

# Übungen zu Informatik I Wintersemester 03/04

Übungsleiter: Dipl.-Inform. Tom Gelhausen



## ▪ Probeklausur

- Fr, 19.12.2003
- 14:00 Uhr – 15:30 Uhr
- Beginnen Sie **jetzt**, das Lernen zu üben!



- Anwendung der Substitution  $\sigma = [s/v]$  auf einen Term t
  1. Ist t die Variable v, so ist  $t\sigma = t[s/v] = v[s/v] = s$
  2. Ist t eine Konstante c oder eine Variable  $\neq v$ , so ist  $t\sigma = t$
  3. Ist  $t = f(t_1, \dots, t_n)$ , f eine Funktion  $\in \Sigma$  und  $t_1, \dots, t_n$  Terme, so ist  $t\sigma := f(t_1\sigma, \dots, t_n\sigma)$

- **Beispiel**  $t = f(g(x), h(y), p(q(z)))$  und  $\sigma = [r/x, s/z]$

$$\begin{aligned}
 t\sigma &= f(g(x), h(y), p(q(z)))\sigma \\
 &= f(g(x)\sigma, h(y)\sigma, p(q(z))\sigma) \quad \text{3.} \\
 &= f(g(x\sigma), h(y\sigma), p(q(z)\sigma)) \\
 &= f(g(r), h(y), p(q(z\sigma))) \\
 &= f(g(r), h(y), p(q(s)))
 \end{aligned}$$

- **Beispiel**  $t = f(g(x), h(y))$  und  $\sigma = [y/x, z/y]$

$$\begin{aligned}
 t\sigma &= f(g(x), h(y))\sigma \\
 &= f(g(x)\sigma, h(y)\sigma) \quad \text{3.} \\
 &= f(g(x\sigma), h(y\sigma)) \\
 &= f(g(y), h(z))
 \end{aligned}$$

~~$f(g(y), h(y))$~~   
 $f(g(z), \dots)$



- $\sigma$  ist Unifikator für zwei Terme  $s$  und  $t \Leftrightarrow \underline{s\sigma = t\sigma}$
- $\sigma$  Unifikator und  $\tau$  beliebige Substitution  $\Rightarrow \sigma\tau$  ist Unifikator
  - Sei  $v := s\sigma$  und  $w := t\sigma$ 
    - $s\sigma = t\sigma \Rightarrow v = w \Rightarrow v\tau = w\tau \Rightarrow s\sigma\tau = t\sigma\tau \Rightarrow \sigma\tau$  ist Unifikator
  - Definiere Relation:  $\sigma\tau \preceq \sigma$ 
    - $\preceq$  definiert Quasiordnung
    - Sei  $\sigma$  Unifikator. Wenn für alle anderen Unifikatoren  $\sigma'$  gilt:  
 $\sigma \preceq \sigma' \Rightarrow \sigma' \preceq \sigma$ , dann ist  $\sigma$  allgemeinsten Unifikator
  - Interpretation:
    - Unifikator  $\sigma\tau$ : Untere Schranke
    - Allgemeinsten Unifikator  $\sigma$ : Größte untere Schranke
- Ein Unifikator muss nicht existieren!
  - **Beispiel:**  $s = x$ ,  $t = f(x)$



- Abweichungspaar  $A(s, t)$  zweier Terme s und t
  - Das erste Paar unterschiedlicher Unterterme, wenn wir s und t von links nach rechts durchlesen.
  - **Beispiel:**
    - $s = f(g(a, h(x)), y, h(z))$
    - $t = f(u, g(a, h(b)), h(c))$
    - ☛  $A(s, t) = (g(a, h(x)), u)$
- Unifikationsalgorithmus, Gegeben: Terme s, t
  - Setze  $\sigma := \varepsilon$  (identische Substitution)
  - Solange es ein Abweichungspaar  $A(s\sigma, t\sigma) = (p, q)$  gibt
    - p ist eine Variable und p kommt in q nicht vor  $\Rightarrow \sigma := \sigma[q/p]$
    - p ist eine Variable und p kommt in q vor  $\Rightarrow s, t$  nicht unifizierbar.
    - q ist eine Variable und q kommt in p nicht vor  $\Rightarrow \sigma := \sigma[p/q]$
    - q ist eine Variable und q kommt in p vor  $\Rightarrow s, t$  nicht unifizierbar.
    - weder p noch q Variable  $\Rightarrow s, t$  nicht unifizierbar.



## ▪ **Beispiel:**

$$s = f(g(a, h(x)), y, h(z))$$

$$t = f(u, g(a, h(b)), h(c))$$

- $\sigma := \varepsilon$
- $A(s\sigma, t\sigma) = A(s, t) = (g(a, h(x)), u) =: (p, q)$
- $q$  ist eine Variable und  $q$  kommt in  $p$  nicht vor  $\Rightarrow \sigma := \sigma[p/q] = [g(a, h(x))/u]$
  
- $\sigma = [g(a, h(x))/u]$
- $s\sigma = f(g(a, h(x)), y, h(z))$
- $t\sigma = f(\underline{g(a, h(x))}, g(a, h(b)), h(c))$
- $A(s\sigma, t\sigma) = (y, g(a, h(b))) =: (p, q)$
- $p$  ist eine Variable und  $p$  kommt in  $q$  nicht vor  $\Rightarrow \sigma := \sigma[q/p]$   
 $\sigma[q/p] = [g(a, h(x))/u, \underline{g(a, h(b))}/y]$



## ▪ **Beispiel:**

$$s = f(g(a, h(x)), y, h(z))$$

$$t = f(u, g(a, h(b)), h(c))$$

$$\sigma = [g(a, h(x))/u, g(a, h(b))/y]$$

$$s\sigma = f(g(a, h(x)), g(a, h(b)), h(z))$$

$$t\sigma = f(g(a, h(x)), g(a, h(b)), h(c))$$

$$A(s\sigma, t\sigma) = (z, c) =: (p, q)$$

## ▪ Zwei Möglichkeiten

- p ist eine Variable und p kommt in q nicht vor  $\Rightarrow \sigma := \sigma[q/p]$

- q ist eine Variable und q kommt in p nicht vor  $\Rightarrow \sigma := \sigma[p/q]$

## ▪ Somit sind die beiden allgemeinsten Unifikatoren

- $\sigma_1 = [g(a, h(x))/u, g(a, h(b))/y, c/z]$

- $\sigma_2 = [g(a, h(x))/u, g(a, h(b))/y, z/c]$



- Logik verwendet faktische Aussagen, z.B.

- *Zürich hat einen Bahnhof.*
- *Bern hat einen Bahnhof.*

und Implikationen, wie

- *Wenn zwei Städte X und Y einen Bahnhof haben, dann sind*
- *X und Y miteinander verbunden.*

- In Prolog werden diese Aussagen formal als Klauseln

- `bahnhof(zürich) .`
- `bahnhof(bern) .`
- `verbunden(X, Y) :- bahnhof(X) , bahnhof(Y) .`

geschrieben.

- Man beachte, dass Konstantenbezeichner (`zürich`) klein, aber Variablenbezeichner (`x`) gross geschrieben werden.
- `_` (Unterstrich) ist ein Variablenbezeichner wechselnder Identität.



- Klauseln kommen in zwei Formen: Fakten und Regeln.
- Das Faktum
  - `bahnhof(zürich).`steht für den Satz
  - *“Zürich hat einen Bahnhof.”*
- Die Regel
  - `verbunden(X, Y) :- bahnhof(X), bahnhof(Y).`steht für die Implikation
  - *“Wenn die Stadt X einen Bahnhof hat und wenn die Stadt Y einen Bahnhof hat, dann sind X und Y miteinander verbunden.”*
- Die linke Seite einer Regel wird Kopf, die rechte Seite Körper genannt.



## ▪ Interpretation als „Folgerung“

- Eine Regel wird so interpretiert, dass der Kopf gilt, wenn alle Bedingungen des Körpers wahr sind.
- Ein Faktum ist bedingungslos wahr.
  - Fakten können deshalb formal als Regeln mit dem Körper *true* betrachtet werden: `bahnhof(zürich) :- true.`

## ▪ Interpretation als „Beweis“

- `bahnhof(zürich)` .
  - *Um zu beweisen, dass Zürich einen Bahnhof hat, ist nichts zu beweisen.*
- `verbunden(X, Y) :- bahnhof(X), bahnhof(Y)` .
  - *Um zu beweisen, dass X und Y verbunden sind, beweise, dass X einen Bahnhof hat, und dann, dass Y einen Bahnhof hat.*



- Die Menge aller Prolog-Klauseln bildet das Prolog-Programm.
  - `bahnhof (zürich) .`
  - `bahnhof (bern) .`
  - `verbunden (X, Y) :- bahnhof (X) , bahnhof (Y) .`
- Ein Prolog-Programm beschreibt durch seine Fakten und Regeln einen bestimmten Sachverhalt.
- Man kann sich über diesen Sachverhalt informieren, indem man Anfragen stellt, wie z.B.
  - „*Sind Zürich und Bern miteinander verbunden?*“
  - formal in Prolog: `?- verbunden (zürich, bern) .`
- Der Prolog-Interpreter (oder Prolog-Compiler) beweist nun, dass die Anfrage eine logische Konsequenz des gegebenen Prolog-Programms ist, und antwortet
  - `yes`



- Noch einmal das Programm
  - `bahnhof (zürich) .`
  - `bahnhof (bern) .`
  - `verbunden (X, Y) :- bahnhof (X) , bahnhof (Y) .`
  
- Die Beweise sind konstruktiv und können deshalb auch für Anfragen nach Unbekanntem verwendet werden:
  - Auf die Anfrage
    - `?- verbunden (bern, S) .`
  - antwortet der Interpreter
    - `S = zürich`
  - Die Variable *S* wird an die Konstante *zürich* gebunden und als Resultat der Berechnung ausgegeben.
  
- Alternative Resultate durch die Eingabe von Strichpunkt `„ ; “`



- SWI-Prolog
- Eingabeaufforderung `?-`
- Eingabezeile beenden `?- blablabla. (Punkt!)`
- Programm laden `?- consult('filename.pl').`
- Programm abändern `?- edit.`
- Allgemeine Hilfe `?- help.`
- Hilfe z.B. zu „consult“ `?- help(consult) .`
- Beenden `?- halt.`



- Um eine Anfrage zu beantworten, sucht der Prolog-Interpreter im gegebenen Programm von oben nach unten nach einer Klausel, deren Kopf sich mit der Anfrage syntaktisch zur Deckung bringen lässt.
  - Handelt es sich um ein Faktum, ist der Beweis gelungen.
  - Handelt es sich um eine Regel wird die Anfrage durch die Bedingungen des Klauselkörpers ersetzt, die der Interpreter dann nacheinander von links nach rechts auf die gleiche Weise zu beweisen versucht.

▫ **Beispiel:**

Stellt man z.B. die Anfrage

◦ ?- `verbunden(bern, S)` .

so findet der Interpreter die Klausel

◦ `verbunden(X, Y) :- bahnhof(X), bahnhof(Y)`

➡ Der Kopf dieser Klausel „paßt“ auf die Anfrage (vergl. „matching“ Kap. 3.6) mit  $\sigma = [\text{bern}/X, S/Y]$



- Nach der Ersetzung durch den Körper der Klausel lautet die neue Anfrage mit zwei Teilanfragen
  - ?- `bahnhof (bern) , bahnhof (S) .`
- Der Interpreter versucht, diese beiden Teilanfragen – auch **Ziele** genannt – von links nach rechts zu beweisen. Das erste Ziel wird durch das gleichlautende Fakt
  - `bahnhof (bern)` ✓im Programm bewiesen, und es bleibt noch die Anfrage
  - ?- `bahnhof (S) .`
- Wieder oben im Programm beginnend, findet der Interpreter das Fakt
  - `bahnhof (zürich)`das sich mit dem Ziel durch  $\sigma = [\text{zürich}/S]$  passen lässt. Dadurch wird auch die Variable *S* der ursprünglichen Anfrage an *zürich* gebunden. Nun ist keine weitere Anfrage übrig. Der Beweis ist gelungen und der Wert von *S* wird als Ergebnis ausgegeben.
  - `S = zürich` ✓



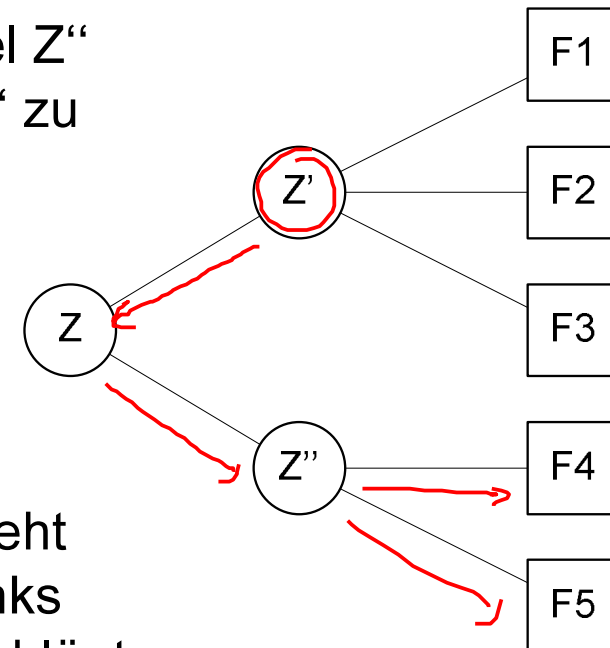
- Die Bindung von Variablen geschieht durch *Unifikation*.



- Beispiel
  - `bahnhof (zürich) .` ↓
  - `bahnhof (bern) .`
  - `?- bahnhof (X) .`
  - Antwort: `X = zürich`
- Es gibt noch eine weitere Möglichkeit, das Ziel auf Fakten zu passen:
  - `X = bern`
- Nach Ausgabe einer Lösung können mittels „;“ weitere Lösungen verlangt werden. Dazu macht der Interpreter die Entscheidung für
  - `bahnhof (zürich)`rückgängig und sucht die nächste Klausel, deren Kopf auf das Ziel paßt. Dieser Vorgang wird **Backtracking** genannt.
- Im Beispielprogramm findet der Interpreter
  - `bahnhof (bern) .`und führt die Berechnung mit der Bindung `Y=bern` fort.



- Der Prolog-Interpreter sucht so lange eine Klausel, bis er eine gefunden hat, deren Kopf auf das gegenwärtige Ziel  $Z'$  paßt.
- Kann keine (weitere) solche Klausel gefunden werden, wird versucht, eine alternative Klausel  $Z''$  für das vorhergehende Ziel  $Z$  zu finden und  $Z''$  zu beweisen.
- Kann der Prolog-Interpreter keine (weitere) Möglichkeit finden, das eingegebene Ziel zu beweisen, antwortet er
  - **No (more) answers**
- Wenn eine Anfrage aus mehreren Zielen besteht beweist der Prolog-Interpreter die Ziele von links nach rechts. Falls einer der Teilbeweise fehlschlägt, wird automatisch Backtracking zum vorhergehenden Ziel ausgelöst.



- Es kann passieren, dass die Antwort, die Prolog gibt, nicht der erwarteten entspricht
  - Die komplette Berechnung eines Ausdrucks kann mit dem Befehl trace überwacht werden.
  - Einzelne Ausdrücke können mit
    - `?- spy(ausdruck/stelligkeit) .`  
überwacht werden.

$\Sigma^{(0)}$   $\Sigma^{(1)}$   $\Sigma^{(2)}$

## ▪ Stelligkeit

- Anzahl der Variablen im Kopf der Klausel

▫ trace	trace/0
▫ bahnhof(bern)	bahnhof/1
▫ verbunden(X,Y)	verbunden/2
<i>verbunden(X,Y,Z)</i>	<i>verbunden/3</i>



- Programm:
  - `bahnhof(zürich) .`
  - `bahnhof(bern) .`
  - `bahnhof(genf) .`
  - `geschlossen(genf) .`
  - `verbunden(X, Y) :- bahnhof(X), bahnhof(Y) .`
  - `erreichbar(X) :- verbunden(X,   ), not(geschlossen(X)) .`
  
- Überwachen, ob Prolog prüft, ob ein Bahnhof geschlossen ist:
  - `spy(geschlossen/1) .` oder
  - `spy(geschlossen) .`
  
- Anfragen
  - `verbunden(bern, genf) .`
  - `erreichbar(bern) .`



- „Prolog Tutorial“ von Norbert E. Fuchs (Grundlage dieser Folien)  
[http://www.ifi.unizh.ch/req/courses/logische\\_programmierung/ws03/documents/Prolog\\_Tutorial.pdf](http://www.ifi.unizh.ch/req/courses/logische_programmierung/ws03/documents/Prolog_Tutorial.pdf)
- „Prolog Programming – A First Course“ von Paul Brna  
<http://computing.unn.ac.uk/staff/cgpb4/prologbook/book.html>  
insbes. „Interlude: Practical Matters“ ab S. 41
- SWI-Prolog <http://www.swi-prolog.org>



- Sowie die Musterlösungen finden Sie unter

**<http://www.infoeins.de/uebungsblaetter.php>**

