

# Projet 4IF ALIA - Septembre 2013

---

- Intervenants
  - Jean-François Boulicaut
  - Mehdi Kaytoue
- Objectifs pédagogiques "Découvrir Prolog : un vecteur de la programmation en logique"



Présentation du 27/9

Projet 4IF à partir du 30/9 (travail en hexanome)

- Semaine 40: "Exercices d'acclimatation"
- Semaine 41: Rendu sur exercices, choix d'un projet  
Programmation et développement
- Semaine 42: Développement et restitution  
sur la semaine 43

# Pourquoi Prolog ?

---

- Il n'y a pas que la programmation impérative/par objets dans la vie ;-)
- Retour sur FGCS (Années 80)
- Le prototypage et les applications de l'Intelligence Artificielle demandent des mécanismes de programmation puissants
  - Données ~ programmes
  - Structures de données (récursives) abstraites
  - Non déterminisme
- A propos du dogme de la programmation impérative efficace

# Différentes présentations de Prolog

---

- Inventé par des anglais en 1974 et implémenté par des français en 1973 ;-)
- Un outil pour le traitement de langages
- Un démonstrateur de théorèmes en logique des prédicats
- Un moteur d'inférences
- Un système de programmation par contraintes
- Un langage de programmation (avec de nombreux BIPs)
  - Programmation de très haut niveau
- Un système de gestion de bases de données déductives

# La vision "Bases de Données"

---

- Faits - Requêtes (buts, littéraux, conjonctions)

lien(Paris,Lyon).

lien(Lyon,Marseille).

lien(Nice,Marseille).

lien(Lyon,Paris).

?-lien(X,Marseille).                      2 succès

?-lien(Marseille,Lyon).                    échec

?-lien(X,Y), lien(Y,X).                   2 succès

- Comprendre l'ordre des réponses (résolution Prolog)
- Faits définis en extension ou en intention

# La vision "Bases de Données" déductive

---

- Faits définis en extension ou en intention (clauses)

lien(paris,lyon).

*lien(paris,lyon,430,37)*

lien(lyon,marseille).

*... lien(X,Y,\_,\_) ...*

lien(nice,marseille).

chemin(X,Y) :- lien(X,Y). (a)

?-chemin(X,marseille).

?-chemin(X,Y).

chemin(X,Y) :- lien(X,Y). (a)

chemin(X,Y) :- lien(X,Z), chemin(Z,Y). (b)

?-chemin(X,marseille).

?-chemin(X,Y).

# La vision "Bases de Données" déductive

---

- Sélections, projections, jointures ... mais aussi, calcul de fermetures transitives ... impossible en SQL !

fermeture(X,Y) :- relation(X,Y). (a)

fermeture(X,Y) :- relation(X,Z), fermeture(Z,Y). (b)

?-fermeture(X,Y).

- La stratégie de résolution Prolog (e.g., backtracking) permet de comprendre le comportement à l'exécution
  - Sémantique déclarative vs. sémantique opérationnelle
- Prolog est un véritable langage de programmation (langage de requête et langage hôte)

# La vision "Bases de Données" déductive

---

- Sémantique déclarative vs. sémantique opérationnelle

fermeture1(X,Y) :- relation(X,Y). (a)

fermeture1(X,Y) :- relation(X,Z), fermeture1(Z,Y). (b)

fermeture2(X,Y) :- relation(X,Z), fermeture2(Z,Y). (b)

fermeture2(X,Y) :- relation(X,Y). (a)

fermeture3(X,Y) :- relation(X,Y). (a')

fermeture3(X,Y) :- fermeture3(Z,Y), relation(X,Z) (b')

fermeture4(X,Y) :- fermeture4(Z,Y), relation(X,Z) (b')

fermeture4(X,Y) :- relation(X,Y). (a')

# Introduction des termes

---

- Au delà des constantes et variables : les termes

lien1(paris,marseille,date(12,5,2008)). ...

lien2(paris,marseille,heure(13,15),[lundi,jeudi]). ...

avant(date(\_,\_,A1),date(\_,\_,A2)) :- A1 < A2.

avant(date(\_,M1,A),date(\_,M2,A)) :- M1 < M2.

avant(date(J1,M,A),date(J2,M,A)) :- J1 < J2.

?- lien1(D,A,date(\_,5,\_)).

?- lien1(paris,lyon,date(J1,M1,A)), lien1(lyon,X,date(J2,M2,A)),  
    avant(date(J1,M1,A),date(J2,M2,A)).

?- lien2(paris,\_,heure(H,\_), L), H > 12, member(jeudi,L).

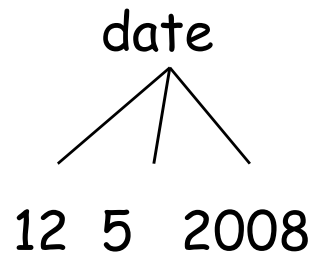


# Termes (1)

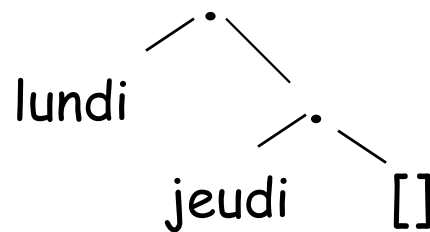
---

- Terme ?
  - Constante
  - Variable
  - Foncteur(Terme, ..., Terme)

date(12,5,2008)



[lundi,jeudi]      .(lundi,.(jeudi,[]))



[[[a,b],[c,d]],[[a],f]]

# Termes (2)

---

- Codage de graphes ?

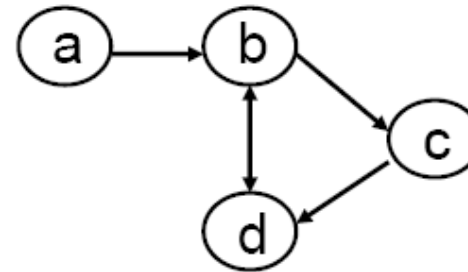
[[a, [b]], [b, [c,d]], [c,[d]], [d,[b]]]

lien(a,b).

lien(b,c).

lien(c,d).

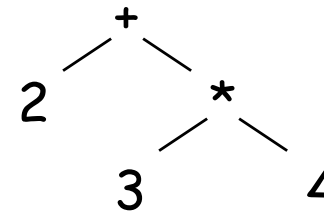
lien(b,d).



- Foncteurs et opérateurs

$2+3*4$

$+(2,*(3,4))$



# Unification

---

- Rendre deux termes identiques sous un ensemble de substitutions :  $T1=T2$  vs.  $T1==T2$
- $[[a,b],c,X]$  unifiable avec  $[Y,c,Y]$  avec  $X$  et  $Y$  instanciées à  $[a,b]$
- L'opérateur  $|$  pour le traitement de listes
- $[X|Y]$ 
  - $[X|Y]=[d,b,c]$  est un succès avec  $X$  instanciée à  $d$  et  $Y$  instanciée à  $[b,c]$
- L'unification n'est pas un "simple passage de paramètres" (typage/modage disponible)

# Prédicats utilitaires sur les listes

---

member(X,[X|\_]).                   /\* également appelé element \*/

member(X,[\_|L]) :- member(X,L).

?- member(b,[a,b,c]).

?- member(X,[a,b,c]).

?- member(a,L).

append([ ], L1, L1).

append([A|L1],L2,[A|L3]) :- append(L1,L2,L3).

?- append([a,b],[c],[a,b,c]), append([a,b],[c,d],X).

?- append(X,Y, [a,b,c,d]).

?- append(X,[a,b],Y).

?- L=[a,b,c], append(\_,[X],L).

# Retour sur le calcul de chemins

---

- Un parcours en profondeur d'abord

lien(a,b).

lien(b,c).

lien(c,d).

lien(b,d).

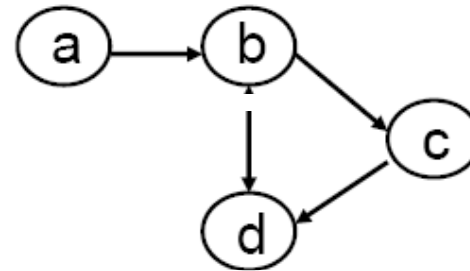
chemin2(X,Y,T) :- chemin(X,Y,[X],T).

chemin(X,X,V,V).

chemin(X,Y,V,T) :- lien(X,Z), chemin(Z,Y,[Z|V],T).

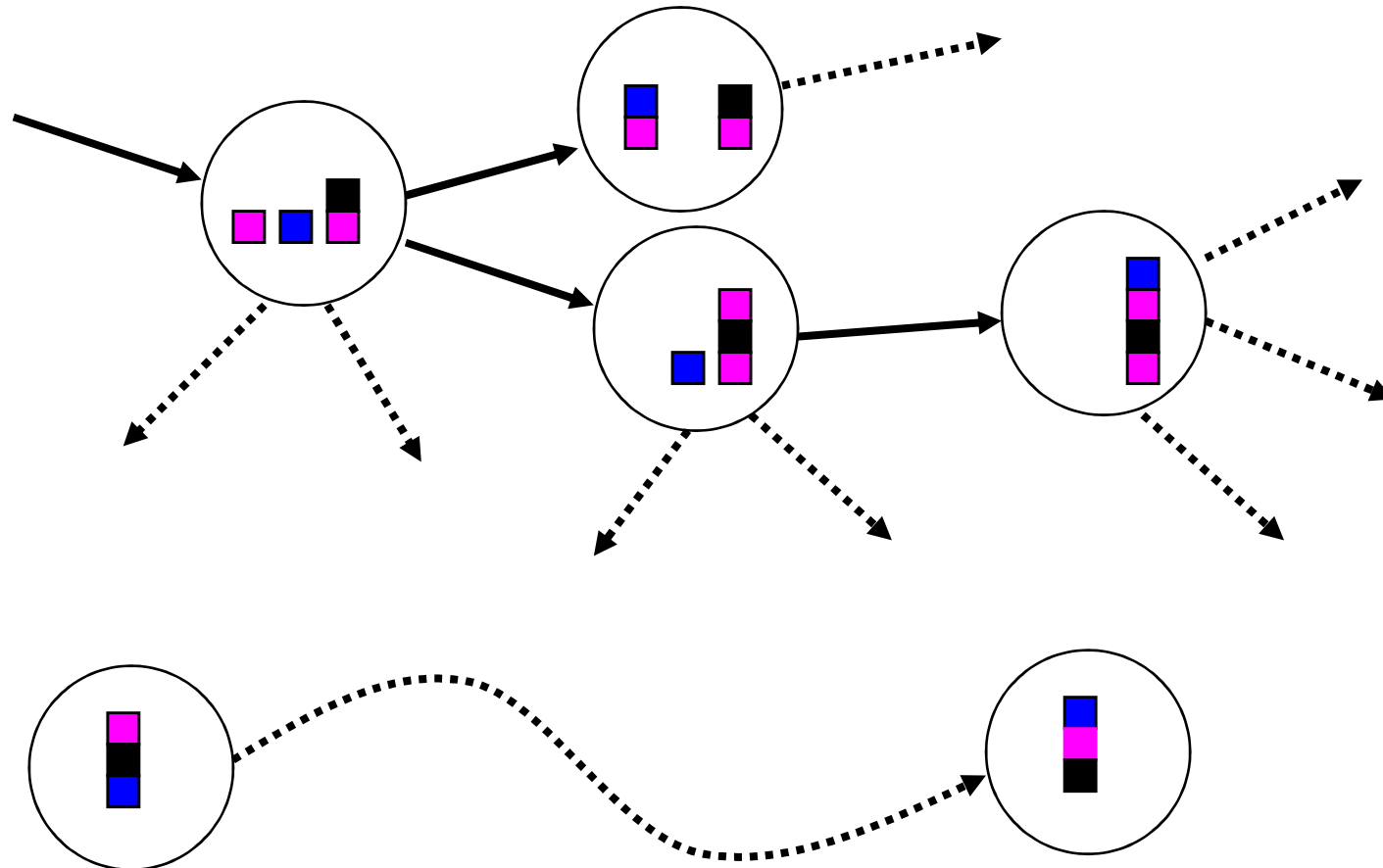
?- chemin2(a,c,L).

?- chemin2(X,d,L).



# Parcours de graphes d'états

---



# Retour sur le calcul de chemins

---

- Une amélioration du parcours en profondeur d'abord

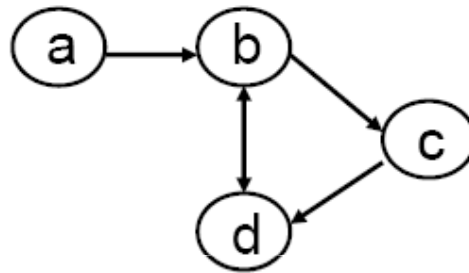
lien(a,b).

lien(b,c).

lien(c,d).

lien(b,d).

lien(d,b).



?- chemin3(a,d,L).

L=[a,b,c,d], L=[a,b,d]

chemin3(X,Y,R) :- chemin(X,Y,[X],T), **reverse**(T,R).

chemin(X,X,V,V).

chemin(X,Y,V,T) :- lien(X,Z), **not(member(Z,V))**,  
chemin(Z,Y,[Z|V],T).

# Mécanismes de contrôle et méta-prédicats

---

- Utilisation de la coupure

$$P \text{ :- } P_1, P_2, \dots, !, P_{n-2}, P_{n-1}, P_n$$

intersec([],\_,[]).

intersec([X|Y], Z, [X|T]) :- member(X,Z), intersec(Y,Z,T).

intersec([X|Y], Z, T) :- not(member(X,Z)), intersec(Y,Z,T).

intersec([],\_,[]) :- !.

intersec([X|Y], Z, [X|T]) :- member(X,Z), !, intersec(Y,Z,T).

intersec([X|Y], Z, T) :- intersec(Y,Z,T).



# Mécanismes de contrôle et méta-prédicats

---

- Négation par l'échec

not P :- P, !, fail.

not P.

- Autres exemples de métaprédicats

forall, once, ...

setof, bagof, findall, ...

?-R=[a,b,c,d], S=[a,c,e,d,r],

setof(X,(member(X,R),member(X,S)),M), M = [a,c,d]

?-setof(ar(X,Y),(lien(X,Y),lien(Y,X)), M), M == [ar(b,d), ...]

# Mécanismes de contrôle et méta-prédicats

---

- Attention: réaliser des négations dites sûres

```
single(X) :- not(married(X)), (homme(X) ; femme(X)).
```

```
homme(bob).
```

```
femme(lola).
```

```
married(tom).
```

```
?-single(X).
```

```
No
```

# Retour sur le calcul de chemins

- Un parcours en largeur d'abord

```
collecter(T,B,R) :- bagof(T,B,R), ! .
```

```
collecter(_,_,[]).
```

```
chemin4(X,Y,R) :- chemin_L([[X]],Y,T), reverse(T,R).
```

```
chemin_L([[X|Xs]|_],X,[X|Xs]).
```

```
chemin_L([[X|Xs]|L],Y,P):-
```

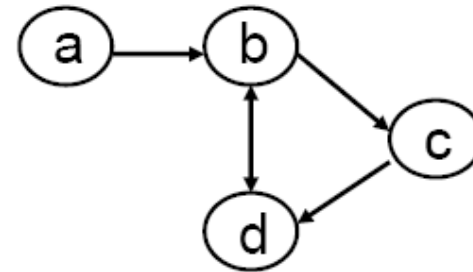
```
    collecter([N,X|Xs],
```

```
                (lien(X,N),not(member(N,[X|Xs]))),Succ_de_X),
```

```
    append(L,Succ_de_X,Z),chemin_L(Z,Y,P).
```

```
?- chemin4(a,d,L).
```

```
L=[a,b,d], L=[a,b,c,d]
```



# "Manipulations de clauses"

---

- Retour sur les possibilités de manipulation symboliques
  - Espace des termes vs. espace des prédicats

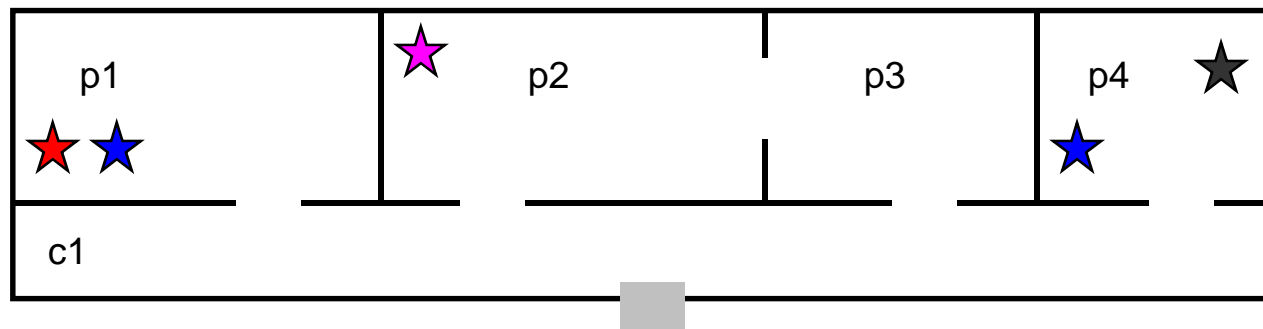
?- T=.. [lien1,[paris],[marseille],[date,12,5,A]],  
T= lien1 (paris,marseille,date(12,5,2008)).

- Assert
- Call
- Retract

*Attention : utiliser dynamic(P) ... mais raisonnements non monotones, difficultés à comprendre les programmes ... pour autant mécanisme clé pour l'IA*

# "Manipulations de clauses"

- Espace des termes vs. espace des prédicats



p1([obj(**s1**),obj(**s2**),acces(c1)]).



p2([obj(**s3**),acces(c1),acces(p3)]).

p3([acces(p2),acces(c1)]).

p4([obj(**s4**),obj(s5),acces(c1)]).

c1([robot,acces(p1),acces(p2), acces(p3), acces(p4)]).

# Mise à jour dynamique de clauses

---

p1([obj(s1),obj(s2),acces(c1)]).

p2([obj(s3),acces(c1),acces(p3)]).

p3([acces(p2),acces(c1)]).

p4([obj(s4),obj(s5),acces(c1)]).

c1([robot,acces(p1),acces(p2), acces(p3), acces(p4)]).

p1([robot,obj(s1),obj(s2),acces(c1)]).

p2([obj(s3),acces(c1),acces(p3)]).

p3([acces(p2),acces(c1)]).

p4([obj(s4),obj(s5),acces(c1)]).

c1([acces(p1),acces(p2), acces(p3), acces(p4)]).

# Mise à jour dynamique de clauses

---

```
lien(X,Y) :- F =.. [X,ArgX], call(F), member(acces(Y),ArgX).
```

```
?-chemin3(p1,p3,L).
```

```
mouvement_Robot(X,Y) :-
```

```
    F =.. [X,ArgX], call(F), member(robot,ArgX), retract(F),  
    select(robot,ArgX,NArgX),
```

```
        NF =.. [X,NArgX], assert(NF),
```

```
    G =.. [Y,ArgY], call(G), retract(G),
```

```
        NG =.. [Y,[robot|ArgY]], assert(NG).
```

```
?-mouvement_Robot(c1,p1).
```

# Le calcul en Prolog

---

- Le statut des variables utilisées en Prolog n'est pas compatible avec la vision impérative habituelle
  - Sémantique de  $X+Y*Z$  ?
  - Sémantique de  $X=X+1$  ?
- Pour faciliter les calculs : le prédicat prédéfini IS

?-  $X$  is  $2*3+2$ ,  $X == 8$

?-  $X$  is 8,  $X$  is  $X+1$

len([],0).

len([T|Q],N) :- len(Q,N1), N is N1+1.

?- len([a,b,c],3).



# Prolog et contraintes (1)

---

colorier(C1,C2,C3,C4,C5)

:- col(C1), col(C2), col(C3), col(C4), col(C5),

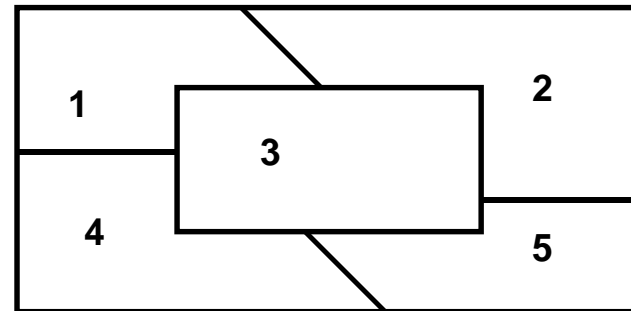
C1 \== C2, C1 \== C3, C1 \== C4, C2 \== C3,

C2 \== C5, C3 \== C4, C3 \== C5, C4 \== C5.

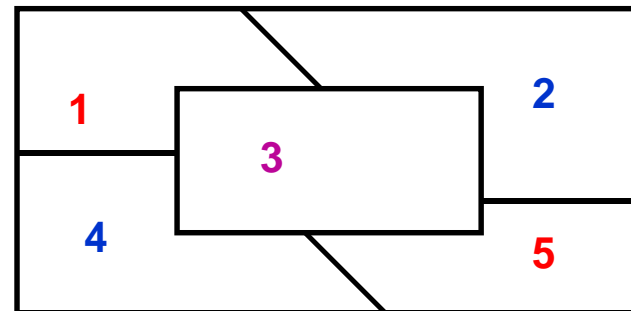
col(1).

col(2).

col(3).



Première solution



# Prolog et contraintes (2)

---

colorier(C1,C2,C3,C4,C5)

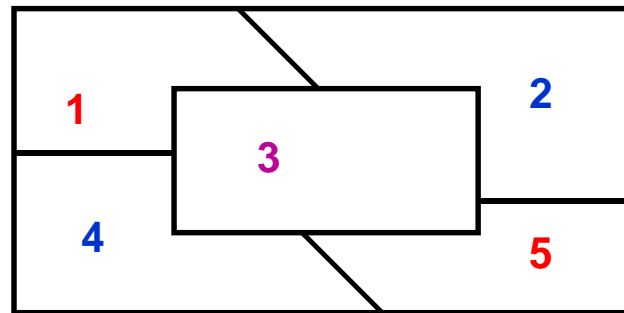
:- voisin(C1,C2), voisin(C1,C3), voisin(C1,C4), voisin(C2,C3),  
voisin(C2,C5), voisin(C3,C4), voisin(C3,C5), voisin(C4,C5).

voisin(X,Y) :- col(X), col(Y), X \== Y.

col(1).

col(2).

col(3).



# Prolog et contraintes (3)

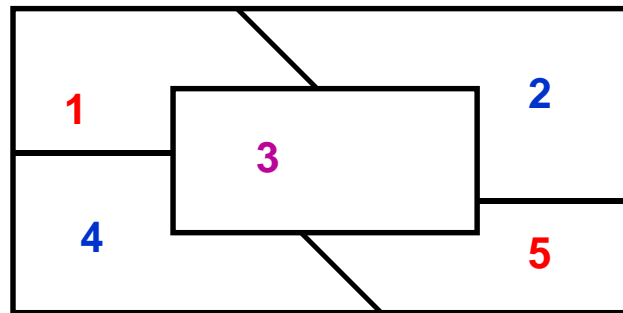
---

```
:-use_module(library('clp/bounds')).
```

```
color(L,N) :- L=[C1,C2,C3,C4,C5], L in 1..N,
```

```
    C1#\=C2, C1#\=C3, C1#\=C4, C2#\=C3,
```

```
    C2#\=C5, C3#\=C4, C3#\=C5, C4#\=C5, label(L).
```



# D'un point de vue pratique

---

- Utilisation de SWI Prolog installé sur les postes, documents dans les répertoires "usuels"
- Exercices d'acclimatation
  - Tous les exemples utilisés dans cette présentation peuvent/doivent être essayés ("intro\_prolog.pl")
  - Thème "généalogie"
    - Par exemple, codage du prédicat `frere(X,Y)`, construction explicite des arbres généalogiques
  - Thèmes "listes" et "ensembles"
    - Voir le fichier "Seance\_1\_ALIA\_4IF.pdf"

# Projet 4IF ALIA

---

## Projet 4IF à partir du 24/9 (travail en hexanome)

- Semaine 40
  - “Exercices d'acclimatation” ... par binômes
- Semaine 41
  - Rendu sur exercices, choix d'un projet et développement
- Semaine 42
  - Développement
  - Restitution en semaine 43.