mit Attributen *Linie*, *Halt*, *nächsterHalt*

U-Bahn-Netz

Linie	*Halt*	*nächsterHalt*
U4	Bockenheimer Warte	Festhalle/Messe
U4	Festhalle/Messe	Hauptbahnhof
U4	Hauptbahnhof	Willy-Brandt-Platz
U4	Willy-Brandt-Platz	Dom/Römer
...	...	...
U7	...	...
U7	Kirchplatz	Leipziger Str.
U7	Leipziger Str.	Bockenheimer Warte
U7	Bockenheimer Warte	Westend
...	...	...

Anfrage:
Gib alle Stationen aus, die von "Bockenheimer Warte" aus ohne Umsteigen zu erreichen sind.

(1) mit max. 1 Zwischenhalt:

$$\Big\{ (x_S) : \exists x_L \Big(\textit{U-Bahn-Netz}(x_L, \text{"Bockenheimer Warte"}, x_S) \lor \exists x_Z \big(\textit{U-Bahn-Netz}(x_L, \text{"Bockenheimer Warte"}, x_Z) \land \textit{U-Bahn-Netz}(x_L, x_Z, x_S) \big) \Big) \Big\}$$

(2) mit max. 2 Zwischenhalten: *analog*

(3) mit *beliebig vielen* Zwischenhalten *???*

Folie 276

Gaifman-Lokalität einer Anfrage

Definition 7.25. Sei $\mathbf{S}$ ein Datenbankschema, sei $r \geqslant 1$.
Eine *Anfrage* Q der Stelligkeit r heißt *Gaifman-lokal*, falls es ein $d \geqslant 0$ gibt, so dass für alle Datenbanken $\mathbf{I} \in \textit{inst}(\mathbf{S})$ und alle Tupel $a \in \mathbf{dom}^r$ und $b \in \mathbf{dom}^r$ gilt:
Falls $\mathcal{N}_d^{\mathbf{I},Q}(a) \cong \mathcal{N}_d^{\mathbf{I},Q}(b)$, *so* $\big(a \in [\![Q]\!](\mathbf{I}) \iff b \in [\![Q]\!](\mathbf{I})\big)$.

Notation hierbei:

- $\mathcal{N}_d^{\mathbf{I},Q}(a)$: die *d-Nachbarschaft* von $a = (a_1, \ldots, a_r)$, d.h.: die Datenbank $\mathbf{I}$, eingeschränkt auf die Elemente $N_d^{\mathbf{I},Q}(a)$, wobei $N_0^{\mathbf{I},Q}(a) := \{a_1, \ldots, a_r\} \cup \text{adom}(Q)$

$$N_{i+1}^{\mathbf{I},Q}(a) := N_i^{\mathbf{I},Q}(a) \cup \left\{ c \in \mathbf{dom} : \begin{array}{l} \text{es gibt ein Tupel } t \text{ in } \mathbf{I}\text{, in dem so-} \\ \text{wohl } c \text{ als auch mind. ein Element} \\ \text{aus } N_i^{\mathbf{I},Q}(a) \text{ vorkommt} \end{array} \right\}$$

- $\mathcal{N}_d^{\mathbf{I},Q}(a) \cong \mathcal{N}_d^{\mathbf{I},Q}(b)$: $\mathcal{N}_d^{\mathbf{I},Q}(a)$ ist isomorph zu $\mathcal{N}_d^{\mathbf{I},Q}(b)$, d.h. es gibt eine bijektive Abbildung f von $X := N_d^{\mathbf{I},Q}(a)$ nach $Y := N_d^{\mathbf{I},Q}(b)$, so dass gilt:
  - $f(a) = b$,
  - $f(c) = c$, für alle $c \in \text{adom}(Q)$,
  - für jedes $R \in \mathbf{S}$ und jedes Tupel $t \in X^{\text{ar}(R)}$ gilt:
    $t \in \mathbf{I}(R) \iff f(t) \in \mathbf{I}(R)$.

Folie 277

Gaifman-Lokalität des Relationenkalküls

Zur Erinnerung:
Eine *Anfrage* Q der Stelligkeit r heißt *Gaifman-lokal*, falls es ein $d \geqslant 0$ gibt, so dass für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und alle Tupel $a \in \mathbf{dom}^r$ und $b \in \mathbf{dom}^r$ gilt:
Falls $\mathcal{N}_d^{\mathbf{I},Q}(a) \cong \mathcal{N}_d^{\mathbf{I},Q}(b)$, *so* $\big(a \in [\![Q]\!](\mathbf{I}) \iff b \in [\![Q]\!](\mathbf{I})\big)$.

Theorem 7.26 (Gaifman-Lokalität (hier ohne Beweis))**.** *Jede Anfrage Q des Relationenkalküls* CALC_{adom} *ist Gaifman-lokal.*

Beweis: Siehe Vorlesung *„Logik und Komplexität“*.

Folie 278

Anwendung der Gaifman-Lokalität

Beispiel 7.27. Die Anfrage *“Gib alle Stationen aus, die von “Bockenheimer Warte” aus ohne Umsteigen zu erreichen sind”* kann nicht im Relationenkalkül (also auch nicht in der relationalen Algebra) beschrieben werden.

Beweis: Siehe Tafel.

Bemerkung:
Mit etwas anderen Methoden kann man auch zeigen, dass der Relationenkalkül *“nicht zählen kann”*. Zum Beispiel kann die Anfrage *“Hat Stephen Spielberg mehr Filme gedreht als Alfred Hitchcock?”* nicht im Relationenkalkül ausgedrückt werden.

Beweismethode:
Ehrenfeucht-Fraïssé-Spiele; siehe Vorlesung *„Logik in der Informatik“*.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Folie 279

Ausdrucksstärke von Anfragesprachen

- *stratifiziertes Datalog*$^\neg$
- *relational vollständige Sprachen:*
  relationale Algebra, nr-Datalog$^\neg$,
  Relationenkalkül (Varianten: active domain, bereichsunabhängig, safe-range)
- *Positive Anfragen:*
  SPCU, SPJRU, nr-Datalog
  positiver existentieller Kalkül PE-CALC$_{adom}$
- *konjunktive Anfragen:*
  SPC, SPJR, regelbasierte konjunktive Anfragen, konjunktiver Kalkül, Tableau-Anfragen
- *azyklische konjunktive Anfragen:*
  azyklische regelbasierte konjunktive Anfragen, Semijoin-Anfragen, konjunktives Guarded Fragment
- *Datalog*

Schematische Darstellung: siehe Tafel

Folie 280

Methoden zum Nachweis von Grenzen der Ausdrucksstärke einer Anfragesprache

- Monotonie von regelbasierten konjunktiven Anfragen und Datalog-Anfragen
- regelbasierte konjunktive Anfragen und Datalog-Anfragen Q sind abgeschlossen unter adom(Q)-Homomorphismen
- Gaifman-Lokalität von Anfragen des Relationenkalküls

Folie 281

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-*vollständig*	EXPTIME-*vollständig*	unent-scheidbar	unent-scheidbar	unent-scheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unent-scheidbar	unent-scheidbar	unent-scheidbar
Positive Anfragen	in AC^0	NP-*vollständig*	entscheidbar	*entscheidbar*	*entscheidbar*
Konjunktive Anfragen	in AC^0	NP-vollständig	in P	NP-vollständig	NP-vollständig
Azykl. Konj. Anfragen	in AC^0	in P *(LogCFL-vollst.)*	in P	in P	in P
Datalog	P-vollständig	EXPTIME-vollständig	entscheidbar	unent-scheidbar	unent-scheidbar

Folie 282

Wichtige Stichpunkte

- Homomorphismus-Satz
- Algorithmus zur Minimierung konjunktiver Anfragen
- Sätze von Chandra und Merlin
- Satz von Trakhtenbrot (und Folgerungen daraus)
- Satz von Knaster und Tarski

Folie 283

Funktionale Abhängigkeiten

- The Chase („Die Verfolgungsjagd“)
- Äquivalenz, Query Containment und Minimierung konjunktiver Anfragen
  unter Berücksichtigung funktionaler Abhängigkeiten
- effizienter Test, ob $\mathcal{F} \models f$, bei Eingabe einer FD-Menge $\mathcal{F}$ und einer FD f
- Armstrong-Kalkül

8.2 Ausblick auf weitere Themen

Folie 284

Einige weiterführende Themen

- Semistrukturierte Daten und XML
- Datenbanken mit unvollständiger Information
- Datenaustausch und Datenintegration
- Datenströme
- Constraint Datenbanken
- Probabilistische Datenbanken

Folie 285

– ENDE –