Kapitel 5

- Pattern search ←
- Production systems
- Blackboard

? a.

Intern

$\neg a$

1. Klausel $a \leftarrow b$ $\quad b \rightarrow a$ $\quad \neg b \vee a$

$(\neg a) \wedge (\neg b \vee a)$

Res

$\neg b$

2. Klausel $b.$ $\quad true \rightarrow b$

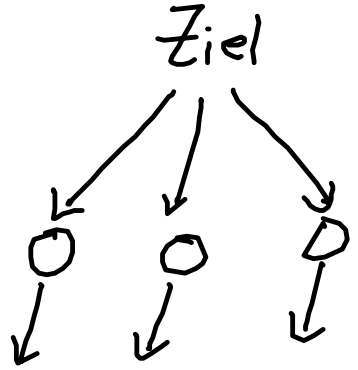
$(\neg b) \wedge (b)$

Widerspruch!

$\Rightarrow \neg a$ Widerspruch

$\Rightarrow a$ wahr

Goal directed search



Pattern search

? vater(X, adam).

vater(gott, adam).

⋮

Metainterpreter

? solve(vater(X, adam)).

↑

Prolog

1. Versuch

```
solve(true) :- !.
solve((A,B)) :- !, solve(A),
                   solve(B).
solve((A;B)) :- !, (solve(A);
                    solve(B)).
solve(A) :- system(A), call(A).
```

```
solve(A) :- clause(A,B),
            solve(B).
```

```
system(=(_,_)).
system(fail).
system(==(_,_)).
system(write(_)).
```

S → X = Y
I → =(X,Y)

1 == 2
==(1,2)

Syntax [vater (adam, abel).

Kopf Body

Intern [vater (adam, abel) :- true.]

? clause (vater (adam, abel), X).

X = true

? solve (vater (X, abel)).

X = adam

? solve (opa (X, isaac)).

⋮

? solve (fail).

↳ call (fail)

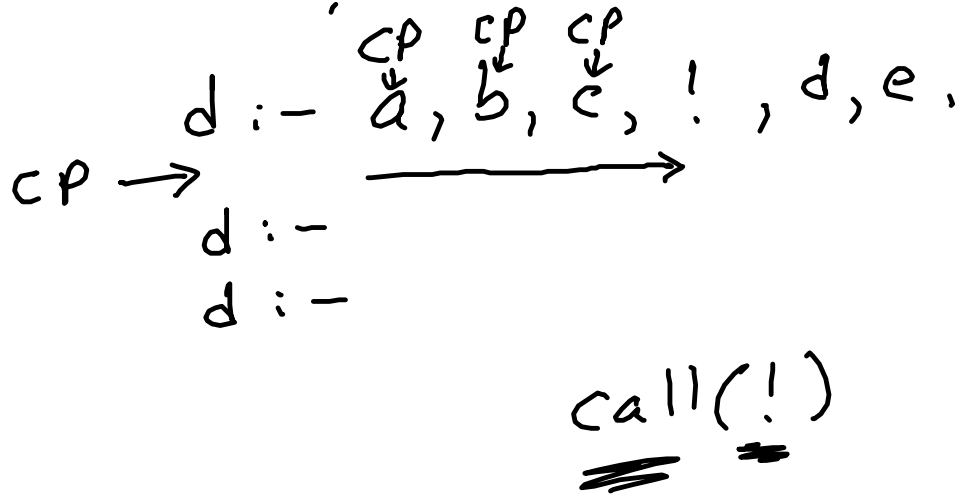
fail

? solve (solve (true)).

? solve (solve (solve (solve (true)))).

solve (solve (A)) :- solve (A).

— . —



call(!)

2. Versuch

cut?

Goal ↓ ↓ ← nach dem cut

```
solve(X) :- solve(X,C,R),
            (C==!,!,solve(R);
             true).
```

Syntax a,b,c,!,d,e

Intern (a,(b,(c,(!,(d,e)))))

Interpretiert ↓

C = !
R = (d,e)

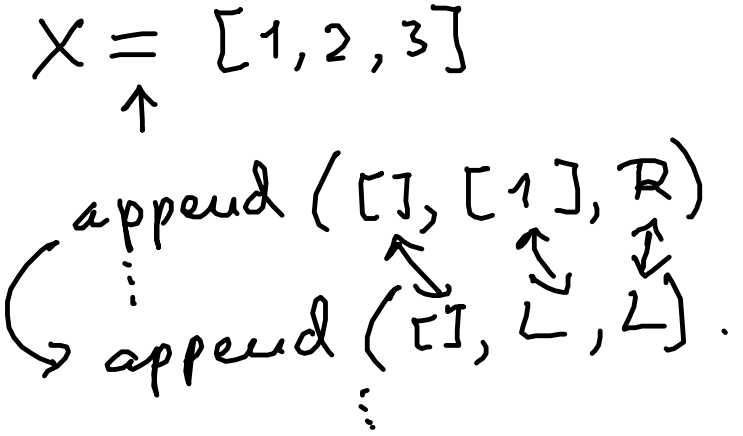
```
solve(true,_,_):-!.
solve(!,!,true):-!.
```

```
solve((A,B),X,Y):-!,
        solve(A,X,R),
        (X==!,!,Y=(R,B);
          solve(B,X,Y)).

solve((A;B),X,Y):-!,
    (solve(A,X,Y),(X==!,!;true);
      solve(B,X,Y)).

solve(A,-,-):-system(A),!,
                call(A).
solve(A,-,-):- clause(A,B),
              solve(B,X,R),
          (X==!,!,solve(R);true).
```

```
:-dynamic vater/2.
:-dynamic solve/3.
  :
```

```
:-dynamic solve/1
:-dynamic system/1
   ⋮
```

Unifikation



x(1) x(2) x(100)

unify(X, Y, Sublist)

? unify(x(1), 2, L).

L = [x(1)/2]

```
unify( X,Y, Sublist) :-
        unify(X, Y, [], Sublist).
```

bis hierher

nach X=Y

unify(x(N), x(N),

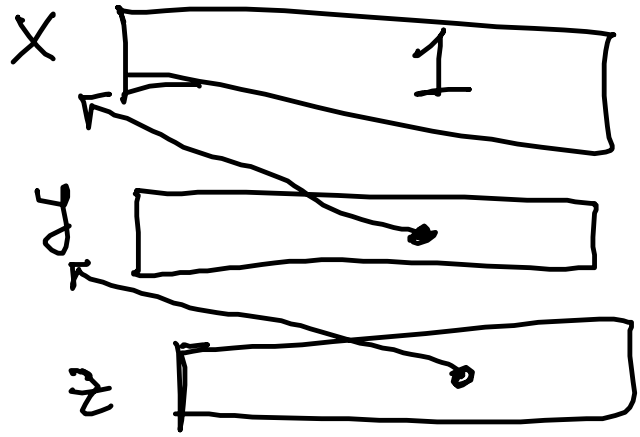
$$X = X$$

? $y = X$, $z = Y$, $X = z$.

x und y sind dasselbe

$X = X$?

true.



X = X

2

z = X , y = 1

x = Y, y = z, z = 1

```
unify(x(N), x(N), Sublist, Sublist) :- !.
unify(x(N), x(M), Sublist, Sublist) :-
    member(x(N)/x(M), Sublist), !;
    member(x(M)/x(N), Sublist), !.
```

?- unify(x(N), x(M), Sublist, [x(N)/x(M) | Sublist]) :- !.

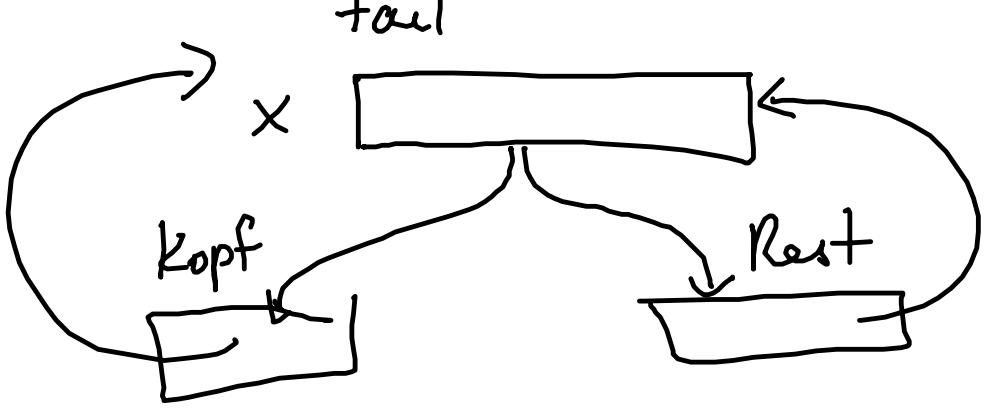
X=1, Y=2 X = Y, ...

```
unify(x(N), Y, Sublist, Sublist) :-
```

```
        member(x(N)/Y, Sublist), !.
unify(x(N), Y, Sublist, _ ):-
        occurs_in(x(N), Y), !, fail.
        :
```

Intermezzo **occur check**

? X = [x|X]. unendliche Strukturen




```
unify( Y, x(N), Sublist, Sublist):-
        (member( x(N)/Y, Sublist);
         member( Y/x(N), Sublist), !.
        :
```

```
unify(X, Y, Sublist, Sublist):-
```

```
        atomic(X), (atomic(Y),!,
                              X==Y; !, fail).

unify (Y, X, Lublist, Rublist) :-
        atomic (X), (atomic(Y),!,
                              X==Y; !, fail).
        ⋮
```

Listen

Strukturen

⋮

{ X(N) schon instanziert