

Curs 9

- 1 Logica Horn
- 2 Sistem de deducție pentru logica Horn
- 3 Rezoluție SLD

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 Horn

Clauze definite. Programe logice. Clauze Horn

□ clauză:

$$\{\neg Q_1, \dots, \neg Q_n, P_1, \dots, P_k\} \quad \text{sau} \quad Q_1 \wedge \dots \wedge Q_n \rightarrow P_1 \vee \dots \vee P_k$$

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

□ clauză program definită: $k = 1$

□ cazul $n > 0$: $Q_1 \wedge \dots \wedge Q_n \rightarrow P$

□ cazul $n = 0$: $\top \rightarrow P$ (clauză unitate, fapt)

Program logic definit = mulțime finită de clauze definite

□ scop definit (țintă, întrebare): $k=0$

□ $Q_1 \wedge \dots \wedge Q_n \rightarrow \perp$

□ clauza vidă □: $n = k = 0$

Clauze definite. Programe logice. Clauze Horn

□ clauză:

$$\{\neg Q_1, \dots, \neg Q_n, P_1, \dots, P_k\} \quad \text{sau} \quad Q_1 \wedge \dots \wedge Q_n \rightarrow P_1 \vee \dots \vee P_k$$

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

□ clauză program definită: $k = 1$

□ cazul $n > 0$: $Q_1 \wedge \dots \wedge Q_n \rightarrow P$

□ cazul $n = 0$: $\top \rightarrow P$ (clauză unitate, fapt)

Program logic definit = mulțime finită de clauze definite

□ scop definit (țintă, întrebare): $k=0$

□ $Q_1 \wedge \dots \wedge Q_n \rightarrow \perp$

□ clauza vidă □: $n = k = 0$

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

Programare logica

- Logica clauzelor definite/Logica Horn: un fragment al logicii de ordinul I în care singurele formule admise sunt clauze Horn
 - formule atomice: $P(t_1, \dots, t_n)$
 - $Q_1 \wedge \dots \wedge Q_n \rightarrow P$
unde toate Q_i, P sunt formule atomice, \top sau \perp
- Problema programării logice: reprezentăm cunoștințele ca o mulțime de clauze definite KB și suntem interesați să aflăm răspunsul la o întrebare de forma $Q_1 \wedge \dots \wedge Q_n$, unde toate Q_i sunt formule atomice
$$KB \models Q_1 \wedge \dots \wedge Q_n$$
 - Variabilele din KB sunt cuantificate universal.
 - Variabilele din Q_1, \dots, Q_n sunt cuantificate existențial.

Limbajul PROLOG are la bază logica clauzelor Horn.

Modele Herbrand

Definim o ordine între modelele Herbrand:

$\mathcal{H}_1 \leq \mathcal{H}_2$ este definită astfel:

*pentru orice $R \in \mathbf{R}$ cu $\text{ari}(R) = n$ și pentru orice termeni t_1, \dots, t_n
dacă $\mathcal{H}_1 \models R(t_1, \dots, t_n)$, atunci $\mathcal{H}_2 \models R(t_1, \dots, t_n)$*

Semantica unui program logic definit KB este dată de
cel mai mic model Herbrand al lui KB !

- Definim $\mathcal{LH}_{KB} := \bigcap \{ \mathcal{H} \mid \mathcal{H} \text{ model Herbrand pentru } KB \}$
- $\mathcal{LH}_{KB} \models KB$.
- Vom caracteriza cel mai mic model Herbrand \mathcal{LH}_{KB} printr-o construcție de punct fix.

Cel mai mic model Herbrand

- O **instanță de bază** a unei clauze $Q_1(x_1) \wedge \dots \wedge Q_n(x_n) \rightarrow P(y)$ este rezultatul obținut prin înlocuirea variabilelor cu termeni fără variabile.
- Pentru o mulțime de clauze definite KB , o formulă atomică P și o mulțime de formule atomice X ,

$oneStep_{KB}(P, X)$ este adevărat

dacă există o instanță de bază a unei clauze

$Q_1(x_1) \wedge \dots \wedge Q_n(x_n) \rightarrow P(y)$ din KB astfel încât P este instanța lui $P(y)$ și instanța lui $Q_i(x_i)$ este în X , pentru orice $i = 1, \dots, n$.

- **Baza Herbrand $B_{\mathcal{L}}$** este mulțimea formulelor atomice fără variabile.
- Pentru o mulțime de clauze definite KB , definim

$$f_{KB} : \mathcal{P}(B_{\mathcal{L}}) \rightarrow \mathcal{P}(B_{\mathcal{L}})$$

$$f_{KB}(X) = \{P \in B_{\mathcal{L}} \mid oneStep_{KB}(P, X)\}$$

Cel mai mic model Herbrand

Fie KB un program logic definit.

- f_{KB} este continuă
- Din teorema Knaster-Tarski, f_{KB} are un cel mai mic punct fix FP_{KB} .
- FP_{KB} este reuniunea tuturor mulțimilor

$$f_{KB}(\{\}), f_{KB}(f_{KB}(\{\})), f_{KB}(f_{KB}(f_{KB}(\{\}))), \dots$$

Propoziție (caracterizarea \mathcal{LH}_{KB})

Pentru orice $R \in \mathbf{R}$ cu $\text{ari}(R) = n$ și pentru orice t_1, \dots, t_n termeni, avem

$$(t_1, \dots, t_n) \in R^{\mathcal{LH}_T} \text{ ddacă } R(t_1, \dots, t_n) \in FP_{KB}$$

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*

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

unde $Q_1 \wedge Q_2 \wedge \dots \wedge Q_n \rightarrow P \in KB$, 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) → ancestor(X, Y)

daughter(X, Y) → ancestor(Y, X)

ancestor(X, Y) ∧ ancestor(Y, Z) → ancestor(X, Z)

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

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

Sistem de deducție

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

$$Q_1 \wedge \dots \wedge Q_n \rightarrow P,$$

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

Sistem de deducție

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

$$Q_1 \wedge \dots \wedge Q_n \rightarrow P,$$

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

Exemplu

Pentru ținta

ancestor(ken, Z),

Sistem de deducție

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

$$Q_1 \wedge \dots \wedge Q_n \rightarrow P,$$

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

Exemplu

Pentru ținta

$$\text{ancestor}(\text{ken}, Z),$$

putem folosi o clauză

$$\text{father}(Y, X) \rightarrow \text{ancestor}(Y, X)$$

cu unificatorul

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

pentru a obține o nouă țintă

$$\text{father}(\text{ken}, Z).$$

Sistem de deducție

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

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

Exemplu

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

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 .

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

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

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

Propoziție (Corectitudine)

Dacă $KB \vdash_b Q$, atunci $KB \models Q$.

Demonstrație [schiță]

- Presupunem că toate clauzele din KB sunt adevărate.
- Ne uităm, inductiv, la cazurile care pot să apară în derivarea lui Q .

□

Completitudine

Teoremă (Completitudine)

Dacă $KB \models Q$, atunci $KB \vdash_b Q$.

Completitudine

Teoremă (Completitudine)

Dacă $KB \models Q$, atunci $KB \vdash_b Q$.

Trebuie să arătăm că

pentru orice structură și orice interpretare,
dacă orice clauză din KB este adevărată, atunci și Q este adevărată,



există o derivare a lui Q din KB .

Completitudine

Teoremă (Completitudine)

Dacă $KB \models Q$, atunci $KB \vdash_b Q$.

Trebuie să arătăm că

pentru orice structură și orice interpretare,
dacă orice clauză din KB este adevărată, atunci și Q este adevărată,



există o derivare a lui Q din KB .

Demonstrația este mai simplă deoarece

este suficient să ne uităm la modelul Herbrand!

Cel mai mic model Herbrand

Teoremă

Pentru orice KB un program logic definit și Q o formulă atomică,

$$KB \vdash_B Q \quad \text{ddacă} \quad \mathcal{LH}_{KB} \models Q \quad \text{ddacă} \quad KB \models Q.$$

Cel mai mic model Herbrand

Teoremă

Pentru orice KB un program logic definit și Q o formulă atomică,

$$KB \vdash_B Q \quad \text{ddacă} \quad \mathcal{LH}_{KB} \models Q \quad \text{ddacă} \quad KB \models Q.$$

Demonstrație (schiță)

Demonstrăm numai prima echivalență.

- Implicația de la stânga la dreapta rezultă ușor din corectitudinea sistemului de inferență *backchain*.

Cel mai mic model Herbrand

Teoremă

Pentru orice KB un program logic definit și Q o formulă atomică,

$$KB \vdash_B Q \quad \text{dacă} \quad \mathcal{LH}_{KB} \models Q \quad \text{dacă} \quad KB \models Q.$$

Demonstrație (schiță)

Demonstrăm numai prima echivalență.

- Implicația de la stânga la dreapta rezultă ușor din corectitudinea sistemului de inferență *backchain*.
- Implicația de la dreapta la stânga este mai complicată.
 - Q apare în interpretările simbolurilor de predicate din \mathcal{LH}
 - Deci Q este obținut după un număr finit n de aplicări ale lui f_{KB}
 - Se arată prin inducție după n că pentru fiecare formulă care apare prin aplicări ale lui f_{KB} există o derivare în sistemul de inferență *backchain*.

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.

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

unde

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$ este o clauză definită din KB (în care toate variabilele au fost redenumite) și
- variabilele din $Q \vee \neg P_1 \vee \dots \vee \neg P_m$ 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(jonSnow)

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

SLD

$$\frac{\neg Q_1 \vee \dots \vee \neg Q_i \vee \dots \vee \neg Q_n}{\theta(\neg Q_1 \vee \dots \vee \neg P_1 \vee \dots \vee \neg P_m \vee \dots \vee \neg Q_n)}$$

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

Rezoluția SLD

Exemplu

father(eddard, sansa)

father(eddard, jonSnow)

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

stark(eddard)

stark(catelyn)

$\theta(X) = \text{jonSnow}$

$\text{stark}(X) \vee \neg \text{father}(Y, X) \vee \neg \text{stark}(Y)$

SLD

$$\frac{\neg Q_1 \vee \dots \vee \neg Q_i \vee \dots \vee \neg Q_n}{\theta(\neg Q_1 \vee \dots \vee \neg P_1 \vee \dots \vee \neg P_m \vee \dots \vee \neg Q_n)}$$

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$ este o clauză definită din KB
- variabilele din $Q \vee \neg P_1 \vee \dots \vee \neg P_m$ 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)

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

$$\theta(X) = jonSnow$$

SLD

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

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$ este o clauză definită din KB
- variabilele din $Q \vee \neg P_1 \vee \dots \vee \neg P_m$ 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)

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

Rezoluția SLD

Exemplu

father(eddard, sansa)

father(eddard, jonSnow)

stark(eddard)

stark(catelyn)

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

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

Rezoluția SLD

Exemplu

father(eddard, sansa)

father(eddard, jonSnow)

stark(eddard)

stark(catelyn)

stark(X) ∨ ¬father(Y, X) ∨ ¬stark(Y)

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

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

Rezoluția SLD

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

- O **derivare** din KB prin rezoluție SLD este o secvență

$$G_0 := \neg Q_1 \vee \dots \vee \neg Q_m, \quad G_1, \quad \dots, \quad G_k, \dots$$

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 $Q_1 \wedge \dots \wedge Q_m$ din KB ,
- $KB \vdash_b Q_1 \wedge \dots \wedge Q_m$,
- $KB \models Q_1 \wedge \dots \wedge Q_m$.

Rezoluția SLD

Teoremă (Completitudinea SLD-rezoluției)

Sunt echivalente:

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

Demonstrație

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

există o *SLD-respingere* a lui $Q_1 \wedge \dots \wedge Q_m$ din KB
ddacă
 $KB \vdash_b Q_1 \wedge \dots \wedge Q_m$

□

Rezoluția SLD - arbori de căutare

Arbori SLD

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

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

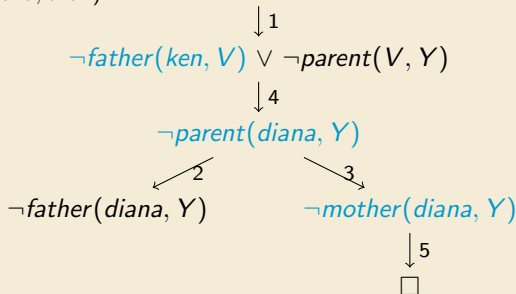
1 $grandfather(X, Z) \vee \neg father(X, Y) \vee \neg parent(Y, Z)$

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

3 $parent(X, Y) \vee \neg mother(X, Y)$

4 $father(ken, diana)$

5 $mother(diana, brian) \quad \neg grandfather(ken, Y)$




Rezoluția SLD

Exemplu

$$2 \quad \text{parent}(X, Y) \vee \neg \text{father}(X, Y)$$

$$\neg \text{parent}(\text{diana}, Y)$$

$$\neg \text{father}(\text{diana}, Y)$$



Aplicarea SLD:

Rezoluția SLD

Exemplu

2 $\text{parent}(X, Y) \vee \neg \text{father}(X, Y)$

$$\neg \text{parent}(\text{diana}, Y)$$

$$\neg \text{father}(\text{diana}, Y)$$


Aplicarea SLD:

□ redenumesc variabilele: $\text{parent}(X, Y_2) \vee \neg \text{father}(X, Y_2)$

□ determin unificatorul: $\theta = X/\text{diana}, Y_2/Y$

□ aplic regula:
$$\frac{\neg \text{parent}(\text{diana}, Y)}{\neg \text{father}(\text{diana}, Y)}$$

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

$p(X, Y) \vee \neg q(X, Z) \vee \neg r(Z, Y)$

$p(X, X) \vee \neg s(X)$

$q(X, b)$

$q(b, a)$

$q(X, a) \vee \neg r(a, X)$

$r(b, a)$

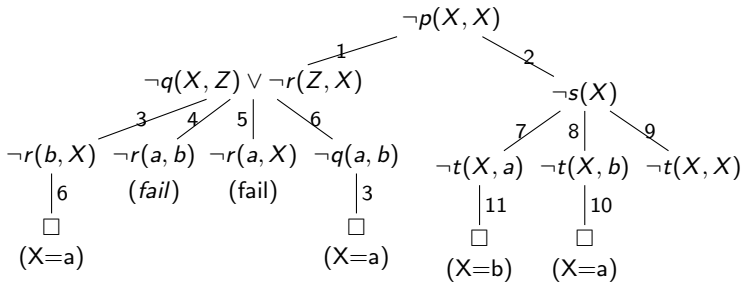
$s(X) \vee \neg t(X, a)$

$s(X) \vee \neg t(X, b)$

$s(X) \vee \neg t(X, X)$

$t(a, b)$

$t(b, a)$



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
```

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:

<i>albedoDecrease</i>	→	<i>warmerClimate</i>
<i>carbonIncrease</i>	→	<i>warmerClimate</i>
<i>warmerClimate</i>	→	<i>iceMelts</i>
<i>iceMelts</i>	→	<i>albedoDecrease</i>
⊤	→	<i>carbonIncrease</i>

<i>carbonInc.</i>	<i>carbonInc. → warmerClim.</i>	<i>warmerClim. → iceMelts</i>
<i>warmerClim.</i>		
<hr/>		
<i>iceMelts</i>		



Pe săptămâna viitoare!