Programmation Logique

- **Objectif** : connaitre un nouveau paradigme de programmation, la programmation logique et apprendre à programmer en Prolog

- **Intérêts** :
  - culture générale informatique
  - Prolog est un langage largement utilisé en IA, en particulier dans le traitement des langues
  - Prolog connaît aussi des applications dans l'industrie (systèmes experts, traitement des langues, bases de données déductives, ...)

- **Références** :
  - *Logique, réduction, résolution*, R. Lalement, **`Masson`**
  - *Prolog*, F. Gannesini, H. Kanoui, R. Pasero, M. van Caneghem, **`InterEditions`**
  - *Intelligence Artificielle & Informatique Théorique*, J.M. Alliot, T. Schiex, P. Brissot, F. Garcia, **`Cepadues`**

Historique de Prolog (1/2)

- 1970 Utiliser la logique comme **langage de programmation**
  Clauses de Horn (R. Kowalski)
  Q-systèmes (A. Colmerauer)

- 1972 Premier interprète Prolog (A. Colmerauer et P. Roussel - Université d'Aix-Marseille)

- 1977 Premier compilateur Prolog (D. H. D. Warren - Université d'Édimbourg)

- 1980 Prolog est choisi comme langage de base pour le projet japonais de "5e génération" (IA)

- 1990 Prolog évolue vers la **Programmation par Contraintes** (Prolog IV)

Historique de Prolog (2/2)

- Première implantation de Prolog due à l'équipe de Colmerauer et Roussel, à Marseille-Luminy en 1972. Première interprétation logique donnée par Kowalski en 1974. Voir pour plus de détails la page de Colmerauer ***http://www.lim.univ-mrs.fr/~colmer/tablematiere.html***
- Deux syntaxes ont été proposées, celle de Marseille et celle d'Edimbourg (la plus courante, proposée par Warren en 1974).
- Prolog est plutôt interprété, mais il existe des versions compilées
- Nombreuses implémentations : *SWI-Prolog, GNU-Prolog, SICTus-Prolog, Prolog II, Visual-Prolog, Quintus-Prolog*, ...

GNU-Prolog

- Nous utiliserons **GNU-Prolog** qui possède une syntaxe Edimbourg.
- GNU-Prolog est développé à l'INRIA et est libre
- GNU-Prolog offre un compilateur en deux parties :
  - compilation en byte-code pour exécution sur la machine virtuelle incluse (Warren Abstract Machine)
  - compilation en code natif (exécution plus rapide)
- GNU-Prolog offre pour l'exécution
  - un interpréteur qui travaille avec le byte-code
  - la possibilité de générer un exécutable pour application stand-alone
- Plus d'info sur *http://gnu-prolog.inria.fr/*

Paradigme de Prolog (1/4)

Prolog = Programmation Logique

- un programme consiste en la déclaration d'une série de faits et de règles de déduction.

```
homme('roger').
homme('robert').
femme('gertrude').
femme('germaine').

humain(X) :- homme(X).
humain(X) :- femme(X).
```

```
enfant_de('roger','gertrude').
enfant_de('gertrude','germaine').

petit_enfant_de(X,Y) :- enfant_de(X,Z),enfant_de(Z,Y).
```


Paradigme de Prolog (2/4)

- L'exécution du programme consiste à prouver un théorème. Le programme répond si le théorème peut être prouvé ou non à partir des déclarations.

Paradigme de Prolog (3/4)

- La programmation logique est un paradigme particulier, ou les calculs se font sous forme de preuves logiques. A distinguer de :
  - *la programmation procédurale* : adaptée aux ordinateurs, épouse leur fonctionnement itératif (C, Pascal, ADA, Java, ...)
  - *la programmation fonctionnelle* : description de traitements par des fonctions, la composition de fonctions et la récursivité (Caml, Lisp, Scheme,...)
- **Prolog = programmation déclarative** (Prolog, Lisp,Caml,...) , et non impérative (C,Java,...)
  - Le programme ne décrit pas des instructions à exécuter, mais une situation correspondant à un problème à résoudre
- la programmation déclarative n'est pas incompatible avec la structuration objet (Prolog++, LogTalk)

Paradigme de Prolog (4/4)

- Le problème est décrit sous forme de formules logiques. L'interpréteur Prolog se charge de le résoudre par des mécanismes logiques.
  - on ne dit pas à la machine ce qu'elle doit faire, comme en programmation impérative, on ne fait que lui *décrire le problème* sous forme logique
  - on spécifie les *propriétés du résultat* du programme et non pas le processus pour arriver à ce résultat
  - Prolog est naturellement récursif
  - Prolog utilise beaucoup les listes
- Pour des raisons calculatoires, les formules exprimées en Prolog sont restreintes aux clauses de Horn
- Revenons sur le formalisme logique ...

Calcul propositionnel

- **Syntaxe** :
  - un ensemble de variables propositionnelles, appellées aussi atomes (notés souvent a,b,c,...)
  - formules propositionnelles formées à partir des atomes, des constantes logiques 0 et 1 et des connecteurs ¬, ⇒, ∨ et ∧.
- Formules bien formées :
  - les atomes sont des formules bien formées
  - si f et g sont des formules bien formées et x est une variable, alors ¬f, f ∨ g, f ∧ g, f ⇒ g sont des formules bien formées
- *Exemples : f ⇒ g ∨ h est bien formée, f ⇒ ∨ h ne l'est pas*

Calcul des prédicats

- **Syntaxe** :
  - un ensemble de variables (notées souvent x,y,z,...)
  - un ensemble de constantes (notées souvent a,b,c,...)
  - un ensemble de fonctions d'arités strictement positives (les constantes peuvent être vues comme des fonctions d'arité 0)
  - termes : toute variable ou constante est un terme, si f est un symbole de fonction d'arité n, et t1, .. , tn sont des termes, f(t1,..,tn) est un terme
  - un ensemble de prédicats (relations) d'arités positives ou nulles
  - atomes : expressions de la forme p(t1,..,tn) où p est un symbole de prédicat n-aire et t1,.., tn sont des termes
  - formules du premier ordre : formées à partir des atomes, des constantes logiques 0 et 1 et clos par les connecteurs ¬, ⇒, ∨, ∧, ∀ et ∃. Les quantificateurs ∀ et ∃ ne portent que sur des variables.
- *Exemple : (homme(Socrate) ∧ (∀x homme(x) ⇒ mortel(x))) ⇒ mortel(Socrate)*
  - *variables : Socrate*
  - *fonctions :*
  - *constantes : Socrate*
  - *prédicats : homme, mortel*
  - *atomes : homme(Socrate), homme(x), mortel(x), mortel(Socrate)*

Formules bien formées

- *Remarque : les parenthèses sont parfois considérées comme faisant partie du vocabulaire logique*

- Formules bien formées :
  - les atomes sont des formules bien formées
  - si f et g sont des formules bien formées et x est une variable, alors ¬f, f ∨ g, f ∧ g, f ⇒ g, ∀x f et ∃x f sont des formules bien formées

- *Exemples :*
  - *(homme(Socrate) ∧ (∀x homme(x) ⇒ mortel(x))) ⇒ mortel(Socrate) est bien formée*
  - *(homme(Socrate) ∧ ⇒ (∀x homme(x) ⇒ mortel(x))) ⇒ mortel(Socrate) est mal formée*

Variables liées et libres

- On distingue les variables liées (quantifiées) et les variables libres (non liées par un quantificateur existentiel ou universel)
  - dans ∀x x ∧ ¬y ⇒ z, x est liée, y et z sont libres

- Une formule sans variable libre est dite formule close

- La logique des prédicats (ou logique du premier ordre) porte sur les formules ainsi définies
  - la logique des propositions (logique d'ordre 0) ne prend en compte que les variables et les connecteurs
  - la logique d'ordre 2 autorise la quantification des fonctions et des prédicats

Formules universelles

- Un littéral est un atome (littéral positif) ou la négation d'un atome (littéral négatif)
- Une formule où les quantificateurs ($\forall$ et $\exists$) apparaissent tous en tête est dite prénexe
- Une formule universelle (resp. existentielle) est une formule prénexe close dont les seuls quantificateurs sont universels (resp. existentiels)
- *Exemples* : - $\forall x\ \forall y\ \forall z$ *enfant(x,y)* $\wedge$ *enfant(y,z)* $\Rightarrow$ *petit_enfant(x,z) est une formule universelle*
  - $\forall x$ *enfant(x,y)* $\exists z$ *enfant(x,z) n'est pas une formule universelle, ni existentielle*

Clauses

- Une clause est une formule universelle dont les sous-formules non quantifiées sont des littéraux ou des disjonctions de littéraux. Une clause (en calcul des prédicats) a donc pour forme générale

  $$\forall x_1, .. , x_n\ a_1 \vee .. \vee a_p \vee \neg b_1 \vee .. \vee \neg b_q$$

  où les a_i et b_i sont des atomes
- *Exemple* : $\forall x$ *homme(x)* $\vee$ *femme(x)* $\vee \neg$*humain(x). Cette clause correspond à la formule* $\forall x$ *humain(x)* $\Rightarrow$ *(homme(x)* $\vee$ *femme(x))*
- La clause vide, notée $\square$, ne contient aucun littéral : elle est toujours fausse

Clauses de Horn

- Une clause de Horn est une clause comportant au plus un littéral positif. Une clause de Horn a donc pour forme générale

  $\forall x_1, .. , x_n \; a \vee \neg b_1 \vee .. \vee \neg b_q$

  où les B_i sont des atomes. Une clause de Horn peut également s'écrire : $\forall x_1, .. , x_n \;\; b_1 \wedge .. \wedge b_q \Rightarrow a$

- *Exemple* : $\forall x,y,z$ pere(x,y) $\vee$ pere(y,z) $\vee \neg$grand-pere(x,z)

- Les seules formules pouvant être écrites en Prolog sont des clauses de Horn

Clauses de Horn et Prolog (1/3)

- En Prolog, on peut écrire des clauses de Horn avec un littéral positif et au moins un littéral négatif. Ce sont des ***règles***.

- *Exemple* : $\forall x \; \forall y \; \forall z$ *enfant(x,y)* $\wedge$ *enfant(y,z)* $\Rightarrow$ *petit_enfant(x,z) est équivalent à* $\forall x \; \forall y \; \forall z \neg$ *enfant(x,y)* $\vee \; \neg$ *enfant(y,z)* $\vee$ *petit_enfant(x,z) s'écrit en Prolog*

```
petit_enfant_de(X,Z) :- enfant_de(X,Y),enfant_de(Y,Z).
```

- En Prolog, on peut écrire des clauses de Horn sans littéral négatif (clauses de Horn positives), mais avec seulement un littéral positif. Ce sont des ***faits***.

- *Exemple* :

```
enfant_de('roger','gertrude').
enfant_de('gertrude','germaine').
```

Clauses de Horn et Prolog (2/3)

- En Prolog, on peut écrire des clauses de Horn sans littéral positif, (clauses de Horn négatives), mais avec seulement des littéraux négatifs. Ce sont des ***buts***.

- *Exemple : petit_enfant_de(B,'germaine')* $\wedge$ neveu(X,Y) *est un but constitué de deux littéraux positifs.*

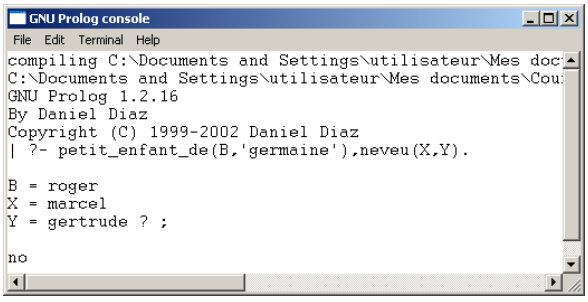
  En fait, les buts sont écrits en Prolog comme des littéraux positifs, mais Prolog utilise implicitement pour raisonner la négation du but

  La négation d'un but constitué par une conjonction de littéraux positifs $a_1 \wedge .. \wedge a_n$ *va donc être manipulée par Prolog sous la forme d'une clause de Horn négative* $\neg a_1 \vee .. \vee \neg a_n$

```
?- petit_enfant_de(B,'germaine'),neveu(X,Y).
```


Clauses de Horn et Prolog (3/3)

- Un programme Prolog est donc un ensemble de faits et de règles, le tout sous forme de clauses de Horn. L'interpréteur Prolog se sert de cette ***base de connaissances*** pour répondre à des buts.



Syntaxe Prolog

- La syntaxe Prolog est très simple comparée à celle des langages impératifs
  - on se limite à la description d'une situation, pas de description des traitements
  - types de données limités : nombres, chaines, identificateurs
  - structure de liste
- Un mécanisme intégré à l'interpréteur permet de raisonner sur les bases de connaissances exprimées sous forme de clauses de Horn
- Des commandes particulières permettent de contrôler le mécanisme de raisonnement

Constantes

- *Les constantes* :
  - Nombres entiers ou flottants : <nombre> ::= {-}[0-9]+{.[0-9]+}
  - Booléens : <booleen> ::= true | false
  - Identificateurs alphanumériques débutant par une minuscule ou encadrés par des apostrophes : <ident> ::= '[A-Za-z0-9_]*' | [a-z] [A-Za-z0-9_]*
  - Chaines de caractères entre guillemets : <chaine> ::= "[A-Za-z0-9_\$@...]*"
- *Exemples :*
  - *jojo, 'jojo', 'Roger' sont des identificateurs*
  - *"jojo" est une chaine de caractères*

Variables

- *Les variables* :
  - Identifiées par des chaînes de caractères débutant par une majuscule ou _
  - <variable> ::= [A-Z] [A-Za-z0-9_]* | _ [A-Za-z0-9_]*
  - Variable indéterminée : _
- *Exemples de variables* : **`X, Y, Variable, Variable_Prolog, _X23`**
- La variable indéterminée est utilisée quand sa valeur n'a pas d'intérêt pour l'utilisateur
- Une variable Prolog est une variable au sens mathématique, elle représente toujours le même objet (inconnu) et ne change pas de valeur dans la même "branche de démonstration"

Prédicats

- *Les prédicats* :
  - identifiés par des chaînes de caractères débutant par une minuscule
  - possède une arité
  - <prédicat> ::= <nom_prédicat>(<liste_paramètres>)
  - <nom_prédicat> ::= [a-z] [A-Za-z0-9_]*
  - <liste_paramètres> ::= <paramètre>{,<paramètre>}*
  - <paramètre> ::= <ident> | <nombre> | <chaine> | <liste>
- La virgule représente le ET
- *Exemples de prédicats* :

```
est_un_chouette_langage(prolog),
sèche_le_cours(Etudiant_X,cours_Prolog),
triangle_rectangle(A,B,C)
```

Faits

- *Les faits* :
  - un fait est un prédicat avec des constantes comme paramètres
  - <fait> ::= <prédicat>
- *Exemples de faits* :
  - `est_un_chouette_langage(prolog)`
  - `aime('Juliette','Roméo')`

Règles

- *Les règles* :
  - une règle est une clause de Horn constituée d'une tête de règle (le littéral positif) et une queue de règle (la conjonction des littéraux négatifs)
  - <règle> ::= <prédicat> :- <queue>.
  - <queue> ::= <prédicat>{,<prédicat>}*
- *Exemples de prédicats* :
  - `neveu(X,Y) :- fils(X,Z), frère_ou_soeur(Y,Z).`
- Ne pas oublier le **point** à la fin pour clore la règle

Les ET et les OU

- La partie "hypothèse" d'une clause de Horn est une conjonction (la virgule étant un ET) :

```
grand_pere(X,Y) :- pere(X,Z),pere(Z,Y).
```

- On veut aussi pouvoir exprimer des disjonctions :
  - première façon : on écrit plusieurs clauses

```
grand_pere(X,Y) :- pere(X,Z),pere(Z,Y).
grand_pere(X,Y) :- pere(X,Z),mere(Z,Y).
```

  - autre façon (pas toujours possible) le point-virgule représente le OU

```
humain(X) :- homme(X);femme(X).
```

  - Il vaut mieux séparer les différentes parties d'une règle

Rôle des prédicats

- Un prédicat exprime une relation entre objets ou/et une propriété d'un objet :

- *Exemples :*
  - **suit_cours(etudiantX,cours_Prolog)**
  - **numEtudiant(etudiantX,345667)**

- Plusieurs possibilités pour établir une même relation :
  - *Hélène est une fille de cinq ans* peut s 'exprimer par *:*
    - *fille(hélène). age(hélène,5).*
    - *fille_age(hélène, 5).*
    - *fille_5ans(hélène).*

- La modélisation dépend des raisonnements qu'on veut mener

Programme Prolog

- *Programme Prolog* : c'est une liste de faits et de règles
- On regroupe généralement les règles qui ont le même prédicat de tête et les faits qui portent sur le même prédicat

```
enfant_de('roger','gertrude').
enfant_de('gertrude','germaine').
enfant_de('marcel','robert').
frère_ou_soeur('gertrude','robert').

neveu(X,Y) :- fils(X,Z),  frère_ou_soeur(Y,Z).
petit_enfant_de(X,Y) :- enfant_de(X,Z),enfant_de(Z,Y).
```

- *Commentaires* : **`/* blabla */`** ou **`% blabla`**

Portée des variables

- Attention : l'ordre de déclaration des règles peut jouer! (on verra plus loin comment)
- Portée des constantes et variables :
  - constante : porte sur tout le programme (tout le code chargé dans l'interpréteur)
  - variable : n'existe que dans la clause où elle se trouve

```
grand_pere(X,Y) :- pere(X,Z),pere(Z,Y).
grand_pere(X,Y) :- pere(X,Z),mere(Z,Y).
```

Ces deux variables sont différentes

Ces deux variables sont les mêmes (ont la même valeur)

Variables anonymées

- Les variables anonymées débutent par _
- Elles sont utilisées pour des variables dont on ne désire pas connaitre la valeur
- *Exemple* :

```
grand_pere(X,Y) :- pere(X,_Z),pere(_Z,Y).
grand_pere(X,Y) :- pere(X,_Z),mere(_Z,Y).
```


Exécution

- On charge du code Prolog dans l'interpréteur en utilisant le prédicat unaire **`consult`**
  - **`consult('nom_fichier').`**
- Possibilité de charger du code de plusieurs fichiers
  - des règles ou faits portant sur le même prédicat peuvent être déclarés dans des fichiers différents
- Une fois les programmes chargés et donc la base de faits et de règles constituée, on peut poser des questions

```
enfant_de('roger','gertrude').
enfant_de('gertrude','germaine').
enfant_de('marcel','robert').
frère_ou_soeur('gertrude','robert').

neveu(X,Y) :- enfant_de(X,Z),  frère_ou_soeur(Y,Z).
petit_enfant_de(X,Y) :- enfant_de(X,Z),enfant_de(Z,Y).
```

Démonstration Prolog

- A partir d 'un programme, on peut poser des questions.
- Le mécanisme Prolog utilise les faits et règles chargés dans l'interpréteur pour répondre aux questions
- On parle de **résolution de question** ou de **démonstration de question**.
- Comment Prolog opère t-il??