Modellierung: Relation (Beziehungen)

Hans besitzt das Fahrrad.

besitzen $\subseteq$ Gegenstände x Menschen

Gegenstände	Menschen
Fahrrad	Hans
Kleingeld	Hans
Villa	Fritz

[Villa, Hans] $\notin$ besitzen

besitzen(Villa, Hans) = falsch

Relation

- Eine n-stellige Relationen ist definierbar als
  - Teilmenge $R \subseteq W_1 \text{ x } ... \text{x } W_n$ bzw.
  - Prädikat $W_1 \text{ x } ... \text{x } W_n \rightarrow$ {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.

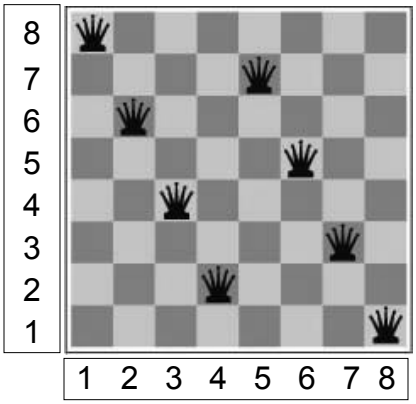
Beispiele:

- Stundenplanung
- 8-Damenproblem

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

$P = \{ \; x/y \;\; | \;\; x \in \{1,...,8\} , y \in \{1,...,8\} \; \}$

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

(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: $L = \bigcap \{ \; C_{ik} \;\; | \;\; 1 \leq i < k \leq 8 \}$

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)

$C_{ik} = \{ [y_1,...,y_8] \mid y_i \neq y_k \wedge i-y_i \neq k-y_k \wedge i_i+y_i \neq k+y_k \}$

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

Vorteil:
Suchraum
ist einfacher

Fallstudie: 8-Damen-Problem

Vergleich der beiden Formulierungen:

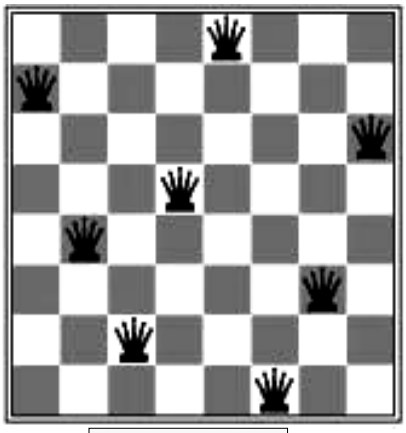
Möglichkeiten bei 1. Formulierung:
Dame jeweils auf ein freies Feld stellen

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

Möglichkeiten bei 2. Formulierung:
Dame jeweils auf ein Feld in ihrer Spalte stellen
$8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 = 8^8$ 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 $\rightarrow$ Götter

Kinder: Götter x Götter $\rightarrow 2^{\text{Götter}}$

$2^M = \{ N \mid N \subseteq M \}$ 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 $\rightarrow$ 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).
```

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
```