

Anfragesprachen mit Rekursion — Datalog

- 3.1 Syntax und Semantik
- 3.2 Statische Analyse
- 3.3 Einschränkung und Erweiterungen: nr-Datalog und Datalog mit Negation

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"
```

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:

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

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

- (3) mit **beliebig vielen** Zwischenhalten ???

(Am Ende von Kapitel 5 werden wir sehen: **Nicht ausdrückbar in Relationaler Algebra**)

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:

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

Anfragesprachen mit Rekursion — Datalog

3.1 Syntax und Semantik

3.2 Statische Analyse

3.3 Einschränkung und Erweiterungen: nr-Datalog und Datalog mit Negation

Notation

- Wie üblich bezeichnen wir mit $adom(P)$ bzw. $adom(Q)$ die Menge der Konstanten, die in einem Datalog-Programm P bzw. einer Datalog-Anfrage Q vorkommen.
Für eine Datenbank I ist $adom(P, I) := adom(P) \cup adom(I)$ und $adom(Q, I) := adom(Q) \cup adom(I)$.
- $R_0(u_0)$ heißt **Kopf** der Regel $R_0(u_0) \leftarrow R_1(u_1), \dots, R_\ell(u_\ell)$;
 $R_1(u_1), \dots, R_\ell(u_\ell)$ heißt **Rumpf** der Regel.
- $edb(P)$ bezeichnet die Menge der Relations-Namen, die ausschließlich im **Rumpf** von Regeln in P vorkommen (die sog. "extensionalen Prädikate von P ").
- $idb(P)$ bezeichnet die Menge der Relations-Namen, 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 .

Datalog: Syntax

Definition 3.1

- (a) Eine **Datalog-Regel** ist ein Ausdruck der Form

$$R_0(u_0) \leftarrow R_1(u_1), \dots, R_\ell(u_\ell)$$

wobei $\ell \geq 0$, $R_0, R_1, \dots, R_\ell \in \mathbf{relname}$ und u_0, u_1, \dots, u_ℓ freie Tupel der Stelligkeiten $arity(R_0), arity(R_1), \dots, arity(R_\ell)$ sind, so dass jede Variable, die in u_0 vorkommt, auch in mindestens einem der Tupel u_1, \dots, u_ℓ 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 Relations-Namen R , der in P vorkommt.

Beispiel

$$P := \left\{ \begin{array}{l} Erreichbar(L, S, Z) \leftarrow U\text{-Bahn-Netz}(L, S, Z), \\ Erreichbar(L, S, Z) \leftarrow Erreichbar(L, S, Z'), U\text{-Bahn-Netz}(L, Z', Z), \\ Ans(Z) \leftarrow Erreichbar(L, \text{"Bockenheimer Warte"}, Z) \end{array} \right\}$$

ist ein Datalog-Programm mit

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

$Q_1 := (P, Ans)$ ist eine Datalog-Anfrage;

$Q_2 := (P, Erreichbar)$ ist noch eine Datalog-Anfrage;

$Q_3 := (P, U\text{-Bahn-Netz})$ 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”

Glücklicherweise kann man zeigen, dass alle drei Ansätze zum gleichen Resultat führen.

Wir betrachten im Folgenden hauptsächlich die Fixpunkt-Semantik, die folgendermaßen definiert ist:

Monotonie von T_P

Bemerkung: Eine alternative (und äquivalente) Definition von T_P :

$$T_P(\mathbf{I})(R) := \{ \text{unmittelbare } R\text{-Konsequenzen von } P \text{ über } \mathbf{I} \},$$

wobei ein Tupel t eine **unmittelbare R -Konsequenz von P über \mathbf{I}** ist, falls

- ▶ $R \in edb(P)$ und $t \in \mathbf{I}(R)$ oder
- ▶ $R \in idb(P)$ und es eine Regel der Form $R(u) \leftarrow R_1(u_1), \dots, R_\ell(u_\ell)$ in P und eine Belegung β gibt, so dass $\beta(u) = t$ und $\beta(u_i) \in \mathbf{I}(R_i)$, für alle $i \in \{1, \dots, \ell\}$.

Lemma 3.3

Für jedes Datalog-Programm P gilt:

Der Operator T_P ist **monoton**, d.h. für alle $\mathbf{I}, \mathbf{J} \in inst(sch(P))$ mit $\mathbf{I} \subseteq \mathbf{J}$ (d.h. $\mathbf{I}(R) \subseteq \mathbf{J}(R)$ f.a. $R \in sch(P)$) gilt: $T_P(\mathbf{I}) \subseteq T_P(\mathbf{J})$.

Beweis: leicht (Übung).

Der “immediate consequence”-Operator T_P

Definition 3.2

Sei P ein Datalog-Programm.

- (a) Für jedes $R \in idb(P)$ seien $Q_{R,1}, \dots, Q_{R,k_R}$ diejenigen Regeln aus P , in deren Kopf das Relationssymbol R steht, und seien $Q'_{R,1}, \dots, Q'_{R,k_R}$ die Regeln, die aus $Q_{R,1}, \dots, Q_{R,k_R}$ entstehen, indem im **Kopf** jeweils R durch das Relationssymbol Ans' ersetzt wird (mit $Ans' \notin sch(P)$ und $arity(Ans') = arity(R)$).
- (b) Der “immediate consequence”-Operator $T_P : inst(sch(P)) \rightarrow inst(sch(P))$ ist folgendermaßen definiert:
Für jedes $\mathbf{I} \in inst(sch(P))$ und jedes $R \in sch(P)$ ist

$$T_P(\mathbf{I})(R) := \begin{cases} \mathbf{I}(R) & \text{falls } R \in edb(P) \\ \llbracket Q'_{R,1} \rrbracket(\mathbf{I}) \cup \dots \cup \llbracket Q'_{R,k_R} \rrbracket(\mathbf{I}) & \text{falls } R \in idb(P) \end{cases}$$

Beispiel: Ist P das Datalog-Programm aus dem Beispiel der Erreichbarkeits-Anfrage, und besteht $\mathbf{I}(Erreichbar)$ aus allen Tupeln, die “mit max. i Zwischenhalten ohne Umsteigen zu erreichen sind”, so besteht $T_P(\mathbf{I})(Erreichbar)$ aus allen Tupeln, die “mit max. $i + 1$ Zwischenhalten ohne Umsteigen zu erreichen sind”.

Fixpunkte

Bemerkung 3.4

- (a) Aus der Monotonie von T_P folgt für jedes $\mathbf{I} \in inst(sch(P))$ mit $\mathbf{I}(R) = \emptyset$ f.a. $R \in idb(P)$, dass:

$$\mathbf{I} \subseteq T_P(\mathbf{I}) \subseteq T_P(T_P(\mathbf{I})) \subseteq \dots \subseteq T_P^i(\mathbf{I}) \subseteq T_P^{i+1}(\mathbf{I}) \subseteq \dots$$

wobei $T_P^0(\mathbf{I}) := \mathbf{I}$ und $T_P^{i+1}(\mathbf{I}) := T_P(T_P^i(\mathbf{I}))$.

- (b) Man sieht leicht, dass $adom(T_P(\mathbf{I})) \subseteq adom(P, \mathbf{I})$ und $adom(T_P^i(\mathbf{I})) \subseteq adom(P, \mathbf{I})$, f.a. $i \in \mathbb{N}$.
- (c) Da $adom(P, \mathbf{I})$ nur endlich viele Elemente besitzt, muss es in der Inklusionskette aus (a) ein $i_0 \in \mathbb{N}$ mit $T_P^{i_0}(\mathbf{I}) = T_P^{i_0+1}(\mathbf{I})$ geben.
Aus der Monotonie von T_P folgt dann: $T_P^{i_0}(\mathbf{I}) = T_P^j(\mathbf{I})$ f.a. $j \geq i_0$.

Notation

- ▶ Ein $\mathbf{J} \in \text{inst}(\text{sch}(P))$ heißt **Fixpunkt von T_P** , falls $T_P(\mathbf{J}) = \mathbf{J}$.
- ▶ $\text{Ab-Stufe}(\mathbf{I}) := \min\{i_0 : T_P^{i_0}(\mathbf{I}) = T_P^{i_0+1}(\mathbf{I})\}$ heißt **“Abschluss-Stufe”** von T_P auf \mathbf{I} .
Die einzelnen Instanzen $T_P^0(\mathbf{I}), T_P^1(\mathbf{I}), T_P^2(\mathbf{I})$ etc. werden **“Stufen des Fixpunktprozesses”** genannt.
 $\text{Ab-Stufe}(P, \mathbf{I})$ gibt also diejenige Stufe an, ab der der Fixpunkt erreicht ist:
$$\mathbf{I} = T_P^0(\mathbf{I}) \subsetneq T_P^1(\mathbf{I}) \subsetneq \dots \subsetneq T_P^{\text{Ab-Stufe}(P, \mathbf{I})}(\mathbf{I}) = T_P^{\text{Ab-Stufe}(P, \mathbf{I})+1}(\mathbf{I}) = T_P^j(\mathbf{I})$$

f.a. $j \geq \text{Ab-Stufe}(P, \mathbf{I})$
- ▶ $T_P^\omega(\mathbf{I}) := T_P^{\text{Ab-Stufe}(\mathbf{I})}(\mathbf{I})$. Klar: $T_P^\omega(\mathbf{I})$ ist ein Fixpunkt von T_P .

Naive Berechnung von $\llbracket P \rrbracket(\mathbf{I})$

Bemerkung:

Ein einfacher Algorithmus, der bei Eingabe von P und \mathbf{I} das Resultat $\llbracket P \rrbracket(\mathbf{I})$ berechnet, kann folgendermaßen vorgehen:

- (1) $\mathbf{J} := \hat{\mathbf{I}}$
- (2) Berechne $\mathbf{J}' := T_P(\mathbf{J})$
- (3) Falls $\mathbf{J}' \neq \mathbf{J}$, so ($\mathbf{J} := \mathbf{J}'$, GOTO (2))
- (4) Gib \mathbf{J} aus

Datenkomplexität dieses Algorithmus: Polynomialzeit.

Theorem 3.6 (Immerman und Vardi — hier ohne Beweis)

- Die Datenkomplexität des Auswertungsproblems für Datalog-Anfragen ist **PTIME-vollständig**.
- Die kombinierte Komplexität des Auswertungsproblems für Datalog-Anfragen ist **EXPTIME-vollständig**.

Fixpunkt-Semantik von Datalog

Definition 3.5

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

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

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

$$\llbracket P \rrbracket(\mathbf{I}) := T_P^\omega(\hat{\mathbf{I}}) \in \text{inst}(\text{sch}(P))$$

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

$$\llbracket Q \rrbracket(\mathbf{I}) := (\llbracket P \rrbracket(\mathbf{I}))(R).$$

Fixpunkt-Semantik vs. Modellbasierte Semantik

Notation:

Für zwei Datenbanken $\mathbf{J}, \mathbf{J}' \in \text{inst}(\mathbf{R})$ bezeichnet $\mathbf{J} \cap \mathbf{J}'$ die Datenbank mit $(\mathbf{J} \cap \mathbf{J}')(R) := \mathbf{J}(R) \cap \mathbf{J}'(R)$, f.a. $R \in \mathbf{R}$.

Satz 3.7 (Knaster und Tarski)

Sei P ein Datalog-Programm und $\mathbf{I} \in \text{inst}(\text{edb}(P))$. Dann gilt:

$$\llbracket P \rrbracket(\mathbf{I}) = \bigcap \{ \mathbf{J} \in \text{inst}(\text{sch}(P)) : T_P(\mathbf{J}) = \mathbf{J} \text{ und } \hat{\mathbf{I}} \subseteq \mathbf{J} \}$$

D.h.: $\llbracket P \rrbracket(\mathbf{I})$ ist die kleinste Erweiterung von $\hat{\mathbf{I}}$, die ein Fixpunkt von T_P ist.

Beweis: Siehe Tafel.

Ausdrucksstärke von Datalog

Bemerkung 3.8

Die Ausdrucksstärken von Datalog-Anfragen und von Anfragen der 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).

Denn aus Lemma 3.3 folgt leicht, dass **jede Datalog-Anfrage monoton** ist.

Beweisbäume

Definition 3.9

Sei P ein Datalog-Programm und $I \in \text{inst}(\text{edb}(P))$. Ein **Beweisbaum für ein Faktum $R(t)$ bzgl. I 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 $\text{Fakten}(I)$
- (4) Für jeden inneren Knoten v mit Kindern v_1, \dots, v_ℓ gibt es eine Regel $R_0(u_0) \leftarrow R_1(u_1), \dots, R_\ell(u_\ell)$ in P und eine Belegung $\beta : \text{Var}(P) \rightarrow \text{dom}$ 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 $i \in \{1, \dots, \ell\}$).

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

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

(siehe Tafel)

Beweisbasierte Semantik von Datalog

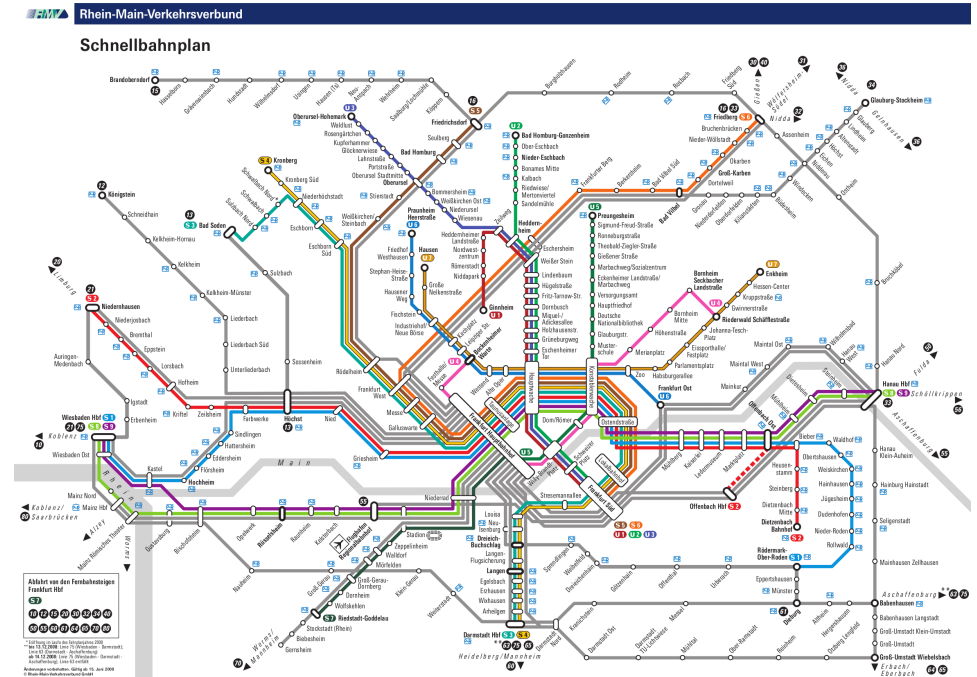
Sichtweise:

- ▶ Ein **Faktum** ist ein Ausdruck der Form $R(t)$ mit $R \in \text{relnam}$ und $t \in \text{dom}^{\text{arity}(R)}$.

- ▶ Eine Datenbank $I \in \text{inst}(\mathbf{R})$ wird mit der folgenden **Menge von Fakten** identifiziert:

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

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



Fixpunkt-Semantik vs. Beweisbasierte Semantik

Satz 3.10

Für jedes Datalog-Programm P , alle $\mathbf{I} \in \text{inst}(\text{edb}(P))$ und alle Fakten $R(t)$ mit $R \in \text{sch}(P)$ und $t \in \text{dom}^{\text{arity}(R)}$ gilt:

$$t \in (\llbracket P \rrbracket(\mathbf{I}))(R) \iff \text{es gibt einen Beweisbaum für } R(t) \text{ bzgl. } \mathbf{I} \text{ und } P.$$

Beweis: Übung.

Erfüllbarkeit

Theorem 3.11

Das

ERFÜLLBARKEITSPROBLEM FÜR DATALOG-ANFRAGEN

Eingabe: Datalog-Anfrage $Q = (P, R)$

Frage: Gibt es ein $\mathbf{I} \in \text{inst}(\text{edb}(P))$ so dass $\llbracket Q \rrbracket(\mathbf{I}) \neq \emptyset$?

ist entscheidbar.

Beweisidee:

Verallgemeinerung des Beweises der Erfüllbarkeit regelbasierter konjunktiver Anfragen (Satz 2.6).

Details: Übung.

Anfragesprachen mit Rekursion — Datalog

3.1 Syntax und Semantik

3.2 Statische Analyse

3.3 Einschränkung und Erweiterungen: nr-Datalog und Datalog mit Negation

Query Containment — Zwei Varianten

(a) Herkömmliches Query Containment:

QUERY CONTAINMENT PROBLEM FÜR DATALOG-ANFRAGEN

Eingabe: Zwei Datalog-Anfragen (P_1, R) und (P_2, R) mit $\text{edb}(P_1) = \text{edb}(P_2)$ und $R \in \text{idb}(P_1) \cap \text{idb}(P_2)$

Frage: Gilt $\llbracket (P_1, R) \rrbracket(\mathbf{I}) \subseteq \llbracket (P_2, R) \rrbracket(\mathbf{I})$ für alle $\mathbf{I} \in \text{inst}(\text{edb}(P_1))$?

(b) Uniformes Containment:

UNIFORMES CONTAINMENT PROBLEM FÜR DATALOG-PROGRAMME

Eingabe: Zwei Datalog-Programme P_1 und P_2 mit $\text{edb}(P_1) = \text{edb}(P_2)$ und $\text{idb}(P_1) = \text{idb}(P_2)$

Frage: Gilt $\llbracket P_1 \rrbracket(\mathbf{I}) \subseteq \llbracket P_2 \rrbracket(\mathbf{I})$ für alle $\mathbf{I} \in \text{inst}(\text{sch}(P_1))$?

Theorem 3.12

(Beweis von (a) siehe Tafel; (b) hier ohne Beweis)

(a) Das **Query Containment Problem** für Datalog-Anfragen ist **unentscheidbar**.

(b) Das **uniforme Containment Problem** für Datalog-Programme ist **entscheidbar**.

Beschränktheit (Boundedness)

- Wir haben gesehen: $\llbracket P \rrbracket(\mathbf{I})$ kann berechnet werden, indem man nach und nach

$$\hat{\mathbf{I}}, T_P(\hat{\mathbf{I}}), T_P(T_P(\hat{\mathbf{I}})), \dots, T_P^{Ab-Stufe(P, \mathbf{I})}(\hat{\mathbf{I}}), T_P^{Ab-Stufe(P, \mathbf{I})+1}(\hat{\mathbf{I}})$$

berechnet.

- Anzahl der Iterationen, die dafür nötig sind: $Ab-Stufe(P, \mathbf{I}) + 1$
- Frage: **Wie groß kann $Ab-Stufe(P, \mathbf{I})$ werden?**
- Klar: – in jeder Iteration kommt mindestens ein Faktum dazu;
 - $adom(\llbracket P \rrbracket(\mathbf{I})) \subseteq adom(P, \mathbf{I})$
- Daher: $Ab-Stufe(P, \mathbf{I}) \leq \sum_{R \in idb(P)} |adom(P, \mathbf{I})|^{arity(R)}$
- Besonders “schön” sind solche Datalog-Programme P , bei denen $Ab-Stufe(P, \mathbf{I})$ gar nicht von \mathbf{I} abhängt. Solche Programme heißen **beschränkt** (engl.: **bounded**).
- Präzise: Ein Datalog-Programm P heißt **beschränkt**, falls eine Zahl $d \in \mathbb{N}$ gibt, so dass für alle $\mathbf{I} \in inst(edb(P))$ gilt: $Ab-Stufe(P, \mathbf{I}) \leq d$.
Den kleinsten solchen Wert d nennen **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).

Beschränktheit (Boundedness)

Theorem 3.13

(hier ohne Beweis)

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 $Ab-Stufe(P, \mathbf{I}) \leq d$ für alle $\mathbf{I} \in inst(edb(P))$?

ist unentscheidbar.

Beispiel

Das Datalog-Programm

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

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 2 und äquivalent zum nicht-rekursiven Programm

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

bzw. zur SPCU-Anfrage $Mag \cup (Person \times \pi_3(\sigma_{1=1}(Trendsetter \times Mag)))$

Anfragesprachen mit Rekursion — Datalog

3.1 Syntax und Semantik

3.2 Statische Analyse

3.3 Einschränkung und Erweiterungen: nr-Datalog und Datalog mit Negation

Nicht-Rekursives Datalog — Beispiel

Beispiel 3.14

(a) Das Datalog-Programm

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

ist nicht-rekursiv.

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

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

“Nicht-rekursiv” vs. “beschränkt” (“bounded”)

Proposition 3.16

- (a) Jedes nr-Datalog-Programm ist beschränkt.
 (b) Jedes beschränkte Datalog-Programm ist äquivalent zu einem nr-Datalog-Programm.

Beweis: Einfache Übung.

Nicht-Rekursives Datalog — Präzise

Definition 3.15

Sei P ein Datalog-Programm.(a) Der **Abhängigkeitsgraph** G_P von P ist der gerichtete Graph mit Knotenmenge $V_P := \text{sch}(P)$ und Kantenmenge

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

(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*

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: $\llbracket (Q_1 \cup Q_2) \rrbracket(\mathbf{I}) := \llbracket Q_1 \rrbracket(\mathbf{I}) \cup \llbracket Q_2 \rrbracket(\mathbf{I})$.
- Der **positive existentielle Kalkül** $\text{PE-CALC}_{\text{adom}}$ ist die Klasse aller Anfragen Q der Form $\{\langle e_1, \dots, e_r \rangle : \varphi\}$, wobei φ eine Formel der Logik erster Stufe ist, in der keins der Symbole $\neg, \forall, \rightarrow, \leftrightarrow$ vorkommt. Semantik: $\llbracket Q \rrbracket(\mathbf{I}) := \{\beta(\langle e_1, \dots, e_r \rangle) : \beta : \text{var} \rightarrow \text{adom}(Q, \mathbf{I}), \text{ so dass } \mathbf{I} \models \varphi[\beta]\}$

Satz 3.17

Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen beschreiben:

- (a) nr-Datalog-Anfragen
 (b) SPCU-Algebra
 (c) positiver existentieller Kalkül $\text{PE-CALC}_{\text{adom}}$

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 3.17

(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 $Q = \{\langle e_1, \dots, e_r \rangle : \varphi\}$ bringe die Formel φ zunächst in disjunktive Normalform, d.h. in eine Formel der Form $\varphi_1 \vee \dots \vee \varphi_\ell$, wobei φ_i eine CQ^- -Formel ist.

Dann gilt für jede Datenbank \mathbf{I} , dass $\llbracket Q \rrbracket(\mathbf{I}) = \llbracket Q_1 \rrbracket(\mathbf{I}) \cup \dots \cup \llbracket Q_\ell \rrbracket(\mathbf{I})$, wobei

$$\llbracket Q_i \rrbracket(\mathbf{I}) = \{ \beta(\langle e_1, \dots, e_r \rangle) : \beta : \mathbf{var} \rightarrow \mathit{adom}(Q, \mathbf{I}), \text{ so dass } \mathbf{I} \models \varphi_i \}.$$

Wir können nun ähnlich wie in Kapitel 2 beim Beweis der Äquivalenz von Konjunktivem Kalkül und SPC-Algebra vorgehen, um eine SPCU-Anfrage Q'_i zu konstruieren, so dass für alle Datenbanken \mathbf{I} gilt: $\llbracket Q'_i \rrbracket(\mathbf{I}) = \llbracket Q_i \rrbracket(\mathbf{I})$.

Details: Übung. \square

Frage: Was soll die Semantik von Datalog⁻ sein?

Beispiel 3.19

Anfrage:

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

Als Datalog⁻-Anfrage:

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

Hier: Semantik intuitiv klar.

Datalog mit Negation

Ziel: Auch Negationszeichen “ \neg ” in Datalog-Regeln zulassen.

Definition 3.18

(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⁻-Regel** ist ein Ausdruck der Form

$$R_0(u_0) \leftarrow L_1(u_1), \dots, L_\ell(u_\ell)$$

wobei $\ell \geq 0$, $R_0 \in \mathbf{relname}$, u_0 ein freies Tupel der Stelligkeit $\mathit{arity}(R_0)$ und $L_1(u_1), \dots, L_\ell(u_\ell)$ Literale, so dass jede Variable, die in u_0 vorkommt, auch in mindestens einem **positiven** Literal $L_i(u_i)$ vorkommt.

(c) Ein **Datalog⁻-Programm** P ist eine **endliche Menge von Datalog⁻-Regeln**.

(d) Eine **Datalog⁻-Anfrage** (P, R) besteht aus einem Datalog⁻-Programm P und einem Relations-Namen R , der in P vorkommt.

Probleme mit der Semantik von Datalog⁻

Beispiel 3.20

(a) $R(x) \leftarrow A(x), \neg R(x)$

“Ausgewertet” über einer DB $\mathbf{I} \in \mathit{inst}(\{A\})$ gilt:

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

Somit: Die Folge $(T_P^i(\hat{\mathbf{I}}))_{i \geq 0}$ hat keinen Fixpunkt.

Außerdem: $T_P(\cdot)$ hat überhaupt keinen Fixpunkt.

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

Hier hat $T_P(\cdot)$ zwei verschiedene minimale Fixpunkte (**minimal bzgl. \subseteq**):

- ▶ FP_1 mit $\mathit{FP}_1(R) = \emptyset$ und $\mathit{FP}_1(S) = \mathit{adom}(\mathbf{I})$
- ▶ FP_2 mit $\mathit{FP}_2(R) = \mathit{adom}(\mathbf{I})$ und $\mathit{FP}_2(S) = \emptyset$

Inflationäre Fixpunkte

Um eine Semantik für Datalog^- festzulegen, könnte man einfach per Definition erzwingen, dass die einzelnen Stufen des Fixpunktprozesses ineinander enthalten sind:

An Stelle von $T_P(\cdot)$ betrachte den Operator $\Gamma_P(\cdot)$ mit

$$\Gamma_P(\mathbf{J}) := \mathbf{J} \cup T_P(\mathbf{J})$$

Dann gilt:

$$\hat{\mathbf{I}} \subseteq \Gamma_P(\hat{\mathbf{I}}) \subseteq \Gamma_P(\Gamma_P(\hat{\mathbf{I}})) \subseteq \dots \subseteq \Gamma_P^i(\hat{\mathbf{I}}) \subseteq \Gamma_P^{i+1}(\hat{\mathbf{I}}) \subseteq \dots$$

Da $\text{adom}(P, \mathbf{I})$ endlich ist, wird irgendwann ein (eindeutig definierter) Fixpunkt von $\Gamma_P(\cdot)$ erreicht. Dieser heißt **inflationärer Fixpunkt von P auf \mathbf{I}** .

Frage: Was soll die Semantik von Datalog^- sein?

Probleme:

- ▶ Inflationäre Fixpunkt-Semantik: unnatürlich (siehe Beispiel 3.21)
- ▶ Fixpunkt-Semantik via $T_P^\omega(\hat{\mathbf{I}})$: für manche Datalog^- -Programme undefiniert (siehe Beispiel 3.20)
- ▶ Semantik via “kleinster Fixpunkt” von $T_P(\cdot)$: ist für manche Datalog^- -Programme nicht eindeutig (siehe Beispiel 3.20)
- ▶ Beweisbasierte Semantik für Datalog^- : ???

Aber für Programme wie in Beispiel 3.21 ist die Semantik “intuitiv klar”.

↪ Betrachte im Folgenden nur Datalog^- -Programme von eingeschränkter Form:

- ▶ Semipositives Datalog^-
- ▶ nr-Datalog $^-$
- ▶ Stratifiziertes Datalog^-

Problem mit der inflationären Fixpunkt-Semantik von Datalog^-

Diese Semantik ist für viele Programme **unnatürlich**.

Beispiel 3.21

$E(L, S, Z)$	\leftarrow	$U\text{-Bahn-Netz}(L, S, Z)$
$E(L, S, Y)$	\leftarrow	$E(L, S, Y), U\text{-Bahn-Netz}(L, Y, Z)$
$\text{Erreichbar_BW}(Z)$	\leftarrow	$E(L, \text{“Bockenheimer Warte”}, Z)$
$\text{Station}(S)$	\leftarrow	$U\text{-Bahn-Netz}(L, S, Z)$
$\text{Station}(Z)$	\leftarrow	$U\text{-Bahn-Netz}(L, S, Z)$
$\text{Ans}(Z)$	\leftarrow	$\text{Station}(Z), \neg\text{Erreichbar_BW}(Z)$

“Intuitive Semantik”:

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

Inflationäre Fixpunkt-Semantik: Alle Stationen.

Semipositives Datalog^-

Negation ist **nur bei edb-Prädikaten** erlaubt.

Man kann leicht zeigen, dass für jedes semipositive Datalog^- -Programm P und alle $\mathbf{I} \in \text{inst}(\text{edb}(P))$ gilt:

- ▶ $T_P(\cdot)$ hat einen eindeutig bestimmten kleinsten Fixpunkt \mathbf{J} mit $\mathbf{J}|_{\text{edb}(P)} = \mathbf{I}$.
- ▶ Dieser wird von der Sequenz

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

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

Definition der Semantik von semipositiven Datalog^- -Programmen P :
Für alle $\mathbf{I} \in \text{inst}(\text{edb}(P))$ setze

$$\llbracket P \rrbracket(\mathbf{I}) := T_P^\omega(\hat{\mathbf{I}})$$

Stratifiziertes Datalog⁻ — Beispiel
$$\begin{aligned}
 E(L, S, Z) &\leftarrow U\text{-Bahn-Netz}(L, S, Z) \\
 E(L, S, Z) &\leftarrow E(L, S, Y), U\text{-Bahn-Netz}(L, Y, Z) \\
 Erreichbar_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\
 Station(S) &\leftarrow U\text{-Bahn-Netz}(L, S, Z) \\
 Station(Z) &\leftarrow U\text{-Bahn-Netz}(L, S, Z) \\
 Ans(Z) &\leftarrow Station(Z), \neg Erreichbar_BW(Z)
 \end{aligned}$$

- ▶ 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⁻.

Stratifiziertes Datalog⁻ — Beispiele $P^1 :=$

$$\begin{aligned}
 E(L, S, Z) &\leftarrow BVG(L, S, Z) \\
 E(L, S, Z) &\leftarrow E(L, S, Y), U\text{-Bahn-Netz}(L, Y, Z) \\
 Erreichbar_BW(Z) &\leftarrow E(L, \text{“Bockenheimer Warte”}, Z) \\
 Station(S) &\leftarrow U\text{-Bahn-Netz}(L, S, Z) \\
 Station(Z) &\leftarrow U\text{-Bahn-Netz}(L, S, Z)
 \end{aligned}$$
 $P^2 :=$

$$Ans(Z) \leftarrow Station(Z), \neg Erreichbar_BW(Z)$$

ist eine Stratifizierung des Datalog⁻-Programms aus Beispiel 3.21.

Das Datalog⁻-Programm $R(x) \leftarrow A(x), \neg R(x)$ ist **nicht stratifizierbar**.

Das Datalog⁻-Programm

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

auch nicht.

Stratifiziertes Datalog⁻ — Präzise

Definition 3.22

Sei P ein Datalog⁻-Programm.

Eine **Stratifizierung von P** ist eine Folge P^1, \dots, P^m von Datalog⁻-Programmen, so dass $m \geq 1$ ist und es eine Abbildung $\sigma : idb(P) \rightarrow \{1, \dots, m\}$ gibt, so dass gilt:

- (1) P^1, \dots, P^m ist eine **Partition von P** (d.h. $P = P^1 \dot{\cup} \dots \dot{\cup} P^m$),
- (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 $\sigma(S) \leq \sigma(R)$, 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, \dots, P^m (“Stratum”: lat. für “Schicht”; Plural: Strata)
- ▶ σ heißt **Stratifzierungs-Abbildung**
- ▶ Ein Datalog⁻-Programm P heißt **stratifizierbar**, falls es eine Stratifizierung von P gibt. **Stratifiziertes Datalog⁻** bezeichnet die Menge aller stratifizierbaren Datalog⁻-Programme.

Test auf Stratifizierbarkeit

Abhängigkeitsgraph für Datalog⁻-Programme

Definition 3.23

Sei P ein Datalog⁻-Programm. Der **Abhängigkeitsgraph $G_P = (V_P, E_P^+, E_P^-)$** ist der gerichtete Graph mit

- ▶ Knotenmenge $V_P := sch(P)$

- ▶ Kantenmengen

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

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

Beispiel: Siehe Tafel: Abhängigkeitsgraph für

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

Test auf Stratifizierbarkeit

Proposition 3.24

Für jedes Datalog⁻-Programm P gilt:

P ist stratifizierbar \iff Im Abhängigkeitsgraph G_P gibt es keinen Kreis, in dem eine Kante aus E_P^- vorkommt.

Beweis:

“ \implies ”: 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!**

“ \impliedby ”: Idee: Nutze eine **topologische Sortierung** der **starken Zusammenhangskomponenten** von $(E_P^+ \cup E_P^-)$, um die einzelnen Schichten zu definieren.

Details: *Übung.*

Spezialfall: nr-Datalog⁻

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

Satz 3.25

Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen beschreiben:

- nr-Datalog⁻-Anfragen
- Relationale Algebra
- Relationenkalkül.

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: Der Beweis wird in den Kapiteln 4 und 5 (Relationale Algebra und Relationenkalkül) gegeben.

Stratifiziertes Datalog⁻ — Semantik

- Sei P^1, \dots, P^m eine Stratifizierung eines Datalog⁻-Programms P .
- Betrachte jede Schicht P^i als ein semipositives Datalog⁻-Programm mit $edb(P^i) \subseteq edb(P) \cup idb(P^1) \cup \dots \cup idb(P^{i-1})$

- Sei $\mathbf{I} \in edb(P)$.

Die **Semantik** $\llbracket P \rrbracket(\mathbf{I})$ von P auf \mathbf{I} ist folgendermaßen definiert: $\llbracket P \rrbracket(\mathbf{I}) := \mathbf{I}^m$, wobei

$$\begin{aligned} \mathbf{I}^1 &:= \llbracket P^1 \rrbracket(\mathbf{I}) \\ \mathbf{I}^2 &:= \llbracket P^2 \rrbracket(\mathbf{I}^1) \\ &\vdots \\ \mathbf{I}^m &:= \llbracket P^m \rrbracket(\mathbf{I}^{m-1}) \end{aligned}$$

Man kann leicht zeigen, dass folgendes gilt:

- Obige Definition hängt nicht von der konkreten Wahl der Stratifizierung von P ab. D.h. für je zwei verschiedene Stratifizierungen P^1, \dots, P^m und Q^1, \dots, Q^n von P gilt: $\llbracket P^m \rrbracket(\mathbf{I}^{m-1}) = \llbracket Q^n \rrbracket(\mathbf{I}^{n-1})$.
- $\llbracket P \rrbracket(\mathbf{I})$ ist der (eindeutig definierte) kleinste Fixpunkt \mathbf{J} von $T_P(\cdot)$ mit $\mathbf{J}|_{edb(P)} = \mathbf{I}$.