Dynamische Prädikate

Übersicht

- Wissensbasis verändern
  - Statische vs. dynamische Prozeduren
- Deklaration dynamic/1
- Klauseln hinzufügen
  - asserta/1, assertz/1
- Klauseln entfernen
  - retract/1
- Prädikate löschen
  - abolish/1
- Klauseln finden
  - clause/2

Wissensbasis verändern

Statische vs. dynamische Prädikate

- Beim Einlesen eines Programmtexts werden die Prädikatsdefinitionen in **statische Prozeduren** (Programm-Stückchen) verwandelt. Diese können später nicht mehr verändert, sondern nur noch aufgerufen (ausgeführt) werden. Das ist **effizient**!
- **Dynamische Prozeduren** dagegen können während des Programmablaufs gelöscht oder erschaffen werden. Dies entspricht dem Löschen oder Hinzufügen von Klauseln auf der Ebene des Programmtexts. Das ist **flexibel**!
  - Im Folgenden wird der Begriff »Klausel« weiterhin zweideutig sowohl für die textuelle Definition als auch für die entsprechende Prozedur verwendet.

Als dynamisch deklarieren…

- **dynamic(*Funktor*/*Stelligkeit*)**
  - Damit eine Prädikatsdefinition zu dynamischen Prozeduren wird, muss sie als **dynamisch deklariert** sein.
  - Die Deklaration muss immer **vor** der ersten Klausel des entsprechenden Prädikats notiert sein.
    - Guter Programmierstil deklariert auch dynamische Prädikate, für die gar keine textuelle Klausel im Programm vorkommt.
  - Beim Verarbeiten von `:- dynamic(`*`pred/n`*`).` werden alle existierenden Klauseln für *pred*/*n* gelöscht.
    - Mehrfache dynamic-Deklarationen für das gleiche Prädikat haben keinen Effekt.

```
:- dynamic(person/1).

person(klara).
person(hans).
```

Inhalt von Datei: "meier.pl"

Klauseln hinzufügen

- **asserta(Klausel)**
  - **fügt** *Klausel* **am Anfang** der Klauseln des dynamischen Prädikats mit gleichem Funktor und Stelligkeit **hinzu**.
  - Falls für das Prädikat noch keine Klauseln existieren, dann gilt es automatisch als dynamisch deklariert.
  - Wenn für das Prädikat bereits statische Klauseln existieren, gibt es eine Fehlermeldung.

```
?- consult('meier.pl').
yes
?- asserta(person(kevin)).
yes
?- person(X).
X = kevin ;
X = klara ;
X = hans ;
no
?- asserta(person(gabi)).
yes
?- person(X).
X = gabi ;
...
```

Klauseln hinzufügen

- **assertz(Klausel)**
  - funktioniert wie asserta/1, **fügt** aber *Klausel* **am Ende** der Klauseln des Prädikats mit gleichem Funktor und Stelligkeit **hinzu**.
  - Wie bei asserta/1 kann das Hinzufügen bei Backtracking nicht rückgängig gemacht werden!
  - Damit kann über Backtracking hinweg gespeichert werden!

```
?- consult('meier.pl').
yes
?- assertz(person(kevin)).
yes
?- person(X).
X = klara ;
X = hans ;
X = kevin ;
no
?- asserta(person(gabi)),
   fail.
no
?- person(gabi).
yes
```


Klauseln löschen

- **retract(Klausel)**
  - **entfernt** die erste **unifizierbare Klausel** aus der Wissensbasis.
  - entfernt je eine weitere Klausel beim Backtracking.
  - Das Löschen wird beim Backtracking nicht rückgängig gemacht!

```
?- consult('meier.pl').
yes
?- retract(person(hans)).
yes
?- person(X).
X = klara ;
no
?- retract(person(_)),
   fail.
no
?- person(X).
no
```


Prädikate löschen

- **abolish(Funktor/Stelligkeit)**
  - tilgt sämtliche Spuren eines dynamischen Prädikates.
  - eliminiert auch die Deklaration als dynamisches Prädikat.
    - ISO-Prolog-Konformität: In SICStus Prolog können so auch statische Prozeduren gelöscht werden.
  - Kein Wiederherstellen bei Backtracking!

```
?- consult('meier.pl').
yes
?- person(X).
X = klara ;
X = hans ;
no
?- person(gabi).
no
?- abolish(person/1).
yes
?- person(gabi).
! Existence error in user:person/1
! procedure user:person/1
  does not exist
! goal:  user:person(gabi)
```


Klauseln finden

- **clause(*Kopf*, *Rumpf*)**
  - sucht in der Wissensbasis Klauseln dynamischer Prädikate, deren Kopf mit ***Kopf*** und deren Rumpf mit ***Rumpf*** unifiziert.
  - Der Rumpf eines Faktums ist `true`.
  - Falls kein entsprechendes Prädikat existiert, schlägt clause/2 fehl.

```
?- consult('meier.pl').
yes
?- clause(person(X), R).
X = hans,
R = true
yes
?- asserta(
     ( person(X):-
         lebt(X) )).
yes
?- clause(person(X), R).
R = lebt(X)
yes
```