Curs 7

Cuprins

Bibliografie:

- Logic Programming, The University of Edinburgh
  https://www.inf.ed.ac.uk/teaching/courses/lp/
- J.W.Lloyd, Foundations of Logic Programming, 1987

Logica de ordinul I - sintaxa

Limbaj de ordinul I $\mathcal{L}$

- unic determinat de $\tau = (\mathbf{R}, \mathbf{F}, \mathbf{C}, ari)$

Termenii lui $\mathcal{L}$, notați $Trm_{\mathcal{L}}$, sunt definiți inductiv astfel:

- orice variabilă este un termen;
- orice simbol de constantă este un termen;
- dacă $f \in \mathbf{F}$, $ar(f) = n$ și $t_1, \ldots, t_n$ sunt termeni, atunci $f(t_1, \ldots, t_n)$ este termen.

Formulele atomice ale lui $\mathcal{L}$ sunt definite astfel:

- dacă $R \in \mathbf{R}$, $ar(R) = n$ și $t_1, \ldots, t_n$ sunt termeni, atunci $R(t_1, \ldots, t_n)$ este formulă atomică.

Formulele lui $\mathcal{L}$ sunt definite astfel:

- orice formulă atomică este o formulă
- dacă φ este o formulă, atunci $\neg\varphi$ este o formulă
- dacă φ și ψ sunt formule, atunci $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$ sunt formule
- dacă φ este o formulă și x este o variabilă, atunci $\forall x\, \varphi$, $\exists x\, \varphi$ sunt formule

Logica de ordinul I - semantică (opțional)

O structură este de forma $\mathcal{A} = (A, \mathbf{F}^{\mathcal{A}}, \mathbf{R}^{\mathcal{A}}, \mathbf{C}^{\mathcal{A}})$, unde

- A este o mulțime nevidă
- $\mathbf{F}^{\mathcal{A}} = \{f^{\mathcal{A}} \mid f \in \mathbf{F}\}$ este o mulțime de operații pe A; dacă f are aritatea n, atunci $f^{\mathcal{A}} : A^n \to A$.
- $\mathbf{R}^{\mathcal{A}} = \{R^{\mathcal{A}} \mid R \in \mathbf{R}\}$ este o mulțime de relații pe A; dacă R are aritatea n, atunci $R^{\mathcal{A}} \subseteq A^n$.
- $\mathbf{C}^{\mathcal{A}} = \{c^{\mathcal{A}} \in A \mid c \in \mathbf{C}\}$.

O interpretare a variabilelor lui $\mathcal{L}$ în $\mathcal{A}$ ($\mathcal{A}$-interpretare) este o funcție $I : V \to A$.

Inductiv, definim interpretarea termenului t în $\mathcal{A}$ sub I notat $t_I^{\mathcal{A}}$.

Inductiv, definim când o formulă este adevărată în $\mathcal{A}$ în interpretarea I notat $\mathcal{A}, I \models \varphi$. În acest caz spunem că $(\mathcal{A}, I)$ este model pentru φ.

O formulă φ este adevărată într-o structură $\mathcal{A}$, notat $\mathcal{A} \models \varphi$, dacă este adevărată în $\mathcal{A}$ sub orice interpretare. Spunem că $\mathcal{A}$ este model al lui φ.

O formulă φ este adevărată în logica de ordinul I, notat $\models \varphi$, dacă este adevărată în orice structură. O formulă φ este validă dacă $\models \varphi$.

O formulă φ este satisfiabilă dacă există o structură $\mathcal{A}$ și o $\mathcal{A}$-interpretare I astfel încât $\mathcal{A}, I \models \varphi$.

Deducție și satisfiabilitate

Fie $\varphi_1, \ldots, \varphi_n, \varphi$ formule în logica propozițională
(enunțuri în calculul cu predicate).

$\{\varphi_1, \ldots, \varphi_n\} \models \varphi$ este echivalent cu

Deducție și satisfiabilitate

Fie $\varphi_1, \ldots, \varphi_n, \varphi$ formule în logica propozițională
(enunțuri în calculul cu predicate).

$\{\varphi_1, \ldots, \varphi_n\} \models \varphi$ este echivalent cu

$\models \varphi_1 \wedge \ldots \wedge \varphi_n \rightarrow \varphi$ este echivalent cu

Deducție și satisfiabilitate

Fie $\varphi_1, \ldots, \varphi_n, \varphi$ formule în logica propozițională (enunțuri în calculul cu predicate).

$\{\varphi_1, \ldots, \varphi_n\} \models \varphi$ este echivalent cu

$\models \varphi_1 \wedge \ldots \wedge \varphi_n \to \varphi$ este echivalent cu

$\models \neg\varphi_1 \vee \ldots \neg\varphi_n \vee \varphi$ este echivalent cu

Deducție și satisfiabilitate

Fie $\varphi_1, \ldots, \varphi_n, \varphi$ formule în logica propozițională (enunțuri în calculul cu predicate).

$\{\varphi_1, \ldots, \varphi_n\} \models \varphi$ este echivalent cu

$\models \varphi_1 \wedge \ldots \wedge \varphi_n \to \varphi$ este echivalent cu

$\models \neg\varphi_1 \vee \ldots \neg\varphi_n \vee \varphi$ este echivalent cu

$\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi$ este satisfiabilă

Literali. Clauze

Literali

- În calculul propozițional un literal este o variabilă sau negația unei variabile.

$$\textit{literal} := \; p \mid \neg p \qquad \text{unde } p \text{ este variabilă propozițională}$$

Literali

- În calculul propozițional un literal este o variabilă sau negația unei variabile.

$$literal := \ p \mid \neg p \qquad \text{unde } p \text{ este variabilă propozițională}$$

- În logica de ordinul I un literal este o formulă atomică sau negația unei formule atomice.

$$literal := \ P(t_1, \ldots, t_n) \mid \neg P(t_1, \ldots, t_n)$$

unde $P \in \mathbf{R}, ari(P) = n$, și $t_1, \ldots, t_n$ sunt termeni.

Literali

- În calculul propozițional un literal este o variabilă sau negația unei variabile.

$$literal := \ p \mid \neg p \qquad \text{unde } p \text{ este variabilă propozițională}$$

- În logica de ordinul I un literal este o formulă atomică sau negația unei formule atomice.

$$literal := \ P(t_1, \ldots, t_n) \mid \neg P(t_1, \ldots, t_n)$$

unde $P \in \mathbf{R}, ari(P) = n$, și $t_1, \ldots, t_n$ sunt termeni.

Clauze

- O clauză este o disjuncție de literali.

Clauze

- O clauză este o disjuncție de literali.
- Dacă $L_1, \ldots, L_n$ sunt literali atunci clauza $L_1 \vee \ldots \vee L_n$ o vom scrie ca mulțimea $\{L_1, \ldots, L_n\}$

clauză = mulțime de literali

Clauze

- O clauză este o disjuncție de literali.
- Dacă $L_1, \ldots, L_n$ sunt literali atunci clauza $L_1 \vee \ldots \vee L_n$ o vom scrie ca mulțimea $\{L_1, \ldots, L_n\}$

  clauză $=$ mulțime de literali

- Clauza $C = \{L_1, \ldots, L_n\}$ este satisfiabilă dacă $L_1 \vee \ldots \vee L_n$ este satisfiabilă.

Clauze

- O clauză este o disjuncție de literali.
- Dacă $L_1, \ldots, L_n$ sunt literali atunci clauza $L_1 \vee \ldots \vee L_n$ o vom scrie ca mulțimea $\{L_1, \ldots, L_n\}$

  clauză = mulțime de literali

- Clauza $C = \{L_1, \ldots, L_n\}$ este satisfiabilă dacă $L_1 \vee \ldots \vee L_n$ este satisfiabilă.
- O clauză C este trivială dacă conține un literal și complementul lui.

Clauze

- O clauză este o disjuncție de literali.
- Dacă $L_1, \ldots, L_n$ sunt literali atunci clauza $L_1 \vee \ldots \vee L_n$ o vom scrie ca mulțimea $\{L_1, \ldots, L_n\}$

  clauză = mulțime de literali

- Clauza $C = \{L_1, \ldots, L_n\}$ este satisfiabilă dacă $L_1 \vee \ldots \vee L_n$ este satisfiabilă.
- O clauză C este trivială dacă conține un literal și complementul lui.
- Când $n = 0$ obținem clauza vidă, care se notează $\square$

Clauze

- O clauză este o disjuncție de literali.
- Dacă $L_1, \ldots, L_n$ sunt literali atunci clauza $L_1 \vee \ldots \vee L_n$ o vom scrie ca mulțimea $\{L_1, \ldots, L_n\}$

  clauză = mulțime de literali

- Clauza $C = \{L_1, \ldots, L_n\}$ este satisfiabilă dacă $L_1 \vee \ldots \vee L_n$ este satisfiabilă.
- O clauză C este trivială dacă conține un literal și complementul lui.
- Când $n = 0$ obținem clauza vidă, care se notează $\square$
- Prin definiție, clauza $\square$ nu este satisfiabilă.

Clauze

- O clauză este o disjuncție de literali.
- Dacă $L_1, \ldots, L_n$ sunt literali atunci clauza $L_1 \vee \ldots \vee L_n$ o vom scrie ca mulțimea $\{L_1, \ldots, L_n\}$

  clauză = mulțime de literali

- Clauza $C = \{L_1, \ldots, L_n\}$ este satisfiabilă dacă $L_1 \vee \ldots \vee L_n$ este satisfiabilă.
- O clauză C este trivială dacă conține un literal și complementul lui.
- Când $n = 0$ obținem clauza vidă, care se notează $\square$
- Prin definiție, clauza $\square$ nu este satisfiabilă.

Rezoluția este o metodă de verificare a satisfiabilității unei mulțimi de clauze.

Logica Horn

Clauze în logica de ordinul I

$$\{\neg Q_1, \ldots, \neg Q_n, P_1, \ldots, P_k\}$$

unde $n, k \geq 0$ și $Q_1, \ldots, Q_n, P_1, \ldots, P_k$ sunt formule atomice.

- formula corespunzătoare este

$$\forall x_1 \ldots \forall x_m (\neg Q_1 \vee \ldots \vee \neg Q_n \vee P_1 \vee \ldots \vee P_k)$$

unde $x_1, \ldots, x_m$ sunt toate variabilele care apar în clauză

- echivalent, putem scrie

$$\forall x_1 \ldots \forall x_m (Q_1 \wedge \ldots \wedge Q_n \to P_1 \vee \ldots \vee P_k)$$

- cuantificarea universală a clauzelor este implicită

$$Q_1 \wedge \ldots \wedge Q_n \to P_1 \vee \ldots \vee P_k$$

Clauze definite. Programe logice. Clauze Horn

- clauză:

$$\{\neg Q_1, \ldots, \neg Q_n, P_1, \ldots, P_k\} \quad \text{sau} \quad Q_1 \land \ldots \land Q_n \to P_1 \lor \ldots \lor P_k$$

  unde $n, k \geq 0$ și $Q_1, \ldots, Q_n, P_1, \ldots, P_k$ sunt formule atomice.

- clauză program definită: $k = 1$
  - cazul $n > 0$: $Q_1 \land \ldots \land Q_n \to P$
  - cazul $n = 0$: $\top \to P$ (clauză unitate, fapt)

  Program logic definit = mulțime finită de clauze definite

- scop definit (țintă, întrebare): k=0
  - $Q_1 \land \ldots \land Q_n \to \bot$

- clauza vidă $\square$: $n = k = 0$

Clauze definite. Programe logice. Clauze Horn

- clauză:

  $$\{\neg Q_1, \ldots, \neg Q_n, P_1, \ldots, P_k\} \quad \text{sau} \quad Q_1 \wedge \ldots \wedge Q_n \to P_1 \vee \ldots \vee P_k$$

  unde $n, k \geq 0$ și $Q_1, \ldots, Q_n, P_1, \ldots, P_k$ sunt formule atomice.

- clauză program definită: $k = 1$
  - cazul $n > 0$: $Q_1 \wedge \ldots \wedge Q_n \to P$
  - cazul $n = 0$: $\top \to P$ (clauză unitate, fapt)

  Program logic definit = mulțime finită de clauze definite

- scop definit (țintă, întrebare): k=0
  - $Q_1 \wedge \ldots \wedge Q_n \to \bot$

- clauza vidă $\square$: $n = k = 0$

Clauza Horn = clauză program definită sau clauză scop ($k \leq 1$)

Clauze Horn țintă

- scop definit (țintă, întrebare): $Q_1 \wedge \ldots \wedge Q_n \to \bot$
  - fie $x_1, \ldots, x_m$ toate variabilele care apar în $Q_1, \ldots, Q_n$

$$\forall x_1 \ldots \forall x_m(\neg Q_1 \vee \ldots \vee \neg Q_n) \mathrel{\text{⫤⊨}} \neg \exists x_1 \ldots \exists x_m(Q_1 \wedge \ldots \wedge Q_n)$$

  - clauza țintă o vom scrie $Q_1, \ldots, Q_n$

Negația unei "întrebări" în PROLOG este clauză Horn țintă.

Programare logica

- Logica clauzelor definite/Logica Horn: un fragment al logicii de ordinul I în care singurele formule admise sunt clauze Horn
  - formule atomice: $P(t_1, \ldots, t_n)$
  - $Q_1 \wedge \ldots \wedge Q_n \to P$
    unde toate Q_i, P sunt formule atomice, $\top$ sau $\bot$

- Problema programării logice: reprezentăm cunoștințele ca o mulțime de clauze definite KB și suntem interesați să aflăm răspunsul la o întrebare de forma $Q_1 \wedge \ldots \wedge Q_n$, unde toate Q_i sunt formule atomice

$$KB \models Q_1 \wedge \ldots \wedge Q_n$$

  - Variabilele din KB sunt cuantificate universal.
  - Variabilele din $Q_1, \ldots, Q_n$ sunt cuantificate existențial.

Limbajul PROLOG are la bază logica clauzelor Horn.

Logica clauzelor definite

Exemplu

Fie următoarele clauze definite:

$$\begin{array}{l} father(jon, ken). \\ father(ken, liz). \\ father(X, Y) \to ancestor(X, Y) \\ daugther(X, Y) \to ancestor(Y, X) \\ ancestor(X, Y) \land ancestor(Y, Z) \to ancestor(X, Z) \end{array}$$

Putem întreba:

- $ancestor(jon, liz)$
- dacă există Q astfel încât $ancestor(Q, ken)$ (adică $\exists Q\, ancestor(Q, ken)$)

Sistem de deducție pentru logica Horn

Sistem de deducție *backchain*

Sistem de deducție pentru clauze Horn

Pentru un program logic definit KB avem

Sistem de deducție *backchain*

Sistem de deducție pentru clauze Horn

Pentru un program logic definit KB avem

- Axiome: orice clauză din KB

Sistem de deducție *backchain*

Sistem de deducție pentru clauze Horn

Pentru un program logic definit KB avem

- Axiome: orice clauză din KB
- Regula de deducție: regula *backchain*

$$\frac{\theta(Q_1) \quad \theta(Q_2) \quad \ldots \quad \theta(Q_n) \quad (Q_1 \land Q_2 \land \ldots \land Q_n \to P)}{\theta(Q)}$$

unde $Q_1 \land Q_2 \land \ldots \land Q_n \to P \in KB$, iar θ este cgu pentru Q și P.

Sistem de deducție

Exemplu

KB conține următoarele clauze definite:

$father(jon, ken).$
$father(ken, liz).$
$father(X, Y) \rightarrow ancestor(X, Y)$
$daugther(X, Y) \rightarrow ancestor(Y, X)$
$ancestor(X, Y) \wedge ancestor(Y, Z) \rightarrow ancestor(X, Z)$

$$\frac{\theta(Q_1) \quad \theta(Q_2) \quad \ldots \quad \theta(Q_n) \quad (Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \rightarrow P)}{\theta(Q)}$$

unde $Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \rightarrow P \in KB$, iar θ este cgu pentru Q și P

Sistem de deducție

Pentru o țintă Q, trebuie să găsim o clauză din KB

$$Q_1 \land \ldots \land Q_n \to P,$$

și un unificator θ pentru Q și P. În continuare vom verifica $\theta(Q_1), \ldots, \theta(Q_n)$.

Sistem de deducție

Pentru o țintă Q, trebuie să găsim o clauză din KB

$$Q_1 \wedge \ldots \wedge Q_n \to P,$$

și un unificator θ pentru Q și P. În continuare vom verifica $\theta(Q_1), \ldots, \theta(Q_n)$.

Exemplu

Pentru ținta

$$ancestor(ken, Z),$$

Sistem de deducție

Pentru o țintă Q, trebuie să găsim o clauză din KB

$$Q_1 \wedge \ldots \wedge Q_n \to P,$$

și un unificator θ pentru Q și P. În continuare vom verifica $\theta(Q_1), \ldots, \theta(Q_n)$.

Exemplu

Pentru ținta

$$ancestor(ken, Z),$$

putem folosi o clauză

$$father(Y, X) \to ancestor(Y, X)$$

cu unificatorul

$$\{Y/ken, X/Z\}$$

pentru a obține o nouă țintă

$$father(ken, Z).$$

Sistem de deducție

$$\frac{\theta(Q_1) \quad \theta(Q_2) \quad \ldots \quad \theta(Q_n) \quad (Q_1 \land Q_2 \land \ldots \land Q_n \to P)}{\theta(Q)}$$

unde $Q_1 \land Q_2 \land \ldots \land Q_n \to P \in KB$, iar θ este cgu pentru Q și P.

Exemplu

$$\frac{\dfrac{father(ken, liz)}{father(ken, Z)} \quad (father(Y, X) \to ancestor(Y, X))}{ancestor(ken, Z)}$$

Puncte de decizie în programarea logica

Având doar această regulă, care sunt punctele de decizie în căutare?

Puncte de decizie în programarea logica

Având doar această regulă, care sunt punctele de decizie în căutare?

- Ce clauză să alegem.
  - Pot fi mai multe clauze a căror parte dreaptă se potrivește cu o țintă.
  - Aceasta este o alegere de tip **SAU**: este suficient ca oricare din variante să reușească.

Puncte de decizie în programarea logica

Având doar această regulă, care sunt punctele de decizie în căutare?

- Ce clauză să alegem.
  - Pot fi mai multe clauze a căror parte dreaptă se potrivește cu o țintă.
  - Aceasta este o alegere de tip **SAU**: este suficient ca oricare din variante să reușească.
- Ordinea în care rezolvăm noile ținte.
  - Aceasta este o alegere de tip **ȘI:** trebuie arătate toate țintele noi.
  - Ordinea în care le rezolvăm poate afecta găsirea unei derivări, depinzând de strategia de căutare folosită.

Strategia de căutare din Prolog

- Regula *backchain* conduce la un sistem de deducție complet:

  Pentru o mulțime de clauze KB și o țintă Q,
  dacă $KB \models Q$,
  atunci există o derivare a lui Q folosind regula *backchain*.

Strategia de căutare din Prolog

- Regula *backchain* conduce la un sistem de deducție complet:

  Pentru o mulțime de clauze KB și o țintă Q,
  dacă $KB \models Q$,
  atunci există o derivare a lui Q folosind regula *backchain*.

- Strategia de căutare din Prolog este de tip *depth-first*,
  - de sus în jos
    - pentru alegerile de tip **SAU**
    - alege clauzele în ordinea în care apar în program
  - de la stânga la dreapta
    - pentru alegerile de tip **ȘI**
    - alege noile ținte în ordinea în care apar în clauza aleasă

Sistemul de inferență backchain

Notăm cu $KB \vdash_b Q$ dacă există o derivare a lui Q din KB folosind sistemul de inferență *backchain*.

Teoremă

Sistemul de inferență backchain *este corect și complet pentru formule atomice fără variabile Q.*

$$KB \models Q \quad \text{dacă și numai dacă} \quad KB \vdash_b Q$$

Sistemul de inferență backchain

Notăm cu $KB \vdash_b Q$ dacă există o derivare a lui Q din KB folosind sistemul de inferență *backchain*.

Teoremă

Sistemul de inferență backchain *este corect și complet pentru formule atomice fără variabile Q.*

$$KB \models Q \quad \textit{dacă și numai dacă} \quad KB \vdash_b Q$$

Sistemul de inferență *backchain* este corect și complet și pentru formule atomice cu variabile Q:

$$KB \models \exists x Q(x) \text{ dacă și numai dacă } KB \vdash_b \theta(Q) \text{ pentru o substituție } \theta.$$

Rezoluție SLD

Regula *backchain* și rezoluția SLD

- Regula *backchain* este implementată în programarea logică prin rezoluția SLD (Selected, Linear, Definite).
- Prolog are la bază rezoluția SLD.

Rezoluția SLD

Fie KB o mulțime de clauze definite.

$$\text{SLD}\quad \boxed{\dfrac{\neg Q_1 \vee \cdots \vee \neg Q_i \vee \cdots \vee \neg Q_n}{\theta(\neg Q_1 \vee \cdots \vee \neg P_1 \vee \cdots \vee \neg P_m \vee \cdots \vee \neg Q_n)}}$$

unde

- $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ este o clauză definită din KB (în care toate variabilele au fost redenumite) și
- variabilele din $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ și Q_i se redenumesc
- θ este c.g.u pentru Q_i și Q

Rezoluția SLD

Exemplu

```
father(eddard,sansa).
father(eddard,jonSnow).

stark(eddard).
stark(catelyn).

stark(X) :- father(Y,X),
stark(Y).
```

```
?- stark(jonSnow)
```

SLD $$\boxed{\frac{\neg Q_1 \vee \cdots \vee \neg Q_i \vee \cdots \vee \neg Q_n}{\theta(\neg Q_1 \vee \cdots \vee \neg P_1 \vee \cdots \vee \neg P_m \vee \cdots \vee \neg Q_n)}}$$

- $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ este o clauză definită din KB
- variabilele din $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ și Q_i se redenumesc
- θ este c.g.u pentru Q_i și Q.

Rezoluția SLD

Exemplu

$father(eddard, sansa)$
$father(eddard, jonSnow)$

$stark(eddard)$
$stark(catelyn)$

$stark(X) \vee \neg father(Y, X) \vee \neg stark(Y)$

$$\frac{\neg stark(jonSnow)}{}$$

$$\theta(X) = jonSnow$$

$$\text{SLD} \quad \boxed{\frac{\neg Q_1 \vee \cdots \vee \neg Q_i \vee \cdots \vee \neg Q_n}{\theta(\neg Q_1 \vee \cdots \vee \neg P_1 \vee \cdots \vee \neg P_m \vee \cdots \vee \neg Q_n)}}$$

- $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ este o clauză definită din KB
- variabilele din $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ și Q_i se redenumesc
- θ este c.g.u pentru Q_i și Q.

Rezoluția SLD

Exemplu

$father(eddard, sansa)$
$father(eddard, jonSnow)$

$stark(eddard)$
$stark(catelyn)$

$stark(X) \vee \neg father(Y, X) \vee \neg stark(Y)$

$$\frac{\neg stark(jonSnow)}{\neg father(Y, jonSnow) \vee \neg stark(Y)}$$

$$\theta(X) = jonSnow$$

$$\text{SLD} \quad \boxed{\frac{\neg Q_1 \vee \cdots \vee \neg Q_i \vee \cdots \vee \neg Q_n}{\theta(\neg Q_1 \vee \cdots \vee \neg P_1 \vee \cdots \vee \neg P_m \vee \cdots \vee \neg Q_n)}}$$

- $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ este o clauză definită din KB
- variabilele din $Q \vee \neg P_1 \vee \cdots \vee \neg P_m$ și Q_i se redenumesc
- θ este c.g.u pentru Q_i și Q.

Rezoluția SLD

Exemplu

$father(eddard, sansa)$
$father(eddard, jonSnow)$
$stark(eddard)$
$stark(catelyn)$
$stark(X) \vee \neg father(Y, X) \vee \neg stark(Y)$

$$\frac{\neg stark(jonSnow)}{\neg father(Y, jonSnow) \vee \neg stark(Y)}$$

Rezoluția SLD

Exemplu

$father(eddard, sansa)$
$father(eddard, jonSnow)$
$stark(eddard)$
$stark(catelyn)$
$stark(X) \vee \neg father(Y, X) \vee \neg stark(Y)$

$$\frac{\neg stark(jonSnow)}{\neg father(Y, jonSnow) \vee \neg stark(Y)}$$

$$\frac{\neg father(Y, jonSnow) \vee \neg stark(Y)}{\neg stark(eddard)}$$

Rezoluția SLD

Exemplu

$father(eddard, sansa)$
$father(eddard, jonSnow)$
$stark(eddard)$
$stark(catelyn)$
$stark(X) \vee \neg father(Y, X) \vee \neg stark(Y)$

$$\frac{\neg stark(jonSnow)}{\neg father(Y, jonSnow) \vee \neg stark(Y)}$$

$$\frac{\neg father(Y, jonSnow) \vee \neg stark(Y)}{\neg stark(eddard)}$$

$$\frac{\neg stark(eddard)}{\square}$$

Rezoluția SLD

Fie KB o mulțime de clauze definite și $Q_1 \land \ldots \land Q_m$ o întrebare, unde Q_i sunt formule atomice.

- O derivare din KB prin rezoluție SLD este o secvență

$$G_0 := \neg Q_1 \lor \ldots \lor \neg Q_m, \quad G_1, \quad \ldots, \quad G_k, \ldots$$

  în care G_{i+1} se obține din G_i prin regula SLD.
- Dacă există un k cu $G_k = \square$ (clauza vidă), atunci derivarea se numește SLD-respingere.

Rezoluția SLD

Teoremă (Completitudinea SLD-rezoluției)

Sunt echivalente:

- *există o SLD-respingere a lui* $Q_1 \land \ldots \land Q_m$ *din* KB,
- $KB \vdash_b Q_1 \land \ldots \land Q_m$,
- $KB \models Q_1 \land \cdots \land Q_m$.

Rezoluția SLD

Teoremă (Completitudinea SLD-rezoluției)

Sunt echivalente:

- *există o SLD-respingere a lui $Q_1 \wedge \ldots \wedge Q_m$ din KB,*
- $KB \vdash_b Q_1 \wedge \ldots \wedge Q_m$,
- $KB \models Q_1 \wedge \cdots \wedge Q_m$.

Demonstrație

Rezultă din completitudinea sistemului de deducție backchain și din faptul că:

există o SLD-respingere a lui $Q_1 \wedge \ldots \wedge Q_m$ din KB

ddacă

$$KB \vdash_b Q_1 \wedge \ldots \wedge Q_m$$

$\square$

Rezoluția SLD - arbori de căutare

Arbori SLD

- Presupunem că avem o mulțime de clauze definite KB și o țintă $G_0 = \neg Q_1 \vee \ldots \vee \neg Q_m$

- Construim un arbore de căutare (arbore SLD) astfel:
  - Fiecare nod al arborelui este o țintă (posibil vidă)
  - Rădăcina este G_0
  - Dacă arborele are un nod G_i, iar G_{i+1} se obține din G_i folosind regula SLD folosind o clauză $C_i \in KB$, atunci nodul G_i are copilul G_{i+1}. Muchia dintre G_i și G_{i+1} este etichetată cu C_i.

- Dacă un arbore SLD cu rădăcina G_0 are o frunză $\square$ (clauza vidă), atunci există o SLD-respingere a lui G_0 din KB.

Rezoluția SLD

Exemplu

- Fie KB următoarea mulțime de clauze definite:
  1. $grandfather(X, Z) :- father(X, Y), parent(Y, Z)$
  2. $parent(X, Y) :- father(X, Y)$
  3. $parent(X, Y) :- mother(X, Y)$
  4. $father(ken, diana)$
  5. $mother(diana, brian)$

- Găsiți o respingere din KB pentru

$$:- grandfather(ken, Y)$$

Rezoluția SLD

Exemplu

- Fie KB următoarea mulțime de clauze definite:
  1. $grandfather(X, Z) \vee \neg father(X, Y) \vee \neg parent(Y, Z)$
  2. $parent(X, Y) \vee \neg father(X, Y)$
  3. $parent(X, Y) \vee \neg mother(X, Y)$
  4. $father(ken, diana)$
  5. $mother(diana, brian)$

- Găsiți o respingere din KB pentru

$$\neg grandfather(ken, Y)$$

Rezoluția SLD

Exemplu

1. $grandfather(X, Z) \vee \neg father(X, Y) \vee \neg parent(Y, Z)$
2. $parent(X, Y) \vee \neg father(X, Y)$
3. $parent(X, Y) \vee \neg mother(X, Y)$
4. $father(ken, diana)$
5. $mother(diana, brian)$

$$\neg grandfather(ken, Y)$$

$$\downarrow 1$$

$$\neg father(ken, V) \vee \neg parent(V, Y)$$

$$\downarrow 4$$

$$\neg parent(diana, Y)$$

$$\swarrow 2 \qquad \searrow 3$$

$$\neg father(diana, Y) \qquad \neg mother(diana, Y)$$

$$\downarrow 5$$

$$\square$$

Rezoluția SLD

Exemplu

2. $parent(X, Y) \vee \neg father(X, Y)$

$$\begin{array}{c} \qquad\qquad \neg parent(diana, Y) \\ \swarrow_{2} \\ \neg father(diana, Y) \end{array}$$

Aplicarea SLD:

Rezoluția SLD

Exemplu

2. $parent(X, Y) \vee \neg father(X, Y)$

$$\neg parent(diana, Y) \xrightarrow{2} \neg father(diana, Y)$$

Aplicarea SLD:

- redenumesc variabilele: $parent(X, Y_2) \vee \neg father(X, Y_2)$
- determin unificatorul: $\theta = X/diana, Y_2/Y$
- aplic regula: $\dfrac{\neg parent(diana, Y)}{\neg father(diana, Y)}$

Rezoluția SLD - arbori de căutare

Exercițiu

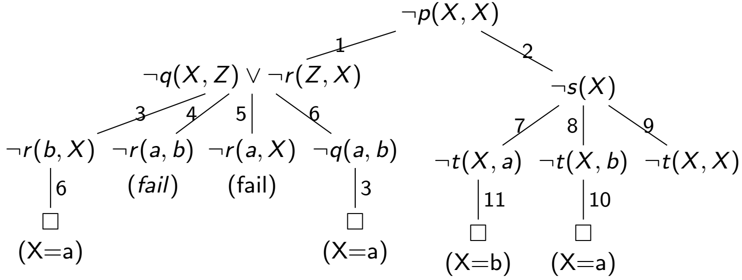
Desenați arborele SLD pentru programul Prolog de mai jos și ținta
`?- p(X,X).`

```
1.  p(X,Y) :- q(X,Z), r(Z,Y).
2.  p(X,X) :- s(X).
3.  q(X,b).
4.  q(b,a).
5.  q(X,a) :- r(a,X).
6.  r(b,a).
7.  s(X) :- t(X,a).
8.  s(X) :- t(X,b).
9.  s(X) :- t(X,X).
10.  t(a,b).
11.  t(b,a).
```

Rezoluția SLD - arbori de căutare

```
1.  p(X,Y) :- q(X,Z), r(Z,Y).
2.  p(X,X) :- s(X).
3.  q(X,b).
4.  q(b,a).
5.  q(X,a) :- r(a,X).
6.  r(b,a).
7.  s(X) :- t(X,a).
8.  s(X) :- t(X,b).
9.  s(X) :- t(X,X).
10.  t(a,b).
11.  t(b,a).
```

$p(X, Y) \vee \neg q(X, Z) \vee \neg r(Z, Y)$
$p(X, X) \vee \neg s(X)$
$q(X, b)$
$q(b, a)$
$q(X, a) \vee \neg r(a, X)$
$r(b, a)$
$s(X) \vee \neg t(X, a)$
$s(X) \vee \neg t(X, b)$
$s(X) \vee \neg t(X, X)$
$t(a, b)$
$t(b, a)$



Limbajul Prolog

- Am arătat că sistemul de inferență din spatele Prolog-ului este complet.
  - Dacă o întrebare este consecință logică a unei mulțimi de clauze, atunci există o derivare a întrebării.
- Totuși, strategia de căutate din Prolog este incompletă!
  - Chiar dacă o întrebare este consecință logică a unei mulțimi de clauze, Prolog nu găsește mereu o derivare a întrebării.

Exemplu

```
warmerClimate :- albedoDecrease.
warmerClimate :- carbonIncrease.
iceMelts :- warmerClimate.
albedoDecrease :- iceMelts.
carbonIncrease.

?- iceMelts.
! Out of local stack
```

Limbajul Prolog

Exemplu

```
warmerClimate :- albedoDecrease.
warmerClimate :- carbonIncrease.
iceMelts :- warmerClimate.
albedoDecrease :- iceMelts.
carbonIncrease.

?- iceMelts.
! Out of local stack
```

Limbajul Prolog

Exemplu (cont.)

Există o derivare a lui *iceMelts* în sistemul de deducție din clauzele:

$$
\begin{array}{rcl}
\mathit{albedoDecrease} & \rightarrow & \mathit{warmerClimate}\\
\mathit{carbonIncrease} & \rightarrow & \mathit{warmerClimate}\\
\mathit{warmerClimate} & \rightarrow & \mathit{iceMelts}\\
\mathit{iceMelts} & \rightarrow & \mathit{albedoDecrease}\\
\top & \rightarrow & \mathit{carbonIncrease}
\end{array}
$$

$$
\dfrac{\dfrac{\mathit{carbonInc}. \qquad \mathit{carbonInc}. \rightarrow \mathit{warmerClim}.}{\mathit{warmerClim}.} \qquad \mathit{warmerClim}. \rightarrow \mathit{iceMelts}}{\mathit{iceMelts}}
$$

Pe săptămâna viitoare!