

# I le langage (Étape : implémentation de PROLOG)

## 1) Les Objets syntaxiques

Les seules entités syntaxiques de prolog sont les termes et les formules

Comme ds L.

- Des termes désignent (directement ou indirectement) des éléments d'un domaine D (domaine de ~~HERBARIUM~~ HERBARIUM)
- Des formules désignent des propriétés concernant les éléments de D

Pour construire termes et formules on utilise un vocabulaire qui comprend

V : ensemble infini de symboles de variables (ident ou majuscules)

Une variable représente un terme quelconque (cf def de substitution)

F : ensemble de symboles de fonctions ( $f_1 + f_2 + c_1$ )  
constante

R : ensemble de symboles de relations (prédicat)

à chaque symbol de fonction et de relation PROLOG attache 1 arité  
termes.

si  $X \in V \rightarrow X$  est un terme

si  $f_n \in F \rightarrow f(t_1, \dots, t_n)$  forme

si  $t_i$  : termes  
 $i \in \{1, \dots, n\}$

si  $f_0 \in F \rightarrow f$  terme (constante)

si  $n$  est un nombre  $\rightarrow n$  terme

Formules :      true  
                    false  $\rightarrow$  Formules

si  $\alpha \in R \rightarrow \alpha(t_1, \dots, t_n)$  forme

si  $t_i$  terme  
 $i \in \{1, \dots, n\}$

si  $\gamma_0 \rightarrow$  Formule (proposition)

$=_{12} \rightarrow = (t_1, t_2)$  Formule

P formule  $\rightarrow$  P, q  
9  $\wedge$  sémantique de  $\wedge$

## 2) Interprétation

- les symboles de relation ~~sont~~ d'ordre n motent (designent) des relations  
ie des sous ensembles de  $D^n$

- les symboles de fonction d'ordre n motent des opérations  
ie des applications de  $D^n$  dans  $D$ .

mais, quel est le domaine  $D$ ? Le domaine des valeurs manipulées par PROLOG

$D$ : ensemble des arbres dont les noeuds sont étiquetés par

- des identificateurs de PROLOG, munis de leur arité
- par des nombres.

Pour l'instant:  $X \in V$ , variable donc terme, designe 1 autre variable (l'autre syntacique du terme)

$f(t_1, \dots, t_n)$  note l'arbre dont la racine est étiqueté par  $f$  et dont les sous arbres sont notés par  $t_1, \dots, t_n$ .

## 3) Qu'est-ce qu'un programme PROLOG?

on fournit 2 réponses

a) la réponse du programmeur

i) syntaxiquement

- Un programme ait un séquence de paquets de clauses
- Un paquet de clauses est une liste non vide de clauses
- une clause est de la forme.

$$r(t_1, \dots, t_n) : - p$$

ou  $r(t_1, \dots, t_n)$ .

$r_m \in R$  symbole de prédicat m-aire

$t_i, i \in [1, n]$  termes.

P : formule construite avec les symboles de V F et R

P ensemble de symboles de prédicat non prédefinis

Exemple : age(jean, 20).

age(pierre, 12).

jame(x) : - age(x, y), <(y, 25).

## ② démantèlement

Pour le programmation, chaque a un sens

Une clause est une affirmation de forme = P<sub>1</sub>

la seule chose à avoir : la signification de

: - "à condition que"

, signifie presque ▲

## Vocabulaire.

tête de clause : partie de clause ayant : -

corps ----- opéros : -

- implicitement les variables de la tête de clause sont universellement quantifiées

- ----- du corps de clause ----- existentiellement -----  
qui n'apparaissent que dans le corps de clause

## • La répère du logicien

① Un prog P en prolog est la notation d'un ensemble de clauses de L

Exemple = H = {age(jean, 20), age(pierre, 12), ∃ age(x, y) ∨ ∃ inf(y, 25),  
∨ jeime(x)}.

En effet toute clause de Prolog peut s'écrire sous forme d'une clause de L  
ayant exactement 1 littoral positif

- devant pour les clauses sans corps

- pour les autres

¬(t<sub>1</sub>, ..., t<sub>n</sub>) : - p signifie ¬(t<sub>1</sub>, ..., t<sub>n</sub>) à condition que p

p → ¬(t<sub>1</sub>, ..., t<sub>n</sub>)

¬p ∨ ¬(t<sub>1</sub>, ..., t<sub>n</sub>)

cas 1 : p réduit à 1 seule formule vide

cas p2:  $p_1, \dots, p_n$

$\neg p_1 \neg \dots \neg p_n$

$\neg(p_1 \neg \dots \neg p_n) \vee \neg(\neg p_1 \neg \dots \neg p_n)$

$\neg p_1 \vee \neg p_2 \vee \dots \vee \neg p_n \vee \perp (\perp)$

Ces clauses ayant exactement 1 littéral positif s'appellent

clauses de HORN

1) Toute clause de L ne possède pas 1 clause de HORN logiquement équivalente.

$\Rightarrow$

PROLOG ne possède pas tt le pouvoir d'expression de L.

Mais, relativement à L, ces clauses possèdent des avantages

1) un ensemble de clauses de HORN n'est jamais contradictoire, il est impossible d'inférer  $\perp$ .

2) Efficacité de la résolution.

2) Un prog PROLOG est la def d'un ensemble (généralement inf) de littéraux positifs qui sont conséquences logique du programme.

$$E = \{ L \mid P \models L\}$$

exemple :  $E = \{ \text{age(jean, 20)}, \text{age(pierre, 12)}, \text{jeune(jean)}, \text{jeune(pierre)} \}$

1) Comment fonctionne Prolog vis à vis d'un programme P

exactement Prolog n'est pas "calculer" tous les littéraux sous logique de P

Prolog dirige la partie de cet ensemble qui intéresse le programmeur

En fait, après la progr/réduction de son prog (clauses de HORN) la PE achache des requêtes

a) syntaxe

exactement la syntaxe d'un coup de clause  $P$ . ( $P$  est une formule)

b) sémantique

? | age(jean, 20).  
| age(pierre, 12).  
| jeune(X):- age(X, Y), <(Y, 25).

$\Leftrightarrow$  note {age(jean, 20), age(pierre, 12),  $\neg \text{age}(x, y) \vee \neg <(y, 25)$   
 $\vee \text{jeune}(x)$ }

R [E 17] j'aime (L)

$\exists$   $U$  j'aime ( $U$ )! Prolog utilise la négation

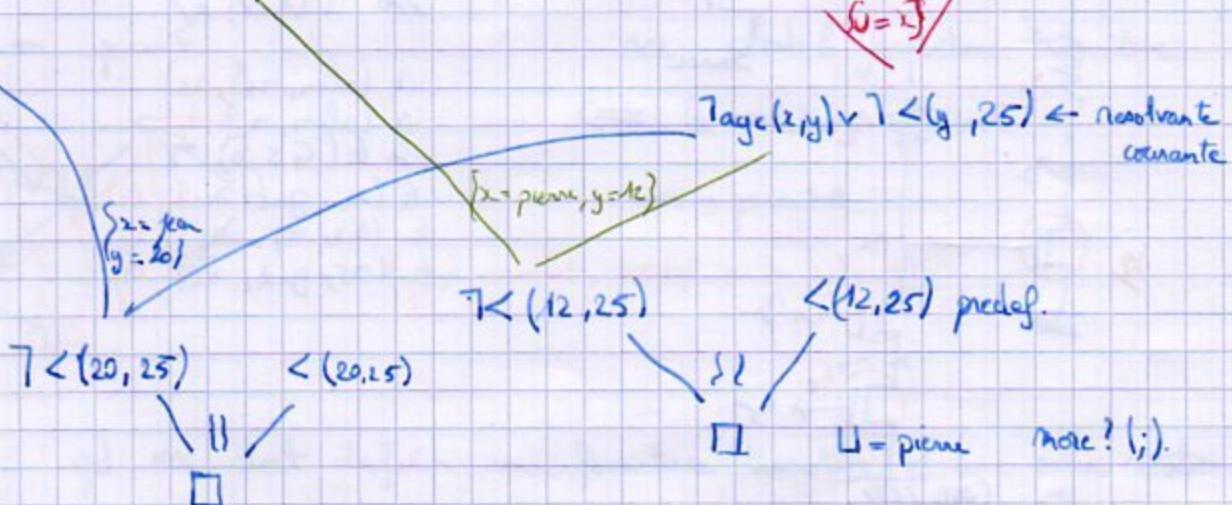
on ajoute aux clauses de P la négation de la requête

$\neg U \neg$  j'aime ( $U$ )

P U { $\neg R$ }

{ age (yann, 20), age (pierre, 12), j'aime (x, y)  $\vee$  K(y, 25)  $\vee$  j'aime ( $x$ ),  $\neg$  j'aime ( $U$ ).

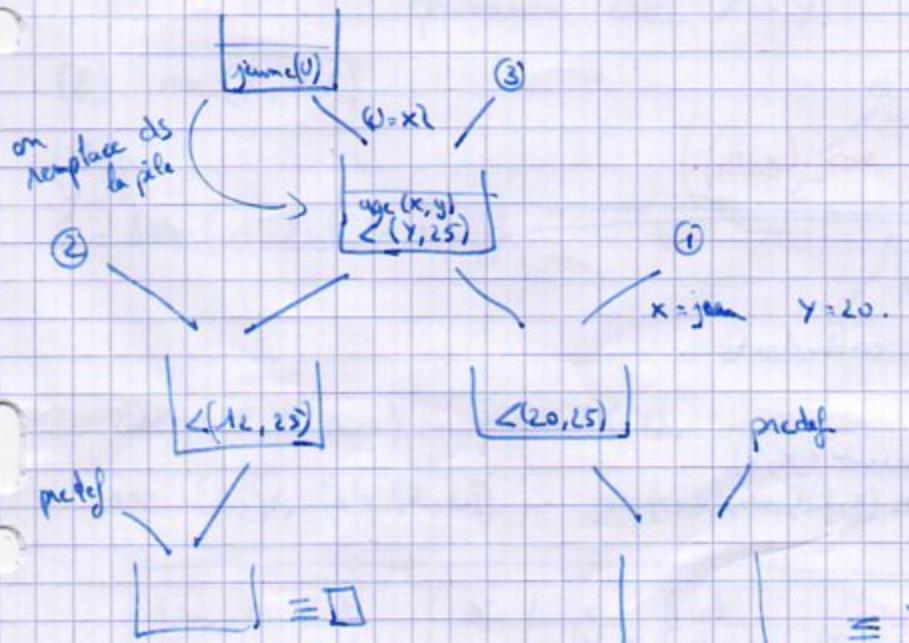
$U = x$



d) mise en œuvre

- le programme P est mis en mémor (avec accès rapide grâce à un index)
- la résultante courante est mise dans une pile (dite pile des buts) (pile de littérature)
- donc initialement, la pile des buts contient la requête

P	age	_____	①
	j'aime( $U$ ):-	_____	②.



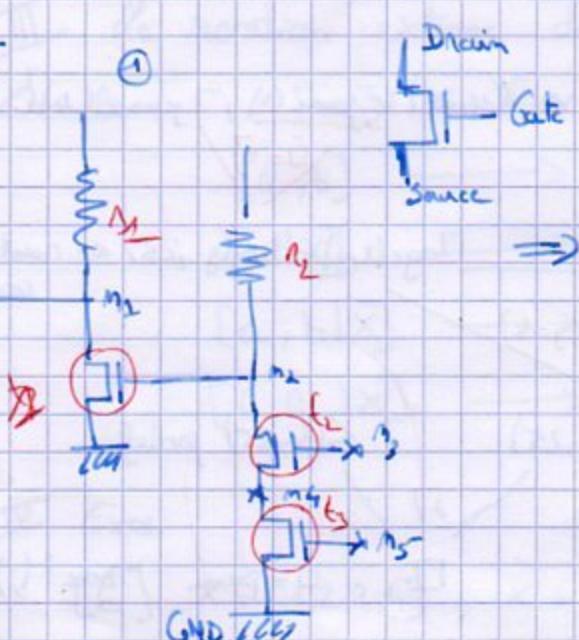
Prolog: développe un arbre de recherche en profondeur d'abord

(+): peu contenu en mémoire car 1 seule branche en mémoire

Cours mac.

I

①



Drain  
Gate  
Source

$\Rightarrow$

Prolog.

$\text{not}(x, y) \vee$

$\neg(\text{alim}, m_1, n_1)$

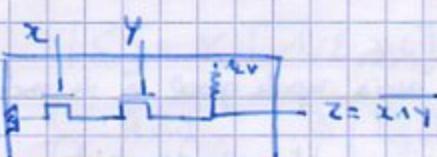
$\neg(\text{alim}, m_2, n_2)$

~~$\neg t(GS D)$~~

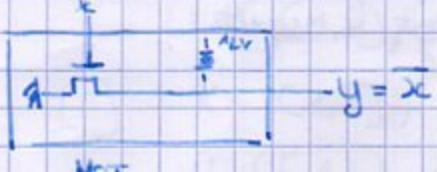
$t(m_2, \text{gate}, m_3, t_1)$

$t(m_3, m_4, m_2, t_2)$

$t(m_5, \text{gate}, m_4, t_3)$



NAND



NOR

nommer les composants

$C = \text{and}(\text{nand}(t_2, t_3, n_2), \text{not}(t_1, n_1))$

Le monde de la famille

parent(Z) trouver les descendants

V<sub>1</sub>

① abc(X, Y) :- parent(X, Y);

abc(X, Y) :- abc(X, Z), abc(Z, Y)

V<sub>2</sub>

① abc(X, Y) :- gabc(X, Z), parent(Z, Y)

mand(A,B,C)

mand(X,Y,Z) :- t(X, U, gnd),

~~t(Y, Z, U)~~

~~t(alim, Z)~~

A

B

mon(A,B)

not(X,Y) :- t(X, V, gnd), ~~t(Y, Z, V)~~, ~~t(alim, Z)~~

and(X,Y,Z) :- mon(X,Y,W), ~~not(W,Z)~~

and(A,B)

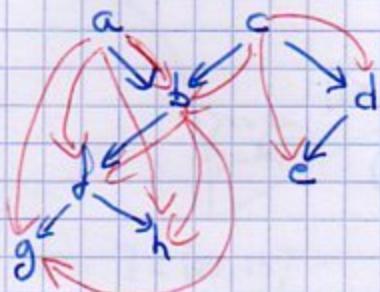
$v_3$

$\text{asc}(x, y) :- \text{parent}(x, z), \text{asc}(z, y)$

du pt de vue logique  $V_1, V_2, V_3$  sont corrects  
seule  $V_3$  marche

programmation de la fermeture transitive

relation parent



$\text{asc}$ : c'est la fermeture transitive

de la relation parent

$\text{asc} \equiv \text{parent}^*$ .

(pas complet, breflet  $a$  en rajoute)

Règle : qd on doit définir une fermeture transitive d'une autre relation  
il faut ~~écrire~~ prescrire la recursivité à gauche (ie éviter l'appel  
récuratif entier du corps de clause).

des listes

I Exemple introductif

Définir le prédicat  $\text{asc}_3 \leftarrow$  d'arité 3.

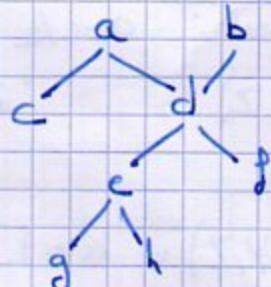
$\text{asc}(X, Y, L)$  "X est ascendant de Y et L est la liste des  
personnes entre X et Y"

[E]  $\text{asc}(X, Y, L)$

$X = a$

$Y = b$

$L = \text{liste}(d, \text{liste}(e, \text{nil}))$ .



$\text{asc}(X, Y, \text{nil}) \leftarrow \text{parent}(X, Y)$ .

$\text{asc}(X, Y, \text{liste}(Z, L)) \leftarrow \text{parent}(X, Z), \text{asc}(Z, Y, L)$ .

$\text{nil} \in \text{liste}$

si  $L \in \text{liste}$

$\text{liste}$

si E élément alors  $\text{liste}(E, L) \in \text{liste}$

## II les listes offertes par PROLOG

non  $\rightarrow \square$

liste  $\rightarrow .$

## III la notation externe des listes

Exemple:  $[a] \longrightarrow .(a, \square)$

$[a, b] \longrightarrow .(a, .(b, \square))$

$[a | X] \longrightarrow .(a, X)$

$[a, b | X] \longrightarrow .(a, .(b, X))$

$[A, B | X] \longrightarrow .(A, .(B, X))$

## IV Exos.

[E]  $X = [1, 2, 3, 4]$ ,  $X = [A, B]$

No

[E]  $X = [1, 2, 3, 4]$ ,  $X = [A | B]$

$A = 1$      $B = [2, 3, 4]$

[E]  $X = [1, 2]$ ,  $Y = [3, 4]$ ,  $U = [X, Y]$

$U = [[1, 2], [3, 4]]$

[E]  $X = [1, 2]$ ,  $Y = [3, 4]$ ,  $U = [X, Y]$ .

$U = [[1, 2], 3, 4]$ .

Définir le predicate conc/3

conc (X, Y, Z)

"Z est la concaténation de X et Y"

$X, Y, Z \in \text{liste}$

conc (X, Y, Z) :- X = [], Z = Y.

\_\_\_\_\_ : - X = [A|R], conc (R, Y, U), Z = [A | U].

conc ([], Y, Y).

conc ([A|R], Y, [A|U]) :- conc (R, Y, U).

## L'arithmétique :

- Les numéros appartiennent au langage
- on permet aussi de mélanger les expressions arithmétiques à condition d'utiliser la syntaxe des termes.

## RECOUP

le prédictat  $\text{is}/2$ .

$\text{is}(\overline{T}, \overline{\text{Te}})$

probz devant fonctionnel

Le terme  $\text{Te}$ , qui doit désigner l'exp arith est évaluée (si possible) et le résultat est unifié à  $T$ .

$[\Sigma] \text{ is } (8, 5+3)$

$[\Sigma] \text{ yes } \text{ is } (8, 5+x)$

$[\Sigma] \text{ is } [5+3, 5+3]$

$[\Sigma] \text{ no } \text{ is } [X, 5+3]$

$\rightarrow \textcircled{1} 5+3 \text{ n'est pas évalué } 5+3+8$

$X=8$ .

Eto préfixe  $/2$

prefixe  $(X, Y)$  la liste  $Y$  commence par la tête  $X$ .

prefixe  $([], Y)$ .

prefixe  $([A|R], [A|L]) :- \text{prefixe } (R, L).$

ou

prefixe  $(X, Y) :- \text{conc.}(X, -, Y).$

Les listes (clim)

| méthode de construction de prédictat

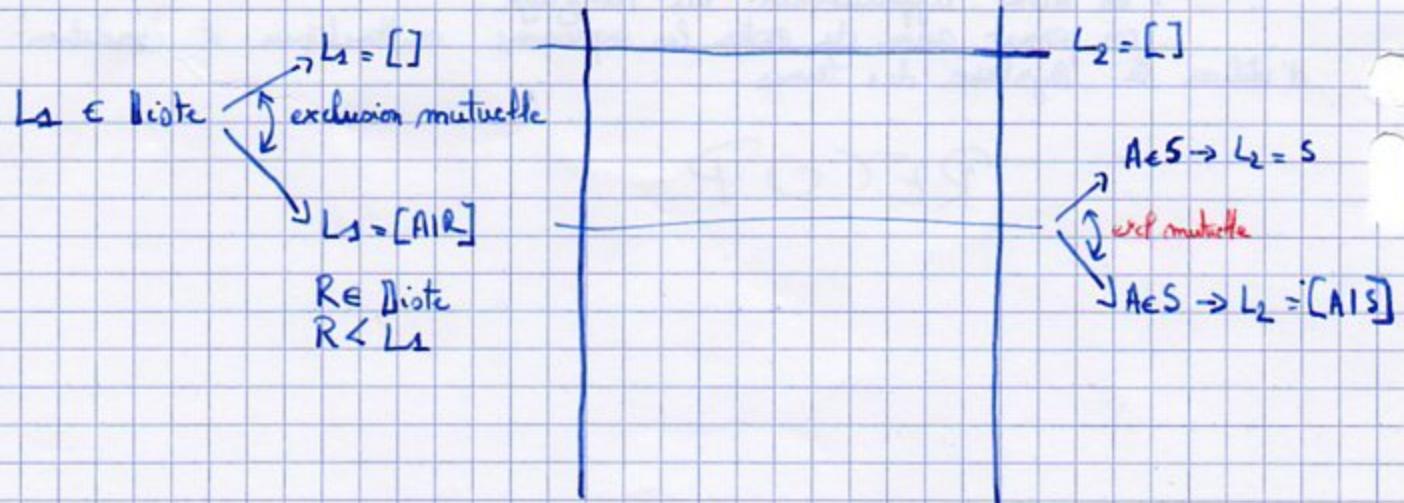
clim  $/2$

$\text{clim}(\overline{L_1}, \overline{L_2})$

$L_2$  contient tout les éléments de  $L_1$ , une fois (et rien d'autre)

méthode :

on raisonne sur  $L_1$  (entrée) (type)



clim ( $L_1, L_2$ ):-  $L_1 = []$ ,  $L_2 = []$ .

clim ( $L_1, L_2$ ):-  $L_1 = [A|R]$ , clim ( $R, S$ ), membre ( $A, S$ ),  $L_2 = S$ .

clim ( $L_1, L_2$ ):-  $L_1 = [A, R]$ , clim ( $R, S$ ), horde ( $A, S$ ),  $L_2 = [A, S]$ .

programme obtenu correct mais inefficace  
[ε] clim ([120 éléments],  $L_2$ ).

↪ réponse très tardive.

constr ( $A, S, S$ ): - membre ( $A, S$ )  
constr ( $A, S, [A|S]$ ): - horde ( $A, S$ )

↳ piste pour améliorer

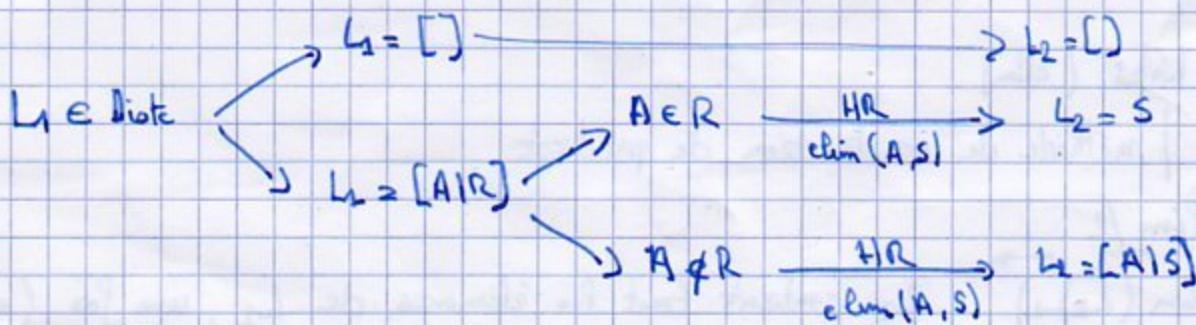
① reporter la synthèse du résultat dans un autre prédict  
eli([ ] [ ])

eli ([A|R],  $L_2$ ):- clim ( $R, S$ ), constr ( $A, S, L_2$ )

constr ( $A, S, S$ ): - membre ( $A, S$ ).

constr ( $A, S, [A|S]$ ): - horde ( $A, S$ )

② prenons l'analyse plus profondément



elim ([ ], [ ])

elim ([ A | R ], S) :- membre (A, R), elim (R, S).

elim ([ A | R ], [ A | S ]) :- horode (A; R), elim (R, S).

Exercice:

① lauf (L, L<sub>0</sub>)

L ⊂ liste  
L<sub>0</sub> la liste des suffixes de L

[ε] lauf ([1,2,3], L)

L = [[1,2,3], [2,3], [3], []]

lauf ([ ], [ ]) .

lauf ([[A|R]], [[A|R]|S]) :- lauf (R, S).

② Perauf (L, S)

L ⊂ liste  
S un suffixe de L

[ε] Perauf ([1,2,3], S).

S = [1,2,3];

Perauf (L, S) :- append (-, S, L).

S = [2,3];

tout ce qui concaténé à S donne L  
L étant la donnée ça marche

S = [];

ou

perauf (L, L).

mais

perauf ([A|R], S) :- perauf (R, S).

---

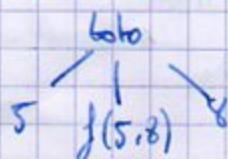
=.. / 2 : éclatement des termes

① =.. (T, L)

le terme T est décomposé, ses éléments figurent dans la liste L.

Ex = ... (toto (5, f(3,8), X), L).

L = [toto, 5, f(3,8), X, 8] La liste est d'autant du terme +1



Ferméture transitive générique

Sait R une relation d'ordre 2 on veut sa fermeture transitive

$ge(R, X, Y) :- R(X, Y).$

$ft(R, X, Y) :- R(X, Z), ft(R, Z, Y)$

L'analyse syntaxique de prolog refuse les variables qui ne sont pas au feuille de l'arbre abstrait

$\hookrightarrow ft(R, X, Y) :- =..(Rel[R, X, Y]), Rel.$

$ft(R, X, Y) :- =..(Rel, [R, X, Z]), Rel, ft[R, Z, Y].$

Arbre arbre. tous les objets (termes) sont des arbres.

Malgrès tt le programmeur a souvent intérêt à ~~se~~ définir son propre domaine des arbres

① arbres binaires Abin

$fdp(A, L)$

② ——— mères Am

$A \in Am$

③ arbre prolog.

L : liste des feuilles de A

Abin

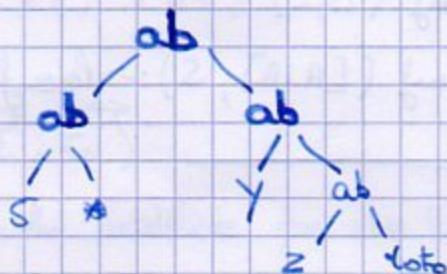
feuille  $\in$  Abin

$ge(Abin)$

$d \in Abin$

$ab(g, d) \in Abin$

feuille(X) :- =..(X, [X]).



liste des feuilles

$fdp(A, [A]) :- feuille(A).$

$fdp(ab(G, D), L) :- fdp(G, M), fdp(D, N), append(M, N, L).$

Am feuille  $\in$  Am La liste d'Am

$am(La) \in Am$

am

$[5, 12, am, 21, am]$

$[17, 14] [5, 1, 4]$

leaf(A, [A]) :- feuille(A).

leaf([an(La), Lf]) :- leaf(La, Lf).

leaflist([ ], [ ]).

leaflist([A|L], [ ]) :- leaf(A), leaflist(L, [ ]), append([A], L, [ ]).

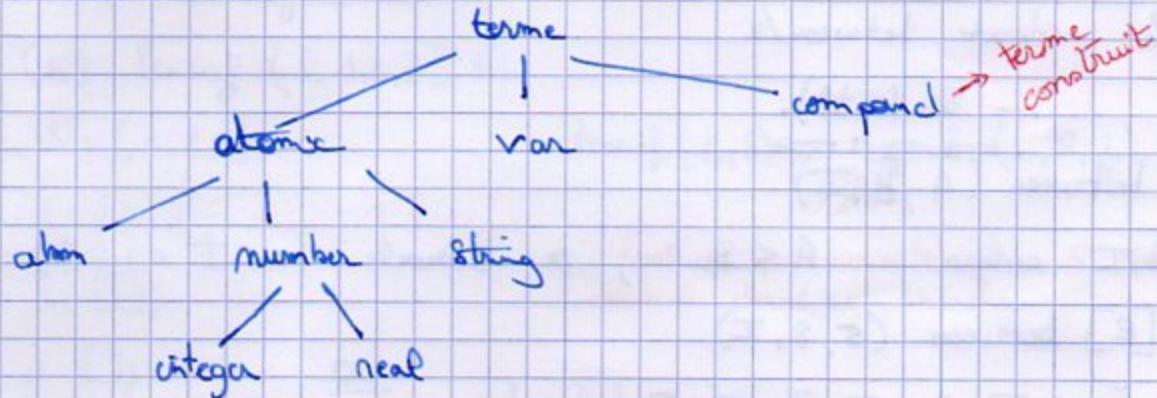
### ③ en PROLOG

leaf(A, [A]) :- feuille(A);

leaf(T, Lf) :- =..(T, [-1FL]), \= (fib, [ ]), leaflist(fib, Lf).

① Des prédicts pour tester la nature des termes

cette classification fournit les noms des prédicts d'arité 1



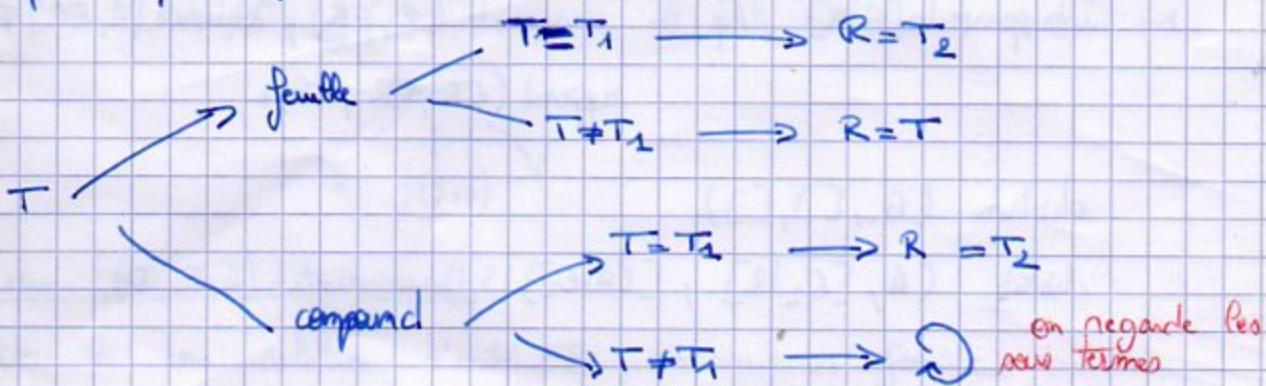
nonvar(X) : "X est atomique ou X est composé"

nongrounded(X) : X contient au moins 1 variable libre.

② subst( $\overrightarrow{T}, \overrightarrow{T_1}, \overrightarrow{T_2}, \overrightarrow{R}$ )

$T, T_1, T_2, R$  sont des termes.

$R$  est le terme  $T$  où toutes les occurrences de  $T_1$  ont été remplacées par  $T_2$ .



subst ( $T, T_1, T_2, R$ ) :-  $\text{famille}(T), T = T_1, R = T_2.$   
 subst ( $T, T_1, T_2, R$ ) :-  $\text{famille}(T), T \setminus= T_1, R = T.$   
 subst ( $T, T_1, T_2, R$ ) :-  $\text{compound}(T), T \setminus= T_1, R = T_2.$   
 subst ( $T, T_1, T_2, R$ ) :-  $\text{compound}(T), T \setminus= T_1, T = ...[F | Lt],$   
 subst ( $L, T_1, T_2, R$ ),  $R = ...[F | R_L].$

itsubst ([ ], -, -, [ ]).

itsubst ( $[Ta | RT], T_1, T_2, [Ra | ST]$ ) :- subst ( $Ta, T_1, T_2, Ra$ ),  
 itsubst ( $RT, T_1, T_2, ST$ ).

$\equiv$  subst ( $T, T, R, R$ )

subst ( $T, T_1, T_2, R$ ) :-  $\text{famille}(T), T \setminus= T_1, R = T$

③ le prédicat between/3.

:- lib(util).

between ( $\overrightarrow{A}, \overrightarrow{B}, \overrightarrow{I}$ )

$A, B, I$  entiers       $A < B$ . , px démontrée par  $I \in [A, B]$

$[E]$  between ( $5, 8, I$ )

$I=6 ; I=7 ; I=5;$

en part

$\overrightarrow{(E, E_p)}$

ensembles modélisés par des listes.

$[E]$  en part ( $[3, 1, 2], E_p$ ).

$E_p [[], [1], [2], [3], [3, 1], [3, 2], [1, 2], [1, 2, 3]].$

en part ( $[], [C]$ ).

en part ( $[A | R], E_p$ ) :- en part ( $R, ER$ ), distrib ( $A, ER, FR$ ),  
 append ( $ER, FR, EP$ ).

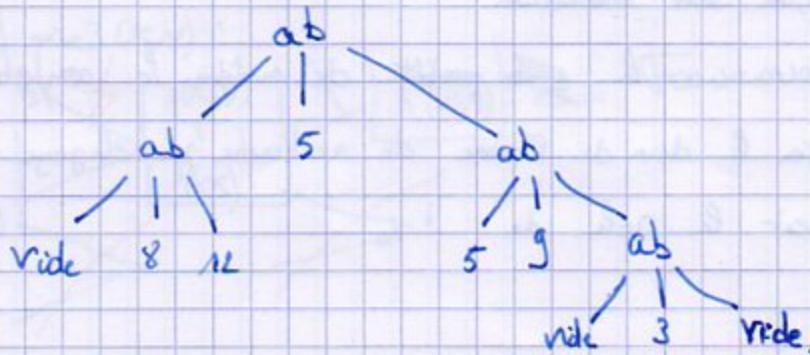
distrib ( $A, [ ], [ ]$ ).

distrib ( $A, [E_1 | R], [C | E_2 | S]$ ) :- distrib ( $A, R, S$ ).

⑤ retour sur les arbres.

Ab

$X \infty \text{entier}$ $X \in Ab$	$\text{vide} \in Ab$	$\forall i \ N_i \in \text{entier}$ $G_i \in Ab$ $D_i \in Ab$ $ab(G_i, N_i, D_i) \in Ab$
--	----------------------	---



haut ( $\overleftarrow{A}, \overrightarrow{H}$ )  $A \in Ab$   $H \in \text{entier}$ .

haut ( $\text{vide}, 0$ ): - integer ( $A$ ):

haut ( $\text{vide}, 0$ ).

haut ( $ab(G, H, D), H$ ): - haut ( $G, H_g$ ), haut ( $D, H_d$ ),  $H = (H_g, \max(H_g, H_d) + 1)$ .

P:  $\forall x \in G, X \leq N$

$\forall x \in D, X > N$

faitab ( $\overleftarrow{L}, \overrightarrow{A}$ )

L: liste d'entiers /  $A \in Abs$  / A contient les

entiers de L et rien d'autre.

faitab ([], vide).

faitab ([A|B], 8): - faitab (R, S), metdans (A, S, B)

metdans (A, vide, A).

— (A, ab(G, N, D), ab(GA, N, D)): -  $A \leq N$ , metdans (A, G, GA).

— (A, ab(G, N, DA)): -  $A > N$ , metdans (A, D, DA).

— (A, X, ab(A, X, vide)): - integer (X),  $A \leq X$ .

— (vide, X, A)): - integer (X),  $A > X$ .

tri ( $\overleftarrow{L_1}, \overrightarrow{L_2}$ )

tri ( $L_1, L_2$ ): - faitab ( $L_2, A$ ), pdif (A, L<sub>2</sub>).

lpdf(A, [A]) :- integer(A).

lpdf(vide, []).

lpdf(ab(G, M, D), L) :- lpdf(G, Lg), lpdf(D, Ld), append(Lg, [M|Ld], L).

La coupure (ou suppression des choix) !/0

à utiliser avec modération

Le programmeur Prolog a la ~~possibilité~~ de modifier le comportement de l'interpréteur (ie modifier le déroulé de l'arbre de recherche par claquage dynamique) c'est le rôle de !/0

A) Semantique "opérationnelle" de la coupure.

Quand ! arrive au sommet de pile des buts.

- il renvoie, et

- tous les choix en attente depuis l'instant où ! est entré dans la pile sont supprimés

Ces choix concernent:

- les autres règles du paquet contenant la coupure

- les autres façons de démontrer les prédicats introduits

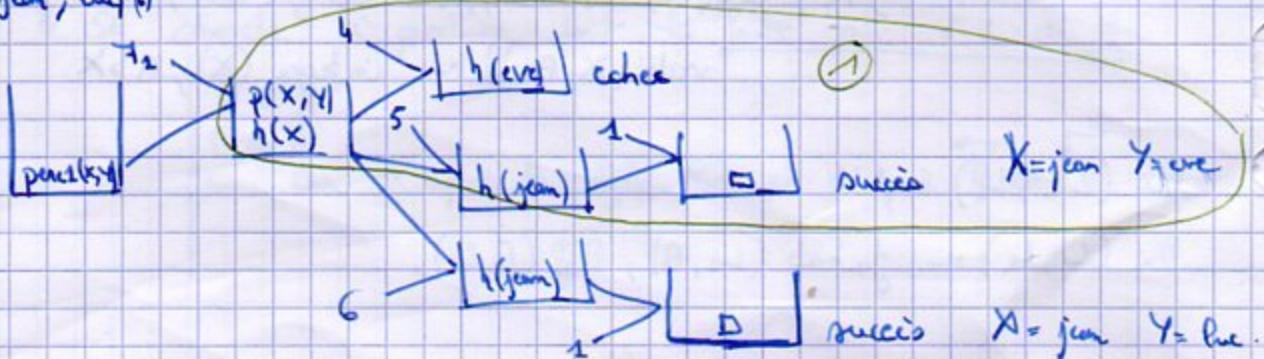
Remarque : l'effet de ce prédicat n'est pas exprimable en logique

→ donc difficile à utiliser

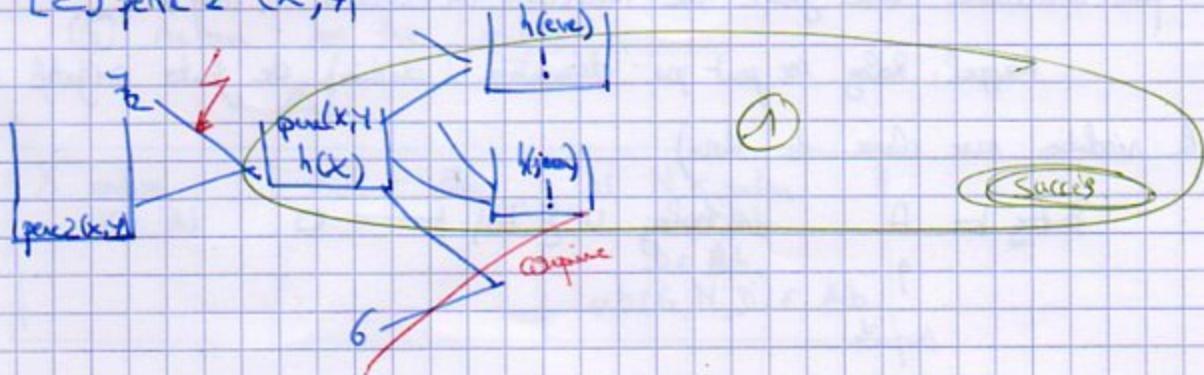
Exemple :

P	p(jean)	(1)	perel(X,Y) :- p(X,Y), h(X). (2)
	h(luc)	(1)	----- 2 -----
	g(eve)	(2)	----- 3 -----
	p(eve, luc)	(4)	, !, h(x). !. (2)
	p(jean, eve)	(5)	(3)
	p(jean, luc)	(6)	

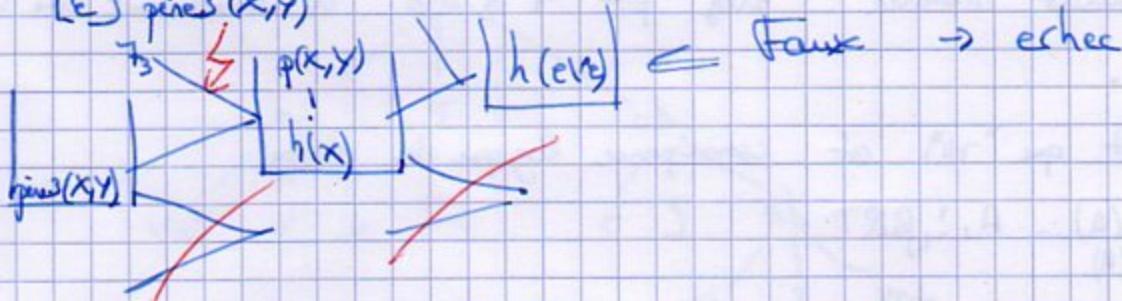
[E] perel(X,Y).



[E] gène 2 (x,y)



[E] père3 (x,y)



B) Comment utiliser la coupe ?

a) pour améliorer l'efficacité des programmes (et du programmation).

La coupe permet alors à élaguer des branches de l'arbre de recherche qui, de toutes façons, auraient abouti à 1 échec (sans la coupe).

Une telle coupe ne modifie pas la logique de prog.

Quand les règles d'un prédictat sont naturellement exclusives, on peut utiliser la coupe sans danger.

mais !

- l'échec des règles dans le paquet est important.

- on peut prendre l'avantage de la reversibilité.

Exemple: elind([],[]):- !.

elind ([A|R],S):- membre(A,R), !, elind(R,S).

elind ([A|R],[A|S]) :- elind(R,S).

Rmq: Dans ce cas d'utilisation de la coupe on est très proche de la conditionnelle généralisée

cas:  $L \rightarrow [] \rightarrow A_1$

$L = [A|R]$  et  $A \in R \rightarrow A_2$

Autrement  $\rightarrow A_3$ .

cas:

↳ pour introduire une forme de négation en PROLOG ('not')

Rappel: Prolog ne peut pas démontrer (découvrir) de faits négatifs (inherents à la résolution avec clause de Horn)

$$\text{Prolog} \vdash A \quad \text{Prolog} \cup \{\neg A\} \vdash \square$$

?

Négation

Mais comment montrer  $\text{Prog} \models \neg A$  ? en considérant une négation spéciale.

On dit que  $\neg A$  est conséquence logique de Prolog.

$\text{not}(A) :- A, !, \text{fail}.$

$\text{not}(A).$

TP: Prototype de SE.

a) Qu'est-ce qu'un SE ?

S.E. = logiciel (algo + données) capable de raisonner

(pratiques d'inferences) sur un domaine réel

partie dynamique + partie statique

moteur d'inference + connaissance  
informatique

Expert | Règles d'inference (propres au domaine)

utilisateur | Faits observés

b) Comment ça fonctionne.

- chaînage avant (TP)  
arrière (Prolog)  
mixte.

c) Vers la logique

Faits observés  $\rightarrow H$  (hypothèse)

Règles de l'expert  $\rightarrow$  règle d'inference.

Le SE cherche à produire tout les faits qui CL de  $H$

$A \in H$   
 $B \in H$   
 $C$  car d'après la règle  $R_2$  et d'après  $A \wedge B$

d) Le TP.

4 faits observés

Tin nononne
Tin familier
Tin vit maison
Tin père de felix.

4 règles

x nononne	$\xrightarrow{R_1}$	x est un chat
x familier	$\xrightarrow{R_2}$	x domestique
x vit maison		x adulte

x est un chat  $\xrightarrow{R_3}$  x mammifère  
 x griffe

x père de y.  
 x propriété p.

e) représentation

fait observés  $\rightarrow$  terme

une + +  $\rightarrow$  liste de termes.

règle d'inférence  $\rightarrow$  règle Prolog

une + "  $\rightarrow$  paquet prolog

Prolog pour écrire des analyseurs syntaxiques!

Nous savons que l'analyse syntaxique se mette "au dessus" de l'analyse lexicale. D'où l'hypothèse de TD, TP.

Nous supposons l'existence d'un analyseur lexical qui consomme un flot de caractères frappés au clavier qui produit une boîte de lettres.

read\_token.

read\_token(X,T)

"unifie X au prochain lexème du flot d'entrée"

" $\rightarrow$  T à la catégorie lexicale du prochain \_\_\_\_\_"

[E] read\_token(A,B).

fatô

A = fatô, B = chom

Définir analyse/2  
analyse(Lx, Lt)

"Unifier Lx à la liste des lexèmes du flist d'entrée "

" — Lt à la catégorie lexicale des lexèmes

" le point ". " marque la fin du flist d'entrée

fullstop

~~analyse([ ], [ ]) :- read\_token(':' , fullstop).~~

analyse([A|X], [B|X]) :- read\_token(A, B), A = '!' , !,  
analyse(X, Y).

analyse([C], [ ])

raison le caractère est consommé  
par le premier read-t.

Méthode 1g ( hésitamment la première, facile à comprendre, inefficace)

on part de la grammaire ( $G = (S, V_n, V_t, P)$ )

• On définit un prédictat  $z_A$  pour chaque élément  $X \in V_t$

$z(C)$

" C est une liste de lexème, dérivable à partir de X  
 $(X \Rightarrow C)$

dans le paquet z on définit une clause pour production de sujet

$\forall$  dans P :  $\nexists (X \Rightarrow P) \in P$  on écrit  $z(C) :- P$

or  $P = w_1. \dots. w_n$

$w_i \in V_n \cup V_t$

d'où  $z(C) :- conc_n(O_1, O_2, \dots, O_n, C)$

$O_i$  dérivable d'après  $w_i$

exemple  $S \rightarrow a S b | C$

$z(C) :- conc_3(O_1, O_2, O_3, C)$ ,  $O_1 = [a]$ ,  $z(O_2)$ ,  $O_3 = [b]$

$z(C) :- C = [c]$ .

Exemple:  $C \rightarrow E \quad 0a \quad T \mid T$

$T \rightarrow T \quad 0a \quad F \mid F$

$F \rightarrow \text{atome} \mid (E)$

$O_a \rightarrow +1$

$O_m \rightarrow *$

$e(C) :- conc(E, [O_a], T, C), e(E), o_a(O_a), t(T)$

$e(C) :- E(C)$

$t(C) :- conc(T, [O_m], F, C), t(T), o_m(O_m), f(F).$

$f([C]) :- atom(C)$

$f(['(' | R]) :- conc(E, ['')'], R), c(E).$

$O_a([T])$

$O_a([-])$

$O_m([C])$

Prélog pour écrire des analyseurs syntaxiques

① analyse lexicale

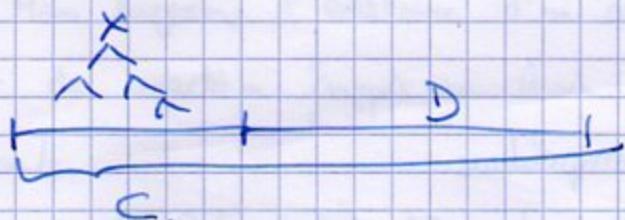
② méthode 1: simple et totalement inefficace.

③ méthode 2: + complexe, mais efficace

Méthode 2: on part de  $G = (S, V_n, V_t, P)$

① Pour chaque mot terminal  $X \in V_t$  on définit un prédicat  $x_2$ .

$x(C, D)$  " au début de la chaîne (l'objet de l'exemple) C, on rencontre une sous chaîne dérivable depuis X. Il reste à analyser ensuite cette sous chaîne D."



② dans le paquet de classes de nom x on a autant de classes qu'il y a de productions de sujet X dans P

$x(C, D)$

Exemple:  $S \rightarrow a S b | c$

$s([a | X_1], Y) :- s(X_1, [b | Y]).$

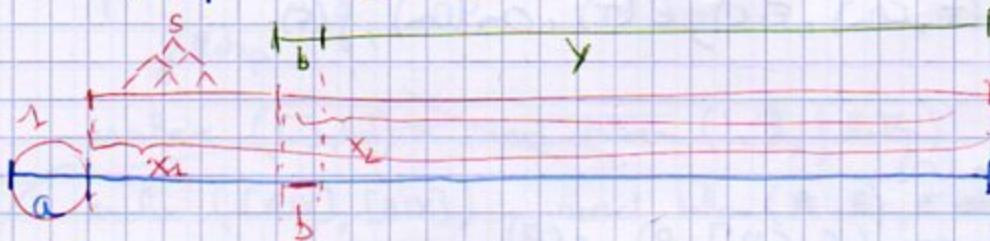
$\Delta([c|Y], Y)$ .

appel:  $[E]$  analyse  $(A, B)$ ,  $\Delta(A, [ ])$ .

version expérimentale

$S(X, Y) :- \overset{1}{X = [a|X_1]}, \overset{2}{\sim(X_1, X_2)}, \overset{3}{X_2 = [b|Y]}$ .

$S(X, Y) :- X = [c|Y]$



⚠ méthode  $\geq 2$  ne supporte pas les GI récursives à Gauche (ex:  $A \rightarrow AaB$ ).

Exercice:  $E \rightarrow T$  "ou"  $E \mid T$

$T \rightarrow L$  "et"  $T \mid L$

$L \rightarrow F$  | "non"  $F$

$F \rightarrow 0 \mid 1 \mid (E)$ .

$e(X, Y) :- t(X, X_2), X_2 = [an|X_3], c(X_3, Y)$ .

$e(X, Y) :- t(X, Y)$ .

$t(X, Y) :- \mathbf{t}(X, X_2), X_2 = [ct|X_3], t(X_3, Y)$ .

$t(X, Y) :- f(X, Y)$ .

$f(X, Y) :- f(X, Y)$

$f(X, Y) :- X = [mon|X_2], f(X_2, Y)$ .

$f(X, Y) :- X = [';|X_2], e(X_2, X_2), X_2 = [';]Y$ .

$f([0|Y], Y)$ .

$f([1|Y], Y)$ .

Prolog pour écrire des compilateurs.

mettre en œuvre des GA. (bcp + facile qu'avec YACC)

Méthode : ① écrire l'analyseur syntaxique (méthode ②) à partir de la grammaire syntaxique

② tester

③ écrire la mire en œuvre du compilateur à partir de la grammaire attribuée

③ + en détail. Si un N-term X possède n attributs non prédictif.  
d'analyse x aura  $M+2$  arguments

$$x(C, D, a_1, a_2, \dots, a_n)$$

Exemple

des phrase du langage d'entrée sont de la forme  
mot1 / mot2 / texte

où mot de la forme lettric +

où texte est de la forme mot { $\sqcup$  mot} $^*$

le code à produire est un texte issue de texte dans lequel

les occurrences du mot1 ont été remplacée par mot2.

## ② Grammaire

$$S \rightarrow M \mid M \mid T \quad ①$$

$$T \rightarrow M \quad ②$$

$$T \rightarrow M \sqcup T \quad ③$$

$$M \rightarrow L \quad ④$$

$$M \rightarrow LM_2 \quad ⑤$$

① Ecrire la GA qui permet de produire texte'

①.1: réfléchir aux attributs à attacher aux non-terminals.

S, T, M.

- pour M un attribut mot : liste des lettres du mot reconnu par M

- pour T → un attribut texte : liste de mots où les occurrences de mots ont été remplacés par mot<sub>2</sub>.

→ mot<sub>1</sub> : mot à remplacer hôte

→ mot<sub>2</sub> : — de remplacement

②.2 écrire la grammaire attribuée qui permet de produire texte'

$$① \quad S \rightarrow M_1 \mid M_2 \mid T$$

$$\begin{aligned} S.\text{texte} &= T.\text{texte} \\ T.\text{mot}_1 &= M_1.\text{mot} \\ T.\text{mot}_2 &= M_2.\text{mot}. \end{aligned}$$

$$⑤ \quad M_1.\text{mot} = [L \mid M_2.\text{mot}]$$

$$④ \quad M.\text{mot} = [L]$$

$$③ \quad T_2.\text{mot}_1 = T_1.\text{mot}_1$$

$$T_2.\text{mot}_2 = T_1.\text{mot}_2$$

~~T<sub>1</sub>.texte = f(mot<sub>1</sub>, mot<sub>2</sub>)~~

$$T_1.\text{Texte} = \left[ \begin{array}{c|c} \text{a: Matchet} = T_1.\text{motor} & T_2.\text{Texte} \\ \text{a: T_1.matchet} & \\ \text{a: M.net} & \end{array} \right]$$

$$\textcircled{2} \quad T.\text{Texte} = \left[ \begin{array}{c} \leftarrow \end{array} \right]$$

in Prolog

$S \rightarrow M / H / T$

$s(X, Y, \text{Texte}) :- m(X, [ / | X_2 ], \text{Motor}),$   
 $m(X_1, [ / | X_2 ], \text{Matchet}),$   
 $t(X_2, Y, \text{Motor}, \text{Matchet}, \text{Texte})$

$m([L | X], Y, [L | Mot]) :- \text{fettne}(L), m(X, Y, Mot)$

$m([L | Y], Y, [L]) :- \text{---}.$

$t(X, Y, Ma, Mn, [Ma | T]) :- m(X, [u | X_2 ], Ma),$   
 $t(X_2, Y, Ma, Mn, T).$

$t(X, Y, Ma, Mn, [M | T]) :- m(X, [u | X_2 ], M), M \neq Ma,$   
 $t(X_2, Y, Ma, Mn, T).$