Document - Introduction au Prolog

Cours « Document et Web Sémantique »

Nicolas Delestre

INSA ROUEN NORMANDIE

Plan

INSA ROUEN NORMANDIE

Définitions 1 / 2

Système formel

≪ modélisation mathématique d'un langage en général spécialisé [...] Les éléments linguistiques, mots, phrases, discours, etc., sont représentés par des objets finis [...] Le propre d'un système formel est que la correction au sens grammatical de ses éléments est vérifiable algorithmiquement, c'est-à-dire que ceux-ci forment un ensemble récursif. ≫ (wikipédia, 2021)

Logique mathématique

≪ discipline des mathématiques introduite à la fin du XIXe siècle, qui s'est donné comme objet l'étude des mathématiques en tant que langage. Les objets fondamentaux de la logique mathématique sont les formules représentant les énoncés mathématiques [...] et les sémantiques qui donnent un "sens" mathématique générique aux formules. ≫ (wikipédia, 2021)

Cf. Science4All, "1+1=2" :
`https://www.youtube.com/watch?v=oKprCgIKWxo&list=PLtzmb84AoqRRgqV5DfE_ykuGQK-vCJ_0t&index=13`

Définitions 2 / 2

Logique informatique

≪ [domaine de l'informatique qui] étudie l'automatisation des calculs et des démonstrations, les fondements théoriques de la conception des systèmes, la programmation et l'intelligence artificielle. L'approche informatique est aujourd'hui cruciale car, en essayant de mécaniser les raisonnements, voire de les automatiser, la logique et les mathématiques vivent une véritable révolution depuis la fin du XXe siècle. ≫ (wikipédia, 2021)

Domaines scientifiques connexes

- Calculabilité
- Complexité

INSA ROUEN NORMANDIE

Les logiques 1 / 3

Les trois composants

- Formule : phrase du langage de la logique, suites de symboles bien formées qui respectent syntaxiquement des règles de construction
- Interprétation : fonction qui associe à toute formule un objet dans un monde (nommé modèle) qui permet de définir la validité des formules
- Déduction : algorithme qui permet de créer de nouvelles formules (théorème, conclusion) à partir de formules connues (axiomes, prémisses) à partir de règles d'inférence :

$$\frac{A_1 \dots A_n}{B} \qquad A_1 \dots A_n \vdash B$$

Les axiomes sont les lois logiques, notés $\vdash A$

INSA ROUEN NORMANDIE

Les logiques 2 / 3

Plusieurs logiques

- Elles se distinguent par les éléments de bases pour écrire des formulent et par les systèmes de déduction
- Plus une logique est « riche », plus son pouvoir de représentation est grand mais plus les algorithmes de déduction sont complexes (au sens de la complexité algorithmique)

Exemples de logique

- Logique des propositions
- Logique des prédicats du premier ordre : extension de la logique des propositions avec des variables, des fonctions, des prédicats et deux quantificateurs
- Logique d'ordre supérieur : extension la logique des prédicats telles que les variables peuvent référencer des fonctions et des prédicats
- Logiques modales : contextualisations des formules, par exemple
  - temporelles : *demain*, il pleut
  - épistémiques : *Paul croit* qu'il pleut

Les logiques 3 / 3

Restreindre une logique

- Pour améliorer les performances des algorithmes de déduction : par exemple les clauses de Horn (toutes les formules sont de la formes $(\neg r_1 \vee \neg r_2 \cdots \vee \neg r_n) \vee h$)
- Pour s'adapter au mieux au domaine à modéliser : par exemple les logiques de description pour représenter des connaissances

INSA ROUEN NORMANDIE

La logique des propositions

- Logique la plus simple, permet de représenter formellement « S'il pleut ou il neige alors il y a des nuages »
- Constituants du langage :
  - variables propositionnelles
  - opérateurs : $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$
  - parenthèses pour enlever des ambiguïtés
- Formules propositionnelles (ou propositions) :
  - une variable est une proposition
  - Si p et q sont des propositions alors $\neg p$, $p \wedge q$, $p \vee q$, $p \rightarrow q$,$p \leftrightarrow q$ et (p) sont des propositions

Si A, B sont des propositions :

- $(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$ est une proposition
- $A \wedge B \wedge$ n'est pas une proposition

Interprétation ou modèle

Principe

- Associer une valeur de vérité (Vrai ou Faux) à chacune des variables propositionnelles
- Utiliser les tables de vérités de bases associées à chaque opérateur

Vocabulaire

- Une formule est *satisfaisable* s'il existe un modèle qui la rend vraie
- Une formule est une *tautologie* si tous les modèles la rendent vraie, notée $\models A$. A est alors appelé théorème
- Une formule est *falsifiable* s'il existe une modèle qui la rend fausse
- Une forumle est une *antilogie* si tous les modèles la rendent fausse

INSA ROUEN NORMANDIE

Comment démontrer qu'une formule est un théorème ?

Première méthode : calculer toutes les interprétations = sa table de vérité

- Avantage : méthode décidable
- Inconvénient : très inefficace, si n propositions 2^n calculs

Deuxième méthode : démontrer la formule

- Théorème d'adéquation : Si $A \vdash B$ alors $A \models B$
- Théorème de complétude : Si $A \models B$ alors $A \vdash B$
- Quatre méthodes :
  - **Frege et Hilbert : une seule règle, le Modus ponens, des axiomes**
  - Déduction naturelle : un seul axiome, plusieurs règles
  - Par résolution : généralisation du Modues ponens, transformation de la formule en forme normale conjonctive (conjonctions de disjonctions, de clauses) afin de produire une disjonction
  - **Méthodes des tableaux : construction d'un arbre (un créant deux fils si $\vee$) à partir de la négation de la formule en essayent de fermer chaque branche (une branche est fermée lorsqu'il y a une contradiction)**

ROUEN NORMANDIE

Fredge et Hilbert

Modus ponens

$$\frac{\vdash A \vdash (A \rightarrow B)}{\vdash B}$$

$\vdash A$ indique que A est un axiome ou un thèorème

Axiomes [GRT02]

1. $\vdash A \rightarrow (B \rightarrow A)$
2. $\vdash (A \rightarrow (A \rightarrow B)) \rightarrow (A \rightarrow B)$
3. $\vdash (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
4. $\vdash (A \leftrightarrow B) \rightarrow (A \rightarrow B)$
5. $\vdash (A \leftrightarrow B) \rightarrow (B \rightarrow A)$
6. $\vdash (\neg B \rightarrow \neg A) \rightarrow (A \rightarrow B)$
7. $\vdash A \vee B \leftrightarrow (\neg A \rightarrow B)$
8. $\vdash A \wedge B \leftrightarrow \neg(A \rightarrow \neg B)$

Exemple

Démonstration que $A \rightarrow A$ est un théorème [GRT02]

- Axiome 1 : $\vdash A \rightarrow (A \rightarrow A)$
- Axiome 2 : $\vdash (A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A)$
- Modus ponens :

$$\frac{\vdash A \rightarrow (A \rightarrow A) \quad \vdash (A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A)}{\vdash A \rightarrow A}$$

Méthode des tableaux

Principe

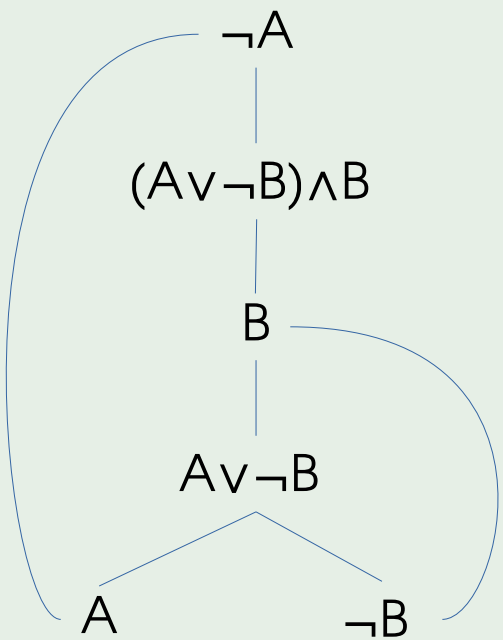
- Représenter la négation de la formule à démontrer à l'aide d'un arbre et essayer ≪ de fermer ≫ chaque branche de l'arbre, c'est-à-dire trouve une contradiction pour chaque branche
- Algorithme (simplifié) :
  - Représenter la formule uniquement avec les opérateurs $\neg$, $\wedge$ et $\vee$
  - Si la formule courante est l'opération binaire :
    - $\wedge$ les deux opérandes apparaissent dans la même branche (règle α)
    - $\vee$ la branche courante est décomposée en deux branches, chacune avec une opérande (règle β)

INSA ROUEN NORMANDIE

Exemple

Démontrer que $((A \vee \neg B) \wedge B) \rightarrow A$ est un théorème (Wikipédia)

- $((A \vee \neg B) \wedge B) \rightarrow A$ peut s'écrire $((\neg A \wedge B) \vee \neg B) \vee A$
- Sa négation est donc $((A \vee \neg B) \wedge B) \wedge \neg A$



Limites de la logique des propositions

- Les propositions sont indépendantes les unes des autres, même si dans une interprétation elles caractérisent un même individu
- Les propositions ne peuvent pas caractériser un ensemble d'individus

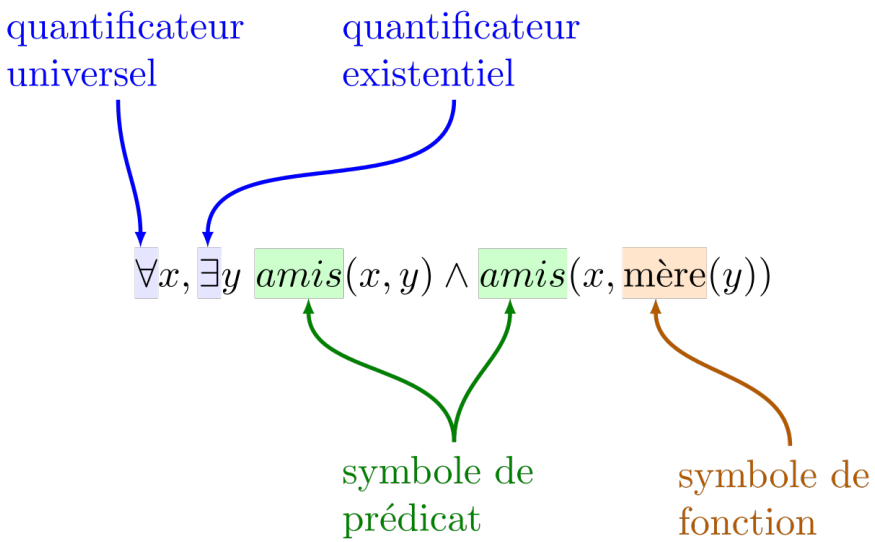
- Tous les hommes sont mortels
- Socrate est un homme
- Socrate est mortel

INSA ROUEN NORMANDIE

La logique des prédicats

- Logique mathématique qui remplace les variables propositionnelles par des prédicats et permet l'utilisation de deux quantificateurs : « tous les hommes sont mortels »
- Constituants du langage :
  - constantes : désignent certains éléments du monde
  - fonctions : relient les éléments du monde
  - variables : représentent les éléments du monde sans les désigner. Une variable peut être liée ou libre
  - prédicats : caractérisent les éléments du monde
  - quatificateurs : $\forall$ et $\exists$
  - opérateurs : $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$
  - parenthèses pour enlever des ambiguïtés
- Formules :
  - $p(t_1, t_2, \ldots, t_n)$ ou p est un prédicat, et t_i des termes (constante, fonction, variable) est une formule
  - Si e et f sont des formules alors $\neg e$, $e \wedge f$, $e \vee f$, $e \rightarrow f$,$e \leftrightarrow f$, (e), $\forall x\ e$, $\exists x\ e$ sont des formules

Exemples



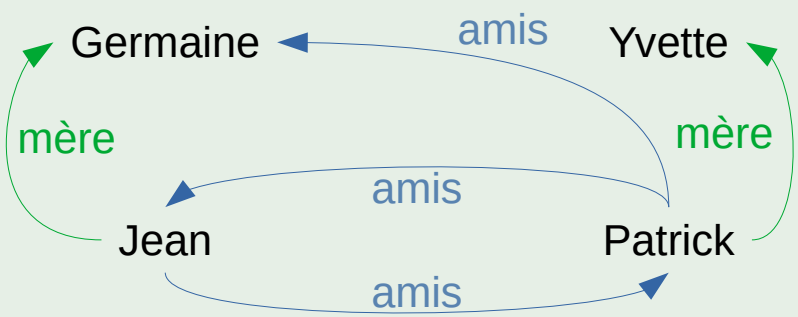
Source wikipédia (2021)

- $\forall x\ homme(x) \rightarrow mortel(x)$
- $homme(Socrate)$
- $mortel(Socrate)$

Interprétation 1 / 2

- Ensemble d'individus caractérisés par les prédicats (faits) et possiblement reliés par les fonctions
- Associer les constantes et variables aux individus
- Comme pour la logique des propositions, une formule peut être **satisfaisable**, **falsifiable**, une tautologie, une antologie

$\forall x, \exists y \ amis(x, y) \wedge amis(x, mere(y))$



Donc la formule est falsifiable

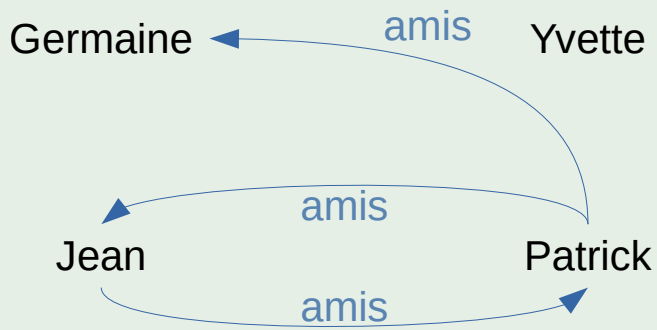
Qu'en est il pour :

- $\exists x, \forall y \ amis(x, y) \wedge amis(x, mere(y))$
- $\exists x, \exists y \ amis(x, y) \wedge amis(x, mere(y))$

Interprétation 2 / 2

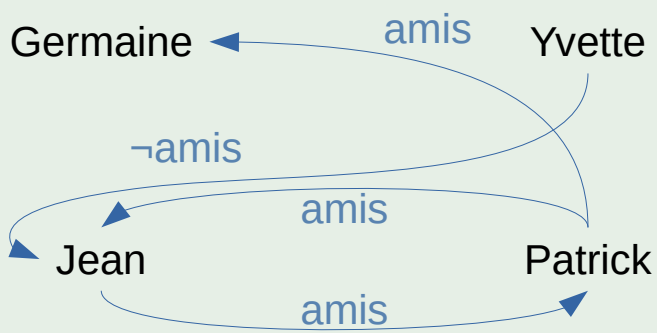
Monde fermé

Tout fait non déclaré est considéré comme Faux



Monde ouvert

Il faut déclarer un fait Faux pour qu'il le soit



Déduction

Plusieurs algorithmes

- Méthodes des tableaux, en ajoutant deux règles nommées γ pour traiter le quantificateur $\forall$ et δ pour $\exists$
- Chaînage avant ou arrière : les formules sont représentées par des implications (donc avec une partie gauche et une partie droite $G \rightarrow D$)

  chaine avant l'objectif est de compléter le modèle grâce à l'implication

  chainage arrière l'objectif est de vérifier une hypothèse (partie droite de l'implication) en vérifiant la validité de la partie gauche

Utilisation en informatique

- Pour une interprétation donnée (faits), trouver les individus telle qu'une formule (question) soit vraie

Paradigmes de programmation

Rappels

- Un paradigme de programmation est la façon d'aborder la résolution d'un problème
- Un paradigme de programmation repose sur des principes, des méthodes et outils

Paradigme de programmation impérative

- Principe : problème et solution sont représentés par des états (état initial et état final) et un programme représente le **comment** passer de l'état initial à l'état final
- Outil : l'affectation permet de passer d'un état i à un état $i + 1$
- Méthode : les schémas (séquentiel, conditionnel et itératif) permettent d'organiser les affectations

INSA ROUEN NORMANDIE

Un nouveau paradigme de programmation

Paradigme de programmation déclarative

- Principe : on décrit le problème, le **quoi** et pas le comment
- Outil : un moteur interprète le problème pour trouver/construire la solution
- Méthode : *dépendant du sous paradigme*

Paradigme de programmation logique

- Sous paradigme de la programmation déclarative
- Principe : on décrit le problème en terme de faits (formules logiques avec uniquement des constantes), de règles (formules logiques faisant intervenir des variables), et une question (formule logique)
- Outil : trouver pour quelles constantes (solution(s)) l'interprétation de la question est vraie
- Méthode : algorithmes de démonstration

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

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

INSA ROUEN NORMANDIE

Introduction 2 / 2

Principe

- Prolog permet au programmeur d'expliciter des faits, des règles et de répondre à des questions en inférant sur ces faits à l'aide de 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 (le modèle) que des règles
- Les atomes (commencent par une minuscule, ou entre simples côtes si utilisation de l'espace ou commençant par une majuscule) 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 utilisent *swi-prolog*

Introduction de la syntaxe 1 / 3

- Fait :

```
fait(atome_constante1,atome_constante2,...,atome_constante3).
```

- Règle :

```
regle(Var_ou_atom_cons01, Var_ou_ayom_cons02, ...):-
            cond1(Var_ou_atom_cons11,Var_ou_atom_cons12,...),
            cond2(Var_ou_atom_cons21,Var_ou_atom_cons22,...),
            ...,
            condn(Var_ou_atom_consn1,Var_ou_atom_consn2,...).
```

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)

INSA ROUEN NORMANDIE

Introduction de la syntaxe 2 / 3

Exemple

```
1  /* homme(X) est vrai si X est un homme */
2  homme(patrick).
3  homme(gerard).
4  homme(louis).
5  homme(pierre).
6
7  /* femme(X) est vrai si X est une femme */
8  femme(therese).
9  femme(sandrine).
10 femme(muriel).
11 femme(germaine).
12 femme(yvette).
13
14 /* enfant_parents(Enfant,Parent1,Parent2) est vrai si Enfant est un enfant de Parent1
       et Parent2 */
15 enfant_parents(gerard,germaine,louis).
16 enfant_parents(therese,yvette,pierre).
17 enfant_parents(patrick,gerard,therese).
18 enfant_parents(muriel,gerard,therese).
19 enfant_parents(sandrine,gerard,therese).
20 enfant_parents(astride,jean,therese).
```

INSA ROUEN NORMANDIE

Introduction de la syntaxe 3 / 3

```
22 /* enfant_parentEnfant,Parent) est vrai si Enfant est un enfant de Parent */
23 enfant_parent(Enfant,Parent) :- enfant_parents(Enfant,Parent,_).
24 enfant_parent(Enfant,Parent) :- enfant_parents(Enfant,_,Parent).
25
26 /* pere_enfant(Pere,Enfant) est vrai si Pere est pere de Enfant */
27 pere_enfant(Pere,Enfant) :- homme(Pere), enfant_parent(Enfant,Pere).
28
29 /* mere_enfantMere,Enfant) est vrai si Mere est mere de Enfant */
30 mere_enfant(Mere,Enfant) :- femme(Mere), enfant_parent(Enfant,Mere).
31
32 /* grandpere_enfant(GrandPere,Enfant) vrai si GrandPere est grand-pere de Enfant */
33 grandpere_enfant(GrandPere,Enfant) :- pere_enfant(GrandPere,Parent), pere_enfant(Parent
       ,Enfant).
34 grandpere_enfant(GrandPere,Enfant) :- pere_enfant(GrandPere,Parent), mere_enfant(Parent
       ,Enfant).
35
36 /* grandmere_enfant(GrandMere,Enfant) vrai si GrandMere est grand-mere de Enfant */
37 grandmere_enfant(GrandMere,Enfant) :- mere_enfant(GrandMere,Parent), pere_enfant(Parent
       ,Enfant).
38 grandmere_enfant(GrandMere,Enfant) :- mere_enfant(GrandMere,Parent), mere_enfant(Parent
       ,Enfant).
```

INSA ROUEN NORMANDIE

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 `semweb/rdf_db` :
    ```
    :- use_module(library(semweb/rdf_db))
    ```
- 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_enfant(gerard,patrick).
true .

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

L'interpréteur de swi-prolog 3 / 3

```
4 ?- pere_enfant(X,patrick).
X = gerard ;
false.

5 ?- pere_enfant(gerard,X).
X = patrick ;
X = muriel ;
X = sandrine ;
false.

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

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

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

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

Fonctionnement du moteur 1 / 5

- Le moteur de Prolog utilise un algorithme de chainage arrière

Définition

Unification « procédé par lequel on essaie de rendre deux formules identiques en donnant des valeurs aux variables qu'elles contiennent » [Bel94]

Point de choix le fait d'avoir des faits et règles compatibles pour essayer de démontrer une question (ou conjonction de question)

Pas de démonstration Le fait de réussir à faire une unification au niveau d'un point de choix. Un pas de démonstration entraîne la démonstration d'une ou plusieurs questions

Backtracking le fait de ne plus avoir à/pouvoir faire d'unification. Le moteur prolog remonte jusqu'au dernier point de choix et essaye d'appliquer la pas de démonstration suivant

Fonctionnement du moteur 2 / 5

Pour une question donnée (conjonction de termes)

Étape 1 :

- Le moteur prolog sélectionne les faits et règles compatibles (sans essayer les unifications) avec le premier terme de la question : c'est un point de choix
  - l'ordre de la liste de faits et règles compatibles est l'ordre du programme Prolog (cela peut donc avoir des conséquences sur les résultats)
  - si cette liste est vide, alors le moteur fait un backtracking avec la valeur *false* (ou *fails*, suivant la version du prolog)
  - s'il réussit à démontrer le premier terme, il essaye de démontrer le suivant. S'il réussit à démontrer tous les termes il fait un backtracking avec la réponse `true`, sinon `false`

INSA ROUEN NORMANDIE

Fonctionnement du moteur 3 / 5

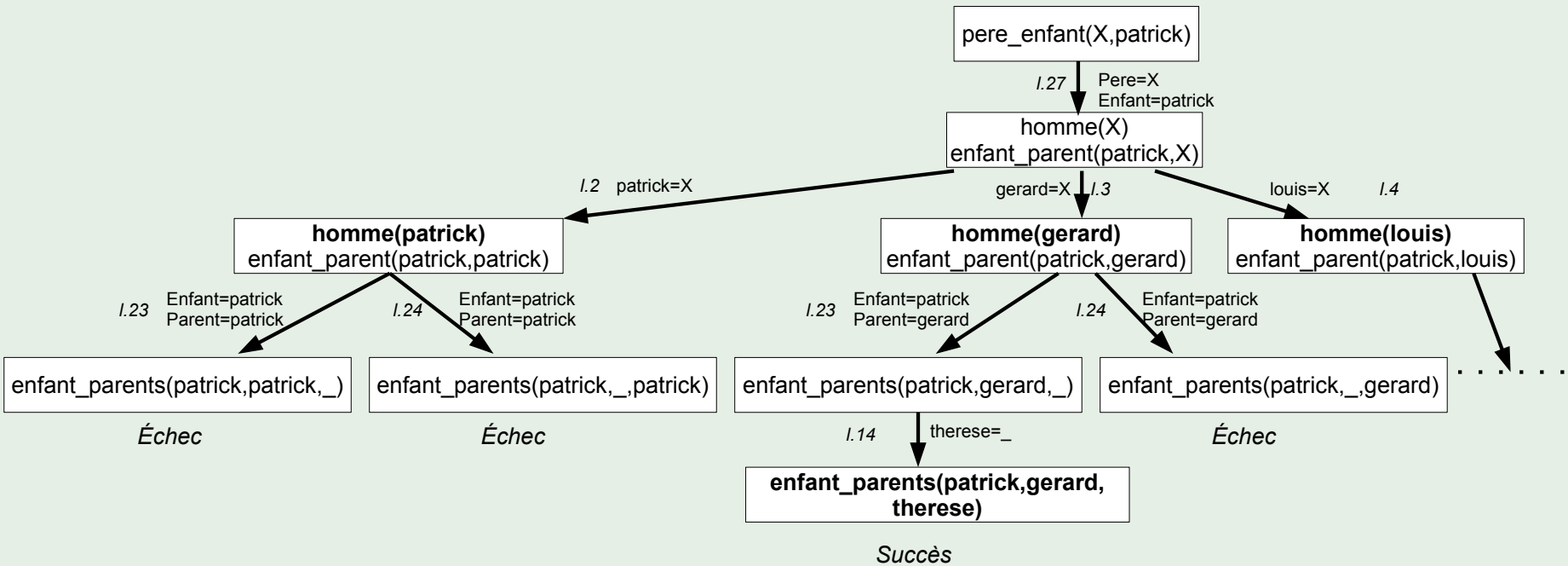
Pour une question donnée (conjonction de termes)

Étape 2 :

- Pour chaque fait et règle compatible, le moteur essaye de faire une(des) unification(s) :
  - Si des unifications sont possibles, ils remplacent les variables par les valeurs :
    - s'il y a encore des variables libres, il fait un pas de démonstration, et essaye donc de démontrer les autres termes de la question.
      Si ce pas de démonstration réussit (obtention de la valeur *true*) il affiche les unifications ayant permis la démonstration (solution). Si le fait ou la règle courante n'est pas le dernier élément de la liste, le moteur propose de poursuivre la recherche de solution.
    - s'il n'y a plus de variables libres alors la démonstration est réussie, il fait un backtracking avec la valeur *true* et les valeurs unifiées aux variables.
  - Sinon il passe au fait ou à la règle suivante
- S'il n'y a eu aucun pas de démonstration, alors le moteur fait un backtracking avec la valeur `false`

Fonctionnement du moteur 4 / 5

pere_enfant(X,patrick).



Fonctionnement du moteur 5 / 5

Pour débogguer

- les prédicats *debug* et *nodebug*
- les prédicats *trace* et *notrace*
- le prédicat *guitracer*

Exemple

```
?- trace.
true.
?- pere_enfant(X,patrick).
   Call: (8) pere_enfant(_6552, patrick) ? creep
   Call: (9) homme(_6552) ? creep
   Exit: (9) homme(patrick) ? creep
   Call: (9) enfant_parent(patrick, patrick) ? creep
   Call: (10) enfant_parents(patrick, patrick, _6780) ? creep
...
   Exit: (9) enfant_parent(patrick, gerard) ? creep
   Exit: (8) pere_enfant(gerard, patrick) ? creep
X = gerard ;
```

Précision concernant la syntaxe et le vocabulaire 1 / 3

Terme

Élément de base du langage. Il y a deux types de termes :

- terme simple : atome, constante ou variable
  - atome :
    - séquence de lettres (majuscule ou minuscule), des chiffres ou tiret bas, commençant par une minuscule
    - séquence de caractères entourée de cotes simples
    - séquence de caractères spéciaux (+, -, *, /, <, >, =, :, ., @, ~)
  - constante :
    - chaine de caractères entourées de double cotes
    - nombre : entier ou flottant représentés en décimale
  - variable : séquence de lettres (majuscule ou minuscule), des chiffres ou tiret bas, commençant par une majuscule ou un tiret bas (le tiret bas seul est appelé variable anonyme)
- terme complexe (ou terme composé ou structure) : association d'un foncteur (un atome) et d'arguments (n'importe quel type de terme) entre parenthèses, séparés des virgules

Précision concernant la syntaxe et le vocabulaire 2 / 3

Atome logique

- Relation entre termes
- Ressemble syntaxiquement à un terme complexe

Clause

- Affirmation inconditionnelle (ou fait) : un seul atome logique qui se termine par un `.`
- Affirmation conditionnnelle (ou règle) : succession d'atomes logiques tel que le premier et le deuxième sont séparés par `:-` et les autres par des `,` qui se termine par un `.`

Prédicats

- Ensemble de clauses de même arité

Précision concernant la syntaxe et le vocabulaire 3 / 3

Programme prolog

- Ensemble de prédicats
- L'exécution d'un programme revient à poser une question

Utilisation des termes complexes 1 / 3

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).
...
```

Utilisation des termes complexes 2 / 3

Avec les termes complexes (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.
```

Utilisation des termes complexes 3 / 3

Remarque

- Sans les termes complexes, 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

INSA ROUEN NORMANDIE

Liste

- Terme complexe `[|]/2` tel que le premier arguments est la tête de liste et le second la queue de liste.
- La liste vide est notée `[]`
- Facilité syntaxique :
  - les éléments de la liste sont entourés de crochets et séparés par des virgules : `'[|]'(1, '[|]'(2, []))` peut être notée `[1,2]`
  - une liste est entourée de crochet dont le premier élément est séparé du reste de la liste (une liste) par une barre :
    `'[|]'(1, '[|]'(2, []))` peut être notée `[1|[2|[]]]`

Exemples

```
1 ?- '[|]'(1,[])=[1].
true.

2 ?- '[|]'(1,'[|]'(2,[]))=[1,2].
true.

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

4 ?- [1,a,b]=[1|[a,b]].
true.

5 ?- [1]=[1|[]].
true.
```

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,[_|L]) :- membre(E,L).
```

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]) :- E\=E1, supprimer(E,L,L1).
```

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]) ?
```

ROUEN NORMANDIE

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([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 ?-
```

INSA ROUEN NORMANDIE

Operateurs 1 / 4

- 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 (y pour « est associatif », x non)

    Par exemple l'addition est infixe, associative à gauche (yfx) : 1+2+3 ⇒ (1+2)+3
  - 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,[_|L]) :- membre(E,L).


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

Operateurs 2 / 4

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 / 4

=/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 terme complexe

Operateurs 4 / 4

Tous les opérateurs de swipl

1200	xfx	`-->, :-`
1200	fx	`:-, ?-`
1150	fx	`dynamic, discontiguous, initialization,meta_predicate, module_transparent, multifile, public, thread_local, thread_initialization, volatile`
1105	xfy	`|`
1100	xfy	`;`
1050	xfy	`->, *->`
1000	xfy	`,`
990	xfx	`:=`
900	fy	`\+`
700	xfx	`<, =, =.., =@=, \=@=, =:=, =<, ==, =\=, >, >=, @<, @=<, @>, @>=, \=, \==, as, is, >:<, :<`
600	xfy	`:`
500	yfx	`+, -, /\, \/, xor`
500	fx	`?`
400	yfx	`*, /, //, div, rdiv, <<, >>, mod, rem`
200	xfx	`**`
200	xfy	`^`
200	fy	`+, -, \`
100	yfx	`.`
1	fx	`$`

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 termes complexes qui ne sont pas évaluées

Exemple

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

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

35 ?-
```

INSA ROUEN NORMANDIE

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[a]. En fait il évalue les deux opérandes puis les unifie (il faut donc que toutes les variables des opérandes soient déjà unifiées)
  - $=:=, =\backslash=, <, >, =<, >=$

a. Ils fonctionnent aussi pour les chaînes de caractères

inserer/3

```
/* inserer(E,L1,L2) est vrai lorsque L2 represente 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]) :- 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/3* 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)

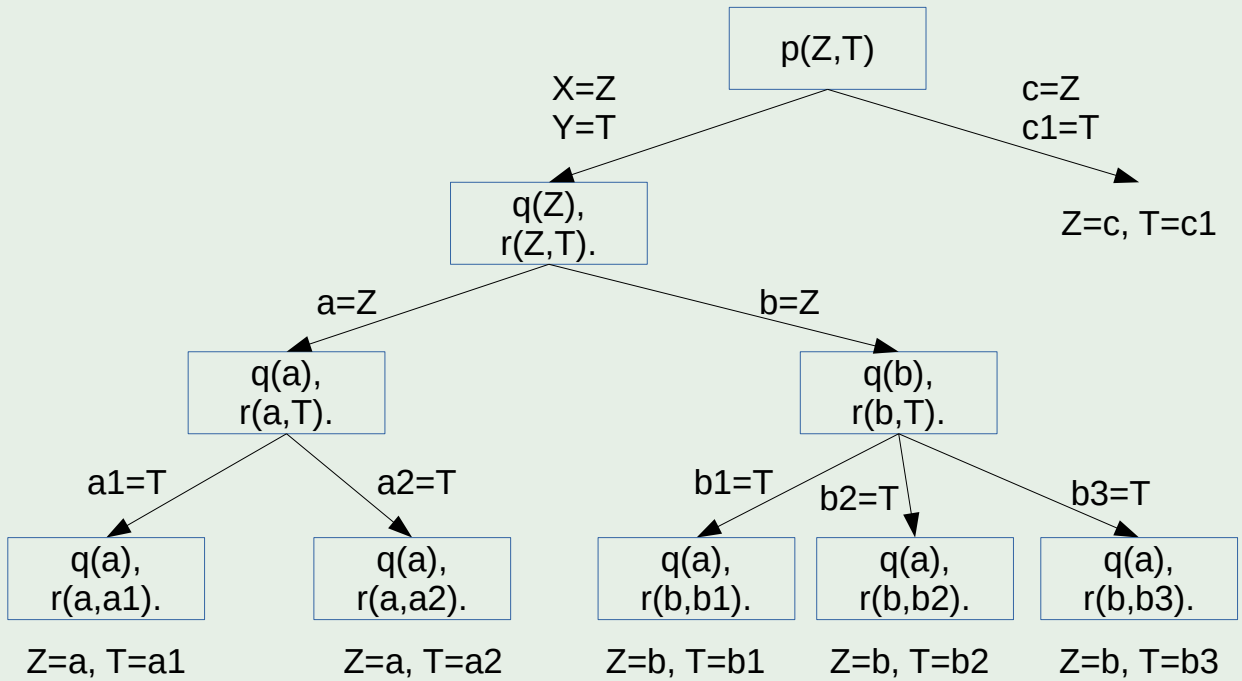
Attention

- 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*, ou n'en donner aucune alors qu'il y en avait)
  - Si la sémantique du prédicat est inchangée, il s'agit d'un *cut* vert
  - Sinon c'est un *cut* rouge

Un exemple sans cut

```
q(a). q(b).
r(a,a1). r(a,a2).
r(b,b1). r(b,b2). r(b,b3).

p(X,Y) :- q(X), r(X,Y).
p(c,c1).
```

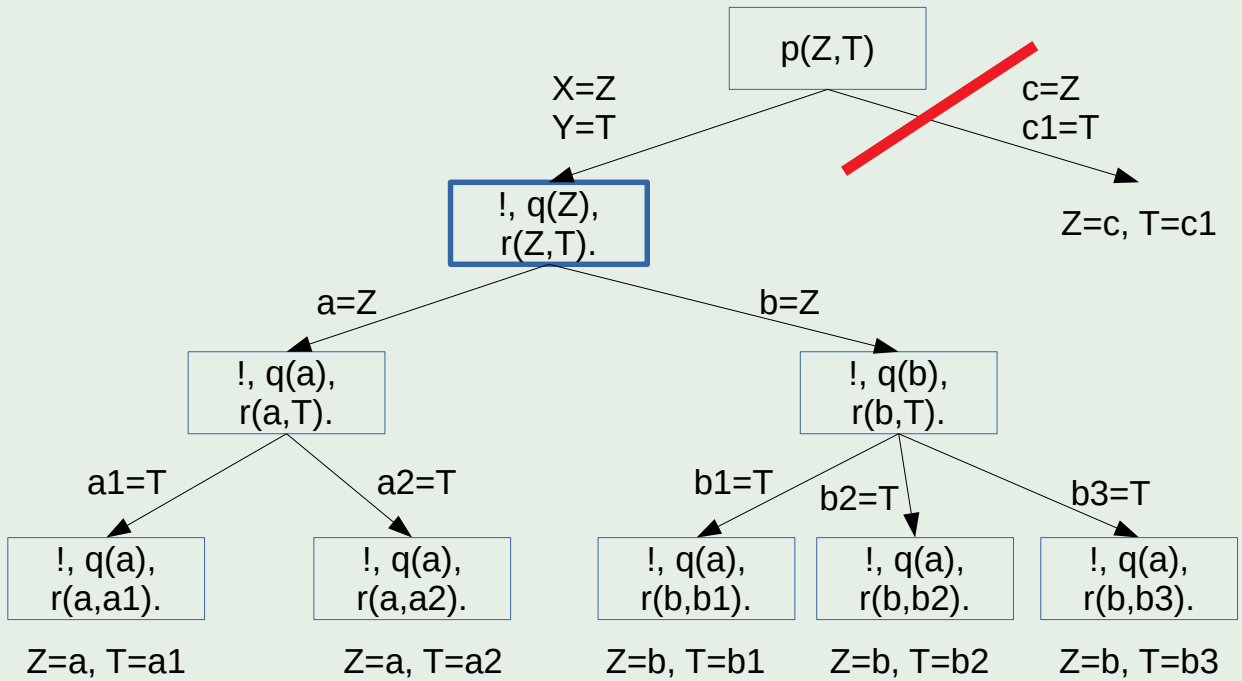


inspiré de « Introduction à PROLOG » Christine Solnon
https://perso.liris.cnrs.fr/christine.solnon/prolog.html

Un exemple avec un cut au début

```
q(a). q(b).
r(a,a1). r(a,a2).
r(b,b1). r(b,b2). r(b,b3).
```

```
p(X,Y) :- !, q(X), r(X,Y).
p(c,c1).
```

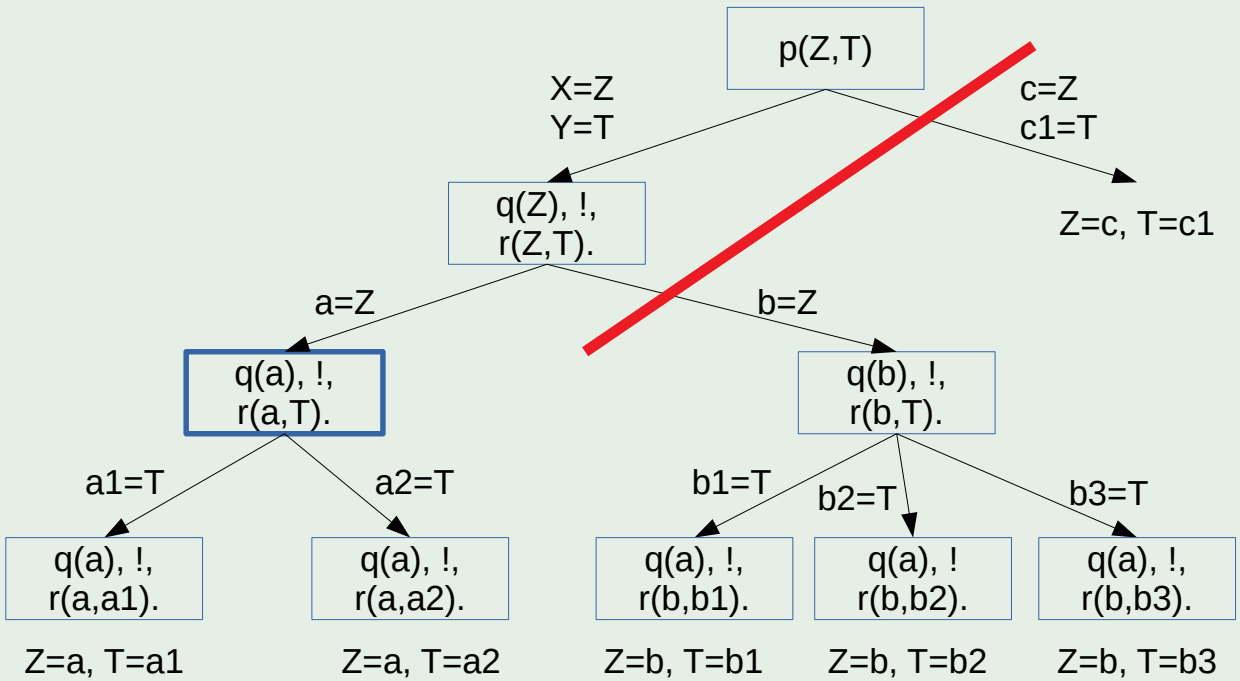


inspiré de « Introduction à PROLOG » Christine Solnon
https://perso.liris.cnrs.fr/christine.solnon/prolog.html

Un exemple avec un cut au milieu

```
q(a). q(b).
r(a,a1). r(a,a2).
r(b,b1). r(b,b2). r(b,b3).

p(X,Y) :- q(X), !, r(X,Y).
p(c,c1).
```

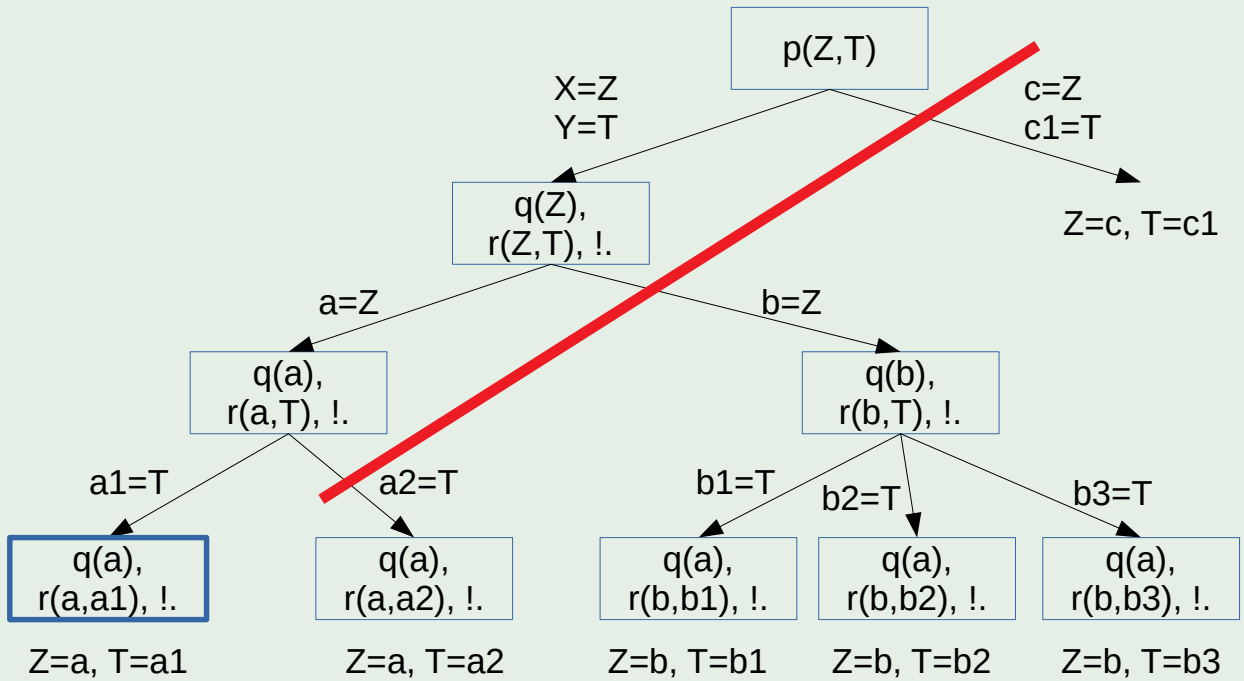


inspiré de « Introduction à PROLOG » Christine Solnon
https://perso.liris.cnrs.fr/christine.solnon/prolog.html

Un exemple avec un cut à la fin

```
q(a). q(b).
r(a,a1). r(a,a2).
r(b,b1). r(b,b2). r(b,b3).

p(X,Y) :- q(X), r(X,Y), !.
p(c,c1).
```



inspiré de « Introduction à PROLOG » Christine Solnon
https://perso.liris.cnrs.fr/christine.solnon/prolog.html

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,[_|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).
```

INSA ROUEN NORMANDIE

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).
```

Les accumulateurs 1 / 9

- *Design pattern* qui permet de « stocker » des résultats

parent_enfants/2

- Développez le prédicat *enfants/2* tel `parent_enfants(P,L)` est vrai lorsque *L* contient les enfants de *P*.

parent_enfants/2

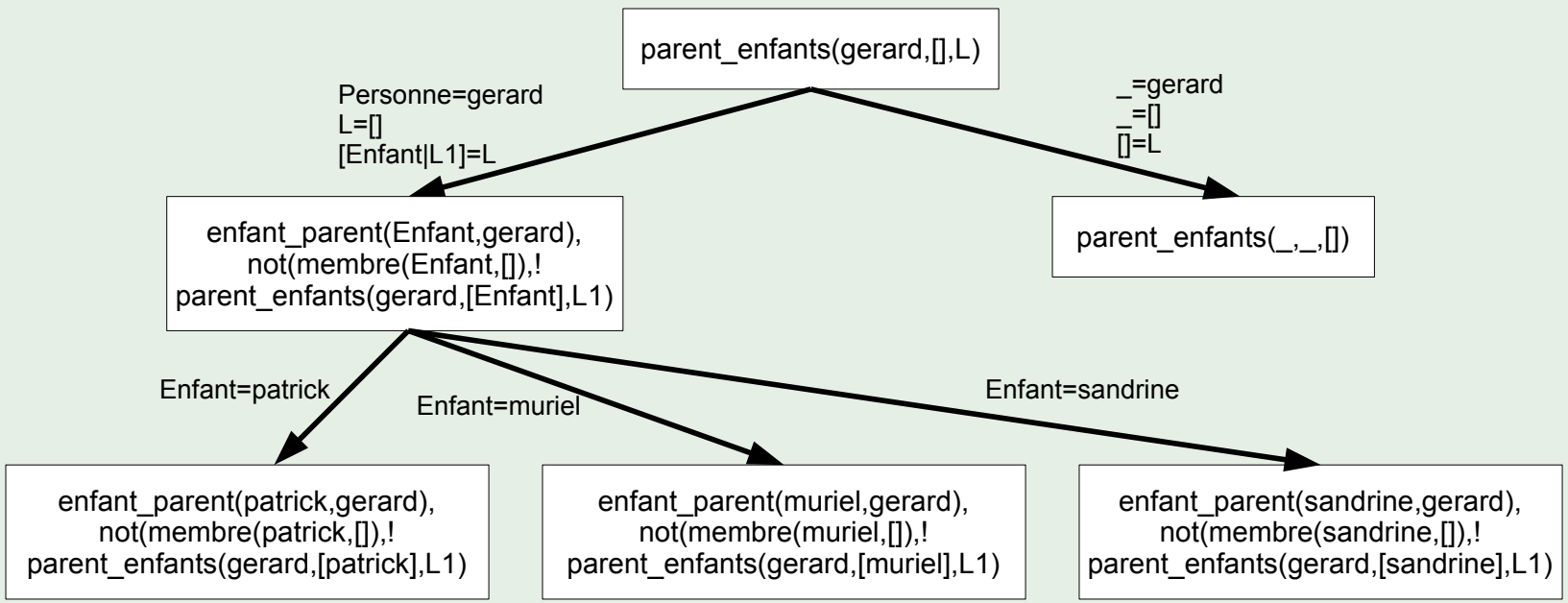
```
/* parent_enfants(Personne,L) est vrai si L est la liste des enfants de Personne */
parent_enfants(Personne,L) :- parent_enfants(Personne,[],L).
```

enfants/3

```
/* parent_enfants(Personne,L1,L2) est vrai si L2 est la liste des enfants de Personne */
/*    la question doit donner la valeur [] a L1*/
/* Exemple typique de construction d'une liste avec des elements qui */
/*    satisfont un predicat (ici enfant_parent/2) */
parent_enfants(Personne,L,[Enfant|L1]) :- enfant_parent(Enfant,Personne),!,not(membre(Enfant,L)),
      parent_enfants(Personne,[Enfant|L],L1).
parent_enfants(_,_,[]).
```

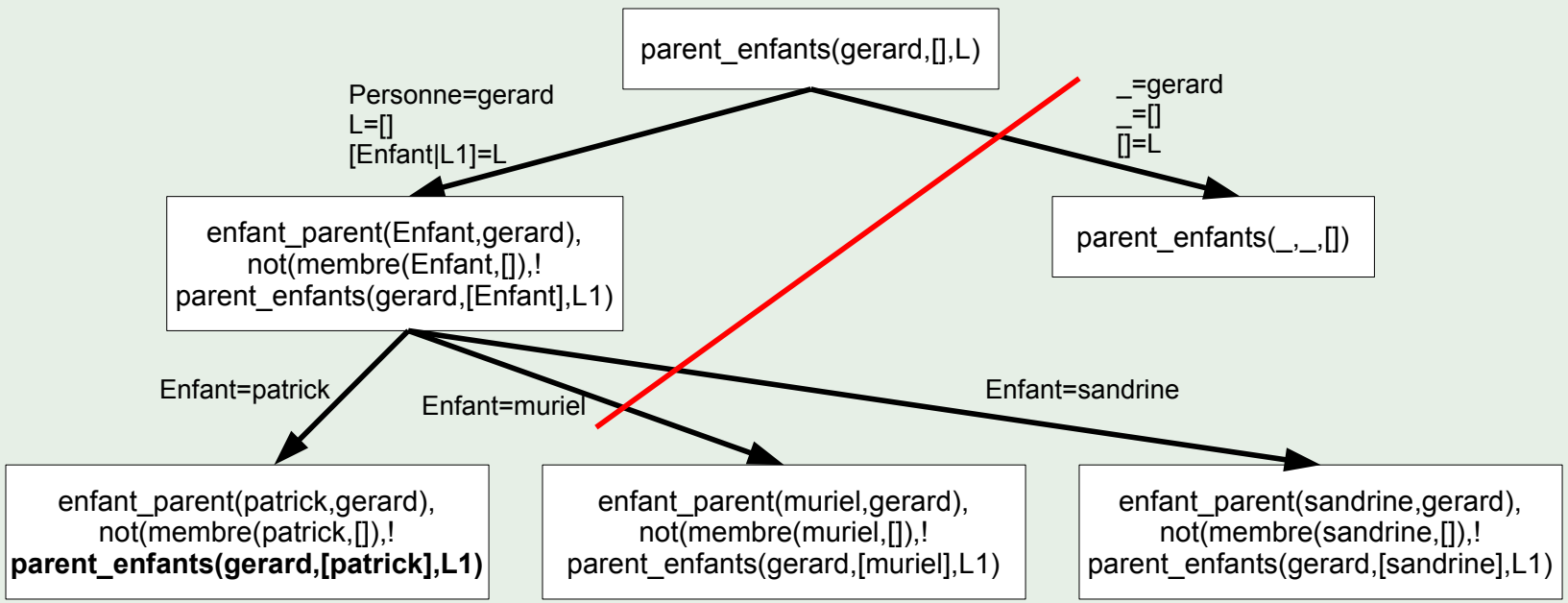
Les accumulateurs 2 / 9

Résultat de *parent_enfants(gerard,[],L)*



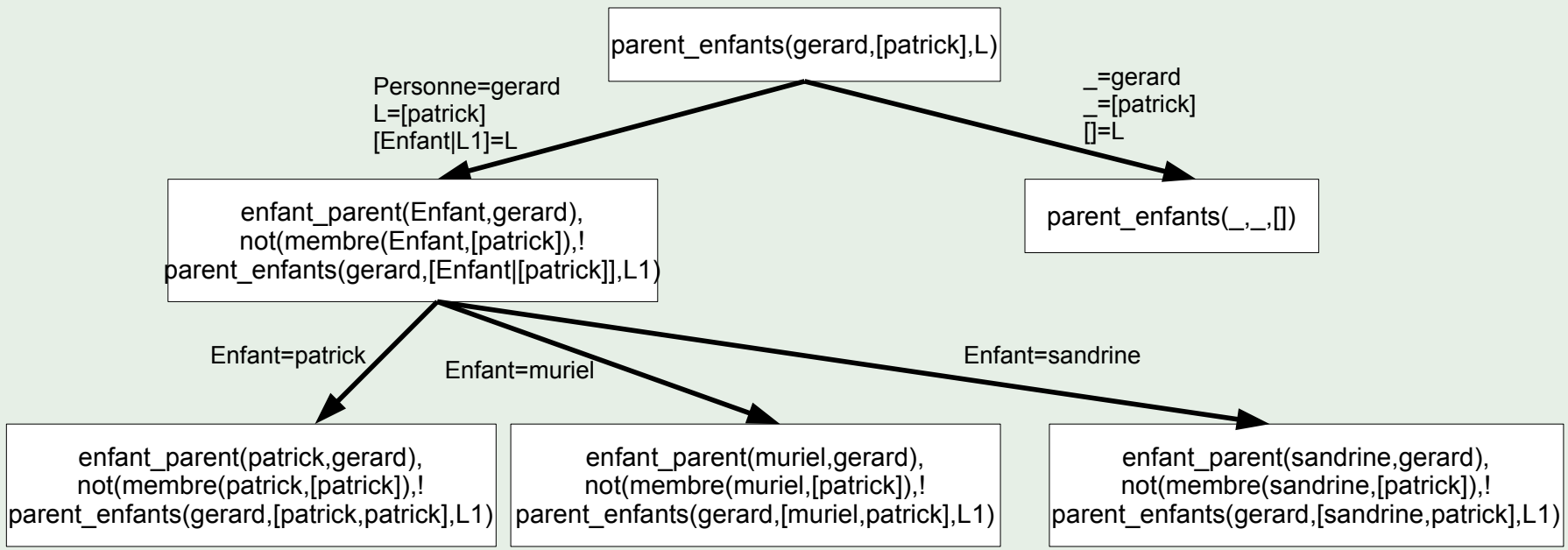
Les accumulateurs 3 / 9

Résultat de *parent_enfants(gerard,[],L)*



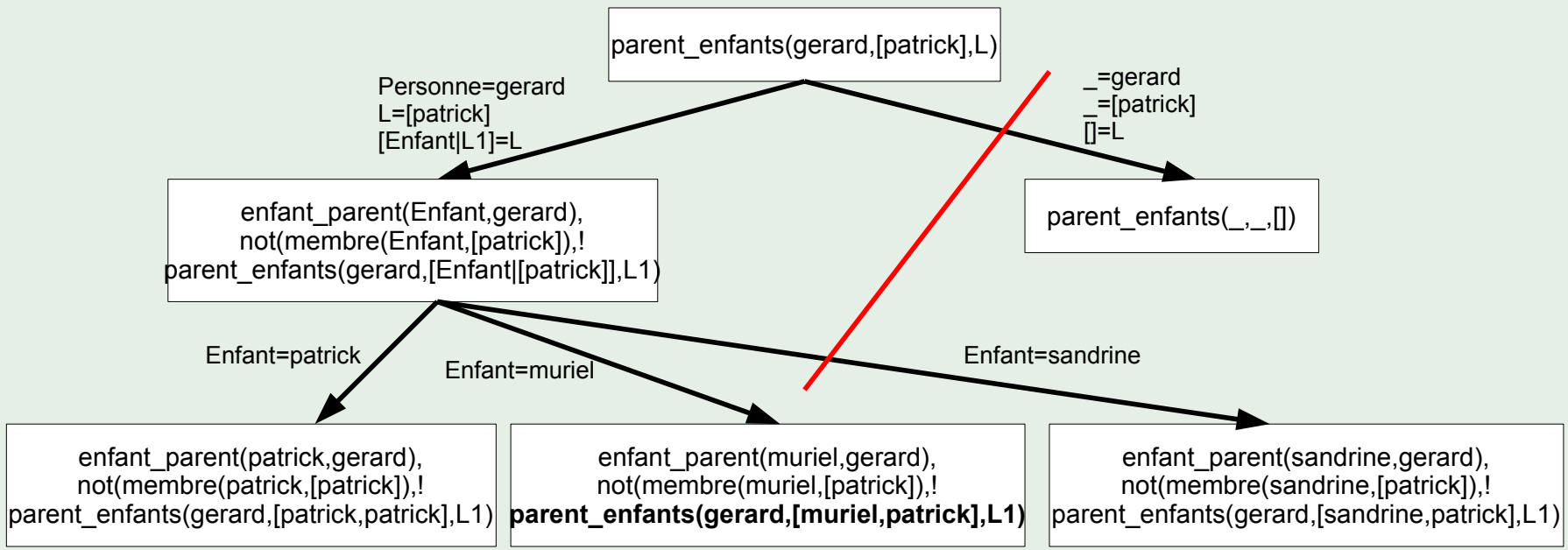
Les accumulateurs 4 / 9

Résultat de *parent_enfants(gerard,[patrick],L)*



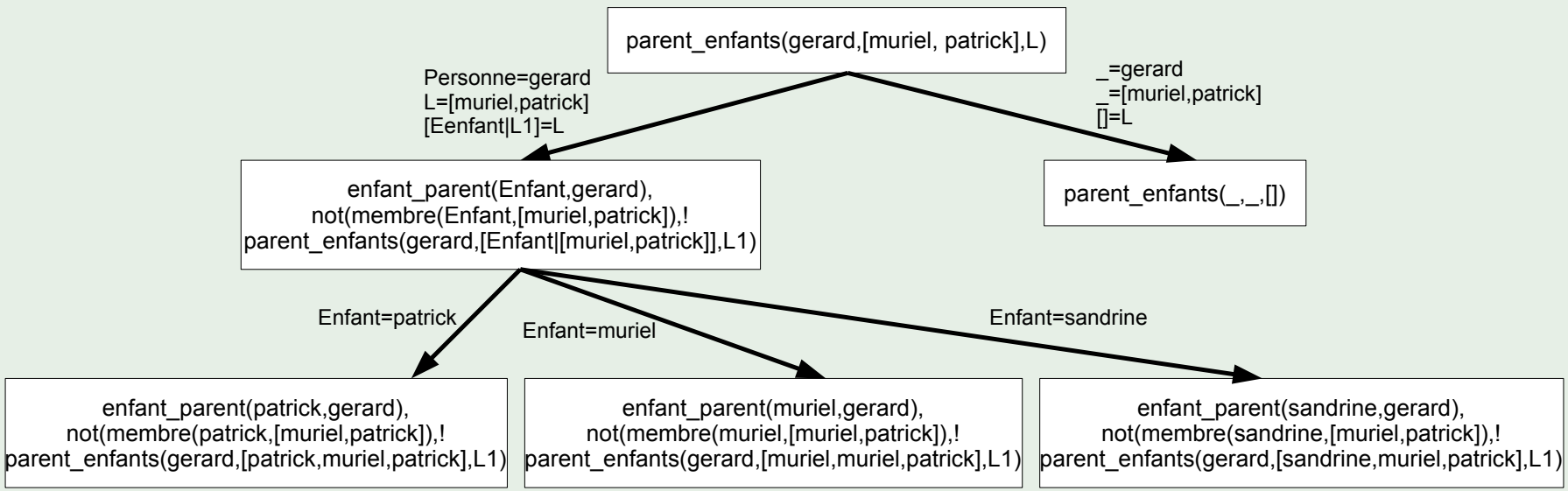
Les accumulateurs 5 / 9

Résultat de *parent_enfants(gerard,[patrick],L)*



Les accumulateurs 6 / 9

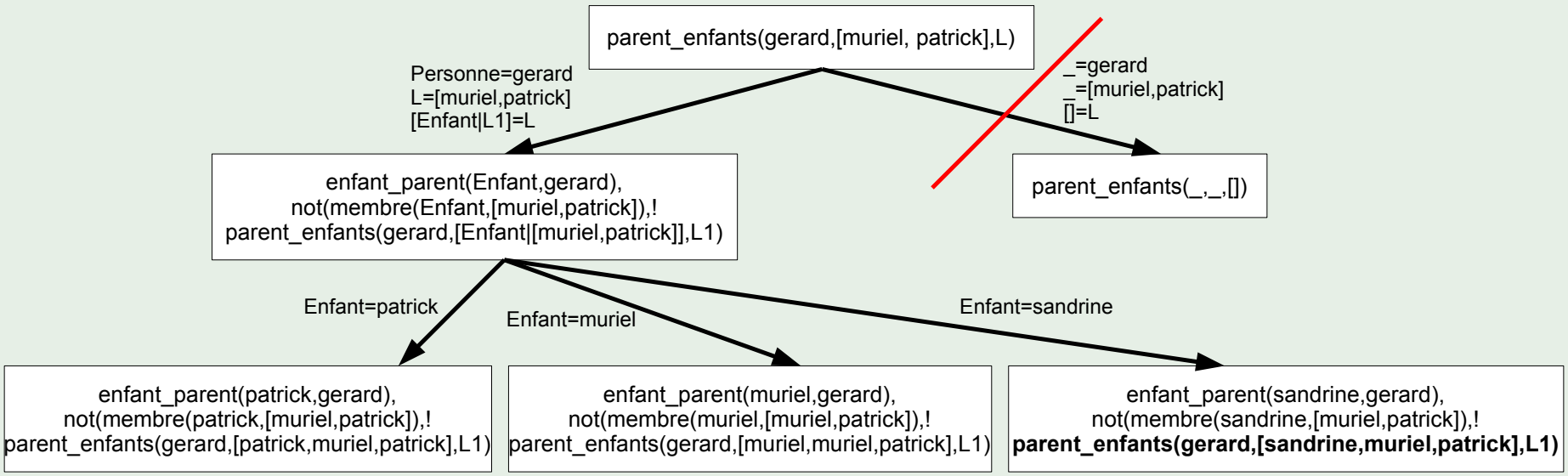
Résultat de *parent_enfants(gerard,[muriel, patrick],L)*



INSA ROUEN NORMANDIE

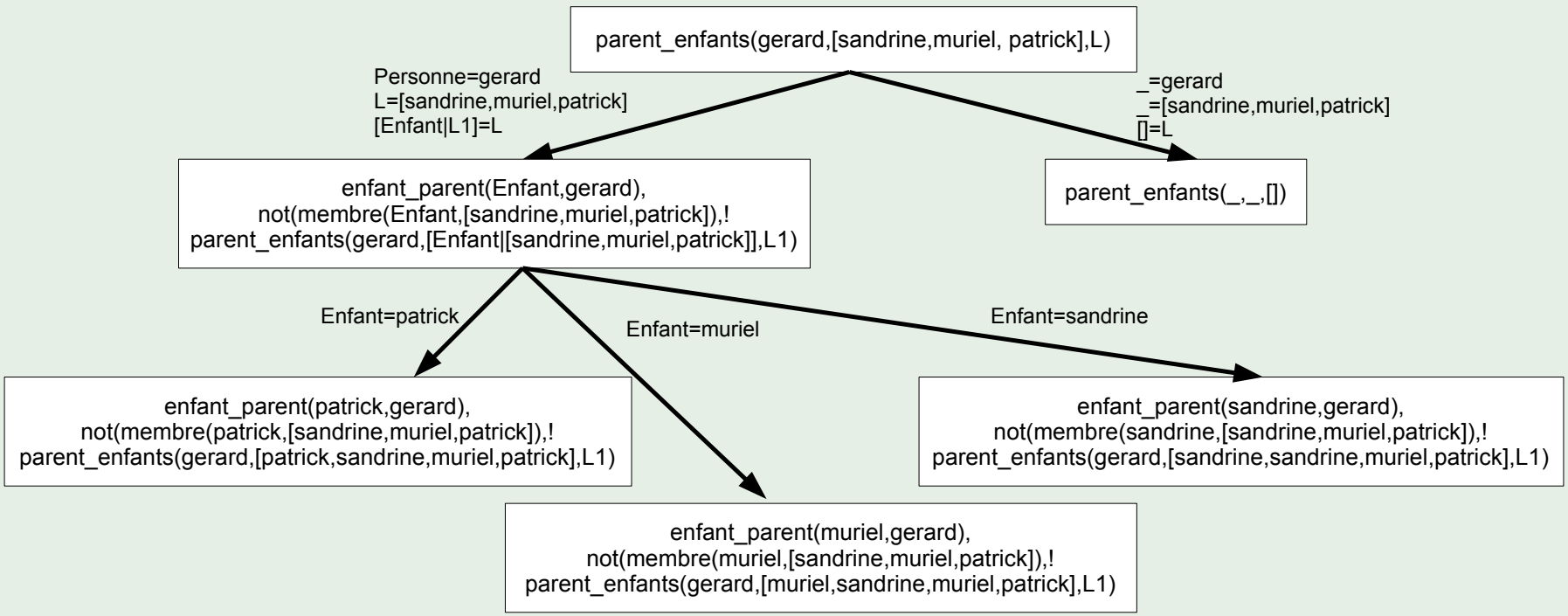
Les accumulateurs 7 / 9

Résultat de *parent_enfants(gerard,[muriel, patrick],L)*



Les accumulateurs 8 / 9

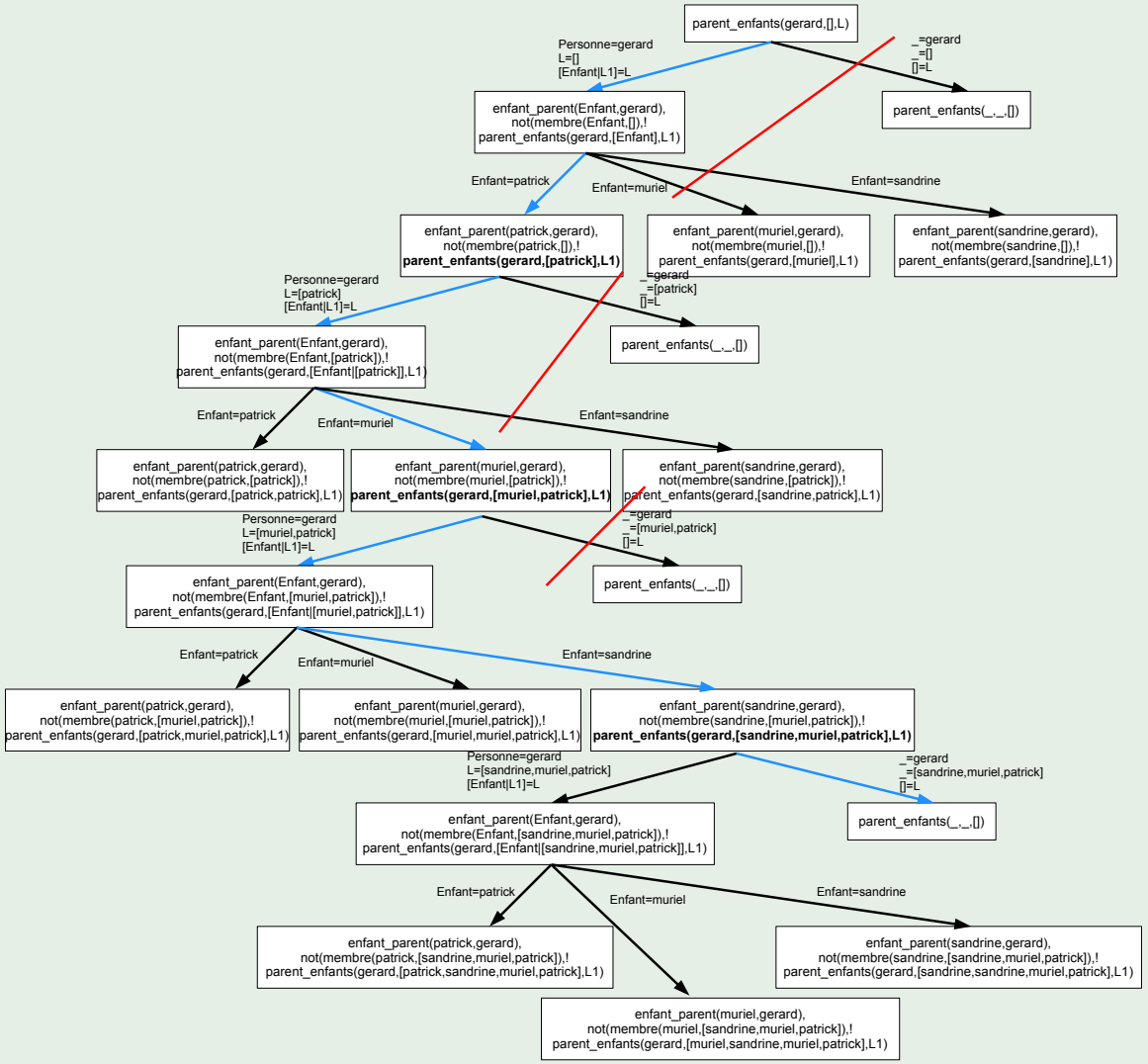
Résultat de *enfants(gerard,[sandrine, muriel, patrick],L)*



INSA ROUEN NORMANDIE

Les accumulateurs 9 / 9

En résumé, résultat de *parent_enfants(gerard,[],L)*



Quelques prédicats utiles

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*, ...

INSA ROUEN NORMANDIE

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_parent(X,gerard),L).
L = [patrick, muriel, sandrine].
```

Remarque

- Les métaprédicat sorte du champ de la logique du premier ordre

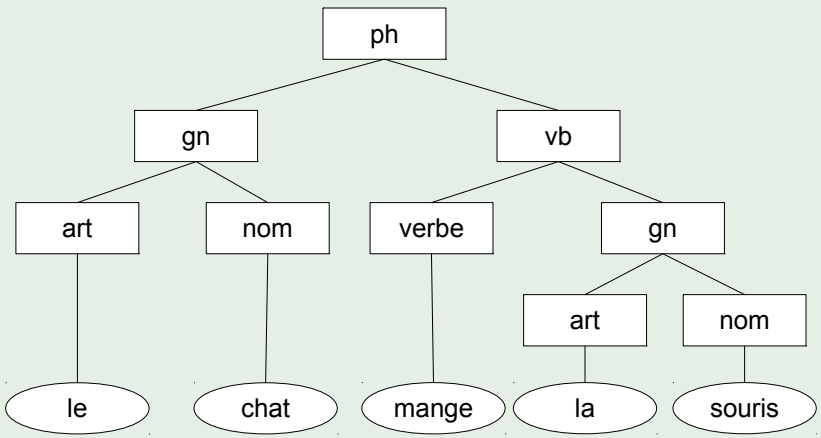
INSA ROUEN NORMANDIE

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 ?-
```

INSA ROUEN NORMANDIE

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] composée entre 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`

INSA ROUEN NORMANDIE

Exemple avec semweb 1 / 5

famille.ttl

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix ex: <http://exemple.com/>.

ex:Personne a rdfs:Class.
ex:Homme a rdfs:Class; rdfs:subClassOf ex:Personne.
ex:Femme a rdfs:Class; rdfs:subClassOf ex:Personne.
ex:aParent a rdfs:Property; rdfs:domain ex:Personne; rdfs:range ex:Personne.
ex:aPere a rdfs:Property; rdfs:subPropertyOf ex:aParent; rdfs:range ex:Homme.

ex:patrick a ex:Homme. ex:gerard a ex:Homme.
ex:louis a ex:Homme. ex:pierre a ex:Homme.

ex:therese a ex:Femme. ex:sandrine a ex:Femme.
ex:muriel a ex:Femme. ex:germaine a ex:Femme.
ex:yvette a ex:Femme.

ex:gerard ex:aParent ex:germaine, ex:louis.
ex:therese ex:aParent ex:yvette, ex:pierre.
ex:patrick ex:aParent ex:gerard, ex:therese.
ex:muriel ex:aParent ex:gerard, ex:therese.
ex:sandrine ex:aParent ex:gerard, ex:therese.
ex:astride ex:aParent ex:jean, ex:therese.

ex:paul ex:aPere ex:edouard.
```

Exemple avec semweb 2 / 5

famille.pl

```
:- use_module(library(semweb/rdf_db)).
:- use_module(library(semweb/rdf_turtle)).
:- rdf_load('famille.ttl').

:- rdf_register_prefix(ex, 'http://exemple.com/').

homme(X) :- rdf(X, rdf:type, ex:'Homme').
:- rdf_meta(homme(r)). /* pour pouvoir utiliser la notation prefixe ex:gerard, r
     signifie ressource */

femme(X) :- rdf(X, rdf:type, ex:'Femme').
:- rdf_meta(femme(r)).

enfant_parents(Enfant, Parent1, Parent2) :- rdf(Enfant, ex:aParent, Parent1),
                                            rdf(Enfant, ex:aParent, Parent2),
                                            Parent1 \= Parent2, !.
:- rdf_meta(enfant_parents(r,r,r)).
```

Exemple avec semweb 3 / 5

```
?- [famille].
true.

?- enfant_parents(ex:gerard,X,Y).
X = 'http://exemple.com/germaine',
Y = 'http://exemple.com/louis'.

?- enfant_parents(ex:gerard,ex:germaine,Y).
Y = 'http://exemple.com/louis'.

?- enfant_parents(ex:gerard,ex:germaine,ex:louis).
true.

?- enfant_parents(ex:gerard,ex:louis,ex:germaine).
true.
```

Exemple avec semweb 4 / 5

mini raisonneur RDFS

```
:- use_module(library(semweb/rdf_db)).

domain(Property, Class) :- rdf(Property, rdfs:domain, Class).
:-rdf_meta(domain(r,r)).

range(Property, Class) :- rdf(Property, rdfs:range, Class).
:-rdf_meta(domain(r,r)).

subPropertyOf(SubProp, SubProp) :- !. /* rdfs6 */
subPropertyOf(SubProp, SuperProp) :- rdf(SubProp, rdfs:subPropertyOf, SuperProp), !.
subPropertyOf(SubProp, SuperProp) :- rdf(SubProp, rdfs:subPropertyOf, X), /* rdfs5 */
                                     rdf(X, rdfs:subPropertyOf, SuperProp).
:-rdf_meta(subPropertyOf(r,r)).

classOf(Individual,Class) :- rdf(Individual, rdf:type, Class), !.
classOf(Individual,Class) :- rdf(Individual, rdf:type, X), /*  rdfs9 */
                             subClassOf(X, Class), !.
classOf(Individual,Class) :- rdf(Individual, Property, _), /*  rdfs2 */
                             domain(Property, Class), !.
classOf(Individual,Class) :- rdf(_, Property, Individual), /*  rdfs3 */
                             range(Property, Class), !.
:-rdf_meta(classOf(r,r)).

/* RDFS entailment: rdfs11 */
subClassOf(SubClass, SuperClass) :- rdf(SubClass, rdfs:subClassOf, SuperClass), !.
subClassOf(SubClass, SuperClass) :- rdf(SubClass, rdfs:subClassOf, X),
                                    rdf(X, rdfs:subClassOf, SuperClass).
:-rdf_meta(subClassOf(r,r)).
```

Exemple avec semweb 5 / 5

```
?- [famille].
% Parsed "famille.ttl" in 0.00 sec; 33 triples
true.

?- [rdfEntailment].
true.

?- classOf(ex:gerard,ex:'Personne'). /* Alors que ex:gerard est déclaré comme étant un
     homme */
true.

?- classOf(ex:jean,X). /* Alors que le type de ex:jean n'est pas déclaré */
X = 'http://exemple.com/Personne'.

?- classOf(ex:edouard,X). /* idem */
X = 'http://exemple.com/Homme'.
```

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.)

INSA ROUEN NORMANDIE

Références

[BBS07] Patrick Blackburn, Johan Bos, and Kristina Striegnitz.
Prolog, Tout de Suite !, volume 1 of *Cahiers de Logique et d'épistémologie*.
College Publications, 2007.
Traduit par Hélène Manuélian.

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

[GRT02] Benoit Gugger, Denis Richard, and Jerzy Tomasik.
Calcul propositionnel.
http://www.lama.univ-savoie.fr/pagesmembres/saber/calcul-pro.pdf, 2002.

INSA ROUEN NORMANDIE