

# Inference ve výrokové a predikátové logice

Aleš Horák

E-mail: [hales@fi.muni.cz](mailto: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:

inženýrní stroj (inference engine)

algoritmy nezávislé na doméně

báze znalostí (knowledge base)

znalosti o doméně

**inference** – vyvozování důsledků z báze znalostí

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

**Bezspornost:**  $i$  je bezsporná  $\Leftrightarrow \forall KB \vdash_i \alpha \Rightarrow KB \models \alpha$

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

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

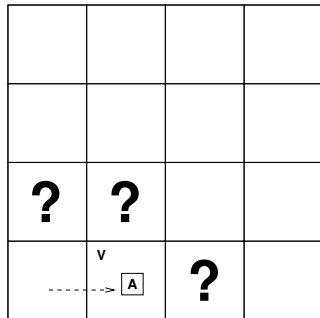
# Historie logického vyvozování

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:

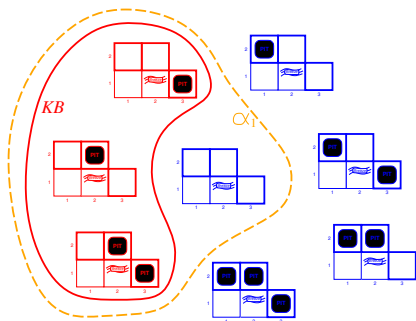
- v  $[1, 1]$  nedetekováno nic
  - krok doprava, v  $[2, 1]$  Vánek
- uvažujeme možné **modely** pro '?'  
(budou nás zajímat jen Jámy)



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

$\alpha_1$  = “[1, 2] je bezpečné pole”       $KB \models \alpha_1$

$\alpha_2$  = “[2, 2] je bezpečné pole”       $KB \not\models \alpha_2$

kontrola modelů  $\rightarrow$  jednoduchý způsob logické inference

# Pravdivostní tabulka pro inferenci

$V_{1,1}$	$V_{2,1}$	$J_{1,1}$	$J_{1,2}$	$J_{2,1}$	$J_{2,2}$	$J_{3,1}$	$KB$	$\alpha_1$
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	<u>true</u>	<u>true</u>
false	true	false	false	false	true	false	<u>true</u>	<u>true</u>
false	true	false	false	false	true	true	<u>true</u>	<u>true</u>
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

$\alpha_1$  = “[1, 2] je bezpečné pole”

## Inference kontrolou modelů

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

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

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

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

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

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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**

*KB*:

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

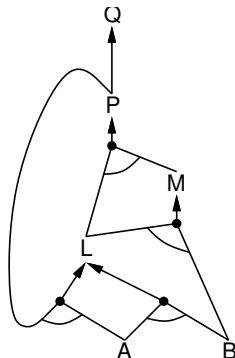
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

*A*

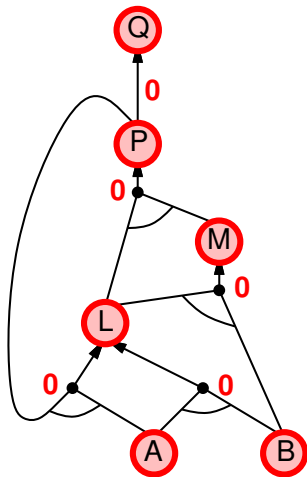
*B*

AND-OR graf *KB*:



## Dopředné řetězení – příklad

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



# Algoritmus dopředného řetězení

**function** PL-FC-ENTAILS?(*KB*, *q*) *# vrací True nebo False*

*# KB je množina klauzulí, q je symbol/dotaz*

*count*  $\leftarrow \{c: \text{num\_premise\_symbols}(c) \text{ for } \forall c \in \text{list\_of\_clauses}(KB \cup q)\}$

*inferred*  $\leftarrow \{s: \text{False} \text{ for } \forall s \in \text{list\_of\_symbols}(KB \cup q)\}$

*queue*  $\leftarrow \text{facts\_queue}(KB)$

**while** *queue*  $\neq \emptyset$  **do**

*p*  $\leftarrow \text{queue.pop}$

**if** *p* = *q* **then return** True

**if** *inferred* [*p*] = False **then**

*inferred* [*p*]  $\leftarrow$  True

**foreach** clause *c* **in** *KB* where *p* is **in** *c*.premise **do**

*count*[*c*]  $\text{--} = 1$

**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

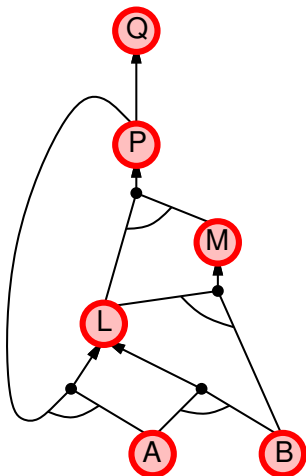
$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$
$$B$$


# 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  $KB$

# 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

$$\begin{aligned}
 &(\neg D \vee \neg B \vee C) \\
 &\wedge (B \vee \neg A \vee \neg C) \\
 &\wedge (\neg C \vee \neg B \vee E) \\
 &\wedge (E \vee \neg D \vee B) \\
 &\wedge (B \vee E \vee \neg C)
 \end{aligned}$$

literály  $\neg D$ ,  $\neg B$  a  $C$

pět klauzulí spojených konjunkcí  $\wedge$



# 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.  $\neg D \vee \neg B \vee C$   $D = \text{False}, B = \text{False}$  nebo  $C = \text{True}$
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 DPLL-SATISFIABLE?(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 = ∅ then return True      # prázdná množina klauzulí je pravdivá
  if ∀ clause ∈ clauses: clause is True in model then return True
  if ∃ clause ∈ clauses: clause is False in model then return False
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P ≠ None then return DPLL(clauses, symbols − P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P ≠ 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  $\alpha$ , 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**

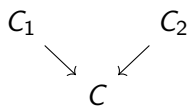
rezoluce používá  $KB$ ,  $\neg\alpha$  v **konjunktivní normální formě** (CNF), např.:

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

## Rezoluční pravidlo

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

- ▶ klauzule:  $C_1 = P_1 \vee P_2 \vee \dots \vee P_n$   
a  $C_2 = \neg P_1 \vee Q_1 \vee Q_2 \vee \dots \vee Q_m$



- ▶ výsledek (rezolventa):  
 $C = P_2 \vee P_3 \vee \dots \vee P_n \vee Q_1 \vee Q_2 \vee \dots \vee Q_m$
- ▶ vyruší se opačné literály  $P_1$  a  $\neg P_1$   
pokud nic nezůstane, 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 PL-RESOLUTION( $KB, \alpha$ ) # vrací True nebo False podle toho, zda  $KB \models \alpha$  nebo  $KB \not\models \alpha$   
   $clauses \leftarrow \text{cnf\_set\_of\_clauses}(KB \wedge \neg \alpha)$   
   $new\_clauses \leftarrow \emptyset$   
  while True do  
    foreach pair of clauses  $C_i, C_j \in clauses$  do  
       $resolvent \leftarrow \text{PL-RESOLVE}(C_i, C_j)$   
      if  $resolvent = \square$  then return True  
       $new\_clauses \leftarrow new\_clauses \cup \{resolvent\}$   
    if  $new\_clauses \subseteq clauses$  then return False  
     $clauses \leftarrow clauses \cup new\_clauses$ 
```

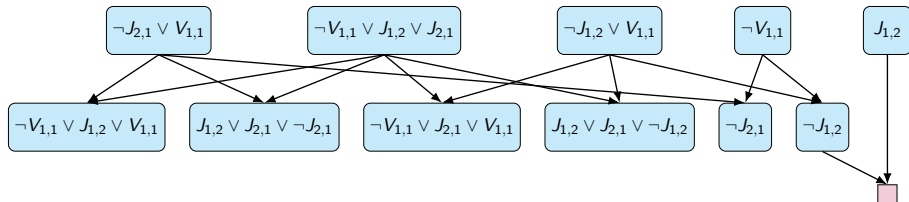
rezoluce pokračuje v odvozování dokud nenastane situace, kdy:

- ▶ nejdou vytvořit žádné nové klauzule, pak  $KB \not\models \alpha$
- ▶ dvě klauzule se rezolují 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

- $\text{mráz} \wedge \text{srážky} \Rightarrow \text{sněží}$   
 $\neg\text{mráz} \vee \neg\text{srážky} \vee \text{sněží}$
- $\text{Leden} \Rightarrow \text{mráz}$   
 $\neg\text{Leden} \vee \text{mráz}$
- $\text{mraky} \Rightarrow \text{srážky}$   
 $\neg\text{mraky} \vee \text{srážky}$

## ▶ fakta – Leden, mraky

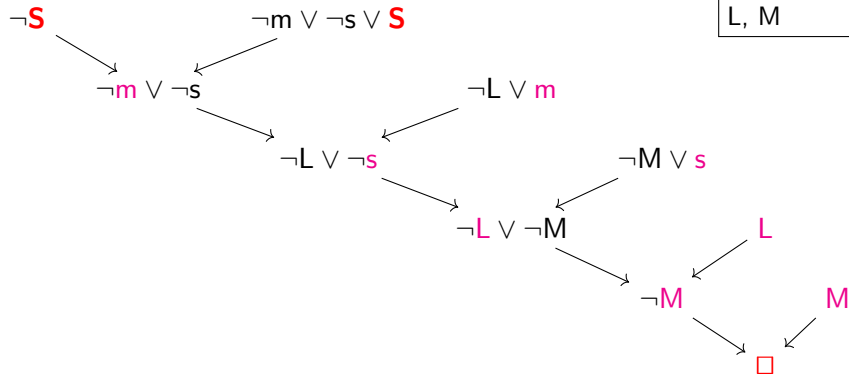
## ▶ dotaz (co se má dokázat)

- **sněží?**  
 $\neg\text{sněží}$

# 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



# Kontrola modelů

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

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

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*)

např. v KB  $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$   
 $\text{King}(\text{John})$   
 $\text{Greedy}(\text{John})$   
 $\text{Brother}(\text{Richard}, \text{John})$

první větu můžeme nahradit

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$   
 $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$

# Unifikace – unifikace

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

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$   
 $\text{King}(\text{John})$   
 $\text{Greedy}(\text{John})$   
 $\text{Brother}(\text{Richard}, \text{John})$

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

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

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

## Unifikace – unifikace

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

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

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

**unify** v poslední větě lze vyřešit **přejmenováním** proměnných

$\text{unify}(\text{Knows}(\text{John}, x), \text{Knows}(x_2, \text{Jane})) = \{x/\text{Jane}, x_2/\text{John}\}$

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

## Zobecněné Modus Ponens

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

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

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

např.

$$\frac{\text{King}(\text{John}), \text{Greedy}(\text{John}), (\text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x))}{\text{Evil}(x)\{x/\text{John}\}}$$

## 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**

$$\frac{\textit{Animal}(F(x)) \vee \textit{Loves}(G(x), x) \quad \neg \textit{Loves}(u, v) \vee \neg \textit{Kills}(u, v)}{\textit{Animal}(F(x)) \vee \neg \textit{Kills}(G(x), x)}$$

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

- ▶ 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  $\forall$  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