Université Akli Mohand Oulhadj – BOUIRA
Faculté des Sciences et Sciences Appliquées
Département d'Informatique

Prolog
Concepts de base

COURS DE PROLOG: INTELLIGENCE ARTIFICIELLE (IA)

Dr Z. Bouzidi
Zair.bouzidi@univ-bejaia.dz

2022 / 2023

PREMIERS PAS EN PROLOG

- Fonctionnement
- Utilisation pour des bases de connaissances
- Listes

PROGRAMMATION LOGIQUE

- **Origines :**
  - 1970, Marseille, Colmerauer
  - Edimbourg, Warren
- **Bibliographie**
  - L. Sterling, E. Shapiro, L'art de Prolog, Masson
  - Clocksin, Mellish, Programmer en Prolog, Eyrolles

LE LANGAGE PROLOG

- Langage d'expression des connaissances fondé sur le **langage des prédicats du premier ordre**
- **Programmation déclarative :**
  - L'utilisateur définit une **base de connaissances**
  - **L'interpréteur Prolog** utilise cette base de **connaissances** pour **répondre** à des **questions**

Constantes & Variables

- **Constantes**
  - **Nombres** : 12, 3.5
  - **Atomes**
    - Chaînes de caractères commençant par une minuscule
    - Chaînes de caractères entre " "
    - Liste vide []
- **Variables**
  - Chaînes de caractères commençant par une majuscule
  - Chaînes de caractères commençant par _
  - La variable « indéterminée » : _

TROIS SORTES DE CONNAISSANCES : FAITS, RÈGLES, QUESTIONS

- **Faits** : P(…). avec P un prédicat
  - pere(ali, meziane).
  - pere(larbi, saadi).
- **Clause de Horn réduite à un littéral positif**
- **Règles** : P(…) :- Q(…), …, R(…).
  - papy(X,Y) :- pere(X,Z), pere(Z,Y).
- **Clause de Horn complète**
- **Questions** : S(…), …, T(…).
  - pere(ali,X), mere(ania,X).
- **Clause de Horn sans littéral positif**

RÉFUTATION PAR RÉSOLUTION

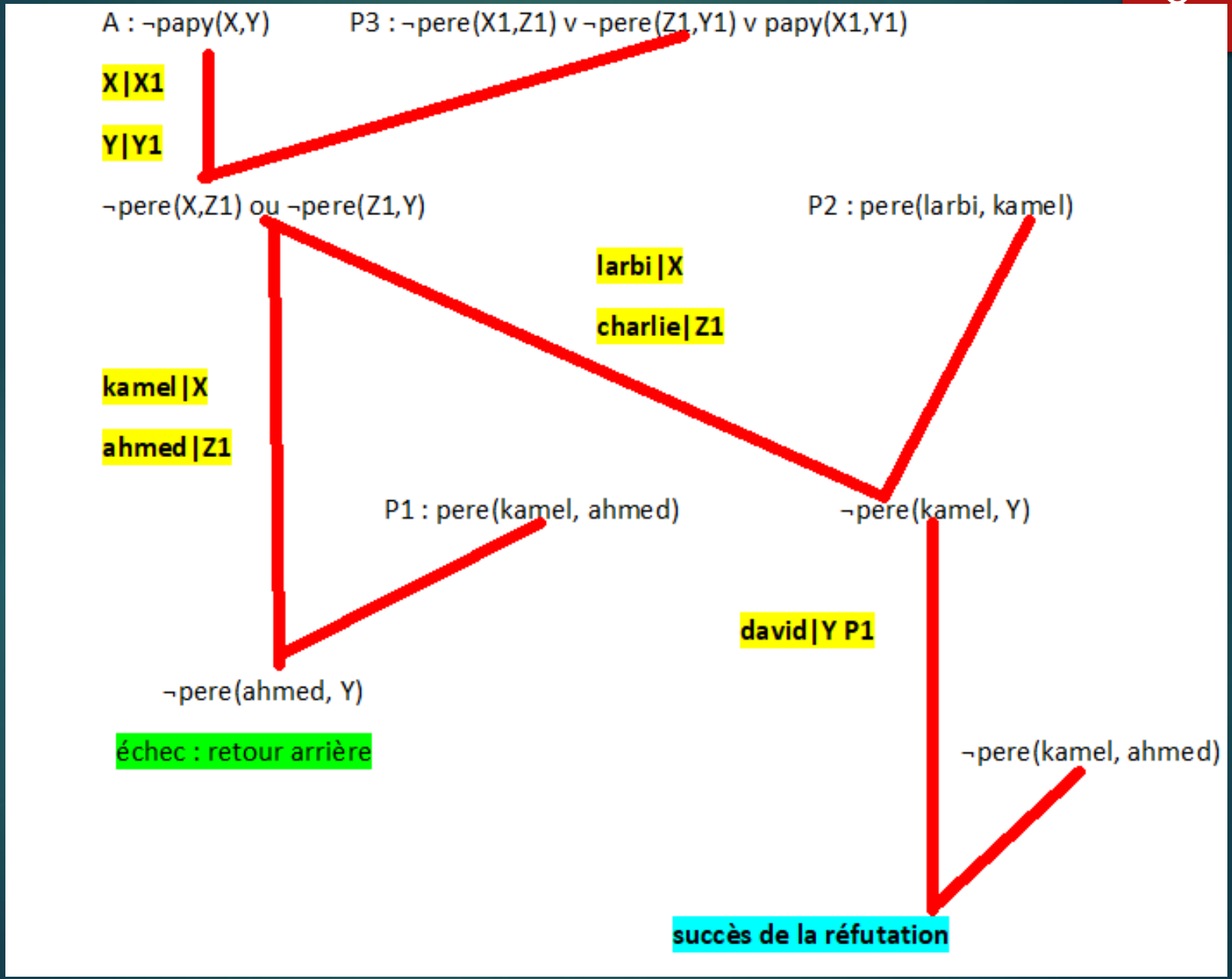
- Programme P
- **P1 : pere(kamel, ahmed).**
- **P2 : pere(larbi, kamel).**
- **P3 : papy(X,Y) :- pere(X,Z), pere(Z,Y).**

- Appel du programme P

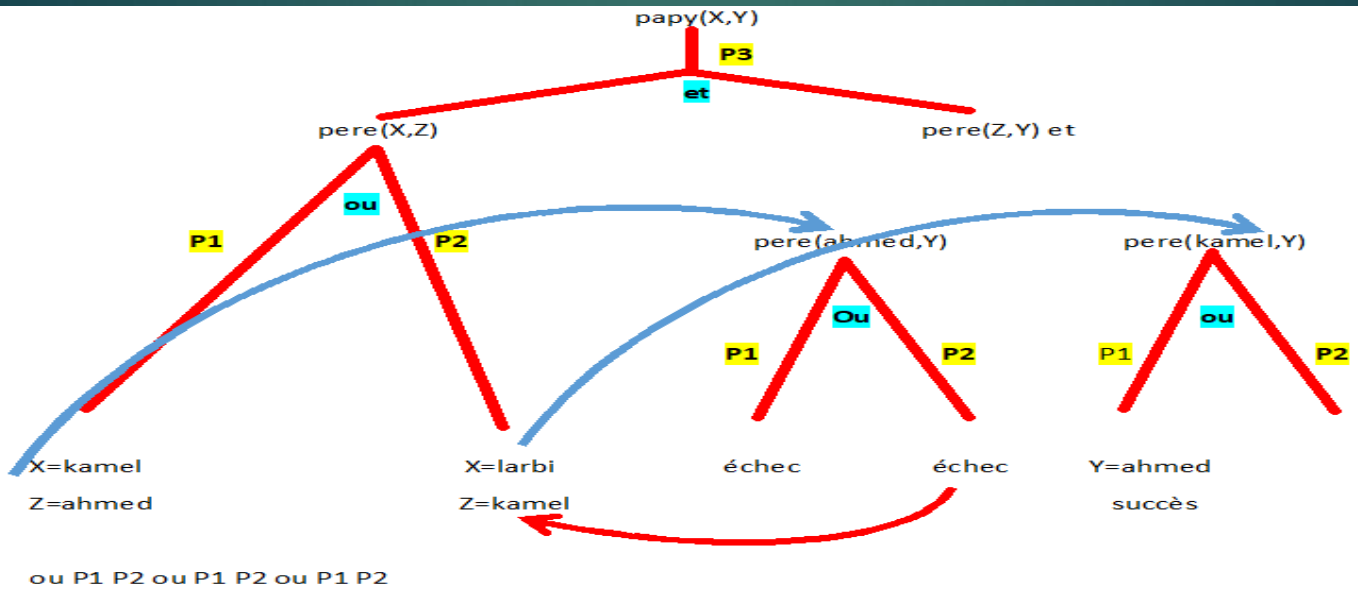
- A : papy(X,Y).
- Réponse :

- **X=larbi, Y=ahmed**

GRAPHE DE RÉSOLUTION



INTERPRÉTATION PROCÉDURALE : ARBRE ET-OU



ou P1 P2 ou P1 P2 ou P1 P2

MON PREMIER PROGRAMME (1)

- pere(kamel,ahmed).
- pere(larbi,kamel).
- papy(X,Y) :- pere(X,Z), pere(Z,Y).

- ?- [pere]. % pere compiled 0.00 sec, 824 bytes
- true.
- ?- listing.
- pere(kamel, ahmed).
- pere(larbi, kamel).
- papy(A, B) :- pere(A, C), pere(C, B).
- true.

MON PREMIER PROGRAMME (2)

- pere(kamel, ahmed).
- pere(larbi, kamel).
- papy(A, B) :- pere(A, C), pere(C, B).
- true.
- pere(kamel,ahmed).
- pere(larbi,kamel).
- papy(X,Y) :- pere(X,Z), pere(Z,Y).
- ?- papy(x,y).
- false.
- ?- papy(X,Y).
- X = larbi
- Y = ahmed
- ?- papy(larbi,X).
- X = ahmed
- true.
- ?- halt.
- lirispc1$

ORDRE DES RÉPONSES

- pere(kamel, ahmed).
- pere(larbi, kamel).
- pere(ahmed, louisa).

- mere(sophia, kamel).
- mere(ania, ahmed).

- parents(E, P, M) :- pere(P, E), mere(M, E).
- ?- parents(X,Y,Z).
- X = ahmed Y = kamel Z = ania ;
- X = kamel Y = larbi Z = sophia ;
- False.
- **Prolog parcourt le paquet de clauses de haut en bas, chaque clause étant parcourue de gauche à droite**

EXERCICES

- Construire l'**arbre ET-OU** permettant à Prolog de donner l'ensemble des réponses satisfaisant la requête parents(X,Y,Z).
- On définit le programme suivant :
  b(1). b(2). c(3). c(4). d(5). d(6).
  a(X,Y,Z) :- b(X), c(Y), d(Z).

- **Donner toutes les réponses à la requête a(X,Y,Z) dans l'ordre où Prolog les fournit.**

L'ÉNIGME POLICIÈRE

- On dispose des informations suivantes :
  - La secrétaire déclare qu'elle a vu l'ingénieur dans le couloir qui donne sur la salle de conférences
  - Le coup de feu a été tiré dans la salle de conférences, on l'a donc entendu de toutes les pièces voisines
  - L'ingénieur affirme n'avoir rien entendu
- On souhaite démontrer que si la secrétaire dit vrai, alors l'ingénieur ment

L'ÉNIGME POLICIÈRE EN PROLOG

- **Ordre 1 :** un individu entend un bruit s'il se trouve dans une pièce voisine de celle où le bruit a été produit
  - **entend(Ind,Bruit) :- lieu(Ind,Piece1), lieu(Bruit,Piece2), voisin(Piece1,Piece2).**
- **Faits relatifs à l'énigme :**
  - **voisin(couloir,salle_de_conf).**
  - **lieu(coup_de_feu,salle_de_conf).**
  - **lieu(ingenieur,couloir) :- secretaire_dit_vrai.**
  - **ingenieur_ment :- entend(ingenieur,coup_de_feu).**

L'ÉNIGME POLICIÈRE EN PROLOG

- **Hypothèse**
  - **secretaire_dit_vrai.**
- Pour la démonstration, on pose la requête :
  - **ingenieur_ment.**

SYMBOLES FONCTIONNELS

- La **fonction** « femme de Ali » est différente du prédicat
  - **femme(ourida, ali).**
    nom(femme(ali),ourida).
    age(femme(ali),25).

- **On peut parler de la femme de ali, mais pas la « calculer »**

PROGRAMMATION RÉCURSIVE

- Un **programme récursif** est un programme qui s'appelle lui-même
- Exemple : factorielle
- factorielle(1) = 1 (**Cas d'arrêt**)
- factorielle(n) = n * **factorielle(n-1)** si n≠1

- **Appel récursif**

POUR ÉCRIRE UN PROGRAMME RÉCURSIF

- Il faut :
  - **Choisir** sur quoi faire l'**appel récursif**
  - **Choisir** comment passer du résultat de l'**appel récursif** au résultat que l'on cherche
  - **Choisir** le(s) **cas d'arrêt**

BOUCLAGE

- **maries(ali, sophia).**
- **maries(badis, slimane).**
- **maries(A, B) : - maries(B, A).**
- ?- maries(ali,sophia).
- true .
- ? - maries(sophia,ali).
- true .
- ? - maries(X,Y).
- X = ali
- Y = sophia ;
- X = badis
- Y = slimane ;
- X = sophia
- Y = ali ;
- X = slimane
- Y = badis ;
- X = ali
- Y = sophia ;
- ...

BOUCLAGE

- **maries(ali, sophia).**
- **maries(badis, slimane).**
- **sont_maries(A, B) : - maries(A, B).**
- **sont_maries(A, B) : - maries(B, A).**
- - sont_maries(X,Y).
- X = ali
- Y = sophia ;
- X = badis
- Y = slimane ;
- X = sophia
- Y = ali ;
- X = slimane
- Y = badis ;
- false.
- ?-

ARITHMÉTIQUE

- **Comparaisons :** =:=, =\=, >, =, =<
- **Affectaction :** is
  - ?- X is 3+2.
  - X=5
- **Fonctions prédéfinies :** -, +, *, /, ^, mod, abs, min, max, sign, random, sqrt, sin, cos, tan, log, exp, ...

UN EXEMPLE : FACTORIELLE (1)

- **fact(0, 1).**
- **fact(N, R) :-**
  - **Nm1 is N-1,**
  - **fact(Nm1, Rnm1),**
  - **R is Rnm1*N.**
- ?- fact(5,R).
- R = 120 ;
- ERROR: Out of local stack
- Exception: (36,276) _G4661 is-36263-1 ?
- abort
- % Execution Aborted

UN EXEMPLE : FACTORIELLE (1)

- ?- **trace**, fact(3,R).
  - **Call**: (8) fact(3, _G237) ? **creep**
  - **Call**: (9) _G308 is 3-1 ? **creep**
  - **Exit**: (9) 2 is 3-1 ? **creep**
  - **Call**: (9) fact(2, _G306) ? **creep**
  - **Call**: (10) _G311 is 2-1 ? **creep**
  - **Exit**: (10) 1 is 2-1 ? **Creep**
  - **Call**: (10) fact(1, _G309) ? **creep**
  - **Exit**: (10) fact(1, 1) ? **creep**
  - **Call**: (10) _G314 is 2*1 ? **creep**
  - **Exit**: (10) 2 is 2*1 ? **creep**
  - **Exit**: (9) fact(2, 2) ? creep
  - **Call**: (9) _G237 is 3*2 ? **creep**
  - **Exit**: (9) 6 is 3*2 ? **creep**
  - **Exit**: (8) fact(3, 6) ? **creep**

UN EXEMPLE : FACTORIELLE (1)

- R = 6 ;
  - **Redo**: (10) fact(1, _G309) ? **creep**
  - **Call**: (11) _G314 is 1-1 ? **creep**
  - **Exit**: (11) 0 is 1-1 ? **creep**
  - **Call**: (11) fact(0, _G312) ? **creep**
  - **Call**: (12) _G317 is 0-1 ? **creep**
  - **Exit**: (12) -1 is 0-1 ? **creep**
  - **Call**: (12) fact(-1, _G315) ? **creep**
  - **Call**: (13) _G320 is-1-1 ? **creep**
  - **Exit**: (13) -2 is-1-1 ? **creep**

- **Il faut faire des cas exclusifs**

UN EXEMPLE : FACTORIELLE (2)

- **fact(0, 1).**
- **fact(N, R) :-**
- **fact(Nm1, Rnm1),**
- **Nm1 is N-1,**
- **R is Rnm1*N.**
- ?- fact(3,R).
- ERROR: Arguments are not sufficiently instantiated
- Exception: (9) 1 is _G241-1 ? creep
- Exception: (8) fact(_G241, _G255) ? creep
- Exception: (7) fact(3, _G195) ? creep
- % Execution Aborted
- ?- 5 is X-1.
- ERROR: Arguments are not sufficiently instantiated
- % Execution Aborted
- ?- plus(3,2,5).
- true.
- ?- plus(X,2,5).
- X = 3 true.

EXERCICE

- Définir un prédicat calculant le nième terme de la suite :
  - $u_0 = 2,\ u_n = 2u_{n-1}+3$

SOLUTION

- Base de Faits: suite.pl

```
suite_U(0, 2).
suite_U(N, R) :- _
    Nm1 is N-1,
    suite_U(Nm1, Rnm1),
    R is 2 * Rnm1 + 3.
```

UNE FACTORIELLE AVEC ACCUMULATEUR

- **fact(N,R) :- fact(N,1,R).**
- **fact(1,R,R).**
- **fact(N,I,R) :-**
- **N>1,**
- **Nm1 is N-1,**
- **NewI is N*I,**
- **fact(Nm1,NewI,R).**
- ?- **trace**, fact(3,N).
- Call: (7) fact(3, _G234) ? creep
- Call: (8) fact(3, 1, _G234) ? creep
- Call: (9) 3>1 ? creep
- Exit: (9) 3>1 ? creep
- Call: (9) _G305 is 1*3 ? creep
- Exit: (9) 3 is 1*3 ? creep

UNE FACTORIELLE AVEC ACCUMULATEUR

- Call: (9) _G308 is 3-1 ? Creep
- Exit: (9) 2 is 3-1 ? creep
- Call: (9) fact(2, 3, _G234) ? creep
- Call: (10) 2>1 ? creep
- Exit: (10) 2>1 ? creep
- Call: (10) _G311 is 3*2 ? creep
- Exit: (10) 6 is 3*2 ? creep
- Call: (10) _G314 is 2-1 ? creep
- Exit: (10) 1 is 2-1 ? creep
- Call: (10) fact(1, 6, _G234) ? creep
- Call: (11) 1>1 ? creep
- Fail: (11) 1>1 ? creep
- Redo: (10) fact(1, 6, _G234) ? creep
- Exit: (10) fact(1, 6, 6)
- ? creep N = 6

DIFFÉRENTS PRÉDICATS DE COMPARAISON

=:= =\=

Expr1 =:= Expr2

réussit si le résultat de l'évaluation de l'expression arithmétique Expr1 est égal au résultat de l'évaluation de l'expression arithmétique Expr2

- ?- A is 3, A=:=3.
- A = 3.
- ?- A is 3, A=:=2+1.
- A = 3.
- ?- a=\=b.
- ERROR

Var is Expr

unifie le résultat de l'évaluation de l'expression arithmétique Expr avec la variable Var

DIFFÉRENTS PRÉDICATS DE COMPARAISON

== \==

- ?- A is 3, A==3.
- A = 3.
- ?- A is 3, A==2+1.
- false.
- ?- a\==b.
- true.
- ?- A==3.
- false.
- ?- p(A)\==p(1).
- true.

- **Comparer deux termes :**
  - **T1==T2 réussit** si T1 est identique à T2
  - **T1 \==T2 réussit** si T1 n'est pas identique à T2

DIFFÉRENTS PRÉDICATS DE COMPARAISON

= \=

- ?- A=3.
- A = 3.
- ?- p(A)\=p(1).
- false.

- **Comparer deux termes :**
  - **T1=T2 unifie** T1 avec T2
  - **T1 \=T2 réussit** si T1 n'est pas unifiable à T2

COMPARAISON ET UNIFICATION DE TERMES

- **Vérifications de type :** var, nonvar, integer, float, number, atom, string, …

- **Comparer deux termes :**
  - **T1==T2 réussit** si T1 est identique à T2
  - **T1 \==T2 réussit** si T1 n'est pas identique à T2
  - **T1=T2 unifie** T1 avec T2
  - **T1 \=T2 réussit** si T1 n'est pas unifiable à T2

COMPARAISON ET UNIFICATION DE TERMES

Expr1 =:= Expr2

réussit si le résultat de l'évaluation de l'expression arithmétique Expr1 est égal au résultat de l'évaluation de l'expression arithmétique Expr2

Var is Expr

unifie le résultat de l'évaluation de l'expression arithmétique Expr avec la variable Var

LISTES

- Liste vide : [ ]
- Cas général :
- [Tete | Queue]
- [a,b,c] = [a | [b | [c | [ ]]]]