Inference ve výrokové a predikátové logice

Aleš Horák

E-mail: hales@fi.muni.cz
http://nlp.fi.muni.cz/uui/

Obsah:

- Inference ve výrokové logice
- Inference v predikátové logice
- Shrnutí

Logický agent – inference

připomínka – komponenty logického agenta:

inferenční stroj (inference engine)
báze znalostí (knowledge base)

algoritmy nezávislé na doméně

znalosti o doméně

inference – vyvozování důsledků z báze znalostí

$KB \vdash_i \alpha$ . . . věta α může být vyvozena z KB pomocí (procedury) i (i odvodí α z KB)

Bezespornost: i je bezesporná $\Leftrightarrow$ $\forall KB \vdash_i \alpha \Rightarrow KB \models \alpha$

Úplnost: i je úplná $\Leftrightarrow$ $\forall KB \models \alpha \Rightarrow KB \vdash_i \alpha$

Pokud je KB pravdivá v reálném světě $\Rightarrow \forall$ *věta* α *vyvozená z KB pomocí bezesporné inference je také pravdivá ve skutečném světě*

Historie logického vyvozování

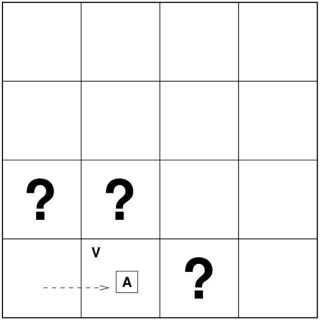
450 př.n.l.	stoikové	výroková logika, inference (pravděpodobně)
322 př.n.l.	Aristoteles	inferenční pravidla, kvantifikátory
1565	Cardano	teorie pravděpodobnosti (výroková logika + nejistota)
1847	Boole	výroková logika (znovu)
1879	Frege	predikátová logika 1. řádu
1922	Wittgenstein	důkaz pomocí pravdivostních tabulek
1930	Gödel	$\exists$ úplný algoritmus pro PL1
1930	Herbrand	úplný algoritmus pro PL1 (redukce na výroky)
1931	Gödel	$\neg\exists$ úplný algoritmus pro aritmetiku
1960	Davis/Putnam	"praktický" algoritmus pro výrokovou logiku
1965	Robinson	"praktický" algoritmus pro PL1 – rezoluce

Inference ve Wumpusově jeskyni

situace:

– v $[1, 1]$ nedetekováno nic
– krok doprava, v $[2, 1]$ Vánek

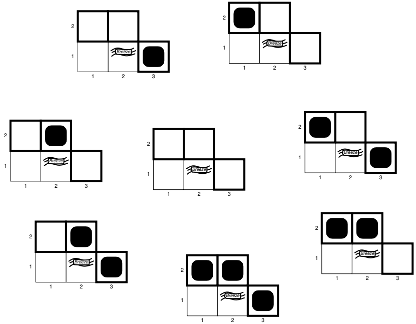
uvažujeme možné modely pro '**?**'
(budou nás zajímat jen Jámy)



3 pole s Booleovskými možnostmi $\{T, F\}$ $\Rightarrow$ $2^3 = $ **8 možných modelů**

Modely ve Wumpusově jeskyni

uvažujeme všech 8 možných modelů:



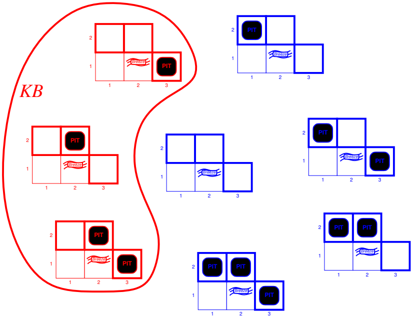
KB = pravidla Wumpusovy jeskyně + pozorování

α_1 = "[1, 2] je bezpečné pole"

α_2 = "[2, 2] je bezpečné pole"

Modely ve Wumpusově jeskyni

uvažujeme všech 8 možných modelů:



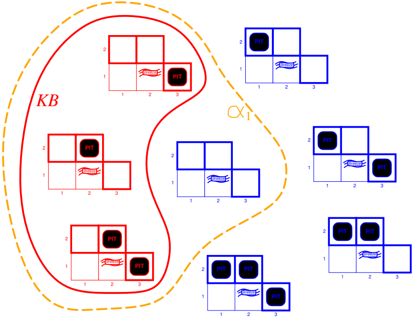
KB = pravidla Wumpusovy jeskyně + pozorování

α_1 = "[1, 2] je bezpečné pole"

α_2 = "[2, 2] je bezpečné pole"

Modely ve Wumpusově jeskyni

uvažujeme všech 8 možných modelů:



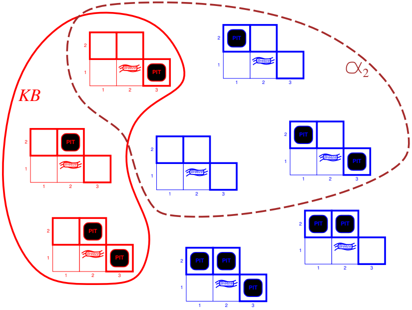
KB = pravidla Wumpusovy jeskyně + pozorování

α_1 = "[1, 2] je bezpečné pole" $KB \models \alpha_1$, pomocí kontroly modelů

α_2 = "[2, 2] je bezpečné pole"

Modely ve Wumpusově jeskyni

uvažujeme všech 8 možných modelů:



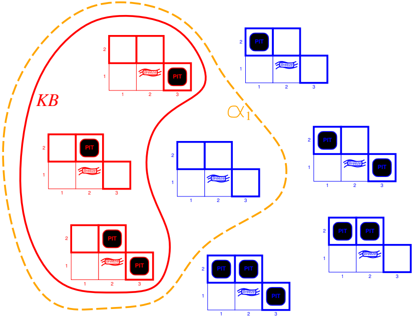
KB = pravidla Wumpusovy jeskyně + pozorování

α_1 = "[1, 2] je bezpečné pole"

α_2 = "[2, 2] je bezpečné pole" $\quad KB \not\models \alpha_2 \quad \Leftarrow \quad \exists$ modely: *KB* je pravdivá $\wedge\ \alpha_2$ je nepravdivá

Modely ve Wumpusově jeskyni

uvažujeme všech 8 možných modelů:



KB = pravidla Wumpusovy jeskyně + pozorování

α_1 = "[1, 2] je bezpečné pole" $KB \models \alpha_1$

α_2 = "[2, 2] je bezpečné pole" $KB \not\models \alpha_2$

kontrola modelů → jednoduchý způsob logické inference

Pravdivostní tabulka pro inferenci

$V_{1,1}$	$V_{2,1}$	$J_{1,1}$	$J_{1,2}$	$J_{2,1}$	$J_{2,2}$	$J_{3,1}$	KB	α_1
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

$KB \models \alpha_1$

KB = pravidla Wumpusovy jeskyně + pozorování

α_1 = "[1, 2] je bezpečné pole"

SliDo

Inference kontrolou modelů

Kontrola všech modelů *do hloubky* je **bezesporná** a **úplná** (pro konečný počet výrokových symbolů)

function TT-ENTAILS?(*KB*, α) *# vrací True, pokud KB* $\models \alpha$
    symbols ← **list_of_symbols**(*KB* $\cup\ \alpha$)
  return TT-CHECK-ALL(*KB*, α, *symbols*, {})

function TT-CHECK-ALL(*KB*, α, *symbols*, *model*) *# shoduje se* α *s KB na modelu?*
  if EMPTY?(*symbols*) **then**
    if PL-TRUE?(*KB*,*model*) **then return** PL-TRUE?(α,*model*)
vrací **true**, pokud je **Alpha** pravdivá v **Modelu**
    else return True *# když je KB nepravdivá, vždy vrací True*
  else
    P ← *symbols*.**first**
    rest ← *symbols*.**rest**
    return (TT-CHECK-ALL(*KB*,α,*rest*,*model* $\cup$ {*P*=True})
      and
      TT-CHECK-ALL(*KB*,α,*rest*,*model* $\cup$ {*P*=False}))

$O(2^n)$ pro n symbolů, NP-úplný problém

Inference kontrolou modelů

- kontrola modelů (*model checking*)
  - procházení pravdivostní tabulky (vždycky exponenciální v n)
  - vylepšené prohledávání s navracením (*improved backtracking*), např. DPLL
  - heuristické prohledávání prostoru modelů (bezesporné, ale neúplné)

Inference kontrolou modelů

- kontrola modelů (*model checking*)
  - procházení pravdivostní tabulky (vždycky exponenciální v n)
  - vylepšené prohledávání s navracením (*improved backtracking*), např. DPLL
  - heuristické prohledávání prostoru modelů (bezesporné, ale neúplné)
- aplikace inferenčních pravidel
  - legitimní (bezesporné) generování nových výroků ze starých
  - **důkaz** = sekvence aplikací inferenčních pravidel např. pravidla pro logickou ekvivalenci

    je možné použít inferenční pravidla jako operátory ve standardních prohledávacích algoritmech
  - typicky vyžaduje překlad vět do normální formy

Dopředné a zpětné řetězení

pokud $KB =$ konjunkce Hornových klauzulí

$\Rightarrow$ můžeme použít dopředné a zpětné řetězení

Hornova klauzule $= \begin{cases} \text{výrokový symbol; nebo} \\ \text{(konjunkce symbolů)} \Rightarrow \text{symbol} \end{cases}$

např.: $KB = C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

Dopředné a zpětné řetězení

pokud KB = konjunkce Hornových klauzulí

$\Rightarrow$ můžeme použít dopředné a zpětné řetězení

Hornova klauzule $= \begin{cases} \text{výrokový symbol; nebo} \\ \text{(konjunkce symbolů)} \Rightarrow \text{symbol} \end{cases}$

např.: $KB = C \land (B \Rightarrow A) \land (C \land D \Rightarrow B)$

pravidlo **Modus Ponens** – pro KB z Hornových klauzulí je úplné

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta}{\beta}$$

Dopředné a zpětné řetězení

pokud KB = konjunkce Hornových klauzulí

$\Rightarrow$ můžeme použít dopředné a zpětné řetězení

Hornova klauzule $= \begin{cases} \text{výrokový symbol; nebo} \\ \text{(konjunkce symbolů)} \Rightarrow \text{symbol} \end{cases}$

např.: $KB = C \land (B \Rightarrow A) \land (C \land D \Rightarrow B)$

pravidlo **Modus Ponens** – pro KB z Hornových klauzulí je úplné

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta}{\beta}$$

algoritmy dopředného nebo zpětného řetězení jsou přirozené a mají **lineární** časovou složitost vzhledem k velikosti báze znalostí

Dopředné řetězení

Idea: aplikuj pravidlo, jehož premisy jsou splněné v KB
přidej jeho důsledek do KB
pokračuj do doby, než je nalezena odpověď

Dopředné řetězení

Idea: aplikuj pravidlo, jehož premisy jsou splněné v KB
přidej jeho důsledek do KB
pokračuj do doby, než je nalezena odpověď

KB:

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

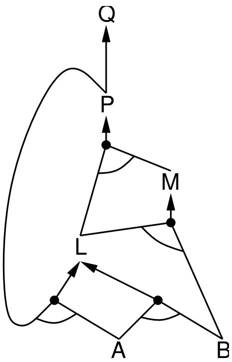
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

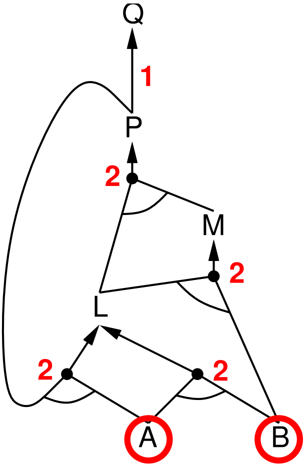
B

AND-OR graf KB:



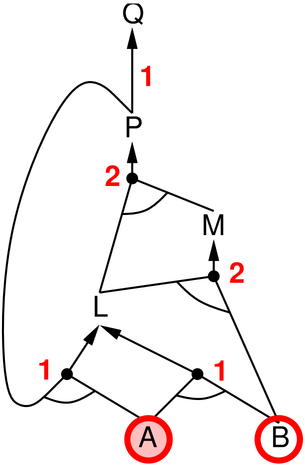
Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



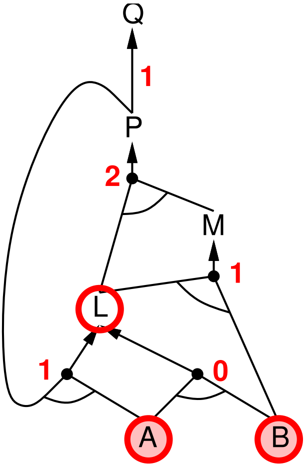
Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



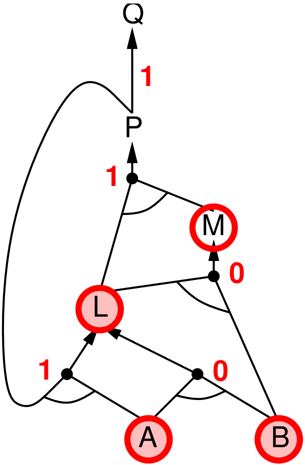
Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



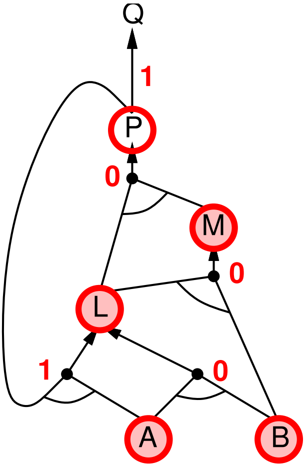
Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



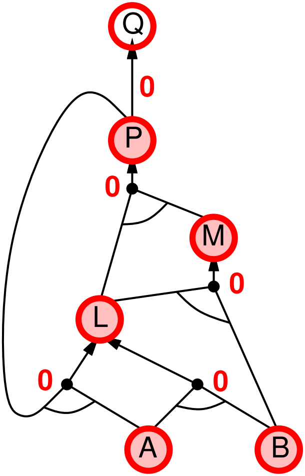
Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



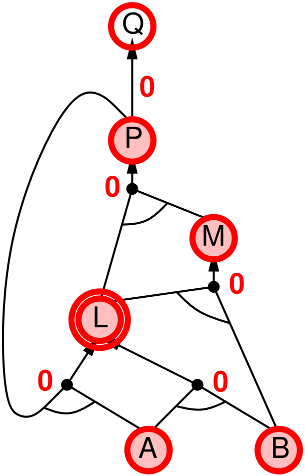
Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



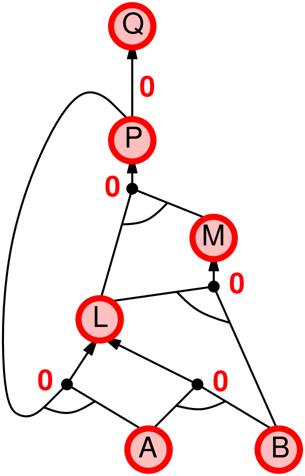
Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



Dopředné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



Algoritmus dopředného řetězení

```
function PL-FC-ENTAILS?(KB, q)          # vrací True nebo False
    # KB je množina klauzulí, q je symbol/dotaz
    count ← {c: num_premise_symbols(c) for ∀c ∈ list_of_clauses(KB ∪ q)}
    inferred ← {s: False for ∀s ∈ list_of_symbols(KB ∪ q)}
    queue ← facts_queue(KB)
    while queue ≠ ∅ do
        p ← queue.pop
        if p = q then return True
        if inferred[p] = False then
            inferred[p] ← True
            foreach clause c in KB where p is in c.premise do
                count[c] −= 1
                if count[c] = 0 then queue.add(c.conclusion)
    return False
```

počet symbolů v premisi každé klauzule

fronta, na začátku obsahuje **fakta** z *KB*

Zpětné řetězení

Idea: pracuje zpětně od dotazu q
zkontroluj, jestli není q už známo
dokaž zpětným řetězením všechny premisy nějakého pravidla, které má q jako důsledek

kontrola cyklů – pro každý podcíl se nejprve podívej, jestli už nebyl řešen (tj. pamatuje si *true* i *false* výsledek)

Zpětné řetězení – příklad

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

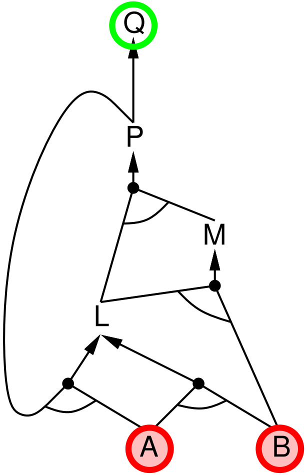
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

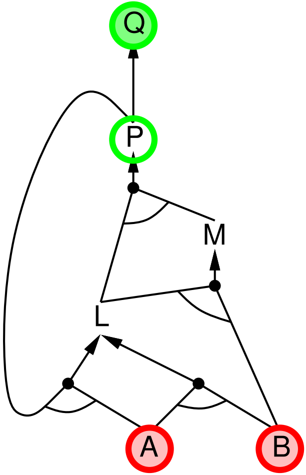
A

B



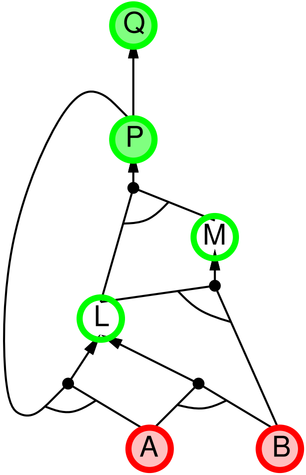
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



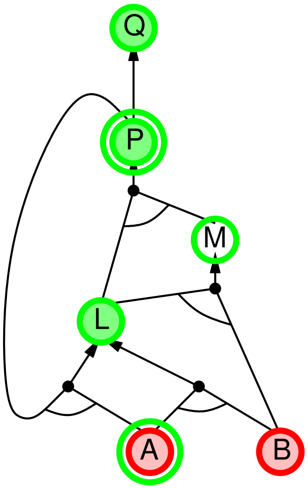
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



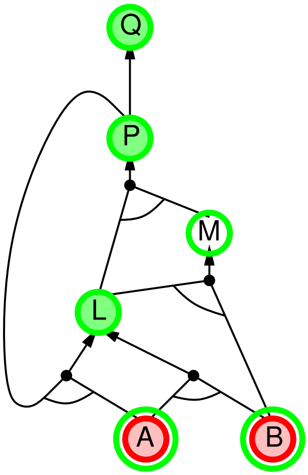
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



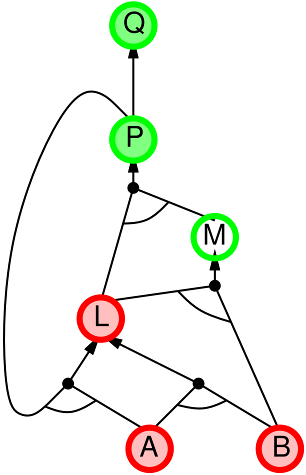
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



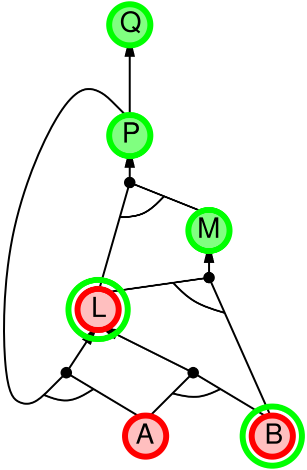
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



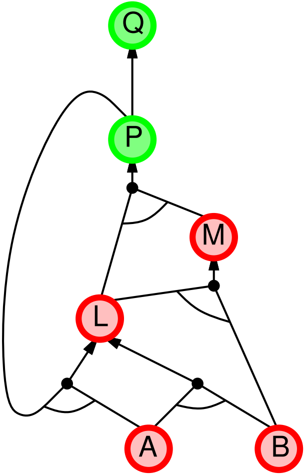
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



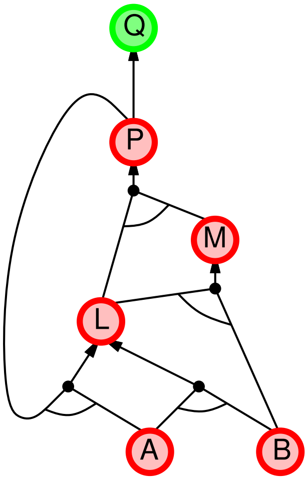
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



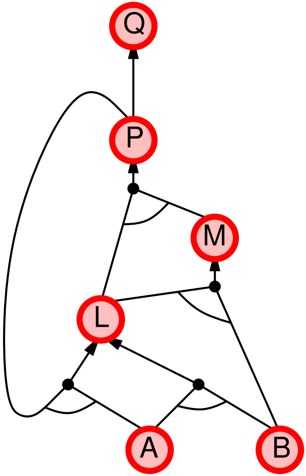
Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



Zpětné řetězení – příklad

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



Porovnání dopředného a zpětného řetězení

- dopředné řetězení je řízeno daty
  - automatické, nevědomé zpracování
  - např. rozpoznávání objektů, rutinní rozhodování
  - může udělat hodně nadbytečné práce bez vztahu k dotazu/cíli

- zpětné řetězení je řízeno dotazem
  - vhodné pro hledání odpovědí na konkrétní dotaz
  - např. "Kde jsou moje klíče?", "Jak se mám přihlásit na PGS?"
  - složitost zpětného řetězení může být mnohem menší než lineární vzhledem k velikosti *KB*

SliDo

Davis-Putnam-Logemann-Loveland (DPLL)

dopředné a zpětné řetězení – pouze pro Hornovy klauzule

DPLL – vylepšené prohledávání s navracením, *improved backtracking*
základní algoritmus typu SAT solver

neomezuje KB, pracuje s formulemi v CNF

připomínka CNF

- klauzule – disjunkce literálů
- literál – symbol nebo negovaný symbol

$$\begin{aligned}&(\underline{\neg D} \lor \underline{\neg B} \lor \underline{C})\\ &\quad\land (B \lor \neg A \lor \neg C)\\ &\quad\land (\neg C \lor \neg B \lor E)\\ &\quad\land (E \lor \neg D \lor B)\\ &\quad\land (B \lor E \lor \neg C)\end{aligned}$$

literály $\neg D$, $\neg B$ a C

pět klauzulí spojených konjunkcí $\land$

DPLL

DPLL vylepšení:

1. **dřívější ukončení** – DPLL detekuje pravdivost věty už z částečného modelu
   klauzule (disjunkce literálů) je pravdivá $\Leftrightarrow$ alespoň jeden literál je pravdivý
   např. $\neg D \vee \neg B \vee C$ $D = False$, $B = False$ nebo $C = True$

DPLL

DPLL vylepšení:

1. **dřívější ukončení** – DPLL detekuje pravdivost věty už z částečného modelu
   klauzule (disjunkce literálů) je pravdivá $\Leftrightarrow$ alespoň jeden literál je pravdivý
   např. $\neg D \vee \neg B \vee C$ $D = False$, $B = False$ nebo $C = True$
2. **heuristika čistých symbolů**, *pure symbol heuristic*
   čistý symbol je ve všech klauzulích buď vždy pozitivní nebo vždy negovaný
   v $KB = \{(A \vee \neg B), (\neg B \vee \neg C), (C \vee A)\}$ jsou A a B čisté symboly, C není

   když čistý symbol nastavíme na True hodnotu literálu (tedy negovaný symbol=False) $\Rightarrow$neporušíme splnitelnost KB

DPLL

DPLL vylepšení:

1. **dřívější ukončení** – DPLL detekuje pravdivost věty už z částečného modelu
   klauzule (disjunkce literálů) je pravdivá $\Leftrightarrow$ alespoň jeden literál je pravdivý
   např. $\neg D \vee \neg B \vee C$ $D = False$, $B = False$ nebo $C = True$
2. **heuristika čistých symbolů**, *pure symbol heuristic*
   čistý symbol je ve všech klauzulích buď vždy pozitivní nebo vždy negovaný
   v $KB = \{(A \vee \neg B), (\neg B \vee \neg C), (C \vee A)\}$ jsou A a B čisté symboly, C není

   když čistý symbol nastavíme na True hodnotu literálu (tedy negovaný symbol=False) $\Rightarrow$neporušíme splnitelnost KB
3. **heuristika jednotkových klauzulí**, *unit clause heuristic*
   jednotková klauzule – obsahuje právě jeden literál (nebo ostatní dříve určeny jako False)
   pro splnění klauzule musí být daný literál pozitivní

DPLL

function $\text{DPLL-S}\textsc{atisfiable}?(s)$ # *je výroková formule s splnitelná?*
  clauses ← **cnf_set_of_clauses**(s) # *množina klauzulí z CNF(s)*
  symbols ← **list_of_symbols**(s) # *seznam symbolů v s*
  return DPLL(*clauses* , *symbols*, {})

function DPLL(*clauses* , *symbols*, *model*) # *jsou klauzule clauses pravdivé v modelu?*
  if *clauses* $= \emptyset$ **then return** True # *prázdná množina klauzulí je pravdivá*
  if $\forall$ *clause* $\in$ *clauses*: *clause* is True **in** *model* **then return** True
  if $\exists$ *clause* $\in$ *clauses*: *clause* is False **in** *model* **then return** False
  P, *value* ← $\text{F}\textsc{ind}\text{-P}\textsc{ure}\text{-S}\textsc{ymbol}$(*symbols*, *clauses* , *model*)
  if $P \neq$ **None then return** DPLL(*clauses*, *symbols* − *P*, *model* ∪ {*P*=*value*})
  P, *value* ← $\text{F}\textsc{ind}\text{-U}\textsc{nit}\text{-C}\textsc{lause}$(*clauses*, *model*)
  if $P \neq$ **None then return** DPLL(*clauses*, *symbols* − *P*, *model* ∪ {*P*=*value*})
  P ← *symbols*.**first** ; *rest* ← *symbols*.**rest**
  return DPLL(*clauses* , *rest* , *model* ∪ {*P*=True}) **or**
      DPLL(*clauses* , *rest* , *model* ∪ {*P*=False})

DPLL– další možná vylepšení

- analýza komponent – identifikace nepropojených klauzulí a jejich separátní zpracování
- uspořádání symbolů a hodnot – jako u splňování podmínek, např. nejčastěji použitý symbol
- inteligentní navracení – zapamatování konfliktů, návrat k nejbližšímu
- náhodné restarty – po určité době začít výpočet od začátku
- chytré indexování

s těmito vylepšeními zvládnou moderní SAT algoritmy problémy v rozsahu desítek milionů symbolů

Rezoluce

obecný inferenční algoritmus – **rezoluce**

- dopředné a zpětné řetězení – úplné pro Hornovy klauzule, ale neúplné pro obecnou KB
- rezoluce – **úplná** (pro důkaz sporem) pro výrokovou i predikátovou logiku
- logické programování – SLD rezoluce

Rezoluce

rezoluční vyvozování je pouze částečně rozhodnutelné:

- může najít důkaz α, když $KB \models \alpha$
- nemůže vždy dokázat, že $KB \not\models \alpha$
  viz *problém zastavení* – důkazová procedura nemusí skončit
  nejde použít pro generování, pouze pro vyvracení

rezoluce je důkaz sporem:

pro důkaz $KB \models \alpha$ ukážeme, že $KB \wedge \neg\alpha$ je nesplnitelné

rezoluce používá KB, $\neg\alpha$ v konjunktivní normální formě (CNF), např.:

$$(P \vee Q) \Rightarrow (Q \Leftrightarrow R) \quad \equiv \quad \begin{array}{ll} & (\neg P \vee \neg Q \vee R) \\ \wedge & (\neg P \vee Q \vee \neg R) \\ \wedge & (\neg Q \vee R) \end{array}$$

Rezoluční pravidlo

algoritmus je založen na opakované aplikaci **rezolučního pravidla** – ze dvou klauzulí odvoď novou klauzuli

$$\begin{tikzcd} C_1 \arrow[dr] & & C_2 \arrow[dl] \\ & C & \end{tikzcd}$$

- klauzule: $C_1 = P_1 \vee P_2 \vee \ldots \vee P_n$
  a $C_2 = \neg P_1 \vee Q_1 \vee Q_2 \vee \ldots \vee Q_m$
- výsledek (rezolventa):
  $C = P_2 \vee P_3 \vee \ldots \vee P_n \vee Q_1 \vee Q_2 \vee \ldots \vee Q_m$
- vyruší se opačné literály P_1 a $\neg P_1$
  pokud nic nezbude, výsledkem je prázdná klauzule $\square$ (odpovídá False)

Rezoluční pravidlo

algoritmus je založen na opakované aplikaci **rezolučního pravidla** – ze dvou klauzulí odvoď novou klauzuli

$$\begin{tikzcd}
C_1 \arrow[dr] & & C_2 \arrow[dl] \\
& C &
\end{tikzcd}$$

- klauzule: $C_1 = P_1 \vee P_2 \vee \ldots \vee P_n$
  a $C_2 = \neg P_1 \vee Q_1 \vee Q_2 \vee \ldots \vee Q_m$
- výsledek (rezolventa):
  $C = P_2 \vee P_3 \vee \ldots \vee P_n \vee Q_1 \vee Q_2 \vee \ldots \vee Q_m$
- vyruší se opačné literály P_1 a $\neg P_1$
  pokud nic nezbude, výsledkem je prázdná klauzule $\square$ (odpovídá False)

postup **rezolučního důkazu tvrzení** F:

– začneme s $\neg F$
– rezolvujeme s klauzulí z KB (která obsahuje F)
– opakujeme až do odvození prázdné klauzule $\square$
– když se to podaří $\rightarrow$ došli jsme ke sporu (pro $\neg F$) $\rightarrow$ **musí platit** F

Rezoluce

function $\mathrm{PL\text{-}R{\scriptsize ESOLUTION}}(KB, \alpha)$ *# vrací True nebo False podle toho, zda* $KB \models \alpha$ *nebo* $KB \not\models \alpha$

    clauses ← **cnf_set_of_clauses**($KB \wedge \neg\alpha$)
    new_clauses ← $\emptyset$
    while True **do**
        foreach pair of clauses C_i, $C_j \in$ *clauses* **do**
            resolvent ← $\mathrm{PL\text{-}R{\scriptsize ESOLVE}}(C_i, C_j)$
            if *resolvent* $= \square$ **then return** True
            new_clauses ← *new_clauses* $\cup$ {*resolvent*}
        if *new_clauses* $\subseteq$ *clauses* **then return** False
        clauses ← *clauses* $\cup$ *new_clauses*

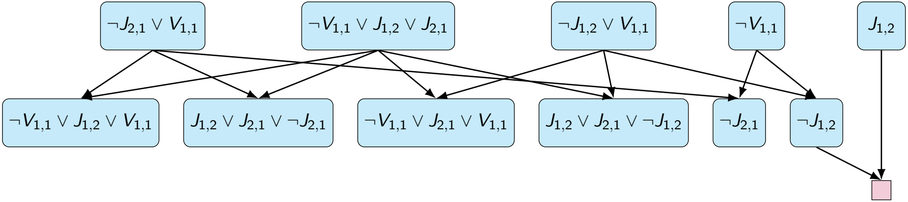
rezoluce pokračuje v odvozování dokud nenastane situace, kdy:

- nejdou vytvořit žádné nové klauzule, pak $KB \not\models \alpha$
- dvě klauzule se rezolvují do prázdné klauzule $\square$, pak $KB \models \alpha$

Rezoluce – příklad z Wumpusovy jeskyně

báze znalostí *KB*:

- pravidlo $V_{1,1} \Leftrightarrow (J_{1,2} \vee J_{2,1})$
  $(\neg J_{2,1} \vee V_{1,1}) \wedge (\neg V_{1,1} \vee J_{1,2} \vee J_{2,1}) \wedge (\neg J_{1,2} \vee V_{1,1})$
- vnímání $\neg V_{1,1}$
- dotaz (co se má dokázat) $\neg J_{1,2}$?



Rezoluce – příklad s výběrem klauzulí

- pravidla
  - mráz $\wedge$ srážky $\Rightarrow$ sněží
    ¬mráz $\vee$ ¬srážky $\vee$ sněží
  - Leden $\Rightarrow$ mráz
    ¬Leden $\vee$ mráz
  - mraky $\Rightarrow$ srážky
    ¬mraky $\vee$ srážky
- fakta – Leden, mraky
- dotaz (co se má dokázat)
  - **sněží**?
    ¬sněží

Důkaz tvrzení "sněží"

S – sněží, s – srážky, m – mráz, L – Leden, M – mraky

$\neg m \vee \neg s \vee S$
$\neg L \vee m$
$\neg M \vee s$
$L,\ M$

$\neg S \qquad \neg m \vee \neg s \vee S$

$\neg m \vee \neg s \qquad \neg L \vee m$

$\neg L \vee \neg s \qquad \neg M \vee s$

$\neg L \vee \neg M \qquad L$

$\neg M \qquad M$

$\square$

SliDo

Obsah

Kontrola modelů

teoreticky můžeme určit **všechny modely** výčtem ze slovníku KB:

Kontrola modelů

teoreticky můžeme určit **všechny modely** výčtem ze slovníku KB:

pro počet objektů $n = 1, \ldots, (\infty)$

Kontrola modelů

teoreticky můžeme určit **všechny modely** výčtem ze slovníku KB:

- pro počet objektů $n = 1, \ldots, (\infty)$
  - pro každý k-ární predikát P_k ze slovníku

Kontrola modelů

teoreticky můžeme určit **všechny modely** výčtem ze slovníku KB:

- pro počet objektů $n = 1, \ldots, (\infty)$
  - pro každý k-ární predikát P_k ze slovníku
    - pro každou možnou k-ární relaci na n objektech

Kontrola modelů

teoreticky můžeme určit **všechny modely** výčtem ze slovníku KB:

- pro počet objektů $n = 1, \ldots, (\infty)$
  - pro každý k-ární predikát P_k ze slovníku
    - pro každou možnou k-ární relaci na n objektech
      - pro každý konstantní symbol C ze slovníku

Kontrola modelů

teoreticky **můžeme** určit **všechny modely** výčtem ze slovníku KB:

- pro počet **objektů** $n = 1, \ldots, (\infty)$
  - pro každý k-ární **predikát** P_k ze slovníku
    - pro každou možnou k-ární **relaci** na n objektech
      - pro každý **konstantní symbol** C ze slovníku
        - pro každou volbu **referenta** pro C z n objektů ...

Kontrola modelů

teoreticky můžeme určit **všechny modely** výčtem ze slovníku KB:

pro počet objektů $n = 1, \ldots, (\infty)$
- pro každý k-ární predikát P_k ze slovníku
  - pro každou možnou k-ární relaci na n objektech
    - pro každý konstantní symbol C ze slovníku
      - pro každou volbu referenta pro C z n objektů …

prakticky je *kontrola modelů* **nepoužitelná**

Kontrola modelů

teoreticky můžeme určit **všechny modely** výčtem ze slovníku KB:

- pro počet objektů $n = 1, \ldots, (\infty)$
  - pro každý k-ární predikát P_k ze slovníku
    - pro každou možnou k-ární relaci na n objektech
      - pro každý konstantní symbol C ze slovníku
        - pro každou volbu referenta pro C z n objektů ...

prakticky je *kontrola modelů* **nepoužitelná**

inference je možná pouze podle inferenčních pravidel (dopředné/zpětné řetězení, rezoluce, ...)

Unifikace – kvantifikátory

aplikace inferenčních pravidel – jak řešit kvantifikátory?

$\exists$ kvantifikátor – řeší Skolemizace (převod PL1 do CNF)

$\forall$ kvantifikátor – obecně proměnnou můžeme nahradit za term bez proměnných (*ground term*)

Unifikace – kvantifikátory

aplikace inferenčních pravidel – jak řešit kvantifikátory?

$\exists$ kvantifikátor – řeší Skolemizace (převod PL1 do CNF)

$\forall$ kvantifikátor – obecně proměnnou můžeme nahradit za term bez proměnných (*ground term*)

např. v KB
$$\forall x\ King(x) \land Greedy(x) \Rightarrow Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

první větu můžeme nahradit
$$King(John) \land Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$$

Unifikace – unifikace

$\forall x \; King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
$King(John)$
$Greedy(John)$
$Brother(Richard, John)$

náhrada všech objektů za $\forall x$ – velice neefektivní

lépe – vybírat hodnoty (substituce), které splňují literály **premisy**:

- $\forall x \; King(x)$ splňuje pouze substituce $\sigma = \{x/John\}$
- σ vyhovuje i $\forall x \; Greedy(x)$
- a tedy platí i závěr $Evil(x)\sigma = Evil(John)$

nemusíme uvažovat $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$, protože neplatí $King(Richard)$

Unifikace – unifikace

potřebujeme “hledač” substitucí – algoritmus **unifikace**

unify$(\alpha, \beta) = \sigma$ *taková, že* $\alpha\sigma = \beta\sigma$

Unifikace – unifikace

potřebujeme "hledač" substitucí – algoritmus **unifikace**

unify$(\alpha, \beta) = \sigma$ *taková, že* $\alpha\sigma = \beta\sigma$

např. $\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(\mathit{John}, \mathit{Jane})) = \{x/\mathit{Jane}\}$

Unifikace – unifikace

potřebujeme "hledač" substitucí – algoritmus **unifikace**

$\textbf{unify}(\alpha, \beta) = \sigma$ *taková, že* $\alpha\sigma = \beta\sigma$

např. $\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(\mathit{John}, \mathit{Jane})) = \{x/\mathit{Jane}\}$
$\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(y, \mathit{Bill})) = \{x/\mathit{Bill}, y/\mathit{John}\}$

Unifikace – unifikace

potřebujeme "hledač" substitucí – algoritmus **unifikace**

unify$(\alpha, \beta) = \sigma$ *taková, že* $\alpha\sigma = \beta\sigma$

např. $\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(\mathit{John}, \mathit{Jane})) = \{x/\mathit{Jane}\}$
$\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(y, \mathit{Bill})) = \{x/\mathit{Bill}, y/\mathit{John}\}$
$\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(y, \mathit{Mother}(y))) = \{y/\mathit{John}, x/\mathit{Mother}(\mathit{John})\}$

Unifikace – unifikace

potřebujeme "hledač" substitucí – algoritmus **unifikace**

unify$(\alpha, \beta) = \sigma$ *taková, že* $\alpha\sigma = \beta\sigma$

např. $\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(\mathit{John}, \mathit{Jane})) = \{x/\mathit{Jane}\}$
$\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(y, \mathit{Bill})) = \{x/\mathit{Bill}, y/\mathit{John}\}$
$\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(y, \mathit{Mother}(y))) = \{y/\mathit{John}, x/\mathit{Mother}(\mathit{John})\}$
$\mathit{unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(\mathit{Bill}, \mathit{Elizabeth})) = \mathit{failure}$

Unifikace – unifikace

potřebujeme "hledač" substitucí – algoritmus **unifikace**

unify$(\alpha, \beta) = \sigma$ *taková, že* $\alpha\sigma = \beta\sigma$

např. $unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
$unify(Knows(John, x), Knows(y, Bill)) = \{x/Bill, y/John\}$
$unify(Knows(John, x), Knows(y, Mother(y))) = \{y/John, x/Mother(John)\}$
$unify(Knows(John, x), Knows(Bill, Elizabeth)) = failure$
$unify(Knows(John, x), Knows(x, Elizabeth)) = failure$

Unifikace – unifikace

potřebujeme "hledač" substitucí – algoritmus **unifikace**

unify$(\alpha, \beta) = \sigma$ *taková, že* $\alpha\sigma = \beta\sigma$

např. $unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
$unify(Knows(John, x), Knows(y, Bill)) = \{x/Bill, y/John\}$
$unify(Knows(John, x), Knows(y, Mother(y))) = \{y/John, x/Mother(John)\}$
$unify(Knows(John, x), Knows(Bill, Elizabeth)) = failure$
$unify(Knows(John, x), Knows(x, Elizabeth)) = failure$

unify v poslední větě lze vyřešit přejmenováním proměnných
$unify(Knows(John, x), Knows(x_2, Jane)) = \{x/Jane, x_2/John\}$

platných unifikací existuje víc, unify vrací **nejobecnější unifikátor**

Zobecněné Modus Ponens

základní inferenční pravidlo – **zobecněné Modus Ponens** (*Generalized Modus Ponens, GMP*)

$$\frac{p_1{}',\ \ p_2{}',\ \ldots,\ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma}$$

kde $\sigma = \bigcup_i \mathit{unify}(p_i{}', p_i)$ pro atomické formule p_i, $p_i{}'$ a q s přejmenováním kolizních proměnných

Zobecněné Modus Ponens

základní inferenční pravidlo – **zobecněné Modus Ponens** (*Generalized Modus Ponens, GMP*)

$$\frac{p_1{}',\ \ p_2{}',\ \ldots,\ p_n{}',\ \ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{q\sigma}$$

kde $\sigma = \bigcup_i \mathit{unify}(p_i{}', p_i)$ pro atomické formule p_i, $p_i{}'$ a q s přejmenováním kolizních proměnných

např.

$$\frac{\mathit{King}(\mathit{John}), \mathit{Greedy}(\mathit{John}), (\mathit{King}(x) \land \mathit{Greedy}(x) \Rightarrow \mathit{Evil}(x))}{\mathit{Evil}(x)\{x/\mathit{John}\}}$$

Zobecněné Modus Ponens

- používá unifikaci
- tato úprava MP se označuje jako lifting (pozvedává MP z jednoduché výrokové logiky bez proměnných na logiku predikátovou)
- hlavní výhoda proti výčtu všech termů – jen ty substituce, které jsou pro pravidlo nutné
- GMP je využito v upravených verzích inferenčních algoritmů – dopředné/zpětné řetězení, rezoluce
  hlavní úpravy – použití unifikace, při True vrací i možné substituce

$$\frac{Animal(F(x)) \lor Loves(G(x), x) \qquad \neg Loves(u, v) \lor \neg Kills(u, v)}{Animal(F(x)) \lor \neg Kills(G(x), x)}$$

$\sigma = \{u/G(x), v/x\}$

- dopředné/zpětné řetězení je pro PL1 neúplné, rezoluce je úplná, i když jen částečně rozhodnutelná (nemusí skončit pro nepravdivé tvrzení)

Shrnutí

logický agent aplikuje **inferenci** na **bázi znalostí** pro vyvození nových znalostí a tvorbu rozhodnutí

základní koncepty logiky:

syntaxe: formální struktura vět
sémantika: pravdivost vět podle modelů
vyplývání: nutná pravdivost věty v závislosti na jiné větě

inference: vyvození věty z jiných vět
bezespornost: inference produkuje jen vyplývající věty
úplnost: inference vyprodukuje ∀ vyplývající věty

výroková logika nemá dostatečnou expresivitu

predikátová logika prvního řádu:

- syntaxe: konstanty, funkce, predikáty, rovnost, kvantifikátory
- větší expresivita – dostatečná pro Wumpusovu jeskyni
- "poslední" logika, pro kterou existuje bezesporná a úplná inference (Gödelovy věty o neúplnosti)

jiné možné logiky:

jazyk	ontologie	pravdivostní hodnoty
výroková logika	fakty	true/false/$\perp$
predikátová logika 1. řádu	fakty, objekty, relace	true/false/$\perp$
temporální logika	fakty, objekty, relace, čas	true/false/$\perp$
teorie pravděpodobnosti	fakty	míra pravděpodobnosti $\in [0,1]$
fuzzy logika	míra pravdivosti $\in [0,1]$	intervaly hodnot