Miriam Butt und Maribel Romero
G220/5109, G212/2728
{miriam.butt|maribel.romero}@uni-konstanz.de

Advanced Computational Semantics
Sommersemester 2008, Hauptseminar

Übung 2, 24.04.2008

- erfahren, wie man Regeln in Prolog implementiert (Modus ponens)
- Variablen in Prolog kennen lernen
- Rekursion in Prolog anwenden
- das in Aufgabe 4) b) angefertigte Programm bitte an sebastian.sulger@uni-konstanz.de schicken

Aufgabe 1)

a) Melde dich wie gewohnt mit dem 'cluser' am Mac an und starte X11.
b) Wechsle mit 'cd AdvCompSem' in das Kurs-Verzeichnis.
c) Ist dort noch kein Verzeichnis mit deinem Namenskürzel vorhanden, kannst du mit 'mkdir deinKürzel' eines anlegen.
d) Wechsle in dein Verzeichnis und starte den emacs mit 'emacs &'; öffne dann eine neue Shell im emacs: 'escape' + 'x', 'shell' eintippen und 'enter' drücken.
e) Danach legst du mit einem Klick auf das Blatt-Symbol oben links eine neue Datei an; nenne diese z.B. 'uebung2.pl'.

Aufgabe 2)

a) Wir wollen uns jetzt anschauen, wie man in Prolog einfache Schlussregeln beschreibt. Dazu tippe bitte folgende kleine Wissensbasis ein:

```
frech(bart).
faul(homer).
wuergt(homer,bart) :- frech(bart).
schlaeft(homer) :- faul(homer).
```

b) Teste das Programm mit folgenden Anfragen:

```
frech(bart).
faul(homer).
wuergt(homer,bart).
schlaeft(homer).
```

Prolog benutzt den logischen Schluss des Modus ponens; dieser besagt: Gilt *Wenn A, dann B* und gilt *A*, dann gilt auch *B*. Prolog weiß, dass gilt `frech(bart)`, es weiß auch, dass gilt `wuergt(homer,bart) :- frech(bart)`, also kann es `wuergt(homer,bart)` beweisen. Zum Format der Regeln: Was links von `:-` steht, wird als "Kopf" bezeichnet, alles rechts von `:-` heißt "Körper". Die vierte Zeile lässt sich daher etwa so übersetzen: "Homer schläft dann, wenn gilt, dass er faul ist."

Aufgabe 3)

a) Wir wollen nun noch kurz einen Blick auf Variablen unter Prolog werfen, bevor wir uns der Rekursion zuwenden. Variablen sind Zeichenketten aus Großbuchstaben, Kleinbuchstaben, Ziffern und Unterstrichen, die entweder mit einem Großbuchstaben oder mit einem Unterstrich beginnen (`X, Y, Bla, _Bla, Tag14, Output` sind also Prolog-Variablen).
b) Erweitere das Programm aus Aufgabe 2) a) um folgende Zeile:

```
sauerAuf(X,Y) :- wuergt(Y,X).
```

Dann lade das Programm neu (du brauchst dazu swi-prolog nicht zu beenden, es reicht, die Wissensbasis neu einzulesen) und teste es mit dem Query

```
sauerAuf(X,Y).
```

Damit fragst du Prolog: "Wer ist sauer auf wen?" Prolog wird der Variablen `X` den Wert `bart` und der Variablen `Y` den Wert `homer` zuweisen, da es beweisen kann, dass gilt `wuergt(homer,bart)`. Man sagt: `X` *unifiziert* mit `bart` und `Y` *unifiziert* mit `homer`.

Aufgabe 4)

a) Wir wollen Rekursion in Prolog anwenden. Betrachte dazu folgendes Beispielprogramm.

```
verdaut(X,Y) :- hat_gegessen(X,Y).
verdaut(X,Y) :-
hat_gegessen(X,Z),
verdaut(Z,Y).

hat_gegessen(frosch,moskito).
hat_gegessen(storch,frosch).
hat_gegessen(schakal,storch).
```

Das Programm besteht aus zwei Regeln und ein paar einfachen Fakten. Die erste Regel ist einfach; sie sagt nur aus, dass wenn X Y gegessen hat, dann verdaut X Y. Die zweite Regel ist rekursiv; sie sagt aus, dass X Y auch dann verdaut, wenn gilt, dass X Z gegessen hat und Z Y verdaut. Wichtig hierbei ist, dass die zweite Regel sowohl im Kopf als auch im Körper das Prädikat `verdaut` enthält. Teste das Programm mit ein paar simplen Abfragen:

```
verdaut(frosch,moskito).
verdaut(schakal,moskito).
```

und mit einer generellen Abfrage über alle Y, die von X verdaut werden:

```
verdaut(X,Y).
```

Wenn Prolog eine Antwort gegeben hat, rufe die nächste ab mit ';' (steht für 'oder') und 'enter'.
b) Die folgende Wissensbasis ist gegeben:

```
direktflug(new_york,frankfurt).
direktflug(frankfurt,muenchen).
direktflug(muenchen,kairo).
direktflug(kairo,singapur).
direktflug(singapur,sydney).
direktflug(sydney,auckland).
```

Natürlich kann man die Flüge verketten. Schreibe ein Prädikat `flugVonNach` (direkt über die Wissensbasis in die gleiche Datei), das zeigen kann, ob man mit diesen Flügen von Stadt A (über Zwischenstationen) nach Stadt B kommt. Das ganze Programm schickst du an sebastian.sulger@uni-konstanz.de.

Aufgabe 5)

a) Aus dem Buch "Learn Prolog Now!" sollten noch folgende Übungen erledigt werden (wie schon im Kurs besprochen): Aufgabe 1.4 und 1.5 (Seite 15 im Buch). Die Lösungen zu Aufgabe 1.4 schickt ihr bitte ein. Bei Aufgabe 1.5 müsst ihr nur Prolog einige Anfragen stellen; die Lösungen dazu bitte nicht einschicken.