Nichtdeterministische Suche nach Beweisbaum

Ausgangspunkt und Zwischenzustände:

Menge von „offenen“ (zu beweisenden) Teilzielen:
subgoals = { $subgoal_1(...)$,..., $subgoal_m(...)$ }

Die Teilziele $subgoal_1(...)$ haben die Form $funktor(T_1,...,T_n)$.

Dabei bezeichnet funktor ein n-stelliges Prädikat,
dessen Argumente jeweils an Terme T_i gebunden sind.

Terme (Strukturen) sind Variablen, Konstante oder komplexere Strukturen, die wiederum Terme enthalten können.

Nichtdeterministische Suche nach Beweisbaum

Ziel (Ende des Verfahrens):

subgoals ={} , d.h. alle Teilziele sind bewiesen
dabei Antwort „yes“ bzw.
Angabe der Terme,
an die die Variablen der Anfrage gebunden wurden.

oder:
kein weiterer Beweisversuch möglich, dabei Antwort „no“

Nichtdeterministische Suche nach Beweisbaum

Zwischenschritte:

Wähle ein zu beweisendes Teilziel:

funktor(T_1,...,T_n) $\in$ subgoals

Wähle eine passende Klausel der zugehörigen Prozedur:

funktor(X_1,..,X_n) :-funktor^1(X^1_1,..,X^1_{n1}),...,funktor^m(X^m_1,..,X^m_{nm})

Unifikation

des Kopfes funktor(X_1,...,X_n) mit Teilziel funktor(T_1,...,T_n) ergibt eine Variablensubstitution σ

(Ersetzung von Variablen durch Terme)

Neuer Zwischenzustand:

subgoals:= σ((subgoals – { funktor(T_1,...,T_n) })

$\cup$ { funktor^1(X^1_1,..,X^1_{n1}),...,funktor^m(X^m_1,..,X^m_{nm})})

Nichtdeterministische Suche nach Beweisbaum

Die Suche ist erfolgreich, wenn

- in jedem Zwischenschritt eine passende Klausel gewählt wird, bei der die Unifikation gelingt,
- am Ende subgoals={} gilt.

In jedem Schritt i erfolgen Substitutionen σ_i von (allen) Variablen.

Die Substitutionen ergeben in ihrer Gesamtheit die Terme, an die die Variablen X der Anfrage gebunden wurden: „Antwort-Substitution“:

$$\sigma(X) = \sigma_k (\sigma_{k-1} (... \sigma_1 (X) ...))$$

Interpreter für Standard-Prolog

Idee:

Systematisches Probieren von Beweismöglichkeiten
(Reihenfolge für „wähle Teilziel/Klausel“)

- Reihenfolge innerhalb einer Prozedur
  (Alternativen für Beweis)
  oben vor unten

- Reihenfolge innerhalb einer Klausel
  (alle subgoals müssen erfüllt werden)
  links vor rechts

Interpreter für Standard-Prolog

Backtracking:

- Alternativen für den Beweis eines Teilziel werden markiert („Backtrack-Punkte“).

- Beim Fehlschlagen eines Beweisversuchs wird am jüngsten Backtrack-Punkt ein alternativer Beweis gestartet („chronologisches Backtracking“).
  Dabei werden zwischenzeitliche Variablenbindungen zurückgenommen.

- Eingabe von „;“ bei Antworten auf existentielle Anfragen wirkt wie Fehlschlag (löst Backtracking aus) .

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

```
?-grandfather(X,ares).
```

Beweisversuch mit 1. Klausel (ggf. neue Variablennamen!)

```
grandfather(X1,Z):-father(X1,Y),father(Y,Z).
```

Backtrackpunkt für 2. Klausel:

```
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

Substitution σ (X1) = X σ (Z) = ares

```
grandfather(X,ares):-
                father(X,Y),father(Y,ares).
```

zu beweisen:

```
father(X,Y).
```

```
father(Y,ares).
```



```
father(X,Y):-parent(X,Y),male(X).
```

zu beweisen:

Beweisversuch mit Klausel (neue Variablennamen):

(Keine Alternativen – kein Backtrackpunkt)
Substitution σ (X2) = X σ (Y1) = Y

zu beweisen:

```
father(Y,ares).
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

zu beweisen:

```
parent(X,Y).
```

Beweisversuch mit 1. Fakt

```
parent(uranus, cronus).
```

Backtrackpunkt für Alternativen

```
parent(gaea, cronus).
```

Substitution σ (X) = uranus σ (Y) = cronus

```
parent(uranus,cronus).
```

Ist Fakt, d.h. keine neuen Teilziele.

Zu beweisen:

```
male(uranus).
```

```
father(uranus,ares).
```



```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

zu beweisen:

```
male(uranus).
```

Beweis mit Fakt

```
male(uranus).
```

gelingt:

```
male(uranus).
```

Keine neuen Substitutionen.
Keine neuen Teilziele.

Zu beweisen:

```
father(uranus,ares).
```

```
father(X,Y):-parent(X,Y),male(X).
```

zu beweisen:

```
father(uranus,ares).
```

Beweisversuch mit Klausel (neue Variablennamen)

```
father(X3,Y2):-parent(X3,Y2),male(X3).
```

(Keine Alternativen – kein Backtrackpunkt)

Substitution σ (X3) = uranus σ (Y1) = ares

```
father(uranus,ares):-
        parent(uranus,ares),male(uranus).
```

zu beweisen:

```
parent(uranus,ares).
```

```
male(uranus).
```



```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

zu beweisen:

```
parent(uranus,ares).
```

Es gibt keinen solchen Fakt

Beweisversuch fehlgeschlagen.

Rückkehr zum jüngsten Backtrack-Punkt

```
parent(gaea,   cronus).
```

beim Beweis für

```
parent(X,Y).
```

```
male(X).
```

```
father(Y,ares).
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

Weitere Fehlschläge folgen bei Beweisversuchen mit


```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

zu beweisen:

```
parent(X,Y).
```

Beweisversuch mit Fakt

```
parent(cronus, zeus).
```

Backtrackpunkt für Alternativen.

```
parent(rhea, hera).
```

Substitution σ (X) = cronus σ (Y) = zeus

```
parent(cronus,zeus).
```

Ist Fakt, d.h. keine neuen Teilziele.

Zu beweisen:

```
male(cronus).
```

```
father(zeus,ares).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

zu beweisen:

```
male(cronus).
```

Beweis mit Fakt

```
male(cronus).
```

gelingt:

```
male(cronus).
```

Keine neuen Substitutionen.

Keine neuen Teilziele.

Zu beweisen:

```
father(zeus,ares).
```



```
father(X,Y):-parent(X,Y),male(X).
```

zu beweisen:

```
father(zeus,ares).
```

Beweisversuch mit Klausel (neue Variablennamen)

```
father(X3,Y2):-parent(X3,Y2),male(X3).
```

(Keine Alternativen – kein Backtrackpunkt)

Substitution σ (X3) = zeus σ (Y1) = ares

```
father(zeus,ares):-
        parent(zeus,ares),male(zeus).
```

zu beweisen:

```
parent(zeus,ares).
```

```
male(zeus).
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

zu beweisen:

```
parent(zeus,ares).
```

Beweis mit Fakt

```
parent(zeus,ares).
```

gelingt:

```
parent(zeus,ares).
```

Keine neuen Substitutionen.

Keine neuen Teilziele.

Zu beweisen:

```
male(zeus).
```



```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

zu beweisen:

```
male(zeus).
```

Beweis mit Fakt

```
male(zeus).
```

gelingt:

```
male(zeus).
```

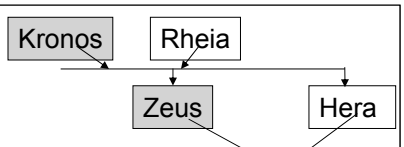
Keine neuen Substitutionen.

Keine neuen Teilziele.

Beweisversuch gelungen mit Substition: σ (X) = cronus

```
?-grandfather(X,ares).
```

Antwort `X = cronus?`

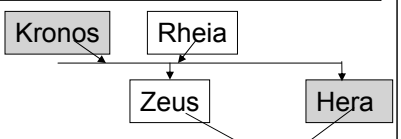


Beweisbaum:

```
grandfather(cronus,ares).
father(cronus,zeus).
father(zeus,ares).
male(cronus).
parent(cronus,zeus).
male(zeus).
parent(zeus,ares).
```



```
?-grandfather(X,ares).
X = cronus?
;
grandfather(X,Z):-father(X,Y),mother(Y,Z).
X = cronus?
```



Zweiter Beweisbaum:

```
grandfather(cronus,ares).
father(cronus,hera).
mother(hera,ares).
male(cronus).
parent(cronus,zeus).
female(hera).
parent(hera,ares).
```

Redundanzen ...

... führen wegen der systematischen Durchmusterung aller Beweisversuche zu Wiederholungen von Resultaten

```
?-grandfather(X,ares).
```

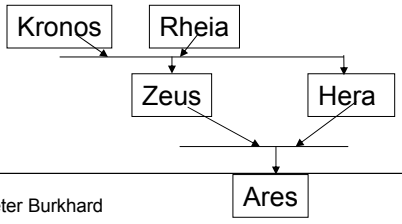
```
grandfather(X,Z):-father(X,Y),father(Y,Z).
```

```
X = cronus?
```

```
;
```

```
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

```
X = cronus?
```



Unifikation (matching, instantiation)

Terme unifizieren:

Durch geeigneten **Unifikator** (Variablen-Substitution σ) als Zeichenkette identisch machen.

```
parent(X,ares).
```

```
parent(zeus,Y).
```

σ (X) = zeus σ (Y) = ares

```
parent(zeus,ares).
```

„instantiation“:

Resultierender Term bei Substitution

Unifikation (matching, instantiation)

in Prolog: Beweisen eines Teilziels erfordert
„Matchen“ von Teilziel und Klauselkopf

```
father(zeus,ares).
father(X3,Y2):-parent(X3,Y2),male(X3).
father(zeus,ares):-
        parent(zeus,ares),male(zeus).
```

- Analogie: „Prozedur-Aufruf“

Unifikation (matching, instantiation)

```
father(zeus,ares).
father(X3,Y2):-parent(X3,Y2),male(X3).
father(zeus,ares):-
        parent(zeus,ares),male(zeus).
```

- Analogie: „Prozedur-Aufruf“
- Parameterübergabe durch Instantiierung:
  Bindung von Variablen für gesamte Klausel
- Variable „gehören“ den Klauseln:
  Lebensdauer bis zum Backtracking
  Sichtbarkeit innerhalb der Klausel

$\sigma(X)=zeus$
$\sigma(Y)=ares$

Kein Überschreiben von Werten

Unifikationsregeln

Terme T_1 und T_2 sind unifizierbar, falls

1. T_1 und T_2 sind identische Konstanten
2. T_1 (bzw. T_2) ist eine Variable:
   T_1 (bzw. T_2) wird an T_2 (bzw. T_1) gebunden.
3. T_1= funktor(T_{11},...,T_{1n}) und T_2 =funktor(T_{21},...,T_{2n})
   sind Strukturen mit identischem funktor (Name, Stelligkeit)
   und die Argumente T_{1i} T_{2i} sind paarweise unifizierbar.

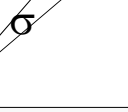
Rekursive Definition,
die Substitution σ ergibt sich schrittweise.

Unifikationsregeln

```
dreieck(P,punkt(X,Y),punkt(1,X)).
dreieck(punkt(2,3),punkt(1,X),punkt(Y,Z)).
```

Variablenseparierung:

```
dreieck(    P,          punkt(X,Y), punkt(1,X)).
dreieck(punkt(2,3),punkt(1,C), punkt(A,B)).
```



σ(P)= punkt(2,3)
σ(X)=1
σ(Y)=C
σ(A) =1
σ(B) =1
σ(C)=C

```
dreieck(punkt(2,3),punkt(1,C), punkt(1,1)).
```

Prolog-Operatoren „=“ und „==“

Das Ziel `term1 = term2` ist erfüllt,
falls `term1` und `term2` unifizierbar sind.

- Variable in `term1` und `term2` werden ggf. instantiiert.

Das Ziel `term1 \= term2` ist erfüllt,
falls `term1` und `term2` nicht unifizierbar sind.

Das Ziel `term1 == term2` ist erfüllt,
falls `term1` und `term2` identisch sind.

- nicht unifizierte Variable sind nicht identisch

Das Ziel `term1 \== term2` ist erfüllt,
falls `term1` und `term2` nicht identisch sind.

Prolog-Operatoren „=“ und „==“

```
?- dreieck(P,punkt(X,Y),punkt(1,X))
   = dreieck(punkt(2,3),punkt(1,X),punkt(Y,Z)).
P = punkt(2, 3)
X = 1
Y = 1
Z = 1

?- dreieck(P,punkt(X,Y),punkt(1,X))
   == dreieck(punkt(2,3),punkt(1,X),punkt(Y,Z)).
No
```

Occur-Check, σ(X) = funktor(..., X, ...)

Beispiel: Programm-Klausel

```
p(X,f (X))
```

Unterschiedliche Reaktionen von Prolog-Systemen auf Anfragen

```
? - p(Y,Y).
```

```
?- p(Y,Y), write(Y).
```

```
?- p(Y,Y), p(Z,Z),Y=Z.
```

- aufwendiger Test.
- Ignorieren?