Darstellung von Funktionen

Eine n-stellige Funktion f(x1,...,xn) ist als
(n+1)-stellige Relation r(x1,...,xn, f(x1,...,xn)) darstellbar

```
addiere(Summand1,Summand2,Summe)
nachfolger(Zahl,Nachfolger)
```

Der Funktionswert muss nicht an der letzten Stelle stehen.

```
multipliziere(Produkt,Faktor1,Faktor2)
```


Relationen und Funktionen

Modellierung von Aufgabenbereichen erfordert
Strukturierung
Beschreibung von Objekten
Beschreibung von Zusammenhängen
Beschreibung von Verfahren

Formale Modelle sind entscheidend für die
- Entwicklung
- Beschreibung
- Bewertung
- Validierung

von Konzepten und Verfahren

Modellierung (z.B. Sprachverarbeitung)

Strukturen z.B.:
- Wörter
- Satzstruktur (Syntax, Grammatik)
- Bedeutungsstruktur (Akteure, Handlung, ...)

Hans kauft das rote Fahrrad von Fritz.
Fritz verkauft sein rotes Fahrrad an Hans.
Fritz schenkt Hans sein rotes Fahrrad.
Hans klaut das rote Fahrrad von Fritz.

Verfahren z.B.
- Wörter erkennen
- Sätze erkennen
- Bedeutung erfassen

Modellierung

Sprache kann beschreiben
- Objekte (z.B. Substantive: Fahrrad)
- Eigenschaften (z.B. Adjektive: rot)
- Beziehungen (z.B. Verben: besitzt)

Hans besitzt ein rotes Fahrrad.

Prädikatenlogisch:
- Prädikate/Relationen beschreiben Eigenschaften und Beziehungen
- Terme beschreiben Objekte

Modellierung: Eigenschaften

Das rote Fahrrad ist drei Jahre alt, leicht reparaturbedürftig und in sehr gutem Zustand. Es kostet 100 € .

Attribut-Werte-Paare:

{ [Farbe,rot], [Alter, 3Jahre], [Zustand, sehr gut], [Funktionsfähigkeit, leicht reparaturbedürftig], [Preis, 100 €] }

Vektor von Attribut-Werten:

[rot, 3, sehr gut, leicht reparaturbedürftig, 100]

$\in W_{Farbe} \times W_{Alter} \times W_{zustand} \times W_{Funktionsfähigkeit} \times W_{Preis}$

Modellierung: Relation (Eigenschaften)

Fahrrad-Angebote

$\subseteq W_{Farbe} \times W_{Alter} \times W_{zustand} \times W_{Funktionsfähigkeit} \times W_{Preis}$

Farbe	Alter	Zustand	Fkts-fähig.	Preis
rot	3	sehr gut	Leicht rep.	100
rot	2	mittel	Ersatzteile	20
gelb	2	gut	ja	100

[blau, 1, sehr gut, sehr gut, 10] $\notin$ Fahrrad-Angebote

Fahrrad-Angebote(blau, 1, sehr gut, sehr gut, 10) = falsch

Modellierung: Relation (Beziehungen)

Hans besitzt das Fahrrad.

besitzen ⊆ Gegenstände x Menschen

Gegenstände	Menschen
Fahrrad	Hans
Kleingeld	Hans
Villa	Fritz

[Villa, Hans] ∉ besitzen

besitzen(Villa, Hans) = falsch

Relation

Eine n-stellige Relationen ist definierbar als
- Teilmenge $R \subseteq W_1 \times ... \times W_n$ bzw.
- Prädikat $W_1 \times ... \times W_n \rightarrow$ {wahr,falsch}

Endliche Relationen können als Tabellen dargestellt werden (Datenbank), z.B. als Faktenmenge in Prolog.

Relationen können z.B.
- Eigenschaften von Objekten (z.B. Datenbank) oder
- Beziehungen zwischen Objekten oder
- Beschränkungen (Constraints)

beschreiben

Modellierung: Relation (Beschränkungen)

Falls das Fahrrad nicht funktionsfähig ist, soll der Zustand höchstens „schlecht“ sein.

Durch Beschränkungen (Constraints)

$C \subseteq W_1 x ... x W_n$

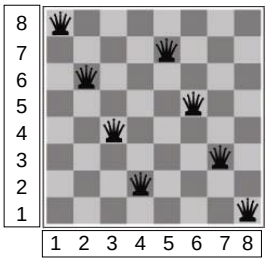
kann verlangt werden, dass nur bestimmte Wertekombinationen zulässig sind.

Beispiele:
- Stundenplanung
- 8-Damenproblem

Constraint-Satisfaction-Problem (CSP).
Bestimme n-Tupel $[w_1, ..., w_n] \in W_1 \times ... \times W_n$,
das gegebenen Beschränkungen $C_1,...,C_k \subseteq W_1 x ... x W_n$
genügt: $[w_1, ..., w_n] \in C_1, ..., [w_1, ..., w_n] \in C_k$

Fallstudie: 8-Damen-Problem

8 Damen auf dem Schachfeld so platzieren,
dass keine eine andere angreifen kann
(im Beispiel nicht erfüllt)



Fallstudie: 8-Damen-Problem

Modellierung:
Position p_i der i-ten Dame (i = 1,...,8) beschrieben durch
$p_i = x_i|y_i$ mit Spalte $x_i \in \{1,...,8\}$, Zeile $y_i \in \{1,...,8\}$

Menge der möglichen Positionen
$P = \{ x|y \mid x \in \{1,...,8\}, y \in \{1,...,8\} \}$

Lösungen haben Form
$l = [x_1|y_1, ..., x_8|y_8] \in P \times P \times P \times P \times P \times P \times P \times P$

Lösungsmenge $L \subseteq P \times P \times P \times P \times P \times P \times P \times P$

Fallstudie: 8-Damen-Problem

Lösungen müssen Constraints erfüllen:
Keine Dame darf eine andere angreifen.

Formulierung:
Dame d_i auf $x_i|y_i$ greift Dame d_k auf $x_k|y_k$ an,
falls $x_i = x_k$ oder $y_i = y_k$ oder $x_i-y_i = x_k-y_k$ oder $x_i+y_i = x_k+y_k$

Constraint C_{ik} bezüglich Dame d_i und d_k (mit $i \neq k$)
(Constraint beschreibt **zulässige** Werte)
$C_{ik} = \{ [x_1|y_1,...,x_8|y_8] \mid x_i \neq x_k \wedge y_i \neq y_k \wedge x_i-y_i \neq x_k-y_k \wedge x_i+y_i \neq x_k+y_k \}$

Lösungen: $L = \cap \{ C_{ik} \mid 1 \leq i < k \leq 8 \}$

Fallstudie: 8-Damen-Problem

Andere Formulierung:
Dame d_i steht in Spalte i, Lösungen haben Form l = [y_1,..., y_8]

Dame d_i auf $i|y_i$ greift Dame d_k auf $k|y_k$ an,
falls $y_i = y_k$ oder $i-y_i = k-y_k$ oder $i+y_i = k+y_k$

Constraint C_{ik} bezüglich Dame d_i und d_k
(Constraint beschreibt zulässige Werte)
$C_{ik} = \{ [y_1,...,y_8] \mid y_i \neq y_k \wedge i-y_i \neq k-y_k \wedge i+y_i \neq k+y_k \}$

Lösungen: $L = \cap \{ C_{ik} \mid 1 \leq i < k \leq 8 \}$

Vorteil:
Suchraum
ist einfacher

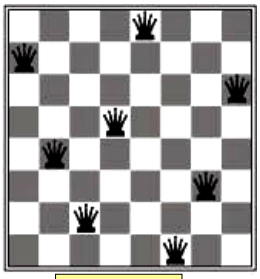
Fallstudie: 8-Damen-Problem

Vergleich der beiden Formulierungen:
Möglichkeiten bei 1. Formulierung:
Dame jeweils auf ein freies Feld stellen

$64 \bullet 63 \bullet 62 \bullet 61 \bullet 60 \bullet 59 \bullet 58 \bullet 57 \approx 3 \bullet 10^{14}$ Möglichkeiten

Möglichkeiten bei 2. Formulierung:
Dame jeweils auf ein Feld in ihrer Spalte stellen
$8 \bullet 8 \bullet 8 \bullet 8 \bullet 8 \bullet 8 \bullet 8 \bullet 8 = 8^8$ Möglichkeiten
Zusätzliche Verbesserung:
Dame jeweils auf bisher nicht angegriffenes Feld stellen
2057 Möglichkeiten

Fallstudie: 8-Damen-Problem



Eine Lösung

Definition von Relationen

Relationen können kombiniert werden z.B.
Mengentheoretisch
(Durchschnitt, Komplement, Projektion...)
Logisch
(Konjunktion,Quantifizierung, ...)

Prolog-Prozeduren definieren neue Relationen (Kopf) aus gegebenen Relationen (Körper). Variable, die im Kopf nicht vorkommen, sind im Körper existentiell quantifiziert.

```
grandfather1(X,Z):-father(X,Y),father(Y,Z).
grandfather1(X,Z):-father(X,Y),mother(Y,Z).
```


Funktionen

Funktionen beschreiben z.B. Abhängigkeiten oder eindeutige Zuordnungen. Im PK1 werden Funktionen syntaktisch durch Terme dargestellt.

Vater(Ares) = Zeus
Kinder(Hera,Zeus) = {Ares, Hephaistos, Hebe}

Vater: Götter → Götter
Kinder: Götter x Götter → $2^{\text{Götter}}$
$2^M = \{ N \mid N \subseteq M \}$ bezeichnet die Potenzmenge

Funktionen können Argumente
von Relationen oder von Funktionen sein.

Vater(Vater(Ares)) = Kronos
Verheiratet(Vater(Ares),Mutter(Ares))

Funktion als Relation

Eine n-stellige Funktion $W_1 \times ... \times W_n \rightarrow W$
kann aufgefasst werden als
n+1 stellige Relation $R \subseteq W_1 \times ... \times W_n \times W$
mit $[w_1, ..., w_n, w] \in R \leftrightarrow f(w_1, ..., w_n) = w$

Hans besitzt das Fahrrad.
besitzen $\subseteq$ Gegenstände x Menschen

Besitzer(Fahrrad) = Hans
Besitzer: Gegenstände → Menschen

Typen, Signaturen

Im Gegensatz zu anderen Programmiersprachen verlangt Prolog i.a. keine Deklaration von Typen für die Argumente (Wertebereiche W_i) von Relationen/Funktionen. Der Typ eines konkreten Arguments ergibt sich aus der Schreibweise.

Allerdings werden bei einigen Prädikaten bestimmte Anforderungen an die Argumente (z.B. Bindung an eine Zahl) verlangt.

Einschränkungen kann es auch geben bzgl.

- Eingabeparametern (müssen instantiiert sein)
- Ausgabeparametern (werden durch Prädikat instantiiert)

Darstellung von Funktionen in Prolog

Ideales Prinzip: Es kann nach Funktionswert und/oder nach Argumentwerten gefragt werden

```
?- addiere(a,b,Summe).
?- addiere(Summand,b,s).
?- addiere(a,Summand,s).
?- addiere(Summand1,Summand2,s).
...
?- addiere(Summand1,Summand2,Summe).
```


Darstellung von Funktionen in Prolog

Umkehrfunktionen unmittelbar definierbar

```
subtrahiere(Minuend,Subtrahend,Differenz)
    :=addiere(Differenz,Subtrahend,Minuend).
```


Primitiv-rekursive Funktionen

```
zahl(o).
zahl(s(X)):-zahl(X).
```

Verwenden o als spezielle Konstante für die kleinste Zahl

```
kleinergleich(o,X):- zahl(X).
kleinergleich(s(X),s(Y)):-kleinergleich(X,Y).
```

Prolog unterscheidet keine Typen.
Damit X explizit auf Zahlen beschränkt ist:
`kleinergleich(o,X):-zahl(X).`

Primitiv-rekursive Funktionen

```
identitaet-i(X1,...Xi,...,Xn,Xi).
constante-c(X1,...,Xn,c).
nachfolger(X,s(X)).
```

```
substitution(X1,...,Xn,F)
   :-f1(X1,...,Xn,F1),..., fm(X1,...,Xn,Fm),
     g(F1,...,Fm,F).
```

```
rekursion(X1,...,Xn,o,F):-g(X1,...,Xn,F).
rekursion(X1,...,Xn,s(X),F)
:-rekursion(X1,...,Xn,X,R),h(X1,...,Xn,X,R,F).
```


Primitiv-rekursive Funktionen

```
add(o,X,X).
add(s(X),Y,s(Z)):- add(X,Y,Z).
```

```
mult(o,X,o).
mult(s(X),Y,Z):- mult(X,Y,W),add(W,Y,Z).
```

```
exp(s(o),o,o).
exp(o,s(X),s(o)).
exp(s(N),X,Y) :- exp(N,X,Z),mult(Z,X,Y).
```

Es kann bei nicht gebundenen Argumenten Probleme geben, deshalb besser expliziter Bezug auf Zahlen:

```
add(o,X,X):- zahl(X).
```

und analog für die anderen Funktionen.

Primitiv-rekursive Funktionen

```
factorial(o,s(o)).
factorial(s(N),F):-
        factorial(N,F1),mult(s(N),F1,F).
```

```
minimum(X,Y,X):-kleinergleich(X,Y).
minimum(X,Y,Y):-kleinergleich(Y,X).
```

```
mod(X,Y,Z):-
 kleiner(Z,Y),mult(Y,Q,W),add(W,Z,X).
```

Alternativ:

```
mod(X,Y,X):-kleiner(X,Y).
mod(X,Y,Z):-add(X1,Y,X),mod(X1,Y,Z).
```


Ackermann-Funktion (Péter-Funktion)

```
ackermann(o,N,s(N)).
ackermann(s(M),o,V):- ackermann(M,s(o),V).
ackermann(s(M),s(N),V):-
     ackermann(s(M),N,V1), ackermann(M,V1,V).
```


Prolog-Arithmetik

```
?- X = 1 + 2 * 3 .
X = 1 + 2 * 3
```

• Behandlung als Term

```
?- X is 1 + 2 * 3 .
X = 7
```

• Behandlung als auszuwertender Arithmetischer Ausdruck

is/2
zweistelliges Prädikat in infix-Schreibweise
Links: ungebundene Variable oder Zahl
Rechts: auswertbarer arithmetischer Ausdruck

Prolog-Arithmetik

`value is expression`

Abarbeitung:
- `expression` wird vom Arithmetik-Evaluierer als arithmetischer Ausdruck ausgewertet
- Resultat wird mit `value` unifiziert

Überschreiben nicht möglich

```
?- X is 1 + 2 * 3 .
X = 7
```

```
?- 7 is 1 + 2 * 3 .
yes
```

```
?- 3 + 4 is 1 + 2 * 3 .
no
```


Prolog-Arithmetik

Verfügbare arithmetische Operationen („reelle“ Zahlen):
+, - , *, /, //, mod und ggf. weitere

Verfügbare arithmetische Vergleichsoperationen:
> , < , >=, =<, =:= , =/= (und ggf. weitere)
Vergleiche für numerisch auswertbare Ausdrücke auf beiden Seiten

Unterschied:
`is` erlaubt Ausdruck nur rechts,
ggf. Unifizierung mit Variabler auf linker Seite
`=:=` überprüft Identität zweier Ausdrücke

Prolog-Arithmetik

Ursprüngliche Definition

```
factorial(o,s(o)).
factorial(s(N),F) :- factorial(N,F1), mult(s(N),F1,F).
```

erlaubt Anfrage

```
?- factorial(X,Y).
```

Bei Prolog-Arithmetik ist diese Anfrage nicht möglich

```
factorial(0,1).
factorial(N,F) :- N> 0, N1 is N-1, factorial(N1,F1), F is N * F1.
```

• Laufzeitfehler, wenn Ausdrücke nicht auswertbar sind

Speziell bei ungebundenen Variablen:
Zwang zu „prozeduraler“ Reihenfolge der subgoals

Prolog-Arithmetik

Ursprüngliche Definition

```
add(Summand1,Summand2,Summe)
```

erlaubt Definition

```
minus(Minuend,Subtrahend,Differenz) :- add(Subtrahend,Differenz,Minuend)
```

Bei Verwendung der Prolog-Arithmetik ist das nicht möglich.

```
addiere(X,Y,S) :- S is X + Y
```
entspricht nicht früherem `add(X,Y,S)`

Operatoren

Standard-Schreibweise in Prolog:
Struktur/Term: funktor(Argumente)
Alternativ: Operator-Schreibweise für bessere Lesbarkeit

		Funktor
Infix-Operator	7 + 9 für Struktur + (7,9)	+/2
Prefix-Operator	- 9 für Struktur - (9)	-/1
Postfix-Operator	9! (Fakultät)	!/1

Zu klären:
- Priorität: $7 + 9 * 2 = 7 + (9 * 2)$
- Assoziativität: $7 - 9 - 2 = (7 - 9) - 2$

Operatoren

Operatoren deklarieren mittels
`op(Priorität, Typ, Name)`

```
op( 500, yfx, '-' ).
op( 500, yfx, '+' ).
op( 400, yfx, '*' ).
```

Vergabe von Prioritäten: 0,...,1200 (Maximum)

Priorität eines Terms: Priorität des Hauptfunktors
Priorität 0 haben:
Atome (außer Operatoren), Zahlen, Variable, Zeichenketten,
in Klammern eingeschlossene Terme

Operatoren

Festlegung der Assoziativität durch Typen:
- Infixoperatoren `xfx, xfy, yfx`,
- Präfixoperatoren `fx, fy`,
- Postfixoperatoren `xf, yf`.

```
op( 500, yfx, '-' ).
op( 500, yfx, '+' ).
op( 400, yfx, '*' ).
```

`f`: Funktor
x: Term mit geringerer Priorität als op
y: Term mit maximal gleicher Priorität wie op
(andernfalls Klammern notwendig)

Beispiele für Operatoren

Standardmäßig im Prolog-Interpreter (built-in-Operatoren)

1200 xfx	:-	
1200 fx	?-	
1100 xfy	;	
1000 xfy	,	
900 fy	not	
700 xfx	=, \=	(Unifikation)
	==, \==	(Identität für Terme)
	<, =:=, >, =<, >=, =\=, is	(Arithmetik)
500 yfx	+, -	
500 fx	+, -,	
400 yfx	*, /, //,mod	

Rekursive Definitionen

```
zahl(0).
zahl(s(X)):-zahl(X).
```

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

```
erreichbar(X,X).
erreichbar(X,Y)
        :- benachbart(X,Z),erreichbar(Z,Y).
```

Transitiver Abschluss von Relationen

Logisch äquivalente rekursive Definitionen:

```
erreichbar(X,Y):- benachbart(X,Z), erreichbar(Z,Y).
erreichbar(X,X).
```

```
erreichbar(X,X).
erreichbar(X,Y):- benachbart(X,Z), erreichbar(Z,Y).
```

```
erreichbar(X,X).
erreichbar(X,Y):- erreichbar(Z,Y), benachbart(X,Z).
```

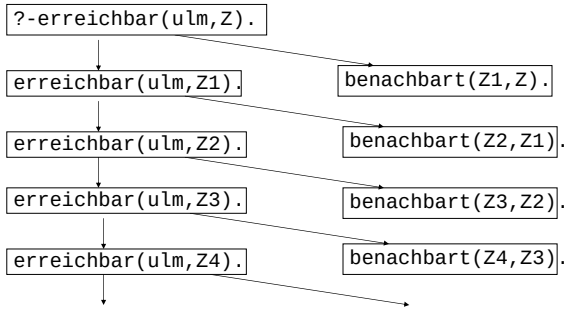
```
erreichbar(X,Y):- erreichbar(Z,Y), benachbart(X,Z).
erreichbar(X,X).
```

Inhaltlich äquivalent z.B. auch:

```
erreichbar(X,Y) :- erreichbar(X,Z), benachbart(Z,Y).
erreichbar(X,X).
```

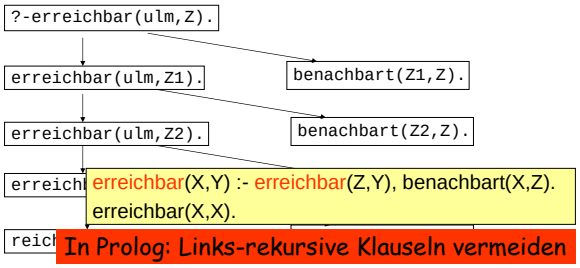

Unendliche Beweisversuche

```
erreichbar(X,Y):-erreichbar(X,Z),benachbart(Z,Y).
```



Deklarative vs.prozedurale Semantik

Unterschiedliche Resultate bei
deklarativer und prozeduraler Semantik



Deklarative vs.prozedurale Semantik

Januskopf von Prolog:
Unterschiedliche Resultate
bei deklarativer und prozeduraler Semantik

Reihenfolge der Beweisversuche
(Auswirkungen z.B. auf rekursive Klauseln, Negation)
(zusätzliche) Arithmetik
Abhängigkeit von „Seiteneffekten“,
z.B. Eingabe-/Ausgabe-Abhängigkeit
Eingriffe in Beweisversuche (cut)
Meta-logische Prädikate
Programm-Modifikation

Eingriff in die Abarbeitung: Cut

Verändern der Suchstrategie: Prädikat **`!/0`** (*Cut*)

`!/0` gelingt stets und löscht Choice-Points für
- aktuelle Klausel
- subgoals im Klauselkörper, die vor dem Cut stehen
- subgoals dieser subgoals usw.

Folge:
Gefundene Lösung wird ,,eingefroren“
Alternativen für Backtracking entfallen

Eingriff in die Abarbeitung: Cut

```
prinz(X):-grandchild(X,cronus),male(X).
```

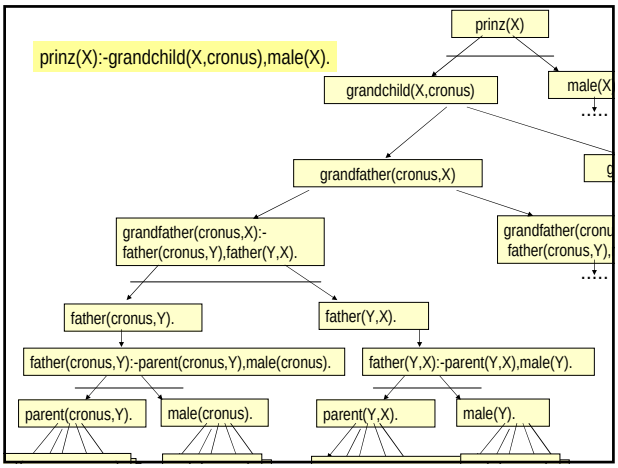
```
?- prinz(X).
X = hermes ;
X = hephaestus ;
X = apollo ;
X = hephaestus ;
no
```

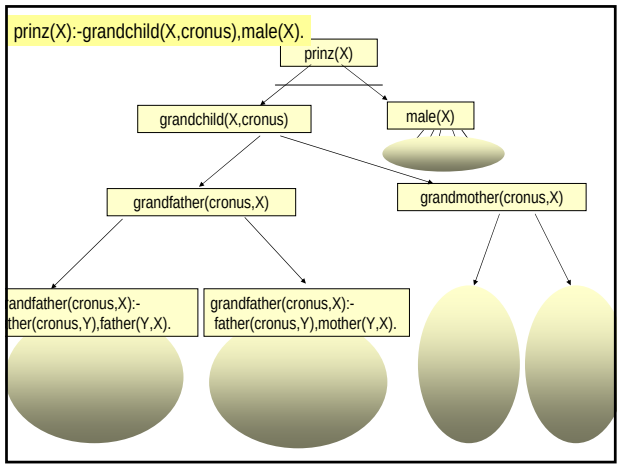
```
kronprinz(X):-grandchild(X,cronus),male(X),!.
```

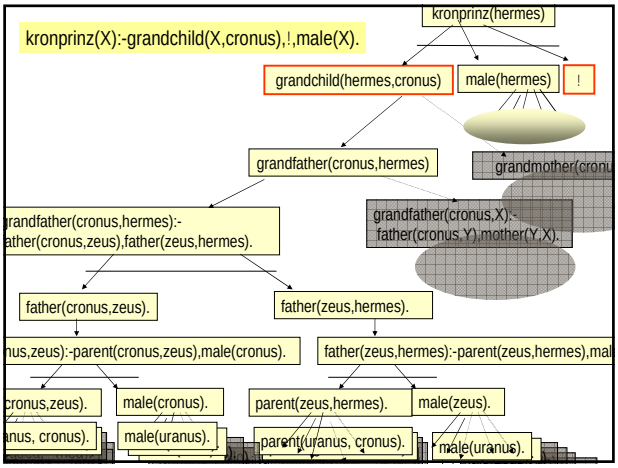
```
?- kronprinz(X).
X = hermes ;
no
```

Worin besteht der inhaltliche Unterschied:

```
kronprinz(X):-grandchild(X,cronus),!,male(X) .
```







Cut zur Beschleunigung

(unnötige) Alternativen vermeiden

```
maximum1(X,Y,X) :-  X >= Y, !.
maximum1(X,Y,Y) :-  X <  Y .
```

Grüner cut: gleiches Resultat ohne Cut

```
maximum2(X,Y,X) :- X >= Y, !.
maximum2(X,Y,Y).
```

Roter cut: anderes Resultat ohne Cut