



Informationsintegration

Query Containment

Ulf Leser

Inhalt dieser Vorlesung

- Query Containment
- Containment Mappings
- Depth-First Algorithmus
- Frozen Facts Algorithmus
- Containment bei Disjunktion, Negation, ...

Query Containment

- Definition

Sei S ein Datenbankschema, I eine Instanz von S und q_1, q_2 Anfragen gegen S . Sei $q(I)$ das Ergebnis einer Anfrage q angewandt auf I . Dann ist

q_1 enthalten in q_2 , geschrieben $q_1 \subseteq q_2$

gdw.

$q_1(I) \subseteq q_2(I)$ für alle möglichen I

- Bemerkung

- Wir können nicht alle Instanzen ausprobieren
- Statt dessen analysieren wir die Anfrage selber
- **Untersuchung der Syntax**, um eine **semantische Aussage** zu treffen

Weitere Beispiele

$q_1(B, D) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D), \text{path}(D, E)$
 $q_2(A, C) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D)$
 $q_1 \subseteq q_2$



$q_1(C, B) :- \text{path}(A, B), \text{path}(C, A), \text{path}(B, C), \text{path}(A, D)$
 $q_2(X, Z) :- \text{path}(X, Y), \text{path}(Y, Z)$
 $q_1 \subseteq q_2$



$q_1(A, C) :- \text{path}(A, B), \text{path}(B, C)$
 $q_2(A, C) :- \text{threeNodePath}(A, B, C)$
 $q_1 \subseteq q_2$



$q_1(B, D) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D), \text{path}(A, E), \text{path}(E, D)$
 $q_2(A, C) :- \text{path}(A, C), \text{path}(C, E), \text{path}(E, A), \text{path}(A, B), \text{path}(D, B)$
 $q_1 \subseteq q_2$



- Definition

- Ein *Symbol Mapping* h von einer Anfrage q_2 in eine Anfrage q_1 ist eine Funktion $h: \text{sym}(q_2) \mapsto \text{sym}(q_1)$
- Für ein Literal $l \in q_2$, $l = \text{rel}(A_1, \dots, A_m)$ ist $h(l)$ definiert als
$$h(l) := \text{rel}(h(A_1), \dots, h(A_m))$$

- Bemerkung

- Ein Symbol Mapping ist eine *Funktion*, bildet also jedes Symbol aus q_2 auf exakt ein Symbol aus q_1 ab

Containment Mappings

- Definition

*Ein **Containment Mapping** (CM) h von Anfrage q_2 nach Anfrage q_1 ist ein Symbol Mapping von q_2 nach q_1 für das gilt:*

1. $\forall c \in \text{const}(q_2)$ gilt: $h(c) = c$
 - Jede Konstante in q_2 wird auf dieselbe Konstante in q_1 abgebildet
2. $\forall l \in q_2$ gilt: $h(l) \in q_1$
 - Jedes Literal in q_2 wird auf (mindestens) ein Literal in q_1 abgebildet
3. $\forall e \in \text{exp}(q_2)$ gilt: $h(e) \in \text{exp}(q_1)$
 - Der Kopf von q_2 wird auf den Kopf von q_1 abgebildet
4. $\text{cond}(q_1) \rightarrow \text{cond}(h(q_2))$
 - Die Bedingungen von q_1 sind logisch restriktiver als die von q_2

Vom CM zum Query Containment

- *Theorem*
 - $q_1 \subseteq q_2$ gdw. es ein CM von q_2 nach q_1 gibt
- *Lemma*
 - $q_1 \equiv q_2$ gdw. es ein CM von q_2 nach q_1 und ein CM von q_1 nach q_2 gibt
- Beweis
 - Nicht schwierig; über die Semantik von Anfragen [CM77]
- Ursprünglich zur **Anfrageminimierung** entwickelt
- Richtung beachten

Beispiel

$$q_1 \subseteq q_2 \quad ?$$

$q_2(A, C) \text{ :- path}(A, B), \text{ path}(B, C), \text{ path}(C, D)$

$q_1(B, D) \text{ :- path}(A, B), \text{ path}(B, C), \text{ path}(C, D), \text{ path}(D, E)$

Mapping: $A \rightarrow A, B \rightarrow B, C \rightarrow C, D \rightarrow D$

$q_2(A, C) \text{ :- path}(A, B), \text{ path}(B, C), \text{ path}(C, D)$

$q_1(B, D) \text{ :- path}(A, B), \text{ path}(B, C), \text{ path}(C, D), \text{ path}(D, E)$

Mapping: $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$

Beispiel

$q_2(T, K) \text{ :- film}(T, Y, R, L), \text{spielt}(T, N, O, K), L > 60, L < 90$
$q_1(A, G) \text{ :- film}(A, B, C, D), \text{spielt}(A, E, F, G), D < 50$

Diagram showing variable mapping from q_2 to q_1 with blue arrows:

- $T \rightarrow A$
- $K \rightarrow G$
- $Y \rightarrow B$
- $R \rightarrow C$
- $L \rightarrow D$
- $N \rightarrow A$
- $O \rightarrow E$
- $K \rightarrow F$
- $K \rightarrow G$

CM: $h(T) \rightarrow A, h(Y) \rightarrow B, h(R) \rightarrow C, \dots$
 $h(L) = D$

Aber: $D < 50 \nrightarrow h(L) > 60 \wedge h(L) < 90$

$q_1 \not\sqsubseteq q_2$

Intuition

- Containment Mapping von q_2 nach q_1 ...
 1. $\forall c \in \text{const}(q_2)$ gilt: $h(c) = c$;
 - Konstante dürfen sich nicht ändern (gleiche Selektionsbedingungen)
 2. $\forall l \in q_2$ gilt: $h(l) \in q_1$
 - **Zusätzliche Literale** in q_1 sind nur Filter auf dem Ergebnis; Containment wird dadurch nicht beeinflusst
 3. $\forall e \in \text{exp}(q_2)$ gilt: $h(e) \in \text{exp}(q_1)$
 - Es müssen auch die **richtigen Variablen** ausgegeben werden; q_1 darf weitere Variable exportieren, aber nicht weniger
 4. $\text{cond}(q_1) \rightarrow \text{cond}(h(q_2))$
 - Bedingungen in q_1 müssen **äquivalent oder strikter** sein (also höchstens Tupel wegfiltern) als die von q_2

Beispiel wiederholt

```
SELECT titel, typ, rolle, kritik
FROM   film, spielt
WHERE  film.titel = spielt.titel;
```

- Globale Anfrage: $q(T, Y, O, K) :- \text{film}(T, Y, _, _), \text{spielt}(T, _, O, K)$
- Was ist mit `filmkritiken` ?

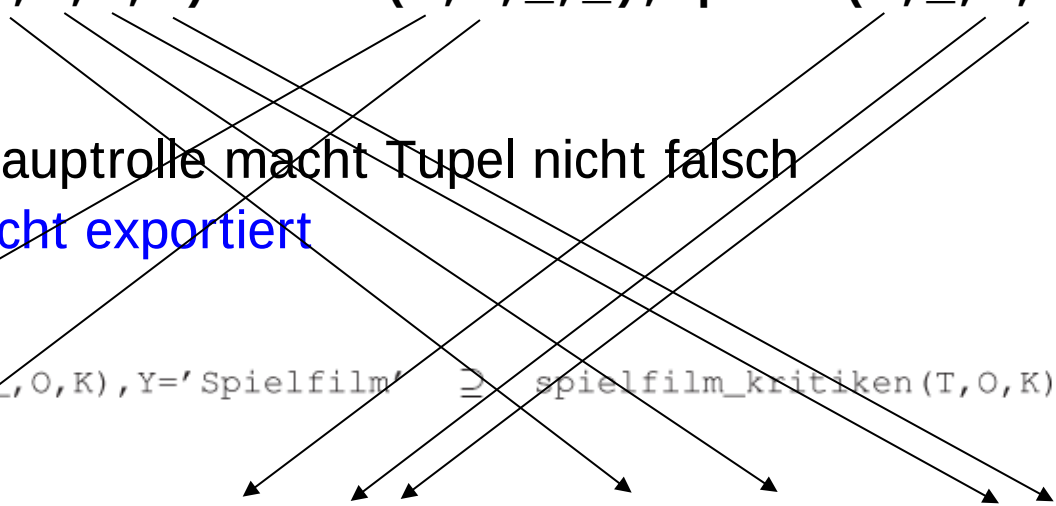
<code>film(T, Y, R, L), L > 79, Y = 'Spielfilm'</code>	$\supseteq$	<code>spielfilme(T, R, L)</code>
<code>film(T, Y, R, L), L < 11, Y = 'Kurzfilm'</code>	$\supseteq$	<code>kurzfilme(T, R)</code>
<code>film(T, _, R, _), spielt(T, S, O, K), O = 'Hauptrolle'</code>	$\supseteq$	<code>filmkritiken(T, R, S, K)</code>
<code>film(T, Y, _, L), spielt(T, S, _, _),</code> <code>schauspieler(S, N), N = 'US', Y = 'Spielfilm'</code>	$\supseteq$	<code>us_spielfilm(T, L, S)</code>
<code>film(T, Y, _, _), spielt(T, _, O, K), Y = 'Spielfilm'</code>	$\supseteq$	<code>spielfilm_kritiken(T, O, K)</code>
<code>film(T, Y, _, _), spielt(T, S, O, _),</code> <code>schauspieler(S, N), Y = 'Kurzfilm'</code>	$\supseteq$	<code>kurzfilm_rollen(T, O, S, N)</code>

Beispiel wiederholt

- Globale Anfrage: $q(T, Y, O, K) :- \text{film}(T, Y, _, _), \text{spielt}(T, _, O, K)$
- Was ist mit `filmkritiken` ?

$\text{film}(T, Y, R, L), L > 79, Y = \text{'Spielfilm'}$ $\supseteq$ $\text{spielfilme}(T, R, L)$
 $\text{film}(T, Y, R, L), L < 11, Y = \text{'Kurzfilm'}$ $\supseteq$ $\text{kurzfilme}(T, R)$?
 $\text{film}(T, _, R, _), \text{spielt}(T, S, O, K), O = \text{'Hauptrolle'}$ $\supseteq$ $\text{filmkritiken}(T, R, S, K)$
 $\text{film}(T, Y, _, L), \text{spielt}(T, S, _, _),$
 $\text{schauspieler}(S, N), N = \text{'US'}, Y = \text{'Spielfilm'}$ $\supseteq$ $\text{us_spielfilm}(T, L, S)$
 $\text{film}(T, Y, _, _), \text{spielt}(T, _, O, K), Y = \text{'Spielfilm'}$ $\supseteq$ $\text{spielfilm_kritiken}(T, O, K)$
 $\text{film}(T, Y, _, _), \text{spielt}(T, S, O, _),$
 $\text{schauspieler}(S, N), Y = \text{'Kurzfilm'}$ $\supseteq$ $\text{kurzfilm_rollen}(T, O, S, N)$

Beispiel wiederholt

- Globale Anfrage: $q(T, Y, O, K) :- \text{film}(T, Y, _, _), \text{spielt}(T, _, O, K)$
 - **filmkritiken**
 - Einschränkung auf Hauptrolle macht Tupel nicht falsch
 - Aber der **Typ wird nicht exportiert**
 - **spielfilm_kritiken**
 $_ \text{film}(T, Y, _, _), \text{spielt}(T, _, O, K), Y = \text{'Spielfilm'} \supseteq \text{spielfilm_kritiken}(T, O, K)$
 - Umschreiben zu
 $\text{film}(T, \text{'Spielfilm'}, _, _), \text{spielt}(T, _, O, K) \supseteq \text{sf_k}(T, \text{'Spielfilm'}, O, K)$
- 

Set-Semantik

- Alles gesagte gilt nur unter **Set-Semantik**
- Beispiel
 - Sei $q_1(X, Y) : -r(X, Y)$; $q_2(X, Y) : -r(X, Y), s(Y, Z)$;
 - Also: $q_2 \subseteq q_1$
 - Aber wenn: $q_2(X, Y) : -r(X, Y), s(Z, Z)$
 - Im Ergebnis ist jedes Tupel (X, Y) aus r **so oft** enthalten, wie (Z, Z) in s enthalten ist
 - Unter Set-Semantik ist das egal
- **Bag-Semantik**
 - Anfragen sind nur dann äquivalent, wenn sie isomorph sind
 - Containment bei Anfragen mit Ungleichheit ist unentscheidbar (PODS2006)

Inhalt dieser Vorlesung

- Query Containment
- Containment Mappings
- Depth-First Algorithmus
- Frozen Facts Algorithmus
- Containment bei Disjunktion, Negation, ...

Wie findet man CMs von q nach v?

- Naive: Jedes mögliche Symbol Mapping s testen
 - Sei $m = |\text{sym}(v)|$, $n = |\text{sym}(q)| \Rightarrow m^n$ Symbol Mappings
- Aber: **Literale müssen auf Literale** abgebildet werden
 - Also müssen alle Symbole jedes Literals in q auf die Symbole **eines Literals** in v der gleichen Relation abgebildet werden
- Vorgehen
 - Wir zählen mögliche Ziele für Literale auf
 - Dabei können wir gleich Bedingung 1 (und 3) testen
 - Übrig bleibt der Test, ob Abbildungen von Literalen **kompatibel** sind
 - Die Abbildung muss eine Funktion bleiben
 - Berechnung **wachsender Containment Mappings**

Suchraum

$v \subseteq q$?

$q = a(\dots), b(\dots), b(\dots), c(\dots)$ $v = b(\dots), c(\dots), a(\dots), b(\dots), d(\dots)$
--

Nummerieren

$q = a(\dots), b^1(\dots), b^2(\dots), c(\dots)$ $v = b^1(\dots), c(\dots), a(\dots), b^2(\dots), d(\dots)$
--

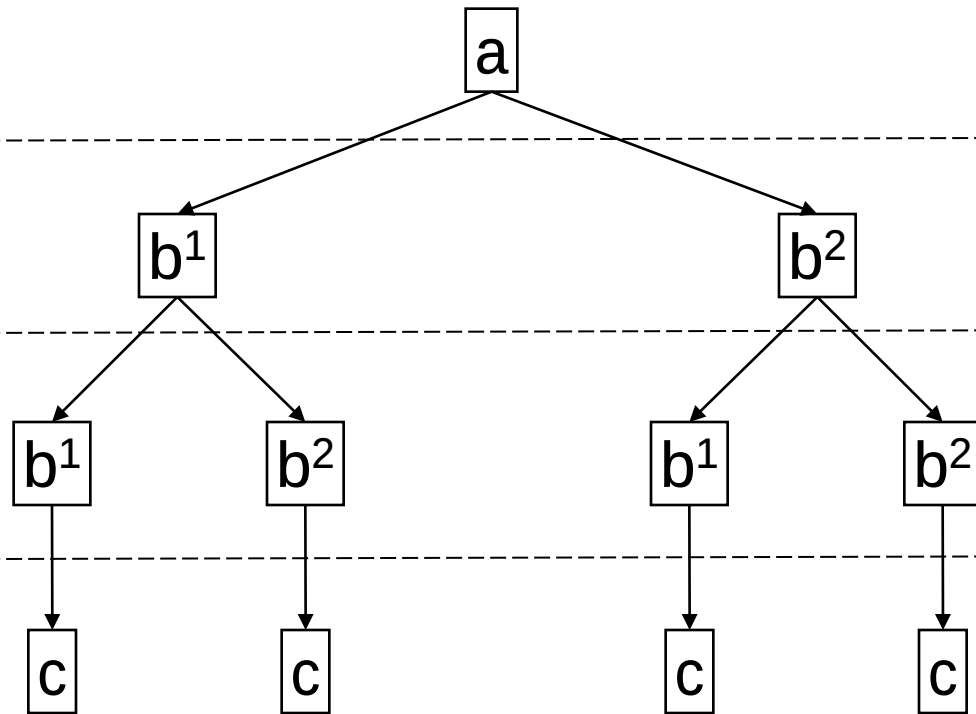
... jedes Literal von q muss auf mindestens ein Literal in v abgebildet werden ...

Möglichkeiten:

$a \rightarrow a$ $b^1 \rightarrow b^1$ $b^1 \rightarrow b^2$ $b^2 \rightarrow b^1$ $b^2 \rightarrow b^2$ $c \rightarrow c$
--

Suchbaum

q =
a(. . .),
b¹(. . .),
b²(. . .),
c(. . .)



Partielle CM

- Definition

*Seien q und v Anfragen mit Literalen $k \in q$ und $l \in v$. Ein **partielles Containment Mapping** h_{kl} von q nach v für l und k ist ein Symbol Mapping (von q nach v), für das gilt*

- 1. $\forall c \in \text{const}(k)$ gilt: $h_{kl}(c) = c$*
- 2. $h_{kl}(k) = l$*
- 3. $\forall e \in \text{exp}(k)$ gilt: $h_{kl}(e) \in \text{exp}(l)$*

- Bemerkung

- Bedingung 4 für CM verschieben wir auf später
- Der Test ist für feste l und k offensichtlich einfach
- Überträgt sich in natürlicher Weise auf **Gruppen aus mehreren Literalen**

Kompatibilität und Vereinigung

- Definition

Kompatibilität von partiellen CM

Gegeben seien zwei Anfragen p und q mit $l, i \in p$ und $j, k \in q$ und $l \neq i$ und $j \neq k$. Seien $h_{l,j}$ und $h_{i,k}$ zwei partielle Containment-Mappings. Man nennt $h_{l,j}$ und $h_{i,k}$ kompatibel, wenn $\nexists v : v \in h_{l,j} \wedge v \in h_{i,k} \wedge h_{l,j}(v) \neq h_{i,k}(v)$. ■

- Definition

Vereinigung kompatibler partieller CM: $\mathbf{h} = \mathbf{h}_1 \oplus \mathbf{h}_2$

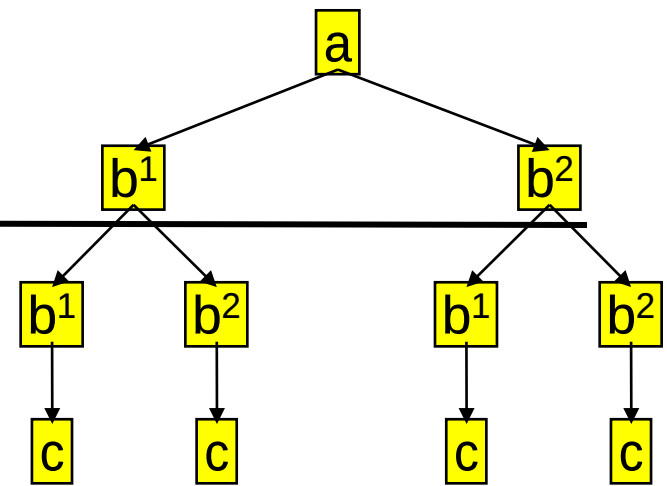
Gegeben seien zwei kompatible Teilmappings $h_{l,j}$ und $h_{i,k}$. Das kombinierte Teilmapping $h = h_{l,j} \circ h_{i,k}$ ist definiert als:

$$\square \quad \forall v \in \text{sym}(h_{l,j}) : h(v) := h_{l,j}(v)$$

$$\square \quad \forall v \in \text{sym}(h_{i,k}) \setminus \text{sym}(h_{l,j}) : h(v) := h_{i,k}(v)$$
 ■

Das ist eindeutig, wenn CM kompatibel sind

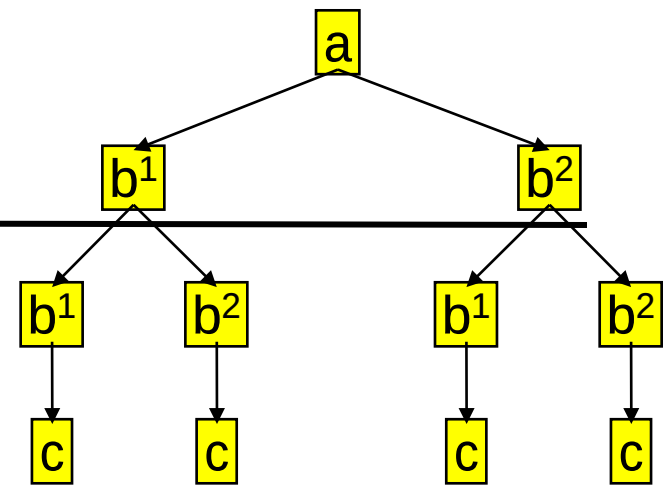
Algorithmus Grobablauf



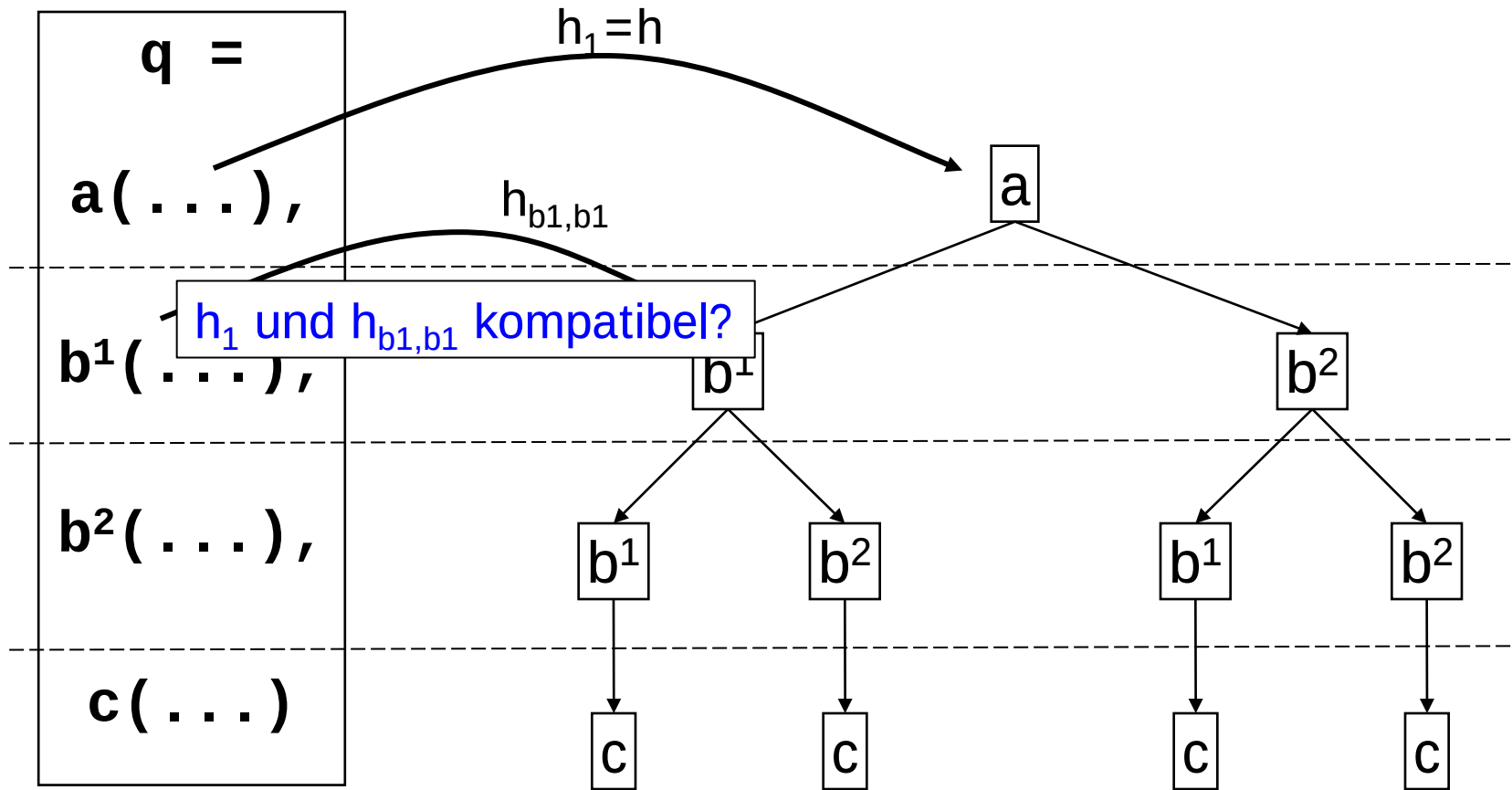
- Im Suchbaum entspricht der Level i dem Literal l_i von q
- Alle Zielliterale von l_i stehen in den Knoten des Levels i des Suchbaums
- h_i ist ein (wachsendes) partielles CM für die ersten i Literale von q
- Algorithmus durchläuft den Suchbaum Depth-First und versucht, ein wachsendes CM aufzubauen

Algorithmus Grobablauf

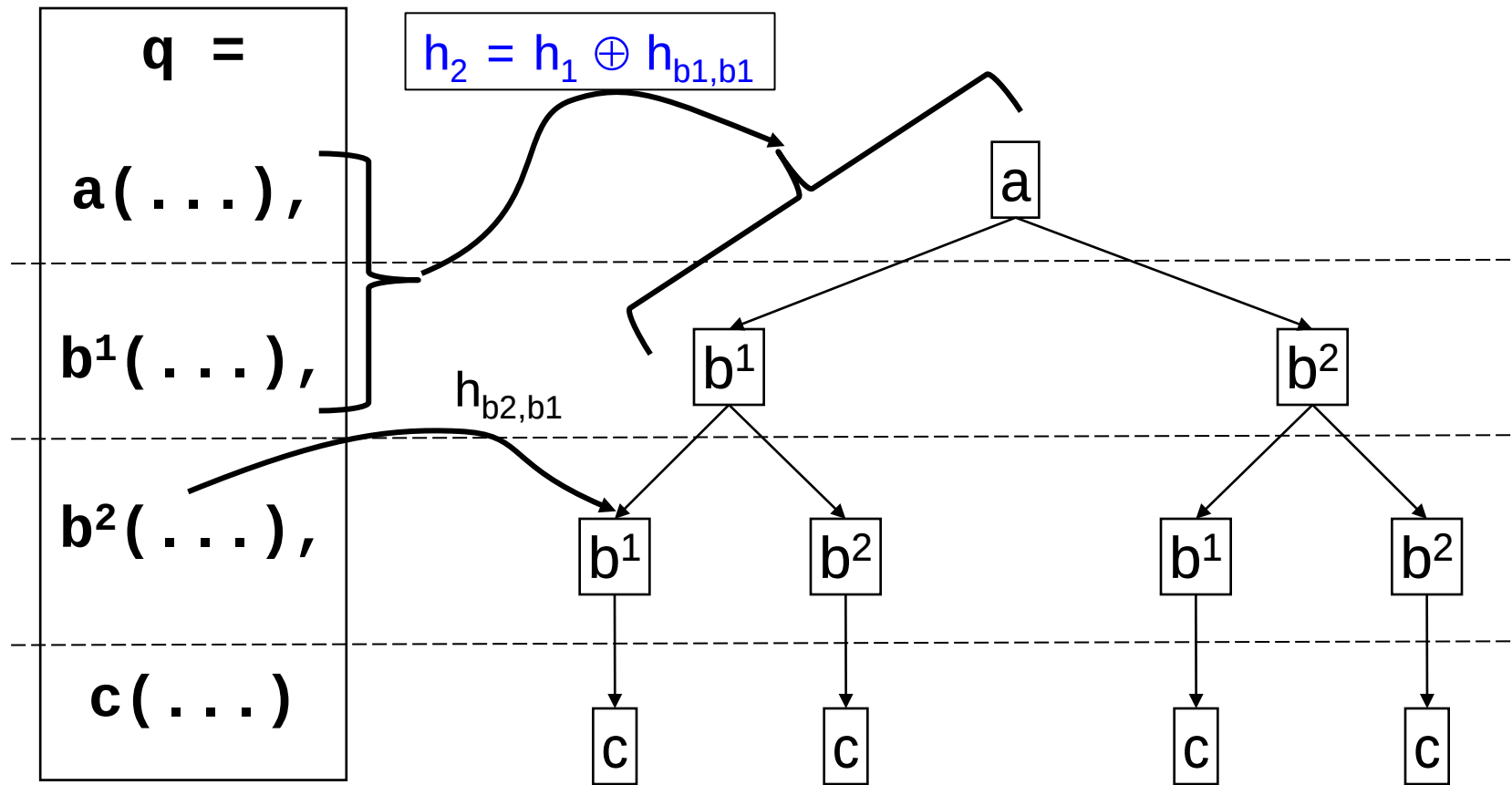
- Starte mit einem leeren CM h_0
- Für jedes Zielliterals auf Level i wird versucht, das h_i zu berechnen
- h_i wird berechnet als Vereinigung des bisherigen h_{i-1} und des partiellen Containment Mappings h von l_i zum gewählten Zielliteral
 - Wenn h und h_{i-1} kompatibel sind
- Wenn man Level $|q|=n$ erreicht hat und h_n berechnen konnte, hat man ein CM von q nach v gefunden und damit $v \subseteq q$ bewiesen

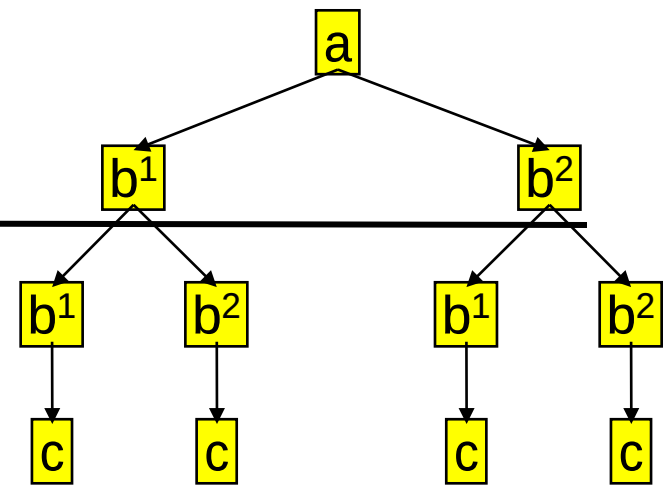


Erstes Literal



Zweites Literal, drittes ...





- Positionen im Suchbaum wird in einem Stack gespeichert
- Stackelemente sind Tripel: (X,Y,Z)
 - X: CM bis zu dieser Position
 - $=h_{i-1}$
 - Y: Weitere Mappingkandidaten im aktuellen Zweig
 - noch nicht untersuchte Geschwister von t
 - Z: Literale aus q, die noch nicht gemapped sind
 - Noch nicht besuchte Level des Suchbaums

Depth-First Algorithmus

Input: Two queries q, v ;

Output: True, if $v \subseteq q$, else false;

```
1:      R = new stack();
2:      L = {l | l ∈ q};           % Alle Literale in q
3:      l = L.first();            % Erstes Literal von q
4:      H = {h | ∃ l' ∈ v ∧ l →h l'} % Alle pot. Ziele von l in v
5:      R.push([], H, L \ l);      % Startpunkt
6:      repeat
7:          (h, H, L) = R.pop();    % Hier weitermachen
8:          if H ≠ ∅ then           % Kandidaten untersuchen
11:             g = H.first();
12:             R.push(h, H \ g, L); % Geschwister später untersuchen
13:             if compatible(h, g) then % CM kann erweitert werden
14:                 if L = ∅ then
15:                     return true; % Blatt erreicht - fertig
16:                 else
17:                     l = L.first(); % Innerer Knoten, Kinder pushen
18:                     H' = {h | ∃ l' ∈ v ∧ l →h l'};
19:                     R.push(h ⊕ g, H', L \ l);
20:                 end if;
21:             end if;
22:         end if;
23:     until R = ∅;                 % Kandidaten sind ausgegangen
24:     return false;
```

Erklärung

- Sei h_{i-1} das Mapping bis Level $i-1$ und l das i -te Literal aus q
- Berechne die Menge H aller **potentiellen Zielliterale** für l
- Teste für alle $t \in H$
 - Bestimme h , das partielle CM von l nach t
 - Wenn es das nicht gibt: springe zu 1;
 - Wenn h und h_{i-1} kompatibel
 - Ist l das letzte Literal aus q ? Ja, terminiere und melde Erfolg
 - Sonst: $h_i = h_{i-1} \oplus h$
 - Untersuche nächstes Literal von q
 - Sonst (h und h_i nicht kompatibel)
 - Sind noch Geschwister von t vorhanden?
 - Ja – testen
 - Nein – terminiere und melde Misserfolg

Beispiel

$$\begin{aligned} q(A, B) &= a(A, C), b(C, B), c(B, A) \\ v(X, X) &= a(X, Y), b^1(Y, Z), b^2(Y, X), c^1(Z, X), c^2(X, X) \end{aligned}$$

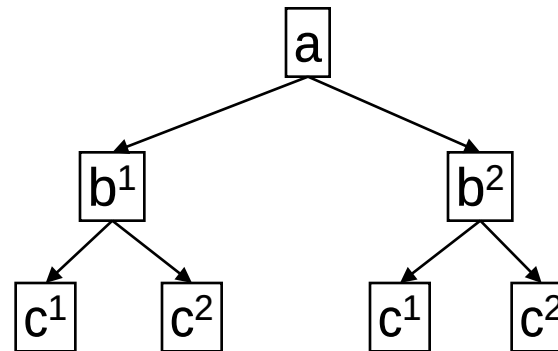
Aktuelles CM	Mapping der Literale von q auf Zielliterale in v
$A \rightarrow X, C \rightarrow Y$	$a(A, C) \rightarrow a(X, Y)$
$A \rightarrow X, C \rightarrow Y, C \rightarrow Y, B \rightarrow Z$	$a(A, C), b(C, B) \rightarrow a(X, Y), b^1(Y, Z)$
$A \rightarrow X, C \rightarrow Y, C \rightarrow Y, B \rightarrow X$	$a(A, C), b(C, B) \rightarrow a(X, Y), b^2(Y, X)$
$A \rightarrow X, C \rightarrow Y, B \rightarrow X, B \rightarrow Z, A \rightarrow X$	$a(A, C), b(C, B), c(B, A) \rightarrow a(X, Y), b^2(Y, X), c^1(Z, X)$
$A \rightarrow X, C \rightarrow Y, B \rightarrow X, B \rightarrow X, A \rightarrow X$	$a(A, C), b(C, B), c(B, A) \rightarrow a(X, Y), b^2(Y, X), c^2(X, X)$

Z nicht exportiert,
backtracking

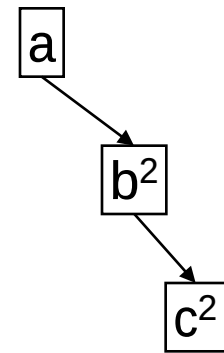
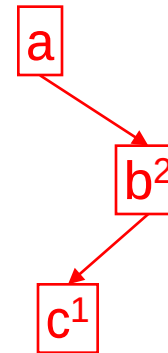
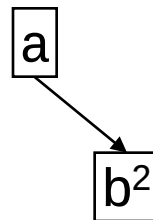
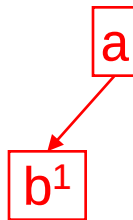
$B \rightarrow X$ / $B \rightarrow Z$ inkomp.,
backtracking

Fertig

Beispielbaum



a



$a(A, C) \rightarrow$
 $a(X, Y)$

$a(A, C), b(C, B) \rightarrow$
 $a(X, Y), b_2(Y, X)$

$a(A, C), b(C, B), c(B, A) \rightarrow$
 $a(X, Y), b^2(Y, X), c^2(X, X)$

$a(A, C), b(C, B) \rightarrow$
 $a(X, Y), b^1(Y, Z)$

$a(A, C), b(C, B), c(B, A) \rightarrow$
 $a(X, Y), b^2(Y, X), c^1(Z, X)$

Was haben wir vergessen?

- Bedingungen in Queries beachten

- (Richtige) Definition

Seien q, v Anfragen und h_1, h_2 partielle CM über Symbolmengen s_1, s_2 , mit $s_1 \subseteq \text{sym}(q)$, $s_2 \subseteq \text{sym}(v)$. Sei Z_1 (Z_2) die Menge von Bedingungen aus q (v), die nur Symbole aus s_1 (s_2) beinhalten

Dann sind h_1 und h_2 kompatibel, wenn gilt:

1. $\forall X: (X \rightarrow A) \in h_1 \wedge (X \rightarrow B) \in h_2 \rightarrow A=B$
2. $Z_2 \rightarrow (h_1 \cup h_2) (Z_1)$

- Algorithmus bleibt unberührt

Komplexität

- Lemma

Seien q und v Anfragen an ein Schema S mit $n=|q|$ und $m=|v|$. Die Suche nach einem Containment Mapping von q nach v durch Aufzählen möglicher Zielliterale benötigt $O(n^m)$ Kompatibilitätstests von partiellen CM.

- Beweis

- WC: Alle Literale beider Anfragen sind von der gleichen Relation
- Für jedes der n Literale aus q gibt es dann m mögliche Ziele in v
- WC: In jedem Zweig ist alles kompatibel bis zum Blatt
- Tests auf Kompatibilität und Vereinigung von CM ist polynomial

- Tatsächlich: Problem ist **NP-vollständig**

- Beweis: Reduktion auf „Exakt Cover“ [CM77]

Real Life

- Das Problem ist linear, wenn **keine Relation zweimal** in einer Anfrage vorkommt
 - bzw. quadratisch, wenn keine Relation mehr als zweimal vorkommt
- Ein (und nicht alle) CM zu finden ist im Average Case viel schneller
- Für die **Anfrageplanung** benötigt man aber alle CM

Wo sind die Ergebnisse?

- Ein Containment Mapping h von q nach v bestimmt auch das Ergebnis von q **in den Ergebnissen von v**
 - Für jedes Tupel t im Ergebnis von v
 - Baue ein Tupel t' gemäß des **umgedrehten Mappings h^{-1}** der Variablen in $\text{exp}(q)$
- Wenn es mehrere CM von q nach v gibt, wiederhole das für jedes solche Mapping
- Die **Union aller t'** ist das Ergebnis von q , dass vom Anfrageplan v berechnet wird

Beispiel

- View $v(A, B, C) : -\text{edge}(A, B), \text{edge}(B, C)$
- Benutzeranfrage $q(X, Y) : -\text{edge}(X, Y)$
- v berechnet

From (A)	Via (B)	To (C)
1	3	3
2	3	4

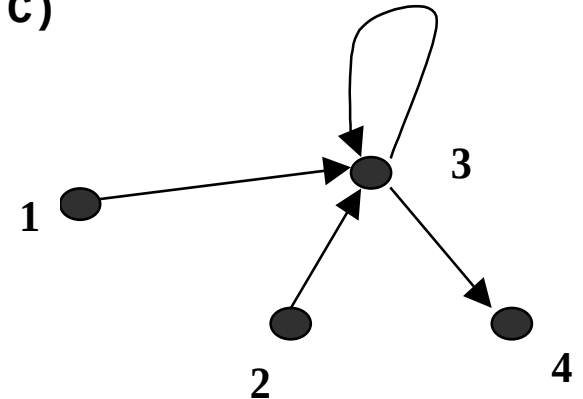
- Es gibt zwei Containment Mapping von q nach v
 - CM1: $X \rightarrow A, Y \rightarrow B$
 - CM2: $X \rightarrow B, Y \rightarrow C$

A	B	C	CM1	X	Y	CM2	X	Y
1	3	3		1	3		3	3
2	3	4		2	3		3	4

Graphanfrage erklärt

- Eine Quelle für Graphkanten
- Eine Korrespondenz
 - $\text{edge}(A, B), \text{edge}(B, C) \subseteq \text{threewaypaths}(A, B, C)$
- Nehmen wir die folgende Extension an

From (A)	Via (B)	To (C)
1	3	3
2	3	4



- Nun führen wir darauf aus
 $q(A, B, C) :- \text{edge}(A, B), \text{edge}(B, C)$

from	via	to
1	3	3
1	3	4
3	3	3
3	3	4
2	3	3
2	3	4

Formale Herleitung

- View $v(A, B, C) : -\text{edge}(A, B), \text{edge}(B, C) \quad (e_1, e_2)$
- Benutzeranfrage $q(X, Y, Z) : -\text{edge}(X, Y), \text{edge}(Y, Z) \quad (f_1, f_2)$
- v liefert

From (A)	Via (B)	To (C)
1	3	3
2	3	4

- Es gibt vier verschiedene Anfragepläne für q
 - $p1(X, Y, Z) : - v(X, Y, _), v(_, Y, Z)$ mit $X \rightarrow A_1, Y \rightarrow B_1, Z \rightarrow C_2$ und $B_1 = B_2$
 - Literale: $f_1 \rightarrow e_1, f_2 \rightarrow e_2$
 - $p2(X, Y, Z) : - v(X, Y, _), v(Y, Z, _)$ mit $X \rightarrow A_1, Y \rightarrow B_1, Z \rightarrow B_2$ und $B_1 = A_2$
 - Literale: $f_1 \rightarrow e_1, f_2 \rightarrow e_1$
 - $p3(X, Y, Z) : - v(_, X, Y), v(Y, Z, _)$ mit $X \rightarrow B_2, Y \rightarrow C_2, Z \rightarrow B_1$ und $C_2 = A_1$
 - Literale: $f_1 \rightarrow e_2, f_2 \rightarrow e_1$
 - $p3(X, Y, Z) : - v(_, X, Y), v(_, Y, Z)$ mit $X \rightarrow B_2, Y \rightarrow C_2, Z \rightarrow C_2$ und $C_2 = B_2$
 - Literale: $f_1 \rightarrow e_2, f_2 \rightarrow e_2$

$p1(X, Y, Z) :- v(X, Y, _), v(_, Y, Z)$ mit $X \rightarrow A_1, Y \rightarrow B_1, Z \rightarrow C_2$ und $B_1 = B_2$
 $p2(X, Y, Z) :- v(X, Y, _), v(Y, Z, _)$ mit $X \rightarrow A_1, Y \rightarrow B_1, Z \rightarrow B_2$ und $B_1 = A_2$
 $p3(X, Y, Z) :- v(_, X, Y), v(Y, Z, _)$ mit $X \rightarrow B_2, Y \rightarrow C_2, Z \rightarrow B_1$ und $C_2 = A_1$
 $p3(X, Y, Z) :- v(_, X, Y), v(_, Y, Z)$ mit $X \rightarrow B_2, Y \rightarrow C_2, Z \rightarrow C_2$ und $C_2 = B_2$

A	B	C
1	3	3
2	3	4



A ₂	B ₂	C ₂
1	3	3
2	3	4

=

P1	X	Y	Z
	1	3	3
	1	3	4
	2	3	3
	2	3	4

A ₁	B ₁	C ₁
1	3	3
2	3	4



A ₂	B ₂	C ₂
1	3	3
2	3	4

=

P2	X	Y	Z

A ₁	B ₁	C ₁
1	3	3
2	3	4



A ₂	B ₂	C ₂
1	3	3
2	3	4

=

P1	X	Y	Z

A ₁	B ₁	C ₁
1	3	3
2	3	4



A ₂	B ₂	C ₂
1	3	3
2	3	4

=

P1	X	Y	Z
	3	3	3
	3	3	4

Zwischenstand

- Wir können Containment von zwei Queries zeigen
- Zur Anfrageplanung
 - Globale Anfrage q
 - Menge von Korrespondenzen K
 - Deren globaler Anfrageteil ist eine Menge von Views V
 - Für jedes $v \in V$ können wir testen, ob $v \subseteq q$
 - Die v , für die das zutrifft, berechnen nur korrekte Tupel für q
 - Ihre Vereinigung ist das (bestmögliche) Ergebnis
- Nächste Schritte
 - **Frozen Facts**: Ein eleganter Algorithmus für Anfragen ohne $<$, $>$
 - **Answering Queries using Views**: Antworten durch Kombination von Views

Inhalt dieser Vorlesung

- Query Containment
- Containment Mappings
- Depth-First Algorithmus
- Frozen Facts Algorithmus
- Containment bei Disjunktion, Negation, ...

Eine andere Formulierung

- Eine relationale SPJ Anfrage ist im Grunde eine **logische Formel** mit Vektoren E und S_i
 - Aus $q(E) :- l_1(S_1), l_2(S_2), \dots, l_n(S_n)$
 - Wird $q(E) \leftarrow l_1(S_1), l_2(S_2), \dots, l_n(S_n)$
 - Mit $S = (S_1 \cup \dots \cup S_n) \setminus E$ schreiben wir kürzer: $q(E) \leftarrow \phi(E, S)$
- Lemma

Seien q_1 und q_2 Anfragen. Sei E_1 (E_2) die Menge der exportierten Variablen von q_1 (q_2) und S_1 (S_2) die Menge der nicht-exportierten Variablen. Seien $\phi_1(E_1, S_1)$, $\phi_2(E_2, S_2)$ die Rümpfe von q_1 , q_2 . Dann:

$$q_1 \subseteq q_2 \text{ gdw. } \phi_1(E_1, S_1) \rightarrow \phi_2(E_2, S_2)$$
 - Bemerkung: Implikation i.A. unentscheidbar, aber wir betrachten nur **bestimmte Formeln** (konjunktive Anfragen)

Implikation und relationale Anfragen

- Relational: $q(T, Y, O, K) \text{ :- film}(T, Y, _, _), \text{ spielt}(T, _, O, K)$
- Implikation: $\text{film}(T, Y, _, _), \text{ spielt}(T, _, O, K) \rightarrow q(T, Y, O, K)$
- Ein RDBMS **berechnet die rechte Seite** für eine gegebene Datenbasis
- Das beweist nicht die allgemeine Gültigkeit der Implikation
- Man kann aber **eine Instanz I^* bauen**, für die gilt: Genau dann wenn $\text{linke_seite}(I^*)$ ein bestimmtes Ergebnis enthält ist $\text{linke_seite} \subseteq \text{rechte_seite}$

Frozen Facts Algorithmus

- Frage: Gilt $v \subseteq q$?
- Wir basteln uns eine Instanz I^*
 - Beginne mit einer leeren Instanz I
 - Erzeuge in I **eine Relation** für jede Relation in v
 - Ersetze in v alle Variablen konsistent mit einer neuen Konstante
 - Sei t der Kopf von v nach dieser Prozedur
 - Für jedes Literal l aus v der Relation r **füge ein Tupel in r** ein, dass als Werte die Konstanten von l enthält
 - Wir „**frieren**“ v ein
 - Das ist nun I^*
 - Führe q auf I^* aus
 - Wenn $t \in \text{result}(q)$, dann ist $v \subseteq q$; sonst nicht

Beispiel

$q_1(a,b) \leftarrow rel_1(a,b), rel_2(b,b);$

$q_2(x,y) \leftarrow rel_1(x,y), rel_2(y,z);$

To check whether $q_1 \subseteq q_2$, we freeze q_1 . Therefore, we consistently replace the variables of q_1 with constants using $[a \rightarrow 0, b \rightarrow 1]$. The frozen query is then:

$q_1(0,1) \leftarrow rel_1(0,1), rel_2(1,1);$

Next, we construct an empty database D' for Σ and insert the tuple $(0,1)$ into rel_1 and $(1,1)$ into rel_2 . Finally, we compute q_2 on D' . Only one tuple, $(0,1)$, is obtained, using the valuation $[x \rightarrow 0, y \rightarrow 1, z \rightarrow 1]$. Since this is identical to the frozen head of q_1 the algorithm returns true, proving that q_1 is contained in q_2 .

To check whether $q_2 \subseteq q_1$, we first freeze q_2 with $[x \rightarrow 0, y \rightarrow 1, z \rightarrow 2]$. This results in a database D' with tuple $(0,1)$ in rel_1 and $(1,2)$ in rel_2 . Evaluating q_1 on D' yields an empty answer. This proves that $q_2 \not\subseteq q_1$.

Beweisskizze [Ull89]

- Wenn $t \notin \text{result}(q)$
 - Dann ist t ein Tupel, dass in $v(I^*)$ enthalten ist aber nicht in $q(I^*)$
 - Also ist $v \not\subseteq q$
- Wenn $t \in \text{result}(q)$
 - Dann ist die „gefrorene“ Instanz I^* ein Beispiel für eine **Instanziierung** von v auf jeder Instanz
 - Die tatsächlichen Werte sind Schall und Rauch
 - Es müssen nur an den richtigen Stellen die selben Werte stehen
 - Dieses Beispiel ist so allgemein, wie v es zulässt
 - Jedes andere Beispiel muss zu dieser Einfrierung homomorph sein
 - Da $t \in \text{result}(q)$, kann man aus diesem Beispiel eine Instanziierung für q konstruieren (und das auf jeder Instanz)
 - Also ist $v \subseteq q$

Bemerkungen

- Funktioniert nur, wenn q und v keine Bedingungen der Art „ $X < c$ “ oder „ $X > c$ “ (oder $X \neq Y$, $X > Y$, ...) enthalten
 - Diese Information können wir nicht einfrieren
- Hat (hier) Probleme bei unterschiedlichen Köpfen
 - Das kann man aber leicht reparieren
- Implementierung ist konzeptionell straight-forward
 - In einer leeren RDBMS Relationen für das Einfrieren von v erzeugen, Tupel einfügen, q (als SQL Kommando) ausführen
- Frage: Wo ist unsere exponentielle Komplexität?
 - Einfrieren ist linear in $|v|$
 - Der Test auf $t \in \text{result}(q)$ ist linear
 - Aber: Für jedes Literal in q haben wir potentiell $|v|$ Tupel in I^*
 - Damit hat $\text{result}(q)$ potentiell $|v|^{|q|}$ Tupel

Andere Prädikate

Recursion.

Proving containment of a recursive query in a non-recursive query is shown to be double exponential in [CV92]. On the other hand, containment of a non-recursive query in a recursive query has the same complexity of containment between two conjunctive queries [RSUV89]. Finally, Shmueli proves that containment of a recursive query in another recursive query is undecidable [Shm93].

Disjunction.

Containment mappings may be used for queries containing disjunction without conditions. For two queries in normal disjunctive form the following holds:

$$q_1 \cup q_2 \cup \dots \cup q_n \subseteq o_1 \cup o_2 \cup \dots \cup o_m \leftrightarrow \forall q_i \exists o_j: q_i \subseteq o_j;$$

This is not true any more if conditions are allowed. Consider the following example:

```
u(x) ← rel(x), 10 ≤ x, x ≤ 20;  
q1(x) ← rel(x), 10 ≤ x, x ≤ 15;  
q2(x) ← rel(x), 15 ≤ x, x ≤ 20;
```

Clearly, $u \subseteq q_1 \cup q_2$ – any result computed by u will be computed either by q_1 or q_2 . But $u \not\subseteq q_1$ and $u \not\subseteq q_2$. Hence, the existence of containment mapping is not a sufficient condition for queries including disjunction and conditions.

Negation.

[LS93] shows how the frozen facts algorithm can be extended to conjunctive queries including negation, and even for recursive queries with stratified negation [RU93].

Literatur

- Übersicht

- Abiteboul, Hull, and Vianu, *Foundations of Databases*. Reading, Massachusetts: Addison Wesley Publishing Company, 1995
- Ullman, *Principles of Database Systems and Knowledge-Based Systems. Volume I*: Computer Science Press, 1988
- Ullman, "Information Integration using Logical Views," ICDT, 1997

- Spezielle Fragen

- Chandra, Merlin, "Optimal Implementation of Conjunctive Queries in Relational Databases,," ACM Symposium on Theory of Computing, 1977
- Ramakrishnan, Sagiv, Ullman, et al., "Proof-Tree Transformation Theorems and their Applications," PODS, 1989
- Shmueli, "Equivalence of DATALOG Queries is Undecidable," *Journal of Logic Programming*, vol. 15, pp. 231-241, 1993
- Chaudhuri, Vardi, "On the Equivalence of Datalog Programs," PODS, 1992