Document - Introduction au Prolog

Cours ≪ Document et Web Sémantique ≫

Nicolas Delestre

Plan

Introduction 1 / 2

Prolog

- Paradigme de PROgrammation LOGique
- Langage inventé par Alain Colmerauer et Philippe Roussel vers 1972 (Marseille)
- ≪ [...]le but n'était pas de faire un langage de programmation mais de traiter les langages naturels, en l'occurrence le Français. ≫

Fondement théorique

- Basé sur le calcul des prédicats du premier ordre (plus exactement les clauses de Horn)
- Les concepts fondamentaux sont l'unification, la récursivité et le backtracking

Introduction 2 / 2

Principe

- Prolog permet au programmeur d'expliciter des faits, des règles et de répondre à des questions en infèrant sur ces faits et règles

Éléments de base du langage

- Les clauses de Horn :
  - $r_1 \wedge r_2 \wedge \cdots \wedge r_n \rightarrow h$ avec $n \in \mathbb{N}$
  - En prolog, elles peuvent représenter aussi bien des faits que des règles
- Les atomes (commencent par une minuscule, ou entre simples côtes si utilisation de l'espace) et les variables (commencent par une majuscule)
- Une question

- Le prolog n'est pas standardisé (la syntaxe peut varier)
- Tous les exemples de ce cours sont basés sur *swi-prolog*

Syntaxe 1 / 3

Syntaxe

- Fait :

```
fait(atome1,atome2,...,atome3).
```

- Règle

```
regle(Var01, Var02, ...):-
                cond1(Var11, Var12, ...),
                cond2(Var21,Var22,..),
                ...,
                condn(Varn1,Varn2,...).
```

Remarques

- Un prédicat est un ensemble de règles ou de faits ayant le même nom et la même arité
- Deux prédicats peuvent avoir le même nom (mais pas la même arité)

Syntaxe 2 / 3

Exemple

```
1 /* homme(X) signifie que X est un homme*/
2 homme(patrick).
3 homme(gerard).
4 homme(louis).
5 homme(pierre).
6
7 /* femme(X) signifie que X est une femme*/
8 femme(therese).
9 femme(sandrine).
10 femme(muriel).
11 femme(germaine).
12 femme(yvette).
13
14 /* enfant(E,PM1,PM2) signifie que E est un enfant de PM1 et PM2*/
15 enfant(gerard,germaine,louis).
16 enfant(therese,yvette,pierre).
17 enfant(patrick,gerard,therese).
18 enfant(muriel,gerard,therese).
19 enfant(sandrine,gerard,therese).
20 enfant(astride,jean,therese).
```


Syntaxe 3 / 3

```
22 /* enfant(E,PM) siginifie que E est un enfant de PM*/
23 enfant(E,PM) :- enfant(E,PM,_).
24 enfant(E,PM) :- enfant(E,_,PM).
25
26 /* pere(P,E) signifie que P est pere de E*/
27 pere(P,E) :- homme(P), enfant(E,P).
28
29 /* mere(P,E) signifie que P est mere de E*/
30 mere(P,E) :- femme(P), enfant(E,P).
31
32 /* grandpere(GP,E) signifie que GP est grand-pere de E*/
33 grandpere(GP,E) :- pere(GP,P), pere(P,E).
34 grandpere(GP,E) :- pere(GP,P), mere(P,E).
35
36 /* grandmere(GP,E) signifie que GP est grand-mere de E*/
37 grandmere(GP,E) :- mere(GP,P), pere(P,E).
38 grandmere(GP,E) :- mere(GP,P), mere(P,E).
```


L'interpréteur de swi-prolog 1 / 3

- On lance l'interpréteur à l'aide de la commande *swipl* (ou *pl*)
- On pose alors des ≪ questions ≫ en invoquant un prédicat suivi d'un point
- Un programme prolog est stocké dans un fichier dont le nom commence par une minuscule et a l'extension `.pl` :
  - Les prédicats *consult/1* ou *load_files/2* permettent charger un programme (le nom du programme entre crochets est un raccourci de *consult*). On ne spécifie pas l'extension.
  - Un programme ne contient que des faits et des règles. Pour qu'un programme pose une question, on préfixe cette question de `:-`, par exemple pour charge le module toto :
    `:- use_module(library(toto))`
- Quelques prédicats particuliers :
  - *halt/0* pour quitter
  - *help/1* pour obtenir de l'aide
  - *assert/1* pour ajouter un fait ou une règle depuis l'interpréteur

L'interpréteur de swi-prolog 2 / 3

- Lorsque plusieurs solutions existent, l'interpréteur en affiche une et demande ce qu'il doit faire pour le reste
  - ? pour plus d'information
  - ; ou espace pour la solution suivante
  - a ou entrée pour arrêter l'affichage des solutions

```
delestre@delestre-portable:~/MesDocuments/Cours/ASI/Document/Prolog/Version1.0/
     exemples$ pl
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.54)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [genealogie].
% genealogie compiled 0.01 sec, 4,532 bytes
true.

2 ?- pere(gerard,patrick).
true .

3 ?- pere(therese,patrick).
fail.
```


L'interpréteur de swi-prolog 3 / 3

```
4 ?- pere(X,patrick).
X = gerard ;
fail.

5 ?- pere(gerard,X).
X = patrick ;
X = muriel ;
X = sandrine ;
fail.

6 ?- grandpere(louis,patrick).
true .

7 ?- grandpere(germaine,patrick).
false.

8 ?- grandpere(X,patrick).
X = louis ;
X = pierre ;
false.

9 ?- grandpere(louis,X).
X = patrick ;
X = muriel ;
X = sandrine ;
false.
```

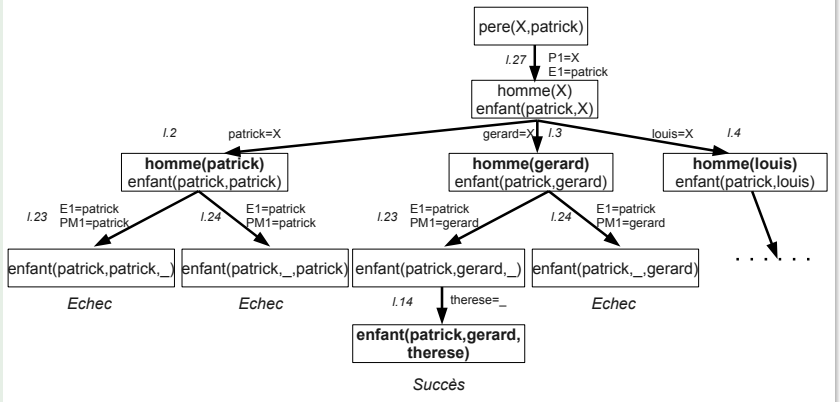

Fonctionnement du moteur 1 / 3

Comment fonctionne le moteur prolog ?

- Le moteur prolog démontre une question :
  - Unification : « procédé par lequel on essaie de rendre deux formules identiques en donnant des valeurs aux variables qu'elles contiennent » [Bel94]
  - Pas de démonstration : Le fait de réussir à faire une unification. Un pas de démonstration entraîne la démonstration d'une ou plusieurs questions
  - Point de choix : le fait d'avoir plusieurs unifications possibles au niveau d'un pas de démonstration
  - Backtracking : le fait de ne plus avoir à/pouvoir faire d'unification. Le moteur prolog remonte jusqu'au dernier point de choix et applique l'unification suivante

Fonctionnement du moteur 2 / 3

pere(X,patrick).



Fonctionnement du moteur 3 / 3

trace,pere(X,patrick).

```
6 ?- trace,pere(X,patrick).
   Call: (8) pere(_G314, patrick) ? creep
   Call: (9) homme(_G314) ? creep
   Exit: (9) homme(patrick) ? creep
   Call: (9) enfant(patrick, patrick) ? creep
   Call: (10) enfant(patrick, patrick, _L237) ? creep
   Fail: (10) enfant(patrick, patrick, _L237) ? creep
   Redo: (9) enfant(patrick, patrick) ? creep
   Call: (10) enfant(patrick, _L229, patrick) ? creep
   Fail: (10) enfant(patrick, _L229, patrick) ? creep
   Redo: (9) homme(_G314) ? creep
   Exit: (9) homme(gerard) ? creep
   Call: (9) enfant(patrick, gerard) ? creep
   Call: (10) enfant(patrick, gerard, _L237) ? creep
   Exit: (10) enfant(patrick, gerard, therese) ? creep
   Exit: (9) enfant(patrick, gerard) ? creep
   Exit: (8) pere(gerard, patrick) ? creep
X = gerard .
```


Les structures de données

- Prolog propose deux structures de données :
  - les arbres
  - les listes

Arbre 1 / 4

- La syntaxe est identique à celle des prédicats (ne pas confondre)
- Les arbres sont nommés (ils peuvent avoir le même nom qu'un prédicat)
- Les arbres permettent de simplifier l'écriture de fait

Arbre 2 / 4

Exemple (inspiré de [Bel94])

- Comment exprimer :
  - Annie possède une voiture qui est une ford escort
  - Jérome possède une voiture qui est une renault twingo
  - Jérome possède une console de jeu playstation 3
  - Jérome possède une console de jeu Xbox 360
  - Hélène possède une renault mégane
  - Hélène possède une console de jeu game boy

Avec uniquement des prédicats

- *possede/2*

```
possede(annie,voiture_annie).
possede(jerome,voiture_jerome).
...
```

- *marque/2*

```
marque(voiture_annie,ford).
...
```

Arbre 3 / 4

Avec des arbres (possede.pl)

```
possede(annie,voiture(ford,escort)).
possede(jerome,console(playstation,3)).
possede(jerome,console(xbox,360)).
possede(jerome,voiture(renault,twingo)).
possede(helene,console(nitendo,game_boy)).
possede(helene,voiture(renault,megane)).
```

Exemple d'interaction

```
1 ?- [possede].
% possede compiled 0.00 sec, 1,676 bytes
true.

2 ?- possede(X,voiture(_,_)).
X = annie ;
X = jerome ;
X = helene.

3 ?- possede(X,voiture(renault,_)).
X = jerome ;
X = helene.
```


Arbre 4 / 4

Remarque

- Sans les arbres, le code ressemble a celui d'une base de données relationnelle (utilisation de clé et de jointure)
- Prolog permet d'avoir à la fois du relationnel et du hiérarchique

Liste

- Les éléments d'une liste sont entourés des crochets et séparés par une virgule
- Les éléments d'une liste peuvent être des atomes, des nombres, des chaînes de caractères, des arbres ou des listes
- La liste vide est notée [ ]
- Une liste non vide peut être considérée comme étant composée d'un élément E suivi d'une liste L : $[E|L]$

Exemples

```
1 ?- [a,b,c]=[a|[b,c]].
true.

2 ?- [1,a,b]=[1|[a,b]].
true.

3 ?- [1]=[1|[]].
true.

4 ?-
```


Quelques algorithmes sur les listes 1 / 3

membre/2

```
/* membre(E,L) est vrai si E est un element de L*/
membre(E,[E|_]).
membre(E,[E1|L]) :- membre(E,L),E\=E1.
```

Exemple de questions

```
21 ?- membre(b,[a,b,c]).
true .

22 ?- membre(X,[a,b,c]).
X = a ;
X = b ;
X = c ;
false.

23 ?- membre(a,[a,b,a,c]).
true;
false.

24 ?- membre(a,L).
L = [a|_G326] ;
ERROR: Out of local stack
```

Quelques algorithmes sur les listes 2 / 3

supprimer/3

```
/* supprimer(E,L1,L2) est vrai lorsque L2 possede tous les
    elements de L1 sauf E */
supprimer(_,[],[]).
supprimer(E,[E|L],L1) :- supprimer(E,L,L1).
supprimer(E,[E1|L],[E1|L1]) :- supprimer(E,L,L1),E\=E1.
```

Exemple de questions

```
24 ?- supprimer(1,[1,2,1,3],L).
L = [2, 3] .

25 ?- supprimer(X,[1,2,1,3],[2,3]).
X = 1 ;
false.

?- supprimer(2,L,[1,3]).
ERROR: Out of local stack
   Exception: (220,267) supprimer(2, _G660795, [1, 3]) ?
```


Quelques algorithmes sur les listes 3 / 3

concatener/3

```
/* concatener(L1,L2,L3) est vrai lorsque L3 est la concatenation
    de L1 et L2 */
concatener([],L,L).
concatener(L,[],L).
concatener([E1|L1],L2,[E1|L3]) :- concatener(L1,L2,L3).
```

Exemple de questions

```
30 ?- concatener([1,2],[3,4],L).
L = [1, 2, 3, 4] .

31 ?- concatener(L,[3,4],[1,2,3,4]).
L = [1, 2] .

32 ?- concatener([1,2],L,[1,2,3,4]).
L = [3, 4] .

33 ?-
```


Operateurs 1 / 3

- Le prédicat *op/3* permet de créer un opérateur à partir d'un prédicat, tel que :
  - le premier paramètre indique la priorité de cet opérateur (entier compris entre 0 et 1200),
  - le deuxième paramètre, un atome, précise l'arité, la notation (préfixe, infixe, suffixe) et l'associativité : fx, fy, xfx, xfy, yfx, xf, yf
  - le troisième paramètre est le prédicat qui définit le comportement de l'opérateur

extrait de `liste.pl`

```
/* membre(E,L) est vrai si E est un element de L*/
membre(E,[E|_]).
membre(E,[E1|L]) :- membre(E,L),E\=E1.

:- op(500,xfx,membre).
```


Operateurs 2 / 3

utilisation de `liste.pl`

```
$ swipl
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [liste].
true.

?- membre(2,[1,2,3]).
true ;
false.

?- 2 membre [1,2,3].
true ;
false.

?- X membre [1,2,3].
X = 1 ;
X = 2 ;
X = 3 ;
false.
```

Operateurs 3 / 3

=/2

- Cet opérateur tente d'unifier l'opérande gauche à l'opérande droit. Il est vrai lorsque l'unification a réussi, faux sinon.

\=/2

- Cet opérateur tente d'unifier l'opérande gauche à l'opérande droit. Il est faux lorsque l'unification a réussi, vrai sinon.

Opérateurs de comparaison

- ==/2, \==/2, @</2, @<=/2, @>/2, @>=/2
- Les nombres sont comparés suivant leurs valeurs, les chaînes de caractères et les atomes suivant l'ordre lexicographique, les variables suivant leurs adresses, les prédicats ou les structures suivant leurs arités puis leurs noms.
- variable < nombre < chaîne de caractères < atome < prédicat et structure

Arithmétique 1 / 3

Expression arithmétique

- Prolog permet de noter les expressions arithmétiques avec l'utilisation d'opérateurs en notation infixe
- Les expressions arithmétiques sont des arbres qui ne sont pas évalués

Exemple

```
33 ?- 2+3=3+2.
false.

34 ?- 3+2=3+2.
true.

35 ?-
```


Arithmétique 2 / 3

L'opérateur *is*

- L'opérateur binaire *is* permet d'unifier l'opérande gauche avec l'opérande droite qui doit être une expression arithmétique. L'opérande droite est évaluée avant l'unification

Exemple

```
35 ?- 5 is 2+3.
true.

36 ?- 5 is 3+2.
true.

37 ?- X is 1+2+3.
X = 6.

38 ?-
```

Attention

- Toutes les variables que peut contenir l'opérande droite doivent être unifiées

Arithmétique 3 / 3

Les opérateurs de comparaison

- Prolog propose 6 opérateurs binaires de comparaison d'expression arithmétique. En fait il évalue les deux opérandes puis les unifie (il faut donc que toutes les variables des opérandes soient déjà unifiées)
  - =:=, =\=, <, >, =<, >=

inserer/3

```
inserer(E,[],[E]).
inserer(E,[A|L1],[E,A|L1]) :- E=<A.
inserer(E,[A|L1],[A|L2]) :- E>A, inserer(E,L1,L2).
```

```
44 ?- inserer(2,[1,3],L).
L = [1, 2, 3] .

45 ?-
```

Cut ou coupe de choix 1 / 3

Qu'est ce que le *cut* ?

- Les trois règles définissant le prédicat *inserer/2* sont obligatoirement disjointes
  - La validité d'une des règles exclut les deux autres
- Le prédicat *!/0* (le *cut*) sert à indiquer au Prolog que si la démonstration en cours est valide, alors tous les unifications issues des points de choix précédents sont à supprimer (plus de backtracking)

Attention

- Attention le *cut* peut changer la sémantique d'un prédicat (par exemple ne donner qu'une seule réponse alors qu'il y en a plusieurs sans le *cut*)
  - Si la sémantique du prédicat est inchangée, il s'agit d'un *cut* vert
  - Sinon c'est un *cut* rouge

Cut ou coupe de choix 2 / 3

membre/2

```
/* membre(E,L) est vrai si E est un element de L*/
membre(E,[E|_]) :- !.
membre(E,[E1|L]) :- membre(E,L).
```

```
2 ?- membre(a,[a,b,a,c]).
true.

3 ?- membre(X,[a,b,c]).
X = a.

4 ?-
```

cut rouge ou vert ?

Cut ou coupe de choix 3 / 3

concatener/3 (cut vert)

```
/* concatener(L1,L2,L3) signifie que L3 est la concatenation de
    L1 et L2 */
concatener([],L,L) :- !.
concatener(L,[],L) :- !.
concatener([E1|L1],L2,[E1|L3]) :- concatener(L1,L2,L3).
```

inserer/3 (cut vert)

```
/* inserer(E,L1,L2) signifie que L2 est le resultat de l'
    insertion (ordre croissant) de E dans L1 */
inserer(E,[],[E]) :- !.
inserer(E,[A|L1],[E,A|L1]) :- E=<A,!.
inserer(E,[A|L1],[A|L2]) :- inserer(E,L1,L2).
```


Négation

- Si F est une formule, la négation de cette formule est notée $not(F)$
- Prolog pratique la négation par l'échec : pour démontrer $not(F)$ il essaie de démontrer F. Si c'est possible alors Prolog conclut $not(F)$ n'est pas démontrable. Sinon il considère $not(F)$ comme démontré
- Pour ne pas avoir de problème il est conseillé d'utiliser *not* avec des variables préalablement unifiées

Exemple de problème

```
10 ?- assert(p(a)).
true.

11 ?- X=b,not(p(X)).
X = b.

12 ?- not(p(X)),X=b.
false.

13 ?-
```

Différence

- La différence est le contraire de l'unification
- Elle est représentée par l'opérateur $\backslash =$
- Elle peut être définie à l'aide du prédicat *not/1* :

```
X \= Y:- not(X=Y).
```


Étendons l'exemple du début 1 / 10

- Développez le prédicat *frere/2* qui permet de savoir si une personne est le frere d'une autre personne
  - *frere*(P, FS) signifie que P est un frere de FS
- Méthode :
  - *frere*(P, FS) :
    - vérifier que P est un homme
    - récupérer tous les frères ou soeurs (*freresOuSoeurs/2*) de P(obtention de $L1$)
    - vérifier que FS appartient à $L1$
  - *freresOuSoeurs*(E, FS) :
    - récupérer le père (*pere/2*) P de E
    - récupérer les enfants (*enfants/2*) $L1$ de P
    - faire de même avec la mère (obtention de $L2$)
    - concatèner (*concatener/3*) $L1$ et $L2$ (obtention de $L3$)
    - supprimer les doublons de $L3$ (obtention de $L4$)
    - supprimer (*supprimer/2*) E de $L4$ (obtention de FS)

Étendons l'exemple du début 2 / 10

frere/2

```
/* frere(P,FS) siginifie que P est un frere de FS*/
frere(P,FS) :- homme(P), freresOuSoeurs(P,L), membre(FS,L).
```

freresOuSoeurs/2

```
/* freresOuSoeurs(E,L) siginifie que L contient les freres ou soeurs de E*/
freresOuSoeurs(E,L) :-
        pere(P,E), mere(M,E),
        enfants(P,L1), enfants(M,L2),
        concatener(L1,L2,L3), supprimer_doublon(L3,L4), supprimer(E,L4,L).
```


Étendons l'exemple du début 3 / 10

enfants/2

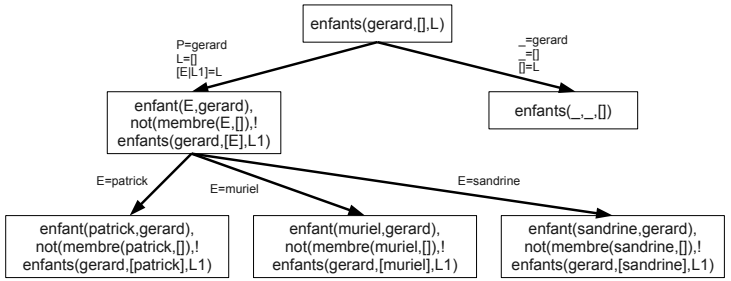
```
/* enfants(P,L) signifie que L est la liste des enfants de P*/
enfants(P,L) :- enfants(P,[],L).
```

enfants/3

```
/* enfants(P,L1,L2) signifie que L2 est la liste des enfants de P */
/*    la question doit donner la valeur [] a L1*/
/* Exemple typique de construction d'une liste avec des elements qui */
/*    satisfont un pedicat (ici enfant/2) */
enfants(P,L,[E|L1]) :- enfant(E,P),not(membre(E,L)),!,enfants(P,[E|L],L1).
enfants(_,_,[]).
```

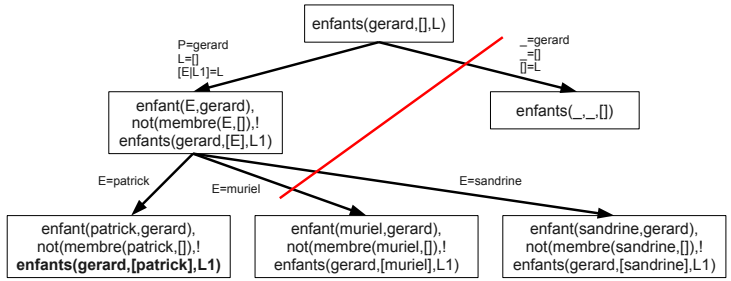
Étendons l'exemple du début 4 / 10

Résultat de *enfants(gerard,[],L)*



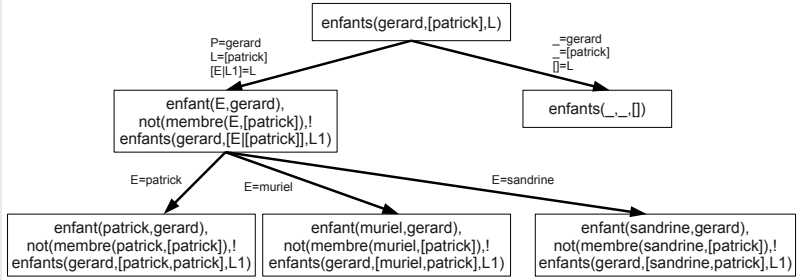
Étendons l'exemple du début 5 / 10

Résultat de *enfants(gerard,[],L)*



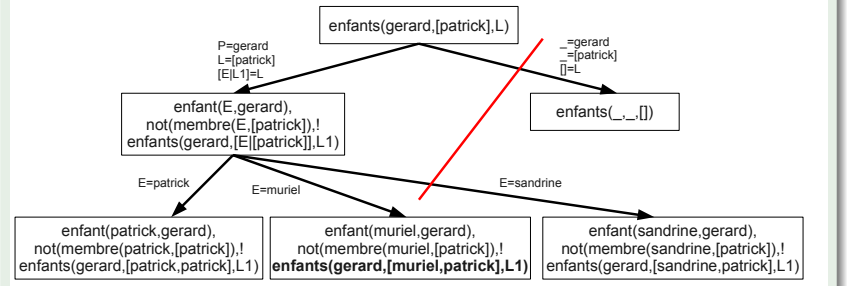
Étendons l'exemple du début 6 / 10

Résultat de *enfants(gerard,[],L)*



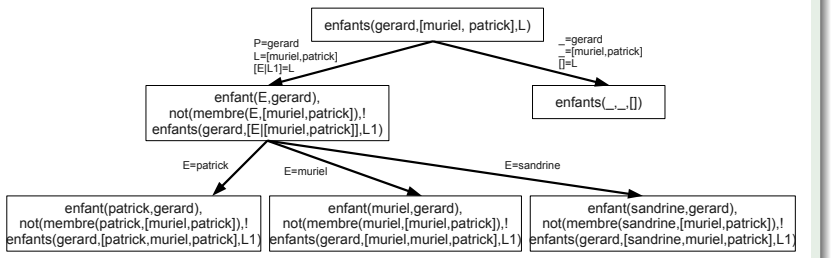
Étendons l'exemple du début 7 / 10

Résultat de *enfants(gerard,[],L)*



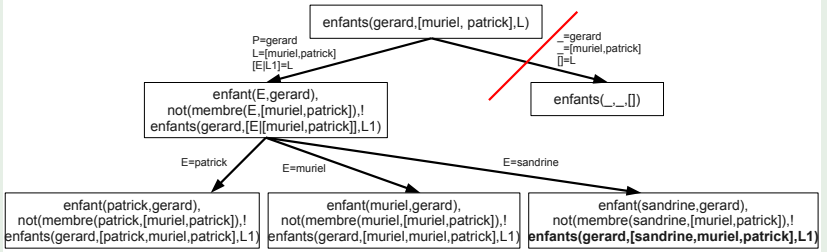
Étendons l'exemple du début 8 / 10

Résultat de *enfants(gerard,[],L)*



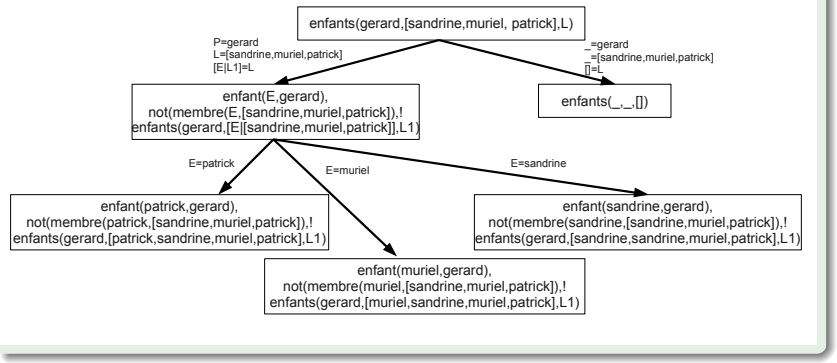
Étendons l'exemple du début 9 / 10

Résultat de *enfants(gerard,[],L)*



Étendons l'exemple du début 10 / 10

Résultat de *enfants(gerard,[],L)*



Quelques prédicats utiles 1 / 2

Sur les variables

- *var/1*, *nonvar/1*, *atom/1*, *integer/1*, *float/1*, *string/1*, *atomic/1*

Sur les listes

- *append/3*, *member/2*, *delete/3*, *last/2*, *reverse/2*, *length/2*, *merge/3*, ...

Sur les entrées sorties

- *read/1*, *write/1*, *writeln/1*, *open/4*, *close/1*, ...

Quelques prédicats utiles 2 / 2

De contrôle : *repeat/0* et *fail/0*

- *fail/0* n'est jamais démontrable
- *repeat/0* toujours démontrable avec un point choix :

```
repeat.
repeat :- repeat.
```

- Ces deux prédicats permettent de faire des boucles

```
affiche :- repeat, writeln("Saisissez une expression prolog (fin. pour sortir) : "),
     read(X), afficher(X), !.
afficher(fin) :- !.
afficher(X) :- write("L'arbre correspondant est:"), display(X), writeln(""), fail.
```


Métaprédicat

- Un métaprédicat est un prédicat qui peut prendre en paramètre des prédicats

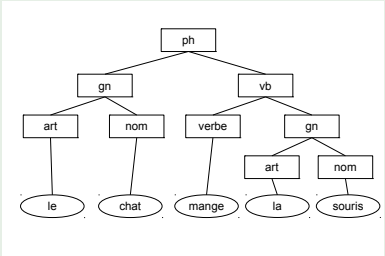
```
?- plus(1,1,X).
X = 2.

?- maplist(plus(1),[1,2,3],L).
L = [2, 3, 4].

?- findall(X,enfant(X,gerard,_),L).
L = [patrick, muriel, sandrine].
```


Un analyseur syntaxique 1 / 5

analyse/2

- Développer un prédicat qui est capable de créer un arbre syntaxique à partir d'une liste de mots (avec vérification en nombre et en genre)

- ≪ Le chat mange la souris ≫



- Prolog :

```
?- analyse([le,chat,mange,la,souris],A).
A = ph(gn(art(le), nom(chat)), vb(verbe(mange), gn(art(la), nom(souris)))) ;
false.
```


Un analyseur syntaxique 2 / 5

article/3

```
/* Issue du cours www.onsefaitchier.com/cours/cours12.pdf :-) */
article(le,masc,sing).
article(la,fem,sing).
article(les,fem,plur).
article(les,masc,plur).
```

nom/3

```
nom(chat,masc,sing).
nom(souris,fem,sing).
nom(souris,fem,plur).
nom(chats,masc,plur).
```

Un analyseur syntaxique 3 / 5

adjectif/3

```
adjectif(petite,fem,sing).
adjectif(gentille,fem,sing).
adjectif(gros,masc,sing).
adjectif(petites,fem,plur).
```

verbe/2

```
verbe(mange,sing).
verbe(mangent,plur).
verbe(avale,sing).
verbe(poursuit,sing).
verbe(poursuivent,plur).
```

analyse/2

```
analyse(L,ph(GpNominal,GpVerbal)) :- append(L1,L2,L), gr_nominal(
    L1,SingPlur,GpNominal), gr_verbal(L2,SingPlur,GpVerbal).
```


Un analyseur syntaxique 4 / 5

gr_nominal/3

```
gr_nominal([Art,Nom],SingPlur,gn(art(Art),nom(Nom))) :- article(Art,Genre,SingPlur),
     nom(Nom,Genre,SingPlur).
gr_nominal([Art,Adjectif,Nom],SingPlur,gn(art(Art),adj(Adjectif),nom(Nom))) :- article(
     Art,Genre,SingPlur),adjectif(Adjectif,Genre,SingPlur),nom(Nom,Genre,SingPlur).
```

gr_verbal/3

```
gr_verbal([V|Vs],SingPlur,vb(verbe(V),COD)) :- verbe(V,SingPlur),
     gr_nominal(Vs,_,COD).
```


Un analyseur syntaxique 5 / 5

```
3 ?- analyse([le,gros,chat,mange,la,souris],Arbre).
Arbre = ph(gn(art(le), adj(gros), nom(chat)), vb(verbe(mange), gn(art(la), nom(souris))
     )) .

4 ?- analyse([_,_,mange,_,_,_],A).
A = ph(gn(art(le), nom(chat)), vb(verbe(mange), gn(art(le), adj(gros), nom(chat)))) ;
A = ph(gn(art(le), nom(chat)), vb(verbe(mange), gn(art(la), adj(petite), nom(souris))))
     ;
A = ph(gn(art(le), nom(chat)), vb(verbe(mange), gn(art(la), adj(gentille), nom(souris))
     )) ;
A = ph(gn(art(le), nom(chat)), vb(verbe(mange), gn(art(les), adj(petites), nom(souris))
     )) ;
A = ph(gn(art(la), nom(souris)), vb(verbe(mange), gn(art(le), adj(gros), nom(chat)))) ;
A = ph(gn(art(la), nom(souris)), vb(verbe(mange), gn(art(la), adj(petite), nom(souris))
     )) ;
A = ph(gn(art(la), nom(souris)), vb(verbe(mange), gn(art(la), adj(gentille), nom(souris
     )))) ;
A = ph(gn(art(la), nom(souris)), vb(verbe(mange), gn(art(les), adj(petites), nom(souris
     )))) ;
false.

5 ?-
```


SEND+MORE=MONEY 1 / 4

- Développer un prédicat qui résoud le problème suivant :

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

tel qu'une lettre représente un chiffre, tous différents les uns des autres

chiffre/1

```
chiffre(X) :- member(X,[0,1,2,3,4,5,6,7,8,9]).
```

retenue/1

```
retenue(X) :- member(X,[0,1]).
```

SEND+MORE=MONEY 2 / 4

tous_dif/2

```
tous_dif([]) :- !.
tous_dif([E|L]) :- not(member(E,L)),tous_dif(L).
```

solution/8 (naïve)

```
solution(S,E,N,D,M,O,R,Y) :-
        chiffre(S),chiffre(E),chiffre(N),chiffre(D),
        chiffre(M),chiffre(O),chiffre(R),chiffre(Y),
        retenue(R1),retenue(R2),retenue(R3),retenue(R4),
        tous_dif([S,E,N,D,M,O,R,Y]),
        S=\=0,M=\=0,M=:=R4,
        M+S+R3=:=10*R4+O,
        O+E+R2=:=10*R3+N,
        R+N+R1=:=10*R2+E,
        D+E=:=10*R1+Y.
```

Plusieurs minutes ($10^8 \times 2^4$ solutions)

SEND+MORE=MONEY 3 / 4

solution2/8

```
solution2(S,E,N,D,M,O,R,Y) :-
        chiffre(S),S=\=0,
        chiffre(E),not(member(E,[S])),
        chiffre(N),not(member(N,[S,E])),
        chiffre(D),not(member(D,[S,E,N])),
        chiffre(M),not(member(M,[S,E,N,D])),M=\=0,
        chiffre(O),not(member(O,[S,E,N,D,M])),
        chiffre(R),not(member(R,[S,E,N,D,M,O])),
        chiffre(Y),not(member(Y,[S,E,N,D,M,O,R])),
        retenue(R1),retenue(R2),retenue(R3),retenue(R4),
        M=:=R4,
        M+S+R3=:=10*R4+O,
        O+E+R2=:=10*R3+N,
        R+N+R1=:=10*R2+E,
        D+E=:=10*R1+Y.
```


SEND+MORE=MONEY 4 / 4

```
6 ?- solution2(S,E,N,D,M,O,R,Y).
S = 9,
E = 5,
N = 6,
D = 7,
M = 1,
O = 0,
R = 8,
Y = 2 ;
false.

7 ?-
```

$$\begin{array}{cccccc} & 9 & 5 & 6 & 7 \\ + & & 1 & 0 & 8 & 5 \\ \hline & 1 & 0 & 6 & 5 & 2 \end{array}$$

Web sémantique 1 / 2

Semweb

- SWI Prolog propose la bibliothèqe (`semweb`) permettant de gérer des triplets RDF [a] composé antre autres des modules :
  - `semweb/rdf_db` permet de requêter les triplets ;
  - `semweb/turtle` permet de charger des triplets au format turtle (surcharge du prédicat *rdf_load/1*) ;
  - `semweb/http_plugin` permet de charger des triplets RDF directement depuis un serveur Web ;
  - `semweb/sparql_client` permet d'exécuter des requêtes SPARQL ;
  - `semweb/portray` permet d'afficher des URI avec des préfixes ;
  - etc.

a. cf. `http://www.swi-prolog.org/pldoc/man?section=semweb-rdf-db`

Web sémantique 2 / 2

ClioPatria

- Le projet ClioPatria est une base de données Web sémantique développée en prolog, proposant :
  - une entrée SPARQL
  - un raisonneur
  - une *front-end* web
- `http://cliopatria.swi-prolog.org`

Conclusion

Conclusion

- Ceci n'est qu'une introduction au Prolog
- Le plus difficile est de penser autrement
- Plusieurs modules swi-prolog sont disponibles (cf. `rep_prolog/library`)) :
  - Chargement, *use_module*(*library*('*nom*')).
  - Exemple
    - nlp, *Natural Language Processing*
    - clp, Programmation Logique avec Contrainte
    - http (*http/*)
- On peut interconnecter Prolog avec d'autres langages (C, C++, Java, etc.)

Références

[Bel94] P. Bellot.
Objectif Prolog.
Masson, 1994.