Vorlesung
Einführung in die Datenbanktheorie

Wintersemester

Prof. Dr. Nicole Schweikardt

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

Kapitel 1:

Einleitung

Abschnitt 1.1:
Einführung ins Thema

„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)

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

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

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)

Datenbanksysteme

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

Wünschenswerte Eigenschaften eines DBS

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)

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?

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

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

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

Wünschenswerte Eigenschaften eines DBS

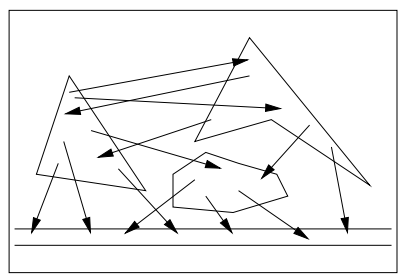
- Unterstützung eines Datenmodells: „Darstellung“ der Daten für den Zugriff
- Bereitstellung einer DB-Sprache:
  - zur Datendefinition: Data Definition Language (DDL)
  - zur Datenmanipulation und zum Datenzugriff:
    Data Manipulation Language (DML)
- Zugangskontrolle: Wer darf wann auf welche Daten 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

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

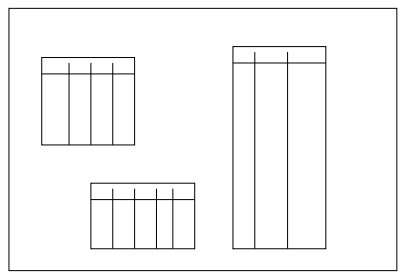
3-Schichten-Modell

Physische Schicht: Datenstrukturen, Speicherorganisation

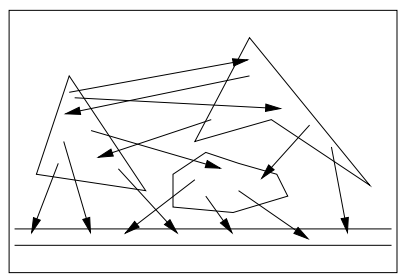


3-Schichten-Modell

Logische Schicht: Daten = Tabellen



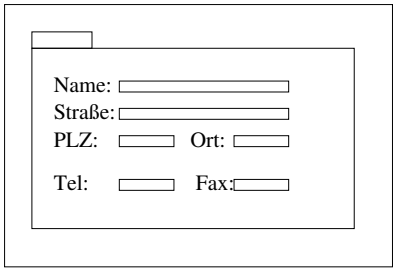
Physische Schicht: Datenstrukturen, Speicherorganisation



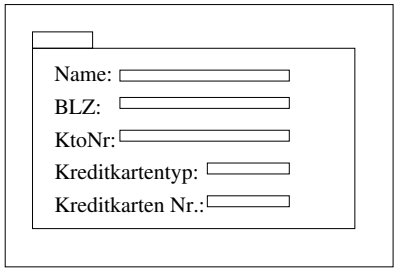
3-Schichten-Modell

Externe Schicht: Verschiedene Ansichten der Daten

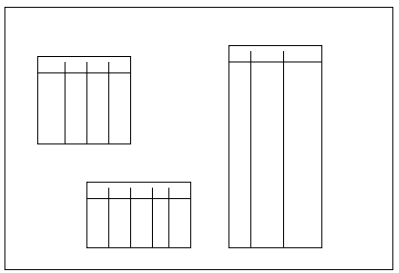
Ansicht 1:



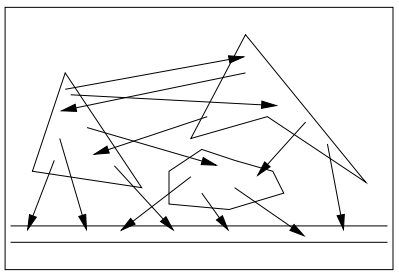
Ansicht 2:



Logische Schicht: Daten = Tabellen

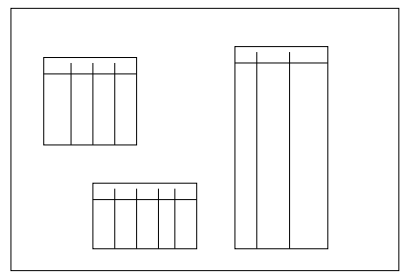


Physische Schicht: Datenstrukturen, Speicherorganisation



3-Schichten-Modell

Logische Schicht: Daten = Tabellen



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, nicht auf physische Schicht

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, nicht 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?

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, nicht 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

Hauptthema dieser Vorlesung: Anfragesprachen

Typische Fragestellungen für diese Vorlesung:

- Wie lassen sich deklarative Anfragen in ausführbare Operationen umsetzen?

$\rightsquigarrow$ Äquivalenz von „Kalkül“ und „Algebra“

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

$\rightsquigarrow$ Ausdrucksstärke von Anfragesprachen

- Wie aufwändig ist die Auswertung von Anfragen prinzipiell?

$\rightsquigarrow$ Auswertungskomplexität

- Wie lässt sich eine gegebene Anfrage möglichst effizient auswerten?

$\rightsquigarrow$ Anfrageoptimierung, statische Analyse (Erfüllbarkeit, Äquivalenz, . . .)

Inhaltsübersicht

Inhaltsübersicht

1. Einleitung
2. Das Relationale Modell
   - Datenmodell
   - Anfragen
   - Datenkomplexität und kombinierte Komplexität
3. Konjunktive Anfragen
   - Deskriptiver Ansatz: regelbasiert, graphisch und logikbasiert

Inhaltsübersicht

1. Einleitung
2. Das Relationale Modell
   - Datenmodell
   - Anfragen
   - Datenkomplexität und kombinierte Komplexität
3. Konjunktive Anfragen
   - Deskriptiver Ansatz: regelbasiert, graphisch und logikbasiert
   - Auswertungskomplexität

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

Inhaltsübersicht

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

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

Inhaltsübersicht

5. Relationale Algebra

- Definition und Beispiele
- Anfrageauswertung und Heuristische Optimierung
- Das Semijoin-Fragment der Relationalen Algebra

5. Relationale Algebra

- Definition und Beispiele
- Anfrageauswertung und Heuristische Optimierung
- Das Semijoin-Fragment der Relationalen Algebra

6. Relationenkalkül

- Syntax und Semantik
- Bereichsunabhängige Anfragen
- Äquivalenz zur Relationalen Algebra
- Auswertungskomplexität
- Statische Analyse

5. Relationale Algebra

- Definition und Beispiele
- Anfrageauswertung und Heuristische Optimierung
- Das Semijoin-Fragment der Relationalen Algebra

6. Relationenkalkül

- Syntax und Semantik
- Bereichsunabhängige Anfragen
- Äquivalenz zur Relationalen Algebra
- Auswertungskomplexität
- Statische Analyse

7. Funktionale Abhängigkeiten

- „The Chase“
- Anfrage-Optimierung unter Berücksichtigung funktionaler Abhängigkeiten
- der Armstrong-Kalkül

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.

Abschnitt 1.2:
Organisatorisches

Organisatorisches

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

Abschnitt 1.3:

Grundlegende Schreibweisen

- $\mathbb{N} := \{0, 1, 2, 3, \ldots\}$
- $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$
- 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

Abschnitt 2.1:

Datenmodell

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)

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).

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).

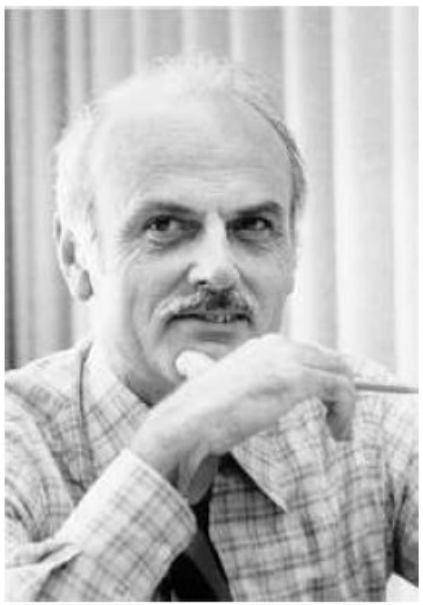
Im Folgenden wird eine präzise Definition des „Relationalen Modells“ gegeben.

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)

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit
$sorte(R) = \{A, B, C, D\}$,
$\text{ar}(R) = 4$

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit
$sorte(R) = \{A, B, C, D\}$,
$\text{ar}(R) = 4$

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von Attributnamen. Diese Menge sei geordnet via $\leqslant_{\textbf{att}}$.

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von Attributnamen. Diese Menge sei geordnet via $\leqslant_{\textbf{att}}$.
- Eine abzählbar unendliche Menge **dom** von „potentiellen Datenbankeinträgen“ („Konstanten“) (**dom** steht für „Domäne“ bzw. „Domain“)

Grundbegriff: Relationsschema

Beispiel-Relation:

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

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

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.

Grundbegriff: Relationsschema

Beispiel-Relation:

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

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

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.

Grundbegriff: Relationsschema

Beispiel-Relation:

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

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

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,

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von Attributnamen. Diese Menge sei geordnet via $\leqslant_{\textbf{att}}$.
- 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 $sorte : \textbf{rel} \to \mathcal{P}_e(\textbf{att})$, die jedem Relationsnamen eine endliche Menge von Attributnamen zuordnet,
  und zwar so, dass für alle $U \in \mathcal{P}_e(\textbf{att})$ gilt: $sorte^{-1}(U)$ ist unendlich.

  Notation: $\mathcal{P}_e(M)$ ist die Menge aller *endlichen* Teilmengen von M.

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von Attributnamen. Diese Menge sei geordnet via $\leqslant_{\textbf{att}}$.
- 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 $sorte : \textbf{rel} \to \mathcal{P}_e(\textbf{att})$, die jedem Relationsnamen eine endliche Menge von Attributnamen zuordnet,
  und zwar so, dass für alle $U \in \mathcal{P}_e(\textbf{att})$ gilt: $sorte^{-1}(U)$ 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.

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von Attributnamen. Diese Menge sei geordnet via $\leqslant_{\textbf{att}}$.
- 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 $sorte : \textbf{rel} \to \mathcal{P}_e(\textbf{att})$, die jedem Relationsnamen eine endliche Menge von Attributnamen zuordnet,
  und zwar so, dass für alle $U \in \mathcal{P}_e(\textbf{att})$ gilt: $sorte^{-1}(U)$ 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 $\text{ar}(R) := |sorte(R)|$.

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von Attributnamen. Diese Menge sei geordnet via $\leqslant_{\textbf{att}}$.
- 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 $sorte : \textbf{rel} \to \mathcal{P}_e(\textbf{att})$, die jedem Relationsnamen eine endliche Menge von Attributnamen zuordnet,
  und zwar so, dass für alle $U \in \mathcal{P}_e(\textbf{att})$ gilt: $sorte^{-1}(U)$ 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 $\text{ar}(R) := |sorte(R)|$.
- Ein Relationsschema ist einfach ein Relationsname R.

Grundbegriff: Relationsschema

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

Wir legen ein für alle Mal fest:

- Eine abzählbar unendliche Mengen **att** von Attributnamen. Diese Menge sei geordnet via $\leqslant_{\textbf{att}}$.
- 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 $sorte : \textbf{rel} \to \mathcal{P}_e(\textbf{att})$, die jedem Relationsnamen eine endliche Menge von Attributnamen zuordnet,
  und zwar so, dass für alle $U \in \mathcal{P}_e(\textbf{att})$ gilt: $sorte^{-1}(U)$ 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 $\text{ar}(R) := |sorte(R)|$.
- Ein Relationsschema ist einfach ein Relationsname R.
- Manchmal schreiben wir kurz $R[U]$ für $sorte(R) = U$ und $R[k]$ für $\text{ar}(R) = k$.

Relationsschema vs. Relation

Beispiel-Relation:

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

ist vom Schema R mit
$sorte(R) = \{A, B, C, D\}$,
$\text{ar}(R) = 4$

Relation $\widehat{=}$ Tabelle

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

Relationsschema vs. Relation

Beispiel-Relation:

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

ist vom Schema R mit
$sorte(R) = \{A, B, C, D\}$,
$\text{ar}(R) = 4$

Relation $\widehat{=}$ Tabelle

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

Definition 2.1

Sei R ein Relationsschema.

- Ein R-Tupel ist eine Abbildung $t : sorte(R) \to \mathbf{dom}$.

Relationsschema vs. Relation

Beispiel-Relation:

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

ist vom Schema R mit
$sorte(R) = \{A, B, C, D\}$,
$\text{ar}(R) = 4$

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“

Definition 2.1

Sei R ein Relationsschema.

- Ein R-Tupel ist eine Abbildung $t : sorte(R) \to \mathbf{dom}$.

Relationsschema vs. Relation

Beispiel-Relation:

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

ist vom Schema R mit
$sorte(R) = \{A, B, C, D\}$,
$\text{ar}(R) = 4$

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“

Definition 2.1

Sei R ein Relationsschema.

- Ein R-Tupel ist eine Abbildung $t : sorte(R) \to \mathbf{dom}$.
- Eine R-Relation ist eine endliche Menge von R-Tupeln.

Relationsschema vs. Relation

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

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 $t : sorte(R) \to \mathbf{dom}$.
- Eine R-Relation ist eine endliche Menge von R-Tupeln.

Relationsschema vs. Relation

Beispiel-Relation:

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

ist vom Schema R mit $sorte(R) = \{A, B, C, D\}$, $\text{ar}(R) = 4$

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 $t : sorte(R) \to \mathbf{dom}$.
- Eine R-Relation ist eine endliche Menge von R-Tupeln.
- $inst(R)$ bezeichnet die Menge aller R-Relationen.
  ($inst$ steht für „Instanzen“ bzw. „instances“)

Grundbegriffe: Datenbankschema und Datenbank

Definition 2.2

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

Grundbegriffe: Datenbankschema und Datenbank

Definition 2.2

- Ein Datenbankschema (kurz: DB-Schema) **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 **S** gehören.

Grundbegriffe: Datenbankschema und Datenbank

Definition 2.2

- Ein Datenbankschema (kurz: DB-Schema) **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 **S** gehören.

- Eine Datenbank (bzw. Datenbankinstanz) **I** vom Schema **S** ist eine Funktion, die jedem Relationsschema $R \in \mathbf{S}$ eine R-Relation zuordnet.

Grundbegriffe: Datenbankschema und Datenbank

Definition 2.2

- Ein Datenbankschema (kurz: DB-Schema) **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 **S** gehören.

- Eine Datenbank (bzw. Datenbankinstanz) **I** vom Schema **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 **S**.

Grundbegriffe: Datenbankschema und Datenbank

Definition 2.2

- Ein Datenbankschema (kurz: DB-Schema) **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 **S** gehören.

- Eine Datenbank (bzw. Datenbankinstanz) **I** vom Schema **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 **S**.

- $schema(\mathbf{I}) := \mathbf{S}$ bezeichnet das Schema der Datenbank **I**.

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)

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

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
...	...	...

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

Datenbankschema der Kinodatenbank:

- Datenbankschema

Datenbankschema der Kinodatenbank:

- Datenbankschema **KINO** $= \{\mathit{Kinos}, \mathit{Filme}, \mathit{Programm}\}$

Datenbankschema der Kinodatenbank:

- Datenbankschema **KINO** $= \{Kinos, Filme, Programm\}$
- $sorte(Kinos) =$

Datenbankschema der Kinodatenbank:

- Datenbankschema **KINO** $= \{\mathit{Kinos}, \mathit{Filme}, \mathit{Programm}\}$
- $\mathit{sorte}(\mathit{Kinos}) = \{\text{Name}, \text{Adresse}, \text{Stadtteil}, \text{Telefon}\}$

Datenbankschema der Kinodatenbank:

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

Datenbankschema der Kinodatenbank:

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

Datenbankschema der Kinodatenbank:

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

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.

Attribute: Benannte vs. Unbenannte Perspektive

Sind die Attributnamen Teil des expliziten Datenbankschemas?

In SQL: ja! Beispiel:

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

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“?

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: $t = (\mathrm{A} : \mathrm{a}, \quad \mathrm{B} : 1, \quad \mathrm{C} : \mathrm{z}, \quad \mathrm{D} : 9)$

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

Beispiel-Relation:

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

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

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)	$\mathbf{I}$, $\mathbf{J}$

Abschnitt 2.2:

Anfragen

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 ("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!

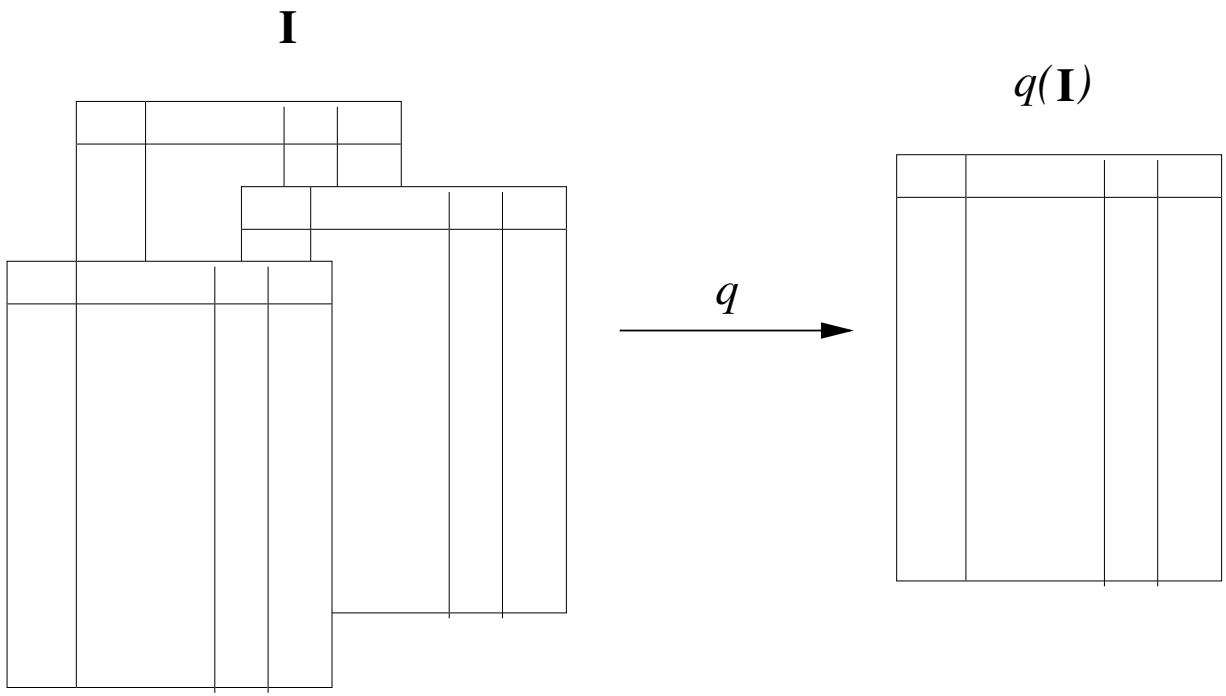
(14) Liste alle Schauspieler und den Regisseur von "Monuments Men" auf!

(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?

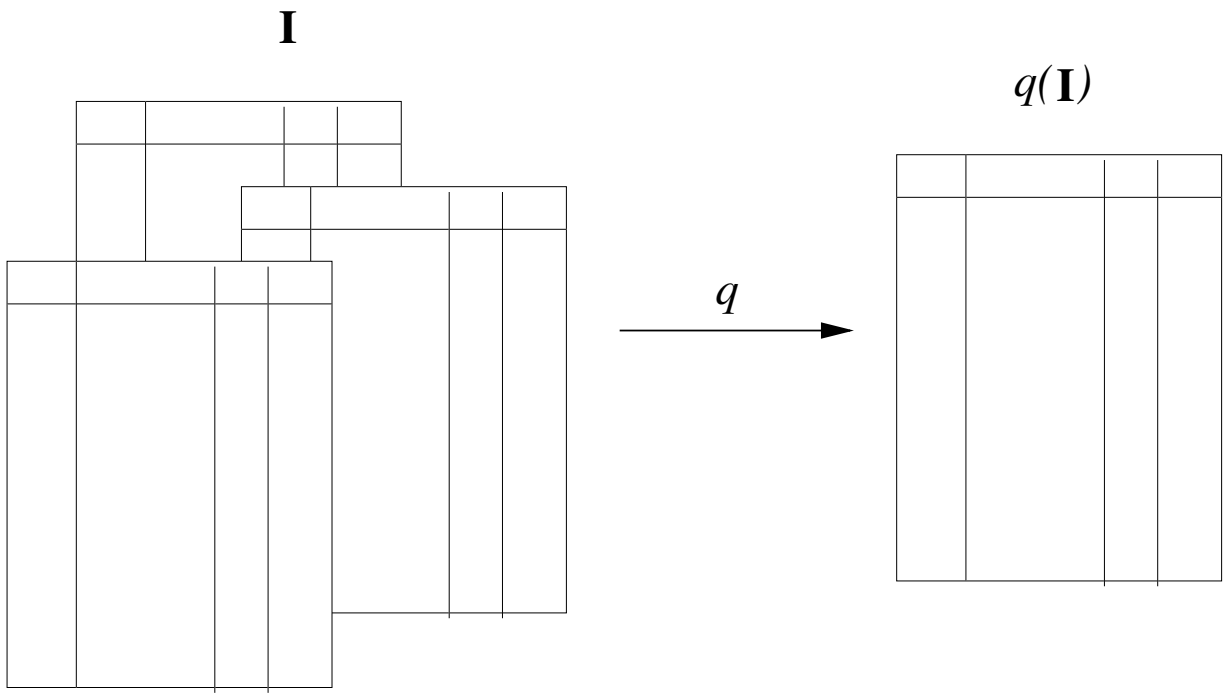
Anfragen und Anfragefunktionen



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“)

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.

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

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.

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“

Erläuterungen zu Eigenschaft (c)

Beispiel-Anfrage:

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

Eigenschaft (c):

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.

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,

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.

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 nicht explizit als "Konstanten" vorkommen, umbenannt, so sollen sie im Ergebnis auch umbenannt werden.*

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 nicht explizit als "Konstanten" vorkommen, umbenannt, so sollen sie im Ergebnis auch umbenannt werden.*

- Mathematische Präzisierung: Begriff der generischen Anfragefunktion

Generische Anfragefunktionen

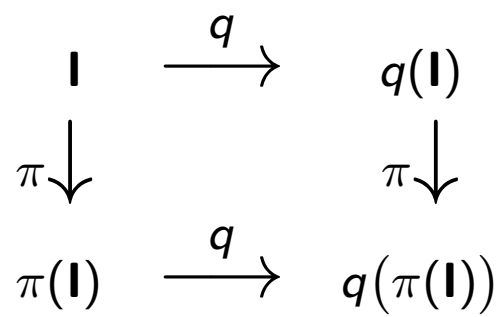
Definition 2.4

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 $\pi|_C = \mathrm{id}$ (d.h. $\pi(c) = c$ für alle $c \in C$) gilt:

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

Illustration:



Generische Anfragefunktionen

Definition 2.4

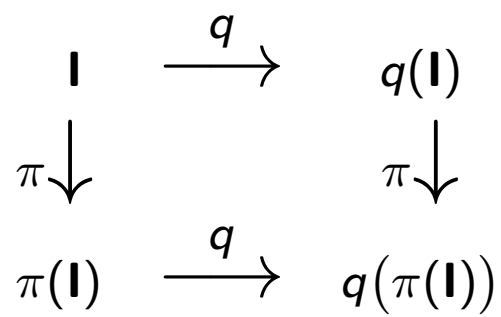
Sei C eine endliche Menge von Datenwerten (kurz: $C \subseteq_e \mathbf{dom}$).

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 $\pi|_C = \mathrm{id}$ (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:



Generische Anfragefunktionen

Definition 2.4

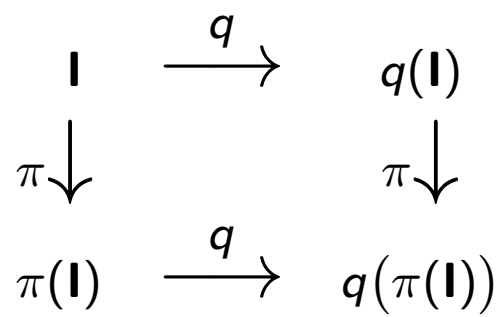
Sei C eine endliche Menge von Datenwerten (kurz: $C \subseteq_e \mathbf{dom}$).

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 $\pi|_C = \mathrm{id}$ (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:



Beispiel:

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

Generische Anfragefunktionen

Definition 2.4

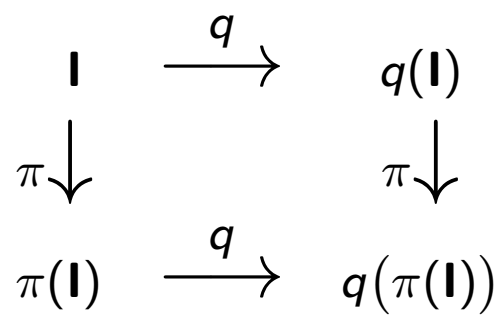
Sei C eine endliche Menge von Datenwerten (kurz: $C \subseteq_e \mathbf{dom}$).

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 $\pi|_C = \mathrm{id}$ (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:



Beispiel:

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

Generische Anfragefunktionen

Definition 2.4

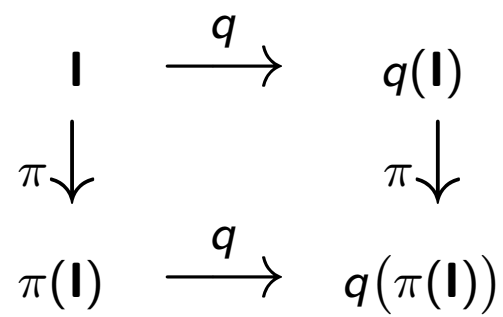
Sei C eine endliche Menge von Datenwerten (kurz: $C \subseteq_e \mathbf{dom}$).

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 $\pi|_C = \mathrm{id}$ (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:



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

Generische Anfragefunktionen

Definition 2.4

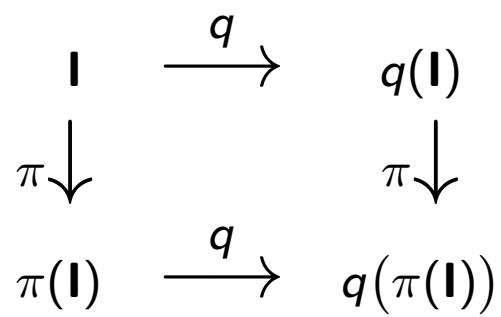
Sei C eine endliche Menge von Datenwerten (kurz: $C \subseteq_e \mathbf{dom}$).

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 $\pi|_C = \mathrm{id}$ (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:



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.

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

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“

Anfragesprachen

Dieselbe Anfragefunktion kann in verschiedenen Anfragesprachen beschrieben werden.

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

- SQL:

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
```

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:

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

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:

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

- Relationenkalkül:

$$\left\{ (x_{Kino}, x_{Adr}) \ : \ \exists x_T\, \exists x_R\, \exists x_Z\, \exists x_{St}\, \exists x_{Tel} \ \left(Filme(x_T, x_R, \text{“Matt Damon”}) \wedge Programm(x_{Kino}, x_T, x_Z) \wedge Kinos(x_{Kino}, x_{Adr}, x_{St}, x_{Tel}) \right) \right\}$$

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:

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

- Relationenkalkül:

$$\left\{ (x_{Kino}, x_{Adr}) \ : \ \exists x_T\, \exists x_R\, \exists x_Z\, \exists x_{St}\, \exists x_{Tel} \ \left(Filme(x_T, x_R, \text{“Matt Damon”}) \wedge Programm(x_{Kino}, x_T, x_Z) \wedge Kinos(x_{Kino}, x_{Adr}, x_{St}, x_{Tel}) \right) \right\}$$

- Relationale Algebra:

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:

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

- Relationenkalkül:

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

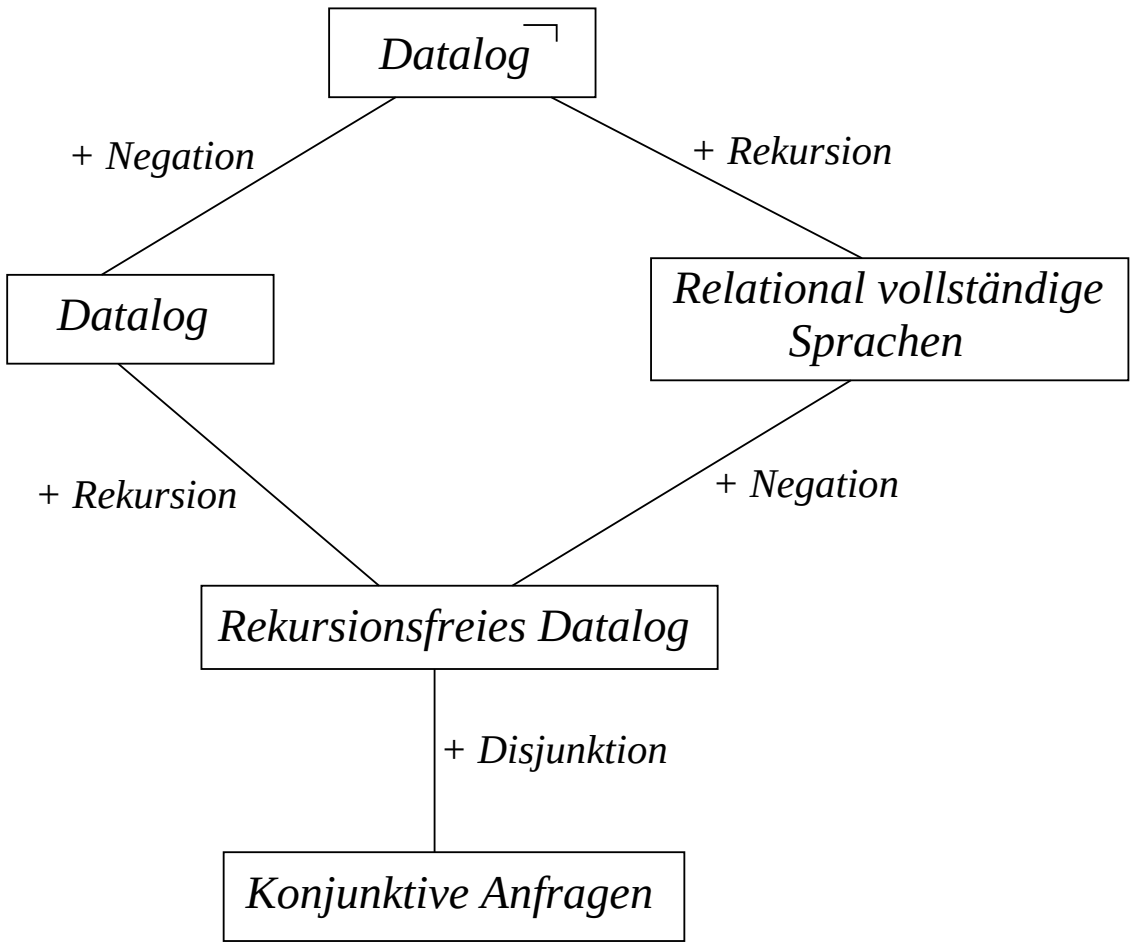
- Relationale Algebra:

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

Hierarchie der Anfragesprachen

Bemerkung: Anfragesprachen unterscheiden sich in ihrer Ausdrucksstärke.

Übersicht:



Abschnitt 2.3:

Datenkomplexität und kombinierte Komplexität

Typische Problemstellungen bzgl. Anfrageauswertung

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

Typische Problemstellungen bzgl. Anfrageauswertung

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

Typische Problemstellungen bzgl. Anfrageauswertung

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 $[\![Q]\!](\mathbf{I}) \neq \emptyset$?

Typische Problemstellungen bzgl. Anfrageauswertung

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?

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 $k = \|Q\|$

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 $n = \|\mathbf{I}\|$

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.

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.

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

Abschnitt 3.1:

Deskriptiver Ansatz: regelbasiert, graphisch und logikbasiert

Regelbasierte Konjunktive Anfragen — Informell

Beispiel-Anfrage:

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

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 $(x_{Titel}, x_{Regie}, \text{"Matt Damon"})$ 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

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 $(x_{Titel}, x_{Regie}, \text{"Matt Damon"})$ 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:

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

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, ...

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 $\mathbf{var} \cup \mathbf{dom}$.

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 $(\mathbf{var} \cup \mathbf{dom})^k$.

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 **S** ein Datenbankschema.
Eine regelbasierte konjunktive Anfrage über **S** ist ein Ausdruck der Form

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

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

Semantik regelbasierter konjunktiver Anfragen

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

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

Semantik regelbasierter konjunktiver Anfragen

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

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

- Mit $\mathrm{var}(Q)$ bezeichnen wir die Menge aller Variablen, die in Q vorkommen.
- Eine Belegung für Q ist eine Abbildung $\beta : \mathrm{var}(Q) \to \mathbf{dom}$.

Semantik regelbasierter konjunktiver Anfragen

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

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

- 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 $u = (e_1, \ldots, e_k)$ setzen wir $\beta(u) := (\beta(e_1), \ldots, \beta(e_k))$.

Semantik regelbasierter konjunktiver Anfragen

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

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

- 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 $u = (e_1, \ldots, e_k)$ setzen wir $\beta(u) := (\beta(e_1), \ldots, \beta(e_k))$.
- Der Anfrage Q ordnen wir die folgende Anfragefunktion $[\![Q]\!]$ zu:

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

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

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:

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:

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

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:

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.

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 $[\![Q_1]\!](\mathbf{I}) =$

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

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 $[\![Q_2]\!](\mathbf{I}) =$

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

Bezeichnungen

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

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

Bezeichnungen

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

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

- $Ans(u)$ wird als Kopf der Regel bezeichnet.

Bezeichnungen

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

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

- $Ans(u)$ wird als Kopf der Regel bezeichnet.
- $R_1(u_1), \ldots, R_\ell(u_\ell)$ wird als Rumpf der Regel bezeichnet.

Bezeichnungen

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

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

- $Ans(u)$ 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.

Bezeichnungen

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

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

- $Ans(u)$ 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.

Bezeichnungen

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

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

- $Ans(u)$ wird als Kopf der Regel bezeichnet.
- $R_1(u_1), \ldots, R_\ell(u_\ell)$ wird als Rumpf der Regel bezeichnet.
- Die Relationsnamen aus **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 $adom(Q)$ bezeichnen wir die Menge aller Konstanten (also Elemente aus **dom**), die in Q vorkommen.

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

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: adom($\mathbf{I}$), ist die Menge aller Elemente aus **dom**, die in $\mathbf{I}$ vorkommen. D.h.:

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

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

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: adom($\mathbf{I}$), ist die Menge aller Elemente aus **dom**, die in $\mathbf{I}$ vorkommen. D.h.:

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

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

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

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

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: $\mathrm{adom}(\mathbf{I})$, ist die Menge aller Elemente aus **dom**, die in $\mathbf{I}$ vorkommen. D.h.:

$$\mathrm{adom}(\mathbf{I}) \;=\; \bigcup_{R \in \mathbf{S}} \mathrm{adom}\left(\mathbf{I}(R)\right)$$

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

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

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

Proposition 3.4

Für jede regelbasierte konjunktive Anfrage Q und jede Datenbank $\mathbf{I}$ (vom passenden DB-Schema) gilt: $\mathrm{adom}\left([\![Q]\!](\mathbf{I})\right) \;\subseteq\; \mathrm{adom}(Q, \mathbf{I})$.

Beweis: siehe Tafel.

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).

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 $q(\mathbf{I}) \neq \emptyset$.

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: siehe Tafel.

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.

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!

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.

„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: $\big(\mathrm{T},\ (x_{Kino},\ x_{Adr})\big)$

Tableaus — Präzise

Definition 3.7

Sei **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 $\text{ar}(R)$.

  (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“.)

Tableaus — Präzise

Definition 3.7

Sei **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 $\text{ar}(R)$.

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

  (D.h. ein Tableau über **S** ist eine „Datenbank vom Schema **S**, die als Einträge auch Variablen enthalten kann“.)

Tableaus — Präzise

Definition 3.7

Sei **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 $\text{ar}(R)$.

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

  (D.h. ein Tableau über **S** ist eine „Datenbank vom Schema **S**, die als Einträge auch Variablen enthalten kann“.)

- Eine Tableau-Anfrage über **S** (bzw. R) ist von der Form (T, u), wobei T ein Tableau über **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).

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.

Semantik von Tableau-Anfragen

Sei $Q = (T, u)$ 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.

Semantik von Tableau-Anfragen

Sei $Q = (T, u)$ eine Tableau-Anfrage.

- $\text{var}(Q)$ bezeichnet die Menge aller Variablen, die in u oder T vorkommen.
  $\text{adom}(Q)$ bezeichnet die Menge aller Konstanten, die in u oder T vorkommen.

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

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.

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 $\beta(\mathsf{T})$).

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

Logikbasierte Variante: Konjunktiver Kalkül

Beispiel-Anfrage:

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

Formulierung als Anfrage des konjunktiven Kalküls:

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

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.

Konjunktiver Kalkül (CQ) — Präzise

Definition 3.8

Sei **S** ein Datenbankschema.
Die Menge CQ[**S**] aller Formeln des konjunktiven Kalküls über **S** ist induktiv wie folgt definiert: (CQ steht für „conjunctive queries“)

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

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

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

Konjunktiver Kalkül (CQ) — Präzise

Definition 3.8

Sei **S** ein Datenbankschema.
Die Menge CQ[**S**] aller Formeln des konjunktiven Kalküls über **S** ist induktiv wie folgt definiert: (CQ steht für „conjunctive queries“)

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

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

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

Insbesondere: Jede Formel in CQ[**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).

Semantik von CQ[**S**]

Sei φ eine CQ[**S**]-Formel.

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

Semantik von CQ[**S**]

Sei φ eine CQ[**S**]-Formel.

- $\mathrm{adom}(\varphi)$ bezeichnet die Menge aller Konstanten (also Elemente aus **dom**), die in φ vorkommen. $\mathrm{var}(\varphi)$ bezeichnet die Menge aller Variablen (also Elemente aus **var**), die in φ vorkommen.

Semantik von CQ[**S**]

Sei φ eine CQ[**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.: $\text{frei}(R(v_1, \ldots, v_r)) = \{v_1, \ldots, v_r\} \cap \mathbf{var}$; $\text{frei}((\varphi \wedge \psi)) = \text{frei}(\varphi) \cup \text{frei}(\psi)$;
  $\text{frei}(\exists x \varphi) = \text{frei}(\varphi) \setminus \{x\}$.

Semantik von CQ[**S**]

Sei φ eine CQ[**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 $\beta : \text{frei}(\varphi) \to \mathbf{dom}$.

Semantik von CQ[**S**]

Sei φ eine CQ[**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 **I** vom Schema **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 $\sigma := \mathbf{S} \cup \mathbf{dom}$.

Semantik von CQ[**S**]

Sei φ eine CQ[**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 **I** vom Schema **S** ordnen wir die logische Struktur

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

  zu. Insbesondere ist $\mathcal{A}_{\mathbf{I}}$ eine σ-Struktur über der Signatur $\sigma := \mathbf{S} \cup \mathbf{dom}$.
- Ist **I** eine Datenbank vom Schema **S** und β eine Belegung für φ, so sagen wir „**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.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**S**]-Formeln für alle Datenbankschemata **S**.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**S**]-Formeln für alle Datenbankschemata **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. $i \in \{1, \ldots, r\}$.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**S**]-Formeln für alle Datenbankschemata **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 $\text{frei}(\varphi) \subseteq \{x_1, \ldots, x_r\}$.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**S**]-Formeln für alle Datenbankschemata **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 $\mathbf{I} \models \varphi[\{x_1/a_1, \ldots, x_r/a_r\}]$.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**S**]-Formeln für alle Datenbankschemata **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.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**S**]-Formeln für alle Datenbankschemata **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 „$\exists x_1 \, \exists x_2 \cdots \exists x_n$“.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**S**]-Formeln für alle Datenbankschemata **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 $\mathrm{frei}(\varphi) \subseteq \{x_1, \ldots, x_r\}$.
- 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 CQ[**S**]-Formeln φ und ψ heißen äquivalent, falls $\mathrm{frei}(\varphi) = \mathrm{frei}(\psi)$ und für jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ und jede Belegung β für φ (und ψ) gilt: $\mathbf{I} \models \varphi[\beta] \iff \mathbf{I} \models \psi[\beta]$.

Konjunktiver Kalkül: Syntax und Semantik

Definition 3.9

Sei **S** ein Datenbankschema.

Eine Anfrage des konjunktiven Kalküls ist von der Form

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

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

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 $\varphi \in \mathrm{CQ}[\mathbf{S}]$, $r \geqslant 0$ 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:

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

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

Wertebereich von Anfragen des konjunktiven Kalküls

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

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

Wertebereich von Anfragen des konjunktiven Kalküls

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

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

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

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

Wertebereich von Anfragen des konjunktiven Kalküls

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

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

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

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

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: $\mathsf{adom}\big([\![Q]\!](\mathbf{I})\big) \ \subseteq \ \mathsf{adom}(Q, \mathbf{I})$.

Beweis: Induktion über den Formelaufbau. Details: Übung.

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:

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:

$$\Big\{ () \;:\; \exists x_{Schauspieler} \Big(\exists x_{Titel1}\ \mathit{Filme}(x_{Titel1}, \text{“Paul Anderson”}, x_{Schauspieler}) \wedge \exists x_{Titel2}\ \mathit{Filme}(x_{Titel2}, \text{“Ridley Scott”}, x_{Schauspieler}) \Big) \Big\}$$

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:

$$\Big\{ () \;:\; \exists x_{Schauspieler} \Big(\exists x_{Titel1}\ \mathit{Filme}(x_{Titel1}, \text{“Paul Anderson”}, x_{Schauspieler}) \wedge \exists x_{Titel2}\ \mathit{Filme}(x_{Titel2}, \text{“Ridley Scott”}, x_{Schauspieler}) \Big) \Big\}$$

und durch die dazu äquivalente Anfrage

$$\Big\{ () \;:\; \exists x_{Schauspieler}\ \exists x_{Titel1}\ \exists x_{Titel2} \Big(\mathit{Filme}(x_{Titel1}, \text{“Paul Anderson”}, x_{Schauspieler}) \wedge \mathit{Filme}(x_{Titel2}, \text{“Ridley Scott”}, x_{Schauspieler}) \Big) \Big\}$$

Eine Normalform für CQ

Definition 3.11

Eine CQ[**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 Normalform für CQ

Definition 3.11

Eine CQ[**S**]-Formel $\varphi(x_1, \ldots, x_r)$ ist in Normalform, falls sie von der Form

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

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.

Eine Normalform für CQ

Definition 3.11

Eine CQ[**S**]-Formel $\varphi(x_1, \ldots, x_r)$ ist in Normalform, falls sie von der Form

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

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.

Eine Normalform für CQ

Definition 3.11

Eine CQ[**S**]-Formel $\varphi(x_1, \ldots, x_r)$ ist in Normalform, falls sie von der Form

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

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.

Beispiel

Die CQ-Formel

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

ist nicht in Normalform, aber äquivalent zur Normalform-Formel

Beispiel

Die CQ-Formel

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

ist nicht in Normalform, aber äquivalent zur Normalform-Formel

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

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

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.

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 $\mathcal{Q}_2 \leqslant \mathcal{Q}_1$

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 nicht $\mathcal{Q}_2 \leqslant \mathcal{Q}_1$.

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.

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: siehe Tafel.

Beispiel zur Übersetzung von Anfragen

Betrachte die Anfrage

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

Beispiel zur Übersetzung von Anfragen

Betrachte die Anfrage

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

Geben Sie eine umgangssprachliche Beschreibung der Anfrage!

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:

siehe Tafel

Beispiel zur Übersetzung von Anfragen

Betrachte die Anfrage

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

Geben Sie eine umgangssprachliche Beschreibung der Anfrage!

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:

siehe Tafel

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

siehe Tafel

Beispiel zur Übersetzung von Anfragen

Betrachte die Anfrage

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

Geben Sie eine umgangssprachliche Beschreibung der Anfrage!

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:

siehe Tafel

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

siehe Tafel

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

Die Kalkül-Anfrage

$$\left\{ (x,y) \;:\; \begin{array}{l} \exists z \exists z' \; \mathit{Programm}(z,x,z') \wedge \\ \exists z \; \mathit{Filme}(x,y,z) \wedge \\ \exists z \; \mathit{Filme}(z,y,\text{“George Clooney”}) \wedge \\ \exists z \; \mathit{Filme}(z_5,\text{“George Clooney”},y) \end{array} \right\}$$

die nicht in Normalform ist, wird durch unseren Beweis zunächst in die Normalform-Anfrage

Die Kalkül-Anfrage

$$\left\{ (x,y) \;:\; \begin{array}{l} \exists z \exists z' \; \mathit{Programm}(z,x,z') \wedge \\ \exists z \; \mathit{Filme}(x,y,z) \wedge \\ \exists z \; \mathit{Filme}(z,y,\text{“George Clooney”}) \wedge \\ \exists z \; \mathit{Filme}(z_5,\text{“George Clooney”},y) \end{array} \right\}$$

die nicht in Normalform ist, wird durch unseren Beweis zunächst in die Normalform-Anfrage

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

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

Einige Erweiterungen der Anfragesprachen

(1) Test auf „Gleichheit“ von Variablen zulassen

(2) Hintereinanderausführung (Komposition) mehrerer Anfrage zulassen

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

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

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

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

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

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

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 $c \in \mathbf{dom}$.

$\rightsquigarrow$ regelbasierte konjunktive Anfragen mit „=“

$\rightsquigarrow$ Konjunktiver Kalkül mit „=“

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 $c \in \mathbf{dom}$.

$\rightsquigarrow$ regelbasierte konjunktive Anfragen mit „=“

$\rightsquigarrow$ 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

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

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 $c \in \mathbf{dom}$.

$\rightsquigarrow$ regelbasierte konjunktive Anfragen mit „=“

$\rightsquigarrow$ 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.

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 $c \in \mathbf{dom}$.

$\rightsquigarrow$ regelbasierte konjunktive Anfragen mit „=“

$\rightsquigarrow$ 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.

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

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.

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.

$\rightsquigarrow$ bereichsbeschränkte regelbasierte konjunktive Anfragen mit „=“

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.

$\rightsquigarrow$ bereichsbeschränkte regelbasierte konjunktive Anfragen mit „=“

Analog wird die Klasse $\mathrm{CQ}^{=}$ aller bereichsbeschränkten Formeln des konjunktiven Kalküls mit „=“ definiert.

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.

$\rightsquigarrow$ bereichsbeschränkte regelbasierte konjunktive Anfragen mit „=“

Analog wird die Klasse $\mathrm{CQ}^{=}$ aller bereichsbeschränkten Formeln des konjunktiven Kalküls mit „=“ definiert.

Beobachtung 3.15

Jede $\mathrm{CQ}^{=}$*-Formel ist entweder unerfüllbar oder äquivalent zu einer* CQ*-Formel. (Details siehe Übung.)*

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.

$\rightsquigarrow$ bereichsbeschränkte regelbasierte konjunktive Anfragen mit „=“

Analog wird die Klasse $\mathrm{CQ}^{=}$ aller bereichsbeschränkten Formeln des konjunktiven Kalküls mit „=“ definiert.

Beobachtung 3.15

Jede $\mathrm{CQ}^{=}$*-Formel ist entweder unerfüllbar oder äquivalent zu einer* CQ*-Formel. (Details siehe Übung.)*

Beispiel für eine $\mathrm{CQ}^{=}$-Anfrage, die nicht erfüllbar ist:

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.

$\rightsquigarrow$ bereichsbeschränkte regelbasierte konjunktive Anfragen mit „=“

Analog wird die Klasse $\mathrm{CQ}^{=}$ aller bereichsbeschränkten Formeln des konjunktiven Kalküls mit „=“ definiert.

Beobachtung 3.15

Jede $\mathrm{CQ}^{=}$*-Formel ist entweder unerfüllbar oder äquivalent zu einer* CQ*-Formel. (Details siehe Übung.)*

Beispiel für eine $\mathrm{CQ}^{=}$-Anfrage, die nicht erfüllbar ist:

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

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

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.

Regelbasierte konjunktive Programme

Definition 3.16

Sei **S** ein Datenbankschema.

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

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

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:

$$Q_i \;:=\; S_i(u_i) \leftarrow \text{Rumpf}_i$$

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

Regelbasierte konjunktive Programme

Definition 3.16

Sei **S** ein Datenbankschema.
Ein regelbasiertes konjunktives Programm über **S** (mit oder ohne „=“) hat die Form

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

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:

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

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).

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

Äquivalenz von Regeln und Programmen

Beobachtung 3.17

Für jedes regelbasierte konjunktive Programm P über einem Relationsschema $\mathbf{S}$ (mit oder ohne „=“) und jedes idb-Prädikat S von P gibt es eine regelbasierte konjunktive Anfrage Q (mit „=“) über $\mathbf{S}$, so dass

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

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

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

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

Die von P durch S_2 definierte Anfrage ist äquivalent zu

Äquivalenz von Regeln und Programmen

Beobachtung 3.17

Für jedes regelbasierte konjunktive Programm P über einem Relationsschema $\mathbf{S}$ (mit oder ohne „=“) und jedes idb-Prädikat S von P gibt es eine regelbasierte konjunktive Anfrage Q (mit „=“) über $\mathbf{S}$, so dass

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

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

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

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

Die von P durch S_2 definierte Anfrage ist äquivalent zu

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

Abschnitt 3.2:
Auswertungskomplexität

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.

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.

Größe der Eingabe: $k + n$, wobei

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

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 $\mathbf{dom} \cup \mathbf{var} \cup \mathbf{S} \cup \{\exists, \wedge, (,), \{, \}, :, , \}$)
- $n := \|\mathbf{I}\|$ 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“

Auswertung konjunktiver Anfragen

Proposition 3.18

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

Beweis: siehe Tafel.

Auswertung konjunktiver Anfragen

Proposition 3.18

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

Beweis: siehe Tafel.

Bemerkung: Das ist exponentiell in der Länge der Anfrage.

Frage: Geht das effizienter?

Boolesche Anfragen

- Zur Erinnerung:

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

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.

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.

Algorithmen mit Verzögerung $f(k, n)$

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 Verzögerung $f(k, n)$ gelöst werden,

falls

Algorithmen mit Verzögerung $f(k, n)$

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 Verzögerung $f(k, n)$ 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

Algorithmen mit Verzögerung $f(k, n)$

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 Verzögerung $f(k, n)$ 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

- 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.

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 Verzögerung $\mathcal{O}(k^3 \cdot n)$ löst.

Beweis: siehe Tafel.

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.

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

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

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“.

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.

NP-Vollständigkeit

Zur Erinnerung:

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

NP-Vollständigkeit

Zur Erinnerung:

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

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

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

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: $f : A \leqslant_p B$)

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.

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

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.

  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.

  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.

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

- **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.

- **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:

  $$x \text{ ist eine „ja“-Instanz für das Problem } A \iff f(x) \text{ ist eine „ja“-Instanz für das Problem } B$$

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

- **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:

  $$x \text{ ist eine „ja“-Instanz für das Problem } A \iff f(x) \text{ ist eine „ja“-Instanz für das Problem } B$$

  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.

- **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:

  $$x \text{ ist eine „ja“-Instanz für das Problem } A \iff f(x) \text{ ist eine „ja“-Instanz für das Problem } B$$

  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:

- **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:

  $$\begin{array}{l} x \text{ ist eine „ja“-Instanz} \\ \text{für das Problem } A \end{array} \quad \Longleftrightarrow \quad \begin{array}{l} f(x) \text{ ist eine „ja“-Instanz} \\ \text{für das Problem } B \end{array}$$

  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:
  $f(G, k) =$

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

- **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:

  $$\begin{array}{l} x \text{ ist eine „ja“-Instanz} \\ \text{für das Problem } A \end{array} \quad \Longleftrightarrow \quad \begin{array}{l} f(x) \text{ ist eine „ja“-Instanz} \\ \text{für das Problem } B \end{array}$$

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

- f heißt **Reduktion** von A auf B.

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

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

- 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.

- 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.

- Wenn es eine Polynomialzeit-Reduktion f von A auf B gibt, wissen wir, dass Folgendes gilt:

- 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.

- 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.

- 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.

- 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.

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: $f : A \leqslant_p B$)

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

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

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

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

  Anschaulich bedeutet $A \leqslant_p B$, dass A „höchstens so schwer“ wie B ist. NP-vollständige Probleme sind also „die schwierigsten Probleme“ in NP.

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

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

  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 $B \in \mathrm{P}$ $\Longrightarrow$

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

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

  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 $B \in \mathrm{P} \quad \Longrightarrow \quad A \in \mathrm{P}$

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

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

  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 $A \leqslant_p B \implies$

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

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

  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 $A \leqslant_p B \implies B \notin \mathrm{P}$

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

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

  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 $A \leqslant_p B \implies$

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

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

  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.

Auswertungskomplexität konjunktiver Anfragen

Theorem 3.21 (Chandra, Merlin, 1977)

Das Auswertungsproblem (kombinierte Komplexität) für Boolesche regelbasierte konjunktive Anfragen ist NP*-vollständig.*

Beweis: siehe Tafel ...

Auswertungskomplexität konjunktiver Anfragen

Theorem 3.21 (Chandra, Merlin, 1977)

Das Auswertungsproblem (kombinierte Komplexität) für Boolesche regelbasierte konjunktive Anfragen ist NP*-vollständig.*

Beweis: siehe Tafel ...

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.*

Auswertungskomplexität konjunktiver Anfragen

Theorem 3.21 (Chandra, Merlin, 1977)

Das Auswertungsproblem (kombinierte Komplexität) für Boolesche regelbasierte konjunktive Anfragen ist NP*-vollständig.*

Beweis: siehe Tafel ...

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.

Folgerung aus der NP-Vollständigkeit

Folgerung:

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

Folgerung aus der NP-Vollständigkeit

Folgerung:

Falls $\mathrm{P} \neq \mathrm{NP}$, 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 Parametrischen Komplexitätstheorie lässt sich Folgendes zeigen:

Folgerung aus der NP-Vollständigkeit

Folgerung:

Falls $\mathrm{P} \neq \mathrm{NP}$, 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 Parametrischen Komplexitätstheorie lässt sich Folgendes zeigen:

> *Theorem (Papadimitriou, Yannakakis, 1997)*
>
> Falls $\mathrm{FPT} \neq \mathrm{W}[1]$, 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).

Folgerung aus der NP-Vollständigkeit

Folgerung:

Falls $\mathrm{P} \neq \mathrm{NP}$, 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 Parametrischen Komplexitätstheorie lässt sich Folgendes zeigen:

> *Theorem (Papadimitriou, Yannakakis, 1997)*
>
> Falls $\mathrm{FPT} \neq \mathrm{W}[1]$, 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 $\mathrm{W}[1]$ sind Komplexitätsklassen, die in der parametrischen Komplexitätstheorie Rollen spielen, die in etwa mit den Rollen von P und NP in der klassischen Komplexitätstheorie vergleichbar sind.

Abschnitt 3.3:

Algebraischer Ansatz: SPC-Algebra und SPJR-Algebra

Algebraische Anfragen — Informell

Beispiel-Anfrage:

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

Als Anfrage in der SPJR-Algebra:

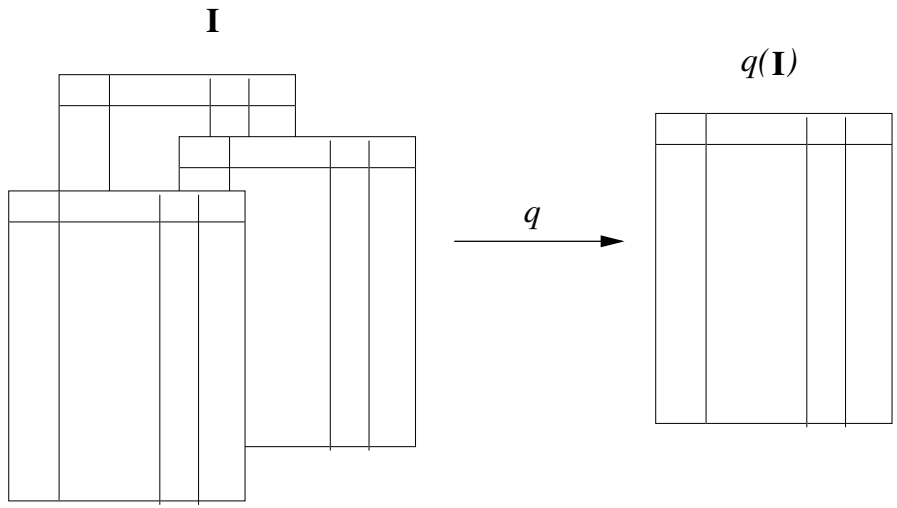
$$\pi_{Kino,Adresse}\Big(\sigma_{\substack{Schauspieler = \\ \text{"Matt Damon"}}}\big(Filme \bowtie Programm \bowtie \delta_{\text{Name} \mapsto \text{Kino}} Kinos\big)\Big)$$

Als Anfrage in der SPC-Algebra:

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

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

Unbenannte Perspektive: Die SPC-Algebra

Selektion: Zwei Varianten:

- Operator $\sigma_{j=a}$, für eine Konstante $a \in \mathbf{dom}$ 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:

$$\sigma_{j=a}(I) := \{t \in I : \text{in der } j\text{-ten Komponente von } t \text{ steht ein } a\}$$

Unbenannte Perspektive: Die SPC-Algebra

Selektion: Zwei Varianten:

- Operator $\sigma_{j=a}$, für eine Konstante $a \in \mathbf{dom}$ 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:

$$\sigma_{j=k}(I) := \left\{ t \in I : \begin{array}{l} \text{in der } j\text{-ten Komponente von } t \text{ steht derselbe Eintrag} \\ \text{wie in der } k\text{-ten Komponente von } t \end{array} \right\}$$

Unbenannte Perspektive: Die SPC-Algebra

Selektion: Zwei Varianten:

- Operator $\sigma_{j=a}$, für eine Konstante $a \in \mathbf{dom}$ 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:

$$\sigma_{j=k}(I) := \left\{ t \in I : \begin{array}{l} \text{in der } j\text{-ten Komponente von } t \text{ steht derselbe Eintrag} \\ \text{wie in der } k\text{-ten Komponente von } t \end{array} \right\}$$

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

„$j{=}a$“ und „$j{=}k$“ werden Selektionsbedingungen genannt.

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 $\geqslant \max\{j_1, \ldots, j_k\}$ und liefert als Ausgabe die folgende Relation der Stelligkeit k:

$$\pi_{j_1,\ldots,j_k}(I) \;:=\; \Big\{ \big(t(j_1), \ldots, t(j_k) \big) \;:\; t \in I \Big\}$$

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

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 $m+n$:

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

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 $m+n$:

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

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 $\big(t(1), \ldots, t(m),\ s(1), \ldots, s(n)\big)$ zu bezeichnen.

Definition 3.23

Sei **S** ein Datenbankschema. Die Klasse der Anfragen der SPC-Algebra über **S** (kurz: SPC[**S**]) ist induktiv wie folgt definiert:

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

Definition 3.23

Sei **S** ein Datenbankschema. Die Klasse der Anfragen der SPC-Algebra über **S** (kurz: SPC[**S**]) ist induktiv wie folgt definiert:

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

Die Semantik $[\![Q]\!]$ von SPC[**S**]-Anfragen Q ist induktiv auf die offensichtliche Art definiert.

Beispiele für Anfragen der SPC-Algebra

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

Beispiele für Anfragen der SPC-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!

Beispiele für Anfragen der SPC-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!

Beispiele für Anfragen der SPC-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!
- (5) Läuft zur Zeit ein Film von "James Cameron"?

Beispiele für Anfragen der SPC-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!
- (5) Läuft zur Zeit ein Film von "James Cameron"?

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

Beispiele für Anfragen der SPC-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!
- (5) Läuft zur Zeit ein Film von "James Cameron"?

  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 $\mathbf{I}$: – siehe Tafel –

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 $i \in \{1, \ldots, n\}$.

Eine Normalform für SPC-Anfragen

Sei **S** ein Relationsschema.

Definition 3.24

Eine SPC[**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.

Eine Normalform für SPC-Anfragen

Sei **S** ein Relationsschema.

Definition 3.24

Eine SPC[**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 SPC[**S**]*-Anfrage Q gibt es eine* SPC[**S**]*-Anfrage Q' in Normalform, die dieselbe Anfragefunktion definiert; und es gibt einen Polynomialzeit-Algorithmus, der Q' bei Eingabe von Q erzeugt.*

Beweis: Übung.

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:

$$\sigma_{A=a}(I) \;:=\; \{\; t \in I \quad : \quad t(A) = a \;\}$$

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:

$$\sigma_{A=B}(I) := \{ t \in I : t(A) = t(B) \}$$

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.

Projektion:

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

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

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

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

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

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?

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

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

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

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 $\Sigma := sorte(R) \cup sorte(S)$:

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

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 $\Sigma := sorte(R) \cup sorte(S)$:

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

Wir benutzen den Operator manchmal auch für einzelne Tupel:

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

Umbenennung (Renaming):

Operator δ_f, für eine Umbenennungsfunktion f, d.h. eine injektive Funktion $f : U \to \mathbf{att}$, 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 $f(U) \cup \big(\mathit{sorte}(R) \setminus U\big)$:

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

Umbenennung (Renaming):

Operator δ_f, für eine Umbenennungsfunktion f, d.h. eine injektive Funktion $f : U \to \mathbf{att}$, 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.

Definition 3.26

Sei **S** ein Datenbankschema. Die Klasse der Anfragen der SPJR-Algebra über **S** (kurz: SPJR[**S**]) ist induktiv wie folgt definiert:

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

Definition 3.26

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

- Für alle Relationsnamen $R \in \mathbf{S}$ ist R eine SPJR[$\mathbf{S}$]-Anfrage der Sorte $\mathit{sorte}(R)$.
- Für alle Konstanten $c \in \mathbf{dom}$ und alle Attributnamen $A \in \mathbf{att}$ ist $\{(A : c)\}$ eine SPJR[$\mathbf{S}$]-Anfrage der Sorte $\{A\}$.
- Ist Q eine SPJR[$\mathbf{S}$]-Anfrage der Sorte Σ, sind $A, B \in \Sigma$ und ist $a \in \mathbf{dom}$ eine Konstante, so sind auch $\sigma_{A=a}(Q)$ und $\sigma_{A=B}(Q)$ SPJR[$\mathbf{S}$]-Anfragen der Sorte Σ.
- Ist Q eine SPJR[$\mathbf{S}$]-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[$\mathbf{S}$]-Anfrage der Sorte $\{A_1, \ldots, A_k\}$.
- Sind Q und P zwei SPJR[$\mathbf{S}$]-Anfragen der Sorten Σ und Π, so ist $(Q \bowtie P)$ eine SPJR[$\mathbf{S}$]-Anfrage der Sorte $\Sigma \cup \Pi$.
- Ist Q eine SPJR[$\mathbf{S}$]-Anfrage der Sorte Σ und ist $f : \Sigma \to \mathbf{att}$ eine Umbenennungsfunktion, so ist $\delta_f(Q)$ eine SPJR[$\mathbf{S}$]-Anfrage der Sorte $f(\Sigma)$.

Die Semantik $[\![Q]\!]$ von SPJR[$\mathbf{S}$]-Anfragen Q ist induktiv auf die offensichtliche Art definiert.

Beispiele für Anfragen der SPJR-Algebra

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

Beispiele für Anfragen der SPJR-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!

Beispiele für Anfragen der SPJR-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!

Beispiele für Anfragen der SPJR-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!
- (5) Läuft zur Zeit ein Film von "James Cameron"?

Beispiele für Anfragen der SPJR-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!
- (5) Läuft zur Zeit ein Film von "James Cameron"?

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

Beispiele für Anfragen der SPJR-Algebra

- (4) Welche Kinos (Name & Adresse) spielen einen Film mit "Matt Damon"?
- (9) Egal, wie die DB aussieht, gib immer ("Terminator", "Linda Hamilton") aus!
- (3) Gib Adresse und Telefonnummer des "Kino International" aus!
- (5) Läuft zur Zeit ein Film von "James Cameron"?

  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 $\mathbf{I}$: – siehe Tafel –

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 $i \in \{1, \ldots, n\}$.

Eine Normalform für SPJR-Anfragen

Sei **S** ein Relationsschema.

Definition 3.27

Eine SPJR[**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 $\delta_{f_1}(R_1), \ldots, \delta_{f_\ell}(R_\ell)$ paarweise disjunkt sind und keins der $A_1, \ldots, A_m$ als Attribut von einem der $\delta_{f_j}(R_j)$ vorkommt.

Eine Normalform für SPJR-Anfragen

Sei **S** ein Relationsschema.

Definition 3.27

Eine SPJR[**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 $\delta_{f_1}(R_1), \ldots, \delta_{f_\ell}(R_\ell)$ 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 SPJR[**S**]*-Anfrage Q gibt es eine* SPJR[**S**]*-Anfrage Q' in Normalform, die dieselbe Anfragefunktion definiert; und es gibt einen Polynomialzeit-Algorithmus, der Q' bei Eingabe von Q erzeugt.*

Beweis: Übung.

Nicht-Erfüllbare Anfragen

Bemerkung:

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

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” und $c =$ “George Clooney”.

Solche Tupel kann es aber nicht geben!

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

Somit ist die Anfrage Q nicht erfüllbar.

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.

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: siehe Tafel.

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: siehe Tafel.

Beispiel zur Übersetzung von Anfragen

Betrachte die Anfrage $Q :=$

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

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

— siehe Tafel —

Abschnitt 3.4:

Homomorphismus-Satz, Statische Analyse und Anfrageminimierung

Vorbemerkung zum Thema „Statische Analyse, Optimierung“

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

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

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: $[\![Q]\!](\mathbf{I}) \subseteq [\![P]\!](\mathbf{I})$.

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

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

Statische Analyse:

- Äquivalenzproblem: Bei Eingabe zweier Anfragen Q, P teste, ob $Q \equiv P$.

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen „Q ist in P enthalten“, falls für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$ 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 $Q \sqsubseteq P$.

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen „Q ist in P enthalten“, falls für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$ 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).

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen „Q ist in P enthalten“, falls für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$ 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 regelbas. konj. Anfragen ist das Erfüllbarkeitsproblem

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen „Q ist in P enthalten“, falls für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$ 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 regelbas. konj. Anfragen ist das Erfüllbarkeitsproblem trivial (Satz 3.6).

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen „Q ist in P enthalten“, falls für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$ 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 regelbas. konj. Anfragen ist das Erfüllbarkeitsproblem trivial (Satz 3.6).
- Für konj. Anfragen mit „=“ (bzw. für SPC- bzw. SPJR-Anfragen) ist das Erfüllbarkeitsproblem

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen „Q ist in P enthalten“, falls für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$ 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 regelbas. konj. Anfragen ist das Erfüllbarkeitsproblem trivial (Satz 3.6).
- Für konj. Anfragen mit „=“ (bzw. für SPC- bzw. SPJR-Anfragen) ist das Erfüllbarkeitsproblem in poly. Zeit lösbar (siehe Übungsaufgaben).

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 $\mathbf{I} \in inst(\mathbf{S})$ gilt: $[\![Q]\!](\mathbf{I}) = [\![P]\!](\mathbf{I})$.

(b) Wir schreiben $Q \sqsubseteq P$ und sagen „Q ist in P enthalten“, falls für alle Datenbanken $\mathbf{I} \in inst(\mathbf{S})$ 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 regelbas. konj. Anfragen ist das Erfüllbarkeitsproblem trivial (Satz 3.6).
- Für konj. Anfragen mit „=“ (bzw. für SPC- bzw. SPJR-Anfragen) ist das Erfüllbarkeitsproblem in poly. Zeit lösbar (siehe Übungsaufgaben).
- Jetzt: Algorithmen für's Query Containment Problem und für's Äquivalenzproblem für konjunktive Anfragen.

Zusammenhänge:

- Äquivalenz vs. Containment:
  - $Q \equiv P \quad \Longleftrightarrow \quad Q \sqsubseteq P$ und $P \sqsubseteq Q$

Zusammenhänge:

- Äquivalenz vs. Containment:
  - $Q \equiv P \iff Q \sqsubseteq P$ und $P \sqsubseteq Q$
  - $Q \sqsubseteq P \iff (Q \vee P) \equiv P$

Zusammenhänge:

- Äquivalenz vs. Containment:
  - $Q \equiv P \iff Q \sqsubseteq P$ und $P \sqsubseteq Q$
  - $Q \sqsubseteq P \iff (Q \vee P) \equiv P$ ..... für Optimierung nutzen!

Zusammenhänge:

- Äquivalenz vs. Containment:
  - $Q \equiv P \iff Q \sqsubseteq P$ und $P \sqsubseteq Q$
  - $Q \sqsubseteq P \iff (Q \vee P) \equiv P$ ..... für Optimierung nutzen!

- Containment vs. Erfüllbarkeit:
  - Q unerfüllbar $\implies Q \sqsubseteq P$ für *alle* Anfragen P

Zusammenhänge:

- Äquivalenz vs. Containment:
  - $Q \equiv P \iff Q \sqsubseteq P$ und $P \sqsubseteq Q$
  - $Q \sqsubseteq P \iff (Q \vee P) \equiv P$ . . . . . für Optimierung nutzen!

- Containment vs. Erfüllbarkeit:
  - Q unerfüllbar $\implies Q \sqsubseteq P$ für *alle* Anfragen P
  - $Q \sqsubseteq P \iff (Q \wedge \neg P)$ ist unerfüllbar . . . . . für Optimierung nutzen!

Zusammenhänge:

- Äquivalenz vs. Containment:
  - $Q \equiv P \iff Q \sqsubseteq P$ und $P \sqsubseteq Q$
  - $Q \sqsubseteq P \iff (Q \vee P) \equiv P$ ..... für Optimierung nutzen!

- Containment vs. Erfüllbarkeit:
  - Q unerfüllbar $\implies Q \sqsubseteq P$ für *alle* Anfragen P
  - $Q \sqsubseteq P \iff (Q \wedge \neg P)$ ist unerfüllbar ..... für Optimierung nutzen!

- Erfüllbarkeit vs. Äquivalenz:
  - Q unerfüllbar $\implies (Q \vee P) \equiv P$ für *alle* Anfragen P
    ..... für Optimierung nutzen!

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:

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

Als Tableau-Anfrage:

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:

$$\begin{aligned} \mathit{Ans}(x_T, x_R) \quad \leftarrow \quad & \mathit{Programm}(x_K, x_T, x_Z), \\ & \mathit{Filme}(x_T, x_R, x_S), \\ & \mathit{Filme}(y_T, x_R, \text{“Sandra Bullock”}) \end{aligned}$$

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”

Homomorphismen

Definition 3.32

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

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

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 $h(M) := \{\, h(t) \;:\; t \in M \,\}$.

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.: $h\big(\mathrm{var}(Q')\big) \subseteq \mathrm{var}(Q) \cup \mathrm{adom}(Q)$.

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

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: $h : x, y, x_1, y_1 \mapsto x, y, x, y$.

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'.

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

(„repräsentiere Var. in T durch Konstanten, die nicht in Q, Q' vorkommen“)

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 $\alpha|_{\mathbf{dom}} = \mathrm{id}$.

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

(„repräsentiere Var. in T durch Konstanten, die nicht in Q, Q' vorkommen“)

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“ $\alpha_C^{-1} : \mathbf{dom} \to \mathbf{var} \cup \mathbf{dom}$

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

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

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

(„repräsentiere Var. in T durch Konstanten, die nicht in Q, Q' vorkommen“)

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“ $\alpha_C^{-1} : \mathbf{dom} \to \mathbf{var} \cup \mathbf{dom}$

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

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

Definition 3.33 (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 $C := \mathrm{adom}(Q) \cup \mathrm{adom}(Q')$ 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 $R \in \mathbf{S}$.

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

(„repräsentiere Var. in T durch Konstanten, die nicht in Q, Q' vorkommen“)

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“ $\alpha_C^{-1} : \mathbf{dom} \to \mathbf{var} \cup \mathbf{dom}$

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

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

Definition 3.33 (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 $C := \mathrm{adom}(Q) \cup \mathrm{adom}(Q')$ 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: Siehe Tafel.

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

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 $d \in \mathbf{dom}$ ist.

Um die kanonischen Datenbanken und die kanonischen Tupel zu Q und Q' zu bilden, betrachten wir

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 $d \in \mathbf{dom}$ ist.

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

Kanonische Tupel:

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 $d \in \mathbf{dom}$ ist.

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

Kanonische Tupel: $u^Q_{Q'} = (a_x, a_y)$ und $u^{Q'}_Q = (a_x, a_y)$.

Kanonische Datenbanken:

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 $d \in \mathbf{dom}$ ist.

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

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 $d \in \mathbf{dom}$ ist.

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

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 $d \in \mathbf{dom}$ ist.

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

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 $d \in \mathbf{dom}$ ist.

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

Um herauszufinden, ob $Q' \sqsubseteq Q$ ist,

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 $d \in \mathbf{dom}$ ist.

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

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.

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:*

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

Beweis: Siehe Tafel.

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:*

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

Beweis: Siehe Tafel.

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: Siehe Tafel.

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:*

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

Beweis: Siehe Tafel.

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: Siehe Tafel.

Bemerkung: Wegen $\big(Q \equiv Q' \iff (Q \sqsubseteq Q' \text{ und } Q' \sqsubseteq Q)\big)$ 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.

Tableau-Minimierung

Definition 3.37

Sei **S** ein Datenbankschema.

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

Tableau-Minimierung

Definition 3.37

Sei **S** ein Datenbankschema.

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

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

Tableau-Minimierung

Definition 3.37

Sei **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 $\pi : \mathrm{var}(Q') \to \mathrm{var}(Q)$ 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.*

Tableau-Minimierung

Definition 3.37

Sei **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 $\pi : \mathrm{var}(Q') \to \mathrm{var}(Q)$ 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.*

Tableau-Minimierung

Definition 3.37

Sei **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 $\pi : \mathrm{var}(Q') \to \mathrm{var}(Q)$ 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): siehe Tafel; (b): Übung.

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.*

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.*

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): siehe Tafel; (b): Übung.

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 $T' \subseteq T$ 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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

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 $\mathrm{T}' \subseteq \mathrm{T}$ 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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

- $\mathbf{S} = \{R\}$, wobei R die Attribute A, B, C hat.
- $Q := \quad \pi_{A,B}\big(\sigma_{B=\text{“5”}}(R)\big) \;\bowtie\; \pi_{B,C}\Big(\pi_{A,B}(R) \bowtie \pi_{A,C}\big(\sigma_{B=\text{“5”}}(R)\big)\Big)$

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 $T' \subseteq T$ 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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

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

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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

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

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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

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

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 $\mathrm{T}' \subseteq \mathrm{T}$ 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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

- $\mathbf{S} = \{R\}$, wobei R die Attribute A, B, C hat.
- $Q := \pi_{A,B}\big(\sigma_{B=\text{“5”}}(R)\big) \bowtie \pi_{B,C}\Big(\pi_{A,B}(R) \bowtie \pi_{A,C}\big(\sigma_{B=\text{“5”}}(R)\big)\Big)$
- zugehörige Tableau-Anfrage: (T, u) mit $u = (x_A, 5, z_C)$ und $\mathrm{T}(R) = \{\ (x_A, 5, x_C),\ (y_A, 5, y_C),\ (y_A, 5, z_C)\ \}$
- minimales Tableau T':

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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

- $\mathbf{S} = \{R\}$, wobei R die Attribute A, B, C hat.
- $Q := \pi_{A,B}\big(\sigma_{B=\text{“5”}}(R)\big) \bowtie \pi_{B,C}\Big(\pi_{A,B}(R) \bowtie \pi_{A,C}\big(\sigma_{B=\text{“5”}}(R)\big)\Big)$
- zugehörige Tableau-Anfrage: (T, u) mit $u = (x_A, 5, z_C)$ und $\mathsf{T}(R) = \{\ (x_A, 5, x_C),\ (y_A, 5, y_C),\ (y_A, 5, z_C)\ \}$
- minimales Tableau T': $\mathsf{T}'(R) = \{\ (x_A, \text{“5”}, x_C),\ (y_A, \text{“5”}, z_C)\ \}$

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 $\mathrm{T}' \subseteq \mathrm{T}$ 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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

- $\mathbf{S} = \{R\}$, wobei R die Attribute A, B, C hat.
- $Q := \pi_{A,B}\big(\sigma_{B=\text{“5”}}(R)\big) \bowtie \pi_{B,C}\Big(\pi_{A,B}(R) \bowtie \pi_{A,C}\big(\sigma_{B=\text{“5”}}(R)\big)\Big)$
- zugehörige Tableau-Anfrage: (T, u) mit $u = (x_A, 5, z_C)$ und $\mathrm{T}(R) = \{\ (x_A, 5, x_C),\ (y_A, 5, y_C),\ (y_A, 5, z_C)\ \}$
- minimales Tableau T': $\mathrm{T}'(R) = \{\ (x_A, \text{“5”}, x_C),\ (y_A, \text{“5”}, z_C)\ \}$
- zugehörige SPJR-Anfrage:

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 $\mathrm{T}' \subseteq \mathrm{T}$ 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 **??**) auf Q' an und werte Q' aus.

Minimierung des Tableaus $\widehat{=}$ Minimierung der Anzahl der Joins,
denn: Anzahl Zeilen im Tableau $= 1 +$ Anzahl Join-Operationen bei der Auswertung

Beispiel 3.40

- $\mathbf{S} = \{R\}$, wobei R die Attribute A, B, C hat.
- $Q := \pi_{A,B}\big(\sigma_{B=\text{“5”}}(R)\big) \bowtie \pi_{B,C}\Big(\pi_{A,B}(R) \bowtie \pi_{A,C}\big(\sigma_{B=\text{“5”}}(R)\big)\Big)$
- zugehörige Tableau-Anfrage: (T, u) mit $u = (x_A, 5, z_C)$ und $\mathrm{T}(R) = \{\ (x_A, 5, x_C),\ (y_A, 5, y_C),\ (y_A, 5, z_C)\ \}$
- minimales Tableau T': $\mathrm{T}'(R) = \{\ (x_A, \text{“5”}, x_C),\ (y_A, \text{“5”}, z_C)\ \}$
- zugehörige SPJR-Anfrage: $Q' := \pi_{A,B}\big(\sigma_{B=\text{“5”}}(R)\big) \bowtie \pi_{B,C}\big(\sigma_{B=\text{“5”}}(R)\big)$.

Abschnitt 3.5:
Azyklische Anfragen

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)

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

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 $Q_1 :=$

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

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 $Q_1 :=$

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

Anfrage 2: Gibt es einen Studenten, der an einem Kurs teilnimmt, der von seinem/r Vater/Mutter veranstaltet wird?

Als regelbasierte konjunktive Anfrage $Q_2 :=$

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

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 $Q_2 :=$

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

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.

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 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) Weg zwischen $R_i(u_i)$ und $R_j(u_j)$ liegt.

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 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) 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.

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 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) 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:

- Join-Baum für $Q_1 := Ans() \leftarrow E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, y_K, y_Z)$
- Es gibt keinen Join-Baum für
  $Q_2 := Ans() \leftarrow E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, x_K, x_Z)$
- Join-Baum für $Q_3 :=$
  $Ans() \leftarrow R(y, z),\ P(x, y),\ S(y, z, u),\ S(z, u, w),\ T(y, z),\ T(z, u),\ R(z', y')$

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 $\mathcal{O}(\|Q'\| \cdot \|\mathbf{I}\| \cdot \log(\|\mathbf{I}\|))$)

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

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 $R \in \mathbf{S}$, $r := \text{ar}(R)$ 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 $[\![R(v_1, \ldots, v_r)]\!](\mathbf{I}) :=$

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

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 $R \in \mathbf{S}$, $r := \text{ar}(R)$ 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 $[\![R(v_1, \ldots, v_r)]\!](\mathbf{I}) :=$

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

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

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

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 $R \in \mathbf{S}$, $r := \text{ar}(R)$ 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 $[\![R(v_1, \ldots, v_r)]\!](\mathbf{I}) :=$

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

(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.

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: siehe Tafel □

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.*

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.*

Beweis: (a): Übung.

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.

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 Tafel. □

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 Tafel. □

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

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 Tafel. □

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.

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.

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

(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".

Einige Details zum Beweis von Lemma 3.45

Beispiel: Probelauf des Algorithmus für die Anfragen

- $Q_1 := \; Ans() \leftarrow E(x_P, x_S),\; D(x_P, x_K, x_Z),\; T(x_S, y_K, y_Z)$
- $Q_2 := \; Ans() \leftarrow E(x_P, x_S),\; D(x_P, x_K, x_Z),\; T(x_S, x_K, x_Z)$
- $Q_3 := \; Ans() \leftarrow R(y, z),\; P(x, y),\; S(y, z, u),\; S(z, u, w),\; T(y, z),\; T(z, u),\; R(z', y')$

Einige Details zum Beweis von Lemma 3.45

Beispiel: Probelauf des Algorithmus für die Anfragen

- $Q_1 := \mathit{Ans}() \leftarrow E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, y_K, y_Z)$
- $Q_2 := \mathit{Ans}() \leftarrow E(x_P, x_S),\ D(x_P, x_K, x_Z),\ T(x_S, x_K, x_Z)$
- $Q_3 := \mathit{Ans}() \leftarrow R(y, z),\ P(x, y),\ S(y, z, u),\ S(z, u, w),\ T(y, z),\ T(z, u),\ R(z', y')$

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

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.

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.

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.

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.

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).

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. □

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.*

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:

- **Algo für Boolesche Anfragen:**

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:

- **Algo für Boolesche Anfragen:** Nutze Lemma 3.45, Lemma 3.44 (b) und Proposition 3.43.

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:

- **Algo für Boolesche Anfragen:** Nutze Lemma 3.45, Lemma 3.44 (b) und Proposition 3.43.
- **Algo für Anfragen beliebiger Stelligkeit:**

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:

- **Algo für Boolesche Anfragen:** Nutze Lemma 3.45, Lemma 3.44 (b) und Proposition 3.43.
- **Algo für Anfragen beliebiger Stelligkeit:** Nutze Algo 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. □

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:

- **Algo für Boolesche Anfragen:** Nutze Lemma 3.45, Lemma 3.44 (b) und Proposition 3.43.
- **Algo für Anfragen beliebiger Stelligkeit:** Nutze Algo 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. □

Bemerkung: Es ist bekannt, dass das Auswertungsproblem für Boolesche azyklische regelbasierte konjunktive Anfragen vollständig ist für die Komplexitätsklasse LogCFL (Gottlob, Leone, Scarcello, 1998).

Konjunktives Guarded Fragment GF(CQ)

Definition 3.47

Sei $\mathbf{S}$ ein Datenbankschema.

Mit GF(CQ)[$\mathbf{S}$] bezeichnen wir die Menge aller Formeln des konjunktiven Kalküls CQ[$\mathbf{S}$] (vgl. Definition 3.8), die zum Guarded Fragment GF[$\mathbf{S}$] gehören, d.h. ... (Details: siehe Tafel)

Konjunktives Guarded Fragment GF(CQ)

Definition 3.47

Sei $\mathbf{S}$ ein Datenbankschema.

Mit GF(CQ)[$\mathbf{S}$] bezeichnen wir die Menge aller Formeln des konjunktiven Kalküls CQ[$\mathbf{S}$] (vgl. Definition 3.8), die zum Guarded Fragment GF[$\mathbf{S}$] gehören, d.h. ... (Details: siehe Tafel)

Konjunktive Sätze des Guarded Fragment sind Formeln aus GF(CQ)[$\mathbf{S}$], die keine freie(n) Variable(n) besitzen.

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.*

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.

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.

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) $\Longleftrightarrow$ (b): Lemma 3.44. (a) $\Longleftrightarrow$ (c): Übung. □

Abschnitt 3.6:

Mengen-Semantik vs. Multimengen-Semantik

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):

$$\{t\} \neq \{t, t\} \neq \{t, t, t\} \neq \cdots$$

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

$\mathbf{I}_F(\mathit{Hersteller})$

Name	*Ort*
Boeing	Seattle
Boeing	New York
Airbus	Hamburg

Notation:

- $|\,(\text{Boeing,Seattle})\,|_{\mathbf{I}_F(\mathit{Hersteller})} = 1$
- $|\,(\text{Boeing,New York})\,|_{\mathbf{I}_F(\mathit{Hersteller})} = 1$
- $|\,(\text{Airbus,Hamburg})\,|_{\mathbf{I}_F(\mathit{Hersteller})} = 1$

$\mathbf{I}_F(\mathit{Bauteil})$

Teil	*Lager*
Motor	Seattle
Motor	Seattle
Flügel	Portland
Cockpit	Seattle
Cockpit	Seattle
Cockpit	Seattle

Notation:

- $|\,(\text{Motor,Seattle})\,|_{\mathbf{I}_F(\mathit{Bauteil})} = 2$
- $|\,(\text{Flügel,Portland})\,|_{\mathbf{I}_F(\mathit{Bauteil})} = 1$
- $|\,(\text{Cockpit,Seattle})\,|_{\mathbf{I}_F(\mathit{Bauteil})} = 3$

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.

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.

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:
  - $B =_b B' \quad :\Longleftrightarrow \quad |a|_B = |a|_{B'}$, für alle $a \in M$

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:
  - $B =_b B' \quad :\Longleftrightarrow \quad |a|_B = |a|_{B'}$, für alle $a \in M$
  - $B \subseteq_b B' \quad :\Longleftrightarrow \quad |a|_B \leqslant |a|_{B'}$, für alle $a \in M$

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:
  - $B =_b B' \quad :\Longleftrightarrow \quad |a|_B = |a|_{B'}$, für alle $a \in M$
  - $B \subseteq_b B' \quad :\Longleftrightarrow \quad |a|_B \leqslant |a|_{B'}$, für alle $a \in M$
  - Insbesondere gilt: $B =_b B' \iff \Big(B \subseteq_b B' \text{ und } B' \subseteq_b B\Big)$

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:
  - $B =_b B' \quad :\Longleftrightarrow \quad |a|_B = |a|_{B'}$, für alle $a \in M$
  - $B \subseteq_b B' \quad :\Longleftrightarrow \quad |a|_B \leqslant |a|_{B'}$, 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 $a \in M$

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:
  - $B =_b B' \quad :\Longleftrightarrow \quad |a|_B = |a|_{B'}$, für alle $a \in M$
  - $B \subseteq_b B' \quad :\Longleftrightarrow \quad |a|_B \leqslant |a|_{B'}$, 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.

Anfragen mit Mulitmengen-Semantik

Beispiel:

SQL-Anfrage:

```
SELECT B1.Teil, B2.Teil
FROM Bauteil B1, Bauteil B2
WHERE B1.Lager = B2.Lager
```

regelbasiert:

$$Ans(x, y) \leftarrow Bauteil(x, z),\ Bauteil(y, z)$$

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

$\mathbf{I}_F(Bauteil)$

Teil	*Lager*
Motor	Seattle
Motor	Seattle
Flügel	Portland
Cockpit	Seattle
Cockpit	Seattle
Cockpit	Seattle

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

$\mathbf{I}_F(Bauteil)$

Teil	*Lager*
Motor	Seattle
Motor	Seattle
Flügel	Portland
Cockpit	Seattle
Cockpit	Seattle
Cockpit	Seattle

- bilde Kreuzprodukt $\mathbf{I}_F(Bauteil) \times \mathbf{I}_F(Bauteil)$ (ohne Duplikatelimination)
- wähle die Tupel, in denen die Lager-Komponenten gleich sind
- streiche die Spalten mit den Lager-Komponenten

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

$\mathbf{I}_F(Bauteil)$

Teil	*Lager*
Motor	Seattle
Motor	Seattle
Flügel	Portland
Cockpit	Seattle
Cockpit	Seattle
Cockpit	Seattle

- bilde Kreuzprodukt $\mathbf{I}_F(Bauteil) \times \mathbf{I}_F(Bauteil)$ (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

- $|\,(\text{Motor,Motor})\,|_M = 4 \quad |\,(\text{Flügel,Flügel})\,|_M = 1 \quad |\,(\text{Cockpit,Cockpit})\,|_M = 9$
- $|\,(\text{Motor,Cockpit})\,|_M = 6 = |\,(\text{Cockpit,Motor})\,|_M$

Konjunktive Anfragen mit Multimengen-Semantik

Multimengen-Semantik $[\![Q]\!]_b$:

Sei $Q := \quad Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ 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 $\mathbf{I} \in inst_b(\mathbf{S})$ liefert als Ergebnis die Multimenge $[\![Q]\!]_b(\mathbf{I})$, so dass für alle Tupel $t \in \mathbf{dom}^r$ gilt:

$$|\, t \,|_{[\![Q]\!]_b(\mathbf{I})} \quad := \sum_{\substack{\beta \,:\, \beta \text{ Belegung} \\ \text{für } Q \text{ mit } \beta(u)=t}} \Big(\; |\beta(u_1)|_{\mathbf{I}(R_1)} \cdot |\beta(u_2)|_{\mathbf{I}(R_2)} \cdots |\beta(u_\ell)|_{\mathbf{I}(R_\ell)} \; \Big)$$

Konjunktive Anfragen mit Multimengen-Semantik

Multimengen-Semantik $[\![Q]\!]_b$:

Sei $Q := Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ 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 $\mathbf{I} \in inst_b(\mathbf{S})$ liefert als Ergebnis die Multimenge $[\![Q]\!]_b(\mathbf{I})$, so dass für alle Tupel $t \in \mathbf{dom}^r$ gilt:

$$|\, t \,|_{[\![Q]\!]_b(\mathbf{I})} := \sum_{\substack{\beta \,:\, \beta \text{ Belegung} \\ \text{für } Q \text{ mit } \beta(u)=t}} \Big(|\beta(u_1)|_{\mathbf{I}(R_1)} \cdot |\beta(u_2)|_{\mathbf{I}(R_2)} \cdots |\beta(u_\ell)|_{\mathbf{I}(R_\ell)} \Big)$$

Äquivalenz und Query Containment von Anfragen:

Seien Q und Q' zwei regelbasierte konjunktive Anfragen derselben Stelligkeit r über einem Datenbankschema $\mathbf{S}$.

- $Q \equiv_b Q' \quad :\Longleftrightarrow$

Konjunktive Anfragen mit Multimengen-Semantik

Multimengen-Semantik $[\![Q]\!]_b$:

Sei $Q := Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ 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 $\mathbf{I} \in inst_b(\mathbf{S})$ liefert als Ergebnis die Multimenge $[\![Q]\!]_b(\mathbf{I})$, so dass für alle Tupel $t \in \mathbf{dom}^r$ gilt:

$$|\, t \,|_{[\![Q]\!]_b(\mathbf{I})} := \sum_{\substack{\beta \,:\, \beta \text{ Belegung} \\ \text{für } Q \text{ mit } \beta(u)=t}} \Big(|\beta(u_1)|_{\mathbf{I}(R_1)} \cdot |\beta(u_2)|_{\mathbf{I}(R_2)} \cdots |\beta(u_\ell)|_{\mathbf{I}(R_\ell)} \Big)$$

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 $\mathbf{I} \in inst_b(\mathbf{S})$

Konjunktive Anfragen mit Multimengen-Semantik

Multimengen-Semantik $[\![Q]\!]_b$:

Sei $Q := Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ 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 $\mathbf{I} \in inst_b(\mathbf{S})$ liefert als Ergebnis die Multimenge $[\![Q]\!]_b(\mathbf{I})$, so dass für alle Tupel $t \in \mathbf{dom}^r$ gilt:

$$|\, t \,|_{[\![Q]\!]_b(\mathbf{I})} := \sum_{\substack{\beta \,:\, \beta \text{ Belegung} \\ \text{für } Q \text{ mit } \beta(u)=t}} \Big(|\beta(u_1)|_{\mathbf{I}(R_1)} \cdot |\beta(u_2)|_{\mathbf{I}(R_2)} \cdots |\beta(u_\ell)|_{\mathbf{I}(R_\ell)} \Big)$$

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 $\mathbf{I} \in inst_b(\mathbf{S})$
- $Q \sqsubseteq_b Q' \quad :\Longleftrightarrow$

Konjunktive Anfragen mit Multimengen-Semantik

Multimengen-Semantik $[\![Q]\!]_b$:

Sei $Q := Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ eine regelbasierte konjunktive Anfrage der Stelligkeit r über einem Datenbankschema $\mathbf{S}$.

- Eine Belegung β für Q ist, wie bisher, eine Abbildung $\beta : \mathrm{var}(Q) \to \mathbf{dom}$.
- Die Auswertung von Q in einer Multimengen-Datenbank $\mathbf{I} \in inst_b(\mathbf{S})$ liefert als Ergebnis die Multimenge $[\![Q]\!]_b(\mathbf{I})$, so dass für alle Tupel $t \in \mathbf{dom}^r$ gilt:

$$|\, t \,|_{[\![Q]\!]_b(\mathbf{I})} := \sum_{\substack{\beta \,:\, \beta \text{ Belegung} \\ \text{für } Q \text{ mit } \beta(u)=t}} \Big(\, |\beta(u_1)|_{\mathbf{I}(R_1)} \cdot |\beta(u_2)|_{\mathbf{I}(R_2)} \cdots |\beta(u_\ell)|_{\mathbf{I}(R_\ell)} \, \Big)$$

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 $\mathbf{I} \in inst_b(\mathbf{S})$
- $Q \sqsubseteq_b Q' \quad :\Longleftrightarrow \quad [\![Q]\!]_b(\mathbf{I}) \subseteq_b [\![Q']\!]_b(\mathbf{I}),$ für alle $\mathbf{I} \in inst_b(\mathbf{S})$

Konjunktive Anfragen mit Multimengen-Semantik

Multimengen-Semantik $[\![Q]\!]_b$:

Sei $Q := Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ 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 $\mathbf{I} \in inst_b(\mathbf{S})$ liefert als Ergebnis die Multimenge $[\![Q]\!]_b(\mathbf{I})$, so dass für alle Tupel $t \in \mathbf{dom}^r$ gilt:

$$|\, t \,|_{[\![Q]\!]_b(\mathbf{I})} := \sum_{\substack{\beta \,:\, \beta \text{ Belegung} \\ \text{für } Q \text{ mit } \beta(u)=t}} \Big(|\beta(u_1)|_{\mathbf{I}(R_1)} \cdot |\beta(u_2)|_{\mathbf{I}(R_2)} \cdots |\beta(u_\ell)|_{\mathbf{I}(R_\ell)} \Big)$$

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 $\mathbf{I} \in inst_b(\mathbf{S})$
- $Q \sqsubseteq_b Q' \quad :\Longleftrightarrow \quad [\![Q]\!]_b(\mathbf{I}) \subseteq_b [\![Q']\!]_b(\mathbf{I})$, für alle $\mathbf{I} \in inst_b(\mathbf{S})$
- Klar: $Q \equiv_b Q' \iff \Big(Q \sqsubseteq_b Q' \text{ und } Q' \sqsubseteq_b Q \Big)$

  Insbes: Äquivalenz ist höchstens so schwer wie Query Containment.

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).*

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.

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 einfacher als Äquivalenz bzgl. Mengen-Semantik.

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 einfacher als Äquivalenz bzgl. Mengen-Semantik.

- Das Query Containment Problem bzgl. Multimengen-Semantik ist vermutlich schwerer als das Query Containment Problem bzgl. Mengen-Semantik.

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 einfacher als Äquivalenz bzgl. Mengen-Semantik.

- Das Query Containment Problem bzgl. Multimengen-Semantik ist vermutlich schwerer 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.

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.

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 nicht entscheidbar.

Ab jetzt wieder

— und bis zum Ende des Semesters —

Mengen-Semantik

Kapitel 4:

Datalog

Abschnitt 4.1:

Syntax, Semantik und Auswertungskomplexität

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:

$$\Big\{ (x_S) : \exists x_L \Big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_S) \vee \exists x_Z \big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_Z) \wedge \textit{U-Bahn-Netz}(x_L, x_Z, x_S) \big) \Big) \Big\}$$

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:

$$\Big\{ (x_S) : \exists x_L \Big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_S) \vee \exists x_Z \big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_Z) \wedge \textit{U-Bahn-Netz}(x_L, x_Z, x_S) \big) \Big) \Big\}$$

(2) mit max. 2 Zwischenhalten: *analog*

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:

$$\Big\{ (x_S) : \exists x_L \Big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_S) \vee \exists x_Z \big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_Z) \wedge \textit{U-Bahn-Netz}(x_L, x_Z, x_S) \big) \Big) \Big\}$$

(2) mit max. 2 Zwischenhalten: *analog*

(3) mit beliebig vielen Zwischenhalten ???

(In einem späteren Kapitel werden wir sehen: Nicht ausdrückbar in Relationaler Algebra)

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:

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''
```

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:

$$\begin{aligned} \mathit{Erreichbar}(L,S,Z) &\leftarrow \mathit{U\text{-}Bahn\text{-}Netz}(L,S,Z) \\ \mathit{Erreichbar}(L,S,Z) &\leftarrow \mathit{Erreichbar}(L,S,Z'),\ \mathit{U\text{-}Bahn\text{-}Netz}(L,Z',Z) \\ \mathit{Ans}(Z) &\leftarrow \mathit{Erreichbar}(L,\text{“Bockenheimer Warte”},Z) \end{aligned}$$

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.

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.

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.

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

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.

Notation

- Wie üblich bezeichnen wir mit $\text{adom}(P)$ bzw. $\text{adom}(Q)$ die Menge der Konstanten, die in einem Datalog-Programm P bzw. einer Datalog-Anfrage Q vorkommen.

  Für eine Datenbank $\mathbf{I}$ ist $\text{adom}(P, \mathbf{I}) := \text{adom}(P) \cup \text{adom}(\mathbf{I})$ und $\text{adom}(Q, \mathbf{I}) := \text{adom}(Q) \cup \text{adom}(\mathbf{I})$.

- $R_0(u_0)$ 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").

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").

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").

- $sch(P) := edb(P) \dot\cup idb(P)$ heißt Schema von P.

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

- $\mathit{edb}(P) = \{\ \mathit{U\text{-}Bahn\text{-}Netz}\ \}$
- $\mathit{idb}(P) = \{\ \mathit{Erreichbar}, \mathit{Ans}\ \}$
- $\mathit{sch}(P) = \{\ \mathit{U\text{-}Bahn\text{-}Netz}, \mathit{Erreichbar}, \mathit{Ans}\ \}$

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.

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"

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:

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 $\text{ar}(Ans') = \text{ar}(R)$).

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 $\text{ar}(Ans') = \text{ar}(R)$).

(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

$$T_P(\mathbf{J})(R) := \begin{cases} \mathbf{J}(R) & \text{falls } R \in edb(P) \\ [\![Q'_{R,1}]\!](\mathbf{J}) \cup \cdots \cup [\![Q'_{R,k_r}]\!](\mathbf{J}) & \text{falls } R \in idb(P) \end{cases}$$

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 $\text{ar}(Ans') = \text{ar}(R)$).

(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".

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

- $R \in edb(P)$ und $t \in \mathbf{J}(R)$ oder
- $R \in idb(P)$ 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 $i \in \{1, \ldots, \ell\}$.

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

- $R \in edb(P)$ und $t \in \mathbf{J}(R)$ oder
- $R \in idb(P)$ 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 $\mathbf{J}, \mathbf{J}' \in inst(sch(P))$ mit $\mathbf{J} \subseteq \mathbf{J}'$ (d.h. $\mathbf{J}(R) \subseteq \mathbf{J}'(R)$ f.a. $R \in sch(P)$) gilt: $T_P(\mathbf{J}) \subseteq T_P(\mathbf{J}')$.

Beweis: leicht (*Übung*).

Der "Stufen"-Operator S_P

Definition 4.4

Sei P ein Datalog-Programm.

Der Stufen-Operator $S_P : inst(sch(P)) \to inst(sch(P))$ ist folgendermaßen definiert:
Für jedes $\mathbf{J} \in inst(sch(P))$ und jedes $R \in sch(P)$ ist

$$S_P(\mathbf{J})(R) \quad := \quad \mathbf{J}(R) \ \cup \ T_P(\mathbf{J})(R).$$

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 $S_P^{i+1}(\mathbf{J}) := S_P\big(S_P^i(\mathbf{J})\big)$.

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. $i \in \mathbb{N}$.

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. $j \geqslant i_0$.

Notation

- Ein $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$ heißt Fixpunkt von S_P, falls $S_P(\mathbf{J}) = \mathbf{J}$.

Notation

- Ein $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$ heißt Fixpunkt von S_P, falls $S_P(\mathbf{J}) = \mathbf{J}$.

- $\mathit{Ab\text{-}Stufe}(P, \mathbf{J}) := \min\{i_0 \ : \ S_P^{i_0}(\mathbf{J}) = S_P^{i_0+1}(\mathbf{J})\}$ 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.

  $\mathit{Ab\text{-}Stufe}(P, \mathbf{J})$ gibt also diejenige Stufe an, ab der der Fixpunkt erreicht ist:

$$\mathbf{J} = S_P^0(\mathbf{J}) \subsetneq S_P^1(\mathbf{J}) \subsetneq \cdots \subsetneq S_P^{\mathit{Ab\text{-}Stufe}(P,\mathbf{J})}(\mathbf{J}) = S_P^{\mathit{Ab\text{-}Stufe}(P,\mathbf{J})+1}(\mathbf{J}) = S_P^j(\mathbf{J})$$

$$\text{f.a. } j \geqslant \mathit{Ab\text{-}Stufe}(P, \mathbf{J})$$

- $S_P^\omega(\mathbf{J}) := S_P^{\mathit{Ab\text{-}Stufe}(P,\mathbf{J})}(\mathbf{J})$. Klar: $S_P^\omega(\mathbf{J})$ ist ein Fixpunkt von S_P.

Fixpunkt-Semantik von Datalog

Definition 4.6

(a) Sei P ein Datalog-Programm, $\mathbf{S} := edb(P)$.
Jeder Datenbank $\mathbf{I} \in inst(\mathbf{S})$ ordnen wir die Datenbank $\hat{\mathbf{I}} \in inst(sch(P))$ mit

$$\hat{\mathbf{I}}(R) := \begin{cases} \mathbf{I}(R) & \text{falls } R \in edb(P) \\ \emptyset & \text{falls } R \in idb(P) \end{cases}$$

zu. Ausgewertet in $\mathbf{I}$ liefert P die Datenbank

$$[\![P]\!](\mathbf{I}) := S_P^\omega(\hat{\mathbf{I}}) \in inst(sch(P))$$

Fixpunkt-Semantik von Datalog

Definition 4.6

(a) Sei P ein Datalog-Programm, $\mathbf{S} := edb(P)$.
Jeder Datenbank $\mathbf{I} \in inst(\mathbf{S})$ ordnen wir die Datenbank $\hat{\mathbf{I}} \in inst(sch(P))$ mit

$$\hat{\mathbf{I}}(R) := \begin{cases} \mathbf{I}(R) & \text{falls } R \in edb(P) \\ \emptyset & \text{falls } R \in idb(P) \end{cases}$$

zu. Ausgewertet in $\mathbf{I}$ liefert P die Datenbank

$$[\![P]\!](\mathbf{I}) := S_P^\omega(\hat{\mathbf{I}}) \in inst(sch(P))$$

(b) Eine Datalog-Anfrage $Q = (P, R)$ liefert auf einer Datenbank $\mathbf{I} \in inst(edb(P))$ die Relation

$$[\![Q]\!](\mathbf{I}) := ([\![P]\!](\mathbf{I}))(R).$$

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 $\big(\ \mathbf{J} := \mathbf{J}',\ \text{GOTO (2)}\ \big)$
(4) Gib $\mathbf{J}$ aus

Datenkomplexität dieses Algorithmus: Polynomialzeit.

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. $R \in \mathbf{S}$.

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 $\bigcap M := \bigcap_{\mathbf{J} \in M} \mathbf{J}$.

Fixpunkt-Semantik vs. Modellbasierte Semantik

Notation:

Für zwei Datenbanken $\mathbf{J}, \mathbf{J}' \in inst(\mathbf{S})$ 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 $M \subseteq inst(\mathbf{S})$ ist $\bigcap M := \bigcap_{\mathbf{J} \in M} \mathbf{J}$.

Satz 4.7 (Knaster und Tarski)

Sei P ein Datalog-Programm und sei $\mathbf{J} \in inst(sch(P))$. Dann gilt:

$$S_P^\omega(\mathbf{J}) \;=\; \bigcap \Big\{\, \mathbf{J}' \in inst(sch(P)) \;:\; S_P(\mathbf{J}') = \mathbf{J}' \text{ und } \mathbf{J} \subseteq \mathbf{J}' \,\Big\}.$$

D.h.: $S_P^\omega(\mathbf{J})$ ist die kleinste Erweiterung von $\mathbf{J}$, die ein Fixpunkt von S_P ist.

Beweis: Siehe Tafel.

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 $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ wird mit der folgenden Menge von Fakten identifiziert:

$$\mathit{Fakten}(\mathbf{I}) \;:=\; \{\; R(t) \quad : \quad R \in \mathbf{S} \text{ und } t \in \mathbf{I}(R) \;\}$$

- Für die beweisbasierte Semantik werden Datalog-Regeln als Schlussregel-Muster in Beweisen betrachtet.

Beweisbäume

Definition 4.8

Sei P ein Datalog-Programm und $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$. 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 $\mathit{Fakten}(\mathbf{J})$

(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 $i \in \{1, \ldots, \ell\}$).

Beweisbäume

Definition 4.8

Sei P ein Datalog-Programm und $\mathbf{J} \in \mathit{inst}(\mathit{sch}(P))$. 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 $\mathit{Fakten}(\mathbf{J})$

(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 $i \in \{1, \ldots, \ell\}$).

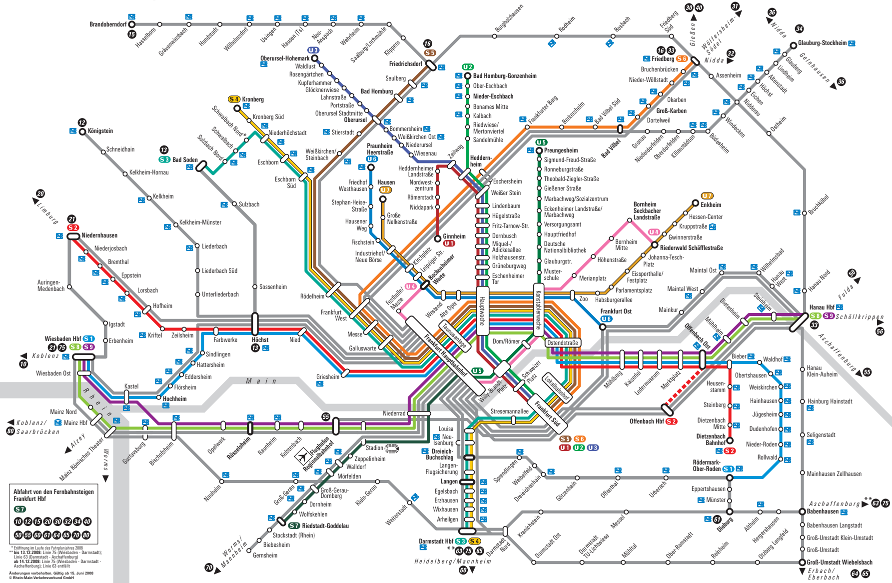
Beispiel: Beweisbäume für $\mathit{Ans}(\text{“Frankfurt Süd”})$ bzgl. dem Frankfurter U-Bahn-Netz und dem folgenden Datalog-Programm:

$$\begin{array}{rcl} E(L,S,Z) & \leftarrow & \mathit{U\text{-}Bahn\text{-}Netz}(L,S,Z) \\ E(L,S,Z) & \leftarrow & E(L,S,Y),\ \mathit{U\text{-}Bahn\text{-}Netz}(L,Y,Z) \\ \mathit{Ans}(Z) & \leftarrow & E(L, \text{“Hauptwache”}, Z) \end{array}$$

(siehe Tafel)

Rhein-Main-Verkehrsverbund

Schnellbahnplan



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.

Abschnitt 4.2:

Grenzen der Ausdrucksstärke von Datalog

Monotonie und Abschluss unter adom(Q)-Homomorphismen

Satz 4.10

Für jede Datalog-Anfrage Q gilt: $[\![Q]\!]$ ist monoton.

Monotonie und Abschluss unter 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).

Monotonie und Abschluss unter 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 adom(Q)*-Homomorphismen.*

Monotonie und Abschluss unter 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 adom(Q)*-Homomorphismen.*

Zur Erinnerung: Auf Übungsblatt 1 wurde Folgendes definiert.

- Ein C-Homomorphismus (für $C \subseteq \mathbf{dom}$) ist

Monotonie und Abschluss unter 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 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 $h_{|C} = \mathrm{id}$.

Monotonie und Abschluss unter 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 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 $h_{|C} = \mathrm{id}$.
- Eine Anfragefunktion q ist abgeschlossen unter C-Homomorphismen, falls für alle C-Homomorphismen h und alle Datenbanken $\mathbf{I}$ und $\mathbf{J}$ gilt:

Monotonie und Abschluss unter 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 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 $h_{|C} = \mathrm{id}$.
- Eine Anfragefunktion q ist abgeschlossen unter C-Homomorphismen, falls für alle C-Homomorphismen h und alle Datenbanken $\mathbf{I}$ und $\mathbf{J}$ gilt:

$$\text{Falls } \; h(\mathbf{I}) \subseteq \mathbf{J}, \; \text{ so ist } \; h(q(\mathbf{I})) \subseteq q(\mathbf{J}).$$

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?"

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.

Abschnitt 4.3:

Datalog zur Simulation von Turingmaschinen

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.

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 $w_0, \ldots, w_{n-1} \in \Sigma$ ist für jedes $a \in \Sigma$

$$\mathbf{I}_w(P_a) \;:=\; \{p \in \{0, \ldots, n{-}1\} \;:\; w_p = a\}$$

und

$$\begin{aligned} \mathbf{I}_w(\text{Min}) &:= \{0\} \\ \mathbf{I}_w(\text{Max}) &:= \{n{-}1\} \\ \mathbf{I}_w(\text{Succ}) &:= \{(p, p{+}1) \;:\; 0 \leqslant p < n{-}1\}. \end{aligned}$$

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:

$$[\![Q_{M,k}]\!](\mathbf{I}_w) = \text{“yes”} \quad \Longleftrightarrow \quad \text{bei Eingabe } w \text{ hält } M \text{ nach höchstens } |w|^k - 1 \text{ Schritten in einem akzeptierenden Zustand an.}$$

Beweis: Siehe Tafel.

Die Auswertungskomplexität von Datalog-Anfragen

Theorem 4.14 (Immerman und Vardi)

(a) *Die kombinierte Komplexität des Auswertungsproblems für Datalog-Anfragen ist EXPTIME-vollständig.*

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.*

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).

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).

Abschnitt 4.4:
Statische Analyse

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

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.

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 $\mathbf{I} \in inst(edb(P_1))$?

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 $\mathbf{J} \in inst(sch(P_1))$?

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.

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: $\textit{Ab-Stufe}(P,\mathbf{I})+1$

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 $\textit{Ab-Stufe}(P,\mathbf{I})$ werden?

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 $\textit{Ab-Stufe}(P,\mathbf{I})$ werden?
- Klar: – in jeder Iteration kommt mindestens ein Faktum dazu
  – $\text{adom}([\![P]\!](\mathbf{I})) \subseteq \text{adom}(P,\mathbf{I})$
- Daher: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant$

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 $\textit{Ab-Stufe}(P,\mathbf{I})$ werden?
- Klar: – in jeder Iteration kommt mindestens ein Faktum dazu
  – $\text{adom}([\![P]\!](\mathbf{I})) \subseteq \text{adom}(P,\mathbf{I})$
- Daher: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant \sum_{R \in idb(P)} |\,\text{adom}(P,\mathbf{I})|^{\text{ar}(R)}$

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 $\textit{Ab-Stufe}(P,\mathbf{I})$ werden?
- Klar: – in jeder Iteration kommt mindestens ein Faktum dazu
  – $\text{adom}([\![P]\!](\mathbf{I})) \subseteq \text{adom}(P,\mathbf{I})$
- Daher: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant \sum_{R \in idb(P)} |\,\text{adom}(P,\mathbf{I})|^{\text{ar}(R)}$
- Besonders "schön" sind solche Datalog-Programme P, bei denen $\textit{Ab-Stufe}(P,\mathbf{I})$ gar nicht von $\mathbf{I}$ abhängt. Solche Programme heißen beschränkt (engl.: bounded).

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 $\textit{Ab-Stufe}(P,\mathbf{I})$ werden?
- Klar: – in jeder Iteration kommt mindestens ein Faktum dazu
  – $\text{adom}([\![P]\!](\mathbf{I})) \subseteq \text{adom}(P,\mathbf{I})$
- Daher: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant \sum_{R \in \textit{idb}(P)} |\,\text{adom}(P,\mathbf{I})|^{\text{ar}(R)}$
- Besonders "schön" sind solche Datalog-Programme P, bei denen $\textit{Ab-Stufe}(P,\mathbf{I})$ 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 $\mathbf{I} \in \textit{inst}(\textit{edb}(P))$ gilt: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant d$.

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 $\textit{Ab-Stufe}(P,\mathbf{I})$ werden?
- Klar: – in jeder Iteration kommt mindestens ein Faktum dazu
  – $\text{adom}([\![P]\!](\mathbf{I})) \subseteq \text{adom}(P,\mathbf{I})$
- Daher: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant \sum_{R \in \textit{idb}(P)} |\,\text{adom}(P,\mathbf{I})|^{\text{ar}(R)}$
- Besonders "schön" sind solche Datalog-Programme P, bei denen $\textit{Ab-Stufe}(P,\mathbf{I})$ 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 $\mathbf{I} \in \textit{inst}(\textit{edb}(P))$ gilt: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant d$.

  Den kleinsten solchen Wert d nennen wir maximale Rekursionstiefe von P.

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 $\textit{Ab-Stufe}(P,\mathbf{I})$ werden?
- Klar: – in jeder Iteration kommt mindestens ein Faktum dazu
  – $\text{adom}([\![P]\!](\mathbf{I})) \subseteq \text{adom}(P,\mathbf{I})$
- Daher: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant \sum_{R \in \textit{idb}(P)} |\,\text{adom}(P,\mathbf{I})|^{\text{ar}(R)}$
- Besonders "schön" sind solche Datalog-Programme P, bei denen $\textit{Ab-Stufe}(P,\mathbf{I})$ 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 $\mathbf{I} \in \textit{inst}(\textit{edb}(P))$ gilt: $\textit{Ab-Stufe}(P,\mathbf{I}) \leqslant d$.

  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).

Beispiel

Das Datalog-Programm

$$\begin{aligned} E(L,S,Z) &\leftarrow \mathit{U\text{-}Bahn\text{-}Netz}(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ \mathit{U\text{-}Bahn\text{-}Netz}(L,Y,Z) \\ \mathit{Ans}(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \end{aligned}$$

ist

Beispiel

Das Datalog-Programm

$$\begin{aligned} E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Ans(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \end{aligned}$$

ist nicht beschränkt.

Beispiel

Das Datalog-Programm

$$\begin{aligned} E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Ans(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \end{aligned}$$

ist nicht beschränkt.

Das Datalog-Programm

$$\begin{aligned} Kauft(x,y) &\leftarrow Mag(x,y) \\ Kauft(x,y) &\leftarrow Person(x),\ Trendsetter(z),\ Kauft(z,y) \end{aligned}$$

ist

Beispiel

Das Datalog-Programm

$$\begin{aligned} E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Ans(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \end{aligned}$$

ist nicht beschränkt.

Das Datalog-Programm

$$\begin{aligned} Kauft(x,y) &\leftarrow Mag(x,y) \\ Kauft(x,y) &\leftarrow Person(x),\ Trendsetter(z),\ Kauft(z,y) \end{aligned}$$

ist beschränkt mit maximaler Rekursionstiefe 3

Beispiel

Das Datalog-Programm

$$\begin{aligned} E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Ans(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \end{aligned}$$

ist nicht beschränkt.

Das Datalog-Programm

$$\begin{aligned} Kauft(x,y) &\leftarrow Mag(x,y) \\ Kauft(x,y) &\leftarrow Person(x),\ Trendsetter(z),\ Kauft(z,y) \end{aligned}$$

ist beschränkt mit maximaler Rekursionstiefe 3 und äquivalent zum nicht-rekursiven Programm

Beispiel

Das Datalog-Programm

$$\begin{aligned} E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Ans(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \end{aligned}$$

ist nicht beschränkt.

Das Datalog-Programm

$$\begin{aligned} Kauft(x,y) &\leftarrow Mag(x,y) \\ Kauft(x,y) &\leftarrow Person(x),\ Trendsetter(z),\ Kauft(z,y) \end{aligned}$$

ist beschränkt mit maximaler Rekursionstiefe 3 und äquivalent zum nicht-rekursiven Programm

$$\begin{aligned} Kauft'(x,y) &\leftarrow Mag(x,y) \\ Kauft'(x,y) &\leftarrow Person(x),\ Trendsetter(z),\ Mag(z,y) \\ Kauft(x,y) &\leftarrow Kauft'(x,y) \\ Kauft(x,y) &\leftarrow Person(x),\ Trendsetter(z),\ Kauft'(z,y) \end{aligned}$$

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.

Abschnitt 4.5:

Einschränkung und Erweiterungen: nr-Datalog und Datalog mit Negation

Nicht-Rekursives Datalog — Beispiel

Beispiel 4.18

(a) Das Datalog-Programm

$$\begin{aligned} \mathit{Kauft}(x,y) &\leftarrow \mathit{Mag}(x,y) \\ \mathit{Kauft}(x,y) &\leftarrow \mathit{Person}(x),\ \mathit{Trendsetter}(z),\ \mathit{Mag}(z,y) \end{aligned}$$

ist nicht-rekursiv.

Nicht-Rekursives Datalog — Beispiel

Beispiel 4.18

(a) Das Datalog-Programm

$$\begin{aligned} \mathit{Kauft}(x,y) &\leftarrow \mathit{Mag}(x,y) \\ \mathit{Kauft}(x,y) &\leftarrow \mathit{Person}(x),\ \mathit{Trendsetter}(z),\ \mathit{Mag}(z,y) \end{aligned}$$

ist nicht-rekursiv.

(b) Äquivalent dazu, aber NICHT nicht-rekursiv ist

$$\begin{aligned} \mathit{Kauft}(x,y) &\leftarrow \mathit{Mag}(x,y) \\ \mathit{Kauft}(x,y) &\leftarrow \mathit{Person}(x),\ \mathit{Trendsetter}(z),\ \mathit{Kauft}(z,y) \end{aligned}$$

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

$$E_P := \left\{ (R, S) : \begin{array}{l} \text{es gibt eine Regel in } P, \text{ in deren Rumpf } R \text{ und in} \\ \text{deren Kopf } S \text{ vorkommt} \end{array} \right\}$$

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

$$E_P := \left\{ (R, S) : \begin{array}{l} \text{es gibt eine Regel in } P, \text{ in deren Rumpf } R \text{ und in} \\ \text{deren Kopf } S \text{ vorkommt} \end{array} \right\}$$

(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.

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

$$E_P := \left\{ (R, S) : \begin{array}{l} \text{es gibt eine Regel in } P, \text{ in deren Rumpf } R \text{ und in} \\ \text{deren Kopf } S \text{ vorkommt} \end{array} \right\}$$

(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*

"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.

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.

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: $[\![(Q_1 \cup Q_2)]\!](\mathbf{I}) := [\![Q_1]\!](\mathbf{I}) \cup [\![Q_2]\!](\mathbf{I})$.

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.

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:
  $[\![Q]\!](\mathbf{I}) := \{\beta((e_1, \ldots, e_r)) : \beta : \text{var} \rightarrow \text{adom}(Q, \mathbf{I}), \text{ so dass } \mathbf{I} \models \varphi[\beta]\}$

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:
  $[\![Q]\!](\mathbf{I}) := \{\beta((e_1, \ldots, e_r)) : \beta : \text{var} \rightarrow \text{adom}(Q, \mathbf{I}), \text{ so dass } \mathbf{I} \models \varphi[\beta]\}$

Satz 4.21

Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen beschreiben:

(a) *nr-Datalog-Anfragen*

(b) *SPCU-Algebra*

(c) *positiver existentieller Kalkül PE-CALC$_{adom}$*

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:
  $[\![Q]\!](\mathbf{I}) := \{\beta((e_1, \ldots, e_r)) : \beta : \text{var} \rightarrow \text{adom}(Q, \mathbf{I}), \text{ so dass } \mathbf{I} \models \varphi[\beta]\}$

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.

Beweis von Satz 4.21

(b) $\Rightarrow$ (a): Induktion über den Aufbau von SPCU-Anfragen (leicht).
Details: siehe Tafel.

Beweis von Satz 4.21

(b) $\Rightarrow$ (a): Induktion über den Aufbau von SPCU-Anfragen (leicht).
Details: siehe Tafel.

(a) $\Rightarrow$ (c): siehe Tafel.

Beweis von Satz 4.21

(b) $\Rightarrow$ (a): Induktion über den Aufbau von SPCU-Anfragen (leicht).
Details: siehe Tafel.

(a) $\Rightarrow$ (c): siehe Tafel.

(c) $\Rightarrow$ (b): Bei gegebener Anfrage $Q = \{(e_1, \ldots, e_r) \ : \ \varphi\}$ 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 $\mathrm{CQ}^=$-Formel ist.

Beweis von Satz 4.21

(b) $\Rightarrow$ (a): Induktion über den Aufbau von SPCU-Anfragen (leicht).
Details: siehe Tafel.

(a) $\Rightarrow$ (c): siehe Tafel.

(c) $\Rightarrow$ (b): Bei gegebener Anfrage $Q = \{(e_1, \ldots, e_r) \ : \ \varphi\}$ 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 $\mathrm{CQ}^=$-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

$$[\![Q_i]\!](\mathbf{I}) \quad = \quad \Big\{ \ \beta((e_1, \ldots, e_r)) \ : \ \ \beta : \mathrm{var} \to \mathrm{adom}(Q, \mathbf{I}), \ \text{so dass } \mathbf{I} \models \varphi_i \ \Big\}.$$

Beweis von Satz 4.21

(b) $\Rightarrow$ (a): Induktion über den Aufbau von SPCU-Anfragen (leicht).
Details: siehe Tafel.

(a) $\Rightarrow$ (c): siehe Tafel.

(c) $\Rightarrow$ (b): Bei gegebener Anfrage $Q = \{(e_1, \ldots, e_r) \ : \ \varphi\}$ 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 $\mathrm{CQ}^=$-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

$$[\![Q_i]\!](\mathbf{I}) \quad = \quad \Big\{ \ \beta((e_1, \ldots, e_r)) \ : \ \ \beta : \mathrm{var} \to \mathrm{adom}(Q, \mathbf{I}), \ \text{so dass } \mathbf{I} \models \varphi_i \ \Big\}.$$

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. □

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.

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 $\text{ar}(R_0)$ 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.

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 $\text{ar}(R_0)$ 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.

Datalog mit Negation

Ziel: Auch Negationszeichen "$\neg$" 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.

Frage: Was soll die Semantik von Datalog$^\neg$ sein?

Beispiel 4.23

Anfrage:

"Gib alle Stationen aus, die von "Bockenheimer Warte" aus nicht ohne Umsteigen zu erreichen sind."

Frage: Was soll die Semantik von Datalog$^\neg$ sein?

Beispiel 4.23

Anfrage:

"Gib alle Stationen aus, die von "Bockenheimer Warte" aus nicht ohne Umsteigen zu erreichen sind."

Als Datalog$^\neg$-Anfrage:

$$
\begin{aligned}
E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z)\\
Erreichbar\_BW(Z) &\leftarrow E(L,\text{"Bockenheimer Warte"},Z)\\
Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Ans(Z) &\leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)
\end{aligned}
$$

Hier: Semantik intuitiv klar.

Probleme mit der Semantik von Datalog¬

Zur Erinnerung: In der Übung wurde gezeigt, dass für jedes Datalog-Programm P und jede Datenbank $I \in inst(edb(P))$ gilt: für jedes $i \in \mathbb{N}$ ist $S_p^i(\hat{\mathbf{I}}) = T_p^i(\hat{\mathbf{I}})$.
Also ist $[\![P]\!](\mathbf{I}) = S_P^\omega(\hat{\mathbf{I}}) = T_p^\omega(\hat{\mathbf{I}})$.

Probleme mit der Semantik von Datalog$^\neg$

Zur Erinnerung: In der Übung wurde gezeigt, dass für jedes Datalog-Programm P und jede Datenbank $I \in inst(edb(P))$ 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 $\mathbf{I} \in inst(\{A\})$ gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \ =$$

Probleme mit der Semantik von Datalog$^\neg$

Zur Erinnerung: In der Übung wurde gezeigt, dass für jedes Datalog-Programm P und jede Datenbank $I \in inst(edb(P))$ 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 $\mathbf{I} \in inst(\{A\})$ gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \ = \ \mathbf{I}(A) \quad \text{für alle } i \geqslant 1$$

Probleme mit der Semantik von Datalog$^\neg$

Zur Erinnerung: In der Übung wurde gezeigt, dass für jedes Datalog-Programm P und jede Datenbank $I \in inst(edb(P))$ 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 $\mathbf{I} \in inst(\{A\})$ gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \ = \ \mathbf{I}(A) \quad \text{für alle } i \geqslant 1$$

aber

$$T_P^i(\hat{\mathbf{I}})(R) \ =$$

Probleme mit der Semantik von Datalog$^\neg$

Zur Erinnerung: In der Übung wurde gezeigt, dass für jedes Datalog-Programm P und jede Datenbank $I \in inst(edb(P))$ 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 $\mathbf{I} \in inst(\{A\})$ gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \ = \ \mathbf{I}(A) \quad \text{für alle } i \geqslant 1$$

aber

$$T_P^i(\hat{\mathbf{I}})(R) \ = \ \begin{cases} \emptyset & \text{falls } i \text{ gerade} \\ \text{adom}(\mathbf{I}) & \text{falls } i \text{ ungerade} \end{cases}$$

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.

(b) $R(x) \leftarrow A(x),\ \neg S(x)$
$S(x) \leftarrow A(x),\ \neg R(x)$

(b) $R(x) \leftarrow A(x),\ \neg S(x)$
$S(x) \leftarrow A(x),\ \neg R(x)$

Hier gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \;=\; S_P^i(\hat{\mathbf{I}})(S) \;=$$

(b) $R(x) \leftarrow A(x),\ \neg S(x)$
$S(x) \leftarrow A(x),\ \neg R(x)$

Hier gilt:

$$S_P^i(\hat{\mathbf{I}})(R) \;=\; S_P^i(\hat{\mathbf{I}})(S) \;=\; \mathbf{I}(A) \quad \text{für alle } i \geqslant 1$$

(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

(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 $\mathrm{FP}_2(S) = \emptyset$

Datalog$^\neg$ und der Stufenoperator S_P

Um eine Semantik für Datalog$^\neg$ festzulegen, könnte man einfach wieder den Stufenoperator S_P betrachten.

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 $\mathbf{J} \in inst(sch(P))$, dass

$$\mathbf{J} \subseteq S_P(\mathbf{J}) \subseteq S_P\big(S_P(\mathbf{J})\big) \subseteq \cdots \subseteq S_P^i(\mathbf{J}) \subseteq S_P^{i+1}(\mathbf{J}) \subseteq \cdots$$

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 $\mathbf{J} \in inst(sch(P))$, dass

$$\mathbf{J} \subseteq S_P(\mathbf{J}) \subseteq S_P(S_P(\mathbf{J})) \subseteq \cdots \subseteq S_P^i(\mathbf{J}) \subseteq S_P^{i+1}(\mathbf{J}) \subseteq \cdots$$

Da $adom(P, \mathbf{J})$ endlich ist, wird irgendwann ein (eindeutig definierter) Fixpunkt von $S_P(\cdot)$ erreicht

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 $\mathbf{J} \in inst(sch(P))$, dass

$$\mathbf{J} \subseteq S_P(\mathbf{J}) \subseteq S_P(S_P(\mathbf{J})) \subseteq \cdots \subseteq S_P^i(\mathbf{J}) \subseteq S_P^{i+1}(\mathbf{J}) \subseteq \cdots$$

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: $S_P^\omega(\mathbf{J})$).

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 $\mathrm{adom}(P, \mathbf{J})$ 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.

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

$$
\begin{aligned}
E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z)\\
Erreichbar\_BW(Z) &\leftarrow E(L,\text{“Bockenheimer Warte”},Z)\\
Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Ans(Z) &\leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)
\end{aligned}
$$

“Intuitive Semantik”:

Alle Stationen, die von “Bockenheimer Warte” aus nicht ohne Umsteigen zu erreichen sind.

Inflationäre Fixpunkt-Semantik:

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

$$
\begin{aligned}
E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z)\\
Erreichbar\_BW(Z) &\leftarrow E(L,\text{“Bockenheimer Warte”},Z)\\
Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Ans(Z) &\leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)
\end{aligned}
$$

“Intuitive Semantik”:

Alle Stationen, die von “Bockenheimer Warte” aus nicht ohne Umsteigen zu erreichen sind.

Inflationäre Fixpunkt-Semantik: Alle Stationen.

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$^\neg$: ???

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$^\neg$

Semipositives Datalog$^\neg$

Negation ist nur bei edb-Prädikaten erlaubt.

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

$$\hat{\mathbf{I}}, \quad T_P(\hat{\mathbf{I}}), \quad T_P^2(\hat{\mathbf{I}}), \quad T_P^3(\hat{\mathbf{I}}), \quad \ldots$$

  erreicht. Notation für diesen Fixpunkt: $T_P^\omega(\hat{\mathbf{I}})$.

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

$$\hat{\mathbf{I}}, \quad T_P(\hat{\mathbf{I}}), \quad T_P^2(\hat{\mathbf{I}}), \quad T_P^3(\hat{\mathbf{I}}), \quad \ldots$$

  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

$$[\![P]\!](\mathbf{I}) \;:=\; T_P^\omega(\hat{\mathbf{I}})$$

Stratifiziertes Datalog$^\neg$ — Beispiel

$$
\begin{aligned}
E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z)\\
Erreichbar\_BW(Z) &\leftarrow E(L,\text{“Bockenheimer Warte”},Z)\\
Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z)\\
Ans(Z) &\leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)
\end{aligned}
$$

- Die Negation ist hier nicht mit der Rekursion verschränkt.
- Sie kann angewendet werden, nachdem $Erreichbar\_BW(\cdot)$ vollständig berechnet ist.
- $\rightsquigarrow$ Grundidee für stratifiziertes Datalog$^\neg$.

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 $\sigma : idb(P) \to \{1, \ldots, m\}$ 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 $S \in idb(P)$ 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 $\sigma(S) < \sigma(R)$.

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 $\sigma : idb(P) \to \{1, \ldots, m\}$ 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 $S \in idb(P)$ 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

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 $\sigma : idb(P) \to \{1, \ldots, m\}$ 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 $S \in idb(P)$ 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.

Stratifiziertes Datalog$^\neg$ — Beispiele

$P^1 :=$

$$\begin{aligned} E(L,S,Z) &\leftarrow BVG(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Erreichbar\_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\ Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \end{aligned}$$

$P^2 :=$

$$Ans(Z) \leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)$$

ist eine Stratifizierung des Datalog$^\neg$-Programms aus Beispiel 4.25.

Stratifiziertes Datalog$^\neg$ — Beispiele

$P^1 :=$

$$\begin{aligned} E(L,S,Z) &\leftarrow BVG(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Erreichbar\_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\ Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \end{aligned}$$

$P^2 :=$

$$Ans(Z) \leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)$$

ist eine Stratifizierung des Datalog$^\neg$-Programms aus Beispiel 4.25.

Das Datalog$^\neg$-Programm $R(x) \leftarrow A(x),\ \neg R(x)$

Stratifiziertes Datalog$^\neg$ — Beispiele

$P^1 :=$

$$\begin{aligned} E(L,S,Z) &\leftarrow BVG(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Erreichbar\_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\ Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \end{aligned}$$

$P^2 :=$

$$Ans(Z) \leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)$$

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.

Stratifiziertes Datalog$^\neg$ — Beispiele

$P^1 :=$

$$\begin{aligned} E(L,S,Z) &\leftarrow BVG(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Erreichbar\_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\ Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \end{aligned}$$

$P^2 :=$

$$Ans(Z) \leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)$$

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

$$\begin{aligned} R(x) &\leftarrow A(x),\ \neg S(x) \\ S(x) &\leftarrow A(x),\ \neg R(x) \end{aligned}$$

Stratifiziertes Datalog$^\neg$ — Beispiele

$P^1 :=$

$$\begin{aligned} E(L,S,Z) &\leftarrow BVG(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Erreichbar\_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\ Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \end{aligned}$$

$P^2 :=$

$$Ans(Z) \leftarrow Station(Z),\ \neg Erreichbar\_BW(Z)$$

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

$$\begin{aligned} R(x) &\leftarrow A(x),\ \neg S(x) \\ S(x) &\leftarrow A(x),\ \neg R(x) \end{aligned}$$

auch nicht.

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

$$E_P^+ := \left\{ (R,S) : \begin{array}{l} \text{es gibt eine Regel in } P, \text{ in deren Kopf } S \text{ vorkommt} \\ \text{und in deren Rumpf } R \text{ positiv vorkommt} \end{array} \right\}$$

$$E_P^- := \left\{ (R,S) : \begin{array}{l} \text{es gibt eine Regel in } P, \text{ in deren Kopf } S \text{ vorkommt} \\ \text{und in deren Rumpf } R \text{ negativ vorkommt} \end{array} \right\}$$

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

$$E_P^+ := \left\{ (R,S) : \begin{array}{l} \text{es gibt eine Regel in } P, \text{ in deren Kopf } S \text{ vorkommt} \\ \text{und in deren Rumpf } R \text{ positiv vorkommt} \end{array} \right\}$$

$$E_P^- := \left\{ (R,S) : \begin{array}{l} \text{es gibt eine Regel in } P, \text{ in deren Kopf } S \text{ vorkommt} \\ \text{und in deren Rumpf } R \text{ negativ vorkommt} \end{array} \right\}$$

Beispiel: *Siehe Tafel:* Abhängigkeitsgraph für

$$\begin{aligned} E(L,S,Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ E(L,S,Z) &\leftarrow E(L,S,Y),\ U\text{-}Bahn\text{-}Netz(L,Y,Z) \\ Erreichbar\_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\ Station(S) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ Station(Z) &\leftarrow U\text{-}Bahn\text{-}Netz(L,S,Z) \\ Ans(Z) &\leftarrow Station(Z),\ \neg Erreichbar\_BW(Z) \end{aligned}$$

Test auf Stratifizierbarkeit

Proposition 4.28

Für jedes Datalog$^\neg$-Programm P gilt:

$$P \text{ ist stratifizierbar} \iff \begin{array}{l}\text{Im Abhängigkeitsgraph } G_P \text{ gibt es keinen}\\ \text{Kreis, in dem eine Kante aus } E_P^- \text{ vorkommt.}\end{array}$$

Test auf Stratifizierbarkeit

Proposition 4.28

Für jedes Datalog$^\neg$-Programm P gilt:

$$P \text{ ist stratifizierbar} \iff \begin{array}{l}\text{Im Abhängigkeitsgraph } G_P \text{ gibt es keinen} \\ \text{Kreis, in dem eine Kante aus } E_P^- \text{ vorkommt.}\end{array}$$

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!

Test auf Stratifizierbarkeit

Proposition 4.28

Für jedes Datalog$^\neg$-Programm P gilt:

$$P \text{ ist stratifizierbar} \iff \begin{array}{l}\text{Im Abhängigkeitsgraph } G_P \text{ gibt es keinen} \\ \text{Kreis, in dem eine Kante aus } E_P^- \text{ vorkommt.}\end{array}$$

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*.

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 $edb(P^i) \subseteq edb(P) \cup idb(P^1) \cup \cdots \cup idb(P^{i-1})$

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

$$\begin{array}{lcl} \mathbf{I}^1 & := & [\![P^1]\!](\mathbf{I}) \\ \mathbf{I}^2 & := & [\![P^2]\!](\mathbf{I}^1) \\ \vdots & \vdots & \vdots \\ \mathbf{I}^m & := & [\![P^m]\!](\mathbf{I}^{m-1}) \end{array}$$

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

$$\begin{array}{lcl} \mathbf{I}^1 & := & [\![P^1]\!](\mathbf{I}) \\ \mathbf{I}^2 & := & [\![P^2]\!](\mathbf{I}^1) \\ \vdots & \vdots & \vdots \\ \mathbf{I}^m & := & [\![P^m]\!](\mathbf{I}^{m-1}) \end{array}$$

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: $[\![P^m]\!](\mathbf{I}^{m-1}) = [\![Q^n]\!](\mathbf{I}^{n-1})$.

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

$$\begin{array}{lcl} \mathbf{I}^1 & := & [\![P^1]\!](\mathbf{I}) \\ \mathbf{I}^2 & := & [\![P^2]\!](\mathbf{I}^1) \\ \vdots & \vdots & \vdots \\ \mathbf{I}^m & := & [\![P^m]\!](\mathbf{I}^{m-1}) \end{array}$$

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 $\mathbf{J}|_{edb(P)} = \mathbf{I}$.

Spezialfall: nr-Datalog$^\neg$

nr-Datalog$^\neg$ = nr-Datalog mit Negation

= stratifiziertes Datalog$^\neg$, eingeschränkt auf Programme P, in deren Abhängigkeitsgraph es keinen gerichteten Kreis (über $(E_P^+ \cup E_P^-)$) gibt.

Spezialfall: nr-Datalog$^\neg$

$$\begin{aligned} \text{nr-Datalog}^\neg &= \text{nr-Datalog mit Negation} \\ &= \text{stratifiziertes Datalog}^\neg\text{, eingeschränkt auf Programme } P\text{, in deren Abhängigkeitsgraph es keinen gerichteten Kreis (über } (E_P^+ \cup E_P^-)) \text{ gibt.} \end{aligned}$$

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 und zum Relationenkalkül gegeben.

Kapitel 5:

Funktionale Abhängigkeiten

Abschnitt 5.1:

Notationen

Zur Erinnerung — Benannte Perspektive

Zu Beginn des Semesters hatten 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 $sorte : \mathbf{rel} \to \mathcal{P}_e(\mathbf{att})$, 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 $sorte(R) = U$.
- Ein Relationsschema ist einfach ein Relationsname R.
- Manchmal schreiben wir kurz $R[U]$ für $sorte(R) = U$.
- Ein R-Tupel ist eine Funktion $t : sorte(R) \to \mathbf{dom}$.
- Eine R-Relation ist eine endliche Menge von R-Tupeln.
- $inst(R)$ bezeichnet die Menge aller R-Relationen.
- $inst(U)$ bezeichnet die Menge aller Relationen über einem Relationsschema der Sorte U (für $U \subseteq_e \mathbf{att}$)

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.

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:

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

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

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.

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 $\rightarrow$ Menge

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 $\rightarrow$ Menge

Zur Anfrage-Optimierung:

Die „optimierte Anfrage“ muss nur auf solchen Datenbanken äquivalent zur Original-Anfrage sein, die die obigen Abhängigkeiten erfüllen.

$\rightsquigarrow$ die minimale, solchermaßen äquivalente Anfrage ist evtl. noch kleiner als die, die durch die Tableau-Minimierung aus Kapitel 3.4 gefunden wird.

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	

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“)

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 $I \in inst(U)$ 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.

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 $I \in inst(U)$ 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

$$I \models \mathcal{F} \quad :\Longleftrightarrow \quad \text{für alle } f \in \mathcal{F} \text{ gilt } \; I \models f.$$

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 $I \in inst(U)$ 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

$$I \models \mathcal{F} \quad :\Longleftrightarrow \quad \text{für alle } f \in \mathcal{F} \text{ gilt } \; I \models f.$$

(d) Eine Schlüsselbedingung ist eine FD der Form $X \to U$.

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“)

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.

Abschnitt 5.2:

The Chase — Die Verfolgungsjagd

Beispiel zu „The Chase — Die Verfolgungsjagd“

Beispiel 5.3

Tableau-Anfrage $Q = (T, t)$ mit

$$T = \begin{array}{|cccc|}\hline A & B & C & D \\ \hline w & x & y & z' \\ w' & x & y' & z \\ \hline \end{array} \qquad \text{und} \qquad t = (w, x, y, z)$$

Klar: Die Anfrage $Q := (T, t)$ ist minimal im Sinne von Kapitel 3.4.

Beispiel zu „The Chase — Die Verfolgungsjagd“

Beispiel 5.3

Tableau-Anfrage $Q = (T, t)$ mit

$$T = \begin{array}{|cccc|}\hline A & B & C & D \\ \hline w & x & y & z' \\ w' & x & y' & z \\ \hline \end{array} \qquad \text{und} \qquad t = (w, x, y, z)$$

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.

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 $\mathbf{I}(R) = I$.

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 $\mathbf{S}$“.

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 $\mathbf{I}(R) = I$.

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 $[\![Q]\!](\mathbf{I})$.

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.

- $Q_1 \sqsubseteq_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) \subseteq [\![Q_2]\!](I)$

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.

- $Q_1 \sqsubseteq_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) \subseteq [\![Q_2]\!](I)$
- $Q_1 \equiv_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) = [\![Q_2]\!](I)$.

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.

- $Q_1 \sqsubseteq_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) \subseteq [\![Q_2]\!](I)$
- $Q_1 \equiv_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) = [\![Q_2]\!](I)$.

Sei $\mathcal{F}$ eine Menge von FDs über R.

- $\mathrm{Mod}(\mathcal{F}) \;:=\; \mathrm{Mod}_R(\mathcal{F}) \;:=\; \{\, I \in \mathit{inst}(R) \;:\; I \models \mathcal{F} \,\}$

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.

- $Q_1 \sqsubseteq_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) \subseteq [\![Q_2]\!](I)$
- $Q_1 \equiv_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) = [\![Q_2]\!](I)$.

Sei $\mathcal{F}$ eine Menge von FDs über R.

- $\mathrm{Mod}(\mathcal{F}) := \mathrm{Mod}_R(\mathcal{F}) := \{ I \in \mathit{inst}(R) \;:\; I \models \mathcal{F} \}$
- $Q_1 \sqsubseteq_{\mathcal{F}} Q_2 \quad :\Longleftrightarrow \quad Q_1 \sqsubseteq_{\mathrm{Mod}(\mathcal{F})} Q_2$

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.

- $Q_1 \sqsubseteq_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) \subseteq [\![Q_2]\!](I)$
- $Q_1 \equiv_{\mathcal{I}} Q_2 \quad :\Longleftrightarrow \quad$ für alle $I \in \mathcal{I}$ ist $\quad [\![Q_1]\!](I) = [\![Q_2]\!](I)$.

Sei $\mathcal{F}$ eine Menge von FDs über R.

- $\mathrm{Mod}(\mathcal{F}) := \mathrm{Mod}_R(\mathcal{F}) := \{ I \in \mathit{inst}(R) \;:\; I \models \mathcal{F} \}$
- $Q_1 \sqsubseteq_{\mathcal{F}} Q_2 \quad :\Longleftrightarrow \quad Q_1 \sqsubseteq_{\mathrm{Mod}(\mathcal{F})} Q_2$
- $Q_1 \equiv_{\mathcal{F}} Q_2 \quad :\Longleftrightarrow \quad Q_1 \equiv_{\mathrm{Mod}(\mathcal{F})} Q_2$

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

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.

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 $z \neq y$.

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.

Verfolgungssequenzen

Definition 5.7

(a) Eine Verfolgungssequenz für (T, t) mittels $\mathcal{F}$ ist eine Folge

$$(T_0, t_0),\ (T_1, t_1),\ (T_2, t_2),\ \ldots$$

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).

Verfolgungssequenzen

Definition 5.7

(a) Eine Verfolgungssequenz für (T, t) mittels $\mathcal{F}$ ist eine Folge

$$(T_0, t_0),\ (T_1, t_1),\ (T_2, t_2),\ \ldots$$

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.

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.)

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

$$T \models \mathcal{F} \quad :\Longleftrightarrow \quad T \models f, \quad \text{für alle } f \in \mathcal{F}.$$

Eigenschaften des Resultats einer Verfolgungssequenz

Lemma 5.9

Sei (T', t') *das Resultat einer terminierten Verfolgungssequenz für* (T, t) *mittels* $\mathcal{F}$.
Dann gilt: $(T', t') \equiv_{\mathcal{F}} (T, t)$ *und* $T' \models \mathcal{F}$.

Beweis: Übung.

Eigenschaften des Resultats einer Verfolgungssequenz

Lemma 5.9

Sei (T', t') das Resultat einer terminierten Verfolgungssequenz für (T, t) mittels $\mathcal{F}$.
Dann gilt: $(T', t') \equiv_{\mathcal{F}} (T, t)$ *und* $T' \models \mathcal{F}$.

Beweis: Übung.

Beobachtung: Jede Verfolgungssequenz ist endlich und kann zu einer terminierten Sequenz vervollständigt werden.

Eigenschaften des Resultats einer Verfolgungssequenz

Lemma 5.9

Sei (T', t') das Resultat einer terminierten Verfolgungssequenz für (T, t) mittels $\mathcal{F}$.
Dann gilt: $(T', t') \equiv_{\mathcal{F}} (T, t)$ *und* $T' \models \mathcal{F}$.

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.

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.

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].

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 $chase(T, t, \mathcal{F})$ das Resultat einer (bzw. sämtlicher) terminierter Verfolgungssequenzen für (T, t) mittels $\mathcal{F}$.

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 $chase(T, t, \mathcal{F})$ das Resultat einer (bzw. sämtlicher) terminierter Verfolgungssequenzen für (T, t) mittels $\mathcal{F}$.

Bemerkung: Von Lemma 5.9 wissen wir, dass $chase(T, t, \mathcal{F}) \equiv_{\mathcal{F}} (T, t)$ und $chase(T, t, \mathcal{F}) \models \mathcal{F}$.

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 $chase(T, t, \mathcal{F})$ berechnet.

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 $chase(T, t, \mathcal{F})$ 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.

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 $chase(T, t, \mathcal{F})$ 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

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 $chase(T, t, \mathcal{F})$ 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.

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 $chase(T, t, \mathcal{F})$ 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

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 $chase(T, t, \mathcal{F})$ 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. □

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) $Q_1 \sqsubseteq_{\mathcal{F}} Q_2 \iff chase(T_1, t_1, \mathcal{F}) \sqsubseteq chase(T_2, t_2, \mathcal{F})$

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.

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.

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 $Q' := (T', t') := \mathit{chase}(T, t, \mathcal{F})$. 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: $Q'' \equiv_{\mathcal{F}} Q' \equiv_{\mathcal{F}} Q$

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 $Q' := (T', t') := \mathit{chase}(T, t, \mathcal{F})$. 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 $\mathit{min}(Q) := \mathit{min}(T, t)$, um die gemäß Theorem 3.39(a) minimale zu Q äquivalente Tableau-Anfrage zu bezeichnen.

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 $Q' := (T', t') := \mathit{chase}(T, t, \mathcal{F})$. 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 $\mathit{min}(Q) := \mathit{min}(T, t)$, 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: $|\mathit{min}(\mathit{chase}(T, t, \mathcal{F}))| \leqslant |\mathit{min}(T, t)|$.

Hier ohne Beweis.

Eine Beweisskizze findet sich auf Seite 178 des Buchs [AHV].

Abschnitt 5.3:

Der Armstrong-Kalkül

Implikation von Abhängigkeiten

Gegeben: Eine Menge $\mathcal{F}$ von FDs über U.

Frage: Welche anderen FDs folgen aus $\mathcal{F}$?

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 $I \models AD \to E$.

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 $\mathcal{F} \models \{f\}$.

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 $\mathcal{G} \models \mathcal{F}$.

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 $\mathcal{F}^{*,U} := \{ X \to Y \ : \ X, Y \subseteq U \text{ und } \mathcal{F} \models X \to Y \}$.

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 $\mathcal{F}^{*,U} := \{ X \to Y \ : \ X, Y \subseteq U \text{ und } \mathcal{F} \models X \to Y \}$.

Klar: Für alle $X \subseteq U$ und alle $Y \subseteq X$ gilt $X \to Y \in \mathcal{F}^{*,U}$.
Insbesondere: $2^{|U|} \leqslant |\mathcal{F}^{*,U}| \leqslant 2^{2 \cdot |U|}$.

Fragen: Wie kann man (1) testen, ob $\mathcal{F} \models X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models X \to Y$ spricht über *alle* Relationen $I \in \mathit{inst}(U)$. Das sind unendlich viele (da **dom** unendlich ist).

Fragen: Wie kann man (1) testen, ob $\mathcal{F} \models X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models X \to Y$ spricht über *alle* Relationen $I \in \mathit{inst}(U)$. Das sind unendlich viele (da **dom** unendlich ist).

Lösung für (1): Reduziere das Problem „$\mathcal{F} \models X \to Y$?“ 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.

Fragen: Wie kann man (1) testen, ob $\mathcal{F} \models X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models X \to Y$ spricht über *alle* Relationen $I \in \mathit{inst}(U)$. Das sind unendlich viele (da **dom** unendlich ist).

Lösung für (1): Reduziere das Problem „$\mathcal{F} \models X \to Y$?“ 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: $\mathcal{F} \models X \to Y \iff Q_X \sqsubseteq_{\mathcal{F}} Q_Y$.

Fragen: Wie kann man (1) testen, ob $\mathcal{F} \models X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models X \to Y$ spricht über *alle* Relationen $I \in inst(U)$. Das sind unendlich viele (da **dom** unendlich ist).

Lösung für (1): Reduziere das Problem „$\mathcal{F} \models X \to Y$?“ 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: $\mathcal{F} \models X \to Y \iff Q_X \sqsubseteq_{\mathcal{F}} Q_Y$.

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: Siehe Tafel.

Fragen: Wie kann man (1) testen, ob $\mathcal{F} \models X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models X \to Y$ spricht über *alle* Relationen $I \in \mathit{inst}(U)$. Das sind unendlich viele (da **dom** unendlich ist).

Fragen: Wie kann man (1) testen, ob $\mathcal{F} \models X \to Y$?
(2) die Hülle $\mathcal{F}^{*,U}$ berechnen?

Problem: Die Definition von $\mathcal{F} \models X \to Y$ 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

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.

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.

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.

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

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.

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.

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: $\mathcal{F} \vdash_U f$.

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:

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).

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

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 $\mathcal{F} \models_U f$.

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

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 $\mathcal{F} \vdash_U f$.

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 5.19 (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. "$\Longleftarrow$": Siehe Tafel.

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 5.19 (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. "$\Longleftarrow$": Siehe Tafel.

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

Abschnitt 6.1:
Definition und Beispiele

Grenzen der Ausdrucksstärke konjunktiver Anfragen

Wir haben gesehen:

- konjunktive Anfragen können nur *monotone* Anfragefunktionen beschreiben (Satz 3.6) $\leadsto$ 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).

Grenzen der Ausdrucksstärke konjunktiver Anfragen

Wir haben gesehen:

- konjunktive Anfragen können nur *monotone* Anfragefunktionen beschreiben (Satz 3.6) $\leadsto$ 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.

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

$$I \cup J \;:=\; \{\, t \;:\; t \in I \text{ oder } t \in J \,\}$$

bzw.

$$I - J \;:=\; \{\, t \in I \;:\; t \notin J \,\}$$

SPJRU, SPCU und relationale Algebra

Definition 6.1

Sei **S** ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der SPJRU[**S**] werden die Definitionen von SPJR[**S**] um die folgende Regel erweitert:

- Sind Q und P zwei SPJRU[**S**]-Anfragen derselben Sorte Σ so ist $(Q \cup P)$ eine SPJRU[**S**]-Anfrage der Sorte Σ

SPJRU, SPCU und relationale Algebra

Definition 6.1

Sei **S** ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der SPJRU[**S**] (bzw. der SPCU[**S**]) werden die Definitionen von SPJR[**S**] (bzw. SPC[**S**]) um die folgende Regel erweitert:

- Sind Q und P zwei SPJRU[**S**]-Anfragen derselben Sorte Σ (bzw. SPCU[**S**]-Anfragen derselben Stelligkeit k), so ist $(Q \cup P)$ eine SPJRU[**S**]-Anfrage der Sorte Σ (bzw. eine SPCU[**S**]-Anfrage der Stelligkeit k).

SPJRU, SPCU und relationale Algebra

Definition 6.1

Sei **S** ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der SPJRU[**S**] (bzw. der SPCU[**S**]) werden die Definitionen von SPJR[**S**] (bzw. SPC[**S**]) um die folgende Regel erweitert:

- Sind Q und P zwei SPJRU[**S**]-Anfragen derselben Sorte Σ (bzw. SPCU[**S**]-Anfragen derselben Stelligkeit k), so ist $(Q \cup P)$ eine SPJRU[**S**]-Anfrage der Sorte Σ (bzw. eine SPCU[**S**]-Anfrage 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 SPJRU[**S**]-Anfragen (bzw. SPCU[**S**]-Anfragen) für alle Datenbankschemata **S**.

SPJRU, SPCU und relationale Algebra

Definition 6.1

Sei **S** ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der SPJRU[**S**] (bzw. der SPCU[**S**]) werden die Definitionen von SPJR[**S**] (bzw. SPC[**S**]) um die folgende Regel erweitert:

- Sind Q und P zwei SPJRU[**S**]-Anfragen derselben Sorte Σ (bzw. SPCU[**S**]-Anfragen derselben Stelligkeit k), so ist $(Q \cup P)$ eine SPJRU[**S**]-Anfrage der Sorte Σ (bzw. eine SPCU[**S**]-Anfrage der Stelligkeit k).

(b) Zur Definition der Klasse der Anfragen der relationalen Algebra über **S** in der benannten Perspektive werden die Definitionen von SPJRU[**S**] um die folgende Regel erweitert:

- Sind Q und P zwei Anfragen der relationalen Algebra derselben Sorte Σ so ist $(Q - P)$ eine Anfrage relationalen Algebra der Sorte Σ

Die Semantik $[\![Q]\!]$ solcher Anfragen Q ist induktiv auf die offensichtliche Art definiert. Mit SPJRU (bzw. SPCU) bezeichnen wir die Klasse aller SPJRU[**S**]-Anfragen (bzw. SPCU[**S**]-Anfragen) für alle Datenbankschemata **S**.

SPJRU, SPCU und relationale Algebra

Definition 6.1

Sei $\mathbf{S}$ ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der SPJRU[$\mathbf{S}$] (bzw. der SPCU[$\mathbf{S}$]) werden die Definitionen von SPJR[$\mathbf{S}$] (bzw. SPC[$\mathbf{S}$]) um die folgende Regel erweitert:

- Sind Q und P zwei SPJRU[$\mathbf{S}$]-Anfragen derselben Sorte Σ (bzw. SPCU[$\mathbf{S}$]-Anfragen derselben Stelligkeit k), so ist $(Q \cup P)$ eine SPJRU[$\mathbf{S}$]-Anfrage der Sorte Σ (bzw. eine 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 SPJRU[$\mathbf{S}$] (bzw. 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 SPJRU[$\mathbf{S}$]-Anfragen (bzw. SPCU[$\mathbf{S}$]-Anfragen) für alle Datenbankschemata $\mathbf{S}$.

Beispiele

- In welchen Kinos wird “Alien” oder “Brazil” gespielt?

Beispiele

- In welchen Kinos wird “Alien” oder “Brazil” gespielt?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel}=\text{"Alien"}}\big(\mathit{Programm}\big) \;\cup\; \sigma_{\text{Titel}=\text{"Brazil"}}\big(\mathit{Programm}\big) \Big)$$

Beispiele

- In welchen Kinos wird “Alien” oder “Brazil” gespielt?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel}=\text{"Alien"}}\big(\mathit{Programm}\big) \;\cup\; \sigma_{\text{Titel}=\text{"Brazil"}}\big(\mathit{Programm}\big) \Big)$$

- Welche Regisseure haben noch nie mit “Matt Damon” gearbeitet?

Beispiele

- In welchen Kinos wird “Alien” oder “Brazil” gespielt?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel}=\text{"Alien"}}\big(\mathit{Programm}\big) \;\cup\; \sigma_{\text{Titel}=\text{"Brazil"}}\big(\mathit{Programm}\big) \Big)$$

- Welche Regisseure haben noch nie mit “Matt Damon” gearbeitet?

$$\pi_{\text{Regie}}\big(\mathit{Filme}\big) \;-\; \pi_{\text{Regie}}\Big(\sigma_{\text{Schauspieler}=\text{“Matt Damon”}}\big(\mathit{Filme}\big)\Big)$$

Beispiele

- In welchen Kinos wird “Alien” oder “Brazil” gespielt?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel}=\text{"Alien"}}\big(\mathit{Programm}\big) \;\cup\; \sigma_{\text{Titel}=\text{"Brazil"}}\big(\mathit{Programm}\big) \Big)$$

- Welche Regisseure haben noch nie mit “Matt Damon” gearbeitet?

$$\pi_{\text{Regie}}\big(\mathit{Filme}\big) \;-\; \pi_{\text{Regie}}\Big(\sigma_{\text{Schauspieler}=\text{“Matt Damon”}}\big(\mathit{Filme}\big)\Big)$$

- Welche derzeit laufenden Filme haben nur Schauspieler, die schon mal in einem Film von “James Cameron” mitgespielt haben?

Beispiele

- In welchen Kinos wird "Alien" oder "Brazil" gespielt?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel}=\text{"Alien"}}\big(\mathit{Programm}\big) \cup \sigma_{\text{Titel}=\text{"Brazil"}}\big(\mathit{Programm}\big) \Big)$$

- Welche Regisseure haben noch nie mit "Matt Damon" gearbeitet?

$$\pi_{\text{Regie}}\big(\mathit{Filme}\big) - \pi_{\text{Regie}}\Big(\sigma_{\text{Schauspieler}=\text{"Matt Damon"}}\big(\mathit{Filme}\big)\Big)$$

- Welche derzeit laufenden Filme haben nur Schauspieler, die schon mal in einem Film von "James Cameron" mitgespielt haben?

$$\pi_{\text{Titel}}\big(\mathit{Programm}\big) - \pi_{\text{Titel}}\Big(\mathit{Filme} \bowtie \Big(\pi_{\substack{\text{Schau-}\\\text{spieler}}}\big(\mathit{Filme}\big) - \pi_{\substack{\text{Schau-}\\\text{spieler}}}\big(\sigma_{\substack{\text{Regie}=\text{"James}\\\text{Cameron"}}}\big(\mathit{Filme}\big)\big)\Big)\Big)$$

Beispiele

- In welchen Kinos wird "Alien" oder "Brazil" gespielt?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel}=\text{"Alien"}}(\mathit{Programm}) \cup \sigma_{\text{Titel}=\text{"Brazil"}}(\mathit{Programm}) \Big)$$

- Welche Regisseure haben noch nie mit "Matt Damon" gearbeitet?

$$\pi_{\text{Regie}}(\mathit{Filme}) - \pi_{\text{Regie}}\Big(\sigma_{\text{Schauspieler}=\text{"Matt Damon"}}(\mathit{Filme})\Big)$$

- Welche derzeit laufenden Filme haben nur Schauspieler, die schon mal in einem Film von "James Cameron" mitgespielt haben?

$$\pi_{\text{Titel}}(\mathit{Programm}) - \pi_{\text{Titel}}\Big(\mathit{Filme} \bowtie \Big(\underbrace{\pi_{\substack{\text{Schau-}\\\text{spieler}}}(\mathit{Filme}) - \pi_{\substack{\text{Schau-}\\\text{spieler}}}\big(\sigma_{\substack{\text{Regie}=\text{"James}\\\text{Cameron"}}}(\mathit{Filme})\big)}_{\text{Schauspieler, die noch nie mit James Cameron gearbeitet haben}}\Big)\Big)$$

Beispiele

- In welchen Kinos wird "Alien" oder "Brazil" gespielt?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel}=\text{"Alien"}}(\mathit{Programm}) \cup \sigma_{\text{Titel}=\text{"Brazil"}}(\mathit{Programm}) \Big)$$

- Welche Regisseure haben noch nie mit "Matt Damon" gearbeitet?

$$\pi_{\text{Regie}}(\mathit{Filme}) - \pi_{\text{Regie}}\Big(\sigma_{\text{Schauspieler}=\text{"Matt Damon"}}(\mathit{Filme})\Big)$$

- Welche derzeit laufenden Filme haben nur Schauspieler, die schon mal in einem Film von "James Cameron" mitgespielt haben?

$$\pi_{\text{Titel}}(\mathit{Programm}) - \pi_{\text{Titel}}\Big(\mathit{Filme} \bowtie \underbrace{\Big(\pi_{\substack{\text{Schau-}\\\text{spieler}}}(\mathit{Filme}) - \pi_{\substack{\text{Schau-}\\\text{spieler}}}\big(\sigma_{\substack{\text{Regie}=\text{"James}\\\text{Cameron"}}}(\mathit{Filme})\big)\Big)}_{\text{Schauspieler, die noch nie mit James Cameron gearbeitet haben}}\Big)$$

Filme mit mind. einem Schauspieler, der noch nie mit James Cameron gearbeitet hat

Ausdrucksstärke

Proposition 6.2

(a) *Jede* SPCU*-Anfrage und jede* SPJRU*-Anfrage ist monoton.*

Ausdrucksstärke

Proposition 6.2

(a) *Jede* SPCU-*Anfrage und jede* SPJRU-*Anfrage ist monoton.*

Beweis:

(a)+(b): Einfache Induktion über den Aufbau der Anfragen.

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:*

$$\text{adom}\left([\![Q]\!](\mathbf{I})\right) \subseteq \text{adom}(Q, \mathbf{I}).$$

Beweis:

(a)+(b): Einfache Induktion über den Aufbau der Anfragen.

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:*

$$\text{adom}\left([\![Q]\!](\mathbf{I})\right) \subseteq \text{adom}(Q, \mathbf{I}).$$

(c)
$$\begin{array}{ccccc} \text{SPC} & < & \text{SPCU} & < & \textit{relationale Algebra (unbenannte Perspektive)} \\ \equiv & & \equiv & & \equiv \\ \text{SPJR} & < & \text{SPJRU} & < & \textit{relationale Algebra (benannte Perspektive)} \end{array}$$

Beweis:

(a)+(b): Einfache Induktion über den Aufbau der Anfragen.

(c): Übung. □

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.

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.*

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 $\pi_{j_1,\ldots,j_k}$ die Indizes j_i nicht paarweise verschieden sein müssen.

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 $\pi_{j_1,\ldots,j_k}$ die Indizes j_i nicht paarweise verschieden sein müssen.

(ii) *Keiner der Operatoren π, $\cup$, $-$, $\times$ ist redundant.*

Beweis: Übung.

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 $i_1, \ldots, i_m, j_1, \ldots, j_m \geqslant 1$.

Zwei Tupel $\overline{a} = (a_1, \ldots, a_r) \in \mathbf{dom}^r$ und $\overline{b} = (b_1, \ldots, b_s) \in \mathbf{dom}^s$ mit $r \geqslant \max\{i_1, \ldots, i_m\}$ und $s \geqslant \max\{j_1, \ldots, j_m\}$ erfüllen θ

$$(\text{kurz: } (\overline{a}, \overline{b}) \models \theta, \text{ bzw. } \theta(\overline{a}, \overline{b}))$$

falls für alle $\ell \in \{1, \ldots, m\}$ gilt: $a_{i_\ell} = b_{j_\ell}$.

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 $i_1, \ldots, i_m, j_1, \ldots, j_m \geqslant 1$.

Zwei Tupel $\overline{a} = (a_1, \ldots, a_r) \in \mathbf{dom}^r$ und $\overline{b} = (b_1, \ldots, b_s) \in \mathbf{dom}^s$ mit $r \geqslant \max\{i_1, \ldots, i_m\}$ und $s \geqslant \max\{j_1, \ldots, j_m\}$ erfüllen θ

$$(\text{kurz: } (\overline{a}, \overline{b}) \models \theta, \text{ bzw. } \theta(\overline{a}, \overline{b}))$$

falls für alle $\ell \in \{1, \ldots, m\}$ 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: $I \bowtie_\theta J := \{ (\overline{a}, \overline{b}) : \overline{a} \in I, \ \overline{b} \in J, \text{ so dass } (\overline{a}, \overline{b}) \models \theta \}$

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 $i_1, \ldots, i_m, j_1, \ldots, j_m \geqslant 1$.

Zwei Tupel $\overline{a} = (a_1, \ldots, a_r) \in \mathbf{dom}^r$ und $\overline{b} = (b_1, \ldots, b_s) \in \mathbf{dom}^s$ mit $r \geqslant \max\{i_1, \ldots, i_m\}$ und $s \geqslant \max\{j_1, \ldots, j_m\}$ erfüllen θ

$$(\text{kurz: } (\overline{a}, \overline{b}) \models \theta, \text{ bzw. } \theta(\overline{a}, \overline{b})),$$

falls für alle $\ell \in \{1, \ldots, m\}$ 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: $I \bowtie_\theta J := \{ (\overline{a}, \overline{b}) : \overline{a} \in I, \ \overline{b} \in J, \text{ so dass } (\overline{a}, \overline{b}) \models \theta \}$

- Semijoin $\ltimes_\theta$, wobei θ eine positive konjunktive Join-Bedingung ist.
  Semantik: $I \ltimes_\theta J := \{ \overline{a} \in I : \text{ex. } \overline{b} \in J, \text{ so dass } (\overline{a}, \overline{b}) \models \theta \}$

Abschnitt 6.2:

Anfrageauswertung und Heuristische Optimierung

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

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

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.

Anfrageauswertung allgemein

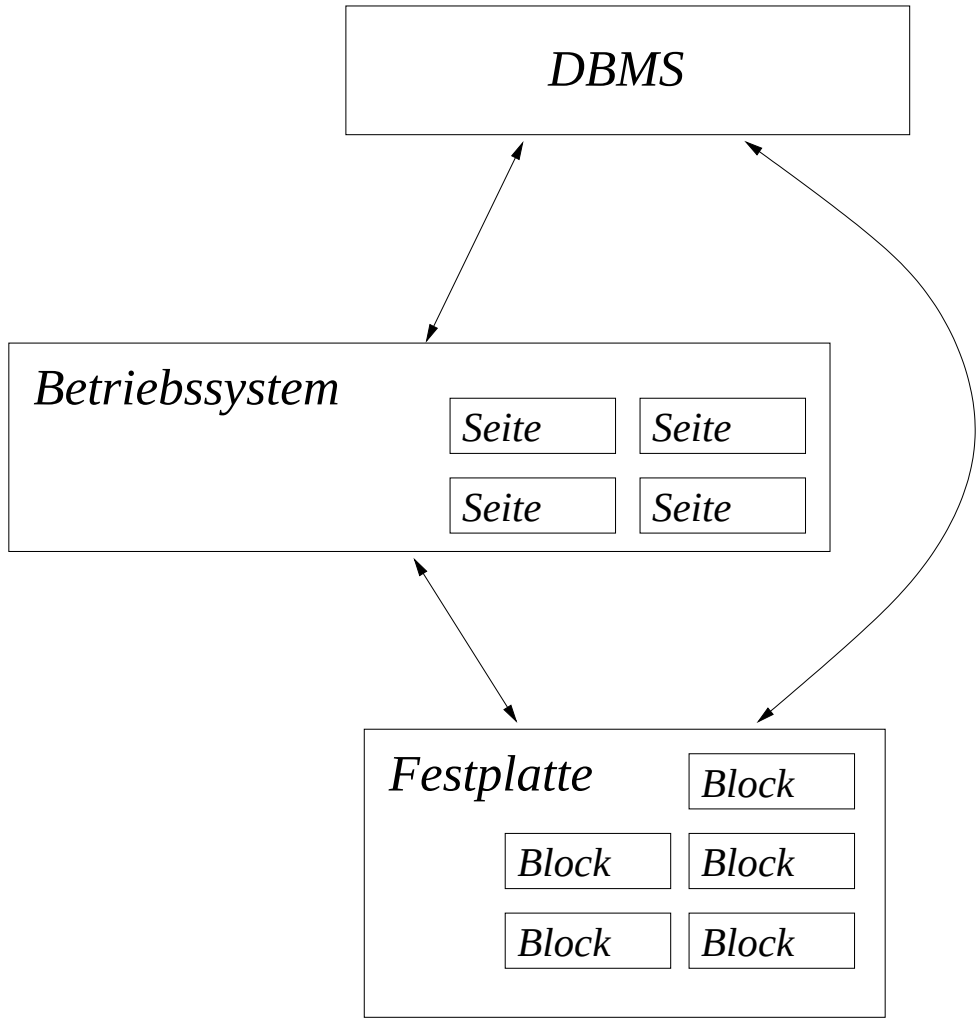
Vorbemerkung:

- 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, ...

Verwaltung des Sekundärspeichers



Hier (der Einfachheit halber): 1 Plattenzugriff $\widehat{=}$ Lesen eines Blocks bzw. einer Seite

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

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

Projektion $\pi_{j_1,\ldots,j_k}(I)$:

- 2 Komponenten:
  - Ändern der einzelnen Tupel (Auswahl und Reihenfolge von Spalten)
  - Duplikatelimination

Projektion $\pi_{j_1,\ldots,j_k}(I)$:

- 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"

Projektion $\pi_{j_1,\ldots,j_k}(I)$:

- 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

Projektion $\pi_{j_1,\ldots,j_k}(I)$:

- 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
    - $\leadsto$ Duplikate werden auf denselben Wert abgebildet und dadurch erkannt
    - bei idealer Hash-Funktion: lineare Zeit

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

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 $\rightsquigarrow$ 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

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 $\rightsquigarrow$ 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 $\mathcal{O}(n_I + n_J)$

Beispiel für Merge-Technik

Berechne $I \bowtie_\theta J$ für Join-Bedingung $\theta := x_1{=}y_4 \wedge x_2{=}y_1$

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.

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

Anfrageauswertung

Proposition 6.4

Das Auswertungsproblem für die relationale Algebra lässt sich in Zeit $(k+n)^{\mathcal{O}(k)}$ *lösen.*

Beweis:

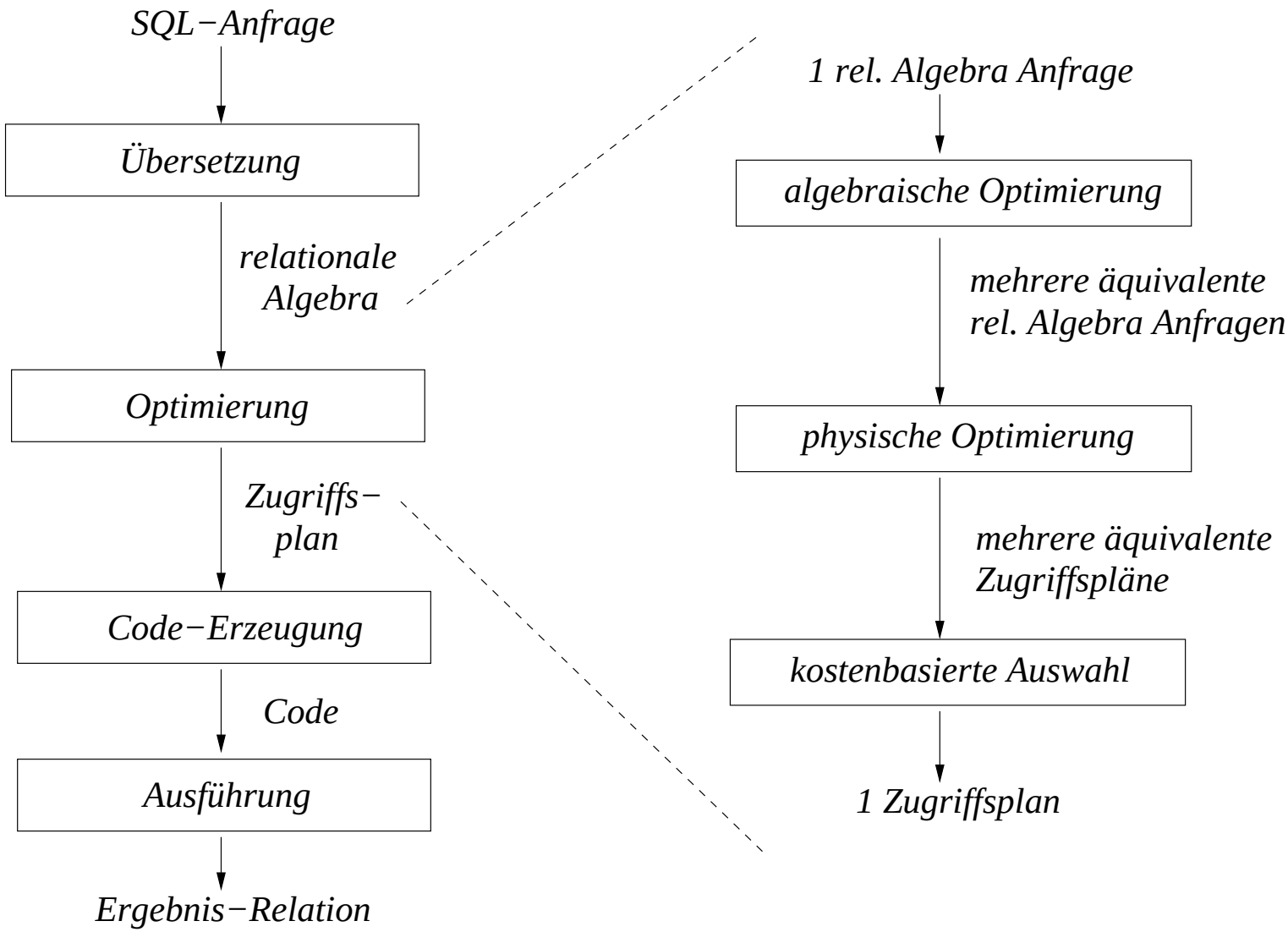
Zeige per Induktion nach dem Aufbau von Anfragen der relationalen Algebra, dass für jede Anfrage Q der Länge k und jede Datenbank $\mathbf{I}$ der Größe n gilt:

(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*.

Anfragebearbeitung durch ein DBMS



Ziel der Optimierung

- möglichst schnelle Auswertung der Anfrage
- möglichst wenige Zugriffe auf Festplatte

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

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''
```

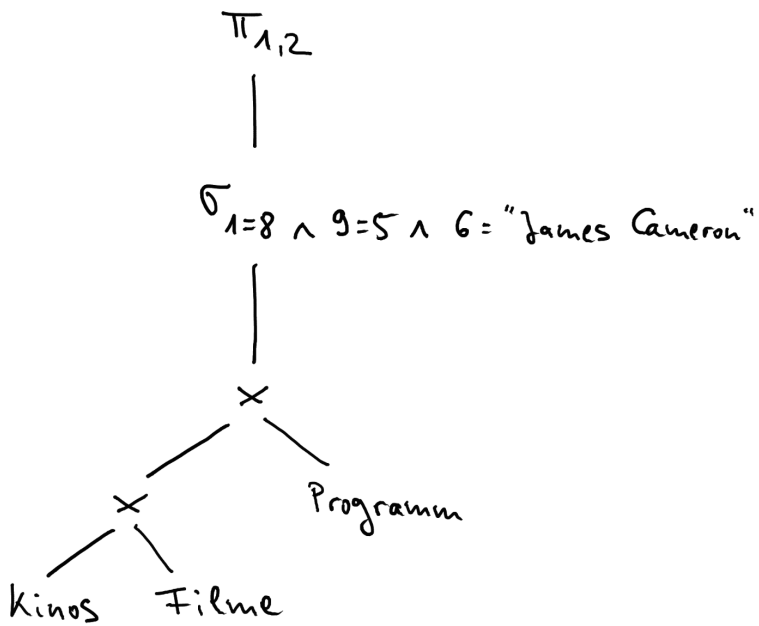
Direkte Übersetzung in relationale Algebra:

$$\pi_{1,2}\Big(\sigma_{\substack{1=8 \wedge 9=5 \wedge \\ 6=\text{"James Cameron"}}}\big(\mathit{Kinos} \times \mathit{Filme} \times \mathit{Programm}\big)\Big)$$

Original-Anfrage

$$\pi_{1,2}\Big(\sigma_{\substack{1=8\wedge 9=5\wedge \\ 6=\text{“James Cameron”}}}\big(\mathit{Kinos}\times\mathit{Filme}\times\mathit{Programm}\big)\Big)$$

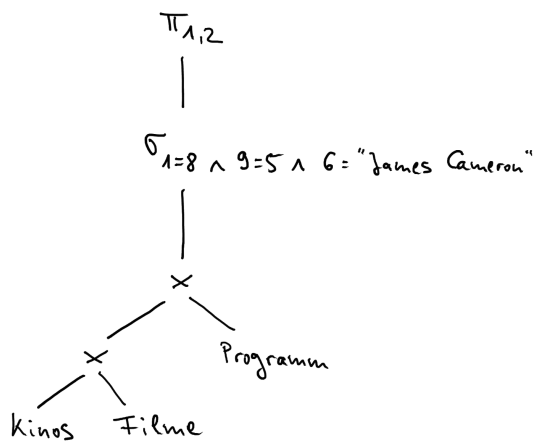
dargestellt als Anfrage-Baum:



Anfrage auswerten auf folgender Beispiel-Datenbank:

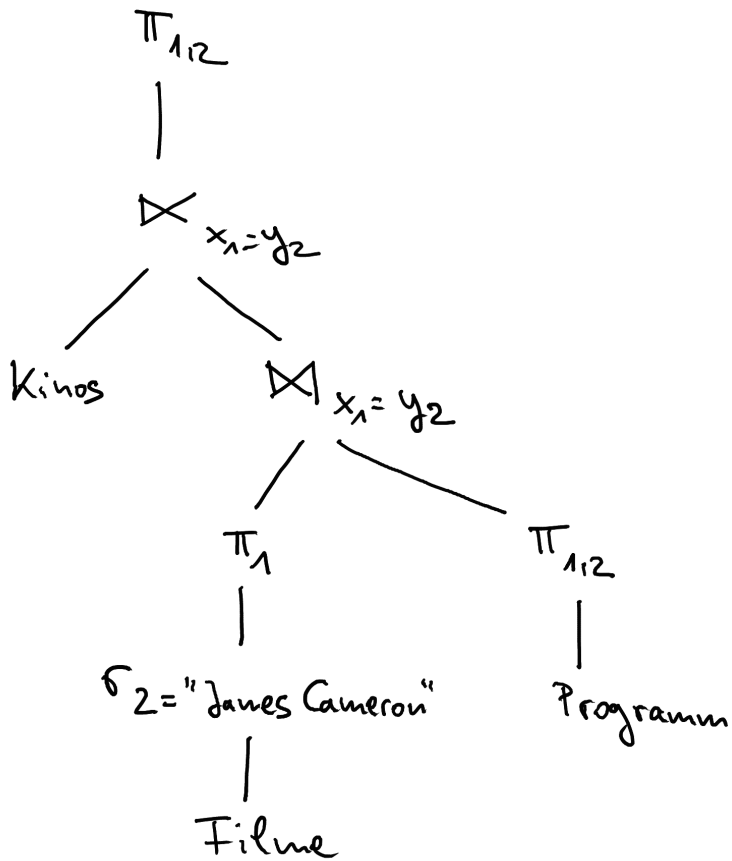
- *Filme*: 10.000 Tupel auf 200 Seiten (je 50 pro Seite); je 5 Tupel pro Film, 10 Filme von James Cameron
- *Programm*: 200 Tupel auf 4 Seiten (je 50 pro Seite); davon 3 Cameron-Filme in 4 Kinos
- *Kinos*: 100 Tupel auf 2 Seiten (je 50 pro Seite)

Anfrage-Baum:



Direkte Auswertung dieser Anfrage führt zu über 10.000.000 Plattenzugriffen.

Viel besserer Plan:



Auswertung dieses Plans in unserer Beispiel-Datenbank führt zu weniger als 250 Plattenzugriffen.

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

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)

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)

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.

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:

$\pi_X(Q_1 \bowtie_\theta Q_2) \quad \longleftrightarrow \quad \pi_X(Q_1 \ltimes_\theta Q_2)$

(5) Vertauschen von Selektion und Vereinigung bzw. Differenz:

$$\sigma_F(Q_1 \cup Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) \cup \sigma_F(Q_2)$$

$$\sigma_F(Q_1 - Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) - \sigma_F(Q_2)$$

(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.

(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

(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

(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:

$$Q \cup Q \quad \longrightarrow \quad Q, \qquad Q \cap Q \quad \longrightarrow \quad Q, \qquad Q \bowtie Q \quad \longrightarrow \quad Q$$

(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:

$$Q - Q \longrightarrow \emptyset, \qquad Q \cap \emptyset \longrightarrow \emptyset, \qquad Q \cup \emptyset \longrightarrow Q, \qquad Q \bowtie \emptyset \longrightarrow \emptyset,$$
$$Q - \emptyset \longrightarrow Q, \qquad \emptyset - Q \longrightarrow \emptyset$$

(11) . . . usw. . . .

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 ("$Q \equiv P$")

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

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
  ($\rightsquigarrow$ kostenbasierte Optimierung)
- Auswahl des am günstigsten erscheinenden Plans

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
  ($\rightsquigarrow$ 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.

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)

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)

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

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 ein Tupel $(\cdot, b) \in R$ und ein Tupel $(c, \cdot) \in S$.

Anfrage: $Ans(x) \leftarrow R(x, x_1),\ S(x_2, x_3),\ T(x_1, x_2)$

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 ein Tupel $(\cdot, b) \in R$ und ein 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

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 ein Tupel $(\cdot, b) \in R$ und ein 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

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 ein Tupel $(\cdot, b) \in R$ und ein 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

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 ein Tupel $(\cdot, b) \in R$ und ein 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

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

Noch ein Beispiel: a und b seien jetzt Konstanten, d.h. $a, b \in$ **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.

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 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

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

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

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.

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

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

Abschnitt 7.1:

CALC_{nat}, CALC_{adom} und CALC_{di}

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")

Die Logik erster Stufe — FO[**S**]

Definition 7.1

Sei **S** ein Datenbankschema.
Die Menge FO[**S**] aller Formeln der Logik erster Stufe über **S** ist induktiv wie folgt definiert:

(A) "Relations-Atome": $R(v_1, \ldots, v_r)$ gehört zu FO[**S**], für alle $R \in \mathbf{S}$, $r := \text{ar}(R)$ und $v_1, \ldots, v_r \in \text{var} \cup \mathbf{dom}$.

(G) "Gleichheits-Atome": $v_1 = v_2$ gehört zu FO[**S**], für alle $v_1, v_2 \in \text{var} \cup \mathbf{dom}$.

(BK) "Boolesche Kombinationen": Falls $\varphi_1 \in \text{FO}[\mathbf{S}]$ und $\varphi_2 \in \text{FO}[\mathbf{S}]$, so gehört auch jede der folgenden fünf Formeln zu FO[**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 $\varphi_1 \in \text{FO}[\mathbf{S}]$ und ist $x \in \text{var}$, so gehören auch die beiden folgenden Formeln zu FO[**S**]: $\exists x\, \varphi_1$ und $\forall x\, \varphi_1$.

Die Logik erster Stufe — FO[**S**]

Definition 7.1

Sei **S** ein Datenbankschema.
Die Menge FO[**S**] aller Formeln der Logik erster Stufe über **S** ist induktiv wie folgt definiert:

(A) "Relations-Atome": $R(v_1, \ldots, v_r)$ gehört zu FO[**S**], für alle $R \in \mathbf{S}$, $r := \text{ar}(R)$ und $v_1, \ldots, v_r \in \text{var} \cup \mathbf{dom}$.

(G) "Gleichheits-Atome": $v_1 = v_2$ gehört zu FO[**S**], für alle $v_1, v_2 \in \text{var} \cup \mathbf{dom}$.

(BK) "Boolesche Kombinationen": Falls $\varphi_1 \in \text{FO}[\mathbf{S}]$ und $\varphi_2 \in \text{FO}[\mathbf{S}]$, so gehört auch jede der folgenden fünf Formeln zu FO[**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 $\varphi_1 \in \text{FO}[\mathbf{S}]$ und ist $x \in \text{var}$, so gehören auch die beiden folgenden Formeln zu FO[**S**]: $\exists x\, \varphi_1$ und $\forall x\, \varphi_1$.

Bemerkung: Benutzen wir die Notation aus der Vorlesung *„Logik in der Informatik"*, so ist FO[**S**] genau die Menge aller Formeln aus FO[σ], 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).

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.

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.

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.

- 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.
  - $\text{frei}\big(R(v_1, \ldots, v_r)\big) := \{v_1, \ldots, v_r\} \cap \text{var}$
  - $\text{frei}\big(v_1{=}v_2\big) := \{v_1, v_2\} \cap \text{var}$
  - $\text{frei}\big(\neg\varphi_1\big) := \text{frei}\big(\varphi_1\big)$
  - $\text{frei}\big((\varphi_1 * \varphi_2)\big) := \text{frei}\big(\varphi_1\big) \cup \text{frei}\big(\varphi_2\big)$ (für alle $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$)
  - $\text{frei}\big(\exists x\, \varphi_1\big) := \text{frei}\big(\forall x\, \varphi_1\big) := \text{frei}\big(\varphi_1\big) \setminus \{x\}$

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.

- 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.
  - $\text{frei}\big(R(v_1, \ldots, v_r)\big) := \{v_1, \ldots, v_r\} \cap \text{var}$
  - $\text{frei}\big(v_1{=}v_2\big) := \{v_1, v_2\} \cap \text{var}$
  - $\text{frei}\big(\neg\varphi_1\big) := \text{frei}\big(\varphi_1\big)$
  - $\text{frei}\big((\varphi_1 * \varphi_2)\big) := \text{frei}\big(\varphi_1\big) \cup \text{frei}\big(\varphi_2\big)$ (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 $\beta : \text{frei}(\varphi) \rightarrow \mathbf{dom}$.

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.

- $\mathrm{adom}(\varphi)$ bezeichnet die Menge aller Konstanten (also Elemente aus **dom**), die in φ vorkommen.
  $\mathrm{Var}(\varphi)$ bezeichnet die Menge aller Variablen (also Elemente aus var, die in φ vorkommen.
  $\mathrm{frei}(\varphi)$ bezeichnet die Menge aller Variablen, die frei in φ vorkommen. D.h.
  - $\mathrm{frei}\big(R(v_1, \ldots, v_r)\big) := \{v_1, \ldots, v_r\} \cap \mathrm{var}$
  - $\mathrm{frei}\big(v_1{=}v_2\big) := \{v_1, v_2\} \cap \mathrm{var}$
  - $\mathrm{frei}\big(\neg\varphi_1\big) := \mathrm{frei}\big(\varphi_1\big)$
  - $\mathrm{frei}\big((\varphi_1 * \varphi_2)\big) := \mathrm{frei}\big(\varphi_1\big) \cup \mathrm{frei}\big(\varphi_2\big)$ (für alle $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$)
  - $\mathrm{frei}\big(\exists x\, \varphi_1\big) := \mathrm{frei}\big(\forall x\, \varphi_1\big) := \mathrm{frei}\big(\varphi_1\big) \setminus \{x\}$

- Eine Belegung für φ ist eine Abbildung $\beta : \mathrm{frei}(\varphi) \rightarrow$ **dom**.
- Sei **d** $\subseteq$ **dom**. Eine Belegung für φ in **d** ist eine Abbildung $\beta : \mathrm{frei}(\varphi) \rightarrow$ **d**.

Natürliche Semantik von FO[**S**] und CALC

Wir werden verschiedene Varianten der Semantik von FO[**S**] betrachten.
Hier zunächst die natürliche Semantik:

Definition 7.2 (natürliche Semantik von FO[**S**])

- Zur Erinnerung: Einer Datenbank **I** vom Schema **S** ordnen wir die logische Struktur $\mathcal{A}_{\mathbf{I}} := \left(\mathbf{dom}, \left(\mathbf{I}(R)\right)_{R\in\mathbf{S}}, (c)_{c\in\mathbf{dom}}\right)$ zu.
- Ist **I** eine Datenbank vom Schema **S** und β eine Belegung für φ (also eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{dom}$), so sagen wir "**I** erfüllt φ unter β" und schreiben $\mathbf{I} \models \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models \varphi$, um auszudrücken, dass $(\mathcal{A}_{\mathbf{I}}, \beta) \models \varphi$.

Natürliche Semantik von FO[**S**] und CALC

Wir werden verschiedene Varianten der Semantik von FO[**S**] betrachten.
Hier zunächst die natürliche Semantik:

Definition 7.2 (natürliche Semantik von FO[**S**])

- Zur Erinnerung: Einer Datenbank **I** vom Schema **S** ordnen wir die logische Struktur $\mathcal{A}_{\mathbf{I}} := \left(\mathbf{dom},\ \left(\mathbf{I}(R)\right)_{R\in\mathbf{S}},\ (c)_{c\in\mathbf{dom}}\right)$ zu.
- Ist **I** eine Datenbank vom Schema **S** und β eine Belegung für φ (also eine Abbildung $\beta : \mathrm{frei}(\varphi) \to \mathbf{dom}$), so sagen wir "**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 **S** ein Datenbankschema. Eine Anfrage des Relationenkalküls CALC[**S**] ist von der Form $\{\ (e_1, \ldots, e_r)\ :\ \varphi\ \}$,
wobei $\varphi \in \mathrm{FO}[\mathbf{S}]$, $r \geqslant 0$ und $(e_1, \ldots, e_r)$ ein freies Tupel ist (d.h. $e_1, \ldots, e_r \in \mathrm{var} \cup \mathbf{dom}$), so dass $\mathrm{frei}(\varphi) = \{e_1, \ldots, e_r\} \cap \mathrm{var}$.
Wir setzen $\mathrm{adom}(Q) := \mathrm{adom}(\varphi) \cup (\{e_1, \ldots, e_r\} \cap \mathbf{dom})$.

Natürliche Semantik von FO[**S**] und CALC

Wir werden verschiedene Varianten der Semantik von FO[**S**] betrachten.
Hier zunächst die natürliche Semantik:

Definition 7.2 (natürliche Semantik von FO[**S**])

- Zur Erinnerung: Einer Datenbank **I** vom Schema **S** ordnen wir die logische Struktur $\mathcal{A}_{\mathbf{I}} := \left(\mathbf{dom}, \left(\mathbf{I}(R)\right)_{R\in\mathbf{S}}, (c)_{c\in\mathbf{dom}}\right)$ zu.
- Ist **I** eine Datenbank vom Schema **S** und β eine Belegung für φ (also eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{dom}$), so sagen wir "**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 **S** ein Datenbankschema. Eine Anfrage des Relationenkalküls CALC[**S**] ist von der Form $\{ (e_1, \ldots, e_r) : \varphi \}$,
wobei $\varphi \in \text{FO}[\mathbf{S}]$, $r \geqslant 0$ und $(e_1, \ldots, e_r)$ ein freies Tupel ist (d.h. $e_1, \ldots, e_r \in \text{var} \cup \mathbf{dom}$), so dass $\text{frei}(\varphi) = \{e_1, \ldots, e_r\} \cap \text{var}$.
Wir setzen $\text{adom}(Q) := \text{adom}(\varphi) \cup (\{e_1, \ldots, e_r\} \cap \mathbf{dom})$.

Semantik: Einer CALC[**S**]-Anfrage Q der obigen Form ordnen wir die folgende "Anfragefunktion" $[\![Q]\!]_{nat}$ zu (f.a. $\mathbf{I} \in \mathit{inst}(\mathbf{S})$):

$$[\![Q]\!]_{nat}(\mathbf{I}) := \{ \beta\big((e_1, \ldots, e_r)\big) : \beta \text{ ist eine Belegung für } \varphi \text{ mit } \mathbf{I} \models \varphi[\beta] \}$$

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{"George Clooney"}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{"George Clooney"}, \text{"George Clooney"}) \Big) \Big\}$$

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \ : \ \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

- Welche Filme laufen nur zu 1 Uhrzeit?

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

- Welche Filme laufen nur zu 1 Uhrzeit?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Kino}\, \exists x_{Zeit} \Big(\mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \forall y_{Kino}\, \forall y_{Zeit} \big(\mathit{Programm}(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit} = x_{Zeit} \big) \Big)$$

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

- Welche Filme laufen nur zu 1 Uhrzeit?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Kino}\, \exists x_{Zeit} \Big(\mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \forall y_{Kino}\, \forall y_{Zeit} \big(\mathit{Programm}(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit} = x_{Zeit} \big) \Big)$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

- Welche Filme laufen nur zu 1 Uhrzeit?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Kino}\, \exists x_{Zeit} \Big(\mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \forall y_{Kino}\, \forall y_{Zeit} \big(\mathit{Programm}(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit} = x_{Zeit} \big) \Big)$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{ (x_T) \;:\; \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \;\wedge\; \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

Beispiele für CALC-Anfragen

- In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

- 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 . . . ?

$$\Big\{ (x_T) \;:\; \forall y_S \Big(\exists x_R\, \mathit{Filme}(x_T, x_R, y_S) \;\rightarrow\; \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \{ (x_{Schausp}) \ : \ \neg Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \}$

Auswertung mit $[\![\cdot]\!]_{\text{nat}}$:

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \{ (x_{Schausp}) \ : \ \neg Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \}$

Auswertung mit $[\![\cdot]\!]_{\mathbf{nat}}$: Q_1 liefert alle (unendlich vielen) $a \in \mathbf{dom}$, die *nicht* als Schauspieler im Film “Alien” mitgespielt haben.

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \{ (x_{Schausp}) \ : \ \neg \mathit{Filme}(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \}$

(2) $Q_2 := \{ (x_S, y_T) \ : \ \mathit{Filme}(\text{“Alien”}, \text{“Ridley Scott”}, x_S) \vee \mathit{Filme}(y_T, \text{“George Clooney”}, \text{“George Clooney”}) \}$

Auswertung mit $[\![\cdot]\!]_{\text{nat}}$: Q_1 liefert alle (unendlich vielen) $a \in \textbf{dom}$, die *nicht* als Schauspieler im Film “Alien” mitgespielt haben.

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \{ (x_{Schausp}) \ : \ \neg Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \}$

(2) $Q_2 := \{ (x_S, y_T) \ : \ Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_S) \vee Filme(y_T, \text{“George Clooney”}, \text{“George Clooney”}) \}$

Auswertung mit $[\![\cdot]\!]_{\text{nat}}$: 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**.

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \{ (x_{Schausp}) \ : \ \neg Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \}$

(2) $Q_2 := \{ (x_S, y_T) \ : \ Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_S) \vee Filme(y_T, \text{“George Clooney”}, \text{“George Clooney”}) \}$

(3) $Q_3 :=$

$$\Big\{ (x_T) \ : \ \forall y_S \Big(\exists x_R \, Filme(x_T, x_R, y_S) \ \rightarrow \ \exists y_T \, Filme(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

Auswertung mit $[\![\cdot]\!]_{\text{nat}}$: 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**.

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \{ (x_{Schausp}) \ : \ \neg Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \}$

(2) $Q_2 := \{ (x_S, y_T) \ : \ Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_S) \vee Filme(y_T, \text{“George Clooney”}, \text{“George Clooney”}) \}$

(3) $Q_3 := \left\{ (x_T) \ : \ \forall y_S \left(\exists x_R \, Filme(x_T, x_R, y_S) \ \rightarrow \ \exists y_T \, Filme(y_T, \text{“Stephen Spielberg”}, y_S) \right) \right\}$

Auswertung mit $[\![\cdot]\!]_{\text{nat}}$: 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.

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 := \{ (x_{Schausp}) \ : \ \neg Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_{Schausp}) \}$

(2) $Q_2 := \{ (x_S, y_T) \ : \ Filme(\text{“Alien”}, \text{“Ridley Scott”}, x_S) \vee Filme(y_T, \text{“George Clooney”}, \text{“George Clooney”}) \}$

(3) $Q_3 :=$
$$\Big\{ (x_T) \ : \ \forall y_S \Big(\exists x_R \, Filme(x_T, x_R, y_S) \ \rightarrow \ \exists y_T \, Filme(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

Auswertung mit $[\![\cdot]\!]_{\text{nat}}$: 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.

Eine weitere problematische Anfrage

$$Q_4 := \{ (x) \ : \ \forall y \, R(x, y) \}$$

Eine weitere problematische Anfrage

$Q_4 := \{ (x) \ : \ \forall y \, R(x, y) \}$

Q_4 liefert als Ergebnis stets die leere Relation, d.h.
$[\![Q_4]\!]_{nat}(\mathbf{I}) = \emptyset$, für alle $\mathbf{I} \in inst(\mathbf{S})$.

Dies ist unschön.

Daher (und weil manche Anfragen unsicher sind in der nat. Semantik), betrachten wir im Folgenden andere Varianten der Semantik.

Relativierte Semantik von FO[**S**] und CALC[**S**]

Definition 7.4

Sei **S** ein Datenbankschema.

(a) Eine relativierte Datenbank über **S** ist ein Paar $(\mathbf{d}, \mathbf{I})$, wobei **I** eine Datenbank vom Schema **S** ist und **d** eine Menge, so dass $\mathrm{adom}(\mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$.

(b) Einer relativierten Datenbank $(\mathbf{d}, \mathbf{I})$ über **S** ordnen wir die logische Struktur $\mathcal{A}_{(\mathbf{d},\mathbf{I})} := \left(\mathbf{d},\ \left(\mathbf{I}(R)\right)_{R\in\mathbf{S}},\ (c)_{c\in\mathbf{d}}\right)$ zu.

Relativierte Semantik von FO[**S**] und CALC[**S**]

Definition 7.4

Sei **S** ein Datenbankschema.

(a) Eine relativierte Datenbank über **S** ist ein Paar $(\mathbf{d}, \mathbf{I})$, wobei **I** eine Datenbank vom Schema **S** ist und **d** eine Menge, so dass $\text{adom}(\mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$.

(b) Einer relativierten Datenbank $(\mathbf{d}, \mathbf{I})$ über **S** ordnen wir die logische Struktur $\mathcal{A}_{(\mathbf{d},\mathbf{I})} := \big(\mathbf{d},\ (\mathbf{I}(R))_{R\in\mathbf{S}},\ (c)_{c\in\mathbf{d}}\big)$ zu.

(c) Eine FO[**S**]-Formel φ heißt interpretierbar über $(\mathbf{d}, \mathbf{I})$, falls $\text{adom}(\varphi) \subseteq \mathbf{d}$.
Eine CALC[**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 **S**, ist φ eine FO[**S**]-Formel, die interpretierbar ist über $(\mathbf{d}, \mathbf{I})$, und ist β eine Belegung für φ in **d** (also eine Abbildung $\beta : \text{frei}(\varphi) \to \mathbf{d}$), so sagen wir "**I** erfüllt φ unter β relativ zu **d**" und schreiben $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models_{\mathbf{d}} \varphi$, um auszudrücken, dass $(\mathcal{A}_{(\mathbf{d},\mathbf{I})}, \beta) \models \varphi$.

Relativierte Semantik von FO[**S**] und CALC[**S**]

Definition 7.4

Sei **S** ein Datenbankschema.

(a) Eine relativierte Datenbank über **S** ist ein Paar $(\mathbf{d}, \mathbf{I})$, wobei $\mathbf{I}$ eine Datenbank vom Schema **S** ist und $\mathbf{d}$ eine Menge, so dass $\mathrm{adom}(\mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$.

(b) Einer relativierten Datenbank $(\mathbf{d}, \mathbf{I})$ über **S** ordnen wir die logische Struktur $\mathcal{A}_{(\mathbf{d},\mathbf{I})} := \big(\mathbf{d},\ (\mathbf{I}(R))_{R\in\mathbf{S}},\ (c)_{c\in\mathbf{d}}\big)$ zu.

(c) Eine FO[**S**]-Formel φ heißt interpretierbar über $(\mathbf{d}, \mathbf{I})$, falls $\mathrm{adom}(\varphi) \subseteq \mathbf{d}$.
Eine CALC[**S**]-Anfrage Q heißt interpretierbar über $(\mathbf{d}, \mathbf{I})$, falls $\mathrm{adom}(Q) \subseteq \mathbf{d}$.

(d) Ist $(\mathbf{d}, \mathbf{I})$ eine relativierte Datenbank über **S**, ist φ eine FO[**S**]-Formel, die interpretierbar ist über $(\mathbf{d}, \mathbf{I})$, und ist β eine Belegung für φ in $\mathbf{d}$ (also eine Abbildung $\beta : \mathrm{frei}(\varphi) \to \mathbf{d}$), 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[**S**]-Anfrage der Form $\{(e_1, \ldots, e_r) : \varphi\}$ und $(\mathbf{d}, \mathbf{I})$ eine relativierte Datenbank über **S**, über der Q interpretierbar ist, so ist

$$[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) := \{\, \beta\big((e_1, \ldots, e_r)\big) : \beta \text{ ist eine Belegung für } \varphi \text{ über } \mathbf{d} \text{ mit } \mathbf{I} \models_{\mathbf{d}} \varphi[\beta] \,\}$$

("Relativiert" soll andeuten, dass Quantifizierung relativiert wird auf Elemente aus $\mathbf{d}$.)

Ein Beispiel

Beispiel 7.5

- **S** enthalte ein 1-stelliges Relationssymbol R
- **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[**S**]-Anfrage

$$Q := \Big\{ \; (x) \; : \; \Big(\; R(x) \; \wedge \; \exists y \, \big(\neg R(y) \; \wedge \; \forall z \, (R(z) \vee z{=}y) \big) \; \Big) \; \Big\}$$

Ein Beispiel

Beispiel 7.5

- **S** enthalte ein 1-stelliges Relationssymbol R
- **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[**S**]-Anfrage

$$Q := \left\{ (x) : \Big(R(x) \wedge \exists y \big(\neg R(y) \wedge \forall z \, (R(z) \vee z{=}y)\big) \Big) \right\}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) =$

Ein Beispiel

Beispiel 7.5

- **S** enthalte ein 1-stelliges Relationssymbol R
- **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[**S**]-Anfrage

$$Q := \{ (x) : (R(x) \wedge \exists y (\neg R(y) \wedge \forall z (R(z) \vee z{=}y))) \}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) = \emptyset$

Ein Beispiel

Beispiel 7.5

- **S** enthalte ein 1-stelliges Relationssymbol R
- **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[**S**]-Anfrage

$$Q := \{ (x) : (R(x) \wedge \exists y (\neg R(y) \wedge \forall z (R(z) \vee z{=}y))) \}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) = \emptyset$
  - $[\![Q]\!]_{\{a,b,c,d\}}(\mathbf{I}) =$

Ein Beispiel

Beispiel 7.5

- **S** enthalte ein 1-stelliges Relationssymbol R
- **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[**S**]-Anfrage

$$Q := \{ (x) : (R(x) \wedge \exists y (\neg R(y) \wedge \forall z (R(z) \vee z{=}y))) \}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) = \emptyset$
  - $[\![Q]\!]_{\{a,b,c,d\}}(\mathbf{I}) = \{(a), (b), (c)\}$

Ein Beispiel

Beispiel 7.5

- **S** enthalte ein 1-stelliges Relationssymbol R
- **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[**S**]-Anfrage

$$Q := \{ (x) : (R(x) \wedge \exists y (\neg R(y) \wedge \forall z (R(z) \vee z{=}y))) \}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) = \emptyset$
  - $[\![Q]\!]_{\{a,b,c,d\}}(\mathbf{I}) = \{(a), (b), (c)\}$
  - $[\![Q]\!]_{\{a,b,c,d,e\}}(\mathbf{I}) =$

Ein Beispiel

Beispiel 7.5

- **S** enthalte ein 1-stelliges Relationssymbol R
- **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[**S**]-Anfrage

$$Q := \{ (x) : (R(x) \wedge \exists y (\neg R(y) \wedge \forall z (R(z) \vee z{=}y))) \}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) = \emptyset$
  - $[\![Q]\!]_{\{a,b,c,d\}}(\mathbf{I}) = \{(a), (b), (c)\}$
  - $[\![Q]\!]_{\{a,b,c,d,e\}}(\mathbf{I}) = \emptyset$

Active Domain Semantik von FO[**S**] und CALC[**S**]

Definition 7.6

(a) Ist $\mathbf{I} \in inst(\mathbf{S})$ und $\varphi \in \text{FO}[\mathbf{S}]$, so ist $\text{adom}(\varphi, \mathbf{I}) := \text{adom}(\varphi) \cup \text{adom}(\mathbf{I})$.
Ist $\mathbf{I} \in inst(\mathbf{S})$ und $Q \in \text{CALC}[\mathbf{S}]$, so ist $\text{adom}(Q, \mathbf{I}) := \text{adom}(Q) \cup \text{adom}(\mathbf{I})$.

Active Domain Semantik von FO[**S**] und CALC[**S**]

Definition 7.6

(a) Ist $\mathbf{I} \in inst(\mathbf{S})$ und $\varphi \in \text{FO}[\mathbf{S}]$, so ist $\text{adom}(\varphi, \mathbf{I}) := \text{adom}(\varphi) \cup \text{adom}(\mathbf{I})$.
Ist $\mathbf{I} \in inst(\mathbf{S})$ und $Q \in \text{CALC}[\mathbf{S}]$, so ist $\text{adom}(Q, \mathbf{I}) := \text{adom}(Q) \cup \text{adom}(\mathbf{I})$.

(b) Ist $\varphi \in \text{FO}[\mathbf{S}]$, $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$ und β eine Belegung für φ in $\text{adom}(\varphi, \mathbf{I})$ (also eine Abbildung $\beta : \text{frei}(\varphi) \to \text{adom}(\varphi, \mathbf{I})$), so sagen wir "$\mathbf{I}$ 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 $\mathbf{d} := \text{adom}(\varphi, \mathbf{I})$ gilt.

Active Domain Semantik von FO[**S**] und CALC[**S**]

Definition 7.6

(a) Ist $\mathbf{I} \in inst(\mathbf{S})$ und $\varphi \in \text{FO}[\mathbf{S}]$, so ist $\text{adom}(\varphi, \mathbf{I}) := \text{adom}(\varphi) \cup \text{adom}(\mathbf{I})$.
Ist $\mathbf{I} \in inst(\mathbf{S})$ und $Q \in \text{CALC}[\mathbf{S}]$, so ist $\text{adom}(Q, \mathbf{I}) := \text{adom}(Q) \cup \text{adom}(\mathbf{I})$.

(b) Ist $\varphi \in \text{FO}[\mathbf{S}]$, $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{S}$ und β eine Belegung für φ in $\text{adom}(\varphi, \mathbf{I})$ (also eine Abbildung $\beta : \text{frei}(\varphi) \to \text{adom}(\varphi, \mathbf{I})$), so sagen wir "$\mathbf{I}$ 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 $\mathbf{d} := \text{adom}(\varphi, \mathbf{I})$ gilt.

(c) Ist Q eine CALC[**S**]-Anfrage der Form $\{(e_1, \ldots, e_r) : \varphi\}$, so definiert Q in der Active Domain Semantik die folgende Anfragefunktion

$$[\![Q]\!]_{adom}(\mathbf{I}) := [\![Q]\!]_{\text{adom}(Q,\mathbf{I})}(\mathbf{I})$$

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).

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: $Q := \{ (x) : (R(x) \wedge \exists y(\neg R(y) \wedge \forall z(R(z) \vee z{=}y))) \}$

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: $Q := \{ (x) : (R(x) \wedge \exists y(\neg R(y) \wedge \forall z(R(z) \vee z{=}y))) \}$

- 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 $\mathrm{adom}(Q, \mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$ auswertet werden.

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: $Q := \{ (x) : (R(x) \wedge \exists y(\neg R(y) \wedge \forall z(R(z) \vee z{=}y))) \}$

- 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 $\text{adom}(Q, \mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$ 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 $\text{adom}(Q, \mathbf{I}) \subseteq \mathbf{d}, \mathbf{d}'$ gilt: $[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) = [\![Q]\!]_{\mathbf{d}'}(\mathbf{I})$.

Beispiele 7.8

Beispiele für bereichsunabhängige Anfragen:

(a) Welche Filme laufen in mindestens 2 Kinos?

$$\Big\{ (x_T) \;:\; \exists x_K \, \exists x_Z \, \exists y_K \, \exists y_Z \big(\mathit{Programm}(x_K, x_T, x_Z) \wedge \mathit{Programm}(y_K, x_T, y_Z) \wedge \neg x_K = y_K \big) \Big\}$$

Beispiele 7.8

Beispiele für bereichsunabhängige Anfragen:

(a) Welche Filme laufen in mindestens 2 Kinos?

$$\Big\{ (x_T) \;:\; \exists x_K \, \exists x_Z \, \exists y_K \, \exists y_Z \big(\mathit{Programm}(x_K, x_T, x_Z) \wedge \mathit{Programm}(y_K, x_T, y_Z) \wedge \neg x_K = y_K \big) \Big\}$$

(b) In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

Beispiele 7.8

Beispiele für bereichsunabhängige Anfragen:

(a) Welche Filme laufen in mindestens 2 Kinos?

$$\Big\{ (x_T) \;:\; \exists x_K \, \exists x_Z \, \exists y_K \, \exists y_Z \big(\mathit{Programm}(x_K, x_T, x_Z) \wedge \mathit{Programm}(y_K, x_T, y_Z) \wedge \neg x_K = y_K \big) \Big\}$$

(b) In welchen Filmen hat "George Clooney" mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Regie} \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

(c) Welche Filme laufen nur zu 1 Uhrzeit?

$$\Big\{ (x_{Titel}) \;:\; \exists x_{Kino} \, \exists x_{Zeit} \Big(\mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \forall y_{Kino} \, \forall y_{Zeit} \big(\mathit{Programm}(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit} = x_{Zeit} \big) \Big) \Big\}$$

(d) Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{ \; (x_T) \; : \; \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \; \wedge \; \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \; \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

(d) Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{ (x_T) \;:\; \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \;\wedge\; \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \; \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

Nicht bereichunabhängig ist ...

$$\Big\{ (x_{Titel}) \;:\; \forall y_S \Big(\exists x_R \; \mathit{Filme}(x_{Titel}, x_R, y_S) \;\rightarrow\; \exists y_T \; \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

CALC_{nat}, CALC_{adom}, CALC_{di}

Definition 7.9

$\mathrm{CALC}_{nat}[\mathbf{S}]$ bezeichnet die Klasse aller in der natürlichen Semantik $[\![\cdot]\!]_{nat}$ interpretierten CALC[**S**]-Anfragen.

$\mathrm{CALC}_{adom}[\mathbf{S}]$ bezeichnet die Klasse aller in der Active Domain Semantik $[\![\cdot]\!]_{adom}$ interpretierten CALC[**S**]-Anfragen.

$\mathrm{CALC}_{di}[\mathbf{S}]$ bezeichnet die Klasse aller bereichsunabhängigen CALC[**S**]-Anfragen (interpretiert in $[\![\cdot]\!]_{adom}$ oder, äquivalent dazu, in $[\![\cdot]\!]_{nat}$).
("di" steht für "domain independent")

CALC_{nat}, CALC_{adom}, CALC_{di}

Definition 7.9

$\mathrm{CALC}_{nat}[\mathbf{S}]$ bezeichnet die Klasse aller in der natürlichen Semantik $[\![\cdot]\!]_{nat}$ interpretierten CALC[**S**]-Anfragen.

$\mathrm{CALC}_{adom}[\mathbf{S}]$ bezeichnet die Klasse aller in der Active Domain Semantik $[\![\cdot]\!]_{adom}$ interpretierten CALC[**S**]-Anfragen.

$\mathrm{CALC}_{di}[\mathbf{S}]$ bezeichnet die Klasse aller bereichsunabhängigen CALC[**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 **S** *gilt: Jede Anfrage aus einer dieser Sprachen kann in polynomieller Zeit in äquivalente Anfragen der anderen Sprachen übersetzt werden.*

Beweis: Siehe Tafel.

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

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?

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 ...

Der Satz von Trakhtenbrot

Theorem 7.11 (Trakhtenbrot, 1950 (hier ohne Beweis))

Sei $\mathbf{S}$ *ein Datenbankschema, das mindestens ein Relationssymbol enthält, dessen Stelligkeit* $\geqslant 2$ *ist. Dann ist das folgende Problem unentscheidbar:*

ERFÜLLBARKEITSPROBLEM FÜR FO[$\mathbf{S}$] ÜBER DATENBANKEN

Eingabe: Ein FO[$\mathbf{S}$]-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*“.

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 CALC_{di}[**S**]) ?

Beweis: Einfache Folgerung aus dem Satz von Trakhtenbrot; siehe Tafel.

Abschnitt 7.2:

Sicherer Relationenkalkül: CALC_{sr}

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 CALC[**S**] nicht feststellen kann, ob dies eine Anfrage in CALC_{di} ist.

- Ziel jetzt: Finde ein syntaktisches Kriterium für CALC[**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.

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 CALC[**S**] nicht feststellen kann, ob dies eine Anfrage in CALC_{di} ist.

- Ziel jetzt: Finde ein syntaktisches Kriterium für CALC[**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.

Safe-Range Normalform (SRNF)

Für jede FO[**S**]-Formel φ sei SRNF(φ) die FO[**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.

Safe-Range Normalform (SRNF)

Für jede FO[**S**]-Formel φ sei SRNF(φ) die FO[**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 $\neg \exists x \neg \psi$

Safe-Range Normalform (SRNF)

Für jede FO[**S**]-Formel φ sei SRNF(φ) die FO[**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 $((\psi_1 \wedge \psi_2) \vee (\neg\psi_1 \ \wedge \ \neg\psi_2))$

Safe-Range Normalform (SRNF)

Für jede FO[**S**]-Formel φ sei SRNF(φ) die FO[**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 $((\psi_1 \wedge \psi_2) \vee (\neg\psi_1 \ \wedge \ \neg\psi_2))$
- 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 $(\neg\psi_1 \wedge \neg\psi_2)$

Safe-Range Normalform (SRNF)

Für jede FO[**S**]-Formel φ sei SRNF(φ) die FO[**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 $((\psi_1 \wedge \psi_2) \vee (\neg\psi_1 \ \wedge \ \neg\psi_2))$
- 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

Safe-Range Normalform (SRNF)

Für jede FO[**S**]-Formel φ sei SRNF(φ) die FO[**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 $((\psi_1 \wedge \psi_2) \vee (\neg\psi_1 \ \wedge \ \neg\psi_2))$
- 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: φ ist in Safe-Range Normalform (kurz: SRNf).

Beispiel für Transformation von φ zu SRNF(φ)

Beispiel 7.13

$$\varphi := \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

Beispiel für Transformation von φ zu SRNF(φ)

Beispiel 7.13

$$\varphi := \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg \exists y_S \neg \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

Beispiel für Transformation von φ zu SRNF(φ)

Beispiel 7.13

$$\varphi := \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg \exists y_S \neg \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg \exists y_S \neg \big(\neg \mathit{Filme}(x_T, x_R, y_S) \vee \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

Beispiel für Transformation von φ zu SRNF(φ)

Beispiel 7.13

$$\varphi := \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg \exists y_S \neg \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg \exists y_S \neg \big(\neg \mathit{Filme}(x_T, x_R, y_S) \vee \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg \exists y_S \big(\neg\neg \mathit{Filme}(x_T, x_R, y_S) \wedge \neg \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

Beispiel für Transformation von φ zu SRNF(φ)

Beispiel 7.13

$$\varphi := \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \quad \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \neg \exists y_S \neg \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \quad \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \neg \exists y_S \neg \big(\neg \mathit{Filme}(x_T, x_R, y_S) \vee \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \quad \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \neg \exists y_S \big(\neg\neg \mathit{Filme}(x_T, x_R, y_S) \wedge \neg \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \quad \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \neg \exists y_S \big(\mathit{Filme}(x_T, x_R, y_S) \wedge \neg \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

Beispiel für Transformation von φ zu SRNF(φ)

Beispiel 7.13

$$\varphi := \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg\exists y_S \neg \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg\exists y_S \neg \big(\neg \mathit{Filme}(x_T, x_R, y_S) \vee \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg\exists y_S \big(\neg\neg \mathit{Filme}(x_T, x_R, y_S) \wedge \neg\exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$\rightsquigarrow \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \wedge \neg\exists y_S \big(\mathit{Filme}(x_T, x_R, y_S) \wedge \neg\exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

$$=: \mathrm{SRNF}(\varphi)$$

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")

- $rr\left(R(v_1, \ldots, v_{\mathrm{ar}(R)})\right) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \mathrm{var}$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \mathrm{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \mathrm{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$
- Für $x, y \in \mathrm{var}$ gilt:

$$rr\big(\psi \wedge x{=}y\big) := rr\big(x{=}y \wedge \psi\big) := \begin{cases} rr(\psi) \cup \{x, y\} & \text{falls } \{x, y\} \cap rr(\psi) \neq \emptyset \\ rr(\psi) & \text{sonst} \end{cases}$$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \mathrm{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$
- Für $x, y \in \mathrm{var}$ gilt:
  $$rr\big(\psi \wedge x{=}y\big) := rr\big(x{=}y \wedge \psi\big) := \begin{cases} rr(\psi) \cup \{x, y\} & \text{falls } \{x, y\} \cap rr(\psi) \neq \emptyset \\ rr(\psi) & \text{sonst} \end{cases}$$
- Ist weder φ_1 noch φ_2 von der Form $x{=}y$ für $x, y \in \mathrm{var}$, so gilt:
  $rr\big(\varphi_1 \wedge \varphi_2\big) := rr(\varphi_1) \cup rr(\varphi_2)$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \mathrm{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$
- Für $x, y \in \mathrm{var}$ gilt:
  $$rr\big(\psi \wedge x{=}y\big) := rr\big(x{=}y \wedge \psi\big) := \begin{cases} rr(\psi) \cup \{x, y\} & \text{falls } \{x, y\} \cap rr(\psi) \neq \emptyset \\ rr(\psi) & \text{sonst} \end{cases}$$
- Ist weder φ_1 noch φ_2 von der Form $x{=}y$ für $x, y \in \mathrm{var}$, so gilt:
  $rr\big(\varphi_1 \wedge \varphi_2\big) := rr(\varphi_1) \cup rr(\varphi_2)$
- $$rr\big(\exists x \psi\big) := \begin{cases} \bot & \text{falls } x \notin rr(\psi) \\ rr(\psi) \setminus \{x\} & \text{sonst} \end{cases}$$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \mathrm{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$
- Für $x, y \in \mathrm{var}$ gilt:
  $$rr\big(\psi \wedge x{=}y\big) := rr\big(x{=}y \wedge \psi\big) := \begin{cases} rr(\psi) \cup \{x, y\} & \text{falls } \{x, y\} \cap rr(\psi) \neq \emptyset \\ rr(\psi) & \text{sonst} \end{cases}$$
- Ist weder φ_1 noch φ_2 von der Form $x{=}y$ für $x, y \in \mathrm{var}$, so gilt:
  $rr\big(\varphi_1 \wedge \varphi_2\big) := rr(\varphi_1) \cup rr(\varphi_2)$
- $$rr\big(\exists x \psi\big) := \begin{cases} \bot & \text{falls } x \notin rr(\psi) \\ rr(\psi) \setminus \{x\} & \text{sonst} \end{cases}$$

Dabei gilt: "Einmal $\bot \rightsquigarrow$ immer $\bot$", d.h. für alle Mengen X gilt:
$\bot = \bot \setminus X = \bot \cap X = X \cap \bot = \bot \cap \bot = \bot \cup X = X \cup \bot = \bot \cup \bot$

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")

- $rr\big(R(v_1, \ldots, v_{ar(R)})\big) := \{v_1, \ldots, v_{ar(R)}\} \cap \text{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \text{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \text{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$
- Für $x, y \in \text{var}$ gilt:
  $rr\big(\psi \wedge x{=}y\big) := rr\big(x{=}y \wedge \psi\big) := \begin{cases} rr(\psi) \cup \{x, y\} & \text{falls } \{x, y\} \cap rr(\psi) \neq \emptyset \\ rr(\psi) & \text{sonst} \end{cases}$
- Ist weder φ_1 noch φ_2 von der Form $x{=}y$ für $x, y \in \text{var}$, so gilt:
  $rr\big(\varphi_1 \wedge \varphi_2\big) := rr(\varphi_1) \cup rr(\varphi_2)$
- $rr\big(\exists x \psi\big) := \begin{cases} \bot & \text{falls } x \notin rr(\psi) \\ rr(\psi) \setminus \{x\} & \text{sonst} \end{cases}$

Dabei gilt: "Einmal $\bot \rightsquigarrow$ immer $\bot$", d.h. für alle Mengen X gilt:
$\bot = \bot \setminus X = \bot \cap X = X \cap \bot = \bot \cap \bot = \bot \cup X = X \cup \bot = \bot \cup \bot$

- $rr(\neg\psi) := \begin{cases} \bot & \text{falls } rr(\psi) = \bot \\ \emptyset & \text{sonst} \end{cases}$

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")

- $rr\big(R(v_1, \ldots, v_{\mathrm{ar}(R)})\big) := \{v_1, \ldots, v_{\mathrm{ar}(R)}\} \cap \mathrm{var}$
- $rr\big(x{=}a\big) := rr\big(a{=}x\big) := \{x\}$, falls $x \in \mathrm{var}$ und $a \in \mathbf{dom}$
- $rr\big(v_1{=}v_2\big) := \emptyset$, falls $\{v_1, v_2\} \subseteq \mathbf{dom}$ oder $\{v_1, v_2\} \subseteq \mathrm{var}$
- $rr\big(\varphi_1 \vee \varphi_2\big) := rr(\varphi_1) \cap rr(\varphi_2)$
- Für $x, y \in \mathrm{var}$ gilt:
  $$rr\big(\psi \wedge x{=}y\big) := rr\big(x{=}y \wedge \psi\big) := \begin{cases} rr(\psi) \cup \{x, y\} & \text{falls } \{x, y\} \cap rr(\psi) \neq \emptyset \\ rr(\psi) & \text{sonst} \end{cases}$$
- Ist weder φ_1 noch φ_2 von der Form $x{=}y$ für $x, y \in \mathrm{var}$, so gilt:
  $rr\big(\varphi_1 \wedge \varphi_2\big) := rr(\varphi_1) \cup rr(\varphi_2)$
- $$rr\big(\exists x \psi\big) := \begin{cases} \bot & \text{falls } x \notin rr(\psi) \\ rr(\psi) \setminus \{x\} & \text{sonst} \end{cases}$$

Dabei gilt: "Einmal $\bot \rightsquigarrow$ immer $\bot$", d.h. für alle Mengen X gilt:
$\bot = \bot \setminus X = \bot \cap X = X \cap \bot = \bot \cap \bot = \bot \cup X = X \cup \bot = \bot \cup \bot$

- $$rr(\neg\psi) := \begin{cases} \bot & \text{falls } rr(\psi) = \bot \\ \emptyset & \text{sonst} \end{cases}$$

Bemerkung: Insbesondere gilt $rr(\varphi) = \bot$ oder $rr(\varphi) \subseteq \mathrm{frei}(\varphi)$

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 $rr(\varphi) \;=\; \{ x_T \} \;=\; \mathsf{frei}(\varphi)$.

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 $rr(\varphi) = \{ x_T \} = \mathsf{frei}(\varphi)$.

(b) Für die SRNf-Formel

$$\psi := \neg \exists y_S \Big(\big(\exists x_R \, \mathit{Filme}(x_T, x_R, y_S) \big) \wedge \neg \big(\exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big)$$

gilt $rr(\psi) = \emptyset \subsetneq \mathsf{frei}(\psi) = \{x_T\}$.

($\psi = \mathrm{SRNF}(\psi')$, wobei ψ' die allerletzte Formel von Beispiel 7.8 ist.

In der Active Domain Semantik ist ψ äquivalent zu φ)

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 $\quad rr(\varphi) \;=\; \{ x_T \} \;=\; \mathsf{frei}(\varphi)$.

(b) Für die SRNf-Formel

$$\psi \;:=\; \neg\exists y_S \Big(\big(\exists x_R\, \mathit{Filme}(x_T, x_R, y_S)\big) \;\wedge\; \neg\big(\exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S)\big) \Big)$$

gilt $\quad rr(\psi) \;=\; \emptyset \;\subsetneq\; \mathsf{frei}(\psi) \;=\; \{x_T\}$.

($\psi = \mathrm{SRNF}(\psi')$, 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

$$\chi \;:=\; R(x) \;\wedge\; \exists y \Big(\neg R(y) \;\wedge\; \neg\exists z \big(\neg R(z) \;\wedge\; \neg z{=}y \big) \Big)$$

gilt $\quad rr(\chi) \;=\; \bot$.

(χ ist die Formel SRNF(”Formel aus Bsp. 7.5”))

Der Sichere Relationenkalkül CALC_{sr}

Definition 7.16

Sei $\mathbf{S}$ ein Datenbankschema.

$\mathrm{CALC}_{sr}[\mathbf{S}]$ bezeichnet die Klasse aller $\mathrm{CALC}[\mathbf{S}]$-Anfragen der Form $\{(e_1, \ldots, e_r) : \varphi\}$, für die gilt: $rr\big(\mathrm{SRNF}(\varphi)\big) = \mathrm{frei}(\varphi)$.

Der Sichere Relationenkalkül CALC_{sr}

Definition 7.16

Sei $\mathbf{S}$ ein Datenbankschema.

$\text{CALC}_{sr}[\mathbf{S}]$ bezeichnet die Klasse aller $\text{CALC}[\mathbf{S}]$-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 $\text{SRNF}(\cdot)$ und $rr(\cdot)$ erhält man direkt einen Algorithmus, der bei Eingabe einer $\text{FO}[\mathbf{S}]$-Formel φ überprüft, ob $rr\big(\text{SRNF}(\varphi)\big) = \text{frei}(\varphi)$.

Somit gibt es also auch einen Algorithmus, der bei Eingabe einer $\text{CALC}[\mathbf{S}]$-Anfrage Q entscheidet, ob Q zu $\text{CALC}_{sr}[\mathbf{S}]$ gehört oder nicht.

Der Sichere Relationenkalkül CALC_{sr}

Definition 7.16

Sei $\mathbf{S}$ ein Datenbankschema.

$\mathrm{CALC}_{sr}[\mathbf{S}]$ bezeichnet die Klasse aller $\mathrm{CALC}[\mathbf{S}]$-Anfragen der Form $\{(e_1, \ldots, e_r) : \varphi\}$, für die gilt: $rr\big(\mathrm{SRNF}(\varphi)\big) = \mathrm{frei}(\varphi)$.

Bemerkung:

Aus den Definitionen von $\mathrm{SRNF}(\cdot)$ und $rr(\cdot)$ erhält man direkt einen Algorithmus, der bei Eingabe einer $\mathrm{FO}[\mathbf{S}]$-Formel φ überprüft, ob $rr\big(\mathrm{SRNF}(\varphi)\big) = \mathrm{frei}(\varphi)$.

Somit gibt es also auch einen Algorithmus, der bei Eingabe einer $\mathrm{CALC}[\mathbf{S}]$-Anfrage Q entscheidet, ob Q zu $\mathrm{CALC}_{sr}[\mathbf{S}]$ 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.

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?

$$\Big\{ (x_{Titel}) \ : \ \exists x_{Regie} \ \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{"George Clooney"}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{"George Clooney"}, \text{"George Clooney"}) \Big) \Big\}$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{ (x_T) \ : \ \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{"Stephen Spielberg"}, y_S) \big) \Big) \Big\}$$

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?

$$\Big\{ (x_{Titel}) \ : \ \exists x_{Regie} \ \Big(\mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg \mathit{Filme}(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von "Stephen Spielberg" mitgespielt haben?

$$\Big\{ (x_T) \ : \ \exists x_R \exists x_S \Big(\mathit{Filme}(x_T, x_R, x_S) \ \wedge \forall y_S \big(\mathit{Filme}(x_T, x_R, y_S) \rightarrow \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

Nicht in $CALC_{sr}$ sind ...

- $\Big\{ (x_T) \ : \ \forall y_S \Big(\exists x_R \, \mathit{Filme}(x_T, x_R, y_S) \ \rightarrow \ \exists y_T \, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$
- $\Big\{ (x) \ : \ \ R(x) \ \wedge \ \exists y \Big(\neg R(y) \ \wedge \ \forall z \, \big(R(z) \ \vee \ z = y \big) \Big) \Big\}$

Ein technisches Lemma

Lemma 7.17

Sei $\mathbf{S}$ *ein Datenbankschema.*

Für jede FO[$\mathbf{S}$]*-Formel* φ *in SRNf mit* $rr(\varphi) \neq \bot$,
für jede Datenbank $\mathbf{I} \in inst(\mathbf{S})$,
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* $x \in rr(\varphi)$: $\beta(x) \in \mathrm{adom}(\varphi, \mathbf{I})$.

und

(2) *Für alle* $\mathbf{d}' \subseteq \mathbf{dom}$ *mit* $\mathrm{adom}(\varphi, \mathbf{I}) \subseteq \mathbf{d}'$ *und* $\beta\big(\mathrm{frei}(\varphi)\big) \subseteq \mathbf{d}'$ *gilt:*

$$\mathbf{I} \models_{\mathbf{d}} \varphi[\beta] \quad \Longleftrightarrow \quad \mathbf{I} \models_{\mathbf{d}'} \varphi[\beta]$$

Beweis: Siehe Tafel.

Relationenkalkül vs. Relationale Algebra

Korollar 7.18

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.*

Beweis: Siehe Tafel.

Relationenkalkül vs. Relationale Algebra

Korollar 7.18

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.*

Beweis: Siehe Tafel.

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.

Relationenkalkül vs. Relationale Algebra

Korollar 7.18

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.*

Beweis: Siehe Tafel.

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.

Abschnitt 7.3:

Statische Analyse und Auswertungskomplexität

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

Statische Analyse für Relational vollständige Sprachen

Satz 7.19

Sei $\mathbf{S}$ *ein Datenbankschema, das mindestens ein Relationssymbol* R *enthält, dessen Stelligkeit* $\geqslant 2$ *ist. Dann sind die folgenden Probleme unentscheidbar:*

Statische Analyse für Relational vollständige Sprachen

Satz 7.19

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 $[\![Q]\!](\mathbf{I}) \neq \emptyset$) ?

Statische Analyse für Relational vollständige Sprachen

Satz 7.19

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

Statische Analyse für Relational vollständige Sprachen

Satz 7.19

Sei **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 S

Eingabe: Anfrage Q der relationalen Algebra über dem Datenbankschema **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 S

Eingabe: Anfragen Q und P der rel. Algebra über dem DB-Schema **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 S

Eingabe: Anfragen Q und P der rel. Algebra über dem DB-Schema **S**

Frage: Ist $Q \sqsubseteq P$ (d.h. gilt für alle $\mathbf{I} \in \mathit{inst}(\mathbf{S})$, dass $[\![Q]\!](\mathbf{I}) \subseteq [\![P]\!](\mathbf{I})$) ?

Statische Analyse für Relational vollständige Sprachen

Satz 7.19

Sei **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 S

Eingabe: Anfrage Q der relationalen Algebra über dem Datenbankschema **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 S

Eingabe: Anfragen Q und P der rel. Algebra über dem DB-Schema **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 S

Eingabe: Anfragen Q und P der rel. Algebra über dem DB-Schema **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*.

Auswertungsproblem: Boolesche Anfragen $\rightsquigarrow$ beliebige Anfragen

Theorem 7.20

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 Verzögerung $\mathcal{O}(k^3 \cdot n)$ 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.

Auswertungsproblem: Boolesche Anfragen $\rightsquigarrow$ beliebige Anfragen

Theorem 7.20

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 Verzögerung $\mathcal{O}(k^3 \cdot n)$ 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).

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.

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: $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}$.

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: $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}$.

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

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

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

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: $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}$.

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

  (1) $B \in \mathrm{PSPACE}$ und
  (2) B ist PSPACE-hart, d.h. für jedes Problem $A \in \mathrm{PSPACE}$ 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: $\big(\, A \text{ PSPACE-hart und } A \leqslant_p B \,\big) \implies B \text{ PSPACE-hart}.$

Auswertungskomplexität des Relationenkalküls

Theorem 7.21 (Chandra, Merlin, 1977)

Das Auswertungsproblem für Boolesche Anfragen des Relationenkalküls ist PSPACE*-vollständig.* *(kombinierte Komplexität)*

Beweis: Siehe Tafel ...

Auswertungskomplexität des Relationenkalküls

Theorem 7.21 (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 ?

Auswertungskomplexität des Relationenkalküls

Theorem 7.21 (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.)

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).

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 $\textsc{Eval}_Q$ das folgende Auswertungsproblem:

  $\textsc{Eval}_Q$

  Eingabe: Datenbank $\mathbf{I}$ (vom zu Q "passenden" Datenbankschema)

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

- Für jede feste Anfrage Q der relationalen Algebra kann $\textsc{Eval}_Q$ in polynomieller Zeit gelöst werden (denn gemäß Proposition 6.4 in Zeit $(k+n)^{\mathcal{O}(k)}$, wobei $k := \|Q\|$)

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 ...

Schaltkreise:

- Wir betrachten Schaltkreise mit $\wedge$, $\vee$, $\neg$-Gattern;
  wobei $\wedge$ und $\vee$-Gatter beliebig viele Eingänge haben dürfen

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 $i_1, \ldots, i_r \in \{1, \ldots, |\,\text{adom}(\mathbf{I})|\}$ 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 adom($\mathbf{I}$) die Konstante c ist.

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 $i_1, \ldots, i_r \in \{1, \ldots, |\,\text{adom}(\mathbf{I})|\}$ 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 $\text{adom}(\mathbf{I})$ die Konstante c ist.

Definition 7.22 ($\text{AC}^0 \mathrel{\widehat{=}}$ "Probleme, die mit Schaltkreisen konst. Tiefe und polyn. Größe lösbar sind")

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 $R \in \mathbf{S}$, $r := \mathrm{ar}(R)$ und alle $i_1, \ldots, i_r \in \{1, \ldots, |\,\mathrm{adom}(\mathbf{I})|\}$ 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 $\mathrm{adom}(\mathbf{I})$ die Konstante c ist.

Definition 7.22 ($\mathrm{AC}^0 \mathrel{\widehat{=}}$ "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 $|\,\mathrm{adom}(\mathbf{I})| = m$ gilt:

$$C_m \text{ akzeptiert Eingabe } \mathbf{I} \iff [\![Q]\!](\mathbf{I}) = \text{"ja"}$$

Theorem 7.23 (Immerman, 1987)

Für jede Boolesche Anfrage Q des Relationenkalküls gehört EVAL_Q *zu* AC^0

Beweis: Hier nur die Beweisidee (Details: *Übung*) anhand der Booleschen Anfrage

$$Q := \Big\{ () : \underbrace{\exists x\, \forall y\, \exists z\, \big(R(x,y) \wedge \neg z{=}c\big)}_{=:\,\varphi} \Big\}$$

Ansatz zur Konstruktion von C_m (auszuwerten über einer Datenbank $\mathbf{I}$ mit $|\,\text{adom}(\mathbf{I})| = m$) :

$$\varphi \quad \rightsquigarrow \quad \bigvee_{i=1}^{m} \bigwedge_{j=1}^{m} \bigvee_{\ell=1}^{m} \Big(R(i,j) \wedge \neg \ell{=}c \Big) \quad \rightsquigarrow \quad \text{Schaltkreis } C_m \text{ (siehe Tafel)}$$
$$\text{der Tiefe} \leqslant \|\varphi\|$$

Abschnitt 7.4:

Grenzen der Ausdrucksstärke

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:

$$\Big\{ (x_S) : \exists x_L \Big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_S) \vee \exists x_Z \big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_Z) \wedge \textit{U-Bahn-Netz}(x_L, x_Z, x_S) \big) \Big) \Big\}$$

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:

$$\Big\{ (x_S) : \exists x_L \Big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_S) \vee \exists x_Z \big(\textit{U-Bahn-Netz}(x_L, \text{“Bockenheimer Warte”}, x_Z) \wedge \textit{U-Bahn-Netz}(x_L, x_Z, x_S) \big) \Big) \Big\}$$

(2) mit max. 2 Zwischenhalten: *analog*

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:

$$\Big\{ (x_S) : \exists x_L \Big(\mathit{U\text{-}Bahn\text{-}Netz}(x_L, \text{“Bockenheimer Warte”}, x_S) \vee \exists x_Z \big(\mathit{U\text{-}Bahn\text{-}Netz}(x_L, \text{“Bockenheimer Warte”}, x_Z) \wedge \mathit{U\text{-}Bahn\text{-}Netz}(x_L, x_Z, x_S) \big) \Big) \Big\}$$

(2) mit max. 2 Zwischenhalten: *analog*

(3) mit beliebig vielen Zwischenhalten ???

Gaifman-Lokalität einer Anfrage

Definition 7.24

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 $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ 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 $\big(a \in [\![Q]\!](\mathbf{I}) \iff b \in [\![Q]\!](\mathbf{I})\big)$.

Gaifman-Lokalität einer Anfrage

Definition 7.24

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 $\mathbf{I} \in \mathit{inst}(\mathbf{S})$ 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 $\big(a \in [\![Q]\!](\mathbf{I}) \iff b \in [\![Q]\!](\mathbf{I})\big)$.

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
  $N_0^{\mathbf{I},Q}(a) := \{a_1, \ldots, a_r\} \cup \mathrm{adom}(Q)$

  $$N_{i+1}^{\mathbf{I},Q}(a) := N_i^{\mathbf{I},Q}(a) \cup \left\{ c \in \mathbf{dom} \;:\; \begin{array}{l}\text{es gibt ein Tupel } t \text{ in } \mathbf{I}, \text{ in dem sowohl } c \text{ als auch} \\ \text{mind. ein Element aus } N_i^{\mathbf{I},Q}(a) \text{ vorkommt}\end{array} \right\}$$

Gaifman-Lokalität einer Anfrage

Definition 7.24

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 $\mathbf{I} \in inst(\mathbf{S})$ 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
  $N_0^{\mathbf{I},Q}(a) := \{a_1, \ldots, a_r\} \cup \text{adom}(Q)$

  $$N_{i+1}^{\mathbf{I},Q}(a) := N_i^{\mathbf{I},Q}(a) \cup \left\{ c \in \mathbf{dom} \;:\; \begin{array}{l}\text{es gibt ein Tupel } t \text{ in } \mathbf{I}, \text{ in dem sowohl } c \text{ als auch} \\ \text{mind. ein Element aus } N_i^{\mathbf{I},Q}(a) \text{ vorkommt}\end{array} \right\}$$

- $\mathcal{N}_d^{\mathbf{I},Q}(a) \cong \mathcal{N}_d^{\mathbf{I},Q}(b)$: $\mathcal{N}_d^{\mathbf{I},Q}(a)$ 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: $t \in \mathbf{I}(R) \iff f(t) \in \mathbf{I}(R)$.

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 $\big(a \in [\![Q]\!](\mathbf{I}) \iff b \in [\![Q]\!](\mathbf{I})\big)$.

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 7.25 (Gaifman-Lokalität (hier ohne Beweis))

Jede Anfrage Q des Relationenkalküls CALC_{adom} *ist Gaifman-lokal.*

Beweis: Siehe Vorlesung „*Logik und Komplexität*“.

Anwendung der Gaifman-Lokalität

Beispiel 7.26

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.

Anwendung der Gaifman-Lokalität

Beispiel 7.26

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

Abschnitt 8.1:
Zusammenfassung

Ausdrucksstärke von Anfragesprachen

- konjunktive Anfragen:
  SPC, SPJR, regelbasierte konjunktive Anfragen, konjunktiver Kalkül, Tableau-Anfragen

Schematische Darstellung: siehe Tafel

Ausdrucksstärke von Anfragesprachen

- konjunktive Anfragen:
  SPC, SPJR, regelbasierte konjunktive Anfragen, konjunktiver Kalkül, Tableau-Anfragen
- azyklische konjunktive Anfragen:
  azyklische regelbasierte konjunktive Anfragen, Semijoin-Anfragen, konjunktives Guarded Fragment

Schematische Darstellung: siehe Tafel

Ausdrucksstärke von Anfragesprachen

- 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

Ausdrucksstärke von Anfragesprachen

- stratifiziertes Datalog$^\neg$

- 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

Ausdrucksstärke von Anfragesprachen

- stratifiziertes Datalog$^\neg$

- 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

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

Methoden zum Nachweis von Grenzen der Ausdrucksstärke einer Anfragesprache

- Monotonie von regelbasierten konjunktiven Anfragen

Methoden zum Nachweis von Grenzen der Ausdrucksstärke einer Anfragesprache

- Monotonie von regelbasierten konjunktiven Anfragen und Datalog-Anfragen

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 $\text{adom}(Q)$-Homomorphismen

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

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$					
Relationale Algebra					
Positive Anfragen					
Konjunktive Anfragen					
Azykl. Konj. Anfragen					
Datalog					

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$					
Relationale Algebra	in AC^0				
Positive Anfragen					
Konjunktive Anfragen					
Azykl. Konj. Anfragen					
Datalog					

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$					
Relationale Algebra	in AC^0				
Positive Anfragen	in AC^0				
Konjunktive Anfragen	in AC^0				
Azykl. Konj. Anfragen	in AC^0				
Datalog					

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$					
Relationale Algebra	in AC^0				
Positive Anfragen	in AC^0				
Konjunktive Anfragen	in AC^0				
Azykl. Konj. Anfragen	in AC^0				
Datalog	P-vollständig				

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig				
Relationale Algebra	in AC^0				
Positive Anfragen	in AC^0				
Konjunktive Anfragen	in AC^0				
Azykl. Konj. Anfragen	in AC^0				
Datalog	P-vollständig				

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig				
Relationale Algebra	in AC^0				
Positive Anfragen	in AC^0				
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0				
Datalog	P-vollständig				

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig				
Relationale Algebra	in AC^0	PSPACE-vollständig			
Positive Anfragen	in AC^0				
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0				
Datalog	P-vollständig				

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig				
Relationale Algebra	in AC^0	PSPACE-vollständig			
Positive Anfragen	in AC^0				
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0	in P (LogCFL-vollst.)			
Datalog	P-vollständig				

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig				
Relationale Algebra	in AC^0	PSPACE-vollständig			
Positive Anfragen	in AC^0	NP-vollständig			
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0	in P (LogCFL-vollst.)			
Datalog	P-vollständig				

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig				
Relationale Algebra	in AC^0	PSPACE-vollständig			
Positive Anfragen	in AC^0	NP-vollständig			
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0	in P (LogCFL-vollst.)			
Datalog	P-vollständig	EXPTIME-vollständig			

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig			
Relationale Algebra	in AC^0	PSPACE-vollständig			
Positive Anfragen	in AC^0	NP-vollständig			
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0	in P (LOGCFL-vollst.)			
Datalog	P-vollständig	EXPTIME-vollständig			

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig			
Relationale Algebra	in AC^0	PSPACE-vollständig	unent-scheidbar	unent-scheidbar	unent-scheidbar
Positive Anfragen	in AC^0	NP-vollständig			
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0	in P (LOGCFL-vollst.)			
Datalog	P-vollständig	EXPTIME-vollständig			

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig			
Konjunktive Anfragen	in AC^0	NP-vollständig			
Azykl. Konj. Anfragen	in AC^0	in P (LOGCFL-vollst.)			
Datalog	P-vollständig	EXPTIME-vollständig			

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig			
Konjunktive Anfragen	in AC^0	NP-vollständig	in P		
Azykl. Konj. Anfragen	in AC^0	in P (LOGCFL-vollst.)			
Datalog	P-vollständig	EXPTIME-vollständig			

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig			
Konjunktive Anfragen	in AC^0	NP-vollständig	in P		
Azykl. Konj. Anfragen	in AC^0	in P (LOGCFL-vollst.)	in P		
Datalog	P-vollständig	EXPTIME-vollständig			

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig	entscheidbar		
Konjunktive Anfragen	in AC^0	NP-vollständig	in P		
Azykl. Konj. Anfragen	in AC^0	in P (LOGCFL-vollst.)	in P		
Datalog	P-vollständig	EXPTIME-vollständig			

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Contain-ment 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		
Konjunktive Anfragen	in AC^0	NP-vollständig	in P		
Azykl. Konj. Anfragen	in AC^0	in P (LOGCFL-vollst.)	in P		
Datalog	P-vollständig	EXPTIME-vollständig	entscheidbar		

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig	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		
Datalog	P-vollständig	EXPTIME-vollständig	entscheidbar		

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig	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		

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig	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		unentscheidbar

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Positive Anfragen	in AC^0	NP-vollständig	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	unentscheidbar	unentscheidbar

Auswertungskomplexität und Statische Analyse

	Daten-komplexität	kombinierte Komplexität	Erfüllbarkeits-problem	Äquivalenz-problem	Query Containment Problem
Stratifiziertes Datalog$^\neg$	P-vollständig	EXPTIME-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
Relationale Algebra	in AC^0	PSPACE-vollständig	unentscheidbar	unentscheidbar	unentscheidbar
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	unentscheidbar	unentscheidbar

Wichtige Stichpunkte

- Homomorphismus-Satz

Wichtige Stichpunkte

- Homomorphismus-Satz
- Algorithmus zur Minimierung konjunktiver Anfragen

Wichtige Stichpunkte

- Homomorphismus-Satz
- Algorithmus zur Minimierung konjunktiver Anfragen
- Sätze von Chandra und Merlin

Wichtige Stichpunkte

- Homomorphismus-Satz
- Algorithmus zur Minimierung konjunktiver Anfragen
- Sätze von Chandra und Merlin
- Satz von Trakhtenbrot (und Folgerungen daraus)

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

Funktionale Abhängigkeiten

- The Chase („Die Verfolgungsjagd“)

Funktionale Abhängigkeiten

- The Chase („Die Verfolgungsjagd“)
- Äquivalenz, Query Containment und Minimierung konjunktiver Anfragen unter Berücksichtigung funktionaler Abhängigkeiten

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

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

Abschnitt 8.2:

Ausblick auf weitere Themen

Einige weiterführende Themen

- Semistrukturierte Daten und XML

Einige weiterführende Themen

- Semistrukturierte Daten und XML
- Datenbanken mit unvollständiger Information

Einige weiterführende Themen

- Semistrukturierte Daten und XML
- Datenbanken mit unvollständiger Information
- Datenaustausch und Datenintegration

Einige weiterführende Themen

- Semistrukturierte Daten und XML
- Datenbanken mit unvollständiger Information
- Datenaustausch und Datenintegration
- Datenströme

Einige weiterführende Themen

- Semistrukturierte Daten und XML
- Datenbanken mit unvollständiger Information
- Datenaustausch und Datenintegration
- Datenströme
- Constraint Datenbanken

Einige weiterführende Themen

- Semistrukturierte Daten und XML
- Datenbanken mit unvollständiger Information
- Datenaustausch und Datenintegration
- Datenströme
- Constraint Datenbanken
- Probabilistische Datenbanken

– ENDE –