LMU – Proseminar „Grundlagen höherer Programmiersprachen“

Thema:

# „Wie ein beispielhaftes Query-System funktioniert“

Vortragender: Petar Tonev

# 1. Vorworte

- ***Ziele und Aufgaben des Vortrags*** – klar zu machen, wie das für Beispiel verwendete Anfragesystem die Herleitung musterbasierter Informationen aus einer Datenbank ermöglicht, sowie zu zeigen, über welchen wesentlichen Mittel es verfügt

- ***Grundlagen des Anfragesystems*** – Mustervergleich (einseitig) und Unifikation (zweiseitig). Strukturierung der Informaton mit Hilfe von Datenströmen.

- ***Implementierung und Beispiele***: der Vortrag wird durch die Query-Sprache aus dem Buch „Structure and Interpretation of Computer Languages“ veranshaulicht.

# 2. Mustervergleich (Pattern Matching)

***Syntax eines Datums:*** ***(*** *(<Wert>*)** *(Wert)* ***)***
**Beispiele: (a), (a a), ((2 a) b ), ( (b a) 7 ((a c) d) ), . . .**

***Syntax eines Musters: (*** *(<Variable> <Variable>* <Wert>*)** ***)***
**Beispiele : (a ?x), (?x ?y), ( (?x a) 7 ((a ?y) d) )**

*Es heißt hier*: in einem Datum können Werte mit beliebiger inneren Verklammerung und Anzahl der Terme vorkommen; bei einem Muster muss es dazu wenigstens eine Variable geben.

## 1) ***Was ein Mustervergleicher ist und was macht er?***

-ein Mustervergleicher ist ein Programm, das überprüft, ob ein Datum zu einem gegebenen Muster passt.

**Beispiel:** das Datum **((a b) c (a b))** entspricht
-dem Muster **(?x c ?x)**, wenn **?x←(a b)**;
-dem Muster **(?x ?y ?z)**, wenn **?x←(a b)** , **?z←(a b)** , **?y←c**
-dem Muster **((?x ?y) c (?x ?y))** wenn **?x←a**; **?y←b**

Das Datum entspricht nicht dem Muster **(?x a ?y)** , weil dies eine Liste mit zweitem Element **a** darstellt.

2) ***Eingabe und Ausgabe eines Mustervergleichers*** – das Anfragesystem erwartet als solchen ein Muster, ein Datum, und einen Bindungsrahmen, der die (eventuell bereits vorhandenen) Bindungen für die Mustervariablen festlegt.

*Bindungsrahmen –ein assoziatives Array, in dem die Schlüsselvariablen Namen sind
***Syntax:*** *[<Variablenname**1**>←<Wert **a**>; . . . <Variablenname **n**>←<Wert **t**>]*

**Beispiel :**

| | |
|---|---|
| **?x** | **a** |
| **?y** | **12** |
| **?z** | **John** |

***Der Mustervergleicher*** überprüft also ob das Datum dem Muster in einer Weise entspricht, die mit den Bindungen aus dem Rahmen konsistent ist:

-falls ja, werden der gegebene Rahmen und eventuelle neue Bindungen abgeliefert.

-falls nein, wird angegeben „Vergleich nicht erfolgreich“

**Beispiel:** Muster **(?x ?y ?x)**, Datum **(a b a)** ,
-bei leerem Bindungsrahmen, =>
Ausgabe :

| | |
|---|---|
| **?x** | **A** |
| **?y** | **B** |

-bei Eingaberahmen

| | |
|---|---|
| **?y** | **a** |
| **?x** | **?** |

=>Fehlschlag (Vergleich nicht erfolgreich)

-bei Eingaberahmen

| | |
|---|---|
| **?y** | **b** |
| **?x** | **?** |

Ausgabe :

| | |
|---|---|
| **?y** | **b** |
| **?x** | **a** |

***3)Nachteile vom Mustervergleich:*** er ist in der Regel sehr aufwendig. Daher wird vermieden, den gesamten Vergleich auf jedes Element der Datenbank anzuwenden. Dies erreicht man durch eine Aufteilung in schnellen Grobvergleich und abschließenden Feivergleich.
-durch *Indizieren* könnte die Datenbank bei der Aufbau so arrangiert werden, dass meist des Grobvergleichs schon erledigt ist .

# 3. Ströme von Bindungsrahmen

*Problem: Darstellung aller Ergebnisse eines Vergleichs*
→ein ***Datenstrom*** ist im allgemein eine „verzögerte“ Liste. Er kann z.B. in *Scheme* wie folgt konstruiert werden :

**(cons <*a*> (delay <*b*>))**, was gleich zu **(cons-stream <*a*> <*b*>)** ist.

Hier ist **delay** eine Sonderform , die den Term <**b**> nicht auswertet, sondern ein sog. *verzögertes Objekt* liefert, das später ausgewertet werden soll (eine Art „Versprechung“ für künftige Auswertung) . Dies erfolgt im Fall durch die Prozedur ***force*:**

**(define (stream-car.stream) (car stream))**
**(define (stream-cdr.stream) (force (cdr stream)))**

So wird zunächst das erste Element des Stromes ausgewertet, und erst bei Anforderung durch *force* (das heißt „forciere“) auch der ***tail*** des Stromes ausgewertet.

**→ein *Strom von Bindungsrahmen*** ist daher eine Liste von Bindungen, manche von denen noch dazukommen können. D.h., ihre Erscheinung ist im allgemein bevorstehend.

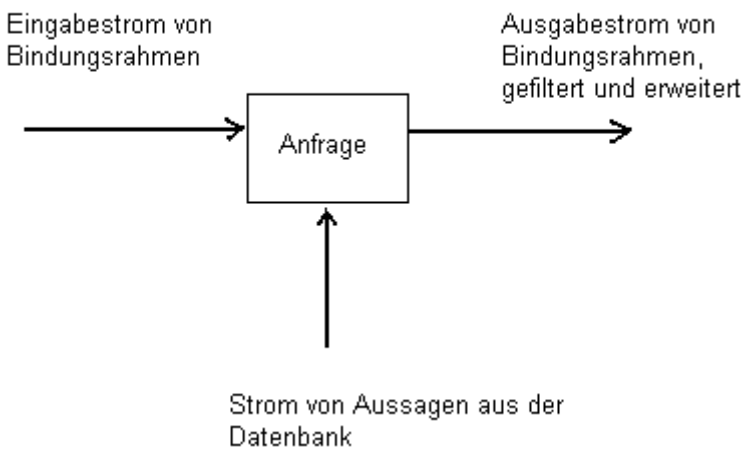
**Beispiel für Strom von Bindungsrahmen:**
**{(?x←2; ?y←5) (?z←?x) (?z←?) …}**

→bei Eingabe eines Rahmens werden die Einträge der Datenbank durchlaufen, wobei für jeden Eintrag erzeugt der Mustervergleicher eine Erweiterung für den Rahmen, oder ein spezielles Symbol fürs Fehlschlagen des Vergleichs.

→alle Ergebnisse werden zu einem Datenstrom zusammengefasst.

→der Ergebnisstrom wird durch Filter geschickt, um eventuelle Fehlschläge auszusortieren. Dieses Verfahren wird unten veranschaulicht:

**Abbildung 1:** *Das Anfragesystem, beschrieben durch Datenströmen*

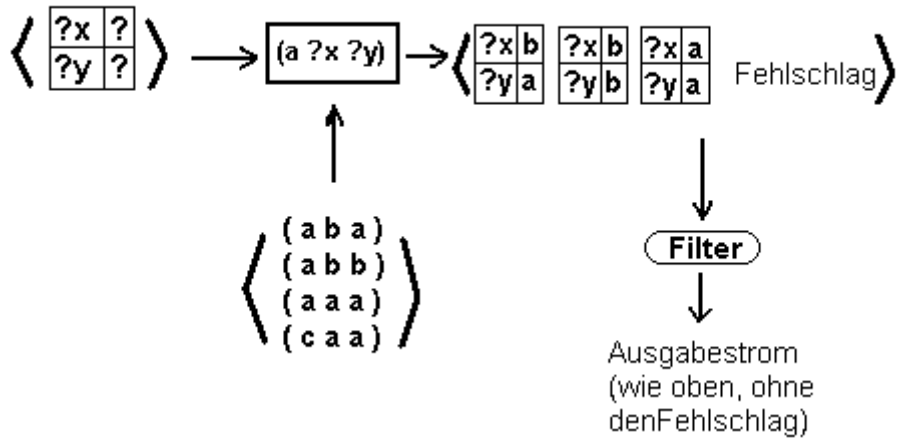


**→bei einfachen Einfragen** besteht der Eingabestrom aus einzigem leeren Bindungsrahmen, der Ausgabestrom – aus alle Erweiterungen des leeren Rahmens. D.h. die Ausgabe ist Kopie des Anfragemusters mit instantiierten(d.h. durch Werte ersetzten) Variablen.

**Syntax von einfachen Anfragen:** *(<Variable>* <Konstante>*)*
*d.h. Variablen und Werte können beliebig oft zukommen*

z.B. **=>(job ?x ?y)**
**(job A a)**
**(job B b)**
**(job C b)** , d.h. alle Aussagen bezüglich der Tätigkeit aus der Datenbank werden aufgelistet.

**Abbildung 2:** *Das Anfragesystem mit beispielhaften Eingabeströmen, Datenkbank und Anfrage*




*Die Abbildung stellt dasselbe Schema wie in Abb.1 dar, hier anhand beispielhaften Strömen und Anfrage*

# 4. Zusammengesetzte Anfragen

*Das sind durch logische Operatoren verknüpfte einfache Anfragen.*

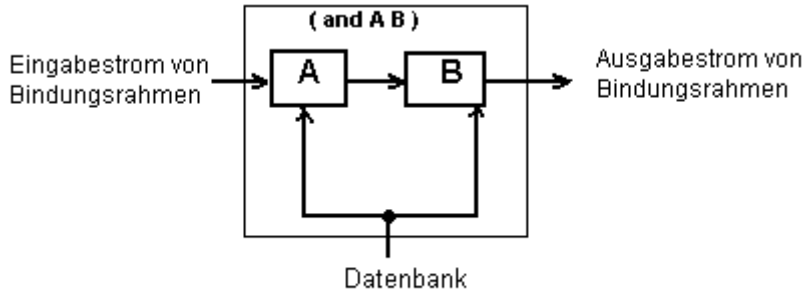
## 4.1. and

**Beispiel: (and (kann-taetigkeit ?x (computer programmierer trainee))**
**(taetigkeit ?person ?x))**

-zuerst finde die Einträge, die dem ersten Muster entsprechen, dann dem zweiten (sequenziell)

-die Rahmen, die der erste Anfragefilter durchlässt werden von zweiten Anfrage noch mal gefiltert und erweitert

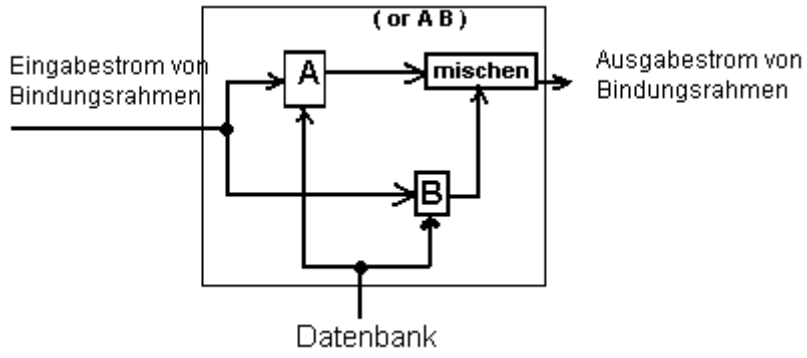
**Abbildung 3:** ***and*** *von zwei Anfragen A und B*




## 4.2. or

-der eingegebene Strom von Bindungsahmen wird für jede Anfrage getrennt erweitert (parallel)
-die beide Ergebnisströme werden dann aneinandergehängt, und somit ein endgültiger Strom erzeugt

**Abbildung 4:** ***or*** *von zwei Anfragen A und B*




**/*** ***Bemerkung zu 4.1 und 4.2*** :Die Verarbeitung von zusammengesetzten Anfragen kann sehr langsam sein: bei **D** Elemente in der Datenbank und **n** Klauseln => Vergleiche in der Größenordnung **Dn.*/**

## 4.3. not

„not“ verhält sich wie ein Filter, der Bindungsrahmen entfernt, die der Anfrage entsprechen.
**Beispiel:**
**(not (taetigkeit ?x (computer programierer)))**

Das Ergebnis ist ein Strom, der nur aus solcher Rahmen besteht, in denen die Bindung für die Variable **?x** dem Anfragemuster nicht entspricht. Hier im Beispiel werden alle Rahmen durchgelassen, wo an der Variable **x** Namen gebunden sind, die den Angestellten, die nicht Programmierer sind, entsprechen.

## 4.4. lisp-value

**Beispiel: (and (gehalt ?person ?betrag)**
**(lisp-value > ?betrag 50000))**

-„lisp-value"verhält sich auch wie ein Filter
-das lisp-Prädikat entfernt aus dem Eingabestrom alle Rahmen, für die es fehlschlägt

Im Beispiel werden alle Personen gefunden, wessen Gehalt über 50000 Einheiten beträgt.

# 5. Unifikation

## 5.1. Substitution, Instanz, Unifikator

→***Substitution*** ist eine Abbildung von der Menge der Variablen in die Menge der Terme $\vartheta: V \to T(\Sigma)$, die nur an endlich vielen Stellen von der Identität abweicht.(dabei ist die Identität die leere Substitution).

→***Instanz*** :eineFormel $\varphi$ ist Instanz einer Formel $\Psi$, falls es eine Substitution $\vartheta$ gibt mit $\varphi = \Psi\vartheta$ (hier wird für $\vartheta$ Postfixnotation verwendet)

→eine Substitution heißt ***Unifikator*** für die Menge **S = {A1,A2,...An}**, falls alle Instanzen der Formel **A1,...An** gleich sind :
**( A1$\vartheta$ = ...=An$\vartheta$ )**

-Gibt es einen Unifikator =>**S** *unifizierbar.*
-*allgemeinster Unifikator* $\vartheta$ - falls es für jeden Unifikator $\sigma$ aus **S** eine Substitution $\beta$ gibt, sodass $\sigma = \vartheta\beta$

**Beispiele:**
1) **{ f( x, y ), f( 3, z ), f( 3, 7 )}**
**$\vartheta$ = { x←3, y←7, z←7 }**

2)**…...{f( x, y), f( 3, z), f( 5, 7) }**
*nicht unifizierbar*

3) **{f(x), f(y)}**
**$\vartheta$ = { x ← y }** *(allgemeinster Unifikator; z.B. {x<-3}, {y<-3} Unifikatoren, aber nicht allgemeinst)*

## 5.2. Unifikation im Anfragesystem

*Wozu braucht man Variablen im Datum?* → Möglichkeit Regeln anzuwenden

♦ **Unifikation** ist eine Verallgemeinerung des Mustervergleichs, wo nun auch das Datum ***Variablen enthalten kann.***

♦ein **Unifikationsalgorithmus** (bzw. ein unifizierendes Programm) stellt für zwei Aussagen fest, ob es möglich ist, an die Variablen Werte zuzuweisen, sodass die Aussagen gleich werden.

**Beispiele:**

**(1)** **⇒(?x a ?y)**
**(?y ?z a)**

**Ausgabe:**

| | |
|---|---|
| **?x** | **a** |
| **?y** | **a** |
| **?z** | **a** |

**(2)** **⇒(?x ?y a)**
**(?x b ?y)**

**Ausgabe: Unifikation erfolglos** *(es gibt keinen Wert für y, mit dem die Muster gleich werden können)*

♦ Bei komplexen Mustern sind also Ableitungen notwendig .

**z.B.** unifiziere **(?x ?x)** und **((a ?y c)(a b c))**

Der Unifikationsalgorithmus muss ableiten, dass
**?x←(a b c), ?y←b, ?z←c**

*Lösung*: durch Menge von Gleichungen zwischen den Musterkomponenten.
*Hier im Beispiel*:
**?x = (a ?y c)**
**?x = (a b ?z)** **=> (a ?y c) = (a b ?z) => a = a, ?y = b, c = ?z => ?x = (a b c)**

/****Bemerkung**:* möglich, dass eine erfolgreiche Unifikation die Werte für die Variablen nicht vollständig bestimmen kann. So können manche Variablen ungebunden bleiben, und andere an variablenenthaltenden Werte gebunden werden.

**Beispiel:** unifiziere **(?x a)** und **((b ?y) ?z)**
*Ableitung:* **?x = (b ?y)**
**a = ?z**
Hier im Beispiel sind **?x** und **?y** weiter nicht auflösbar (es ist nicht möglich, an **?x** und **?y** Werte zuzuweisen, sodass die beiden Muster gleich. Dabei **?y** nicht eingeschränkt => es ergibt sich durch die Unifikation keine Bindung für **?y**.
**?x** ist jedoch eingeschränkt, und zwar durch **(b ?y)** => Bindung für **?x**, die Variable beinhaltet) ***/**

# 6. Anwendung von Regeln

→Syntax von Regeln: **(rule <*Folgerung*><*Rumpf*>)**
Die Folgerung sieht wie ein Muster Aus, der Rumpf kann in der Regel zusammengesetzt sein.
→Verfahren:

(1) Unifiziere die Anfrage mit der Folgerung der Regel, um eine Erweiterung für den (ursprünglichen) Rahmen zu finden
(2) Werte den Rumpf der Regel aus, relativ zu dem erweiterten Bindungsrahmen

*/*Dieses Verfahren ist sehr ähnlich dem, das die imperative Sprache Lisp benutzt*/*

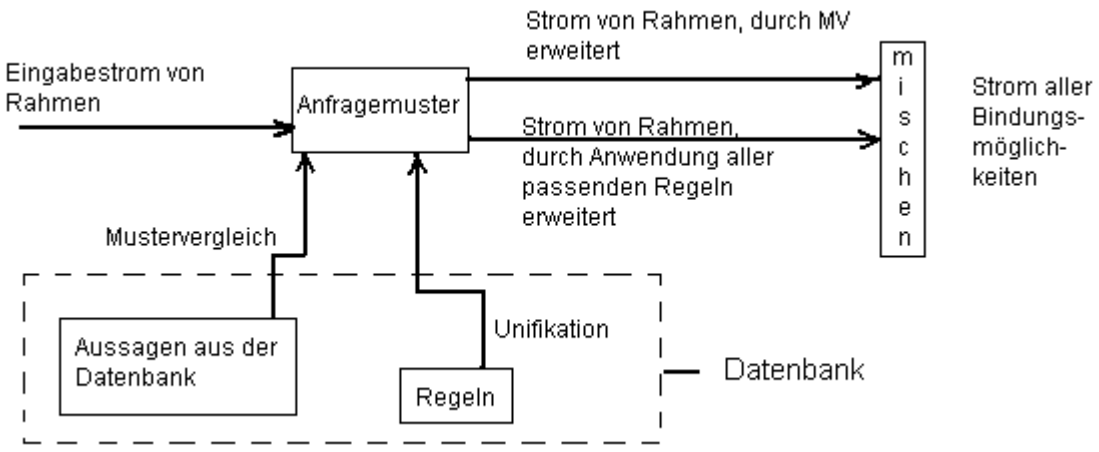
**Beispiel:**
**(wohnt-in-der-naehe ?x (Hacker Alyssa p))**
*anzuwendende Regel:*

```
(rule ((wohnt-in-der-naehe ?person-1 ?peson-2)                  // Folgerung
       (and (adresse ?person-1 (stadt . ?rest-1))               // Rumpf
            (adresse ?person-2 (stadt . ?rest-2))
            (not (lisp-value equal? ?person-1 ?person-2))))     //damit P1≠P2
```

1) Die Unifikation ergibt: **?person-2←(Hacker Alyssa P)**
2) Relativ zu diesem Rahmen werte den Rumpf aus => es ergibt sich eine Erweiterung des Rahmens um eine Bildung für **?person-1** =>**?x** mit diesem Wert instantiiert.

## Einfache (nicht zusammengesetzte) Anfragen

**Abbildung 5:** *Auswertung von einfachen Anfragen bei Anwendung von Regeln*



Hier in der Abbildung wird beschrieben, wie können einfache Anfragen mit sowohl Aussagen aus der Datenbank als auch Regeln (hier liegt die Erweiterung im Vergleich zu Abb. 1) ausgewertet werden.

→dabei werden zwei Ergebnisströme erzeugt: einen von erweiterten Bindungsrahmen, der durch Matching des Musters mit allen vorhandenen Aussagen erhalten ist, und zweiten, der sich durch die Anwendung aller der Anfrage passenden Regeln ergibt.

→durch Aneinanderhängen der beiden Ströme wird ein einziger erzeugt, der aus allen Bindungsmöglichkeiten besteht, denen das angegebene Muster entspricht, und die konsistent mit den ursprünglichen Bindungsrahmen sind.