Kapitel 5 Relationenkalküle als angewandte Prädikatenkalküle

Kap. 5.1 Prädikatenkalkül

Def: Terme:

- Konstante: a, b, c ... sind Terme
- Variablen: x, y, z ... sind Terme
- Funktoren: f, g, h ...
- wenn $t_1, t_2, ..., t_k$ Terme sind,
- dann ist $f(t_1, t_2, ..., t_k)$ Term.

Beispiele: f(a), g(x, b), h(y, z, x, x)

h(g(b,x), f(g(b,x)), f(a), h(a, b, c, d))

i.e. Aufschichtung von Termen beliebig weit

Def: Prädikate: (stützen sich auf Terme ab)

- Prädikatbezeichner (Prädikatoren) P, Q, R, ...
- wenn $t_1, t_2, ..., t_k$ Terme sind, dann ist
  $P(t_1, t_2, ..., t_k)$ Prädikat bzw. Prädikatsausdruck, Atom.

Def: (logischer) Ausdruck, Formel

- Atom ist ein Ausdruck
- logische Konnektoren: $\wedge \quad \vee \quad \neg$
- seien A, B Ausdrücke =>
  $(A \wedge B), (A \vee B), (\neg A)$ sind Ausdrücke
- Quantoren: $\forall \quad \exists$
- sei A Ausdruck => $\forall x\ A$ und $\exists x\ A$ sind Ausdrücke
  x ist ***gebunden***, sonst ist x ***frei***

Beispiele: P(5,3,), P(x,4), P(y,z)

(P(5,3) $\vee$ Q(a, b, c)),

R(g(f(x), 7),4)

$\forall$x P(x,4)

$\exists$ y Q(x, y, 3)

P(5,3) $\wedge \neg \exists$x P(x,y) $\vee$ R(g(f(x),7),4)

Klammern weglassen wegen Operatorpräzendenzen

$\neg$

$\wedge$

$\vee$

beliebige Umbenennung gebundener Variablen innerhalb ihres Bindungsbereiches (müssen eindeutig bleiben)

Hinweis: Bis hierher syntaktisches Spiel mit Zeichenreihen !!

Interpretation: ***Semantik, Bedeutung der Zeichenreihen!***

I : f $\rightarrow$ f';

P $\rightarrow$ P' mit

f' : $D_1 \times D_2 \times ... \times D_j \rightarrow D$

P': $D_1 \times D_2 \times ... \times D_m \rightarrow$ Bool

Bool = B = {**true**, **false**} = {0,1}

j,m sind Stelligkeit von f' bzw. P'

D_i : Individuenbereiche, Domänen, Konstanten

f' und P' sind Abbildungen, die eine Auswertung von Termen und Ausdrücken ermöglichen, f' und P' werden berechnet oder gespeichert

Beispiele: f' ≡ + ; Individuen : int

+ (3,4) = 3 + 4 → 7 bzw. f'(3,4) → 7

P' ≡ "ist Teiler von"

P'(3,6) → **true**

P'(3,5) → **false**

P'(3,x) → ? hängt von Belegung von x ab

∃x P'(3,x) ∃y P'(y,11)

∀y ∃x P'(x,y) ∃x ∃y P'(x,y) ...

Fazit:

1. Wahrheitswert einer **geschlossenen** Formel ist wohldefiniert, d.h. unabhängig von Variablenbelegung! Das ist das übliche Interesse bei Prädikatenlogik und bei Sätzen der Mathematik
2. Wahrheitswert einer **offenen** Formel ist abhängig von Variablenbelegung!

Def: ***Antwort*** **=** Menge der Belegungen für die freien Variablen, für die eine Formel **true** wird.

Beispiel: DHP(Name, Note) ∧ Note < 1,5 ***Antwort?***

∃x ∃y **[**DHP(x, y) ∧ x < 1,5 **]** ***Antwort?***

Kapitel 5.2. Relationenkalkül 1 (Tupelkalkül)

mehrere Domänen : D_1, D_2, ..., D_k

z.B. **integer** **real** **string** **date** **time** **Bool** **char** (7)
char (∗) **numeric** (6,2) **integer** (8) ... etc.

liefern Konstante unterschiedlichen Typs für Terme

Variablen und Belegungen:

In Logik: meistens Bereich für Variablenbelegungen identisch für alle Variablen: "Individuenbereich" wird deshalb in Formeln unterschlagen.

In Relationen Kalkül: Relationen haben Zweck, Bereiche für Variablenbelegungen zu definieren, z.B.

$x \in R$ mit

$R \subset D_1 \times D_2 \times ... \times D_k$

$\Rightarrow$ Spezifikation von ***Tupelvariable***:

$x \in R$, $y \in S$, $z_1 \in S$, $z_2 \in S$

etc., wird außerhalb von Formeln angegeben bzw. indirekt in SQL – Statements.

Alternative: für x beliebige Strukturen und Belegungen zulassen und Prädikat R'(x) einführen mit

R'(x) = true **iff** $x \in R$

Attributvariable:

sei A_i Attribut von Relation R,

sei x $\in$ R, d.h. Belegungsbereich von *Tupelvariable* x sind die Tupel von R

x.i bzw. x.A_i ist ***Attributvariable*** (Punktnotation) für R.

Variablenbelegung:

x $\in$ R kann beliebiges Tupel von R sein,

dann ist x.i mit entsprechender Komponente dieses Tupels belegt (implizit)

z.B. x := (6, 1, 'blau')

x.2 = 1 x.3 = 'blau'

Terme: Satz von konkreten Operatoren, z.B.

+ - ÷ : ∗ für Attributvariablen

mit Domänen **int**, **real**

Terme werden mit Attributvariablen gebildet!

t.price ∗ 1.16

part.cost + labor.time ∗ worker.rate

today – worker.birthdate

mit Tupelvariablen: part $\in$ PART,

labor $\in$ LABOR worker $\in$ WORKER

Merke: Unterscheide zwischen Tupelvariable und Relation!

Prädikate:

Vergleichsprädikate: **< = . . . >**

mit Infix-Schreibweise bzw.

LT LE EQ NE GE GT

LIKE für **string** Vergleich mit Vergleichsmuster da%ba%

Mengenvergleichs-Prädikate: zum Vergleich von Mengen von Tupeln, z.B. von Relationen, Antworten

CONTAINS	NOT CONTAINS
contains	**not contains**
equal	**not equal**

a **in** R bzw. $a \in R$ ist Kurzform für:

$$a = (a_1, a_2, ..., a_k) \wedge$$

$\exists r\ (r.1 = a.1. \wedge r.2 = a.2 \wedge ... \wedge r.k = a.k)$

Ausdrücke: aufgebaut mit

- Vergleichsprädikaten für Attributvariablen
- ∀ ∃ über Tupelvariablen
- Mengenvergleichen

Beispiele: Ausdrücke, Formeln zur Anfrageformulierung,
- berechnung für Uni-Schema:

PROF	**MITARB**
Zi#	M-Name
P-Name	forscht
Tel#	betreut

O208, Paul, 48095261 Specht, OMNIS, Bayer
O108, Bayer, 48095171 Reiser, SFB, Bayer

Anfrage 1: *“Mitarbeiter von Prof. Bayer?”*

Tupelvar: px $\in$ PROF , my $\in$ MITARB

Anfrage – Ausdruck: QA1

my.betreut = ‘Bayer’ $\equiv$ QA1 (my)

{my.Name | my.betreut = ‘Bayer’}

Freie Variable ***my***,
Antwort = alle Belegungen von my,
so daß QA1 **true** wird

Anfrage 2: *“Chef von Specht mit Name und Tel#?”*

Anfrage – Ausdruck: QA2

my.M-Name = ‘Specht’ $\wedge$ my.betreut = px.P-name $\equiv$ QA2(my, px)

{ px.P-Name, px.Tel# | my.M-Name = ‘Specht’
$\wedge$ my.betreut = px.P-name }

Hier wird my wegprojiziert

Hinweis: QA1(my), QA2(my, px)
sind Query-Ausdrücke bzw. offene Formeln mit freien Variablen
my, px
$\Rightarrow$ QA1, QA2 werden **true** für bestimmte Belegungen von px, my

Als Antworten auf QA1, QA2 interessieren nur Teile der Belegungen, deshalb Projektion

Partieller Abschluß einer Formel mit ∃:
Falls nur Teile der Antwort interessieren, z.B.
∃my: QA2(my, px)

Projektionsliste: Übergang von Tupelvariablen zu Attributvariablen bei freien Tupelvariablen z.B.

px.P-Name, px.Tel# | QA2 (my, px)
oder mit expliziter Quantifizierung:
px.P-Name, px.Tel# | ∃ **my**: QA2 (my, px)

Vorgriff auf SQL:

```
select px.P-Name, px.Tel#
from   px ∈ PROF, my ∈ MITARB
where  my.M-Name   = 'Specht'   and
       my.betreut        = px.P-Name
```

Zusammenfassung Tupelkalkül:

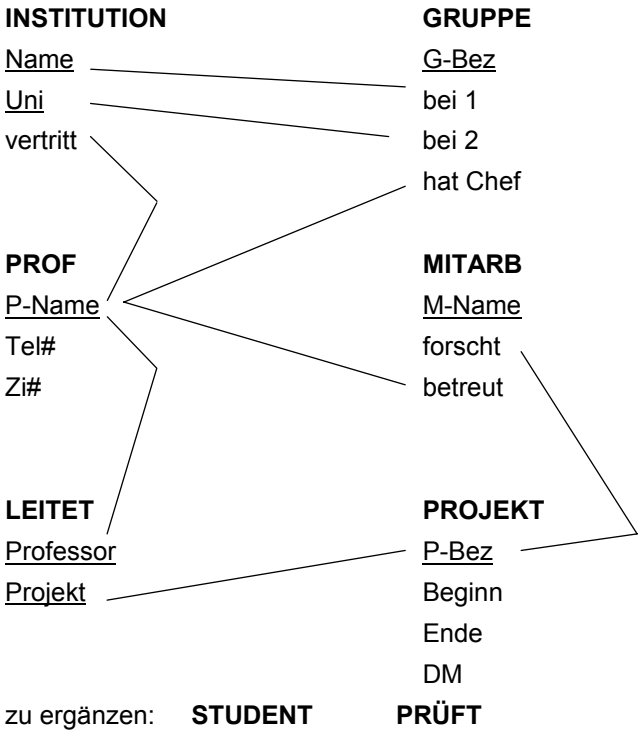
1. Bindungsbereich von Tupelvariablen
   festlegen ~ Relation
2. Query-Ausdruck QA formulieren unter Verwendung von Attributvariablen, Konstanten und eingebauten Vergleichsoperatoren, Quantoren
3. Was für Antwort nicht interessiert mit ∃ quantifizieren. Antwort auf QA ist die Belegung der freien Variablen, so daß QA **true** wird.
4. Für noch freie Tupel-Variablen die interessierenden Attributvariablen angeben, “Projektionsliste”, i.e. Erweiterung des standard Prädikatenkalküls.
   $x.i_1, ..., x.i_k, ..., z.j_1, ..., z.j_e : QA(...)$

   offener prädikatenlogischer Ausdruck

   x ... z sind genau die Tupelvariablen, die in QA frei sind!
5. Implizite Quantifizierung mit ∃, falls freie Tupel Variablen nicht in Projektionsliste stehen.

Kap. 5.3. Beispiele Tupel Kalkül

Relationen Graph: Schema mit Join-Möglichkeiten:



zu ergänzen: **STUDENT** **PRÜFT**

1. *“Professoren mit Tel#, Zi#”*

px : px ∈ PROF

2. *“Tel# von Prof. Paul”*

px.Tel# : px ∈ PROF ∧
px.P-name = ‘Paul’

3. *“Mitarbeiter, die von Prof. Bode betreut werden”*

mx : mx ∈ MITARB ∧
mx.betreut = ‘Bode’

4. *“Name und Tel# des Betreuers von Specht”*

py.P-Name, py.Tel# :
mx ∈ MITARB ∧ py ∈ PROF ∧
mx.Name = ‘Specht’ ∧
mx.betreut = py.P-Name

i.e. Verfolgung genau eines Verweises

5. *“Alle Mitarbeiter von Projekten, die vor 1998 enden”*

mx : mx ∈ MITARB ∧ pz ∈ PROJEKT ∧
pz.Ende < [1998 : 01 : 01] ∧
mx.forscht = pz.P-Bez

i.e. Rückverfolgung von vielen Verweisen (Optimierung?)

6. *“Leiter von Projekten, die vor 1998 enden mit Tel#, Zi#“*

x : x ∈ PROF ∧ z ∈ LEITET ∧ y ∈ PROJEKT ∧
x.P-Name = z.Professor ∧
z.Projekt = y.P-Bez ∧
y.Ende < [1998 : 01 :01]

i.e. 2 Joins mit Restriktion

7. *“Projekte mit Mitarbeitern, die an TU promovieren”* ...

Allgemeine Vorgehensweise für Anfrageformulierung:

anhand des Relationen-Graphen mit Fremdschlüssel Diagramm:

- gewünschte Info: $\Rightarrow$ Projektionsliste
- Verweis – Verbindungen: Join – Bedingungen, z.B. x.A = y.B
  daraus ergeben sich die beteiligten Relationen!
- Restriktionen:$\Rightarrow$ oft Konstantenvergleich, Intervall-Bedingungen, z.B.
  mx.betreut = 'Bode' oder
  pz.Ende < [1998 : 01 : 01] $\wedge$
  pz.Ende $\geq$ [1996 :08 :01]
- Tupel-Variablen über benötigten Relationen

Kap 5.4. Relationenkalkül 2 (Domänen Kalkül)

- Variablen haben als Bereich Domänen
- Relationen werden als Prädikate aufgefaßt, i.e.
  Prädikat R' für Relation R
  R'(x,y,z) als offenes Atom, d.h.

  R'(x,y,z) wird **true** für Belegung
  x := a y := b z := c
  iff (a,b,c) $\in$ R
  Rel. R als Menge von Tupeln aufgefaßt.

Beispiel aus Tupel-Kalkül

$rx \in R \quad \wedge \quad sy \in S$

{ rx.i, sy.j | Q (rx, sy) **}**

wird im **Domänen – Kalkül**:

R' (D_1, D_2, D_3) S' (D_4, D_5)

x_1 x_2 x_3 y_1 y_2

(Domänen – Variablen!)

mit Prädikat Q' und mit Identifizierung von Attributen durch Positionen, d.h. keine Attr. Namen:

aus Q wird Q':

$R'(x_1, x_2, x_3) \wedge S' (y_1, y_2) \wedge$

$Q' (x_1, x_2, x_3, y_1, y_2)$

d.h. x_i, x_j ähnlich wie Attribut-Var.

Vereinfachung bei Equi-Joins:

Join-Bedingung x.A = y.B in Tupel-Kalkül wird durch Variablen-Identifizierung ausgedrückt:

siehe Query 4, S. 21 in Kap. 5.3:

$mx \in$ MITARB $\wedge$ $py \in$ PROF $\wedge$
mx.Name = 'Specht' $\wedge$
mx.betreut = py.P-Name

im Domänenkalkül:

MITARB´ (x, y, z) $\wedge$ PROF´ (u, v, w) $\wedge$
x ='Specht' $\wedge$ z = u

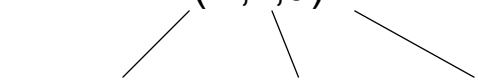
$\Rightarrow$ MITARB´ ('Secht', y, z) $\wedge$
PROF´ (z, v, w)

Hauptproblem bei Domänenkalkül

Semantische Bedeutung von Positionen von Variablen?

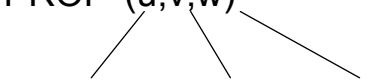
MITARB' (x, y, z) äquivalent zu:

MITARB' (w,v,u)



Projektionsliste und weitere freie Variablen:

PROF' (u,v,w)



{ u,v | MITARB´ (x,y,z) ∧ PROF´ (u,v,w) ∧ x = 'Specht' ∧ z = u **}**

oder:

{ z,v | MITARB´ ('Specht', y, z) ∧ PROF´ (z,v,w) **}**

Frage: freie Variablen y, w?

werden implizit mit ∃ quantifiziert, d.h.

{ z, v | ∃ y ∃ w : MITARB´ ('Specht', y, z) ∧ PROF´ (z, v, w) }

Betrachte als Definition eines neuen Prädikats CHEF (z, v)

Die Belegung der freien Variablen z, v ist Antwort auf Anfrage.

Zusammenfassung:

Tupel-Kalkül: relationale DB-Systeme,
Theorie-Fundament für SQL

Domänen-Kalkül:

Üblich bei Logik-Programmierung,
z.B. PROLOG u. deduktive DBSe

Beispiel aus MVV im Domänenkalkül (PROLOG Notation):

Stops (Linie, Haltestation)

Umsteigen ('S3' , H, 'U5') $\leftarrow$ Stops ('S3', H) $\wedge$
Stops ('U5', H)

Stops als Beziehung zwischen zwei Entities (Linie und Haltestation) aufgefaßt.

dasselbe Beispiel *Umsteigen* im Tupel-Kalkül:

{ ‘S3‘, s1.2, ‘U5‘ |

(s1 $\in$ Stops $\wedge$ s2 $\in$ Stops $\wedge$
s1.1 = ‘S3‘ $\wedge$ s2.1 =‘U5‘ $\wedge$ s1.2 = s2.2) **}**

mit SQL-Syntax:

select ‘S3‘, s1.2, ‘U5‘
from Stops s1, Stops s2
where (s1.1 = ‘S3‘ $\wedge$ s2.1 = ‘U5‘ $\wedge$ s1.2 = s2.2)

Hinweis: die Kompaktere Schreibweise des Domänenkalküls ändert sich bei “breiten” Relationen und semantischen Attributnamen zugunsten des Tupelkalküls.