

Deduktive Datenbanksysteme

Problem: Transitiver Abschluss ist in PL1 nicht formulierbar (mit zustandsabhängiger Formulierung möglich)

Diskussion:

Typ 5: $q_1(\dots), \dots, q_n(\dots) : -.$

Typ 6: $q_1(\dots), \dots, q_n(\dots) : -p_1(\dots), \dots, p_m(\dots).$

\Rightarrow Übungsaufgabe

Nur Typ 1 und Typ 4: $q(\dots) : -p_1(\dots), \dots, p_n(\dots), n \geq 0$ (ist die Hornklauselform und wird bei definiten Datenbanken genutzt)

Definite Datenbanken

$$\begin{aligned} q(\dots) &: -. \text{(Fakt)} \\ q(\dots) &: -p_1(\dots), \dots, p_n(\dots). \text{(deduktive Regel, } p_{1-n} \text{ Teilziele)} \end{aligned} \tag{1}$$

- Mit IBen (Integritätsbedingungen) (+ Typ2, Typ3) ($: -p_1(\dots), \dots, p_n(\dots)$)
- Typ5 + Typ6 \Rightarrow **Disjunktive Datenbank**
- Definite Datenbank + negative Atome im Rumpf von Hornklauseln erlaubt
 \Rightarrow Volles Datalog

Formulierung von Anfragen

Klauseln vom Typ: $: -p_1(\dots), \dots, p_n(\dots)$, geschrieben $? - p_1(\dots), \dots, p_n(\dots)$

Beispiele:

- $? - ag(X, m).$
 - Bedeutung: Welche Kurse bietet 'm' an?
 - DRC: (x) / ANGEBOT(X, m)
- $? - ag(a3, m).$
 - Bedeutung: Bietet 'm' den Kurs 'a3' an?
 - DRC: () / ANGEBOT(a3, m)
- $? - ag(X, m), bl(X, s, j).$

- Bedeutung: Gib alle von 'm' angebotene Kurse, die 's' als Wiederholer belegt hat
- DRC: $(x) / \text{ANGEBOT}(x, 'm') \wedge \text{BELEGUNG}(x, 's', 'y')$
- Wie ist $(x) / \text{ANGEBOT}(x, 'm') \wedge (\exists y) \text{BELEGUNG}(x, 's', y)$ formulierbar?
 - Bedeutung: Gib die Dozenten der von s als Wiederholer belegte Kurse
 - Formulierung: $? - Ksm(X).$
 $Ksm(X) : -ag(X, m), bl(X, s, y)$
 - Bequemer: $? - ag(X, Y*), bl(X, s, y)..$, * Kennzeichnet die Ausgabevariable

In Anfragesprachen werden Vergleichsausdrücke benötigt. Dazu sind in Datalog spezielle vordefinierte Prädikate vorhanden. Für jeden Vergleichsoperator wird die Existenz eines solchen Prädikates angenommen.

Zunächst: Beschränkte Variablen in Regeln. Sei eine Regel r gegeben:

- Jede Variable, die als Argument in einem gewöhnlichen Prädikat im Rumpf von r vorkommt ist beschränkt.
- Jede Variable, die in einem Teilziel $X = c$ oder $c = X$ von r vorkommt, ist beschränkt.
- Eine Variable X ist beschränkt, wenn sie in einem Teilziel $X = Y$ oder $Y = X$ von r vorkommt mit Y ist schon als beschränkt bekannt.

Definition: sicher

Eine Regel heißt sicher, wenn alle in ihr vorkommenden Variablen beschränkt sind.

Beispiele:

- $Kls(X, Y) : -bl(Z, s, j), ag(Z, Y), X = Z$. **sicher**
- $vsj(X, Y) : -bl(Y, s, j)$. **nicht sicher** (X ist nicht beschränkt)
- $vs(X, Y) : -vs(X, Z), kp(Z, Y)$. **sicher, wenn vs terminiert**
- $kla(Z, Y) : -bl(Z, V, j), ag(Z, Y), V \neq s$. **sicher**

Bemerkung: Falls keine Build-in Prädikate erlaubt sind (/vorkommen):
 Eine Regel ist sicher genau dann wenn jede Variable im Kopf der Regel auch im Rumpf der Regel vorkommt.

Definition: Datalog Programm

Ein Datalog-Programm P (ohne IBen(Integritätsbedingungen)) ist eine endliche Menge von Horn-Klauseln mit Jedes $d \in P$ ist entweder

- ein Fakt $q(\dots)$. ohne Variable
- eine sichere Regel $q(\dots) : -p_1(\dots), \dots, p_n(\dots)$. mit $q \in iPraedikat$

Ein $d \in P$ heißt auch **Datalog-Klausel** Alle Fakten zu extensionalen Prädikaten sind als in DB-Relationen gespeichert zu denken.

Beispiel Datenbankzustand:

$$\begin{aligned}
 & ag(a1, m). \\
 & kp(c2, a0). \\
 & \dots \\
 & rb(a1, r1, t1) \quad (Kurs a1 im Raum r1 zu t1) \\
 & rb(a3, r2, t4)
 \end{aligned} \tag{2}$$

Angebot:

Kursnummer	Dozent
a1	m
...	...

Kursplan:

Kursnummer	Voraussetzung
c2	a0
...	...

Raumbelegung:

Kursnummer	Raum	Zeit
a1	r1	t1
...

Belegung:

Kursnummer	Teilnehmer	Wiederholer
a1	s	j
...

$$\begin{aligned}
 & vs(X, Y) : -Kp(X, Y). \\
 & vs(X, Y) : -vs(X, Z), Kp(Z, Y). \\
 & stdpl(W, X, Y, Z) : -bl(X, W, V), rb(X, Y, Z). \\
 & ueberschneidungen(X, Y) : -Kp(Z, X), Kp(Z, Y), rb(X, V, T), rb(Y, W, T), X \neq Y.
 \end{aligned} \tag{3}$$

Deklarative Semantik

Extensionale Prädikate eines Programms (ext. Rel, ext. DB): EDB
 Intentionale Prädikate eines Programms (int. Rel, int. DB): IDB

Bedeutung Bedeutung eines Datalog-Programms P: Menge derjenigen Grundatome zu den intentionalen Prädikaten von P, die logisch aus P gefolgert werden können.
 (Jedes Modell von P ist auch ein Modell von $f \in F$). Mit Zielklausel $g(\dots)$, $g \in \text{Praed}$: Aus P logisch folgerbare Grundatome zu g, die von $g(\dots)$ subsummiert
 (überdeckt) werden.

$$\frac{g(a,X)}{g(a,b)}$$

$$\frac{g(a,X)}{g(a,c)}$$

In P werden Werte aus Wertebereichen verwendet, ebenso in Darstellung der extensionalen Prädikate als DB-Relation. Daher können wir $Kost_A$ und Dom indefnitizieren. Mithilfe der Herbrand Interpretation kann die Semantik festgelegt werden (ist möglich).

Herbrand-Interpretation /-Modelle Gewöhnliche Interpretation:

$$\begin{aligned}
 Konst_A &= \{a, b\}, Dom = \{\circ, \square\} \\
 k(a) &= \circ \\
 k(b) &= \square \\
 ext(p(\cdot, \cdot)) &= \{(\circ, \square), (\square, \square)\}
 \end{aligned} \tag{4}$$

eine mögliche Herbrand-Interpretation (passt dazu)

$$\begin{aligned}
 Konst_A &= Dom = \{a, b\} \\
 k(a) &= a \\
 k(b) &= b \\
 ext(p(., .)) &= \{(a, b), (b, b)\}
 \end{aligned} \tag{5}$$

Entsprechende Herbrand-Interpretation. Betrachte alle Paare zu $p(., .)$, teste gemäß gegebener (gewöhnlicher) Interpretation in $ext(p(., .))$.

$$\begin{aligned}
 Konst_A &= \{a, b\}, Dom = \{\circ, \square\} \\
 k(a) &= \square \\
 k(b) &= \square \\
 ext(p(., .)) &= \{(\circ, \square), (\square, \square)\}
 \end{aligned} \tag{6}$$

(a,a) wird zu $(\square, \square) \in ext(p(., .))$

Herbrand-Interpretation

$$\begin{aligned}
 Konst_A &= Dom = \{a, b\} \\
 k(a) &= a \\
 k(b) &= b \\
 ext(p(., .)) &= \{(a, a), (a, b), (b, a), (b, b)\}
 \end{aligned} \tag{7}$$

Bei beiden Interpretationen sind die gleichen Formeln gültig bei Beschränkung auf quantorenfreie Formeln ohne Variablen (und ohne Funktionen).

Beispiel Erste Interpretation: $p(a, b) \wedge p(b, a) \Rightarrow (\square, \square) \in ext(p(., .)) \wedge \dots$ bzw. $(a, b) \in ext(p(., .)) \wedge (b, a) \in ext(p(., .))$

Menge von Konstanten und Prädikatensymbole ist endlich, daher ist die Anzahl der möglichen Herbrand-Interpretationen endlich.

Satz von Gödel / Skolem (Herbrand, 1930)

Eine Klauselmenge P hat ein Modell genau dann wenn P hat ein Herbrand-Modell. Daraus folgt, dass ein Verfahren analog zu Wahrheitstabellen in der Aussagenlogik möglich ist.

Beispiel $F = \{p(a) \Rightarrow q(b), p(a) \wedge q(b)\}, q(b)?$

p	q	$p(a) \Rightarrow q(b)$ erfüllt?	$p(a) \wedge q(b)$ erfüllt?	$p(a) \Rightarrow q(b) \text{ und } p(a) \wedge q(b)$ erfüllt?
{}	{}	✓	-	-
{}	{a}	✓	-	-
{}	{b}	✓	✓	✓
{}	{a, b}	✓	✓	✓
{a}	{}	-	✓	-
...
{b}	{b}	✓	✓	✓

Jedes Modell von F ist auch ein Modell von q(b), d.h. q(b) kann aus F logisch gefolgert werden. Gilt bei Klauselmengen, aber **Vorsicht bei allgemeinen Formeln**.

Beispiel: $\{p(a), (\exists X)(\neg p(X))\}$ Formelmenge, keine Klauselmenge

Modell (vgl. Übung):

$$\begin{aligned} Dom &= \{0, 1\} \\ k(a) &= 0 \\ ext(p(.)) &= \{(0)\} \end{aligned} \tag{8}$$

Aber: Es gibt kein durch `ext` bestimmtes Herbrand-Modell:

1. $ext(p(.)) = \{(a)\}, Konst = Dom = \{a\}$
2. $ext(p(.)) = \{\}$

Herbrand-Modell muss genügend viele Elemente enthalten, damit der Satz von Gödel / Skolem gelten kann. **Skolemisierung** bedeutet, dass man alle Existenzquantoren durch Funktionen ersetzt:

$$(\forall X_1, \dots, X_n)(\exists y)(F) \rightsquigarrow (\forall X_1, \dots, X_n)(F[f(X_1, \dots, X_n)/y]) \tag{9}$$

Bemerkung: Skolemisierung

Jede Formel der PL1 Logik kann man in einer erfüllbarkeitsäquivalente Formel in Skolemform umformen:

1. Pränexnormalform

2. Umformungen à la $(\forall X_1, \dots, X_n)(\exists y)(F) \rightsquigarrow (\forall X_1, \dots, X_n)(F[f(X_1, \dots, X_n)/y])$
mit jeweils einem neuen Funktionssymbol

Dies ist eine Art “Materialisierung” der durch den Existenzquantor gebundenen Variablen.

Beispiel (von oben) $\{p(a), \neg p(y)^1\}$ erfüllbar $\iff \{p(a), (\exists X)(\neg p(X))\}$ erfüllbar

Vorsicht: Semantische Äquivalenz von Formeln und ihren Skolem-Normalformen im Allgemeinen nicht gegeben.

Skolem-NF: $(\exists X)(p(X)) : p(a)$ Interpretation I mit

$$\begin{aligned} Dom &= \{1, 2\} \\ k(a) &= 1 \\ ext(p(.)) &= \{(2)\} \end{aligned} \tag{10}$$

$\rightsquigarrow \models_I (\exists X)(p(X))$ Belegung von X mit allen Elementen aus Dom, d.h. auch mit 2.
Aber $\not\models_I p(a)$, da $(1) \notin ext(p(.))$.

Im Kontext von Datalog

- Herbrand-Universum: Konst
- Herbrand-Basis (HB): Menge aller Grundatome
($\rightsquigarrow EDB^2, IDB^3$ (Fakten in der Datenbank und solche die Ableitbar sind) $\subset HB$)
- Herbrand-Interpretation: (Konst, id, ext), d.h. jedes Konstantensymbol wird als es selbst interpretiert. (Verglichen mit relationaler Interpretation, dort k ist Bijektion)
- Jede Herbrand-Interpretation ist eindeutig bestimmt durch ext (Extension, Ausprägung), da Konst und id unveränderlich sind
- Jedes Prädikat ist eindeutig bestimmt durch die Angabe der Grundatome, für die es “wahr” liefert.
- extentional: genaue Antwort in Tupel
- intentional: Beispielsweise Formeln als Antworten

¹neue Variable

²gegeben durch Datenbankzustand

³muss ausgerechnet / gefolgt werden

Definition: Herbrand-Interpretation

Einfache Definition: Eine Herbrand-Interpretation ist eine Teilmenge der Herbrand-Basis.

Beispiel Kursverwaltung

$$\begin{aligned} \text{Aus DB-Rel. KURSPLAN } &\rightsquigarrow kp \in ePräd \\ vs(X, Y) : -kp(X, Y) &\rightsquigarrow vs \in iPräd \end{aligned} \tag{11}$$

Gesucht: Durch Programm P bestimmte Grundatome (Fakten).

Definition: Grundatom

Ein Grundatom f ist eine logische Folgerung einer Menge D von Datalog Klauseln (z.B. $D \models f$) \diamondsuit_{Def} . Jedes Herbrand Modell von D ist auch ein Modell von f .

Da f ein Grundatom ist gilt $D \models f \implies f$ ist in jedem Herbrand-Modell von D enthalten. Das heißt $f \in \bigcap\{I | I \text{ Herbrand-Modell von } D\}$.

Sei $f \in \bigcap\{I | I \text{ Herbrand-Modell von } D\}$, dann ist f ein Grundatom und jedes Modell von D auch in Modell von f .

Definition: Mege aller Konsequenzen

$$cons(D) =_{def} \{f \in HB_D | D \models f\}$$

Satz 1.1

$$cons(D) = \bigcap\{I | I \text{ Herbrand-Modell von } D\}$$

Aufgrund der Eigenschaften unserer Regel ist der Schnitt / $cons(D)$ ein Modell, dies gilt es zu beweisen:

Da $cons(D) \subseteq HB_D$, ist $cons(D)$ eine Herbrand-Interpretation.

Satz 1.2

$cons(D)$ ist ein Herbrand-Modell von D .

Beweis

z.Z. Jedes $d \in D$ ist gültig in dieser Interpretation, also $\text{cons}(D)$ ($\models_{\text{cons}(D)} d$).

Beweis Falls d ein Grundatom ist gehört d zu jedem Herbrandmodell von D , daraus folgt $d \in \text{cons}(D)$.

Sei d eine Regel $q(\dots) : -p_1(\dots), \dots, p_m(\dots)$. Sei ϱ eine Belegung für die Variable von d . Annahme: $(\forall 1 \leq i \leq m)(||p_i(\cdot)||^\varrho \in \text{cons}(D))$, sonst d gültig unter $\text{cons}(D)$. Für jedes Herbrand-Modell I von D gilt, dann $||p_i(\cdot)||^\varrho \in I, i = 1, \dots, m$ und damit $||q(\cdot)||^\varrho \in \text{cons}(D)$, da d eine Horn-Klausel ist. (Der Schluss ist z.B. für $d = q_1(\dots), q_2(\dots) : -p_1(\dots), \dots, p_n(\dots)$ nicht möglich.)

Damit $D \models_{\text{cons}(D)} ||q(\dots)||^\varrho \in \text{cons}(D)$. Wir erhalten, dass $\text{cons}(D)$ ein Herbrand-Modell von D ist. Da d beliebig gewählt wird folgt der Satz.

$\text{cons}(D)$ ist offensichtlich eindeutig bestimmt und das kleinste Herbrand-Modell von D . Damit: Semantik eines Datalog-Programms P ist gegeben durch das kleinste Herbrand-Modell von P oder (äquivalent) durch $\text{cons}(P) = \{f \in HB_P \mid O \models f\}$.

Beispiel $r \in ePräd, p, q \in iPräd, Konst = \{1, 2, 3\}$.

Fakten:

$$\begin{aligned} & r(1). \\ & p(X) : -q(X). \\ & q(X) : -r(X). \end{aligned} \tag{12}$$

Herbrand-Modelle:

$$\begin{aligned} & \{r(1), p(1), q(1)\} \\ & \{r(1), p(1), q(1), q(2), p(2)\} \\ & \{r(1), p(1), q(1), q(2), p(2), p(3)\} \end{aligned} \tag{13}$$

Fixpunkt-Semantik

Deklarative Semantik liefert keinen brauchbaren Algorithmus. Die Fixpunkt-Semantik führt direkt zu einem Algorithmus für die Schrittweise Berechnung des kleinsten Herbrand-Modells.

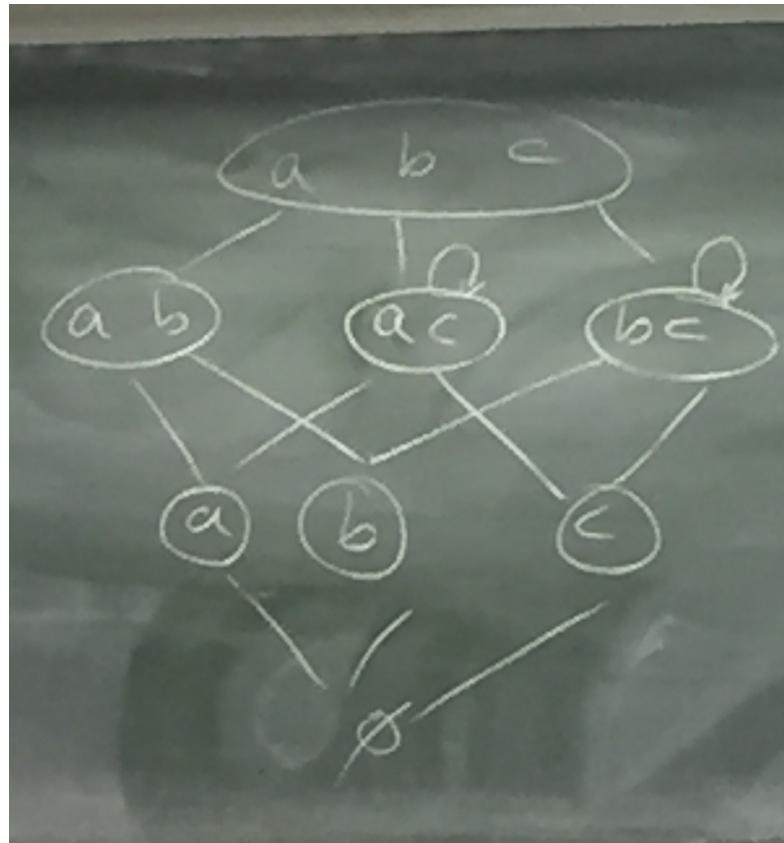


Figure 1:

$$\begin{aligned} c &\in ac \\ \tau(c) &\subseteq \tau(ac) \\ \tau(c) &= c! \\ \tau(c) &= ac \end{aligned} \tag{14}$$

$$\begin{aligned} c &\in bc \\ \tau(c) &\subseteq \tau(bc) \\ \tau(c) &= c! \\ \tau(c) &= bc \end{aligned} \tag{15}$$

Dies ist ein Widerspruch

P Datalog-Programm τ_P liefert Fakten von P vereinigt mit Fakten, die in einem Schritt aus der Argumentenmenge von τ_P mit den Regeln von P abgeleitet werden können.

Ableitung in einem Schritt

Definition: Substitution

Eine Substitution ist eine endliche Menge der Form

$$\{X_1/t_1, \dots, X_n/t_n\}, X_1, \dots, X_n \text{ unterschiedliche Variablen, } t_1, \dots, t_n \text{ Terme, } X_i \neq t_i \quad (16)$$

Falls alle t_i Konstanten sind, ist dies eine **Grundsubstitution**.

Sei θ eine Substitution, t ein Term (Variable oder Konstante), so gilt

$$t\theta =_{def} \begin{cases} t_i, & \text{falls } t/t_i \in \theta \\ t, & \text{sonst} \end{cases} \quad (17)$$

Sei L ein Literal, $L\theta$ bezeichnet dasjenige Literal, das aus L entsteht, indem alle Variablen X_i mit L_i für die $X_i/t_i \in \theta$ gilt, simultan durch t_i ersetzt werden. Analog $d\theta$ für eine Datalog-Klausel d.

Beispiel

$$L = p(X, Y, a), \theta = \{X/a, Y/X\} L\theta = p(a, X, a) \quad (18)$$

Sei D eine Menge von Datalog Klauseln. In einem Schritt aus D ableitbare Menge von Fakten:

$$fakt_1(D) = \{f \in HB_D \mid (\exists Regel L_0 : -L_1, \dots, L_n)(\exists Sicht \theta)(\{L_1\theta, \dots, L_n\theta\} \subseteq D \wedge f = L_0\theta)\} \quad (19)$$

(Annahme: built-in Prädikate geeignet berücksichtigt \Rightarrow Algorithmus FAKT_1)

Definiere $\tau_P|2^{HB} \Rightarrow 2^{HB}$. $\tau_P(I) =_{def} P_F \cup fakt_1(P_R \cup I).P_f$ = Menge der Fakten von P. P_P Menge von Regeln von P.

Satz 1.3

Für jedes Datalog-Programm P ist τ_P eine monotone Transformation auf $(2^{HB}, \subseteq)$.

Beweis

Seien $I_1, I_2 \in HB_P$ mit $I_1 \subseteq I_2$. z.Z.: $\tau_P(I_1) \subseteq \tau_P(I_2)$.

Sei $f \in \tau_P(I_1)$. Falls $f \in P_F$, dann gilt auf $f \in \tau_p(I_2)$. $f \in fakt_1(P_R \cup I_1)$ da $I_1 \subseteq I_2$, gilt $P_R \cup I_1 \subseteq P_R \cup I_2$ und somit $f \in fakt_1(P_R \cup I_2)$ wg. Monotonie von Fakt_1 auf $(2^{HB}, \subseteq)$.

Beispiel

$$\begin{aligned} P &= p(1) \\ &= p(2). \\ &= r(1). \end{aligned} \tag{20}$$

$$\begin{aligned} q(X) &: -s(X), r(X). \\ s(X) &: -p(X). \end{aligned}$$

$$\rightsquigarrow^{T_p(\emptyset)}$$

$$\begin{aligned} p_1 \\ p_2 \\ r_1 \end{aligned} \tag{21}$$

$$\rightsquigarrow^{T_p(\cdots)}$$

$$\begin{aligned} p_1 \\ p_2 \\ r_1 \\ s(1)(fakt_1) \\ s(2)(fakt_1) \end{aligned} \tag{22}$$

$$\rightsquigarrow^{T_p(\cdots)}$$

$$\begin{array}{ll}
 p_1 & \\
 p_2 & \\
 r_1 & \\
 s(1) & \\
 s(2) & \\
 q(1)(\text{fakt_1}) &
 \end{array} \tag{23}$$

Satz 1.4

Für jedes Datalog-Programm P gilt $\text{cons}(P) = \text{lf}_p(\tau_p)$ (lf = least fixpunkt).

Beweis

:

1) $\text{cons}(P)$ ist ein Fixpunkt von τ_P .

$\tau_p(\text{cons}(P)) = P_F \cup \text{fakt_1}(P_R \cup \text{cons}(P))$. $\text{cons}(P)$ ist kleinstes Herbrand-Modell von P, d.h. alle Regeln sind gültig unter $\text{cons}(P) \Rightarrow \text{fakt_1}(P_R \cup \text{cons}(P)) = \text{cons}(P) \setminus P_F$. (und kein Fakt kann aus $\text{cons}(P)$ entfernt werden, ohne dass die Modellereigenschaft verloren geht.) $\Rightarrow \tau_P(\text{cons}(P)) = P_F \cup \text{cons}(P) \setminus P_F = \text{cons}(P)$

2) $\text{cons}(P)$ ist minimaler Fixpunkt von τ_P .

Annahme: $(\exists I \not\subseteq \text{cons}(P))(\tau_P(I) = I)$.

$\text{cons}(P)$ ist minimales Herbrand-Modell $\Rightarrow I$ ist kein Herbrand-Modell. Da $P_F \subseteq I$ wg. Annahme $\tau_P(I) = I$ folgt mindestens eine Regel ist nicht erfüllt, d.h. $(\exists h_0, \dots, h_n \in P_R)(\exists \text{Substitution } \theta)(\{h_1\theta, \dots, L_n(\theta)\} \subseteq I \wedge L_1\theta \in I)$. Aber $L_0 \in \text{fakt}_1(P_R \cup I)$ nach Definitio von fakt_1. Da auch $L_0\theta \in P_F$ wegen $P_F \subseteq I$, folgt $\tau_P(I) \neq I$. Noch Fixpunkttheorem (Kuaster / Tarski) ist minimaler Fixpunkt von τ_P kleinstes Fixpunkt von τ_P .

Vorgehensweise bei Fixpunktberechnung

bottom-up

Bezeichnung: Forward-chaining

Da " \Leftarrow " die natürliche Richtung für die Anwendung von Regeln ist. Da Regeln sicher sind ist $L_i\theta$ stets ein Fakt θ_i . Betrachte Eignung der Methode zur Anfrageauswertung, z.B. ?- vs(c4, y). "Alle Kurse, die Voraussetzung von Kurs c4 sind"

Ineffizient Berechnung des lfp und anschließende Suche zu vs(c4, y) passende Fakten \Rightarrow Vermeide möglichst Berechnung nicht relevanter Fakten (später).

Starte mit Zielklausel, Suche ein Regel die zu einem Atom der Klausel passt. Ersetze das Atom durch angepassten Rumpf der gewählten Regel, neue Anfrage, iterieren.

Bezeichnung: Backward-chaining

Beispiel

$$\begin{aligned} P &= p(1) \\ &= p(2). \\ &= r(1). \end{aligned} \tag{24}$$

$$\begin{aligned} q(X) &: -s(X), r(X). \\ s(X) &: -p(X). \end{aligned}$$

Prozedurale Semantik

Beweistheoretische Sicht, übertragen auf Datalog Programm P aufgefasst als Theorie 1. Stufe

Semantik: $\{f \in HB_P \mid P \vdash f\}$

Wie können Ableitungen (Beweise) von Fakten systematisch durchgeführt werden?

Regeln:

$$\begin{aligned} (1) &vs(X, Y) : -Kp(X, Y). \\ (2) &vs(X, Y) : -vs(X, Z), kp(Z, Y). \\ P &= \{Kp(c4, a3), \dots, kp(c2, a0), (1), (2)\} \end{aligned} \tag{25}$$

Zeige $P \vdash vs(c4, a0)$.

$$\begin{aligned} (1) &= (\forall X)(\forall Y)(kp(X, Y) \Rightarrow vs(X, Y)) \\ (2) &= (\forall x)(\forall Y)(\forall Z)(vs(X, Z) \wedge kp(Z, Y) \Rightarrow vs(X, Y)) \end{aligned} \tag{26}$$

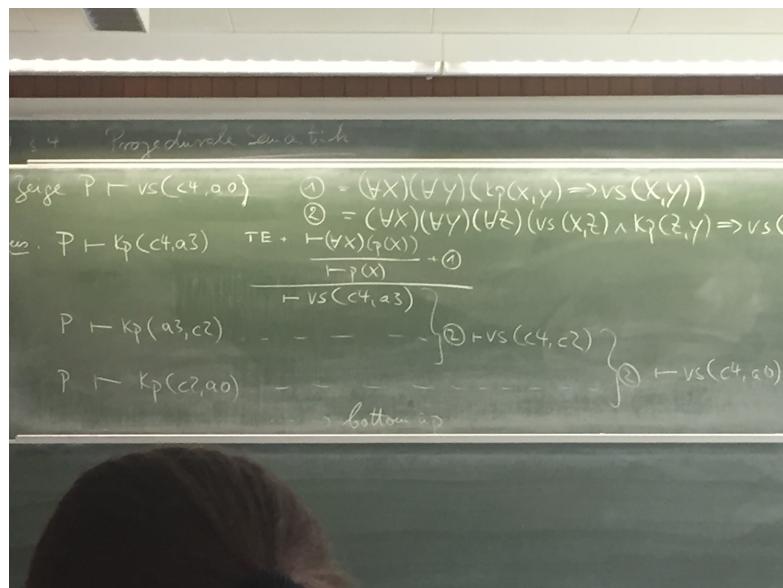


Figure 2:

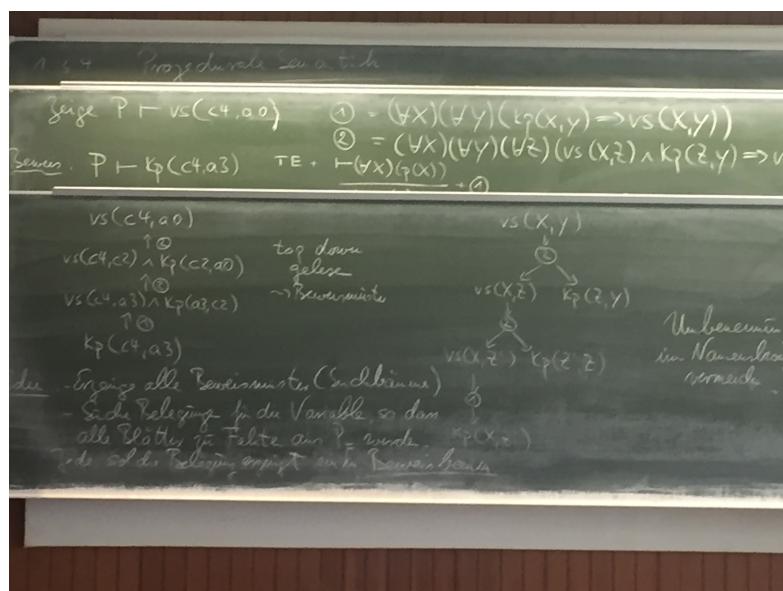


Figure 3:

Beweis

Idee

- Erzeuge alle Beweismuster (Suchbäume)

- Suche Belegungen für die Variable, so dass alle Blätter zu Fakten aus P werden
- jede solche Belegung erzeugt einen Beweisbaum.

Zunächst Suche nach “passenden” Köpfen von Regeln erfordert Definition.

Definition: Unifizierbar

Seien L_1 und L_2 heißen **unifizierbar**, wenn $(\exists \text{ Substitution } \Theta)(L_1\Theta = L_2\Theta)$. Θ heißt dann **Unifikator**.

Beispiel $L_1 = vs(X, Z)$

$L_2 = cs(X, Y)$

$\Theta = \{Z/Y\}$ und $\Theta = \{Y/Z\}$ sind Unifikatoren von dem Paar L_1, L_2 , aber auch $\Theta = \{X/a, Y/a, Z/a\}$. Die ersten beiden sind spezifischer als das letzte

$L_1 = q(X, Y, c) L_2 = q(W, b, Z)$ Unifikation z.B.:

$$\begin{aligned}\Theta_1 &= \{X/WY/bZ/c\} \\ \Theta_2 &= \{X/T, W/T, y/b, Z/c\} \\ \Theta_3 &= \{X/a, W/a, Y/b, Z/c\}\end{aligned}\tag{27}$$

Nicht unifizierbar: $L_1 = q(X, c, U) L_2 = q(W, G, Z)$ oder $L_1 = q(X, a, X) L_2 = q(b, Y, Y)$

Definition: Komposition

Sei $\Theta = \{X_1/t_1, \dots, X_n/t_n\}, \varsigma = \{Y_1/n_1, \dots, Y_m/n_m\}$ Substitutionen.
Die Komposition $\Theta\varsigma$ von Θ und ς erhält man aus

$$X_1/t_1\varsigma, \dots, X_m/t_m\varsigma, Y_1/n_q, \dots, Y_m/n_m\tag{28}$$

Durch Streichen von Elementen der Form Z/Z sowie Y_i/n_i mit $Y_i = X_j$ für ein $j \in \{1, \dots, n\}$

Beispiel $\theta = \{X/a, Y/W\}\varsigma = \{X/bmY/V, W/Z\}$
 $\Theta\varsigma = \{X/a, YZ, W/Z\}$

Definition: allgemeinere Substitution

Seien Θ, ς Substitutionen, Θ heißt allgemeiner als ς $(\exists \text{Substitution } \delta)(\Theta \delta = \varsigma)$. Seine L_1, L_2 Literale. Ein allgemeinster Unifikator (mgu) von $L_1 \text{ in } L_2$ ist ein Unifikator von $L_1 \text{ und } L_2$, der allgemeiner als alle anderen Unifikatoren ist.

Beispiel Θ_2 ist allgemeiner als Θ_3 ; betrachte $\delta = \{T/a\}$, es gilt $\Theta_2 \delta = \Theta_3$. Θ_1 ist allgemeiner als Θ_2 , $\delta = \{W/T\}$. mgu ist i.A. nicht eindeutig bestimmt. $L_1 = p(X, X) L_2 = p(V, W)$ mgu:

$$\begin{aligned} & \{X/W, V/W\} \\ & \{X/V, W/V\} \\ & \{V/X, W/X\} \end{aligned} \tag{29}$$

Beispiel $L_1 = q(X, Y, c)^4 L_2 = q(W, b, Z)^5$

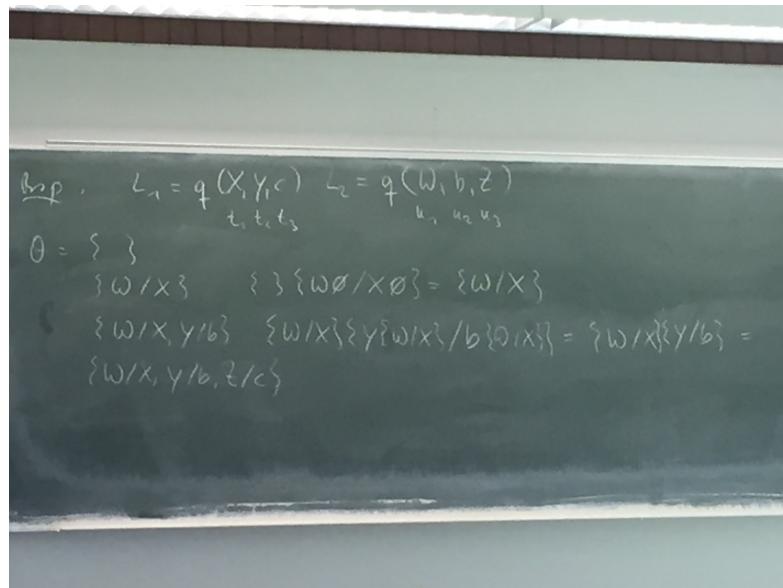


Figure 4:

⁴t1, t2, t3⁵k1 k2 k3

Suchbaum

(Beweismuster zu einem Programm P)

- Wurzel ist mit einem “Ziel” $g = p(t_1, \dots, t_n), p \in iPr \cdot \cdot \cdot ad$, bekannt.
- Knoten entlang eines Pfades von der Wurzel aus sind abwechselnd mit Atome und Regeln benannt. (Ziel-, Regelknoten)
- Alle Blattknoten sind mit Atomen benannt
- Sei k ein mit einem Atom $a(s_1, \dots, s_l)$ benannten Knoten (keine Blattknoten), dann ist der unmittelbare Nachfolger von k mit einer Regel von P benannt, deren Kopf mit $q(sq, \dots, s_l)$ unifizierbar ist
- sei k' ein mit einer Regel $r = L_0 : -L_1, \dots, L_m$ benannte Knoten der unmittelbare Vorgänger von k' sei mit $q(s_1, \dots, s_l)$ benannt. Dann hat k' m unmittelbare Nachfolger, die mit $I_1mgu(q(s_1, \dots, s_l), I_0), \dots, I_mgu(q(s_1, \dots, s_l), I_0)$ benannt sind. Dabei sind I_0, \dots, I_m Atome, die aus L_1, \dots, L_m durch eventuelle Umbenennung von Variablen hevorgehen (die Variablen seien stets so umbenannt, dass sie im Baum eindeutig sind).

Beispiel

Suchbaum → Beweisbaum

Gegeben Sei der Suchbaum S und eine Grundsubstitution Θ mit folgenden Eigenschaften:

- Θ ordnet jeder Variable in S ein Konstantensymbol aus der Menge der in P vorkommenden Konstantensymbole zu.
- Für jedes Blatt von S gilt, dass Θ , angewandt auf die Benennung des Blattes, einen Fakt aus P_F liefert (build-in-Prädikate seien entsprechend berücksichtigt)

Beweisbaum B entsteht aus S durch Anwendung von Θ auf alle Benennungen von Zielknoten. B repräsentiert einen Beweis für $g\Theta$, g benennung der Wurzel von S .

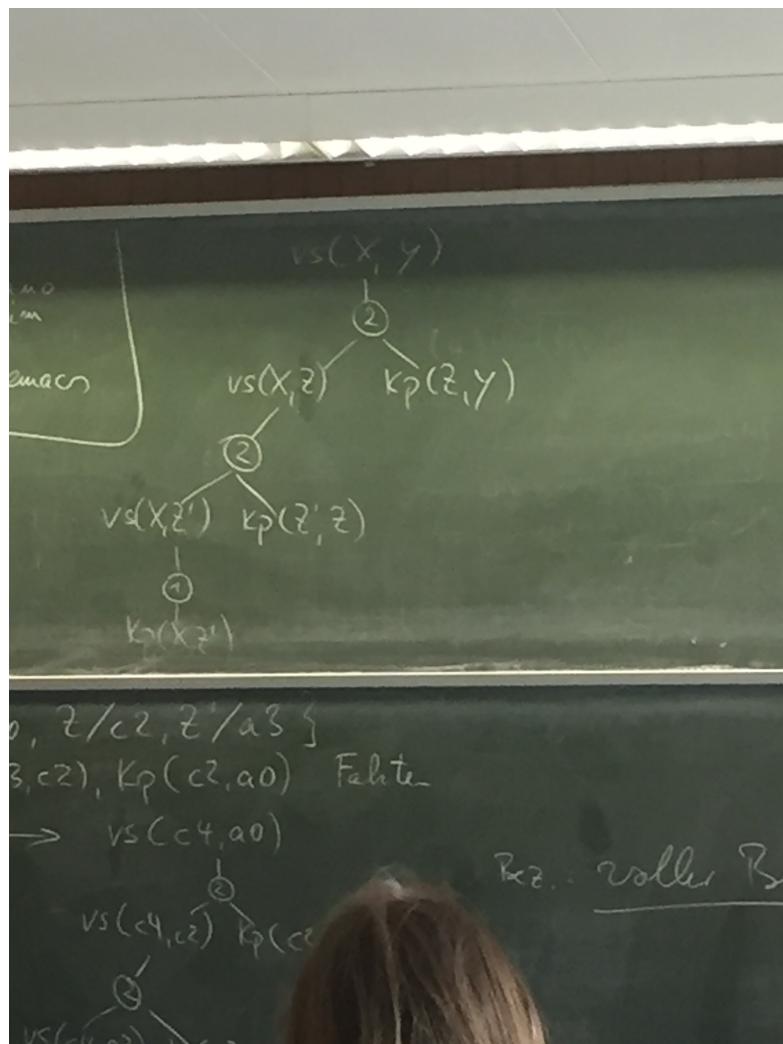


Figure 5:

Beispiel:

$$\Theta = \{X/c4, Y/a0, Zc2, Z'/a3\}$$

$$kp(c4, a3), kp(a3, c2), kp(c2, a0) \text{ Fakten}$$

Bezeichnung : Voller Beweisbaum

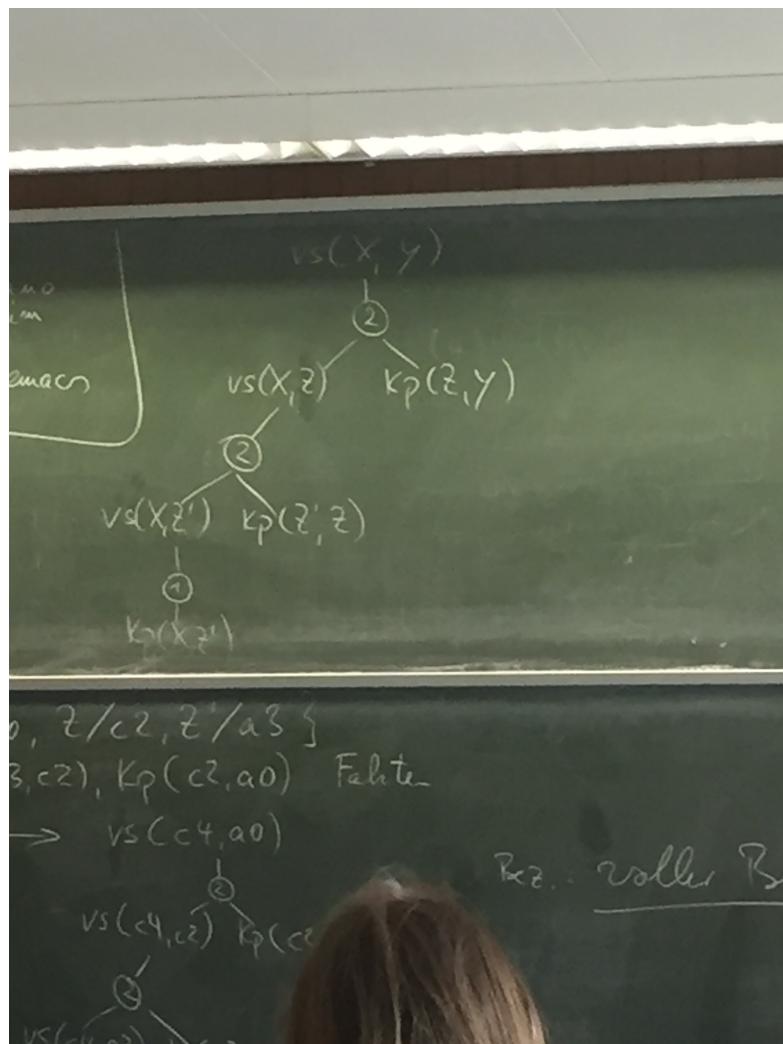


Figure 6:

Systematische Erzeugung der Suchbäume

Sukzessive alle Bäume der Tiefe 1,2, ... Erzeugen (breadth first)

Definition: Tiefe eines Baums maximale Anzahl von Zielknoten auf einem Pfad von einem Blattknoten zur Wurzel. Entsprechend Knoten der Tiefe i, Ebene i eines Baumes. Zusätzlich: Spezielle Suchbäume (Tiefe 0) für Fakten aus P.

Beispiel : Ziel $g = vs(c4, y)$

- **Tiefe 1:** $vs(c4, y), \Theta$, kein Beweisbaum
- **Tiefe 2:** $vs(c4, y)$
 $kp(c4, y), \{y/a3\}$

Offensichtlich

- Zu jedem endlichen Beweise, der mit Grundatomen “Startet”, kann ein entsprechender Beweisbaum auf den beschriebenen Weg erhalten werden.
- Zu jedem Fakt $f \in cons(P)$ gibt es einen endlichen Beweis, der mit Grundatomen startet, siehe Fixpunktberechnung → voller Beweisbaum

⇒ Methode Suchbaum / Beweisbaum ist vollständig.

Da ein Beweisbaum mit $g\Theta$ als Benennung der Wurzel einen Beweis für $g\Theta$ darstellt

⇒ Methode Suchbaum / Beweisbaum ist korrekt.

Satz 1.5

Sei P ein Datalog-Programm. Die Suchbaum / Beweisbaum Methode, angewandt auf alle Ziele $q(X_1, \dots, X_{Stelligkeit(q)})$, q intentionales Prädikatesymbol von P , liefert $cons(P)$ als Ergebnis

Frage: Abbruchkriterium bei der Erzeugung von Suchbäumen

Es gilt

- Beweisbäume sind isomorph zu den Suchbäumen, aus denen sie erzeugt wurden
- Zu jedem Fakt $f \in cons(P)$ gibt es einen vollen Beweisbaum, in dem auf jeder Zielknotenebene ein neuer Fakt (neues Teilziel) auftritt

Beweis Sei B ein voller Beweisbaum, der diese Eigenschaft nicht erfüllt. Dann gibt es ein i , so dass in B auf Ebene i nur Fakten auftreten, die schon auf vorhergehenden Ebenen auftreten. Die Teilbäume von B mit Wurzel auf Ebene i können “höher gehängt” werden ⇒ Es gibt einen äquivalenten, vollen Beweisbaum mit geringerer Tiefe.

- Es gibt nur endlich viele verschiedene Grundatome, die aus den Prädikaten und Konstantensymbolen gebildet werden können. Konkret: Seien $ap(p)$ die Anzahl verschiedener Prädikatensymbole die in p vorkommen
- $ak(p)$ die Anzahl verschiedener Konstantensymbole, die in P vorkommen

Dann ist $\max.fakt(P) = ap(P) * ak(P)^{\max.st(P)}$ eine obere Schranke für die Anzahl verschiedener Grundatome. **Damit:**

Satz 1.6

Die Suchbaum / Beweisbaum Methode bleibt vollständig für ein Programm P , wenn nur Bäume mit max. Tiefe $\max.fakt(P)$ betrachtet werden. Statt breadth first andere Vorgehensweise denkbar:

- Depth first mir Abbruch bei $\max.fakt(P)$ und backtracking
- Dynamische Abbruchbedingungen, abhängig von erzeugten Faktenmenge (limit 10)
- Erkennen und Ausnutzen identischer (Teil-) Zielknoten (\rightsquigarrow gerichteter, azyklischer Graph)

Zusammenfassen von Such und Beweisbäumen, Berücksichtigung von Fakten aus $P_F \Rightarrow$ frühzeitiges Erkennen von Suchbäumen zu denen keine Beweisbäume existieren

Sei g das Ziel (Wurzelbenennung), dann erzeugt ein Baum der mit der leeren Menge endet, genau ein Ergebnis.

$g\Theta_1, \dots, \Theta_n, (\Theta_1, \dots, \Theta_n)$ die in dieser Reihenfolge angewandten Unifikatoren (von der Wurzel aus) $\Theta = \Theta_1, \dots, \theta_n$ heißt Antwortsubstitution.

Resolutionsmethode

Für allgemeine Klauselformen entwickelte Methode zum automatischen Beweisen.

Literatur J Allen Robinson: A machine-oriented logic base on the resolution principle. Journal of the ACM 12, S.23-41, 1965

In Logikprogrammierung (Horn-Klauseln): SLD-Resolution (Linear Resolution with Selected Function for Definite Clauses).

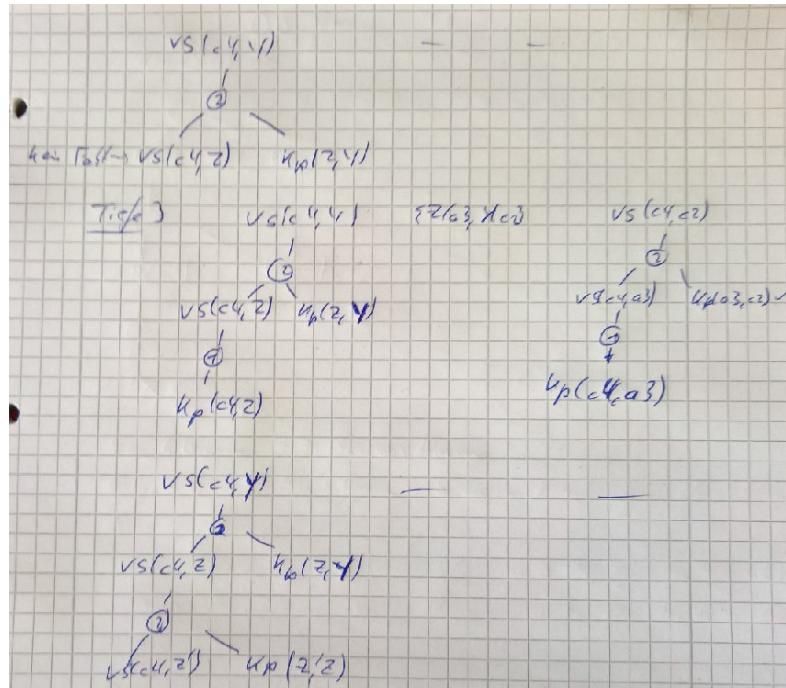


Figure 7:

- S (Selection function): Legt fest, wie Literale aus einem Ziel (Klausel mit einem oder mehreren negierten Literalen) auszuwählen sind
- L (Linear): In jedem Schritt wird genau ein Literal ausgewählt für die Unifikation mit dem Kopf einer Regel einer einem Fakt
- D (Definit): Beschränkung auf Programme mit definiten Klauseln

Bemerkung Methode steht für ganze Klasse von Algorithmen. Methode auch bekannt unter LUSH (Linear resolution with Unrestricted Selection function for Horn clauses)

Übergang Such / Beweisbaum \rightsquigarrow Resolution Nachweis der Unerfüllbarkeit einer Klausel

Seien $E = \{\neg M_1, \dots, \neg M_i, \dots, \neg M_K\}$, $r = L_0 : -L_1, \dots, L_m$ eine Regel und $S(E) = \neg M_i$. Es existiere $mgu(M_i, L_0) = 0$.

Resolvente von E und r: $\{\neg M_1\Theta, \dots, \neg M_{i-1}\Theta, \neg L_1\Theta, \dots, L_m\Theta \neg M_{i+1}\Theta, \dots, \neg M_K\Theta\}$

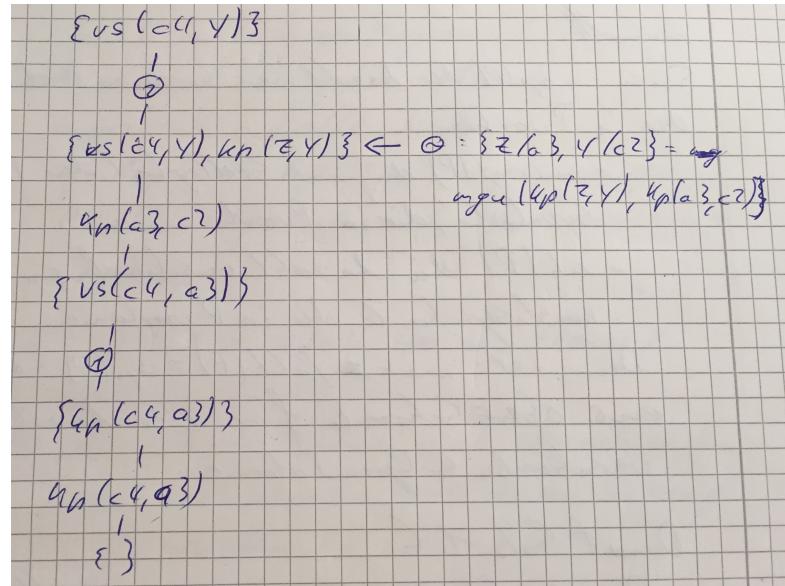


Figure 8:

- Unerfüllbarkeit: Anwendung der Methode ende mit \square (leere Klausel, entspricht Falsch)
 - Ergebnis: Sei $g = \{\neg p(t_1, \dots, t_n)\}$ das Ziel, $\{p(t_1, \dots, t_n)\Theta/\Theta$ Antwortsubstitution für eine Ableitung von g , die mit \square endet }
- Ohne Beweis.

Satz 1.7

Die SLD-Resolution liefert korrekte und vollständige Ergebnisse für alle Datalog-Programme.

Beispiel m alle Kurse $\neq a3$ mit ihren Studierenden, die Voraussetzung von c4 sind.

$$\begin{aligned}
 r_1 &= vs(X, Y) : -Kp(X, Y). \\
 &vs(c4, Y) : -vs(v4, Z), Kp(Z, Y). \\
 &m(Z, W) : -vs(c4, Z), ag(Z, X), bl(Z, W, U), Z \neq a3.
 \end{aligned}$$

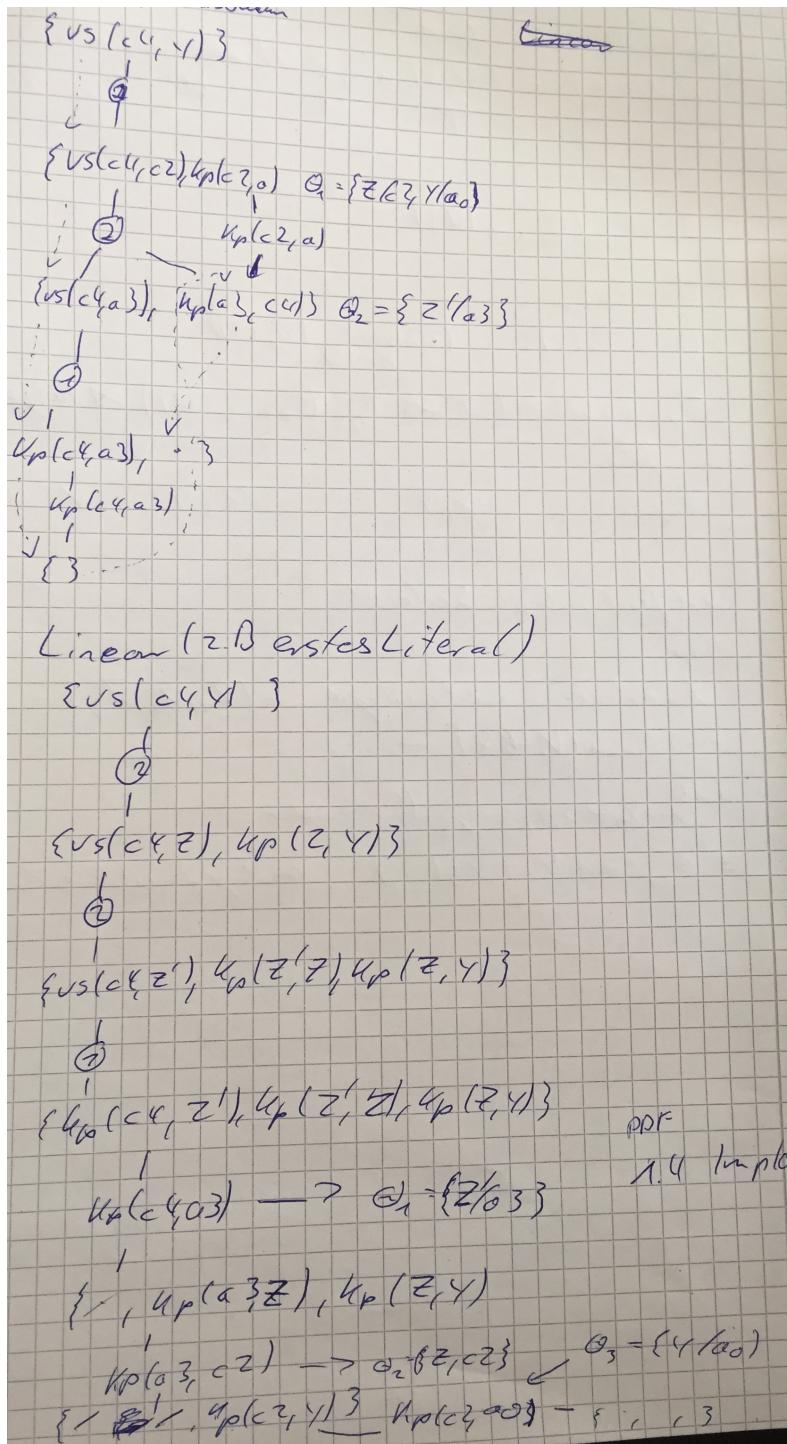


Figure 9:

Safe 3 unter Ben

$$u(V, Y, X) := \frac{+ (X, Y)_c u(S, Z, Y, \cancel{W}))}{\top} \quad \begin{matrix} V = a \\ \text{et} \\ Dv \end{matrix}$$

$S_1 = \overline{\pi}$

$S_2 = \overline{\pi_{2,3}} (\zeta_{@1=a} (u))$

$D_V = \{c\}_V$

$D_w = \overline{\pi_1}(\top) \quad \text{Umb}_{x \rightarrow w} (\overline{\pi_1}(\top))$

$\overline{G}_{V=a \wedge W=x} (\top \bowtie \overline{\pi_{2,3}} (\zeta_{@1=a} (u)) \bowtie \{a\}_V \bowtie \overline{\pi_1}(\top))$
kanne fallen

Typen: $X, Y \quad Z, Y \quad V, W$

rectified rules Seite 14

Bsp

Analogie weitergegeb $u(Z_1, Z_2, Z_3)$:- in P
 $\rightsquigarrow \Gamma_{\text{KURSPLAN}}(V, Y, X) := \dots$

Umb $V \rightarrow Y_1, W \rightarrow Y_2, X \rightarrow Y_3$ ($G_{V=a \wedge W=x} = \dots$)
 $\equiv G_{Y_1=a \wedge Y_2=Y_3} = \dots$ analog f. $u(Z_1, Z_2, Z_3)$

Figure 10:

1.Schritt

- $r_1 \checkmark$
- $r_2 \rightsquigarrow \tilde{r}_2 = vs(Q, Y) : -vs(C4, Z), kp(Z, Y), Q = c4.$
- $r_3 \checkmark$

2.Schritt

- $E_{r_1} = \text{KURSPLAN}$
- $E_{r_2} = \sigma_{Q=c4} \underbrace{\Pi_2(\zeta_{@1=c4}(VS))}_{Z} \bowtie^6 \underbrace{KURSPLAN}_{Z}, Y \bowtie \underbrace{\{c4\}_Q}_{Q}$
- $E_{r_3} = \sigma_{Z \neq a3} \underbrace{\Pi_2(\zeta_{@1=c4}(VS))}_{Z} \bowtie ANGEBOT \bowtie BELEGUNG$

⁶Join Operation

3.Schritt vereinfacht $Q \Rightarrow X^7$, damit \tilde{r}_1 und \tilde{r}_2 passen.

$$\begin{aligned} & vs(X, Y) \\ & vs(X, Y) \\ & m(Z, W) \end{aligned}$$

4.Schritt

$$\begin{aligned} VS &= \underbrace{KURSPLAN}_{XY} \cup \underbrace{(\Pi_2(\varsigma_{@1=c4}(VS)) \bowtie KURSPLAN \bowtie \{c4\}_X)}_X Y \\ M &= (\varsigma_{Z \neq a3}(\Pi_2(\varsigma_{@1=c4}(VS)) \bowtie ANGEBOT \bowtie BELEGUNG))[z, w] \end{aligned}$$

Schreibweise für Gleichungssysteme

$$R_i = E_i \underbrace{(R_1, \dots, R_n)}_{DB-Relationen}, i = 1, \dots, n$$

Vergleich der Ausruckskraft⁸ von Datalog und Relationale Algebra:

- Reines Datalog ohne Rekursion entspricht RA^+ (monotone Relationale Algebra)
- Reines Datalog ohne Rekursion entspricht RA^+ ohne Gleichungssysteme ($\downarrow RA^+$)

Beispiel: Kursplan (kp):

X	Y
c4	a3
a3	c2
c4	a2
c2	a0

$$vs(X, Y) : -Kp(X, Y). \quad vs(X, Y) : -\underbrace{vs(X, Z)}_{XZ}, \underbrace{kp(Z, Y)}_{ZY}. \rightsquigarrow vs = \Pi_{1,3}(VS \bowtie KURSPLAN) \cup KU$$

Jacobs entspricht Gauss-Seidel

⁷kommt nicht im Rumpf von \tilde{r}_2 vor

⁸genauere Definition später

$$R^0 = \emptyset$$

$$Q^1 = \emptyset$$

$$R^1 = KP \leftarrow (\emptyset_{X2} \bowtie KP_{ZY})[X, Y]$$

$$Q^2 = KP$$

$$R^2 = \{(c4, c2), (a3, a0)\} \cup KP \quad (KP_{XZ}) \bowtie KP_{ZY})[X, Y] \cup KP_{X,Y}$$

$$Q^3 = R^2$$

$$R^3 = \{(c4, a0)\} \cup R^2 \quad (((KP_{XZ} \bowtie KP_{ZY})^9)[X, Y] \cup KP_{XY})_{XZ})[X, Y] \cup KP_{XY}$$

Anmerkung

Berücksichtigung der “neuen” Information für VS hätte genügt. Grund: $(VS' \cup VS'') \bowtie KURSPLAN = VS' \bowtie KURSPLAN \cup VS'' \bowtie KURSPLAN$

Nach Schritt 4

$$\begin{aligned} R_1 \bowtie R_2 \cup R_2 \bowtie R_3 &\Rightarrow^{mit \delta-Berücksichtigung 10} \\ \delta R_1 \bowtie \cup R_2 \bowtie R_3 \cup R_1 \bowtie \delta R_2 \cup \delta R_2 \bowtie R_3 \cup R_1 \bowtie R_2 \cup R_2 \bowtie \delta R_3 \\ &\Rightarrow^{11} \delta R_1 \bowtie R_2 \cup R_1 \bowtie \delta R_2 \cup \delta R_2 \bowtie R_3 \cup R_w \bowtie \delta R_3 \end{aligned}$$

⁹Es wird zu viel berechnet beim Join. Siehe unten

¹¹Ist suboptimal, weil wieder zu viel berechnet wird

$$\begin{aligned}
 \delta R^0 &= KP \\
 R^0 &= KP \\
 \delta Q^1 &= KP \\
 \delta R^1 &= \{(c4, c2), (a3, a0)\} \cup KP \\
 \delta R^1 &= \{(c4, c2), (a3, a0)\} \cup KP \\
 R^1 &= Kp \cup \{(c4, c2), (a3, a0)\} \\
 \delta Q^2 &= \{(c4, c2), (a3, a0)\} \\
 \delta R^2 &= \delta Q^2 \bowtie Kp \cup Kp = \{(c4, a0)\} \cup Kp \\
 R^2 &= Kp \cup \{(c4, c2), (a3, a0)\} \cup \{(c4, a0)\} \\
 \delta Q^3 &= \{(c4, a0)\} \\
 \delta R^3 &= \delta Q^3 \bowtie Kp \cup Kp = Kp \\
 \delta R^3 &= \emptyset \\
 R^3 &= R^2 \cup \emptyset
 \end{aligned}$$

① $g = vs(c4, y)$

P: $\begin{array}{l} vs(x, y) = kp(x, y) \\ vs(x, y) = vs(z, k), kp(z, y) \end{array} \rightsquigarrow \begin{array}{l} vs(c4, y) = kp(c4, y) \\ vs(c4, y) = vs(c4, z), kp(z, y) \end{array}$

Figure 11:

$g = vs(c4, y)$

P: $\begin{array}{l} vs(x, y) = kp(x, y) \\ vs(x, y) = vs(z, k), kp(z, y) \end{array}$

top down: $\begin{array}{l} vs(c4, y) \\ vs(c4, z) kp(z, y) kp(c4, y) \\ vs(c4, z) kp(z, z) kp(c4, z) \end{array}$

$\begin{array}{l} \text{magic_vs (c4).} \\ vs(x, y) = kp(x, y) \\ vs(x, y) = \text{magic_vs}(x), vs(x, z), kp(z, y) \end{array}$

X leaves with 'c4' ameh.

Figure 12:

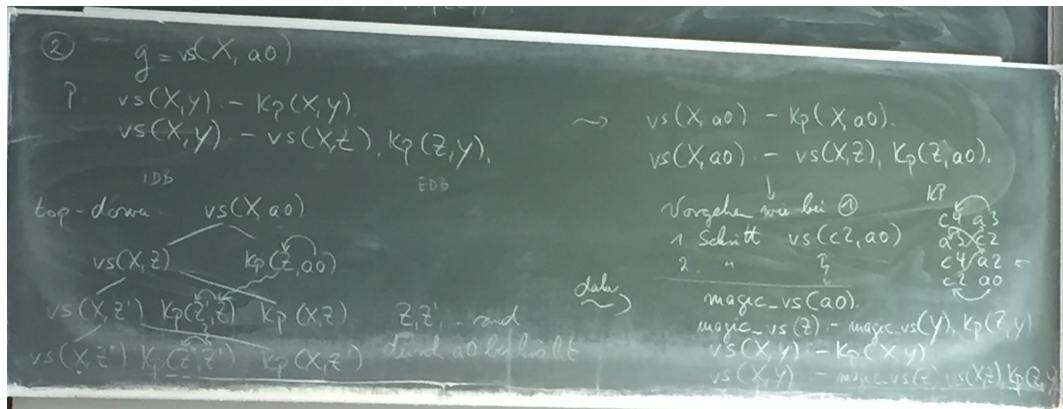


Figure 13:

Beispiel

$$r_2 = vs^{fb}(X, Y) : -vs^{fb}_(X, Z), Kp(Z, Y).$$

i) ✓

ii) ... magic_r2_vs^{fb}_1(X, Z) ...

iii) ... magic_r2_vs^{fb}_1(Z) ...

iv) vs^{fb}(Y)

$$magic_r2_vs^{fb}_1(Z) : -magic_vs^{fb}(Y), kp(Z, Y).$$

$$r_0 = query^f(X : -vs^{fb}_(X, a0)).$$

i) ...

$$ii) query^f(X) : -magic_r0_vs^{fb}_1(X, a0)$$

$$iii) query^f(X) : -magic_r0_vs^{fb}_1(a0)$$

iv) -

$$v) vs^{fb}(Y)$$

$$magic_r0_vs^{fb}_1(a0) : -magic_query^f().$$

$$r_0 \rightsquigarrow ^{12} magic_r0_vs^{fb}_1(a0) : -.$$

$$r_2 \rightsquigarrow magic_r2_vs^{fb}_1(Z) : -magic_vs^{fb}(Y), kp(Z, Y).$$

¹² “Sideways information passing”

$$\begin{aligned} p(X) &: \neg\neg q(X) \\ p(X) \cup q(X) \\ q(X) &: \neg\neg p(X) \end{aligned}$$

$$P = \{ag(a3, o) : \neg\neg ag(a3, m)\}.$$

“Falls ‘m’ Kurs a3 nicht anbietet, bietet o den Kurs an”

Mit CWA:

$$\begin{aligned} ag(a3, m) &\notin ANGEBOT \\ ag(a3, o) &\notin ANGEBOT \end{aligned}$$

Da beide keine logische Folgerung von P (müsste in allen Modellen gültig sein)
Nimm Modelle $\{ag(a3, o)\}$.

$$\begin{aligned} \{ag(a3, o)\} \\ \Rightarrow^{CWA^{13}} \neg ag(a3, m) \text{ und } \neg ag(a3, o) \text{ sind gültig.} \\ \text{Aber } \neg ag(a3, m) \Rightarrow ag(a3, 0) \rightsquigarrow \text{Widerspruch zur CWA} \end{aligned}$$

Beachte folgende HB-Interpretation von P.

$$\begin{aligned} I_1 &= \{ag(a3, o)\} \text{ und } I_2 = \{ag(a3, m)\} (I_1, I_2 \text{ sind minimale Modelle}) \\ I_3 &= \{ag(a3, o), ag(a3, m)\} \text{ ist ebenfalls Modell von P} \end{aligned}$$

In I_1 gilt mit CWA: $\neg ag(a3, m)$

In I_2 gilt mit CWA: $\neg ag(a3, o)$

Grund für Problem: Implikation $\neg ag(a3, m) \Rightarrow ag(a3, o)$ äquivalent zu $ag(a3, m) \vee ag(a3, 0)$. Welches Modell soll ausgezeichnet werden? Semantik?

¹³Closed World Assumption

Offensichtlich: I Modell von P \diamond I Modell von $P' = \{ag(a3, m) : -ag(a3, o)\}$
 Gehe wie bei Fuxpunktberechnung vor:

P Bekannte Fakten $EDB = \{ag(a1, m), ag(c4, q), ag(a3, d), \dots\}$
 $ag(a3, m) \notin EDB$
 CWA: $\neg ag(a3, m)$ gilt $\Rightarrow^{\text{Regel}} ag(a3, 0)$

P' $ag(a3, o) \notin EDB$
 CWA: $\neg ag(a3, 0)$ gilt $\Rightarrow^{\text{Regel}} ag(a3, m)$

Daraus folgt: Wähle $I_1(I_2)$ als Modell (Semantik) von P(P') \rightsquigarrow Auswertungsrichtung gemäß Implikation legt Semantik fest. D.h. Semantik ist abhängig von der Art des Aufschreibens der Klauseln. Daraus ergibt sich die Frage nach einer besseren Semantikdefinition.

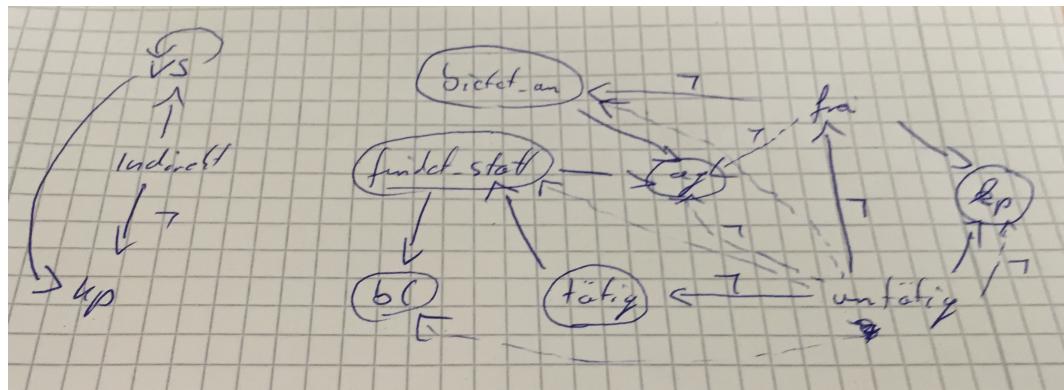


Figure 14:

$$\begin{aligned}
 & r(1), s(1), s(2). \\
 U(X) : -r(X).r(1) & \rightsquigarrow u(1) \\
 q(X) : -s(X), \neg u(X). & \Leftarrow s(1) \rightsquigarrow q(1) \\
 & \Leftarrow s(2) \rightsquigarrow q(2)
 \end{aligned}$$

Disjunktives Datalog

Einige Bemerkungen zu disjunktivem Datalog. Negationen im Rumpf sind umgeschrieben genau ein Oder (Disjunktion).

Form 6: $q_1(\dots), \dots, q_n(\dots) : -p_1(\dots), \dots, p_n(\dots)$.

Negation in Rümpfen erlaubt \Rightarrow Frage: Sind Disjunktionen im Kopf von Regeln notwendig?

Beispiel: Tim ist 18 oder¹⁴ 16 Jahre alt: $alter(tim, 18), alter(tim, 16) : -.$

Logisch äquivalent zu:¹⁵

$$\begin{aligned} alter(tim, 18) &: --\neg alter(tim, 16). \\ alter(tim, 16) &: --\neg alter(tim, 18).^{\text{16}} \end{aligned}$$

\rightsquigarrow Falls auf "Tim ist nicht 18 Jahre alt." etwas mit CWA geschlossen werden kann, folgt definitiv "Tim ist 16 Jahre alt.".

Mehrere Semantiken in Literatur vorgeschlagen, etwa:

"General Closed World Assumption" (GCWA)

Idee: Weil mehrere minimale Herbrand Modelle möglich sind \rightsquigarrow Schluss auf das Negative, $\neg p$, genau dann möglich wenn p in keinem minimalen Modell enthalten ist.

Beispiel: $\{alter(tim, 18)\}, \{alter(tim, 16)\}$ sind die beiden minimalen Modelle $\{alter(tim, 18), alter(tim, 16)\}$ ist auch ein Modell, aber nicht minimal.

Minimale Modelle: $\{q, r\}, \{q, s\}$

$$\begin{aligned} P = \{p, q : - . \\ q : - . \\ r, s : - .\} \end{aligned}$$

p ist in keinem minimalen Modell enthalten $\rightsquigarrow \neg p$ kann unter GCWA angenommen werden.

¹⁴ Altagserfahrung: Exklusiv, Theoretisch liegt dieses Wissen um exklusivität nicht vor.

¹⁵ Tim ist 18 oder 16 Jahr alt, mehr wissen wir nicht (oder formal gesehen inklusives Oder)

¹⁶ Tim ist 18 Jahre alt, wenn er nicht 16 Jahre alt ist

Eigenschaften GCWA erfüllt die Eigenschaften der

- + Konsistenz: Negative Information, die auf Grund der GCWA hinzugefügt wird, führt nicht zu Widerspruch in der Theorie.
- + Stabilität: Negative Information, die auf Grund der GCWA hinzugefügt wird, verringert die positive Information nicht und vergrößert sie auch nicht.
- + Reduktion: Für definite Programme entspricht die GCWA der CWA.
- - Monotonie: CWA garantiert Monotonie, GCWA nicht.

Monoton

Seien P, P' definite oder disjunkte Programme. Sei R eine Negationsregel mit $R(P)$ definiert die Menge der negativen Atome, die gemäß R für P angenommen werden können.

R ist monoton $\Diamond P \subseteq P' \Rightarrow R(P') \subseteq R(P)$

Beispiel

$$\begin{aligned} P &= \{p(a) \vee p(b)\} \\ P' &= P \cup \{p(a)\} \\ P'' &= P' \cup \{p(b)\} \end{aligned}$$

Success:

$$\begin{aligned} \text{Success von } P &= \{p(a) \vee p(b)\} \\ \text{Success von } P' &= \{p(a) \vee p(b), p(a)\} \\ \text{Success von } P'' &= \{p(a) \vee p(b), p(a), p(b)\} \end{aligned}$$

Unknown:

$$\begin{aligned} \text{Unknown von } P &= \{p(a), p(b)\} \\ \text{Unknown von } P' &= \emptyset \\ \text{Unknown von } P'' &= \emptyset \end{aligned}$$

Mit GCWA:

$$\begin{aligned} \text{Durch GCWA von } P &= \emptyset \\ \text{Durch GCWA von } P' &= \{\neg p(b)\} \\ \text{Durch GCWA von } P'' &= \emptyset \end{aligned}$$

Dadurch ist zu sehen, dass die Monotonie verletzt (zwischen P' und P''), damit ist die GCWA nicht monoton.

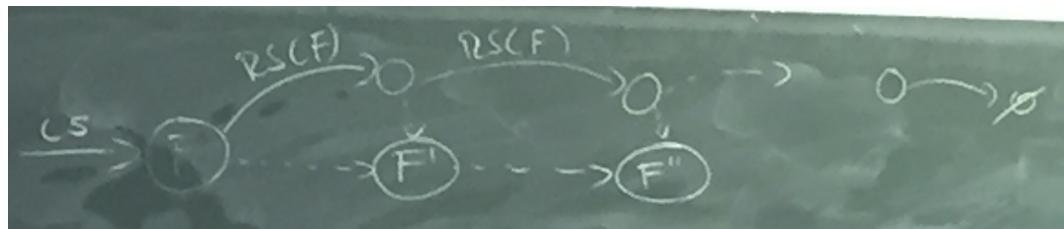


Figure 15:

$\text{ANCESTRY}(X, Y) :- \neg \text{MOTHERHOOD}(X, Y)$.

$\text{ANCESTRY}(X, Y) :- \neg \text{MOTHERHOOD}(X, Z), \text{ANCESTRY}(Z, Y)$.

KOMponenten		FLÜGELTEILE	
Berechnung	Teil Anzahl	Berechnung	Teil Anzahl
Flügel	Strolche 5	Strolche	5
Flügel	Fahrgestell 1	Fahrgestell	1
Flügel	Niete 100	Niete	100
Fahrgestell	Niete 8	Niete	50
Fahrgestell	Schrauer 3	Niete	8
Schrauer	Niete 4	Schrauer	3
Strolche	Niete 10	Strolche	12

initial subquery

recursion (1)

(2)

Figure 16:

Rekursion in SQL

Flüge:

Flugnr.	Ab	Ziel	Kosten
LH577	FRA	JSK	700
RA170	LHR	JFK	400
LH122	HAM	FRA	100
BA730	FRA	LHR	150
...

Reisen:

Ziel	Route	Gesamtkosten
fra	“fra”	100 ¹⁷
jfk	“FRA,JFK”	800 ¹⁸
lhr	“Fra,lhr”	250 ¹⁹
jfk	“fra,lhr,jfk”	650 ²⁰
...

Negation in SQL

(except & not)

Forderung : Stratifikation

Beispiel : Gegeben DAG(K1, K”). Gesucht Paare von Knoten, die nur mit Pfade in gerader Länge verbunden sind.

```
with recursive UAB(K1, K2) as
(( select * from DAG)
union
(( select d.K1, g.k2
  from DAG d, GAB g
  where d.K2 = g.K1)
except
  (select * from GAB))) ,
GAB(K1, K2) as (select d.K1, UAB u where d.K2 = u.K1)
))
```

21

```
with recursive UAB(K1, K2) as
(( select * from DAG)
union
(( select d.K1, u.K2 from DAG d, DAG d1 , UAB u
```

²¹GAB noch nicht bekannt

```

where d.K2 = d1.K1 and d1.K2 = u.K1)
except
  (select * from GAB)) ,
  GAB(K1, K2) as ((select d.K1, d.K2 from DAG d, DAG d1
    where d.K2 = d1.K1)
  union
    (select d.K1, g.K2 from DAG d, DAG d1, GAB g
      where d.K2 = d1.K1 and d1.K2 = g.K1))
)
)
select x from UAB;

```

2: Ausdruckskraft und Komplexität von Anfrage-sprachen

Ausdruckskraft infromell: Welche Information kann mit Hilfe der Anfrage der Sprachart einer beliebigen Datenbank ermittelt werden.

Formalisierung für relationale Datenbanken

Definition: Anfrage

Sei $\sigma = \{(RT_1, \alpha_1), \dots, (RT_n, \alpha_n)\}$ ein relationales Datenbankschema über $\alpha = \bigcup_{i=1}^n \alpha_i$ mit Wertebereichsfunktion dom . Sei α in eine genügend große Attributmenge α_0 eingebettet²².

Eine Anfrage q auf σ ist eine partielle Funktion

$$q|Z\sigma \Rightarrow R_\beta^\infty$$

dabei gilt $\beta \subseteq \alpha_0$ und R_β^∞ ist die Menge der verallgemeinerten DB-Relationen über β . (unendliche Tupelmenge erlaubt)

²²Enthalten darin

Definition: Anfragesprache

Eine Anfragesprache zu σ ist eine Menge L_0 von Ausdrücken zusammen mit einer Bedeutungsfunktion (in Zeichen: $(L_\sigma^{23}, \mu^{24})$), so dass für jeden Ausdruck $e \in L_\sigma$ gilt: $\mu(e)$ ist eine Anfrage von σ

Definition: Ausdruckskraft

Die Ausdruckskraft einer Anfragesprache (L_σ, μ) zu einer DB-Schema σ ist definiert als $\mu(L_\sigma) =_{Def} \{\mu(e) | e \in L_\sigma\}$. Eine Sprache (L_σ, μ') ist **ausdrucksstärker** als eine Sprache (L_σ, μ) , wenn gilt: $\mu(L_\sigma) \subseteq \mu'(L'_\sigma)$ Im Fall $\mu(L_\sigma) = \mu'(L'_\sigma)$ werden die Sprachen **äquivalent** genannt

Informell Welche Information können mit Hilfe der Anfragen der Sprache aus einer beliebigen Datenbank ermittelt werden

Problem Information mit Algebra geht nicht: “Die Mitarbeiter die über 10k verdienen sind grade die Manager”. Intrinsiche Informationen stellen ein Problem dar.

Definition: Anfragesprache von ρ

Sei $\rho = \{(RT_i, \alpha_i) | i \in Lm\}$ ein relationales Datenbankschema über $\alpha = \bigcup \alpha_i$ mit Werteberichsfunktion dom . Sei α in eine genügend große Attributmenge α_0 eingebettet. Eine Anfrage q ist eine partielle Funktion mit $q : \delta_\rho \rightarrow R_\beta^\infty, \delta \subseteq \alpha_0, R_\beta^\infty$ ist die Menge der verallgemeinerten DB-Relationen über β (unendliche Teilmengen sind erlaubt).

Eine Anfragesprache zu ρ ist eine Menge L_ρ von Ausdrücken mit einer Bedeutungsfunktion μ (schreibe (L_ρ, μ)), so dass für jeden Ausdruck $e \in L_\rho$ gilt $\mu(e)$ ist eine Anfrage an ρ .

²³Sprache

²⁴Bedeutungsfunktion

Definition: Ausdrucksstärke einer Anfragesprache von ρ

Die Ausdrucksstärke einer Anfragesprache (L_ρ, μ) zu ρ ist definiert als $\mu(L_\rho) = \{\mu(e) | e \in L_\rho\}$

Definition: Äquivalenz von Sprachen

$(L'_\rho, \mu') = (L_\rho, \mu)$, wenn $\mu'(L'_\rho) = \mu(L_\rho)$. Kleiner und größer analog. Falls Anfragesprache L für alle ρ äquivalent zu Anfragesprache L' ist werden beide als äquivalent bezeichnet.

Vergleich Datalog - Relationenalgebra

- Reines Datalog ohne Rekursion entspricht RA^+ (monotone Relationenalgebra)
- Reines Datalog mit Rekursion entspricht RA Gleichungssysteme
- Datalog mit Negation ohne Rekursion entspricht RA
- volles Datalog > RA
- Datalog mit geschichteter Negation < Inflationäre Semantik

Berechenbarkeitsmodelle

Alegräisch logische Definitionen

- allgemein rekursivee Funktionen (Gödel / Herbrand 1936)
- λ -def. Funktionen (A. Church 1936)
- μ -rekursivee Funktionen und partiell definierte Funktionen (Gödel / Kleene 1936)

Wortersetzungssysteme

- Turing Maschine (1936)
- Postsche Kanonische Systeme (1945)
- Markov-Algorithmen (1951)

Theoretische Berechnungsmodelle

- unbeschränkte Registermaschine (1963)
- Random Access Machine (1964)

Ausdruckskraft von Relationenalgebra und ???

Grobe Charakterisierung der RA (ohne Erweiterung)

- Operatoren können keine neuen Werte erzeugen, d.h. Werte in Ergebnissen müssen in Operanden auftauchen
- es können beliebige Relationstypen über den gegebenen Wertebereichen auftreten / erzeugt werden (Umbenennung)
- Für einen festen Datenbankzustand δ zu ρ gilt: mit RA-Ausdrücken können alle Relationen erzeugt werden, die nur Informationen erhalten der schon in Z enthalten ist.

Operatoren: $\cup, \cap, \setminus, \bowtie$, Projektion, Selektion (mit allg. Vergleichsausdrücken, Division, Umbenennung)

Sonderstellung: Komplement

Sei R eine DB Relation über β , T_β sei die Menge aller DB-Tupel über β . $R[\text{kompl}] = T_\beta \setminus R$ (i.A. eine unendliche Tupelmenge)

Beispiel: Noch offene Blätter für jeden angemeldeten

Satz 2.1

Sei e ein zu einem gegebenen Datenbankschema ρ passender RA-Ausdruck ohne Komplement. Dann gibt es einen äquivalenten zu e passenden Ausdruck der RA e' in dem als Operatoren nur Selektionen mit einfachen Vergleichsausdrücken, direktes Produkt, Projektion, Umbenennung, Differenz, Vereinigung vorkommen. Als Operanden ermittelt e' neben Relationstypbezeichnern aus ρ nur (extensionale) DB-Relationen der Form $\{(A, C), A \in \alpha_\rho, c \in \text{dom}(A)\}$

Weitere Einschränkungen sind möglich → Projektionen nur auf ein Attribut, Vergleichsausdrücke = und \neq .

Falls Komplementbildung hinzugenommen, Differenz nicht mehr notwendig:

$$R \setminus S = (R[kompl] \cup S)[kompl]$$

Satz 2.2

Eine derart ??? Operationsmenge der RA ist minimal, d.h. es kann keine Operation entfernt werden, ohne dass die Ausdruckskraft eingeschränkt ist.

Beweisidee: Diskutiere die speziellen Eigenschaften und zeige dass diese Operation über Eigenschaften verfügt, die keiner anderen Operation oder Kombination von Operationen zukommt. Z.B. Vereinigung: Streiche, Neuverknüpfung 2er Relationen dann nur noch über Differenz und direktes Produkt möglich. Vereinigung erlaubt das hinzufügen eines neuen Tupels unter Attribut Kombination, direktes Produkt und Differenz nicht

Mit der Relationenalgebra kann nicht allgemein die transitive Hülle berechnet werden.

Satz 2.3

Sei RT der Bezeichner eines beliebigen, zweistelligen Relationstyp über abzählbaren unendlichen Wertebereich. Sei Rt^* der Relationstyp für die transitive Hülle von Relationen des Typs RT. Es gibt keinen Ausdruck der Relationenalgebra mit der Eigenschaft $e(RT) = RT^*$.

Gegeben Wertebereich $\{a_1, a_2, \dots\}$ ohne Ordnungsrelation. Betrachte $R_l = \{(a_i, a_i + 1) | i \in [l - 1]\}$ für $l \in \mathbb{N}\}$. R_l ist eine mathematische Relation, die Tupel sind also geordnet. Zeige $e(R_l) \neq R_{*l}$ für jeden RA - Ausdruck e, der zu Wertebereichen passt und für genügend großes l. $e(R_l)$ bedeutet R_l für RT eingesetzt. RA-Operationen sind anzupassen (wir haben keine Bezeichner \leadsto Permutationen einführen).

- Projektion: ²⁵ erlaubte Permutationen (vgl $p(x) :- r(y,x)$ ist Projektion in r)
- Selektionen mit atomaren Vergleichsausdrücken der Form $i = a_m, i \neq a_m, i =$

²⁵entspricht \exists : es gibt da was in der spalte, aber was das ist, ist mir egal

$j, i \neq j$ für $i, j \in [k], m \in [l]$ k ist bestimmt durch Größe der konstruierten Tupel $(b_1, \dots, b_k), b_i \in \{a_1, \dots, a_l\}$

Beweis

$$\begin{aligned} R_l &= \{(a_1, a_2), (a_2, a_3), \dots, (a_{l-1}, a_l)\} \\ e(R_l) &= R_l^+ \end{aligned}$$

Einfache Erweiterung, um Hüllenberechnung zu ermöglichen:
Erlaubte Gleichungen (vgl. Datalog-Übersetzung) der Form

$$\begin{aligned} RT &= f(RT) \\ (RT, \{A_1, \dots, A_k\}), \{A_1, \dots, A_k\} &\subseteq \alpha_0 \\ RT &\in \{RT_1, \dots, RT_m\} \end{aligned}$$

$f(RT)$: RA-Ausdruck, der bis auf RT zu σ passt, und der nicht die Differenz enthält (\sim monoton).

Beispiel : Sei $(S, \{A, B\})$ ein binärer Relationstyp und o die Komposition von Relationen mit Darstellung $R_1 o R_2 = (R_1 \bowtie R_2)[1, 4]$.

Hülle von S: kleinster Fixpunkt von $R = RoS \cup S$.
Berechnung (Fixpunktsatz): $f^{m_0}(\emptyset) = \bigcup_{i=1}^{\infty} So \cdots o S^{26}$, m_0 Zahl mit $f^{m_0}(\emptyset) = f^{m_0+1}(\emptyset)$

Selektionen:

$$i = a_m \quad i \neq a_m \quad i = j \quad i \neq j$$

Andere Formen sind nicht von Interesse, da Wertebereich ohne Ordnungsrelation und ohne spezielle Operationen festgelegt. Z.B. nicht möglich $(R_l[1] \bowtie R_l[2])[2 > 1]$. Elemente Operatorn in RA-Ausdrücken, die zu RT passen:

- RT
- endl. Teilmenge des Wertebereichs

²⁶i-Mal

Betrachte Kalkülausdruck der Form $\{(X_1, \dots, X_K)/\Psi(X_1, \dots, X_K)\}$ mit folgenden Vergleichsausdrücken:

- $X_i = a_m, \quad X_i \neq a_m$
- $X_i = X_j \uparrow c, \quad X_i \neq X_j \uparrow c, \quad c \in \mathbb{Z}$ beliebig, mit Interpretation (b_1, \dots, b_K) erfüllt $X_i = X_j \uparrow c \Leftrightarrow \neg(\exists m)(b_j = a_m \wedge b_i = a_{m+c})$

Beispiel: $(a_1, a_4, a_3, a_5, a_1)$ erfüllt $X_4 = X_2 \uparrow 1$. Setze ein $m = 4 \quad b_2 = a_4 \quad b_4 = a_5$.

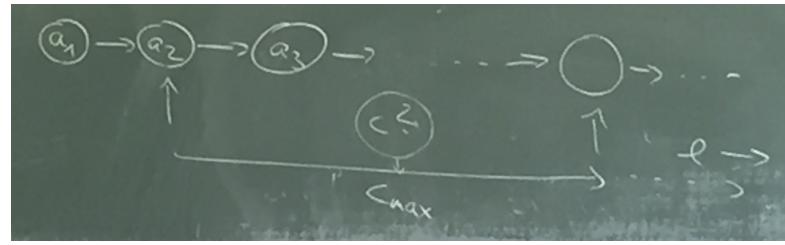


Figure 17:

Idee:

Lemma 2.1

Sei e ein beliebiger RA-Ausdruck, der zum Wertebereich $\{a_1, a_2, a_3, \dots\}$ passt und zu RT. Dann lässt sich $e(R_l)$ für ein genügend großes l darstellen als

$$e(R_l) = \{(X_1, \dots, X_K)/\Psi(X_1, \dots, X_K)\} \subseteq \{a_1, \dots, a_l\}^K$$

mit $K \in \mathbb{N}$, Ψ aussagenlogischer Ausdruck in disjunktiver Form, wobei atomare Ausdrücke nur Vergleichsausdrücke der oben genannten Form auftreten.

Beweis: (durch Induktion über den Aufbau von e) **Annahme:** Abarbeitungsreihenfolge für e festgelegt, etwa durch vollständige Klammerung. Die Tiefe von e sei m .

Induktionsanfang: $n = 0 \rightsquigarrow$ Operand ist R_l oder Teilmenge $\{c_1, \dots, c_m\} \subseteq \{a_1, \dots, a_l\}$

$$\begin{aligned} R_l &: \{(X_1, X_2) | X_2 = X_1 \uparrow 1\}^{27} \\ \{c_1, \dots, c_n\} &: \{(x) | x = c_1 \vee x = c_2 \vee \dots \vee x = c_m\} \end{aligned}$$

Induktionsannahme: Lemma gilt für alle Anfragen mit Tiefe kleiner gleich m.
Induktionsschluss:

$$\begin{aligned} e &= e_1 \cup e_2 \quad e = e_1 \setminus e_2 \quad e = e_1 \times e_2 \\ \Psi_1() \wedge \neg\Psi_2() \quad (b_1, \dots, b_k, b'_1, \dots, b'_{k'})/\Psi_1() \wedge \Psi_2() \quad ()/\Psi() \wedge F() \end{aligned}$$

Schwieriger: Projektion, allgemein darstellbar als:

Permutationen + Projektionen mit Elimination der letzten Stelle. Wirkung von Permutationen klar (wir haben in Ψ ja Variablen) \Rightarrow interessant: $e = e_1[1, \dots, K-1]$.

Sei $e \sim \{(X_1, \dots, X_K) | \Psi(X_1, \dots, X_K)\} \rightsquigarrow e \sim \{(X_1, \dots, X_{K-1}) | (\exists X_K)(\Psi(X_1, \dots, X_K))\}^{28}$

$\Psi = \Psi_1 \vee \dots \vee \Psi_P$ nach Annahme, Ψ_i konjunktiver Ausdruck

Damit $e \sim \bigcup_{i=1}^P \{(X_1, \dots, X_{K-1}) | (\exists X_K)(\Psi_i(X_1, \dots, X_K))\}$

\Rightarrow Betrachte nur noch *konjunktive Ausdrücke* Ψ (\cup erledigt)

Fall 1: Es gibt in Ψ keine atomaren Ausdrücke der Form $X_K = a_j, X_K = X_i \uparrow c, X_i = X_K \uparrow c$ mit K Index der letzten Komponente. Sei Ψ' die Konjunktion aller Atome von Ψ , die kein X_K enthalten²⁹

Dann gilt

$$* e \sim \{(X_1, \dots, X_{K-1}) | \Psi'(X_1, \dots, X_{K-1})\}$$

Beispiel

$$\begin{aligned} e_1 &\sim \{(X_1, X_2, X_3) | X_2 = X_1 \uparrow 1 \wedge X_2 \neq X_3 \uparrow 0 \wedge X_3 \neq X_1 \uparrow 2\} \\ e &= e_1[1, 2] \rightsquigarrow e \sim \{(X_1, X_2) | X_2 = X_1 \uparrow 1\} \end{aligned}$$

Begründung für *

Annahme: (b_1, \dots, b_{K-1}) erfüllt Ψ' , für genügend großes l kann für X_k ein a_m gewählt werden (\exists), so dass alle Atome $X_k \neq a_j, X_k \neq X_i \uparrow c, X_i \neq X_K \uparrow c$ erfüllt sind. Dan erfüllt (b_1, \dots, b_{K-1}) auch $(\exists X_K)(\Psi(X_1, \dots, X_K))$.

Umgekehrt: (b_1, \dots, b_{K-1}) erfülle $(\exists X_K)(\Psi(X_1, \dots, X_K))$. Dann erfüllt (b_1, \dots, b_{K-1}) insbesondere alle Atome, die X_K nicht enthalten.

²⁷Wegen $R_l = \{(a_1, a_2), (a_2, a_3), \dots\}$

²⁸Nicht gewünschte Form

²⁹Wirf Atome mit $X_K \neq \dots$ raus

Fall 2: Es gebe in Ψ ein Atome der Form $X_K = a_j$, $X_K = X_i \uparrow c$ oder $X_i = X_K \uparrow c$ sei es mit ξ benannt.

Falls mehrere vorhanden: wähle ein aus, ersetze X_K in allen anderen Vorkommen durch a_j , $X_i \uparrow c$ bzw $X_i \uparrow c'(c' = -c)$. Lasse ξ weg.

Mögliche Folge dieser Einsetzung im Ausdruck Ψ :

- atomare Ausdrücke, die direkt zu wahr oder falsch ausgewertet werden können:
Bei wahr weglassen, bei falsch $e \sim \{X_1, \dots, X_{K-1}\} = \emptyset$
- nicht zulässige Ausdrücke: \rightsquigarrow geeignet umformen:

$$\begin{aligned} a_j \uparrow c &\rightarrow a_{j+c} \quad a_j = X_k \uparrow c \rightarrow X_K = a_{j-c} \\ X_i \uparrow c &= a_i \rightarrow X_i = a_{j-c} \\ X_i \uparrow c \uparrow c' &\rightarrow X_i \uparrow c'', c'' = c + c' \\ X_i \uparrow c &= X_k \uparrow c' \Rightarrow X_i = X_k \uparrow c'' \text{ mit } c'' = c' - c \end{aligned}$$

Analog bei \neq Falls nicht “falsch” als Ergebnis enthalten werden kann gilt $e \sim \{(X_1, \dots, X_{K-1}) | \Psi'(X_1, \dots, X_{K-1})\}$, wobei Ψ' aus Ψ wie folgt erhalten wird:

- Ersetze und forme um wie oben angegeben
- Ausgewähltes Atom ξ hat die Form $X_K = X_i \uparrow c, c \geq 0$ oder $X_i = X_k \uparrow c, c \leq 0$. Füge Atome $X_i \neq a_j$ für $l - c < j \leq l$ hinzu.³⁰
- Ausgewähltes Atom ξ hat die Form $X_K = X_i \uparrow c, c < 0$, oder $X_i = X_K \uparrow c, c > 0 \rightarrow$ Füge Atome $x_i \neq a_j$ für $1 \leq j \leq c$ hinzu. Damit: Zu jeder Belegung a_i von x_i wird ein a_j mit $a_j = a_{i+c}$ gefunden.

Beispiel: $(\exists x_3)(\dots, x_3 = x_2 \uparrow 2 \dots) \rightsquigarrow (\dots \wedge X_2 \neq a_4 \wedge x_2 \neq a_5 \wedge \dots)$

Beweis von Satz 2.3

Betrachte Wertebereich $\{a_1, a_2, a_3, \dots\}$ und passende RA-Ausdrücke mit RT i Teilmenge des Wertebereichs als Operanden.

Annahme: Es gibt e mit $e(RT) = RT^+$

\rightsquigarrow Für jede Relation R_l gilt $e(R_l) = R_l^+$

\rightsquigarrow (Lemma 2.1) $e \sim \{(X_1, X_2) | \Psi(X_1, X_2)\}$

³⁰Damit X_K nicht aus dem Wertebereich führt

1. Fall: Jeder konjunktive Ausdruck von Ψ hat ein Atom der Form

$$x_i = a_j, \quad x_2 = a_j, \quad x_1 = x_2 \uparrow c, \quad x_2 = x_1 \uparrow c$$

Bei genügend großen l gibt es ein Intervall $[q_1, q_2]$, in dem Indizes m und $m+d$ liegen, so dass (a_m, a_{m+d}) keinen konjunktiven Ausdruck von Ψ erfüllt, aber in R_l^+ enthalten ist! Wähle m größer als jedes vorkommende j (es gibt ein größtes festes „ j “ in Ψ) bzw d größer als jedes „ c “ \Rightarrow kein konjunktiver Term erfüllt (a_m, a_{m-d}) .

2. Fall: Es gibt einen konjunktiven Ausdruck in Ψ , dessen Atome alle von der Form

$$x_i \neq a_j, \quad x_2 \neq a_j, \quad x_1 \neq x_2 \uparrow c, \quad x_2 \neq x_1 \uparrow c$$

Falls l genügend groß, gibt es ein Paar (a_{m+d}, a_m) , das den Ausdruck erfüllt, aber nicht in R_l^+ liegt. Das führt zum Widerspruch.

Einfache Erweiterung, um Hüllenberechnung zu ermöglichen:
Erlaubte Gleichungen (vgl. Datalog-Übersetzung) der Form

$$\begin{aligned} RT &= f(RT) \\ (RT, \{A_1, \dots, A_k\}), \{A_1, \dots, A_k\} &\subseteq \alpha_0 \\ RT &\in \{RT_1, \dots, RT_m\} \end{aligned}$$

$f(RT)$: RA-Ausdruck, der bis auf RT zu σ passt, und der nicht die Differenz enthält (\rightsquigarrow monoton).

Beispiel : Sei $(S, \{A, B\})$ ein binärer Relationstyp und \circ die Komposition von Relationen mit Darstellung $R_1 \circ R_2 = (R_1 \bowtie R_2)[1, 4]$.

Hülle von S: kleinster Fixpunkt von $R = RoS \cup S$.

Berechnung (Fixpunktsatz): $f^{m_0}(\emptyset) = \bigcup_{i=1}^{\infty} S \circ \dots \circ S$ ³¹, m_0 Zahl mit $f^{m_0}(\emptyset) = f^{m_0+1}(\emptyset)$

Frage: Wann soll eine Anfragesprache als vollständig bezeichnet werden?

Oben: $q|\Xi_\delta \rightarrow R_\beta^\infty \rightsquigarrow$

³¹i-Mal

- getypt (vollständige Signatur), mögliche Verallgemeinerung: $q|_{\Xi_\delta} \rightarrow \bigcup_{B \subseteq \alpha} R_\beta^\infty$
- q Funktion \leadsto nicht möglich: "Gib Daten für einen beliebigen Angestellten."

Bsp:

```
select *
from ANG a, (select count(*) as N from ANG) as N of ANG
where cast((rand() * 1000) / N of ANG N) as integer
= (select rank() over (order by a.Name) as Rank_number
from ANG a)
```

Forderungen

1. q berechenbar (partiell rekursiv) $\leadsto R_\beta$ statt R_β^∞
2. q generisch (isomorphietreu)

zu 2. Sei mit Dom die Menge aller Wertebereichelemente bezeichnet. Sei $C \subseteq Dom$, C endlich, sei $z \in \Xi_\delta$

q heißt C-generisch, falls für jede Permutation ρ auf Dom mit $\rho(x) = x$ für alle $x \in C$ folgendes Diagramm kommutiert:

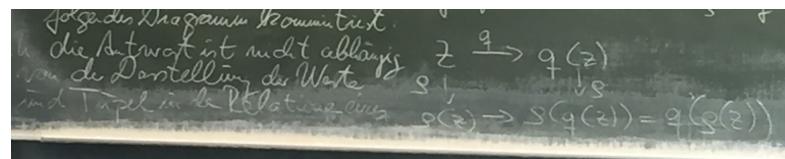


Figure 18:

Falls $C = \emptyset$ heißt q generisch.

Bedeutung von C: enthält die in q verwendeten Konstanten(symbole). $C = \emptyset$ immer möglich: Verwende "konstante" Relationen als Operanden ($\{A : c\} \leadsto$ keine Konstanten(symbole) in Anfrage notwendig).

Definition: Vollständigkeit einer Anfragesprache

Eine Anfragesprache (L_0, μ) zu einem DB-Schema σ heißt vollständig, wenn gilt:

1. Für jeden Ausdruck $e \in L_\sigma$ ist $\mu(e)$ berechenbar und generisch

Betrachte RA : 

$$R[A] = \{(1), (4)\}$$

$$s : (1\ 2\ 3), (4\ 5) \quad (\text{als Permutation})$$

$$R[A] = \{(2), (5)\}$$

$$s^{-1} : (1\ 3\ 2)(5, 4)$$

$$s^{-1} : (1), (4)$$

Figure 19:

2. Für jede berechenbare und generische Anfrage q an σ gibt es einen Ausdruck $e \in L_\sigma$ mit $\mu(e) = q$.

Definition: Abgeschlossenheit einer Anfragesprache

Eine allgemeine Anfragesprache L mit Interpretationsvorschrift μ heißt abgeschlossen, wenn sie zu jedem Datenbank-Schema σ eine vollständige Anfragesprache (L_σ, μ) enthält. Offensichtlich:

- Alle RA-Anfragen sind berechenbar und generisch
- Die RA ist nicht abgeschlossen (s. Satz 2.3)

Frage: Wie RA zu vollständiger Sprache erweitern? Problem? Effizient? (Weitere Diskussion später)

Oben: Fixpunktoperator. Reicht allein nicht.

Bsp. PERSON(Vname, Geschlecht)

Anfrage: "Ist die Anzahl weiblicher Personen gerade?"

0.0.1 Vorschlag für abgeschlossene Sprache QL

Programmiersprache (Zuweisungen, while-Konstrukt, Hintereinanderausführung von Anweisungen) mit eingeschränkter Menge von Datenbankoperationen.

- Gleichheit
- Komplement
- Durchschnitt
- Test auf Leerheit
- vereinfachte Versionen der Projektion und des kartesischen Produkts \rightsquigarrow Simulation beliebiger Turingmaschinen

Vereinfachung: ein Wertebereich Dom , Relationen $\subset \text{Dom}^i$, i Stelligkeit der Relation ($\text{rank}(R)$)

0.1 QL-Syntax:

Seien x_1, x_2, \dots Variablen, r_1, r_2, \dots Relationen

Definition: QL-Terme Die Menge der **QL-Terme** ist induktiv definiert:

1. E (Gleichheit) ist ein Term, r_i, x_i sind Terme
2. Falls e, e' Terme sind, dann $(e \cap e'), (\neg e), (e \downarrow), (e \uparrow)$ und $e \circlearrowright$

Definition: QL-Programme Die Menge der **QL-Programme** ist induktiv definiert durch

1. $x_i := e$ ist ein Programm für einen Term e und $i \geq 1$
2. Für Programme P, P' sind $(P; P')$ und **while** x_i do P Programme. Es gelten die üblichen Klammereinsparungsregeln

Beispiel

```

 $x_2 := x_1, \quad x_3 := E \downarrow\downarrow;$ 
 $\text{while } x_2 \text{ do } (P; x_2 := e; x_3 := E);$ 
 $\text{while } x_3 \text{ do } (P'; x_3 := E);$ 

```

QL-Semantik Seien $\sigma = \{(RT_1, s_1), \dots, (RT_m, s_m)\}$, $s_i \in \mathbb{N}$, $z \in \xi_\sigma$.

- R_S^\emptyset : leere Relation der Stelligkeit s
- R_S : Menge der Relationen mit Stelligkeit
- $R_{-1} = \{R_0^\emptyset\}$

$\rightsquigarrow E \in R_2$ mit $E = \{(d, d) | d \in \text{Dom}\}$

- $r_i = \begin{cases} z(RT_i) & \text{falls } i \leq m \\ R_0^\emptyset & \text{sonst} \end{cases}$
- “ \cap ” entspricht Durchschnitt, falls beide Argumente gleiche Stelligkeit haben, sonst R_0^\emptyset
- “ \neg ”: $R_S \rightarrow R_S: \neg e =_{def} Dom^s - e$
- “ \downarrow ” $R_S \rightarrow R_{s-1}: e \downarrow =_{def} \{(d_2, \dots, d_s) | (d_1, \dots, d_s) \in e\}$
- “ \uparrow ” $R_S \rightarrow R_{s+1}: e \uparrow =_{def} \{(d_2, \dots, d_s, d) | (d_1, \dots, d_s) \in e, d \in Dom\}$
- “ \circlearrowleft ” $R_S \rightarrow R_S: e \circlearrowleft =_{def} \{(d_1, \dots, d_{s-2}, d_s, d_{s-1}) | (d_1, \dots, d_s) \in e\}$

Bemerkung

$$\begin{aligned} \neg(Dom^S) &= R_S^\emptyset & e \downarrow = \{()\} & \text{falls Stelligkeit von } e \text{ gleich 1 und } e \neq R_1^\emptyset \\ \{()\} \uparrow &= E \downarrow = Dom \\ E \downarrow &= \{(d)\} \\ E \downarrow\downarrow &= \{()\} \\ E \downarrow\downarrow\downarrow &= \emptyset? \end{aligned}$$

- Alle Variablen werden mit R_0^\emptyset initialisiert
- Test von x_i in while-Anweisung ist wahr $\Diamond_{def} x_i$ ist nicht leer

Satz 2.4

QL ist abgeschlossen

Beweisidee Zeige zunächst, dass einige übliche Operationen auf Relationen ausdrückbar sind, etwa zählen:

$$0 : E \downarrow\downarrow (= \{()\})$$

Ebenso ausrückbar:

$$\begin{aligned} e_1 \times e_2 &(\text{kartesisches Produkt}) \\ e[s_1, \dots, s_p] &(\text{allgemeine Projektion}) \\ e \uparrow &(\text{Projeziere links hineind}, d_1, \dots, d_s) \end{aligned}$$

Sei e Darstellung von $i \ i+1 : e \uparrow \ i-1 : e \downarrow$

Test von e auf 0: Teste $e \downarrow$ auf Leerheit mit if + while \rightsquigarrow s.o.

Transitive Hülle $x_1 := E; \ x_2 := r_1; \ x_3 := \neg(\neg E \cap \neg x_2); \ x_4 := \neg E \cap x_2;$
 while x_4 nicht leer do $(x_1 := x_3; x_3 := x_3 \cup (((x_3 \times x_2) \cap \uparrow E \uparrow)_{[1,4]}); x_4 = \neg x_1 \cap x_3)$

Einfach: QL-Programme sind partiell rekursiv und generisch. Schwieriger: Jede berechenbare und generische Anfrage lässt sich als QL-Programm schreiben.

Andere Idee für Erweiterung: Erweiterte TRC geeignet um arithmetische Operationen für \mathbb{N} , (Gödel: In einer Sprache der Arithmetik erster Stufe lassen sich alle rekursiven Funktionen ausdrücken)

Tupelorientierter Relationenkalkül

PL1-Logike \rightarrow getypter Kalkül (DB-Schema beachten)

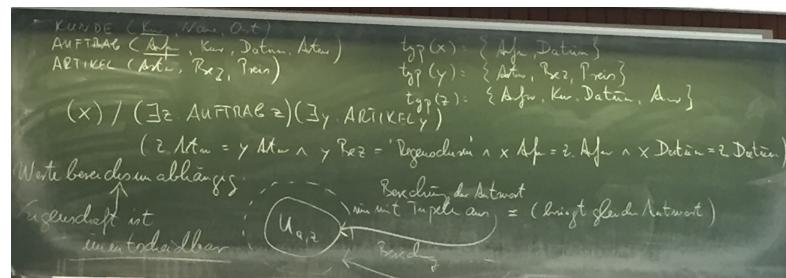


Figure 20:

Beispiel:

$$\begin{aligned}
 typ(x) &= \{A, C\}, \quad typ(y) = \{B, C\}, \quad typ(z) = \{A, B\} \\
 (x, yB) / R.x \wedge S.y \wedge (\forall z : Tz)(z.A = x.A \wedge z.B > y.B) \\
 R.x &\rightsquigarrow R[A \rightarrow x_A, C \rightarrow x_C] \\
 S.y &\rightsquigarrow S[B \rightarrow y_B, C \rightarrow y_C]^{32} \\
 (\forall z : T.z)(\dots) &\rightsquigarrow (((R \setminus R)[A][A \rightarrow z.A][Kompl] \\
 &\times \dots \times (R \setminus R)[A][A \rightarrow x_A][kompl] \times \dots \times x_c \dots)[z.A = x.A]) \\
 &\cap \dots \text{ analog } \dots \times (R \setminus R)[A][A \rightarrow x_A][kompl] \times \dots \times x_c
 \end{aligned}$$

³² $\{\emptyset\}$ entspricht nicht-leere Relation über leerer Attributmenge, d.h. enthält das total undefinierte Tupel

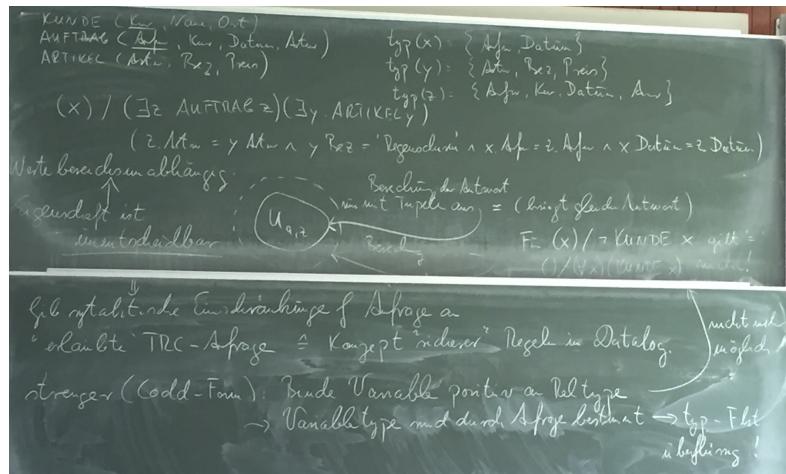


Figure 21:

Komplexität von Anfragesprachen

Zur Erinnerung: Komplexitätsklassen

Zeit

- $\text{DTIME}(n) \leq \text{DTIME}(n^2) \leq P = \text{PTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^K)$
- $\text{NTIME}(n) \leq \text{NTIME}(n^2) \leq NP = \text{NPTIME} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^K)$
- $\text{EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^{P(K)})$

Platz

- $\text{LOGSPACE} = \text{DTAPE}(\log n)^{33}$

$$\begin{aligned} \text{DTAPE}(n) &\subseteq \text{DTAPE}(n^2) \cdots \text{DPSPACE} = \bigcup_{d \in \mathbb{N}} \text{DTAPE}(n^K) \\ \text{NTAPE}(n) &\subseteq \text{NTAPE}(n^2) \cdots \text{NPSPACE} = \bigcup_{d \in \mathbb{N}} \text{NTAPE}(n^K) \end{aligned}$$

Inklusionssatz

$$\begin{aligned} \text{LOGSPACE} &\subseteq^? \text{NLOGSPACE} \subseteq^? P \subseteq^? NP \\ &\subseteq^? \text{PSPACE} \subseteq \text{NPSPACE} \subseteq \text{EXPTIME} \subseteq \text{REK} \end{aligned}$$

³³mit Speicherplatz $O(S(n))$

- P \rightsquigarrow explizite Anfrage
- PSPACE \rightsquigarrow while Anfrage

Frage Was ist die Problemgröße bei Anfragen an Datenbanken?

2 Möglichkeiten:

- Länge der Anfrage (DB fest) \rightsquigarrow "Anfragenkomplexität"
- Größe der DB (q fest) \rightsquigarrow "Datenkomplexität"

Erkennungsproblem für Anfragen Gegeben ein DB-Zustand z zu einem DB-Schema σ , Anfrage $q|\phi_\sigma \rightarrow R_\beta$ und Tupel t über β . \rightsquigarrow Stelle fest, ob $t \in q(z)$.

$$\begin{aligned} & \{code_E(z)\#code_E(t)|t \in q(z), E \text{ Aufzählung von } DOM_{q,z}\} \\ & E = \{d_1, d_2, \dots\} \end{aligned}$$

Definition: Datenkomplexität einer Anfrage

Die Datenkomplexität³⁴ einer Anfrage q ist die Komplexität ihres Erkennungsproblems. Größe des Ergebnisses (der Antwort) ist nicht erfasst. Falls gewünscht —**Antwortkomplexität**: $\{code_E(z) + code_e(q(z))|\dots\}$ Komplexität der Konstruktion des Ergebnisses erfasst.

Unterschied nicht wesentlich bei Komplexitätsklassen, die bzgl. Polynomfaktor unempfindlich sind (ab "P" aufwärts)

Aber Mit Antwortkomplexität keine Unterscheidung zwischen leichten und schwierigen Anfragen, die große Ergebnisse haben möglich

Beispiel

$$\begin{aligned} \sigma &= \{(G, \{V_1, V_2\})\}(Graph) \\ 1) &kart(G) = G[V_1] \times G[V_2] \\ 2) &pfade(G) = \{(x, y)|\text{es gibt einen Pfad zwischen } x \text{ und } y\} \\ 3) &ident(G) = G \end{aligned}$$

Betrachte Zeitkomplexität:

³⁴kurz: Komplexität

- 1) \leftrightarrow 2) Antwortkomplexität: $O(n^2)$ bei 1) und 2)
Datenkomplexität: 1) $O(n)$, 2) $O(n^2)$
- 1) \leftrightarrow 3): Antwortkomplexität 1) $O(n^2)$, 3) $O(n)$
Datenkomplexität: $O(n)$ 1) + 3)

Betrachte \bowtie Datenkomplexität: $O(n \log n)$, Antwortkomplexität: $O(n^2)$

Im Folgenden: Komplexität entspricht Datenkomplexität. Zunächst: $L \Leftrightarrow$ Komplexitätsklasse

Problem: Falls auf Wertebereichen keine Ordnung angenommen:

- Für PRTIME und Klassen darunter sind keine Sprachen bekannt, die genau einer Komplexitätsklasse entsprechen
- Anderer Zusammenhang von Interesse: Bezug auf vollständige Probleme einer Klasse

Definition: Vollständiges Problem: schweres Problem (Problem lösen entspricht Menge entscheiden).

Definition hartes Problem Ein Problem heißt hart für eine Komplexitätsklasse K unter einem Reduktionstyp, falls jedes Problem in K mit einer Reduktion des Typs auf p reduziert werden kann. Falls p in K : p ist K -vollständig

Definition: Vollständigkeit bezüglich einer Komplexitätsklasse K Eine Anfragesprache L_σ zu einem DBSchema σ ist vollständig bezüglich einer Komplexitätsklasse, falls gilt: Sei $Q_\sigma K$ die Menge aller Anfragen an σ mit Komplexität K :

1. $\{\mu(e) | e \in L_\sigma\} \subseteq Q_\sigma K$
2. $(\exists e \in L_\sigma)(\text{das Erkennungsproblem für } e \text{ ist vollständig bezüglich } K)$

L heißt vollständig bezüglich K , falls L_σ für jedes σ vollständig ist bezüglich K . Wichtig in diesem Zusammenhang: Komplexität der Reduktion.

z.B. Vollständigkeit bei logarithmischen Platz bezüglich K bedeutet: Jede Anfrage kann mit logarithmischen Platzaufwand auf die vollständige Anfrage reduziert werden. Reduktion muss nicht in L ausdrückbar sein. Es kann einfache Anfragen in QK geben, die nicht in L ausdrückbar sind (obwohl einige vollständige Anfrage in QK in L ausdrückbar sind)

DB Menge undifferenzierter Elemente generisch \rightsquigarrow einheitliche Behandlung in Anfrage. Wie ist “erstes Element” spezifiziert?

Turingmaschine (TM)

Kodiert die Eingabe auf Band (“Hinschreiben”) liefert zusätzliche Information, die Lösung trivial macht (erstes Element steht links) (lineare Zeit) \rightsquigarrow andere “Rechengeräte” erforderlich (keine Ordnung auf Band) \rightsquigarrow Forschung “generisch” kann Aufgaben schwierig machen.

PSPACE - vollständige Probleme

:

- QSAT $\Phi(x_1, \dots, x_n)$ in konjunktiver Normalform: $(\exists x_1)(\forall x_2)(\exists x_3) \dots : \Phi(x_1, \dots, x_n)$ ist wahr?
- GO
- GEOGRAPHY (einzigartige Stadtnamenfolge mit letzter Buchstabe gleich erster Buchstabe des nächsten)

Komplexität von Kalkülanfragen (RA)

Annahme: TM hat lesebeschränktes Eingabeband und Arbeitsband, Kodierung des DB-Zustands und der Anfrage auf Eingabeband sonst bedingt Kodierung des Zustandes, dass keine Komplexität unter “linear” möglich ist.

Beispiel für Kodierung $code_E$ Nimm alle Elemente $U \subseteq DOM$ als durch Aufzählung E gegeben an: $E = \{c_1, c_2, \dots\}$
 $code_E|c_i \rightarrow$ Binärdarstellung von i $\rightsquigarrow |code_E(c_i)| \leq \lceil \log(i) \rceil$ für jedes i

Kodierung

- Tupel: $t = (a_1, \dots, a_k) \rightsquigarrow^{code_E(t)} [code_E(a_1)\#code_E(a_2)\#\dots\#code_E(a_k)]$
- Relation: $R_i \rightsquigarrow^{code_E(R_i)} R_i code_E(t_1) \dots code_E(t_n)$ in durch $code_E$ gegebener lexikographischer Ordnung
- Zustand $z \rightsquigarrow^{code_E(z)} code_E(z(R_1))) \dots code_E(z(R_m)))$

The image shows handwritten text on a chalkboard. At the top left, there is a question mark followed by $R(A, B)$. Below it, there are two rows of letters: $a b$ and $b a$. To the right, there is $S(C, D)$ with two rows of letters: $c c$ and $c c$. Below this, there is a query: $\rightsquigarrow R[0\#1][1\#0]S[10\#10]$.

Figure 22:

Satz 2.9

Kalkülanfragen (RA-Anfragen) sind in QLOGSPACE

Beweis Sei q eine DRC-Anfrage an ein DB-Schema σ mit Antworttyp β konstruierte TM T_q , die das Erkennungsproblem für q löst, wobei die Länge des Arbeitsbandes logarithmisch durch die Länge des Eingabebandes beschränkt ist.

Annahme: T_q startet mit $code_E(z)\#code_E(t)$ auf Eingabeband, $z \in \phi\sigma, t \in DOM_z^\beta$ (mit Konstanten aus z bildbare Tupel über β). E Aufzählungsfunktion für DOM_Z , q in Pränexnormalform mit Typ β . Zeige durch Induktion über die Anzahl der Quantoren von q . Berechnung kann mit $K \cdot |code_E(z)\#code_E(t)|$, K Konstatne, Felder des Arbeitsbandes erfolge.

Beispiel

Induktionsannahme Jede Anfrage in PNF mit weniger als n Quantoren kann in LOGSPACE abgewertet werden.

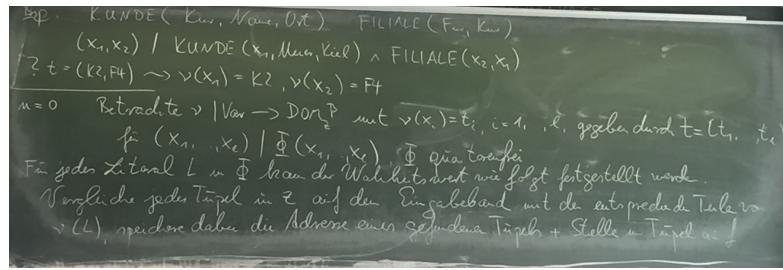


Figure 23:

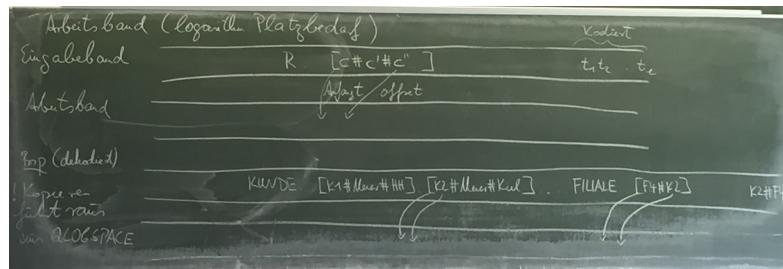


Figure 24:

Induktionsschluss $n - 1 \rightarrow n$: ϕ habe n Quantoren und sei in der Form $(\exists x)(\Phi(x))((\forall x)(\Phi'(x)))$

Idee Belege x mit den Konstanten auf dem Eingabeband in der Reihenfolge ihres Vorkommens (beschränkte Interpretation) für Merken einer solchen Konstante notwendig: $\log(DOM_z)$ Felder:

$$\log(|DOM_z|) \leq \log(|code_E(z)\#code_E(z)\#code_E(t)|)$$

Nach Induktionsannahme: Φ' kann mit $K \cdot \log(|code_E(z)\#code_E(t)|)$ Feldern ausgewertet werden \leadsto vollständige Berechnung kann mit $(K+1) \cdot \log(|code_E(z)\#code_E(t)|)$ Feldern durchgeführt werden.

Even in QLOGSPACE, aber nicht in DRC ausdrückbar \Rightarrow Welche Anfragesprache entspricht genau QLOGSPACE?

Problem Ohne gegebene Ordnung auf Wertebereich (bzgl. eine Nachfolgerrelation) ist bis jetzt keine solche Sprache bekannt. Dies gilt für alle Klassen innerhalb von P, P eingeschlossen. In Klassen oberhalb von NP: Ordnung kein Problem, da

in Sprache selbst ausdrückbar \rightsquigarrow Annahme: Ordnung auf DOM_z , z Zustand.

Ordnung gegeben durch Relation NF(Vor, Nach), Nachfolgerrelation. DB-Schema entsprechend ergänzt $\Rightarrow DOM_Z \subseteq \mathbb{N}$ annehmbar mit 0 als minimales Element und m als maximales Element. Betrachte Operator trans (transitive Hülle): Sei $\Phi(\vec{x}, \vec{y})$ ein DRC-Ausdruck, der eine Binärrelation auf K-Tupeln repräsentiert. $\Phi[Trans]$ bezeichne die reflexive, transitive Hülle von Φ . Sei DRC^{pos_trans} die Anfragesprache die aus DRC entsteht, indem der Operator “trans” in nicht negierten Teilausdrücken erlaubt wird.

Satz 2.10

$$\text{QNLOGSPACE} = DRC^{pos_trans}$$

Beweisidee “ \supseteq ” einfach: Falls $\Phi(\vec{x}, \vec{y})$ in QNLOGSPACE, dann auch $\Phi(\vec{x}, \vec{y})[trans]$. Rate ein $\Phi(\vec{x}, \vec{y})$ -Pfad.

“ \subseteq ” Ausdruck in DRC^{pos_trans} gesucht, der NLOGSPACE-TM M enstpricht.
2 wesentliche Ideen:

- Zustand von M kann mit logarithmischen Aufwand kodiert werden (mit endl. vielen Variablen)
- Mit trans-Operator können ausgedrückt werden:
 - Addition “x+y=z”
 - Test eines bestimmten Bits in einem Wort auf “1” (on(w,j))

\rightsquigarrow Feststellbar was Kopf von M sieht
 \rightsquigarrow Durch DRC-Ausdruck darstellbar: NEXT(z_1, z_2) z_n Zustandsbeschreibungen von M, z_2 folgt aus z_1 mit einem Übergang
 \rightsquigarrow PATH(z_1, z_2) darstellbar mit einer oder mehreren Anwendungen von trans
 \rightsquigarrow PATH(z_0, z_j), z_0 Anfangs- & z_j Endzustand

Für QLOGSPACE: deterministischer trans-Operator genügt.

$$dtrans : \Phi_d(\vec{x}, \vec{y}) =_{def} \Phi(\vec{x}, \vec{y}) \wedge (\forall \vec{z})(\neg \Phi(\vec{x}, \vec{z}) \vee \vec{y} = \vec{z})$$

Bemerkung Pos kann entfallen, da DRC^{pos_trans} unter Negation abgeschlossen ist (gilt nur bei Vorhandensein einer Ordnung!)



Figure 25:

Zusammenhang von Sprachklassen innerhalb von DRC

Betrachte Anfragen der Form

$(x_1, \dots, x_l) | (\exists y_1) \dots (\exists y_k)(\Phi(x_1, \dots, x_l, y_1, \dots, y_k))$, Φ quantorenfrei
 Q_{\exists} umfasst speziell die konjunktiven Anfragen:

$$(x_1, \dots, x_l) | (\exists y_1) \dots (\exists y_k) \left(\bigwedge_j R_j(u_{j1}, u_{jn_j}) \right)^{35}$$

Klasse der konjunktiven Anfragen ist Äquivalent zur Klasse der

- Datalog-Regeln ohne Rekursion: $R(x_1, \dots, x_l) : -R_1(u_{11}, \dots, u_{1n_1}), \dots R_m(u_{m1}, \dots, u_{mn_m})$
- (einfache) Tableau-Anfrage (vgl. QbE mit Einschränkung)
- RA-Anfragen mit positiver Selektion, Projektion (ohne Wiederholung), Join und Umbenennung
- RA-Anfragen mit positiver Selektion, Projektion, Kreuzprodukt

Für Äquivalenz mit RA-Anfrage gefordert: "Erfüllbarkeit, d.h. Anfragen dürfen nicht äquivalent sein zu"

$$q^\emptyset | \phi_\rho \rightarrow R_\beta \text{ mit } q^\emptyset(z) = \emptyset \text{ für alle } z \in \phi_\rho$$

Erweiterung von Kalkülsprachen und Relationenalgebra

a) **DRC** Füge Fixpunktoperator hinzu (s.o.)

Fixpunktoperator partiell, d.h. nicht immer definiert, z.B.: $\Phi(R) = (x = 0 \wedge \neg R(0) \wedge \neg R(1)) \vee (x = 0 \wedge R(1)) \vee (x = 1 \wedge R(0))$

- Inflationäre Semantik $\leadsto DRC^{lfp+}$

³⁵ u_{jn} Konstante oder Variable aus $\{x_1, \dots, y_k\}$

Definition: Sei $\sigma' = \sigma \cup \{(S, \alpha_S), (S, \alpha_S) \notin \sigma\}$.

$q(S) = (x_1, \dots, x_{|\alpha_s|}) | \Phi(x_1, \dots, x_{|\alpha_s|})$ Anfrage zu σ' und $z \in \delta_\rho$

$lfp+(q(S))$ bezeichnet den Grenzwert der Folge A_0, A_1, \dots mit $A_0 = \emptyset_{\alpha_S}, A_i = A_{i-1} \cup \mu(q(S))(Z'), Z'(RT_i) = z(RT_i), (RT_i, \alpha_i) \in \sigma, Z'(S) = A_{i-1}$ Offensichtlich: In PTIME, da monoton und jeden Zwischenergebnis Beschränkt durch $\sum_{i=1}^m |\text{Dom}|^{|\alpha_i|}$ beschränkt, m Anzahl der Relationstypen in σ .

- Nicht-inflationäre Semantik $\sim DRC^{lfp}$

Definition Wie oben bei lfp+ mit Unterschied $A_0 = \emptyset_{\alpha_j}, A_i = \mu(q(s))(z')$ Offensichtlich In PSPACE, in PTIME kann nicht gefolgert werden.

b) RA Füge while-Anweisung hinzu (Schachtelungen erlaubt) vgl. QL.
Auch hier

- Inflationär: Erzeuge Monotonie durch While Anweisungen der Form