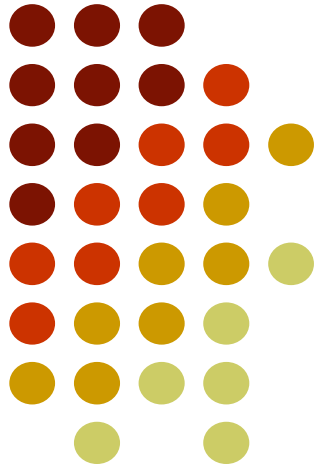
Prolog

Partie 1 : Notions de base



Jean-Michel Adam

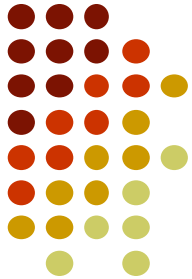
Université Grenoble Alpes
L2 - MIASHS
Grenoble - France

Organisation de l'enseignement

- 30 heures :
  - 5 Cours (5 x 1 h 30) + 1 h 30 DS (CC)
  - 6 TD (6 x 1 h 30) : Groupe 1 cog / Groupe 2 eco
  - 6 TP (6 x 2 h) : Groupe 1 cog, Groupe 2 eco, Groupe 3 eco
- Evaluation
  1. Contrôle continu : DS + un TP à rendre (50 %)
  2. Examen final (50 %)
- Supports de cours, sujets de TD et TP:

  http://miashs-www.u-ga.fr/~adamj/prolog.html

Planning 2020



Semaine du	horaire	Mardi	Jeudi	Vendredi
20/01/20	10 h 15 - 11 h 45 (1 h 30)	CM1 PROLOG	TD1 Groupe 1 (éco)	TD1 Groupe 2 (cog)
27/01/20	10 h 15 - 12 h 15 (2 heures)	TP1 Groupe 3 salle machine	TP1 Groupe 1 salle machine	TP1 Groupe 2 salle machine
03/02/20	10 h 15 - 11 h 45 (1 h 30)	CM2 PROLOG	TD2 Groupe 1 (éco)	TD2 Groupe 2 (cog+eco)
10/02/20	10 h 15 - 12 h 15 (2 heures)	TP2 Groupe 3 salle machine	TP2 Groupe 1 salle machine	TP2 Groupe 2 salle machine
17/02/20	10 h 15 - 11 h 45 (1 h 30)	CM3 PROLOG	TD3 Groupe 1 (éco)	TD3 Groupe 2 (cog)
24/02/20				
02/03/20	10 h 15 - 12 h 15 (2 heures)	TP3 Groupe 3 salle machine	TP3 Groupe 1 salle machine	TP3 Groupe 2 salle machine
09/03/20	10 h 15 - 11 h 45 (1 h 30)	**CC Prolog**	TD4 Groupe 1 (éco)	TD4 Groupe 2 (cog+eco)
16/03/20	10 h 15 - 12 h 15 (2 heures)	TP4 Groupe 3 salle machine	TP4 Groupe 1 salle machine	TP4 Groupe 2 salle machine
23/03/20	10 h 15 - 11 h 45 (1 h 30)	CM4 PROLOG	TD5 Groupe 1 (éco)	TD5 Groupe 2 (cog+eco)
30/03/20	10 h 15 - 12 h 15 (2 heures)	TP5 Groupe 3 salle machine	TP5 Groupe 1 salle machine	TP5 Groupe 2 salle machine
06/04/20	10 h 15 - 11 h 45 (1 h 30)	CM5 PROLOG	TD6 Groupe 1 (éco)	TD6 Groupe 2 (cog+eco)
13/04/20	10 h 15 - 12 h 15 (2 heures)	TP6 Groupe 3 salle machine	TP6 Groupe 1 salle machine	TP6 Groupe 2 salle machine
20/04/20				

Coordonnées de votre enseignant

Jean-Michel Adam
Maître de conférences en informatique
UFR SHS Bâtiment Michel Dubois
Bureau C013

Jean-Michel.Adam@univ-grenoble-alpes.fr

Laboratoire LIG – Bâtiment IMAG
Equipe MeTAH

Plan du cours

1 – Introduction à la programmation logique

2 – Le langage Prolog

3 – Les listes

4 – Les graphes et les arbres en Prolog

5 – Grammaires et automates d’états finis en Prolog

Partie 1
-
Introduction

Prolog

- Prolog est un langage de programmation à part.
- Le nom Prolog vient de **Pro**grammation **Log**ique.
- Il est utilisé principalement en **Intelligence Artificielle**.
- Ce qui est original, c'est qu'en Prolog, il suffit
  - de **décrire** ce que l'on sait sur le domaine étudié, (en Intelligence Artificielle, on appelle cela une base de connaissances),
  - puis on pose une **question** à propos de ce domaine

  et Prolog va nous répondre, sans que l'on ait à lui dire comment construire sa réponse !

Historique

- 1965 : John Robinson décrit les principes théoriques de la programmation logique via la règle de résolution.
- 1972 : création de Prolog par A. Colmerauer et Ph. Roussel à Marseille (Luminy), pour le traitement des langues naturelles
- 1977: premier compilateur par D.H. Warren à Edimbourg
- 1980 : reconnaissance de Prolog comme langage de développement en Intelligence Artificielle
- Depuis, plusieurs versions ont été développées avec un standard ISO, plusieurs extensions possibles.

Les principaux paradigmes en programmation

- Programmation impérative (Java L1)
- Programmation fonctionnelle (Scheme L1)
- Programmation orientée objets (Java L2)
- Programmation déclarative (Prolog L2)

La programmation impérative

- Démarche algorithmique qui décrit la façon de traiter les données pour atteindre un résultat par une série d'actions (instructions).
- L'ordre d'exécution des instructions est impératif : déterminé à l'avance.
- Le déroulement du programme est parfaitement déterministe.
- Importance des structures de contrôle
- Exemples : C, Basic, Pascal, Fortran, Java vu en L1, …

La programmation fonctionnelle

- La programmation consiste à faire de la composition de fonctions.
- Proche de la programmation impérative ; cependant ses fondements (λ-calcul) ainsi que l'absence de branchements et d'affectation (dans sa forme théorique) en fait un mode de programmation à part.
- Dans la programmation fonctionnelle, on distingue deux grandes familles de langages :
  - les langages fortement typés, ex : ML, Caml, Haskell
  - les langages non typés, ex : Lisp, Scheme

La programmation orientée objet

- Décomposition d’un objet en attributs et méthodes.
- Notions de classe et d’interface, de sous-classe, d’héritage, de polymorphisme
- Exemples : Smalltalk, C++, Java, C#, ActionScript, VisualBasic, Ada, Python.

Programmation déclarative

- La programmation **logique** qui est dite **déclarative**.
- La **programmation déclarative**
  - Le programmeur ne s'occupe pas de la manière d'obtenir le résultat; par contre, le programmeur doit faire la description du problème à résoudre en décrivant les objets concernés, leurs propriétés et les relations qu'ils vérifient.
  - Cette description constitue la **base de connaissance**
  - Ensuite, le mécanisme de résolution intégré au langage, général et universel, parcourt de façon non déterministe toutes les possibilités du problème et calcule les solutions.
  - En Prolog, la résolution s'appuie sur une déduction logique de toutes les conséquences de la base de connaissance via **l'unification**.

Information complémentaires sur Prolog

- La syntaxe et la sémantique du langage Prolog est l'ensemble des règles qui définissent comment un programme Prolog est écrit et interprété.
- Les règles sont énoncées dans la norme ISO/IEC 13211, bien qu'il existe des différences dans les implémentations de Prolog, voir:
  http://en.wikipedia.org/wiki/Comparison_of_Prolog_implementations
- Dans ce cours nous utiliserons la syntaxe dite Edinburgh.

Prolog utilisé en TP

- SWI-Prolog
  - Fonctionne sous Linux, Windows et MacOS.
  - Disponible gratuitement (licence BSD) au département informatique de l'Université de Psychologie d'Amsterdam

    http://www.swi-prolog.org/download/stable
- Autres Prolog gratuits
  - Il y a également Visual Prolog Personal Edition, gratuit pour un usage privé
  - GNU-Prolog (par l'INRIA).

Bibliographie

- ***Cours en ligne***
  - ***http://pcaboche.developpez.com/article/prolog/presentation/***
  - ***http://www.learnprolognow.org/*** ***(in English)***

- ***Pour aller plus loin:***
  - ***Foundations of Logic Programming,*** J. W. Lloyd, Springer-Verlag New York, Inc. New York, NY, USA ©1984 ISBN:0-387-13299-6
  - ***Prolog,*** F. Giannesini, H. Kanoui, R. Pasero et M. Van Caneghem, InterÉditions, 1985
  - ***Programming in Prolog,*** W.F. Clocksin, C.S. Mellish, Springer-Verlag Telos; 4th edition (September 1994)
  - ***The art of Prolog*** , 2nd Edition, Advanced Programming Techniques, L. Sterling, E. Shapiro, the MIT Press, 1994, ISBN-13: 978-0-262-19338-2
  - ***Logic Programming and Prolog (2nd ed),*** Ulf Nilsson and Jan Maluszynski, 2000

Question

- Que veut dire l’acronyme Prolog?

Question

- Quel est le paradigme de programmation lié à Prolog?
  1. Orienté-objet
  2. Fonctionnel
  3. Déclaratif
  4. Impératif

Question

- Que doit faire le programmeur Prolog?

Question

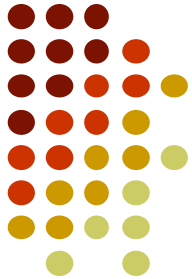
- Comment va répondre un programme Prolog?

Partie 2
-
Bases de connaissance

Construction d’un programme Prolog

- **Base de connaissance** : description du problème à résoudre.
- Une base de connaissance est composée de **Clauses**:
  - Les **Faits**
  - Les **Règles**
- L’utilisateur pose ensuite une **Question**
- Le programme donne sa **Réponse**

Base de connaissance 1
Collection de Faits



```
mange.
herbivore(chevre).
mange(loup, chevre).
```

```
?- mange.
true.
?- herbivore(chevre).
true.
?- herbivore(loup).
false.
?- carnivore(loup)
ERROR: toplevel: Undefined procedure carnivore/1
```

Les Faits

- Les faits sont des données élémentaires que l’on considère vraies.
- Ce sont des formules atomiques constituées d'un **prédicat** suivi ou pas d’une liste ordonnée d’**arguments** mis entre parenthèses qui sont les objets auxquels s'appliquent le prédicat.
- Un programme Prolog est au moins constitué d’un ou plusieurs faits car c’est à partir d’eux que Prolog va pouvoir rechercher des preuves pour répondre aux requêtes de l’utilisateur
- Ce sont en quelque sorte les hypothèses de travail.

Les Faits

- Généralement, on place toutes les déclarations de faits au début du programme même si ce n'est pas obligatoire.
- Un fait se termine toujours par un point « . »

Exemples :

- Henri IV est le père de Louis XIII se traduit par : **pere(henri4,louis13).** ou par : **pere('Henri4','Louis13').**
- Marie de Médicis est la mère de Henri IV se traduit par : **mere(marie-de-medicis,henri4).**

  ou **mere('Marie de Medicis', 'Henri4').**
- 0! = 1 se traduit par : **factorielle(0,1).**

Les Faits

Forme : **nomprédicat(arg$_1$,arg$_2$, …, arg$_n$).**

Les parenthèses et arguments sont optionnels

Exemples:

- univers.
- astre(terre).
- étoile(soleil).
- satellite(terre, soleil).
- satellite(lune, terre).
- capitale('France', ‘Paris').

Base de connaissance 2
Introduction des Règles

```
mange(loup, chevre).
cruel(loup) :- mange(loup, chevre).
carnivore(loup) :- mange(loup, chevre), animal(chevre).
```

```
?- cruel(loup).
true.
?- carnivore(poireau).
false.
?- carnivore(loup)
ERROR: carnivore/1: Undefined procedure: animal/1
```

Pas d'individu poireau => Pas d'évaluation du prédicat carnivore => false

Individu loup => Evaluation du prédicat carnivore => prédicat non défini: animal

Règles Prolog

- Enoncent la dépendance d'un prédicat par rapport à d'autres prédicats
- Concernent des catégories d'objets / faits
  - fait : enColere*(fido).*
  - règle : *Fido aboie si Fido est en colère* s'écrit :
    aboie(fido) :- enColere*(fido).*
- Le « **si** » s'écrit « **:-** » en Prolog et correspond à l'implication ⇒

  enColere*(fido)* ⇒ *aboie(fido).*

  s'écrit en Prolog : *aboie(fido) :-* enColere*(fido).*

Règles Prolog

- Un programme Prolog contient presque toujours des règles, cependant ce n'est pas une obligation.
- Si les faits sont les hypothèses de travail, les règles sont des relations qui permettent à partir de ces hypothèses d'établir de nouveaux faits par déduction

  (si on a démontré F1 et F1⇒F2 alors on a démontré F2).
  - Cette règle est appelée en logique le Modus Ponens

Règles Prolog

- Il peut y avoir plusieurs conditions derrière le « :- », séparées par des virgules ou des points-virgules
  - La virgule correspond à un ET logique (conjonction)
  - Le point-virgule correspond à un OU logique (disjonction)
  - Exemple :

    La relation telle que si on est le père du père ou de la mère de quelqu'un alors on est son grand-père se traduit par :

```
grandpere(xavier,yves) :- pere(xavier,joe), pere(joe,yves).
grandpere(xavier,yves) :- pere(xavier,joe), mere(joe,yves).
```

ou encore par :

```
grandpere(xavier,yves) :- pere(xavier,joe), (pere(joe,yves) ; mere(joe,yves)).
```

Base de connaissance 3
Introduction des Variables

```
mange(loup, chevre).
carnivore(X) :- mange(X, Y), animal(Y).
cruel(X) :- mange(X, _).
```

```
?- cruel(loup)
true
?- carnivore(loup)
ERROR: carnivore/1: Undefined procedure: animal/1
?- carnivore(poireau)
false
```

Constantes vs Variables

- Constantes: commencent par une minuscule ou entre apostrophes ou entiers ou flottants
- Variables : commencent par une majuscule ou par _
  - Servent à dénoter une catégorie d'objets
  - Les variables utilisées dans un fait ou une règle sont universellement quantifiées par Prolog, avec le quantificateur ∀ (« quel que soit » ou « pour tout »).
  - Les variables utilisées dans une requête sont existentiellement quantifiées par Prolog, avec le quantificateur ∃ (« existe-t-il »).

Variables

- Sont utilisées dans les règles afin de les généraliser :
  - *grandpere(xavier,yves) :- pere(xavier,joe), pere(joe,yves).*
  - grandpere(xavier,yves) :- pere(xavier,joe), mere(joe,yves).

  Généralisé par :
  - *grandpere(X,Y) :- pere(X,Z), (pere(Z,Y) ; mere(Z,Y))*
- Peuvent aussi être utilisées dans les faits

  Superman est plus fort que tout le monde se traduit par :

  plusfort(superman,X).

Variable anonyme

- Notée '_' (tiret bas)
- Représente un objet dont on ne souhaite pas connaître la valeur, c'est une variable muette.
- ?- invite(X,_).
  Prolog ne donnera que les valeurs pour X
- Attention, une variable commençant par '_' et de longueur >= 2 n'est pas muette par défaut en SWI-Prolog. Activation par :
  set_prolog_flag(toplevel_print_anon, false).

Exercice

- Exprimer en Prolog les propositions suivantes, identifier les objets, les faits, les règles
  - *la chèvre est un animal herbivore*
  - *le loup est un animal cruel*
  - *toute chose cruelle est carnivore*
  - *un animal carnivore mange de la viande et un animal herbivore mange de l'herbe*
  - *un animal carnivore mange des animaux herbivores*
  - *les carnivores et les herbivores boivent de l'eau*
  - *un animal consomme ce qu'il boit ou ce qu'il mange*
  - Question : y a-t-il un animal cruel et que consomme-t-il ?

Solution possible

la chèvre est un animal herbivore

le loup est un animal cruel

toute chose cruelle est carnivore

un animal carnivore mange de la viande et un animal herbivore mange de l'herbe

un animal carnivore mange des animaux herbivores

les carnivores et les herbivores boivent de l'eau

un animal consomme ce qu'il boit ou ce qu'il mange

Question : y a-t-il un animal cruel et que consomme-t-il ?

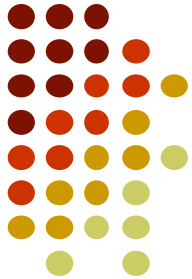
```
animal(chevre).
herbivore(chevre).
animal(loup).
cruel(loup).

carnivore(X) :- cruel(X).

mange(X,viande):- animal(X), carnivore(X).
mange(X,herbe):- animal(X), herbivore(X).
mange(X,Y):- animal(X), animal(Y), carnivore(X),herbivore(Y).

boit(X,eau):- animal(X), (carnivore(X); herbivore(X)).

consomme(X,Y) :- mange(X,Y); boit(X,Y).
?- animal(X), cruel(X), consomme(X,Y).
```

Partie 3
-
Mécanisme de Résolution

Résolution d’une requête

- Comment Prolog répond à une requête?
- 3 mécanismes principaux
  - Règle de généralisation
  - Règle d’instanciation
  - Règle de résolution

Règle de généralisation



- Sert à traiter les variables dans les requêtes

  Exemple: ?- cruel(X).

- Loi de généralisation: p(a) donc ∃ X tq: p(X)

  Exemple : cruel(loup) donc ∃ X tq: cruel(X)

  X = loup

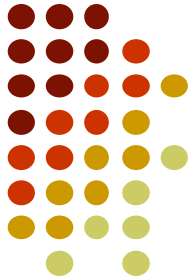
- Prolog cherche donc un fait qui est une instance de la requête. Il va essayer de remplacer les variables par des constantes.

Règle d’instanciation

- Sert à traiter les variables dans les faits = faits universels

  Exemple: plusMalin(arseneLupin,Y).
- Loi d’instanciation: ∀ X p(X) donc p(a)
- Exemple: plusMalin(arseneLupin,policier)
- Prolog remplace les variables des faits universels par toutes les constantes de la base de connaissance

Règle de résolution



- Sert à la déduction de nouveaux faits à partir d'anciens faits et des règles
- Exemple: cruel(X) :- tue(X,Y).
  tue(lion, gazelle).

  Permet de déduire cruel(lion).
- Modus Ponens Général:
  - F :- F1, F2,…, Fn
  - Pour démontrer F, il faut démontrer F1, et F2,…, et Fn.
- Règle de résolution permet d'appliquer le modus ponens

Représenter les Faits et Règles

- Forme générale : F :- F1, F2,…, Fn.
- C'est une clause de Horn*
- F est la tête de la clause
- F1, F2,…, Fn est appelée la queue de la clause
- Un fait = clause dont la queue est vide
  - aime(X, bach). « Pour tout X, X aime Bach. »
- Une règle = clause dont la queue est non vide.
  - a_un_salaire(X) :- employeur(_,X).

* Leur nom vient du logicien Alfred Horn qui a remarqué leur importance en 1951

Exercice

- Démontrer en utilisant les règles de généralisation, d'instanciation et le modus ponens que la requête a au moins une solution avec la base de connaissance suivante .

  (pas besoin d'appliquer en détail la règle de résolution)

```
aimeSkier(paul).
aEquipement(zoe).
neige(X).
damePiste(chamrousse).
ski(X) :- stationOuverte(Y), aEquipement(X).
stationOuverte(X) :- neige(X), damePiste(X).
```

```
?-ski(X).
```

Partie 4
-
Syntaxe Prolog

Atomes

- Chaîne de caractères commençant par une **lettre minuscule** et contenant des lettres, chiffres et le caractère de soulignement (_)
- Chaîne de caractères entre **guillemets simple** : 'Vincent', 'Léo Ferré'
  - Permet l'utilisation d'espaces
- Symboles spéciaux :

  =:= et =\= et ; et :-
- Certains de ces atomes ont une signification prédéfinie (opérateurs).

Nombres

- Prolog définit des entiers et des réels. Nous nous concentrerons sur les entiers ici.
- Entiers : `-4, 567, 0, -34`

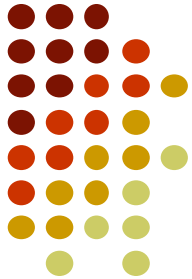
  `0xA2C2, 0xffff` (hexadécimal)

  `0o777, 0o11` (octal)

  `0b1111, 0b10` (binaire)
- Réels : `2.183720` (point décimal)

  `1.32e-12` (flottant)

Variables



- Chaîne de caractère démarrant par une **lettre majuscule** ou le **caractère de soulignement**
- Peut contenir des lettres, chiffres ou le caractère de soulignement
- Expl : X, Y, _var
- _ est appelée la **variable anonyme**

Termes complexes

- **Terme simple :** atome, nombre ou variable
- Un **terme complexe** est construit via un **foncteur** suivi d'une séquence d'**arguments**
- **Foncteur** (nom de prédicat) : **atome**
- **Argument** : **terme simple ou complexe**
- Les arguments sont placés après le foncteur, entre parenthèses et séparés par des virgules
- L'**arité** d'un foncteur est son nombre n d'arguments. Indiquée en suffixant par /n
- Exemple : donne/2 diffère de donne/3

Exercice

- Parmi les chaînes de caractère suivantes, lesquelles sont des atomes, lesquelles sont des variables et lesquelles ne sont ni l'un ni l'autre ?

1. `vINCENT`
2. `Football`
3. `variable23`
4. `Variable2000`
5. `petit_pain`
6. `'petit pain'`
7. `petit pain`
8. `'Jules'`
9. `_Jules`
10. `'_Jules'`

Partie 5
-
Unification

Unification : Idée de base

- Exemple :

  Base de connaissance : elfe(galadriel).

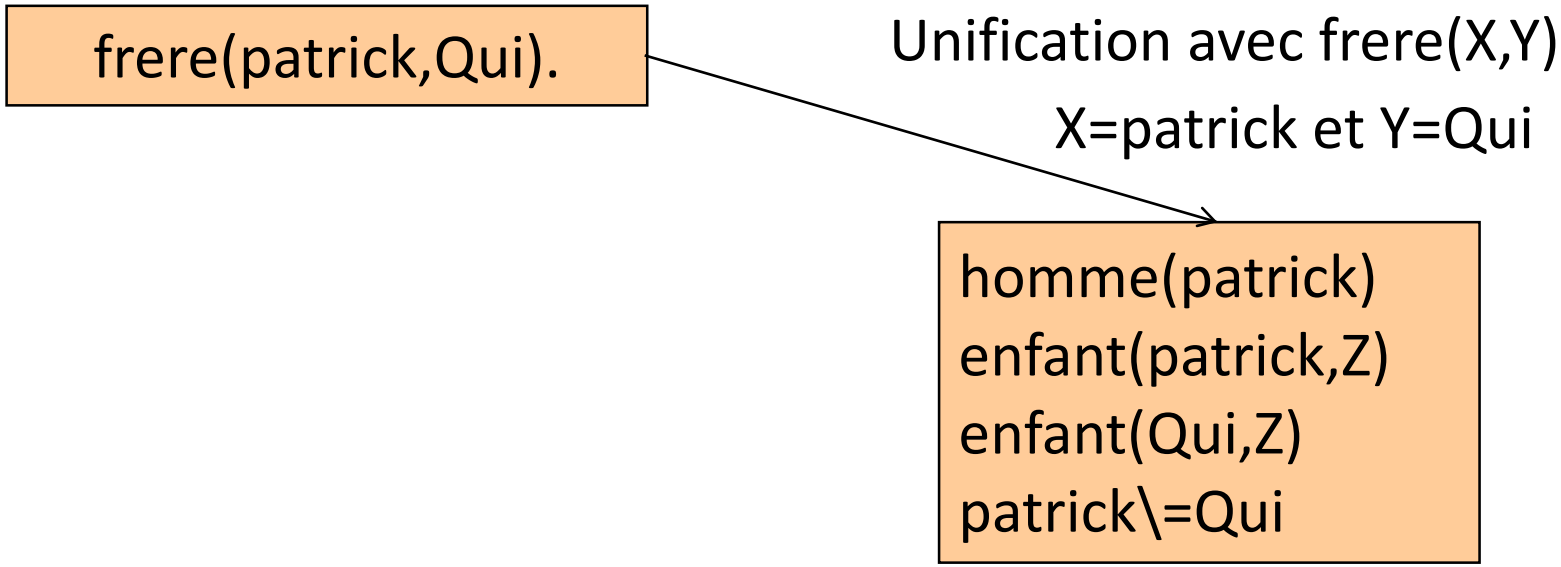
  Question :?- elfe(X).

- Comment fait Prolog pour dire que galadriel est une valeur possible pour X ?
- **Unification** : procédé par lequel on essaie de rendre deux formules identiques en donnant des valeurs aux variables qu'elles contiennent.
- Unification réussie : substitution possible

Unification dans les Clauses

- L'unification réussie dans la tête se propage aux clauses

frere(X,Y) :- homme(X), enfant(X,Z), enfant(Y,Z), X\=Y.



Définition Précise

- Si t1 et t2 sont des **constantes**, t1 et t2 unifient ssi ils sont le même atome ou le même nombre.
- Si t1 est une **variable** et t2 quelconque, t1 et t2 unifient ssi t1 est instanciable en t2.
- Si t1 et t2 sont des **termes complexes**, t1 et t2 unifient ssi :
  - Ils ont le même foncteur et même arité
  - Tous leurs arguments correspondants unifient
  - Les instanciations de variables sont compatibles

Conséquence

- L'unification peut réussir ou échouer.
  - *e(X)* et *e(2)* peuvent être unifiés.
  - e(X,X) et e(2,3) ne peuvent être unifiés
- Le résultat n'est pas forcément unique : ensemble de solutions
- Prédicat d'unification : « = ». Teste si l'unification a réussie.
  - Ecriture 1 : =(a,b).
  - Ecriture 2 : a=b.
- Unification optimiste : pere(X)=X unifie
- Unification pessimiste, n'unifie pas :

  unify_with_occurs_check(pere(X),X).

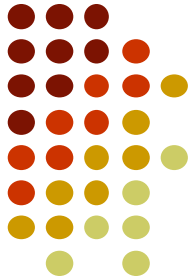
Exercice

- Dire si les expressions suivantes unifient et comment :
  - *?- sam = sam.*
  - *?- sam = lea.*
  - *?- 'sam' = sam.*
  - *?- '1' = 1 .*
  - *?- X = Y.*
  - *?- X = sam, X = lea.*
  - ?- k(s(g),Y) = k(X,t(k)).
  - ?- k(s(g), t(k)) = k(X,t(Y)).
  - ?- aime(X,X) = aime(sam,lea).

Démonstration Prolog (1)

- Pour répondre à une question, Prolog cherche à unifier la question, soit avec un fait, soit avec la tête d'une règle.
- Si l’unification avec la tête d'une règle réussit : substitution des variables de la queue par les valeurs correspondantes dans la tête.
- L'algorithme est appelé SLDNF résolution (Selective Linear Definite Negation as Failure)

Résultat Final



- Prolog continue à unifier jusqu'à vérification de la requête initiale
- Si l’unification échoue : Prolog répond faux (false)
- Si l’unification réussit : Prolog répond soit vrai (true), soit donne la valeur des variables (Exemple : X=lea) s'il y en avait dans la requête.
- On demande à Prolog de chercher d'autres valeurs en appuyant sur « ; » ou « Espace »
- A la fin de la démonstration, Prolog a construit une liste de couples valeur-variable

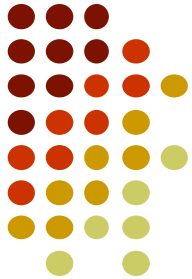
Parcours

- **Parcours de la base du haut vers le bas**
- **Parcours des clauses de gauche à droite**
- Quand il y a plusieurs manières d'unifier, on a un **point de choix**. Cas possibles :
  - Différentes règles définissant un même prédicat
  - Disjonction de prédicats dans une queue
  - Instanciation d'une variable (plusieurs possibilités)

Points de choix

- Prolog effectue des essais consécutifs dans l'ordre de déclaration des prédicats
- Cette séquence d'essais peut être représentée par un arbre: l'arbre de recherche
- Les nœuds de l'arbre sont les points de choix

Arbre de recherche (1)



- Pour résoudre une question, Prolog construit l'arbre de recherche de la question :
  - Racine de l'arbre : la question initiale
  - Nœuds : points de choix (formules à démontrer)
  - Passage d'un nœud vers son successeur en effectuant une unification

Arbre de recherche (2)

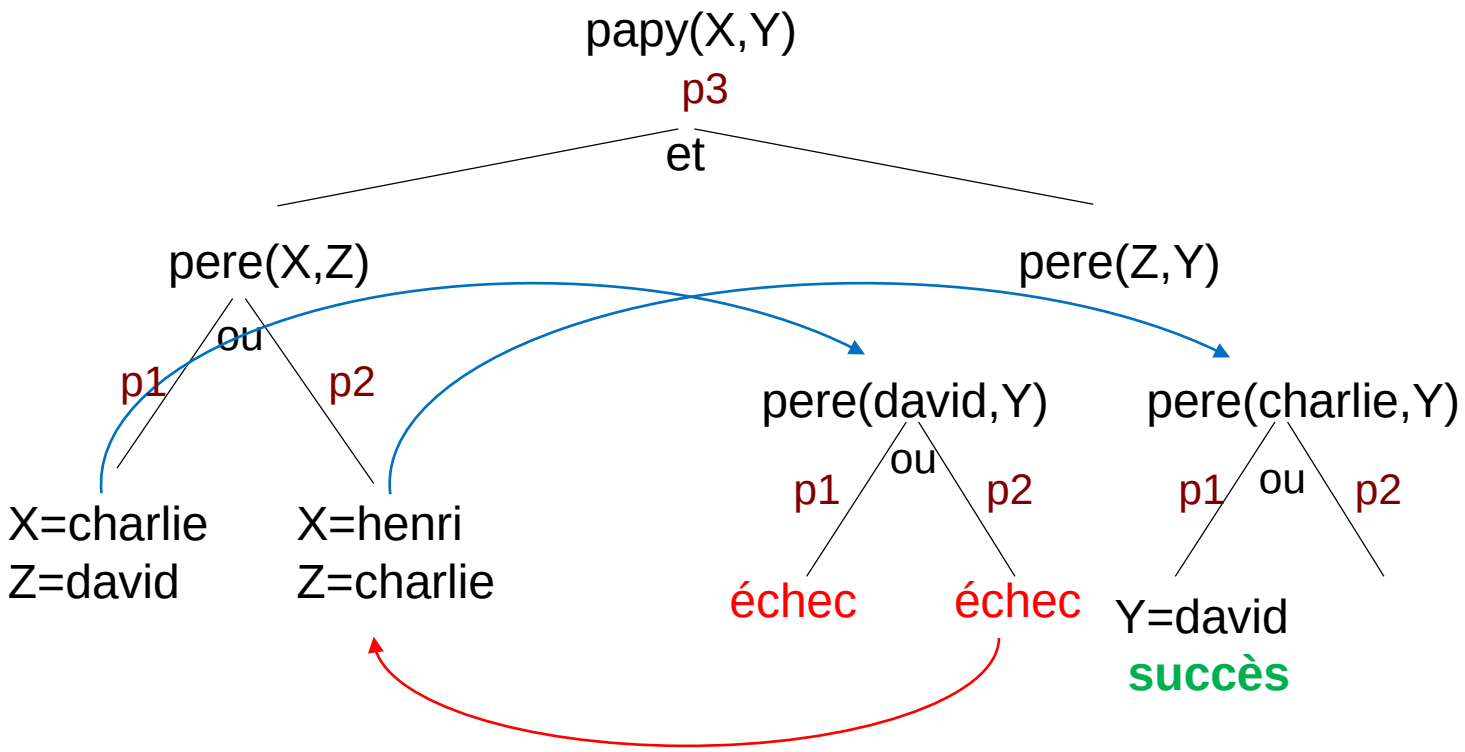
- Les nœuds sont démontrés de gauche à droite, dans l’ordre de déclaration dans la règle
- Nœuds d'échec : aucune règle ne permet de démontrer la première formule du nœud
- Nœuds de succès : ne contient plus aucune formule à démontrer, tout a été démontré et les éléments de solution sont trouvés en remontant vers la racine de l’arbre

Arbre de recherche (3)

Exemple de programme:

pere(charlie, david). (p1)
pere(henri, charlie). (p2)
papy(X,Y) :- pere(X,Z), pere(Z,Y). (p3)

Appel du programme : papy(X,Y).



Stratégie de Prolog

- Parcours en profondeur d’abord
- nœud de succès : c’est une solution, Prolog l’affiche et peut chercher d’autres solutions
- nœud d'échec : remontée dans l’arbre jusqu'à un point de choix possédant des branches non explorées
- Puis parcours de gauche à droite dans l'ordre des conditions

Backtracking

- Remontée dans l'arbre jusqu'au point de choix : **backtrack**. Si un tel nœud de choix n'existe pas, la démonstration est terminée, il n'y a pas d'autre solution.
- **Bactrack :** toutes les unifications faites depuis le dernier point de choix sont défaites
- Possibilité de branche infinie et donc de recherche sans terminaison… Attention à :
  - L'ordre des littéraux dans la queue de clause
  - L'ordre des clauses

Exemple de démonstration

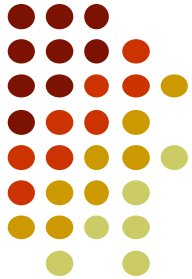
planete(X) :- astre(X), etoile(Y), satellite(X,Y).

faits:

astre(lune).
astre(terre).
astre(soleil).
astre(venus).
satellite(venus, soleil).
satellite(lune, terre).
satellite(terre, soleil).
étoile(vega).
étoile(soleil).

?- planete(X).

Exemple de démonstration

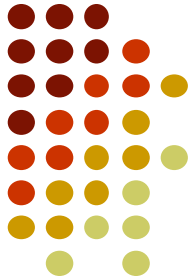


planete(X) :- astre(lune), etoile(Y), satellite(lune,Y).

faits:

astre(lune).
astre(terre).
astre(soleil).
astre(venus).
satellite(venus, soleil).
satellite(lune, terre).
satellite(terre, soleil).
étoile(vega).
étoile(soleil).

Exemple de démonstration



planete(X) :- astre(lune), etoile(vega), satellite(lune, vega).

faits: échec !

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

Exemple de démonstration

planete(X) :- astre(lune), etoile(soleil), satellite(lune, soleil).

faits: backtrack échec !

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

Exemple de démonstration

planete(X) :- astre(terre), etoile(Y), satellite(terre, Y).

faits: backtrack

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

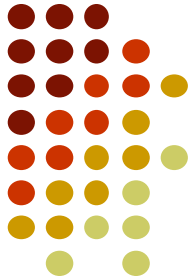
satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

Exemple de démonstration



planete(X) :- astre(terre), etoile(vega), satellite(terre, vega).

faits: échec !

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

Exemple de démonstration

planete(X) :- astre(terre), etoile(soleil), satellite(terre, soleil).

faits:

backtrack

true !

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

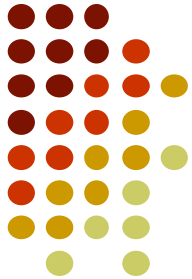
étoile(vega).

étoile(soleil).

X = terre,

Y = soleil

Exemple de démonstration



planete(X) :- astre(soleil), etoile(Y), satellite(soleil, Y).

faits: backtrack ..... échec !

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

Exemple de démonstration

planete(X) :- astre(venus), etoile(Y), satellite(venus, Y).

faits: backtrack

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

Exemple de démonstration

planete(X) :- astre(venus), etoile(vega), satellite(venus, vega).

faits: échec !

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

Exemple de démonstration

planete(X) :- astre(venus), etoile(soleil), satellite(venus,soleil).

faits:

astre(lune).

astre(terre).

astre(soleil).

astre(venus).

satellite(venus, soleil).

satellite(lune, terre).

satellite(terre, soleil).

étoile(vega).

étoile(soleil).

true !

X = venus,

Y = soleil

Exercice

- Soit la base de connaissance et la requête suivante. Construire l'arbre de recherche Prolog.

```
rectangle(a).
rectangle(b).
losange(b).
carre(X) :- rectangle(X),losange(X).
```

```
?- carre(X).
```