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: $\neg; \forall, \exists; \wedge, \vee; \rightarrow, \leftrightarrow$ ($\rightarrow$ = $\neg x \vee y$)

Formel $P(t_1,\dots,t_n)$ 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 Q E_6 Q
(o: Operator $\in \{\wedge, \vee\}$)
- $\neg \forall x F = \exists x \neg F$
- $\neg \exists x F = \forall x \neg F$
- $\exists x F \vee \exists x G = \exists (F \vee G)$
- $\forall x F \wedge \forall x G = \forall x (F \wedge G)$
- $\forall x \forall y F = \forall y \forall x F$
- $\exists x \exists y F = \exists y \exists x F$
- $(Q x F) \circ G = Q x (F \circ G)$
- $Q x_1 (F(x_2) \circ G(x_1, \dots)) = F(x_2) \circ Q x_1 G(x_1, \dots)$

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

Horn formel: ≙ Wissensbasis (Verknüpfung von Fakten)
- literal: (negierte) Atomformel bspw. $P(x,y)$, $\neg Q(x,y,z)$
- Hornklausel: v-Verknüpfte Literale bspw.: $P(x,y) \vee Q(x)$
- Hornformel: ∧-Verknüpfte Hornklauseln

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

⇒ 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 — X is 3+4 ✓(7) * E_2; 10 is 3+4 ✗(false)
== identisch — p(X)==p(X)
\== nicht id. — p(X)==p(Y)
= unifizierbar — p(X)=p(Y)
\= nicht uni. — 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, b(t) falsch — X/t, c(t) wahr — ?: backtracking

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

Unifizierbarkeit:
$P(x,a) = P(a,a)$ mit x/a ✓
$Q(a,x,y) = Q(z,b,c)$ mit $z/a, x/b, y/c$ ✓
$P(x,x) \neq P(a,b)$ ↯ 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:
$\forall x\, p(x,x) \wedge \underbrace{\forall X \forall Y (\exists Z (p(X,Z) \wedge p(Z,Y)) \rightarrow p(X,Y))}_{\text{nur Länge 2 } \Rightarrow \text{ Lösung}}$
$\forall X \forall Y \forall Z (p(X,Z) \wedge p(Z,Y) \rightarrow p(X,Y))$
⇒ p(X,X). (p(X,Y):- p(X,Z), p(Z,Y).) führt zu Rekursion
besser:
tp(X,Y):- p(X,Y); p(X,Z), tp(Z,Y).

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 vorherigen Tiefe werden erneut besucht), nur wenig höher als Tiefensuche

informiert: (mit Heuristik ≙ Bewertungsfkt. für Knoten)
- gierige Suche:
  Verwende Pfad, auf dem sich kürzester Teilpfad befindet (Teilpfad mit kleinster Heuristik). Beachtet nicht bereits entstandene Kosten.
- A*-Suche:
  $f(v) = g(v) + h(v)$
  g(v): bereits verbrauchte Kosten bis v; h(v): (geschätzte) Kosten bis Ziel
  + optimal: findet kürzesten Weg !

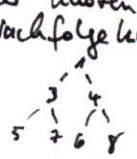
zulässige Heuristik h: nie überschätzen!
↳ gute Schätzung: Luftlinie
↳ möglich: 0 (nutzlos ⇒ gierige Suche)
+ schneller bei guter Heuristik
- hoher Speicherbedarf (worst-case: alle Knoten gespeichert)
- keine (einfache) Rückgabe des Pfades (vgl. Breitensuche)
- IDA*: A* + iterative Tiefensuche (anstatt 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))
- Min-Heap: Operationen nur in O(log n), ggf. nochmal O(log n).

⇒ jeder Knoten kleiner als Nachfolgeknoten:



⇒ Suche mit Min-Heap:
- Starthnoten 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:
- Δ[α, _]
  - ∇[*, β≤α]
    - Δ
    - ✗
- ∇[_, β]
  - Δ[α≥β, *]
    - ∇
    - ✗

* → bleibt dann −∞ bzw. ∞

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

Bsp.:
Δ[0,0]
- ∇[0,0]
  - Δ0
- ∇[−∞,−1]
  - x−1
  - ✗

Prolog-Code: (E_1)

- Hamming:
  - h0(9,_,0). h0(_,9,0). (leerstellen egal)
  - h0(A,A,0). (gleiche Zielpos.)
    h0(_,_,1). (sonst)
  - h(Board, H0):- flatten(Board, B), goal(goalB), flatten(GoalB,G), maplist(h0, B, G, Diffs), sumlist(Diffs, H0), !.
- Manhatten:
  : ⇒ wie Hamming
  h0(_,_,X):-
  X is abs($X_1 - X_2$) + abs($Y_1 - Y_2$)

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 (E3) vs. Imperative Programmiersprache:

Abfragen wie tier("uomo") auch dort einfach, jedoch nicht tier(X) ⇒ Rückgabemenge

Hinweis, vgl. Abbruch: (E_4)
Abbruchbedingung immer zuerst:
✓ zug(X,Y):- zug(Y,X) ↯
zug(1,2)

Bayes'sche Netze

Grundlagen:

- $P(A,B) = P(A \cap B)$
- $P(B|A) = \frac{P(A,B)}{P(A)}$

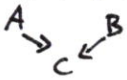
$\Leftrightarrow P(A,B) = P(B|A) \cdot P(A)$

$\Leftrightarrow P(A) = \frac{P(A,B)}{P(B|A)}$

⇒ Marginalisierung / disjunkte Zerlegung:

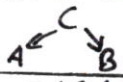
- $P(A) = P(A,B) + P(A,\bar{B}) = P(A|B) \cdot P(B) + P(A|\bar{B}) \cdot P(\bar{B})$
- unabhängige A, B:
  - $P(A,B) = P(A) \cdot P(B)$ · $P(A|B) = P(A)$
  - $P(A,B|X) = P(A|X) \cdot P(B|X)$
- $P(A|B) = 1 - P(\bar{A}|B)$
- $P(\sum A_k) = \sum P(A_k)$
- $P(\bigcup A_k) \leq \sum P(A_k)$

- bedingt unabh., wenn
  $P(A,B|C) = P(A|C) \cdot P(B|C)$



(A und B sind unabh.)

- A und B unabhängig gegeben C:



- wenn DAG (directed asymmetric graph) (⇔ keine Loops), dann topologische Sortierung möglich:

$P(A_n | A_1, \dots, A_{n-1}) = P(A_n | \text{Eltern}(A_n))$

$\Rightarrow P(A_1, \dots, A_n) = \prod_{k=1}^{n} P(A_k | \text{Eltern}(A_k))$

Inferenz: ist NP-vollständig!

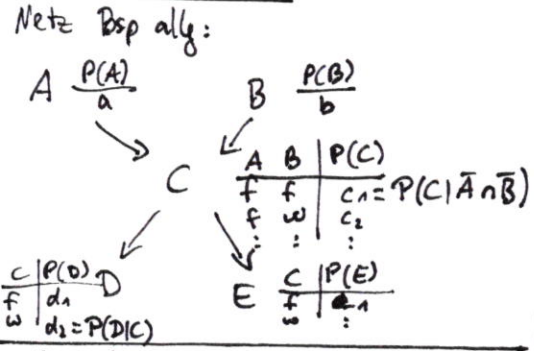
↳ Inferenz-Approximation durch Sampling:

- Stichprobe von Belegung ziehen (wahr/falsch)
  Bsp.: $\underbrace{A=w,\ B=w}_{\text{zufällig wählen}},\ C=w$ ⇒ Ergebnis „ziehen"
- Gezogene Werte für Schätzer P(C) nutzen

\+ Einfach zu implementieren

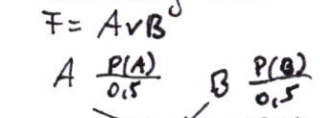
− schlecht bei Knoten mit kleinen Wk.

Netz Bsp aly:



Berechnung einer Formel

$F = A \vee B$



Rechnung:

$P(F) = P(F,A,B) + \dots + P(F,\bar{A},\bar{B})$

$= P(F|A,B) \cdot \underbrace{P(A) \cdot P(B)}_{\text{stets gleich, da } 0{,}5} + \dots + P(F|\bar{A},\bar{B}) \cdot \underbrace{P(\bar{A}) \cdot P(\bar{B})}_{\text{stets gleich, da } 0{,}5}$

Anz. von A,B,... $= (\frac{1}{2})^2 \cdot (P(F|A,B) + \dots + P(F|\bar{A},\bar{B}))$

$= (\frac{1}{2})^2 \cdot 3$ ← Anzahl von 1en in P(F)

Prolog-Funktionen:

- atomic(X) ... atomare Formel
- number(X) ... Zahl
- append(L1, L2, L) ... L ist konkatenierte Liste L1+L2
  ↳ auch L_1/L_2 aus L errechenbar
  ⇒ append(X, L2, L) → X = ...
  oder: alle möglichen Kombinationen:
  append(L1, L2, [a,b]) → L1 = ... L2 = ...
- member(X, [X|_]). ∀X ist an Kopf ≙ match(..)

member(X, [_|T]) :- member(X|T).

⇒ member(Needle | Haystack).

- match(Neede, [Needle | Rest], Rest, Needle).
  Haystack; optional

Suchen:

- findall(X, Bedingung, Liste)
  ↑ alle aus X mit Bedingung erfüllt

- :- [idDfs]

idDfs(Start, Goal, RetPath).

↳ definiere Adjazenzen!:

adj(X, Y) :- benachbarte Knoten

↳ bspw.: (adj0(X,Y); adj0(Y,X)), bed(X), bed(Y).

ggf.: definiere adj0(X, Y). (gültige Übergänge)

- :- [idas]

↳ definiere Heuristik

h(Head, H)

akt. Knoten · Heuristik

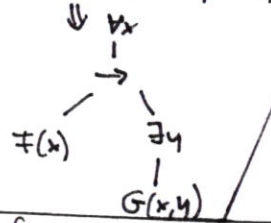
Grammatik: ggf. mit genus (m/w)

Satz = Nominalphr. + Verbalphr. ← ggf. mit (in)transitiven: „jagen" vs. „jagt etwas"

Artikel + Nomen · Verb + Nominalphr.

Formel → Baum: (E_5)

$\forall x (F(x) \rightarrow \exists y\, G(x,y))$



Aussagenumformung: (E_6)

- Jeder Mensch ist M oder F:
  $\forall x (Men(x) \rightarrow M(x) \vee F(x))$
  → gilt auch für „Stuhl"
  ⇒ $Men(x) \wedge [M(x) \vee F(x)]$ ist falsch!
- Jeder M und F ist Men.:
  $\forall x (M(x) \vee F(x) \rightarrow Men(x))$
  → trotz „und" ein „oder", das meint: $\forall x (M(x) \rightarrow Men(x)) \wedge \forall x (F(x) \rightarrow Men(x))$
- kein Men ist gleichzeitig M und F:
  $\forall x (Men(x) \rightarrow \neg (M(x) \wedge F(x)))$