

# Manipulation der Datenbasis und Aufsammeln von Lösungen

---

Heute:

- eingebaute Prolog-Prädikate zur Manipulation der Datenbasis
- Programmieretechnik: Memoisierung
- eingebaute Prolog-Prädikate zum Aufsammeln von Lösungen

# assert/1

Mit dem Prädikat **assert/1** ist es möglich Prolog Klauseln dynamisch zur Datenbasis hinzuzufügen.

```
?- assert(happy(mia)).
```

```
?- assert( (s(X):- a(X)) ).
```

# retract/1

Mit dem Prädikat **retract/1** ist es möglich Prolog Klauseln aus der Datenbasis zu löschen.

```
?- retract(happy(mia)).
```

```
?- retract( (s(X):- a(X)) ).
```

# asserta/1 und assertz/1

Mit **assert/1** haben wir keine Kontrolle darüber, wo Prolog die Klauseln in die Datenbasis hinschreibt.

- **asserta/1** schreibt Klauseln an den Anfang der Datenbasis.
- **assertz/1** schreibt Klauseln an das Ende der Datenbasis.

# Memoisierung

---

*Memoisierung* oder *caching* ist eine Programmier-technik, die darin besteht, dass man die Zwischenergebnisse einer Berechnung speichert, damit sie nicht neu berechnet werden müssen.

Dieses Vorgehen macht Programme (die häufiger die gleichen Berechnungen ausführen müssen) schneller.

# Fibonacci Zahlen

Die Fibonacci Zahlen:

0 1 1 2 3 5 8 13 21 34 ...

Die mathematische rekursive Definition:

$$\text{fib}(0) = 0$$

$$\text{fib}(1) = 1$$

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2), \quad n > 1$$

# Fibonacci Zahlen in Prolog

---

```
%% fib(0)=0
fib(0,0).
%% fib(1)=1
fib(1,1).
%% fib(n)=fib(n-1)+fib(n-2), für n> 1
fib(N,Res) :- N >1,
               N1 is N-1,
               N2 is N-2,
               fib(N1,R1),
               fib(N2,R2),
               Res is R1+R2.
```

# Fibonacci Zahlen in Prolog: Auswertung

---

?- fib(4,N).

N=3

?- fib(18,N).

N= 2584

?- fib(40,N).

N= 102334155

**Problem:** die Berechnung dauert zu lange, weil der Berechnungsbaum exponentiell wächst.

# Fibonacci Zahlen mit Memoisierung

---

```
:- dynamic(fib/2).
```

```
fib(0,0).
```

```
fib(1,1).
```

```
fibM(N,Res):- fib(N,Res),!.
```

```
fibM(N,Res) :- N > 1,  
               N1 is N-1,  
               N2 is N-2,  
               fibM(N1,R1),  
               fibM(N2,R2),  
               Res is R1+R2,
```

```
asserta(fib(N,Res)).
```

# Fibonacci Zahlen in Prolog: Auswertung

---

?- fib(50,N).

$$N = 1.25863 * 10^{10}$$

?- fib(70,N).

$$N = 1.90392 * 10^{14}$$

?- fib(1000,N).

$$N = 4.34666 * 10^{208}$$

Die Berechnung ist schnell (wenn kleinere Zwischenergebnisse vorher schonmal berechnet wurden).

# Aufsammeln von Lösungen

---

Das Prädikat **findall/3** sucht nach allen möglichen Beweisen für *Goal* und sammelt in der Liste *List* alle Instanziierungen von *Object* in diesen Beweisen.

*findall(Object,Goal,List).*

## findall/3

```
n(1).           n(2).  
n(3).           n(4).
```

```
?- findall(X, n(X), List).  
List=[1,2,3,4]
```

```
?- findall(number(X),n(X),List).  
List=[number(1),number(2),number(3),number(4)]
```

```
?- findall(X,(n(X),X>2),List).  
List=[3,4]
```

```
?- findall(X,(n(X),X>4),List).  
List=[]
```

```
?- findall(Y,n(X),List).  
  
List=[_I1, _I2, _I3, _I4]
```

```
person(hans,30).  
person(martin,28).  
person(julia, 45).  
person(hans, professor).  
person(martin, arzt).  
person(julia, anwalt).
```

```
?- findall(X, person(X,Y), List).  
List = [hans, martin, julia, hans, martin, julia]
```

```
?- findall(Y, person(X,Y), List).  
List = [30, 28, 45, professor, arzt, anwalt]
```

## **bagof/3**

---

**bagof** macht das gleiche wie **findall**, aber falls in *Goal* Variablen sind, die in *Object* nicht vorkommen, gruppiert **bagof** die Lösungen entsprechend der verschiedenen Instantiierungen dieser Variablen.

*bagof(Object, Goal, List).*

## bagof/3

```
?- bagof(Y, person(X,Y), List).
```

```
X = hans
```

```
List = [30, professor] ;
```

```
X = martin
```

```
List = [28, arzt] ;
```

```
X = julia
```

```
List = [45, anwalt] ;
```

```
No
```

## bagof/3

```
?- bagof(Y, X^ person(X,Y), List).
```

```
List = [30, 28, 45, professor, arzt, anwalt]
```

```
?- findall(List, bagof(Y, person(X,Y), List), L).
```

```
L = [[30,professor], [28, arzt], [45,anwalt]]
```

```
?- findall(p(X,List), bagof(Y, person(X,Y), List),  
L).
```

```
L = [p(hans, [30, professor]),
```

```
      p(martin, [28, arzt]),
```

```
      p(julia, [45, anwalt])]
```

## setof/3

**setof** arbeitet wie **bagof**, aber zusätzlich ist die Ergebnisliste sortiert und enthält kein Element mehrfach.

*setof(Object,Goal,List)*

```
?- setof(Y, person(X,Y), List).
```

```
X = hans
```

```
List = [30, professor] ;
```

```
X = martin
```

```
List = [28, arzt] ;
```

```
X = julia
```

```
List = [45, anwalt] ;
```

```
No
```

## setof/3

```
?- findall(X, person(X,Y), List).
```

```
List = [hans, martin, julia, hans, martin, julia]
```

```
?- bagof(X, Y^ person(X,Y), List).
```

```
List = [hans, martin, julia, hans, martin, julia]
```

```
?- setof(X, Y^ person(X,Y), List).
```

```
List = [hans, julia, martin]
```

# Aufgaben

1. Gegeben die Wissensbasis auf Folien 13. Nehmt an, ihr macht nacheinander die folgenden Anfragen. Wie sieht die Wissensbasis nach jeder Anfrage aus?  

```
?- asserta(person(julia,55)).  
?- retract(person(julia,X)), fail.  
?- retract(person(X,Y)), atom(Y).
```
2. Formuliere eine Prologanfrage, die als Ergebnis die Liste der Berufe aller Personen in der Datenbasis auf Folie 13 ausgibt.
3. Formuliere eine Prologanfrage, die die Ergebnisse von Aufgabe 2 alphabetisch geordnet ausgibt.
4. Definiere ein Prologprädikat, das die folgende Funktion berechnet:  
$$f(0) = 0, \quad f(1) = 1, \quad f(2) = 2$$
$$f(n) = f(n-3) + f(n-2) + f(n-1), \quad n > 2.$$
  
Das Prädikat soll Zwischenergebnisse speichern.  
(Verwende die Memoisierungstechnik wie in fib)

# Zusammenfassung

---

Heute haben wir gesehen

- **assert/1, retract/1,**
- Memoisierung
- **findall/3, bagof/3, setof/3**

Nächste Woche Montag: Klausur!

**Übungsaufgaben:** Auf der Webseite. Wer noch Punkte zur Klausurzulassung braucht, sollte bis **Sonntagabend** abgegeben haben.