

# Tutorium 06: Prolog

---

Paul Brinkmeier

30. November 2021

Tutorium Programmierparadigmen am KIT

# Wiederholung

---

- Terme  $t$ : Variable ( $x$ ), Funktion ( $\lambda x.t$ ), Anwendung ( $t\ t$ )
- $\alpha$ -Äquivalenz: Gleiche Struktur
- $\eta$ -Äquivalenz: Unterversorgung
- Freie Variablen, Substitution, RedEx
- $\beta$ -Reduktion:  
 $(\lambda p.b)\ t \Rightarrow b[p \rightarrow t]$

- Auswertungsstrategien (von lässig nach streng):
  - Volle  $\beta$ -Reduktion
  - Normalreihenfolge
  - Call-by-Name
  - Call-by-Value
- Datenstrukturen:
  - Church-Booleans
  - Church-Zahlen
  - Church-Listen
- Rekursion durch Y-Kombinator

# Übungsblatt 4

---

# Y-Kombinator

---

# Einführung in Prolog

---



## SWI Prolog

- Prolog ist eine Programmiersprache, wenn auch eine seltsame
- $\leadsto$  gut wird man durch Übung
- Zum Üben:
  - SWI-Prolog — gängige Prolog-Umgebung
  - [SWISH](#) — SWI-Prolog Web-IDE zum Testen
  - VIPR, VIPER — PSE-Tools des IPD, auf der [Seite der Übung](#) verlinkt



```
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).  
parent(X, Y) :- mother(X, Y).  
parent(X, Y) :- father(X, Y).  
  
mother(inge, emil).  
mother(inge, petra).  
father(emil, kunibert).
```

?- grandparent(inge, kunibert).  $\leadsto$  yes.

## Prolog — Regelsysteme als Programmiersprache

```
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).  
parent(X, Y) :- mother(X, Y).  
parent(X, Y) :- father(X, Y).  
  
mother(inge, emil).  
mother(inge, petra).  
father(emil, kunibert).
```

---

mother(inge, emil)

---

parent(inge, emil)

---

---

father(emil, kunibert)

---

parent(emil, kunibert)

---

grandparent(inge, kunibert)

```
a(b, c, d).  
defg.  
bintree(bintree(1, 2), bintree(3, bintree(4, 5))).  
list(cons(1, cons(2, cons(3, nil)))).  
'Abcd'('X', 'Y', 'Z').
```

- Funktor  $\approx$  Name + Liste von Prolog-Ausdrücken
- Liste leer  $\leadsto$  „Atom“
- Name wird immer klein geschrieben
  - Großbuchstaben: bspw. 'List'
- Auch mathematische Ausdrücke sind Funktoren:  
 $17 + 25 \approx '+'(17, 25)$

```
?- X = pumpkin.  
?- Y = honey_bunny.  
?- Z = vincent.  
?- [A, B, C] = [1, 2, 3].  
?- f(L, rechts) = f(links, R)
```

- Variablen werden immer groß geschrieben
- = ist nicht Zuweisung, sondern Unifikation
- Unifikation  $\approx$  (formales) Pattern-Matching

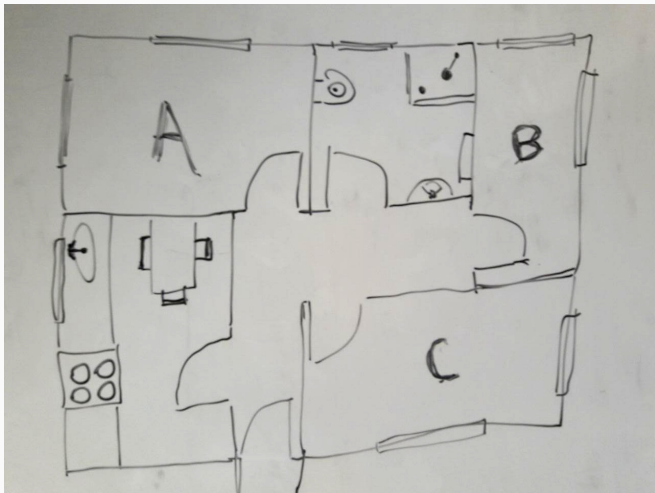
```
main :-  
    [A, B, C] = [1, 2, 3],  
    max(A, C, X),  
    Y is X * 2,  
    Y > 0,  
    not(Y > 10),  
    !.
```

- Funktionsaufruf  $\approx$  „Zielerfüllung“ in Prolog
- Mögliche Ziele:
  - Unifikationsziel
  - Funktorziel
  - Arithmetische Zuweisung
  - Arithmetischer Vergleich
  - Nicht-Erfüllung
  - Cut

- Prolog-„Programme“  $\approx$  Datenbanken
- Ausführung  $\approx$  Abfrage in der Datenbank
- Datenbank-Inhalt: Regeln, bestehend aus:
  - Regelkopf — Ein Funktor  $\leadsto$  kann auch Atom sein
  - Teilziele — Liste von Zielen, um diese Regel zu erfüllen
  - Keine Teilziele  $\leadsto$  Fakt

# Prolog-Aufgaben

---





- Alice, Bob und Carl ziehen in eine WG
- Die drei sind Mathematiker;  
jeder will eine eigene Zahl von 1 bis 7 für sein Zimmer
- Die Summe der Zahlen soll 12 sein
- Alice mag keine ungeraden Zahlen

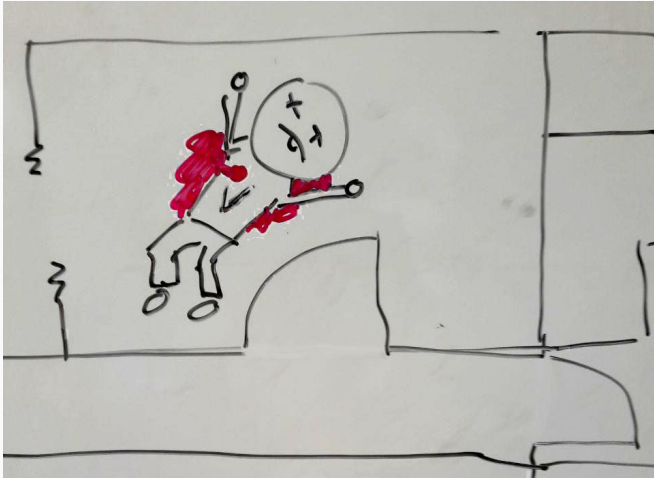
Findet alle 14 möglichen Kombinationen, die Zimmer zu nummerieren.

```
% mathematiker_wg.pl

alice(2).
alice(4).
alice(6).
...

nummerierung(A, B, C) :-
    alice(A),
    bob(B),
    carl(C),
    ...
    12 ::= A + B + C.
```

# Detektivrätsel



Im Fall des Mordes an ihrem Nachbarn Victor sind nun Alice, Bob und Carl die einzigen Verdächtigen und Zeugen.

- Alice:
  - Bob war mit dem Opfer befreundet.
  - Carl und das Opfer waren verfeindet.
- Bob:
  - Ich war überhaupt nicht daheim!
  - Ich kenne den garnicht!
- Carl:
  - Ich bin unschuldig!
  - Wir waren zum Zeitpunkt der Tat alle in der WG.

```
% detektiv.pl

aussage(alice, freund(bob)).
aussage(alice, feind(carl)).
...

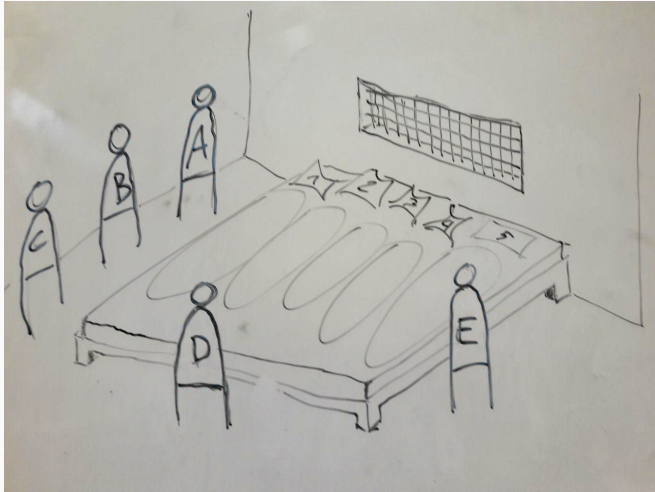
% Widersprüche

widerspruch(freund(X), feind(X)).
...
```

```
taeter(T) :-  
    delete([alice, bob, carl], T, Rest),  
    not(inkonsistent(Rest)).  
  
inkonsistent(Zeugen) :- ...
```

- delete/3 wurde in der Vorlesung definiert.
- Implementiert: inkonsistent/1  
Überprüft Aussagen von Zeugen paarweise auf Widerspruch

# Schlafplätze im Gefängnis



## Dinesman's multiple-dwelling problem

Bob kommt nun ins Gefängnis. Aaron, Bob, Connor, David und Edison müssen sich zu fünft ein sehr breites Bett teilen.

- Aaron will nicht am rechten Ende liegen
- Bob will nicht am linken Ende liegen
- Connor will an keinem der beiden Enden liegen
- David will weiter rechts liegen als Bob
- Connor schnarcht sehr laut;  
Bob und Edison sind sehr geräuschempfindlich
  - $\leadsto$  Bob will nicht direkt neben Connor liegen
  - $\leadsto$  Edison will nicht direkt neben Connor liegen

Wie können die 5 Schlafplätze verteilt werden?



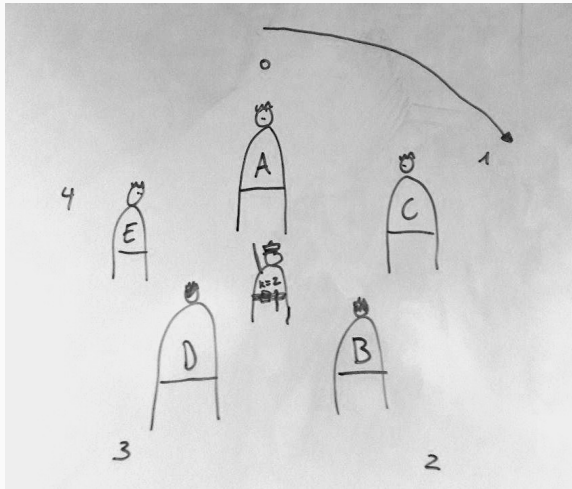
# Schlafplätze im Gefängnis

```
% schlafplaetze.pl

bett(X) :- member(X, [1, 2, 3, 4, 5]).

schlafplaetze(A, B, C, D, E) :-
    bett(A), bett(B), bett(C), bett(D), bett(E),
    distinct([A, B, C, D, E]),
    % weitere Tests
```

- Fügt weitere benötigte Tests ein
- Implementiert:
  - `distinct/1` prüft Listenelemente auf paarweise Ungleichheit
  - `adjacent/2` prüft, ob  $|A - B| = 1$



- Aaron, Bob, Connor, David und Edison sollen 4 Einheiten Putzdienst übernehmen
- Da sie sich nicht einigen können, wer aussetzen darf, wendet ein Wärter folgendes Vorgehen an:
  - Die fünf werden im Kreis aufgestellt
  - Der Wärter stellt sich in die Mitte
  - Beginnend bei 12 Uhr dreht er sich im Uhrzeigersinn und teilt jeden  $k$ -ten (bspw.  $k = 2$ ) Insassen zum Putzdienst ein
    - D.h. es werden immer  $k - 1$  Insassen übersprungen

An welcher Stelle muss Bob stehen, um nicht putzen zu müssen?

- Aaron, Bob, Connor, David und Edison sollen 4 Einheiten Putzdienst übernehmen
- Da sie sich nicht einigen können, wer aussetzen darf, wendet ein Wärter folgendes Vorgehen an:
  - Die fünf werden im Kreis aufgestellt
  - Der Wärter stellt sich in die Mitte
  - Beginnend bei 12 Uhr dreht er sich im Uhrzeigersinn und teilt jeden  $k$ -ten (bspw.  $k = 2$ ) Insassen zum Putzdienst ein
    - D.h. es werden immer  $k - 1$  Insassen übersprungen

An welcher Stelle muss Bob stehen, um nicht putzen zu müssen?

An welcher Stelle muss Bob bei 41 Insassen und  $k = 3$  stehen?

```
% putzdienst.pl

% Bspw.
% ?- keinPutzdienstFuer([a, b, c, d, e], 2, X)
keinPutzdienstFuer(L, K, X) :-
    Countdown is K - 1,
    helper(L, Countdown, K, X).

helper([X], _C, _K, X) :- !.
...
```

- Weitere Fälle für helper/4:
  - $C = 0 \leadsto$  Element entfernen
  - Ansonsten: Element hinten wieder anhängen

Zum Nachlesen und Vergleichen mit Lösungen in anderen Programmiersprachen:

- WG — [Rosetta Code: Department Numbers](#)
- Detektiv — [github.com/Anniepoo/prolog-examples](https://github.com/Anniepoo/prolog-examples)
- Schlafplätze — SICP, S. 418
- Putzdienst — [Rosetta Code: Josephus problem](#)