Prädikatenlogik

V: Menge Variablen ($x \in V$) } Elem. sind Terme
K: Menge Konstanten
F: Menge Funktionssymbole

$f(t_1,\dots,t_n)$ = Term

F, G: Formeln:
$F \wedge G$, $F \vee G$, $\neg F$, (F) auch Formeln.
$\forall x F$, $\exists x F$ auch Formeln.

Priorität Operatoren: $x \to y = \neg x \vee y$
$\neg$; $\forall, \exists$; $\wedge, \vee$; $\to, \leftrightarrow$

Formel $P(t_1,..,t_n)$ wahr, wenn $(t_1,\dots,t_n) \in P$.
$\exists x F$: es gibt wahres Elem. im Universum
$\forall x F$: F ist für alle x wahr

Rechenregeln: (Q: Quantor $\in \{\forall, \exists\}$) * E_5 E_6
- $\neg \forall x F = \exists x \neg F$ (o: Operator $\in \{\wedge, \vee\}$)
- $\neg \exists x F = \forall x \neg F$
- $\exists x F \vee \exists x G = \exists (F \vee G)$
- $\forall x F \wedge \forall x G = \forall x (F \wedge G)$
- $\forall x \forall y F = \forall y \forall x F$
- $\exists x \exists y F = \exists y \exists x F$
- $(Qx F) \circ G = Qx (F \circ G)$
- $Qx_1 (F(x_2) \circ G(x_1,..)) = F(x_2) \circ Qx_1 G(x_1,..)$

Pränexform: (ist für alle Aussagen vorhanden)
$F = Q_1 y_1 \dots Q_n y_n G$
($Q_1 \dots Q_n$: paarweise Verschieden; G: enthält keine Quantoren)

Hornformel: $\hat{=}$ Wissensbasis (Verknüpfung von Fakten)
- Literal: (negierte) Atomformeal bspw. $P(x,y)$, $\neg Q(x,y,z)$
- Hornklausel: v-Verknüpfte Literale bspw.: $P(x,y) \vee Q(x)$
- Hornformel: $\wedge$-Verknüpfte Hornklauseln

Spezialfall: mind. 2 Literale & genau 1 positives:
$\neg A_1 \vee \dots \vee \neg A_{n-1} \vee A_n = \neg(A_1 \wedge \dots \wedge A_{n-1}) \vee A_n = A_1 \wedge \dots \wedge A_{n-1} \to A_n$

⇒ Expertensystem:
Wissensbasis ↔ Inferenz-System ← Anfrage → Antwort

Prolog: Hornformeln mit allen Variablen allquantifiziert. Klausel = Zeile * E_3

- Prädikat/Konstante: klein
- Variable: groß

← = :-
∨ = ;
∧ = ,

_ : anonyme Variable
not(X=Y): Ungleich

is ...: Auswertung
== identisch
\== nicht id.
= unifizierbar
\= nicht uni.

X is 3+4 ∨(-1) * E_2
10 is 3+4 ∨(false)
p(X) == p(X)
p(X) == p(Y)
p(X) = p(Y)
p(X) \= q(X)

=:= arithm. gleich 2=:=1+1
=\= arithm. ungl. 3=\=1+1

[] : leere Liste
[H|T]: H: Kopf, T: Rest der Liste

Existenzquantor durch Skolemisierung: $\exists x P(x) = P(a)$

Und-Oder-Baum (Suchbaum):
a(X) :- b(X), c(X), d(X) ⇒
a(X) :- b(X); c(X); d(X) ⇒

⇒ Darstellung Unifizierung:
a(X) :- b(X); c(X).
a(X) wahr
X/t, X/t, X/t
b(t) falsch, c(t) wahr, d: backtracking

a(X,Y) :- b(X,Y).
a(p, Y)
X/p, Y/q
b(p,q)
⇒ Ersetzung angewandt von Ursprungsformel

Unifizierbarkeit:
P(X,a) = P(a,a) mit x/a ✓
Q(a,x,y) = Q(z,b,c) mit z/a, x/b, y/c ✓
P(x,x) ≠ P(a,b) ↯ x nicht gleichzeitig a und b

if-else in Prolog:
"wenn regen, dann nass, sonst trocken"
a(regen, nass). a(X, trocken) :- X \== regen.

Relationen:
symmetr. Hülle: q(X,Y) :- p(X,Y); p(Y,X).
reflex. Hülle: p(X,X).
transitive Hülle: (NICHT: q(X,Z) = p(X,Z); p(X,Y), p(Y,Z).)
Berechnung: $\forall x\, p(X,Y) \wedge \forall X \forall Y \forall Z (\exists Z (p(X,Z) \wedge p(Z,Y)) \to p(X,Y))$ (Z: max. Länge 2)
$\forall X \forall Y \forall Z (p(X,Z) \wedge p(Z,Y) \to p(X,Y))$
⇒ p(X,X). (p(X,Y) :- p(X,Z), p(Z,Y).) führt zu Rekursion
tp(X,Y) :- p(X,Y); p(X,Z), tp(Z,Y).
besser:

Suche

uninformiert:
- Tiefen-/Breitensuche
- Exponentieller Laufzeit/Speicherbedarf (wenn besuchte Knoten gespeichert werden)

- Breitensuche:
  + liefert immer kürzesten Pfad
  + auch für ∞ Graphen
  - alle Knoten gespeichert
- Tiefensuche
  + Nur Knoten auf akt. Pfad gespeichert
  - nicht immer kürzester Pfad
  - nicht für ∞ Graphen

Iterative Tiefensuche
- Tiefensuche mit stets erhöhender Tiefenschranke
+ Vorteile aus Tiefen- & Breitensuche
- Rechenzeit länger als Breitensuche (alle Kanten aus vorheriges Tiefe werden erneut besucht), nur wenig höher als Tiefensuche

informiert: (mit Heuristik $\hat{=}$ Bewertungsfkt. für Knoten)
- gierige Suche:
  Verwende Pfad, auf dem sich kürzester Teilpfad befindet (Teilpfad mit kleinster Heuristik). Beachtet nicht bereits entstandene Kosten.
- A*-Suche:
  $f(v) = g(v) + h(v)$
  bereits verbrauchte Kosten bis v → (geschätzte) Kosten bis Ziel
  + optimal: findet kürzesten Weg
  zulässige Heuristik h: nie überschätzen!
  ↳ gute Schätzung: Luftlinie
  ↳ möglich: 0 (nutzlos ⇒ gierige Suche)
  + schneller bei gutes Heuristik
  - hoher Speicherbedarf (worst-case: alle Knoten gespeichert)
  - keine (einfache) Rückgabe des Pfades (vgl. Breitensuche)
- IDA*: A* + iterative Tiefensuche (anstatt Tiefenschranke: Schranke für Bewertungsfunktion)

Heuristiken Zahlenpuzzle: * E_1
- Hamming-Distanz: wie viele Stellen sind nicht korrekt!
- Manhatten/City-Block: wie viele Bewegungen ist jedes Elem. vom Ziel entfernt (Summe)

Allg.:
Knoten [und deren Heuristik] sind Plättchendarstellungen, nicht einzelne Plättchen.

Implementation Heuristik-Suche
- Liste: schlecht (sortieren braucht jedes Mal O(log n), ggf. nochmal O(log n).
- Min-Heap: Operationen nur in O(log n)

⇒ jeder Knoten kleiner als Nachfolgeknoten:

⇒ Suche mit Min-Heap:
- Starthknoten notieren: (A, f=g+h)
- G* Wurzelknoten entfernen und f für Nachbarknoten berechnen und entsprechend ober-/unterhalb einfügen
- Suche beendet, wenn Wurzelknoten zum Ziel führt.

Bsp.: (A,6) ⇒ (A-C,4), (A-B,7)

Min-Max / α-β:

1: Max gewinnt
0: unentschieden
-1: Min gewinnt
⇒ Min sucht immer kleinsten Nachfolgepfad, Max sucht größten

α-β:
- $\Delta[\alpha, \_]$
- $\nabla[\_, \beta]$
- $\nabla[\_, \beta \leq \alpha]$ (* → bleibt dann -∞ bzw. ∞)
- $\Delta[\alpha \geq \beta, \_]$

⇒ Nur weitersuchen, wenn Min-/Max-Wert des Vaterknotens potentiell verändert werden kann.

Bsp.: $\Delta[0,0]$, $\nabla[0,0]$, $\nabla[-\infty,-1]$, ...

Prolog-Code: (E_1)

Hamming:
- h0(S,S,0). h0(-,S,0). (leerstellen egal)
- h0(A,A,0). (gleiche Zielpos.)
  h0(_,_,1). (sonst)

h(Board, H0) :- flatten(Board, B), goal(goal(G)), flatten(Goal(B,G)), maplist(h0, B, G, Diffs), sumlist(Diffs, H0), !.

Manhatten:
: ⇒ wie Hamming
h0(_,_,X) :-
X is abs(X1-X2)
+ abs(Y1-Y2)

Prolog-Code: (E_2)

Bsp.: x+3*y+x*2 = A+B+C
⇒ A = x, B = 3*y, C = x*2

Vereinfachung/Ersetzen:
Einfach: s0(1*X, X).
s0(0+X, X).
⋮
s0(X, X). zum Abbruch * E_4

Erweitert:
s0(A*X+B*X, C*X) :- s0(A+B,C).

Berechnung:
S(A*B, C) :- s(A,SA), s(B,SB), s0(SA*SB, C).
↳ gleiches für +
↳ am Schluss wieder s(A,A).

Prolog (E_3) vs. Imperative Programmiersprache:

Abfragen wie ties("uqmo") auch dort einfach, jedoch nicht ties(X) ⇒ Rückgabemenge

Hinweis, vgl. Abbruch: (E_4)

Abbruchbedingung immer zuerst:
zug(X,Y) :- zug(Y,X). ↯
zug(1,2)