# ALIA 4IF – Octobre 2021

- Jean-François Boulicaut & Sylvie Calabretto


- Objectifs pédagogiques

"Découvrir Prolog : un vecteur de la programmation en logique ;
Un outil pour le développement de programmes intelligents"

Distanciel (zoom)

Cours (30/9, 4/10)

Travaux pratiques (travail consolidé par hexanôme)

- "Exercices d'acclimatation" ; Programmation en Prolog d'un jeu à 2 joueurs (Type OTHELLO) ; Programmation et démonstration

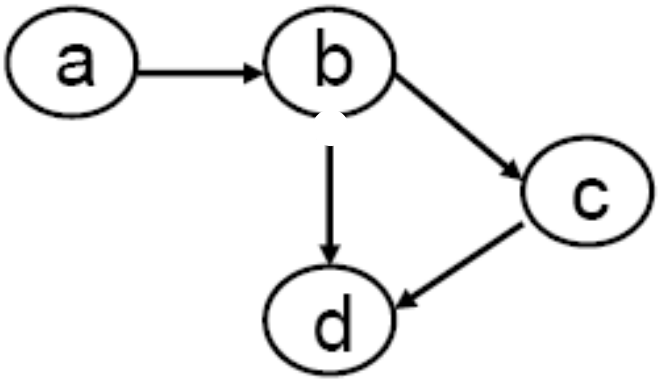
# Pourquoi Prolog ?

- Il n'y a pas que la programmation impérative/par objets dans la vie ;-)
- Retour sur FGCS (Années 1980)
- Le prototypage et les applications de l'Intelligence Artificielle demandent des mécanismes de programmation puissants
  - Données ~ Programmes
  - Programmation symbolique et structures de données abstraites
  - Non déterminisme
- A propos de l'utilisation de Prolog dans l'industrie ?

# Un programme testant l'existence de chemins

- Existence de chemins dans un graphe sans cycle

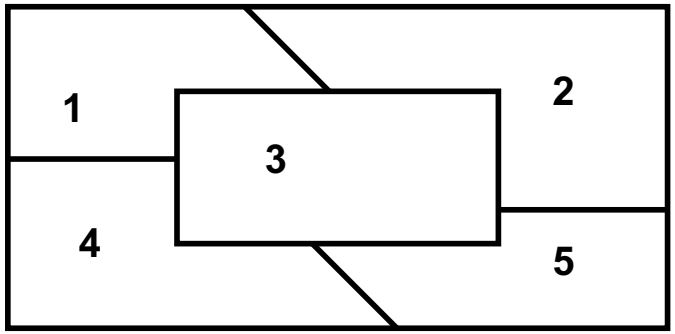
```
lien(a,b).
lien(b,c).
lien(c,d).
lien(b,d).
chemin(X,X).
chemin(X,Y) :- lien(X,Z), chemin(Z,Y).
```



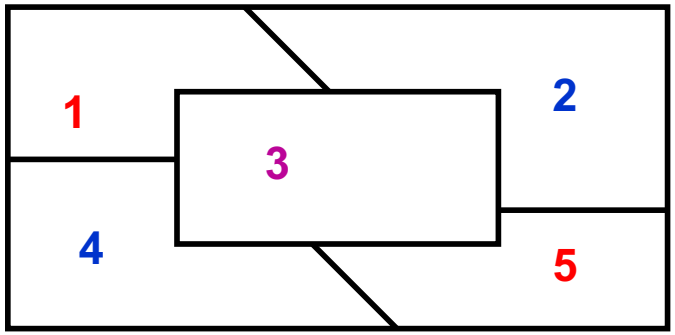

```
?- chemin(a,d).
?- chemin(D,c).
```

# Le “Générer et Tester”

```
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



# … des exemples “simples”

- Généalogie …

```
pere(a,b).
pere(b,c).
mere(e,d).
…
grand_pere(X,Y) :- pere(X,F), pere(F,Y).
…                    Bonnes modélisations du réel qui nous intéresse ?
frere_ou_soeur (X,Y) :-
    pere(P,X), pere (P,Y),
        mere(M,X), mere(M,Y).
```

# 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 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, litéraux, disjonctions, 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(lyon,marseille).
lien(nice,marseille).

*lien(paris,lyon,430,37)*
*... lien(X,Y,_,_) ...*

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

?-chemin(D,marseille).
?-chemin(D,A).

# La vision “Bases de Données” déductive

- Faits définis en extension ou en intention (clauses)

parent(jacques, jean).  masculin(jacques).

parent(jean, claire).  masculin(jean).

…  feminin(claire).

…

```
frere(X,Y) :- parent(Z,X), parent(Z,Y), masculin(X), X\==Y.
grand_parent(X,Y) :- parent(X,Z), parent(Z,Y).
arriere_grand_parent(X,Y) :- parent (X,Z), grand_parent(Z,Y).
?-frere(louis,jean).
?-frere(X,Y).
? (parent(X,bob) ; grand_parent(X,bob) ;
                arriere_grand_parent(X,bob)).
```

# 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

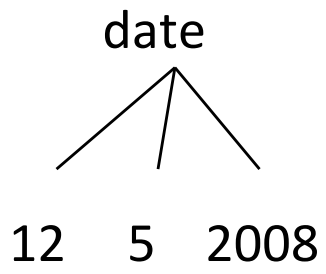
- Au delà des constantes et variables : les termes

```
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).
```

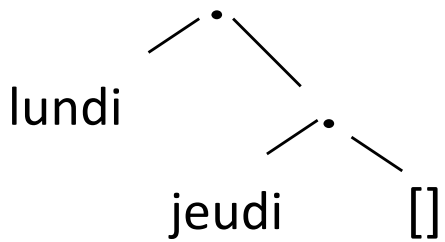
# Termes (1)

- 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 ?

  graphe([[a, [b]], [b, [c,d]], [c,[d]], [d,[b]]]).

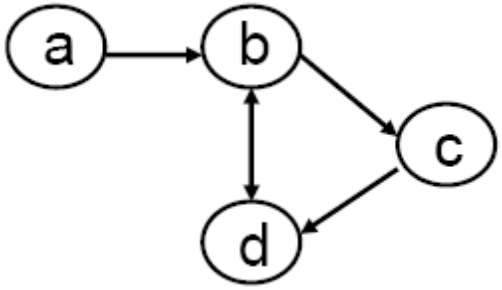
  lien(a,b).

  lien(b,c).

  lien(c,d).

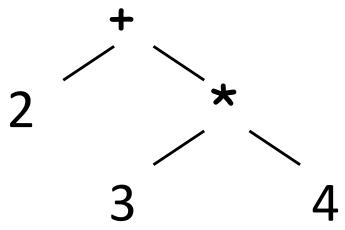
  lien(b,d).

  lien(d,b).




- Foncteurs et opérateurs

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



# Unification

- Rendre deux termes identifiques sous un ensemble de substitutions : T1=T2 vs. T1==T2
  - ?- X=a, X==a, Y=X, Y==a, X=b.
  - ?- [[a,b],c,X] = [Y,c,Y].
- 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]
  - ?- X=[T|Q], T = a, Q=[b,c,d], X==[a,b,c,d].
  - ?- X=[T|Q], T = a, Q=[b,Y,d], X==[a,b,c,d]. /* X=[a,b,c,d] */
- L'unification n'est pas un "simple passage de paramètres" (typage/modage disponible)

# Prédicats simples sur les listes

```
member(X,[X|_]).                    /* également appelé element */
member(X,[_|L]) :- member(X,L).
```

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

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

?- member(a,Liste).

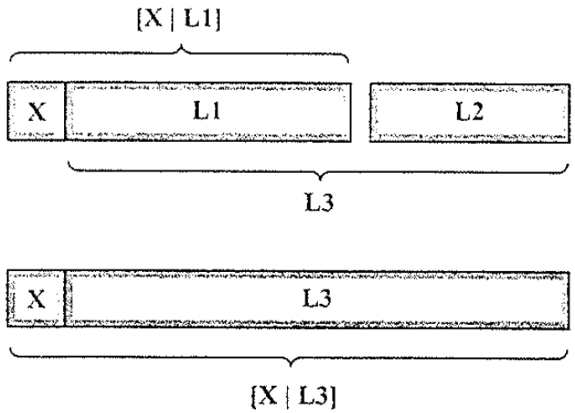
```
append([ ], L1, L1).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

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

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

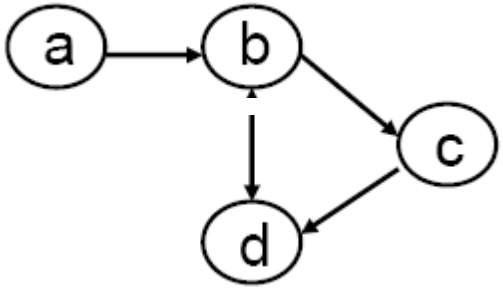
?- append(L,[a,b],R]).

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



# Retour sur le calcul de chemins

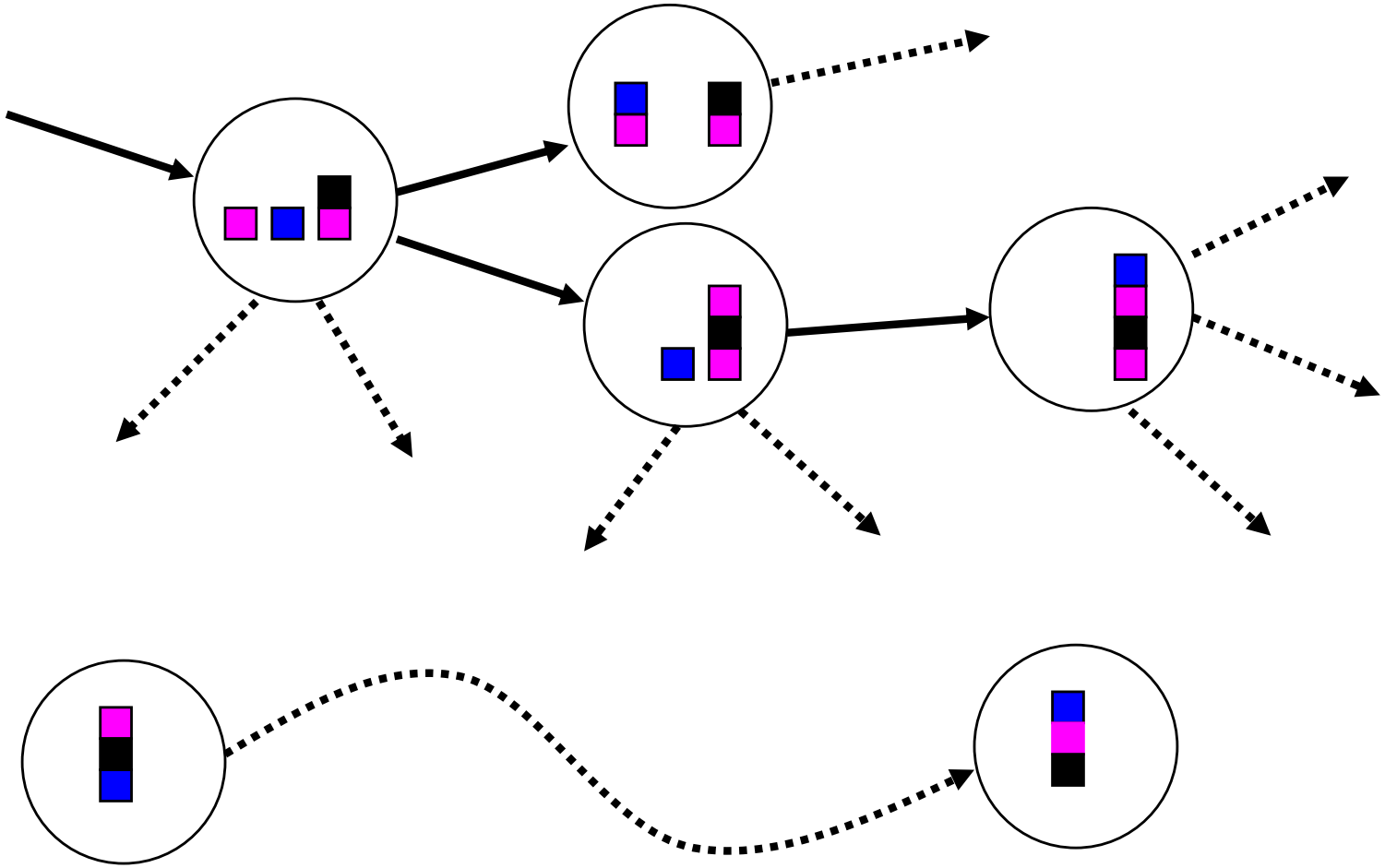
- Un parcours en profondeur d'abord




```
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(D,d,L).
```

# Parcours de graphes d’états



# Retour sur le calcul de chemins

- Une amélioration du parcours en profondeur d'abord

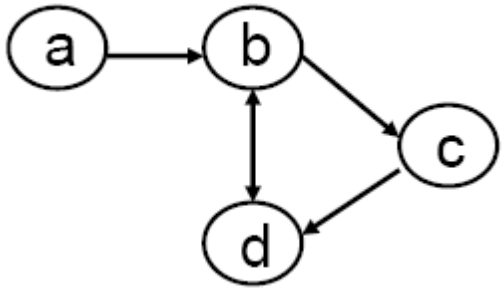
lien(a,b).

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), **renverser(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).

# Est-ce facile de renverser une liste ?

```
renverser([],[]).
renverser([T|Q],R) :- renverser(Q,R1), append(R1,[T],R).
```

```
?- renverser([a,b,c,d],[d,c,b,a]).
?- renverser([a,b,c,b],L).
?- renverser(L,[a,b,c,b]).
```

Et le “coup de l’accumulateur”

```
renverser(L,R) :- rev(L,[],R).
rev([],A,A).
rev([T|Q],A,R) :- rev(Q,[T|A],R).
```

# Mécanismes de contrôle et méta-prédicats

- Utilisation de la coupure

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

couleur(rouge).

couleur(bleu).

taille(grand).

taille(petit).

| | |
|---|---|
| c1(X,Y) :- couleur(X), taille(Y). | ?-c1(C,T). |
| c2(X,Y) :- ! , couleur(X), taille(Y). | ?- c2(C,T). |
| c3(X,Y) :- couleur(X), !, taille(Y). | ?-c3(C,T). |
| c4(X,Y) :- couleur(X), taille(Y), !. | ?-c4(C,T). |

# Mécanismes de contrôle et méta-prédicats

- Utilisation de la coupure

$$P :- P_1, P_2, ..., !, P_{n-2}, P_{n-1}, P_n$$

```
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

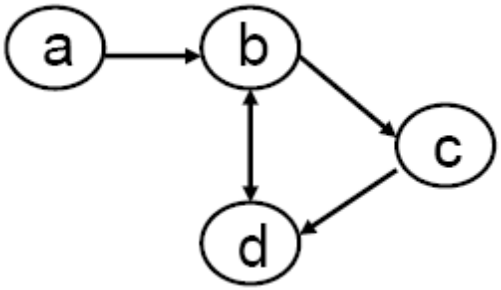
- Attention: réaliser des négations dites sûres

```
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

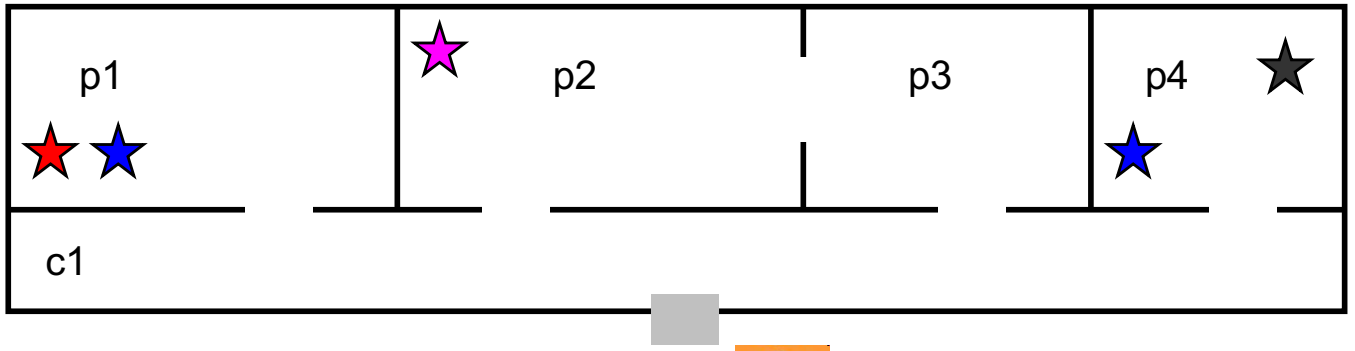
```
?- T=.. [lien1,[paris],[marseille],[date,12,5,A]],
        T= lien1(paris,marseille,date(12,5,2008)).
```

  - 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).
```

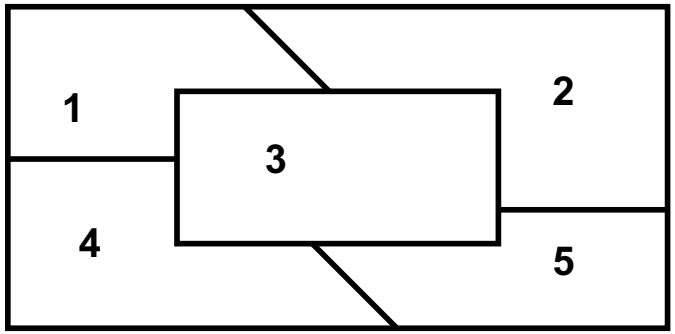
# Le calcul en Prolog

- 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

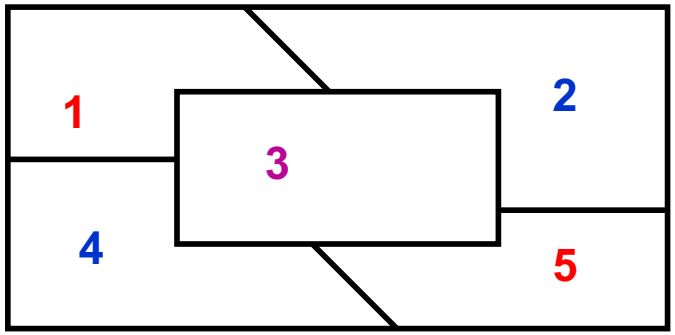
```
?- X is 2*3+2, X ==8

?- 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).
```

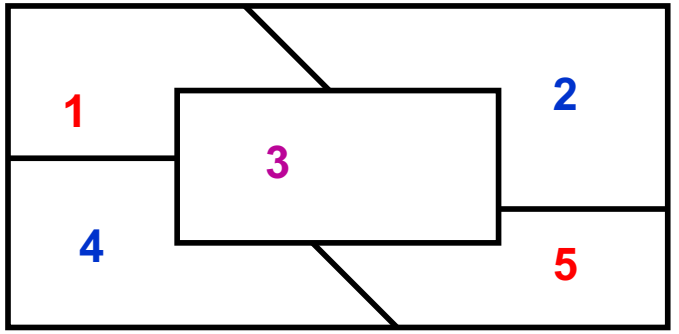



Première solution



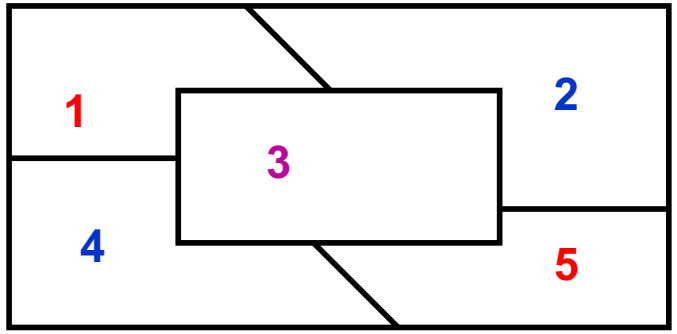
# Prolog et contraintes (2)

```
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 - SWISH
- Exercices d’acclimatation
  - Tous les exemples utilisés dans cette présentation peuvent/doivent être essayés (“intro_prolog.pl”)
  - Thème “généalogie” avec construction explicite des arbres généalogiques
  - Thèmes “listes” et “ensembles”