3-Prolog

Intelligence Artificielle

Enseignant : Dr. Ahmed MAALEL

maalel.ahmed@gmail.com

Avant de commencer ..

Vos Questions

Le langage PROLOG

- Développer au début des années 70
- Un programme logique permet de représenter des **faits** (p.ex. Socrate est un homme) et des **règles** (p.ex. Tout homme est mortel)
- Un **PRO**gramme **LOG**ique est un ensemble de phrases qui ont en général la forme suivante :

  - $P_1 \wedge p_2 \wedge \ldots \wedge P_n \rightarrow C$
  - ce qui peut se lire, déclarativement « C est vrai si P_1 et p_2 et p_3 et … sont vrais »

Un programme logique est utilisé par un système pour répondre à des questions posées par l'utilisateur

Syntaxe du langage PROLOG

- Basée sur la logique des prédicats
- Représentation des opérateurs logiques :

Langage	**conjonction**	**disjonction**	**implication**	**négation**
Français	et	ou	si, alors	non
Logique des prédicats	$\wedge$	$\vee$	$\rightarrow$	$\neg$
PROLOG	,	;	:-	not

Syntaxe du langage PROLOG

Dans un programme PROLOG :

- Les noms des constantes commencent par une minuscule
- Les noms des variables commencent par une majuscule
- La conjonction d'expressions s'effectue avec une virgule
- L' **implication** s'effectue avec l'opérateur « **:-** » et sa représentation est inversée

  - Exemple : conséquent :- condition1, condition2.

– Toute expression, appelée clause, se termine par un point

  - Exemple : Parent(pierre, jean).

– Les commentaires sont précédés du symbole «% ».

  - Exemple : %ceci est un commentaire

Du langage naturel à PROLOG

- Il n'existe pas d'algorithme universel pour transformer une expression en langage naturel en une autre expression en un langage logique

- Le but est d'opérationnaliser la connaissance de résolution de problèmes tout en minimisant la perte d'informations

Du langage naturel à PROLOG

– En langage naturel :

Socrate est un homme. Tout homme est mortel.	Socrate est-il mortel? Oui.

– En PROLOG

Programme :	À la console :
homme(socrate). mortel(X):- homme (X).	?-mortel(socrate). yes

La console PROLOG

- Un environnement de programmation Prolog comprend une console, où l'utilisateur peut poser des questions (lancer des **buts**, faire des **appels**, des **requêtes**).
- Un appel à la console doit être précédé de l'opérateur **« ?- »**

La console PROLOG

Si le système peut prouver que le but est vrai et que ce dernier ne contient aucune variable, alors il répond « **yes** » à la console, sinon, il répond « **no** »

Exemple :

- Programme logique :

```
homme(socrate). %Socrate est un homme
```

- Console PROLOG :

```
?-homme(socrate). %Est-ce que Socrate est un homme?
yes
?- homme(platon). %Est-ce que Platon est un homme?
no
```

La console PROLOG

Si le système peut prouver que le but est vrai et que ce dernier contient des variables, alors il affiche à la console les substitutions avec lesquelles le but est vrai (i.e. la réponse à la question est oui si les variables de l'expression prennent certaines valeurs), sinon, il répond **« no »**.

Exemple :

•Programme logique :

```
homme(socrate). %Socrate est un homme
```

•Console PROLOG :

```
?-homme(X). %Qui est un homme?
X=socrate
```

La console PROLOG

- Si le but peut être vrai pour différentes substitutions, l'utilisateur peut voir les différents résultats en appuyant sur « ; » après chaque succès jusqu'à ce que le système affiche « **no** », signifiant qu'il n'existe plus de substitutions avec lesquelles le but est vrai.

- Si le système rencontre un problème en tentant de prouver un but (p.ex. manque d'espace mémoire, prédicat non défini), alors il affiche un **message d'erreur**.

Unification et substitution

Unifier deux clauses : déterminer avec quelles substitutions de variables, les deux clauses sont identiques

- Opérateur d'unification : « = »
- Opérateur de substitution : « / »
- **X/a** signifie « substituer la variable X par la constante a »
- PROLOG utilise le symbole = pour représenter la substitution, en réponse à une requête contenant des variables

Unification et substitution

- Exemples à la console Prolog :

1. ?- homme(X) = homme(socrate).

   X = socrate

2. ?- a(X, p(X), q(Y)) = a(2,Z,q(3)).

   X = 2, Y = 3, Z = p(2)

- Exemple en Prolog (déclaratif) :

```
...
A=1,
A1is A+1,
...
```

La variable A, une fois unifiée, ne peut être désunifiée car il est logiquement impossible que l'expression « A=1, A=2 » soit vraie. On utilisera plutôt une nouvelle variable.

La résolution PROLOG et le retour-arrière

Un programme prolog est composé de clauses exprimant des faits et des règles.

•**La résolution prolog** est le processus permettant de répondre à une question posée à la console

•***Exemples :***

- Faits (ou affirmations) :

voisin(youssef,ibrahim).

- Règles (d'inférence ou de déductions) :

citoyen_naissance(X, Y) :- citoyen(X, Y), née(X, Y).

- Questions (résolution de problèmes) :

?- Voisin(youssef,ibrahim).

La résolution PROLOG et le retour-arrière

- **Un arbre de résolution Prolog** est la trace du raisonnement suivi pour résoudre un but donné.
- Chaque **nœud** est un ensemble de buts ou de sous-buts.
- Chaque **branche** représente une unification entre le sous-but le plus à gauche du nœud parent et un fait ou la tête d'une règle dans le programme logique.

  → *Chaque branche est étiquetée par la règle ou le fait utilisé dans l'unification ainsi que, s'il y a lieu, les substitutions de variables nécessaires pour que l'unification réussisse.*

La résolution PROLOG et le retour-arrière

Un arbre de résolution Prolog (suite)

- Chaque **feuille** représente soit un succès, soit un échec
  - Si l'unification avec une clause donnée est impossible, la branche correspondante est étiquetée avec la raison de l'échec et le nœud-fils est « Échec »
  - Si l'unification avec une clause donnée est un succès et qu'il ne reste plus de sous-buts à résoudre, alors le nœud-fils est « succès ».
- L'arbre est construit de haut en bas et de gauche à droite.
- Le **retour-arrière** : considérer d'autres alternatives lorsque l'unification échoue entre un sous-but et une clause.

La résolution PROLOG et le retour-arrière

Construire un arbre de résolution :

- numéroter les clauses
- construire l'arbre à partir de la question initiale
- indiquer pour chaque branche les numéros des clauses utilisées et les substitutions de variables s'il y a lieu
- préciser si succès ou échec aux feuilles de l'arbre

La résolution PROLOG et le retour-arrière

Exemple :

–Programme logique :

```
parent(pierre,sebastien).
parent(nathalie, sebastien).
parent(marie, nathalie).
parent(robert, nathalie).
grand_parent(G, E) :- parent(G, P), parent(P, E).
```

–Question : « Qui sont les grands-parents de Sébastien? »

```
?- grand_parent(X, sebastien).
X=marie ;
X=robert ;
No
```

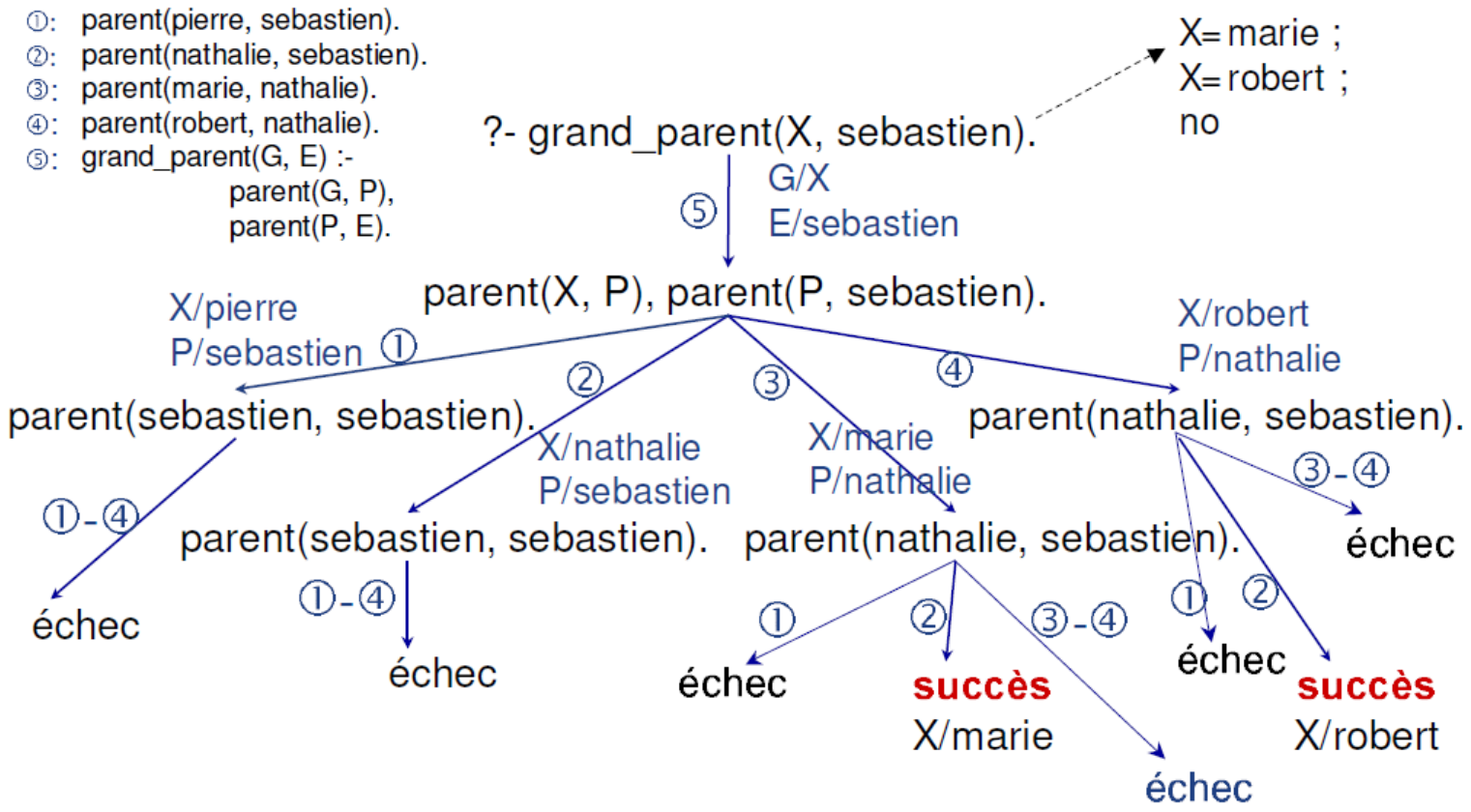
–Arbre de résolution ?

La résolution PROLOG et le retour-arrière

Exemple

→ **Arbre de résolution :**

1. parent(pierre,sebastien).
2. parent(nathalie, sebastien).
3. parent(marie, nathalie).
4. parent(robert, nathalie).
5. grand_parent(G, E) :- parent(G, P), parent(P, E).


①: parent(pierre, sebastien).
②: parent(nathalie, sebastien).
③: parent(marie, nathalie).
④: parent(robert, nathalie).
⑤: grand_parent(G, E) :-
parent(G, P),
parent(P, E).
?- grand_parent(X, sebastien).
X= marie ;
X= robert ;
no
⑤
G/X
E/sebastien
parent(X, P), parent(P, sebastien).
X/pierre
P/sebastien ①
②
③
④
X/robert
P/nathalie
parent(sebastien, sebastien).
X/nathalie
P/sebastien
X/marie
P/nathalie
parent(nathalie, sebastien).
①-④
③-④
échec
parent(sebastien, sebastien).
parent(nathalie, sebastien).
échec
①-④
①
②
③-④
①
②
échec
échec
succès
X/marie
échec
échec
succès
X/robert

Définir un prédicat de manière récursive en Prolog

Un prédicat récursif : 2 composants

1) Condition(s) d'arrêt
   - État final
   - Situation dans laquelle le but est atteint
2) Condition(s) récursive(s)
   - État intermédiaire
   - Le prédicat fait appel à lui-même

Définir un prédicat de manière récursive en Prolog

Exemple : la factorielle d'un nombre

–Le prédicat fact(X,F) est vrai si la factorielle de X est F

```
%clause d'arrêt : la factorielle de 0 vaut 1
fact(0,1).
%clause récursive : la factorielle F de N vaut N
%multiplié par la factorielle de N-1
fact(N,F):- N>0, N1 is N-1, fact(N1,F1), F is N* F1.
```

Place à la pratique