# Conception et implémentation

IFT-17586 Intelligence artificielle I - E2003

## Séance 11

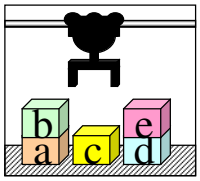
- Conception de la tâche de planification en Prolog

# Planification

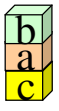
- Description des états
  - L'état initial (p. 285)

    [ surtable(a), surtable(c), surtable(d),
    sur(b,a), sur(e,d),
    libre(b), libre(c), libre(e),
    tenir() ]




  - Le but à atteindre (p. 291)

    [ sur(b,a), sur(a,c) ]



# Planification

- Description des opérateurs (les actions du bras du robot) :
  - Opérateur → P = liste des préconditions
    → A = liste à ajouter
    → S = liste à supprimer
  - Formulation possible en PROLOG :

```
possible( Opérateur, P ).
ajoute( Opérateur, A ).
supprime( Opérateur, S ).
```



# Planification

- Description des opérateurs
  - Exemple (p. 292)

| | |
|---|---|
| ramasser(X) | P : [tenir(), libre(X), surtable(X)]<br>A : [tenir(X)]<br>S : [surtable(X), tenir()] |

```
possible( ramasser(X), [pince(vide), libre(X), surtable(X)] ).
ajoute( ramasser(X), [tenir(X)] ).
supprime( ramasser(X), [surtable(X), tenir()] ).
```

# Planification

- Dérivation des plans :
  - Choisir un but, satisfaire ses conditions
  - Continuer jusqu'à ce que tous les buts soient satisfaits
- Requête à poser :
  - plan( État, Buts, Plan_à_appliquer, État_Final)

```
|?-plan(   [surtable(a), surtable(c), surtable(d), sur(b,a),
          sur(e,d), pince(vide), libre(b), libre(c), libre(e)],
          [sur(b,a), sur(a,c)], Plan, EtatFinal).
```

# Planification

- Procédure :
  1. Sélectionner un but non satisfait ($\notin$ état courant)
  2. Trouver l'opérateur qui permet d'atteindre ce but avec le prédicat **ajoute( Opérateur, À_ajouter )**
  3. Trouver les préconditions de cet opérateur avec le prédicat **possible( Opérateur, Conditions )**
  4. Satisfaire ces préconditions en utilisant le planificateur **plan( ÉtatCourant, Condition, SousPlan1, ÉtatIntermédiaire1 )**
  5. Appliquer l'opérateur à ÉtatIntermédiaire1 en ajoutant les nouveaux éléments à l'état et en supprimant les anciens (mise à jour)
  6. Recommencer en ❶ tant que tous les buts ne sont pas satisfaits dans l'état courant

## Prédicats prédéfinis

- append/3 : append(Liste1, Liste2, Liste12)
  - Ce prédicat peut être utilisé comme générateur pour la liste Liste1. Ex :

```
| ?- append(A,B,C).
A = [] , B = C = _ ;
A = [_44024] , B = _ , C = [_44024|B] ;
A = [_44024,_46538] , B = _ , C = [_44024,_46538|B] ;
etc.
```

## Prédicats à définir

- satisfait/2 : satisfait(Etat, Buts)
  - Tous les membres de la liste Buts sont dans la liste Etat.
- select/3 : select(Etat, Buts, But)
  - L'élément But est membre de la liste Buts, mais pas de la liste Etat
- atteint/2 : atteint(Operation, But)
  - La liste Ajouter de l'opération Operation contient l'élément But
- applique/3 : applique(EtatCourant, Operation, EtatSuivant)
  - La liste EtatSuivant contient les éléments de la liste EtatCourant, moins les éléments de la liste Supprimer de l'opération Operation, plus les éléments de la liste Ajouter de l'opération Operation

## Le planificateur :

```
% Cas d'arrêt : tous les buts sont satisfaits
plan(Etat, Buts, [], Etat) :- satisfait(Etat, Buts).

% Cas général : sélectionner un but non satisfait,
% le satisfaire, l'appliquer et recommencer
plan(EtatInitial, Buts, Plan, EtatFinal) :-
        append(SousPlan1, [ Operation | SousPlan2 ], Plan),
        select(EtatInitial, Buts, But),
        atteint(Operation, But),
        possible(Operation, Condition),
        plan(EtatInitial, Condition, SousPlan1, EtatIntermed1),
        applique(EtatIntermed1, Operaiton, EtatIntermed2),
        plan(EtatIntermed2, Buts, SousPlan2, EtatFinal).
```

# Exemple

|?-plan( [surtable(a), surtable(c), surtable(d), sur(b,a), sur(e,d), pince(vide), libre(b), libre(c), libre(e)], [sur(b,a), sur(a,c)], Plan, EtatFinal ).

Plan = [ depiler(b,a), empiler(b,e), ramasser(a), empiler(a,c), depiler(b,e), empiler(b,a) ] ,

EtatFinal = [ surtable(c), surtable(d), sur(e,d), sur(a,c), libre(e), sur(b,a), libre(b), tenir() ]



# Exemple

- Un premier retour-arrière est inévitable car :

  append(SousPlan1, [ Operation | SousPlan2 ], Plan)

  donne au premier essai :

  SousPlan1 = [] , Operation = _ , SousPlan2 = _ ,

  Plan = [Operation|SousPlan2]

  Ainsi, avec la première opération possible (ex : empiler(a,c), le premier appel récursif de plan/4 échoue :

  plan( [surtable(a), … ], [libre(c),tenir(a)], [], EtatIntermed1)

- Ce premier appel récursif de plan/4 échouera à cause de SousPlan1 jusqu'à ce que SousPlan1 contienne autant d'éléments que d'opérations nécessaires pour réaliser les conditions de l'opération à partir de l'état initial.

# Exercices

Donnez la réponse de PROLOG à la question suivante (en supposant que l'ordre des opérateurs est : empiler, déposer, ramasser, dépiler) :

| ?- plan( [surtable(a), sur(b,a), libre(b), tenir()], [sur(a,b)], **Plan**, **EF**).

Dans quel ordre les opérations faisant partie de la liste « Plan » sont-elles successivement ajoutées ?