Einführung in das Programmieren – Prolog
Sommersemester 2006

Teil 7: Wissensbasis

Version 1.0

Gliederung der LV

Teil 1: Ein motivierendes Beispiel

Teil 2: Einführung und Grundkonzepte
- Syntax, Regeln, Unifikation, Abarbeitung

Teil 3: Arithmetik

Teil 4: Rekursion und Listen

Teil 5: Programmfluß
- Negation, Cut

Teil 6: Verschiedenes
- Ein-/Ausgabe, Programmierstil

Teil 7: Wissensbasis
- Löschen und Hinzufügen von Klauseln

Teil 8: Fortgeschrittene Techniken
- Metainterpreter, iterative Deepening, PTTP, Differenzlisten, doppelt verkettete Listen

Hinzufügen von Klauseln zur Wissensbasis

- Wissensbasis kann während der Abarbeitung eines Programms verändert werden
- Standardprädikate zum Erweitern von Prologprogrammen

  assert(C): Fügt eine Klausel `C` am Ende der entsprechenden Prozedur ein

  assertz(C) : Äquivalent zu `assert(C)`

  asserta(C) : Fügt Klausel `C` als erste Klausel der Prozedur ein
- Das Einfügen der Prädikate wird bei nachfolgendem Backtracking *nicht rückgängig gemacht*.

- Es können beliebige Prädikate hinzugefügt werden, d.h. auch solche, die bereits per File eingelesen wurden
- . . . und so kann sich auch das laufende Programm selbst ändern
- Prolog hat jedoch einen Schutzmechanismus, um unbeabsichtigte Änderungen zu vermeiden $\longrightarrow$ solche Prädikate bei der Definition als dynamisch markieren (dynamic/1)

Löschen von Klauseln aus Wissensbasis

- Streichen von Klauseln auch möglich

  retract(C): Streicht *erste* Klausel des Programms, die mit der Parameterklausel `C` unifiziert
  Bei Backtracking werden weitere unifizierende Klauseln gestrichen
  falls `retract(C)` gelingt bleiben vorgenommene Variablenbindungen erhalten

  retractall(C): Streicht alle matchenden Klauseln aus dem Speicher

  abolish(Name) : streicht *alle* Klauseln des Prädikats mit dem *Namen* `Name`

  abolish(Name,Arity) : streicht alle Klauseln des Prädikats mit dem Funktor `Name` und der Stelligkeit `Arity`

Zugriff auf Prologprogramme

clause(Head,Body) überprüft, ob eine Klausel in der Wissensbasis ist, deren Kopf mit erstem und deren Rumpf mit zweitem angegebenen Argument unifiziert werden können.
Erstes Argument muß Funktor und Stelligkeit erkennbar machen.

```
append([], L, L]).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).

clause(append(A,B,C),Y).
A = [], B = _3, C = _3, Y = true ;
...
```

listing druckt aktuellen Inhalt der Datenbank (für ein Prädikat)

Metaprädikate

Prädikatre sind nichts weiter als Prologterme, d.h. auch Prologterme können als Prädikat interpretiert werden

call, apply Interpretiere gegebenen Term als Anfrage und führe sie aus

- D.h., Anfragen können zur Laufzeit zusammengebastelt und dann ausgeführt werden