Clauses de Horn

Définition

on note $p(C)$ (resp. $n(C)$) le nombre de littéraux positifs (resp. négatifs) dans la clause C. Une *clause de Horn* est une clause C telle que $p(C) \leq 1$.
Si $p(C) = 1$ on appelle *tête* de C le littéral positif de C.

Théorème

si R est la résolvante de C_1 et C_2 alors

$$p(R) = p(C_1) + p(C_2) - 1 \text{ et } n(R) = n(C_1) + n(C_2) - 1.$$

Donc si C_1 et C_2 sont des clauses de Horn alors R aussi.

Démonstration.

On obtient R en enlevant un littéral positif et un négatif à $C_1 \vee C_2$.
Donc $p(R) = p(C_1) + p(C_2) - 1 \leq 1 + 1 - 1 = 1$

Notes

Définition

On distingue 4 formes de clauses de Horn C :

1. la clause vide si $p(C) = 0$ et $n(C) = 0$: $C = \square$
2. une clause *négative* si $p(C) = 0$ et $n(C) > 0$
3. une clause *positive* si $p(C) = 1$ et $n(C) = 0$
4. une clause *définie* si $p(C) = 1$ et $n(C) > 0$

On note N pour une clause négative ou vide, P, Q un littéral positif
La forme d'une résolvante dépend de celle des prémisses :

- Nég + Nég, Pos + Pos : pas de résolvante
- Nég + Pos : $\neg P \vee N + P \to N$, Nég ou $\square$
- Nég + Déf : $\neg P \vee N + P \vee N' \to N \vee N'$, Nég (car $N' \neq \square$)
- Pos + Déf : $P + Q \vee \neg P \vee N \to Q \vee N$, Pos ou Déf
- Déf + Déf : $P \vee N + Q \vee \neg P \vee N' \to Q \vee N \vee N'$, Déf,

Il *faut* utiliser 1 clause Nég pour : obtenir une clause Nég, obtenir $\square$

Notes

La forme d'un facteur dépend de celle de la prémisse :

- Pos : P, pas de factorisation possible
- Nég : $\neg P \vee \neg P' \vee N \to \neg P \vee N$, Nég
- Déf : $Q \vee \neg P \vee \neg P' \vee N \to Q \vee \neg P \vee N$, Déf

Théorème

Si H est un ensemble insat. de clauses de Horn, alors il existe une réfutation de H par résolution et factorisation, et l'ensemble des clauses négatives de cette réfutation forme une "branche" de la preuve.
Si H est minimalement insat. ($\forall \psi \in H$, $H \setminus \{\psi\}$ est sat.) alors il contient exactement une clause négative

Théorème (admis)

Si H est un ensemble insat. de clauses de Horn, alors il existe une réfutation de H par résolution ne contenant qu'une seule "branche" (c'est une preuve linéaire).

Notes

Donc on réfute une clause négative par résolutions successives avec des clauses Pos ou Déf de H. Exemple :

$P = p(a,a)$, $D = p(f(x), f(f(y))) \vee \neg p(x,y)$ et $N = \neg p(f(f(a)), z)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg p(f(f(a)),z) \qquad D}{\neg p(f(a),y)}\{z \mapsto f(f(y))\} \qquad D\{y \mapsto y'\}}{\neg p(a,y')}\{y \mapsto f(f(y'))\} \qquad P}{\square}\{y' \mapsto a\}$$

En composant les substitutions on obtient

$$\{z \mapsto f(f(f(f(a)))),\ y \mapsto f(f(a)),\ y' \mapsto a\}$$

On voit donc qu'à chaque réfutation de N par H on peut associer une substitution σ des variables de N telle que $N\sigma$ admet une réfutation propositionnelle par des instances de H
On peut alors voir H comme un *programme*, N comme une expression à évaluer (une *requête*) et σ comme un *résultat* de cette évaluation

- S'il y a plusieurs réfutations, il y a plusieurs résultats,
- les réfutations d'un N diffèrent selon les clauses Déf et Pos utilisées dans la réfutation

Notes

PROLOG

Convention lexicale : les identificateurs commençant par

- une majucule sont dans $\mathcal{V}$
- une minuscule sont dans Σ (Σ_n^{F} ou Σ_n^{P} déterminé selon le contexte)

Syntaxe :
une clause $P \vee \neg Q_1 \vee \cdots \vee \neg Q_n$ (équivalente à $Q_1 \wedge \cdots \wedge Q_n \Rightarrow P$)
s'écrit P :- $Q_1, \ldots, Q_n$. (P si Q_1 et $\cdots$ et Q_n)
P est la *tête* de la clause, $Q_1, \ldots, Q_n$ sa *queue*
Si $n = 0$ (clause positive) on écrit simplement P.
La requête s'écrit $Q_1, \ldots, Q_n$.
On cherche à prouver que $H \models_1 Q_1 \wedge \cdots \wedge Q_n$ donc à réfuter $H \cup \{\neg Q_1 \vee \cdots \vee \neg Q_n\}$; on cherche les réfutations *dans l'ordre des clauses et des littéraux donnés*.

Notes

(C_1) `pere(pepe,titine).`
(C_2) `mere(titine,totor).`
(C_3) `pere(pepe,rafa).`
(C_4) `pere(rafa,gudule).`
(C_5) `gp(X,Y) :- pere(X,Z), pere(Z,Y).`
(C_6) `gp(X,Y) :- pere(X,Z), mere(Z,Y).`

Requêtes : `gp(pepe,Y).`

C_5 : `pere(pepe,Z), pere(Z,Y).`
C_1 : `pere(titine,Y).`
échec
C_3 : `pere(rafa,Y).`
C_4 : $\square$ réponse `Y=gudule`
C_6 : `pere(pepe,Z), mere(Z,Y).`
C_1 : `mere(titine,Y).`
C_2 : $\square$ réponse `Y=totor`
C_3 : `mere(rafa,Y).`
échec

si on intervertit C_5 et C_6 on obtient `Y=totor` avant `Y=gudule`

Notes

Programme pour parcourir l'arbre de recherche et renvoyer les unificateurs composés, avec $H = \{C_1, \ldots, C_n\}$ et $C_i = T_i \vee N_i$ (N_i clause négative ou vide dont les littéraux sont ordonnés de gauche à droite)

$$\mathrm{R}(N) = \bigcup_{i=1}^{n} \mathrm{R}'(N, i)$$

$\mathrm{R}'(\neg Q \vee N, i) =$
soit π **tq** $\mathrm{Var}(C_i\pi) \cap \mathrm{Var}(\neg Q \vee N) = \emptyset$
si $\sigma = \mathrm{unif}(Q, T_i\pi)$ **alors**
si $N = N_i = \square$ **alors retourner** $\{\sigma\}$
sinon retourner $\sigma\mathrm{R}(N_i\pi\sigma \vee N\sigma)$
sinon retourner $\emptyset$

(où $\sigma\{\sigma_1, \ldots, \sigma_n\} \stackrel{\text{def}}{=} \{\sigma\sigma_1, \ldots, \sigma\sigma_n\}$)
C'est un *interpréteur PROLOG*
Le parcours se fait en profondeur d'abord, on peut donc partir dans une branche infinie, donc cet algorithme est *incomplet*

Notes

Exemple

(C_1) `plus(X,s(Y),s(Z)) :- plus(X,Y,Z).`
(C_2) `plus(X,0,X).`

`plus(0,X,Y).`
C_1 : `plus(0,Y,Z). X'=s(Y), Y'=s(Z)`
C_1 : `plus(0,Y,Z). Y'=s(Y), Y'=s(Z)`
$\cdots \infty$

si C_2 est avant C_1 :

`plus(0,X,Y).`
C_2 : $\square$ `X'=0, Y'=0`
C_1 : `plus(0,Y,Z). X'=s(Y), Y'=s(Z)`
C_2 : $\square$ `Y'=0, Z'=0`
C_1 : `plus(0,Y,Z). Y'=s(Y), Y'=s(Z)`
$\cdots \infty$

Notes

On représente les données par des termes
En particulier les listes $[t_1, t_2, \ldots, t_n]$ peuvent être représentées par les termes $f(t_1, f(t_2, \ldots, f(t_n, \emptyset) \cdots))$
syntaxe spéciale pour les listes : $f =$ `|`, $\emptyset =$ `[]`, et :
notation infixée : `|`(t_1, l) est noté `[`t_1`|`l`]`
liste explicite : le terme `[`t_1`|` $\cdots$ `[`t_n`|[]]` $\cdots$`]` est noté `[`t_1`,` $\ldots$ `,`t_n`]`

Exemple

`[X,Y,Z]` = `[X|[Y|[Z|[]]]]` = `[X|[Y,Z]]` = `[X|[Y|[Z]]]`

Attention :

- `[X,Y]` est une liste de longueur 2, et toute liste de longueur 2 s'unifie avec `[X,Y]` (par exemple `[a,a]`, `[a,f(a)]`, `[[a],[]]`...)
- `[X|L]` est une liste de longueur ≥ 1, et toute liste de longueur ≥ 1 s'unifie avec `[X|L]` (par exemple `[a]`, `[a,b,c]`, `[a,[]]`...)
- `[a|b]` n'est pas une liste

Notes

Pour calculer $f(t_1, \ldots, t_n)$ on décrit un prédicat $p(t_1, \ldots, t_n, s)$ équivalent à $s = f(t_1, \ldots, t_n)$, en supposant que les t_i seront unifiés avec des termes fermés. On commence par les clauses positives $p(t_1, \ldots, t_n, s)$ pour les valeurs connues.

Exemple

Calcul de la longueur d'une liste :
(C_1) `lngr([],0).`
(C_2) `lngr([X|L],s(N)):- lngr(L,N).`
Remarque : la valeur de `X` n'est pas utilisée car `X` n'apparaît qu'une fois ; on peut écrire
(C_2) `lngr([_|L],s(N)):- lngr(L,N).`

On peut parfois aussi calculer l'inverse de la fonction f

Exemple

la requête `lngr(L,s(0)).` renvoie `L=[_G996]`

Notes