Clauze Horn

Clauzele Horn sunt o submultime a FOL, ce este suficient de expresivă pentru multe cazuri și în care procedura de rezoluție funcționează bine.
În sistemele bazate pe rezoluție, clauzele sunt utilizate în două scopuri:
1. Pentru a exprima disjuncții obișnuite ca [Rain, Sleet, Snow] pentru a reprezenta cunoștințe incomplete.
2. Clauze de tipul [¬Child, ¬Male, Boy] - deși poate fi citită ca disjuncție "cineva nu este copil, sau nu este de gen masculin, sau este băiat", este mult mai natural s-o înțelegem ca un condițional "dacă cineva este copil și este de gen masculin atunci este băiat".

Def. O clauză Horn conține cel mult un literal pozitiv.
Dacă nu există niciun literal pozitiv, avem o clauză Horn negativă.
Obs. Clauza vidă este o clauză Horn negativă.

Clauza Horn $\{\neg p_1, \ldots, \neg p_n, q\}$ poate fi citită "dacă p_1 și ... p_n atunci q" și se mai scrie $p_1 \wedge \ldots \wedge p_n \Rightarrow q$ pentru a accentua condiționalul. Afirmațiile "if-then" s.n. reguli.
FOL tratează disjuncțiile și cunoștințele incomplete într-o formă mult mai generală decât clauzele Horn.

Derivări prin rezoluție a clauzelor Horn

Obs. Două clauze negative nu pot intra împreună în rezoluție.
O clauză negativă și una pozitivă produc prin rezoluție o clauză negativă. Două clauze pozitive produc prin rezoluție o clauză pozitivă.
Deci rezoluția peste clauze Horn implică întotdeauna o clauză pozitivă.

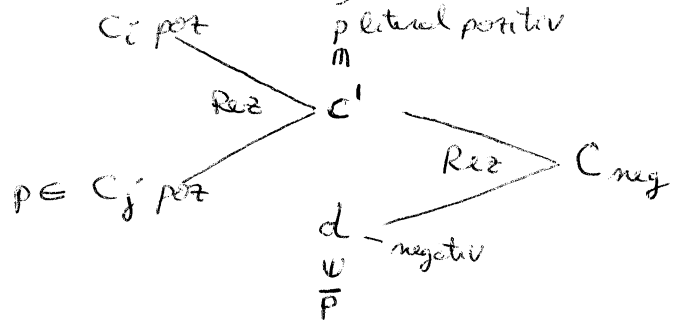
Prop. Dacă S este o mulțime de clauze Horn, $S \vdash C$ unde C este o clauză negativă, atunci există o derivare a lui C în care toate clauzele noi (adică cele ce nu sunt în S) sunt negative.

Dem [$C_1 \dots C_n = C$ derivare dacă $C_i \in S$ sau C_i este rezolventa a 2 clauze precedente din derivare]

Presupunem că avem o derivare cu clauze noi pozitive. Fie C' ultima dintre ele.

$$C_1, \dots, C', \underbrace{\dots C_n = C}_{\text{negative}}$$
C' pozitivă

În loc să generăm clauze negative folosind C', generăm aceste clauze folosind părinții pozitivi ai lui C'



C_i Rez $C_j \longrightarrow C'$ $\quad C'$ Rez $d \longrightarrow C_{neg}$

se înlocuiește cu

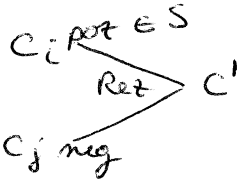
$(C_j$ Rez $d)$ Rez $C_i \longrightarrow C_{neg}$

Deci putem să obținem C_{neg} fără a folosi C'. Atunci renunțăm la C' din derivarea inițială și repetăm procedura pentru fiecare clauză pozitivă nouă. Astfel, le eliminăm pe toate

Prop. Dacă S este o mulțime de clauze Horn, $S \vdash C$, unde C este o clauză negativă, atunci există o derivare a lui C în care fiecare clauză nouă derivată este negativă și este rezolventa clauzei precedente din derivare cu o clauză din S.

Dem. Din propoziția anterioară, putem presupune că toate clauzele noi din derivare sunt negative. Deci clauzele pozitive sunt din S

$$C_1, \dots C', \dots C_n = C$$
clauză derivată negativă



$S \Downarrow$

C_k neg — poz — C_2 neg — Rez — C_1 neg — Rez — C

C_k' poz din S $\quad C_2'$ poz din S $\quad C_1'$ poz $\in S$

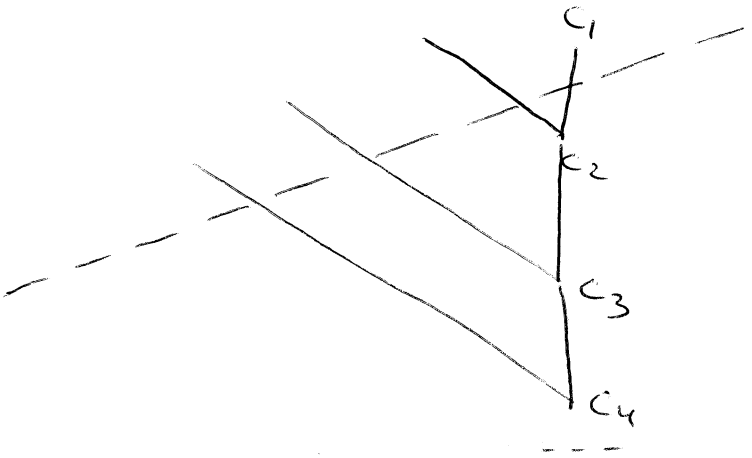
și renumtăm la toate celelalte clauze ce nu apar în acest lanț

Deci există o derivare a unei clauze negative C (inclusiv []) dintr-o mulțime S de clauze Horn dacă există o derivare în care fiecare clauză nouă este rezolventa negativă a clauzei precedente din derivare și a clauză din S.

Rezoluția SLD (Selected literals, Linear pattern, over Definite clauses)

Este o formă restricționată a rezoluției în care fiecare clauză nouă introdusă este rezolventa clauzei precedente și a unei clauze din mulțimea originală S. Această variantă de rezoluție este suficientă pentru clauzele Horn.

Dacă S este o mulțime de clauze (nu neapărat Horn), o derivare SLD din S a unei clauze C este o secvență $C_1, \dots C_n$ a.î. $C_n = C$, $C_1 \in S$ și C_{i+1} este rezolventa dintre C_i și o clauză din S. Se notează $S \vdash_{SLD} C$.



Cu excepția lui C_1, nu se menționează explicit elementele din S.

În general, dacă $S \vdash_{SLD} []$ atunci $S \vdash []$, dar reciproca nu este adevărată.

De exemplu, pentru $S = \{[p, q], [\neg p, q], [p, \neg q], [\neg p, \neg q]\}$ S nu poate fi satisfăcută, deci $S \vdash []$.

Pentru a genera [] prin rezoluție, ultimul pas trebuie să fie de forma $[p]$ și $[\bar{p}]$ pentru un literal p. Dar S nu conține clauze unitare, deci nu avem un element în S pentru ultimul pas al rezoluției.

Adică $S \vdash []$ fără a exista o derivare SLD a [] din S.

Dacă S este o mulțime de clauze Horn, atunci $S \vdash []$ dacă $S \vdash_{SLD} []$.

Mai mult, putem presupune că fiecare clauză nouă $C_2, \dots C_4$ este negativă.

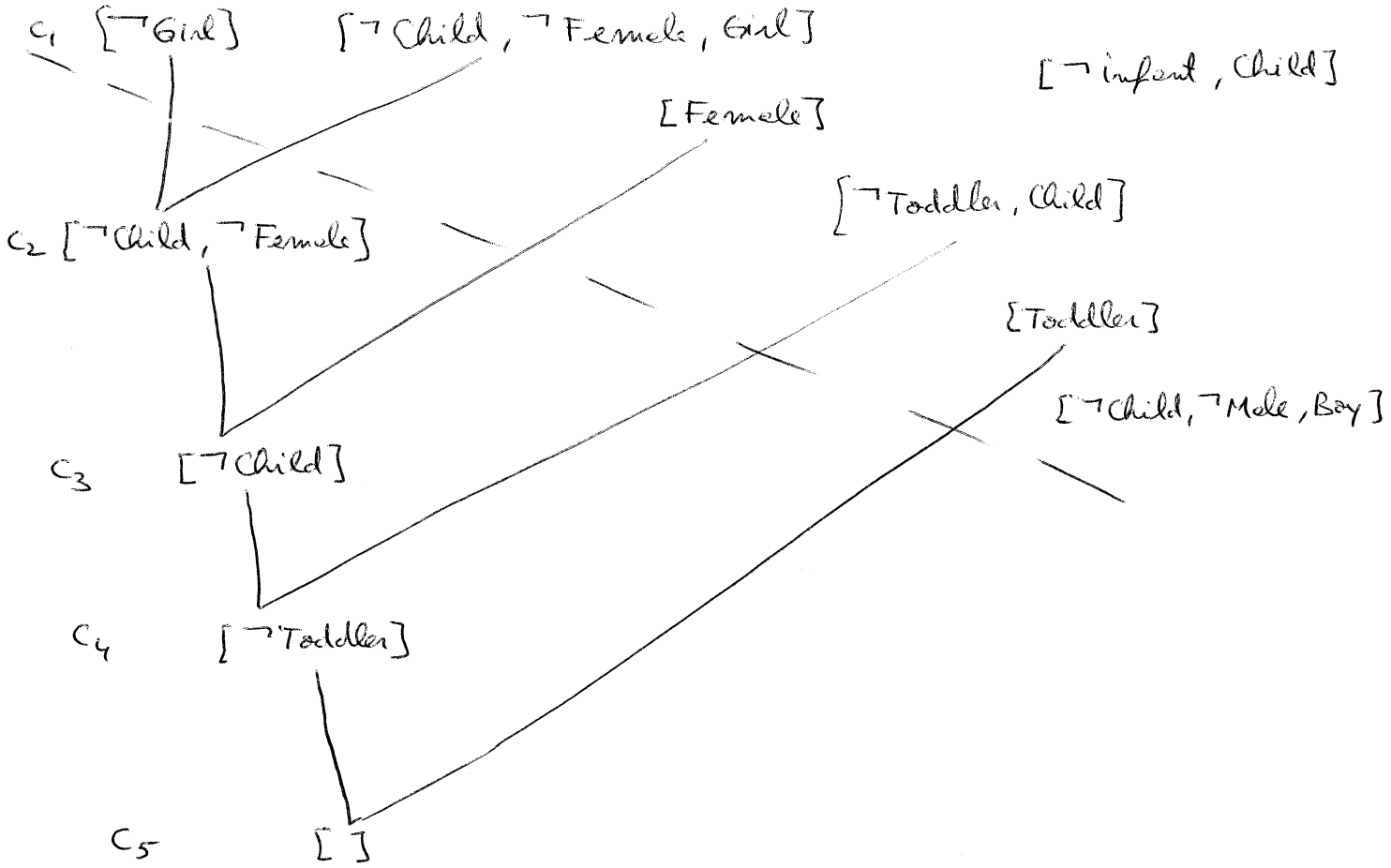
C_2 are un părinte pozitiv și unul negativ deci C_1 poate fi doar cel negativ

În cazul Horn, derivările SLD ale clauzei vide încep cu o clauză negativă din mulțimea originală.

Exemplul 1. Fie KB Horn:
- Toddler
- Toddler $\supset$ Child
- Child $\wedge$ Male $\supset$ Boy
- Infant $\supset$ Child
- Child $\wedge$ Female $\supset$ Girl
- Female

Întrebare: Girl

Obs. $[\neg Girl]$ este singura clauză negativă din S, deci $C_1 = [\neg Girl]$



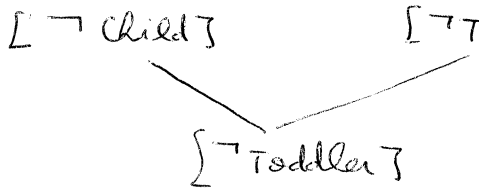
C_1, C_2, C_3, C_4, C_5 derivare SLD

Arbori de scop

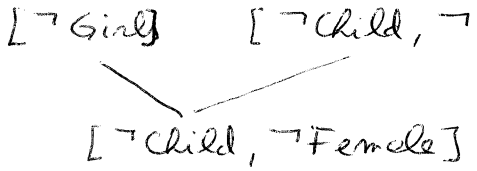
Toți literalii din toate clauzele unei derivări SLD Horn cuta [] sunt negativi. Pentru a produce [], avem nevoie de clauze pozitive din KB pentru a elimina literalii negativi.

Dacă avem [¬Toddler] în derivare și [Toddler] în S spunem că scopul Toddler este atins (rezolvat) (adică este eliminat de o clauză unitară din S)

Dacă avem o clauză pozitivă ce elimină literalul dar introduce alți literali negativi

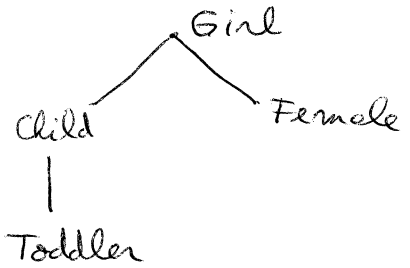


Spunem că scopul Child se reduce la scopul Toddler



scopul Girl se reduce la două subscopuri: Child și Female

În exemplul 1, derivarea SLD poate fi reformulată astfel: Pornim cu scopul Girl, acesta se reduce la două subscopuri Child și Female; scopul Female este rezolvat; Child se reduce la Toddler; Toddler este rezolvat.



Arborele de scop asociat derivării SLD

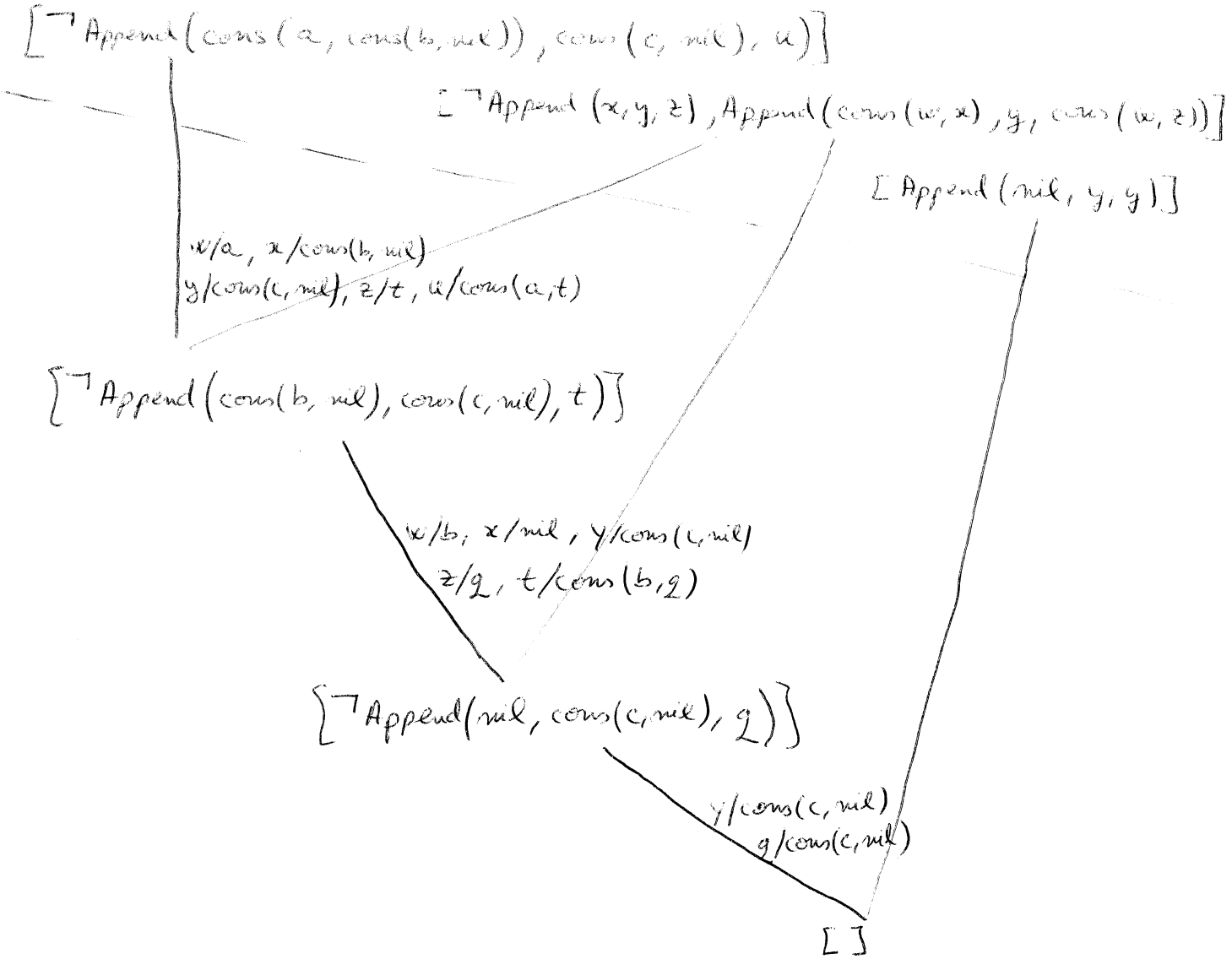
Într-o derivare completă SLD, frunzele arborelui vor fi scopuri rezolvate

Privite ca arbori de scop, clauzele Horn și derivările SLD constituie baza limbajului de programare PROLOG.

Exemplul 2 - Concatenare de liste

KB { Append (nil, y, y) (nil: listă vidă)
   { Append (x, y, z) ⇒ Append (cons(w, x), y, cons(w, z)) (w: termen, x: listă)

întrebare: ∃u. Append (cons(a, cons(b, nil)), cons(c, nil), u)



Arborele de scop: $[\neg Append(cons(a, cons(b, nil)), cons(c, nil), u)]$

$\downarrow$

$[\neg Append(cons(b, nil), cons(c, nil), t)]$

$\downarrow$

$[\neg Append(nil, cons(c, nil), q)]$

Răspunsul $u = cons(a, cons(b, cons(c, nil)))$ se obține din derivare

$q/cons(c, nil) \longrightarrow t \neq cons(b, q) \longrightarrow u/cons(a, t)$

În cazul Horn, nu este necesar să folosim predicate răspuns în derivările SLD deoarece $S \vdash \exists x.\alpha$ dacă există un termen t a.î. $S \vdash \alpha_t^x$

Calculare derivinilor SLD

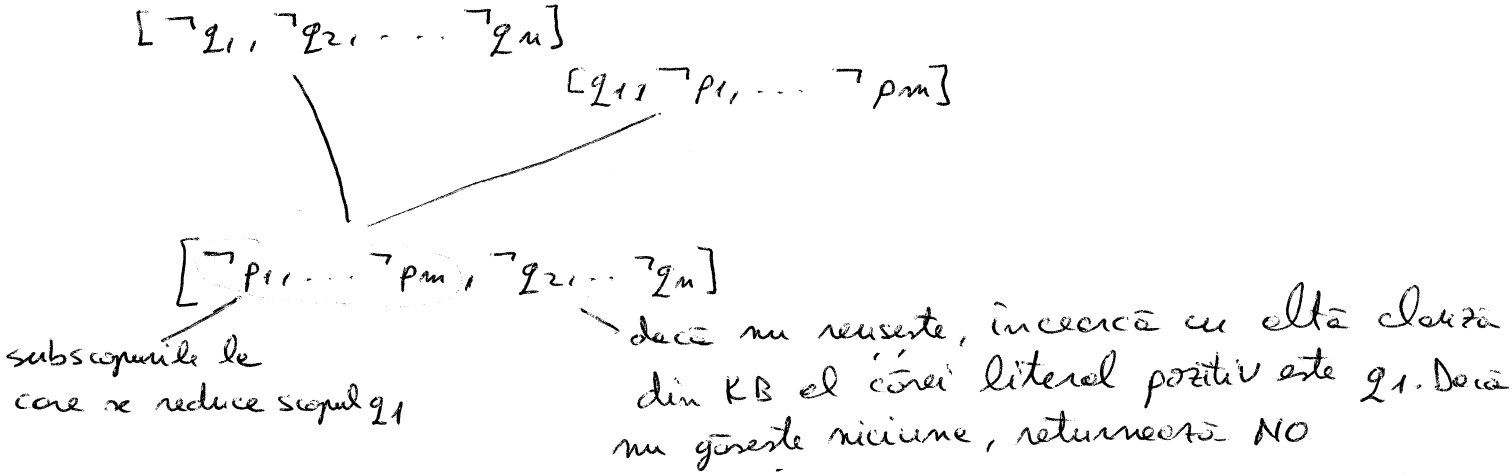
Avem o bază de cunoștințe KB cu clauze Horn pozitive (reprezentând propoziții "if-then") și vrem să aflăm dacă o mulțime de atomi poate fi dedusă logic. Adică dacă KB plus o clauză formată dintr-unul sau mai multi literali negativi nu poate fi satisfăcută. Scopul considerat constă în a determina satisfacerea tuturi mulțimi de clauze Horn ce conține doar o clauză negativă.

Înlănțuire înapoi

intrare: o listă finită de propoziții atomice $q_1 \dots q_n$
ieșire: DA sau NU, după cum KB deduce sau nu toți q_i, $i=1,n$

```
procedure SOLVE [q1, ... qn]
    if n=0 then return DA
    for each clauza c ∈ KB
        if c=[q1, ¬p1, ... ¬pm] and SOLVE[p1 ... pm, q2 ... qn]
            return DA
    return NO
```

Căutarea se face înapoi, de la scopuri către faptele din KB.



Procedura se mai numește depth-first pentru că încearcă să rezolve noile subscopuri p_i înainte de a rezolva q_j.
Se mai numește left-to-right pentru că încearcă să rezolve scopurile q_i în ordine $i=1,2,\dots$
Aceasta este maniera în care PROLOG rezolvă scopurile.

Obs. Procedura poate întoarce întotdeauna răspunsul corect în cazul propozițional (dacă în KB nu apare $[p, \neg p]$)

În alte situații, complexitatea poate fi exponențială.

De exemplu dacă avem $2n$ atomi $p_0, \dots p_{n-1}, q_0, \dots q_{n-1}$

și clauzele
$$\begin{cases} p_{i-1} \Rightarrow p_i \\ p_{i-1} \Rightarrow q_i \\ q_{i-1} \Rightarrow p_i \\ q_{i-1} \Rightarrow q_i \end{cases} \quad 0 < i < n$$
avem $4n-4$ clauze

Pentru orice i, SOLVE$[p_i]$ și SOLVE$[q_i]$ vor fi satisfăcute

$i = 1$ $[\neg p_0, p_1]$, $[\neg q_0, p_1]$ 2^1 pași

pp că pentru $k-1$ sunt necesari 2^{k-1} pași cel puțin

$p_{k-1} \Rightarrow p_k$ se efectuează $2^{k-1} + 2^{k-1} = 2^k$ pași cel puțin

$q_{k-1} \Rightarrow p_k$

Înlănțuirea înainte

Se pleacă de la faptele din KB spre scopuri.

intrare: o listă finită de propoziții atomice $q_1, \dots, q_n$

ieșire: DA sau NU, după cum KB implică sau nu toți q_i

procedure
1.- dacă toate scopurile q_i sunt rezolvate return DA
2.- Verifică dacă există în KB o clauză $[p, \neg p_1, \dots \neg p_n]$ în care toți atomii negativi sunt rezolvați iar atomul pozitiv nu este rezolvat
3.- dacă da, atunci marchează p ca rezolvat și mergi la 1
altfel return NO

Deci marcăm un atom ca fiind rezolvat când determinăm că este dedus logic de KB.

În exemplul 1 Girl nu e marcat/rezolvat

[Toddler] - are 0 atomi negativi deci marchez Toddler

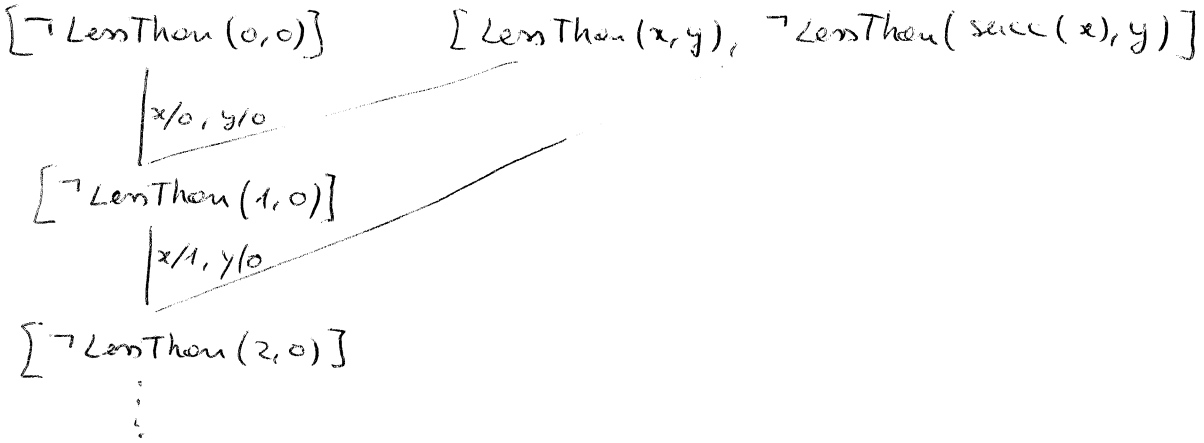
[Child, ¬Toddler] - marchez Child

[Female] - îl marchez

[Girl, ¬Child, ¬Female] - marchez Girl - return DA

Înlănțuirea înainte este mult mai eficientă decât înlănțuirea înapoi. La fiecare iterație, vom percurge KB pentru a găsi o clauză cu un atom ce nu a fost marcat. Dacă rezultatul nu va fi exponențial

În cazul propozițional putem determina dacă o bază de cunoștințe Horn deduce logic un atom. Dar pentru FOL, chiar cu clauze Horn există posibilitatea ca o procedură să nu se termine.



Problema determinării dacă o mulțime de clauze Horn din FOL deduce un atom este nedecidabilă.