28. PROLOG

- je netypový
- je deklarativní – definují se cíle, ne postup jejich dosažení

- logický programovací jazyk, který se využívá v umělé int. a pro zpracování přirozeného jazyka → pro to byl původně navržen (počítač)

- **termy**: proměnné (X, Y, ...), konstanty (3, 10, ...), struktury ($f(t_1, ..., t_n)$)
- **klauzule**: $H :- B_1, B_2, ..., B_n.$
  H.
  - z nich se skládá celý program
  - program je v podstatě soubor klauzulí
  - klauzule jsou Hornovy klauzule → $\neg B_1 \lor \neg B_2 \lor ... \lor \neg B_n \lor H$
    ⇒ buď některá z těch podklauzulí neplatí a nebo to H platí (to dává smysl)

  ⇒ PREDIKÁTOVÁ LOGIKA 1. ŘÁDU

  - jsou převoditelné na implikaci
    $\neg B_1 \lor \neg B_2 \lor ... \lor \neg B_n \lor H \Rightarrow \neg(B_1 \land B_2 \land ... \land B_n) \lor H$
    $\Rightarrow (B_1 \land B_2 \land ... \land B_n) \to H$
    - tedy jestliže jsou splněny všechny B_i podcíle, tak je splněna i ta celá klauzule
  - program je konečná množina Hornových klauzulí
  - klauzule mají jeden pozitivní literál H a žádné nebo více negativních literálů $\neg B_i$
- **cíle**: ?- B
  - dotazujeme se nad tím programem
  - ověřujeme zda daný cíl platí

  ?- vnuk(petr, pavel)

Klauzule:

muž(Petr). muž(Jan).
žena(Anna). žena(Věra).
matka(Anna, Petr). otec(Jan, Anna).
rodič(X,Y) :- matka(X,Y).
rodič(X,Y) :- otec(X,Y).

- v definici klauzule jsou tedy uvedeny podcíle které musí platit aby platil ten hlavní cíl (klauzule) a je mezi nimi tedy AND (∧)

→ toto je podcíl

→ mezi rodič s matkou a rodič s otcem je jakoby OR (∨) a pokud první neplatí tak se provede backtracking a jde se na druhý (backtrackingu může zabránit operátor řezu)

Klauzule jsou:
- **Fakta**: – bez negativního prvku (není tam to $\neg B_i$) (Hornovy klauzule)
  – muž(Jan). → jen 1 pozitivní prvek
- **Pravidla**: – s alespoň jedním negativním prvkem (alespoň jedno to $\neg B_i$) (Hornovy klauzule)
  – vnuk(X,Y) :- dite(X,Z), dite(Z,Y).
    (vnuk – 1 pozitivní prvek; dite(X,Z), dite(Z,Y) – 2 negativní prvky)

přepis:

$\neg$dite(X,Z) ∨ $\neg$dite(Z,Y) ∨ vnuk(X,Y)

+ pro běh klauzulí slouží cíle

– cíle: – hornovy klauzule bez pozitivního prvku
  – ptáme se jestli to platí, chceme ji z dané hypotézy (programu) ověřit
  – ?-vnukem(Petr, Pavel)

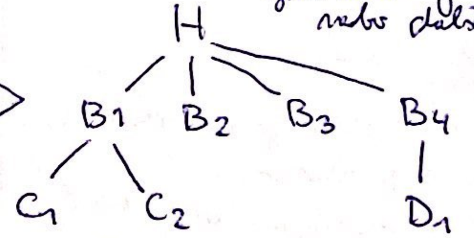
Odvozování

– máme množinu klauzulí (→ program) a pak nějaké cíle které máme dokázat/odvodit
– dělá se to SDL-rezolucí
– dokazuje se nesplnitelnost těch negativních formulí
– vytvoří se rezoluční strom

– hledáme jestli jsou podcíle někde jako fakta nebo další klauzule

H :- B_1, B_2, B_3, B_4.
B_1 :- C_1, C_2.
B_4 :- D_1.

⟹



SDL-rezoluce

– vezme se ten hlavní cíl (→ H) a jeho podcíle (→ B_i) se expandují do hloubky
– tedy v tom programu se procházejí fakty a klauzule shora dolů a hledá se takový fakt či klauzule se kterým by se mohl aktuální podcíl unifikovat
– těla klauzulí se zpracovávají zleva doprava tedy B_1 pak B_2 ... atd.
– jestliže pro každý podcíl bylo možné najít fakt se kterým jej šlo unifikovat a nebo pro něj byla nalezena klauzule která byla tímto stylem zase vyhodnocena a je splněná tak je splněn i celý hlavní cíl
– pokud některý z podcílů selže, tak se provádí zpětné navracení (backtracking) a hledá se jiný odpovídající fakt nebo klauzule (se kterými by se dal ten podcíl zase unifikovat) – pokud se nic takového nenajde, tak selže celá klauzule
– backtracking lze v určitém místě zakázat pomocí operátoru řezu – takže se celý ten dotaz (cíl) vyhodnotí jako selhání a ani se v těch klauzulích a faktech nehledá další pro unifikaci

Unifikace

– základní a jediná operace
– pokud se volají procedury nebo klauzule zadáním cíle a nebo třeba zadáním predikátu = A = B nebo A != B
– přiřazení hodnoty do proměnné X = 5
– test na rovnost
– selektor ze seznamu L=[X|_]
– máme klauzuly otec(X,Y) :- ..... a tím se zadáme cíl
  otec(Jan, Pavel) tak se provede unifikace
  v tomto případě substituce [Jan/X] a [Pavel/Y] a to i do pravé strany klauzule

Vestavěné predikáty

- X is 1+2 přiřazení / unifikace
  (1+2 – vyhodnocení, X is … – unifikace)
- integer(X), atom(X), … test zda X je …
- X == 4 totožnost
- program je DB klauzulí
  → vložení dynamicky do DB (do programu) klauzuli C
  - assert(C)
  - asserta(C) – na začátek
  - assertz(C) – na konec
  - odstranění klauzule C z databáze
    retract(C)
- A = B explicitní unifikace
- fail – vždy selže
- true – vždy uspěje
- not
- ! – vždy uspěje ale jen 1x – při backtrackingu už ne a nelze se přes něj vracet
  ⇒ operátor řezu

Seznamy – heterogenní rekurzivní struktura ⇒ polymorfní typování

- prázdný seznam je atom []
- k reprezentaci neprázdného seznamu slouží funktor • a tedy •(head, tail)
  ↳ má arita 2 – přijímá hlavičku a tělo

např.: •(a, []) je [a]

•(a, •(b, •(c, []))) je [a, b, c]

- píšeme ve hranatých závorkách bez nutnosti pro oddělení head a body |
  tedy [a|[b,c]] je [a, b, c]
  ↳ head je jeden prvek; [b,c] → je seznam, nebo prázdný seznam

Operátor řezu !

- je to operátor který uspěje (je splněný) pouze jednou – při backtrackingu už ne a tedy se přes něj nelze vracet a zkoušet unifikovat podcíle před ním znovu a najít jinak
- backtracking platí jen po něj!

zelený řez – nemění logiku, pouze zrychluje

```
max(X,Y,X) :- X>Y, !.
max(X,Y,Y) :- Y>=X.
```

→ když X>Y tak už nebude zbytečně zkoušet to druhé

červený řez – nutný aby program fungoval správně

```
max2(X,Y,X) :- X>Y, !.
max2(X,Y,Y).
```

– pokud první platí, tak se nesmí na druhý, to zkusit nadto

Správné použití operátoru řezu

- `not(A):-A,!,fail.`
  `not(A).`

  – unifikuje se A a pokud je False tak hned první podcíl selže a selže tedy i celá první klausule a jde se unifikovat druhá a ta je vždy True
  – pokud A je True tak první podcíl uspěje a pak uspěje i operátor řezu ale False ne neuspěje a tak se zabacktrackovat ale to se nepodaří (Fail) tak se ani na tu druhou klausuli nejde

- `max(X,Y,X):-X>Y,!.`
  `max(X,Y,Y).`

  – zde bez operátoru řezu by nám to vracelo jak X tak i Y pokud by bylo větší X a pokud Y tak správně
  – pomocí operátoru řezu zamezí aby se šlo na druhý řádek (druhou klausuli) pokud je splněno X>Y protože druhá klausule je splněna vždy když se na ni přijde

Špatné použití operátoru řezu

- `matka(Jana, Jan).`
  `otec(Petr, Jan).`
  `rodic(X,Y):-matka(X,Y),!.`
  `rodic(X,Y):-otec(X,Y).`

  → relace musí mít malá písmena jinak by se Jan bral jako proměnná!
  → je to špatné použití protože při dotazu rodic(X, Jan) to vrátí jen matku a ne otce pokud má oba

→ projít si celou tu jednu stránku SDL-rezoluce v materiálech od Patrika !!!