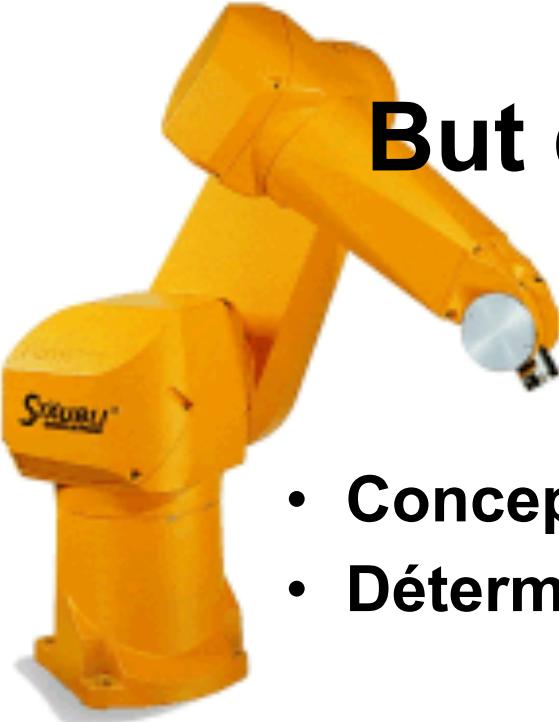


Planification

IAMSI Cours 7-8

Système STRIPS – Planification linéaire
Calcul des situations
Programmation en ASP
Planification non linéaire

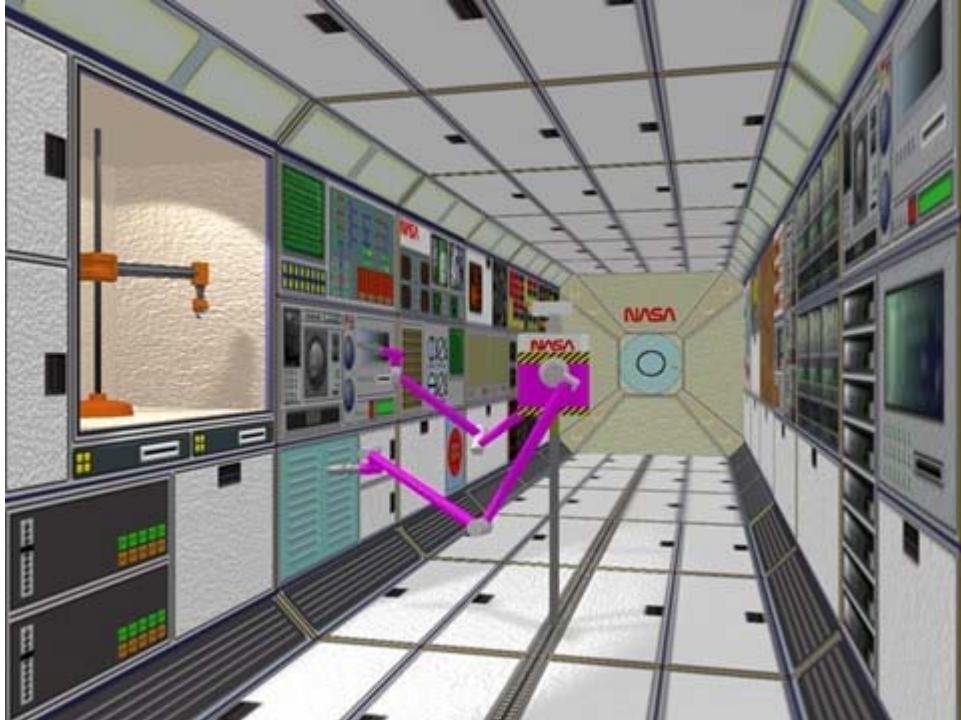




But de la planification

- Conception de robots manipulateurs
- Détermination de séquences d' actions





L'objet de la planification

- Construire une **séquence d'actions** qui réalise un **but** à partir d'un **état initial du monde**.

C
N
R
S

- Étant donné
 - Un ensemble d'opérateurs (définissant les actions possibles d'un agent),
 - Un état initial et
 - Un état but ou des conditions terminales,
- Calculer un plan, c'est-à-dire
 - Un séquence d'opérateurs telle que leur exécution à partir de l'état initial change le monde dans un état satisfaisant la description du but ou la condition terminale.
- Les buts sont généralement des conjonctions de buts réalisables



Planification versus résolution de problèmes



F3 robot

- La planification et les méthodes de résolution de problème traitent souvent les **mêmes sortes de problèmes**
- La planification est **plus puissante** du fait des méthodes et des représentations utilisées
- Les **états**, les **buts** et les **actions** se décomposent en **ensembles de propositions** (habituellement de **la logique du 1^{er} ordre**)
- La recherche opère souvent dans *l'espace des plans* plutôt que dans *l'espace des états* (*bien qu'il y ait des planificateurs à espaces d'états*)
- Les sous buts peuvent être **planifiés indépendamment**, ce qui réduit la complexité de la planification



Suppositions usuelles



- **Temps atomique**: chaque action est indivisible
- Les **actions concurrentes ne sont pas admises** (*bien que les actions n'aient pas toujours à être strictement ordonnées les unes par rapport aux autres dans le plan*)
- **Actions déterministes**: les résultats d'une action sont complètement déterminés – il n'y a pas d'incertitude sur les effets
- L'agent est la **seule cause de changement du monde**
- **L'agent est omniscient**: il dispose d'une connaissance totale et complète des états du monde
- “**Closed World Assumption**”: tout ce qui est connu pour être vrai est donné dans la description de l'état. Tout ce qui n'est pas explicité est supposé faux.





Action

- **Résolution de problèmes:**

Procédure

Par exemple, règles de production

- **Planification:**

Description transparente des conditions et des actions





États

- **Résolution de problèmes:**
Description intégrale de chaque état
Exemple, taquin (description de l'ensemble des positions des pièces)
- **Planification:**
Description partielle: seules les ***modifications*** ou les transformations sont indiquées





Buts

- **Résolution de problèmes:**
Conditions terminales – il n'est pas possible de les décomposer
« Boîte noire »
- **Planification:**
But « décomposable » en sous buts qui, eux-mêmes sont reliés aux actions
Exemple: « trouver du lait, des bananes et une perceuse sans fil »





Plans

- **Résolution de problèmes:**

Séquence d' actions

Exemple: « aller de la place Jussieu au métro Jussieu puis descendre prendre la ligne 10 jusqu'à Charles Michel sortir du métro, puis prendre la ligne 70 jusqu'au terminal – station Radio France... »

- **Planification:**

Donner un ensemble d' actions qui ne sont pas nécessairement ordonnées

Exemple: « descendre la station Charles Michel et prendre le bus 70 »





Trois idées clés de la planification

- **Représentations transparentes des états, des buts et des actions**
On utilise en général un langage formel, par exemple la logique des prédicats du premier ordre ou un sous-ensemble de celle-ci pour représenter les états, les buts et les actions. On fera en sorte que ce langage rende manifeste la connexion entre états et actions.
- **Le planificateur est libre d'ajouter des actions au plan lorsque c'est nécessaire.**
- **Il n'est pas nécessaire toujours partir de l'état initial pour constituer une séquence d'actions**

Par exemple, si l'on cherche du lait ne peut établir comme sous but « acheter du lait » et ajouter comme action « acheter du lait au supermarché »

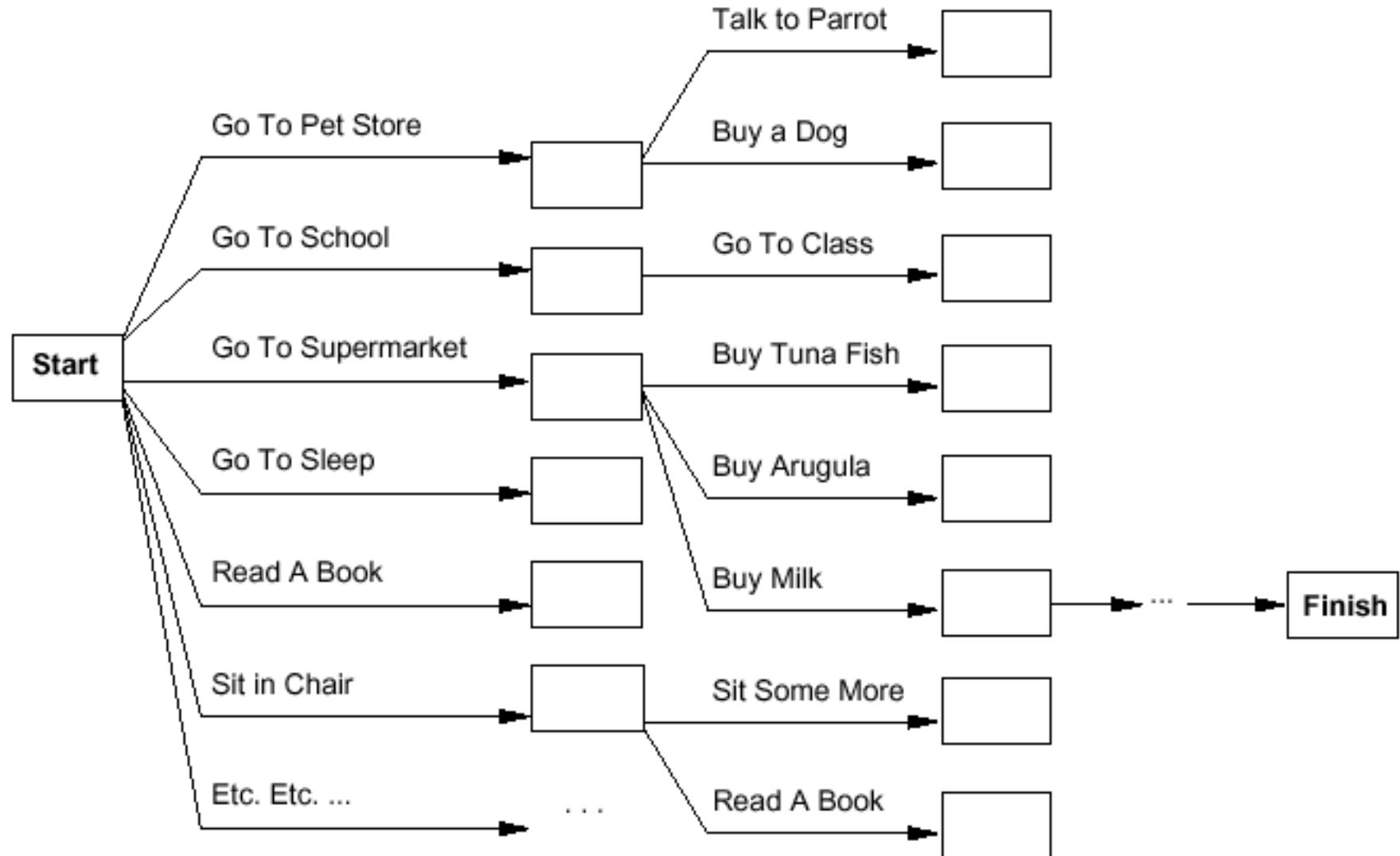
- **Le monde est décomposable en sous parties indépendantes**

Ainsi, si l'on à des buts conjonctifs du type chercher du lait, des bananes et une perceuse sans fil, le planificateur construira trois sous buts indépendants puis éventuellement il regroupera les deux premiers, puisque dans un cas comme dans l'autre il faut aller chez l'épicier.

Remarque : cette décomposition est impossible dans le cas du taquin



« Chercher du lait, des bananes et une perceuse sans fil »



Le monde des blocs est un micro monde qui consiste en une table, un ensemble de blocs et une poignée de robot.

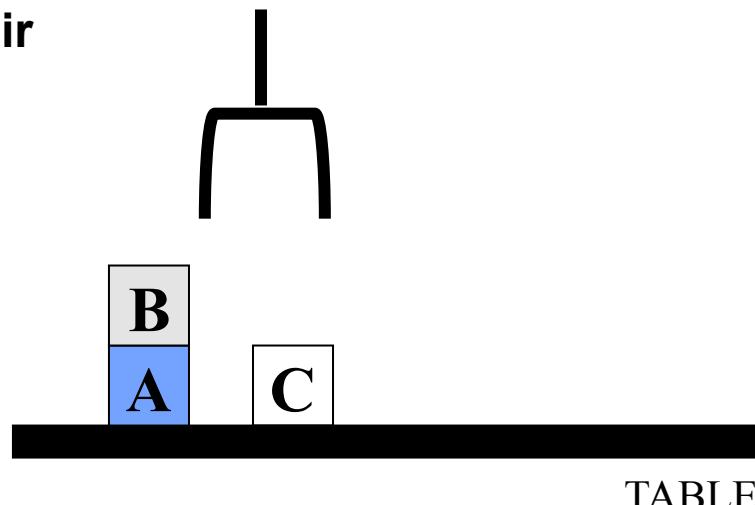
Contraintes du domaine:

- Il ne peut pas y avoir plus d'un bloc sur un autre
- Il n'y a pas de limite au nombre de blocs présents sur la table
- La poignée du robot ne peut saisir qu'un seul bloc

Représentation typique:

ontable(a)
ontable(c)
on(b,a)
handempty
clear(b)
clear(c)

Le monde des blocs



Approches principales

- **GPS / STRIPS**
- **Programmation en ASP**
- **Situation calculus**
- Planification partiellement ordonnée
- Décomposition hiérarchique
- **Planification avec contraintes**
- **Planification réactive**



General Problem Solver



- Le système “General Problem Solver” (GPS) fut le premier planificateur (Newell, Shaw, and Simon)
- GPS engendre des actions qui réduisent la différence entre l'état courant et le but
- GPS utilise une analyse fins-moyens
 - Comparer ce qui est donné ou connu avec ce qui est désiré et sélectionner ce qu'il faut faire pour combler cette différence
 - Utiliser une table de différences pour identifier des procédures qui réduisent chaque type de différence.
- GPS était un planificateur sur l'espace d' états: il opérait dans l' espace d' état spécifié par un état initial, des conditions terminales et un ensemble d' opérateurs



Représentations classiques pour la planification

- Approche classique utilisée dans STRIPS 1970
- Les états sont représentés par des conjonctions de littéraux instanciés
 - `at(Home) ^ ~have(Milk) ^ ~have(bananas) ...`
- Le buts sont des conjonctions de littéraux qui peuvent comprendre des variables (supposées existuellement quantifiées)
 - `at(?x) ^ have(Milk) ^ have(bananas) ...`
- Il n'est pas nécessaire de spécifier totalement les états
 - Les faits non spécifiés sont supposés faux ou n'être pas importants
 - Représentation de beaucoup de cas en peu d'espace
 - Représenter le changement plutôt que toutes les situations
- A la différence des démonstrateurs de théorème, il n'est pas nécessaire de prouver la véracité du but, mais il faut construire une séquence d'actions qui permet de l'atteindre

Représentation Opérateur/action

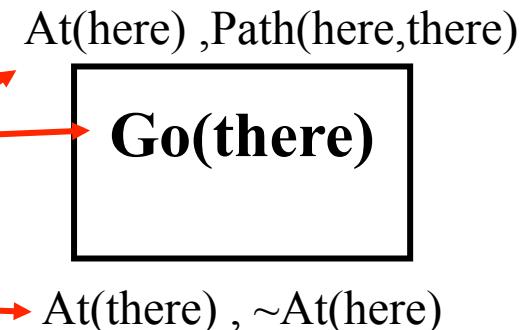
- **Les opérateurs ont trois composants:**
 - Action
 - Précondition – conjonction de littéraux positifs
 - Effet - conjonction de littéraux positifs ou négatifs qui décrivent comment la situation change quand l'opérateur est appliqué

- **Exemple:**

Op[Action: Go(there),

Precond: At(here) ^ Path(here,there),

Effect: At(there) ^ ~At(here)]



- **Toutes les variables sont universellement quantifiés**
- **Les variables de situation sont implicites**
 - La précondition doivent être vraie dans l' état antérieur à l' action; les effets dans l' état postérieur



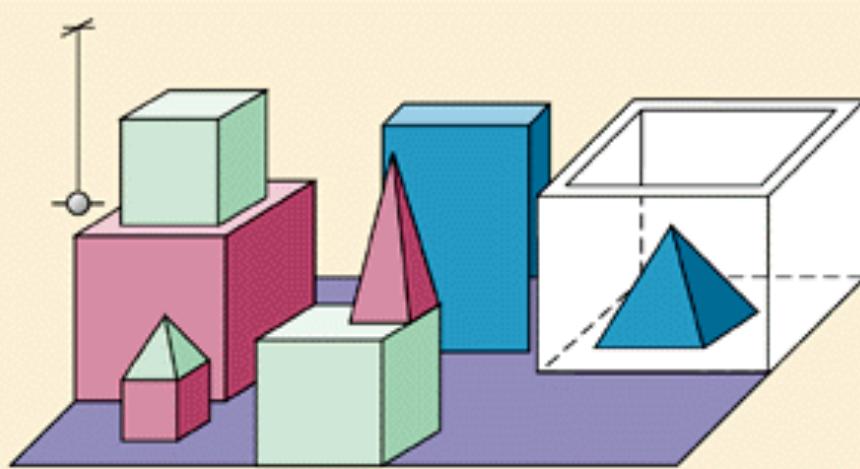
Opérateurs du monde des blocs

- Quatre opérateurs de base:
 - `stack(X,Y)`: mettre le bloc X sur le bloc Y
 - `unstack(X,Y)`: enlever le bloc X du bloc Y
 - `pickup(X)`: prendre le bloc X
 - `putdown(X)`: mettre le bloc X sur la table
- Chacun est représenté par
 - Une liste de préconditions
 - Un liste de faits à ajouter (ajout)
 - Un liste de faits à enlever (destruction)
 - Optionnellement, un ensemble de contraintes sur les variables
- Par exemple:

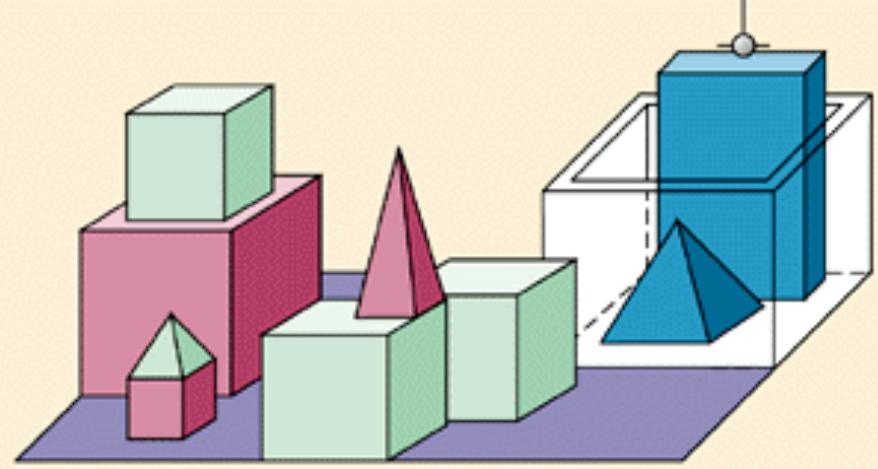
```
préconditions(stack(X,Y), [holding(X),clear(Y)])
destruction(stack(X,Y), [holding(X),clear(Y)]).
ajout(stack(X,Y), [handempty, on(X,Y), clear(X)])
contraintes(stack(X,Y), [X\==Y, Y\==table, X\==table])
```



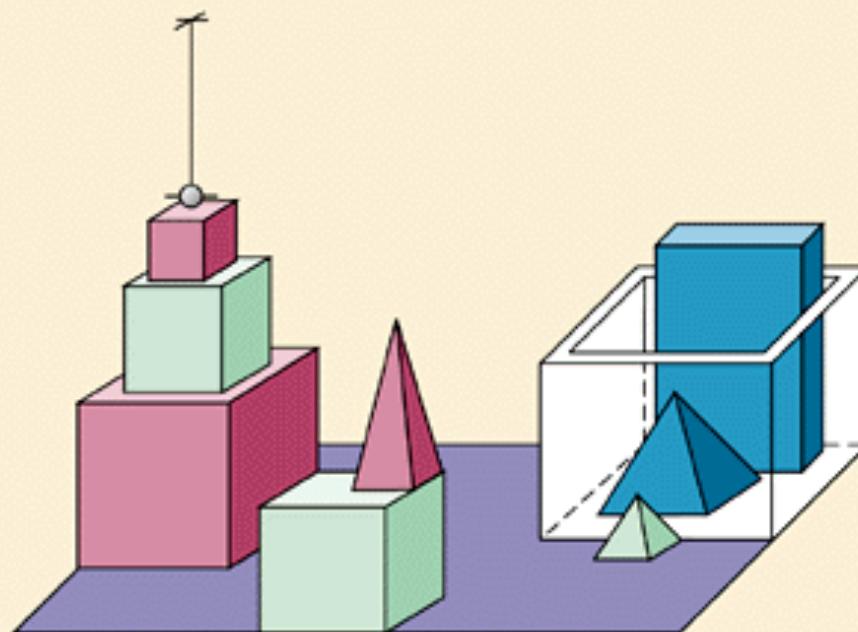
I



(a) "Pick up a big red block."



(b) "Find a block which is taller than the one you are holding and put it into the box."



(c) "Will you please stack up both of the red blocks and either a green cube or a pyramid?"

Monde des blocs - opérateurs

operator(stack(X,Y),

P [holding(X),clear(Y)],

Ajout [handempty,on(X,Y),clear(X)],

Destr [holding(X),clear(Y)],

Contr [X\==Y,Y\==table,
X\==table]).

operator(unstack(X,Y),

P [on(X,Y), clear(X), handempty],

A [holding(X),clear(Y)],

D [handempty,clear(X),on(X,Y)],

C [X\==Y,Y\==table,X\==table]).

operator(pickup(X),

P [ontable(X), clear(X), handempty],

A [holding(X)],

D [ontable(X),clear(X),handempty],

C [X\==table]).

operator(putdown(X),

P [holding(X)],

A [ontable(X),handempty,clear(X)],

D [holding(X)],

C [X\==table]).



Le planificateur STRIPS

- **STRIPS contient deux structures:**
 - **Etats** – ensemble de tous les prédictats vrais à un moment donné.
 - **Pile de buts** – une pile de buts à résoudre ou d'opérateurs, avec le **but courant** sur le sommet de la pile.
- **Algorithme**
 1. **Si le but courant n'est pas satisfait (sur l'état)**, examiner la liste d'ajout des opérateurs et insérer l'un des opérateurs avec les préconditions de cet opérateur sur la liste d' états (sous buts)
 2. **Quand le but courant est satisfait (sur l'état)**, le dépiler
 3. **Quand un opérateur est sur le sommet de la pile**, enregistrer son déclenchement sur la séquence du plan et utiliser les listes d' ajout et de destruction pour mettre à jour l'état courant



Planification : recherche

- Le planificateur considère tous les sous buts et essaye de les satisfaire chacun indépendamment:

$$G: \quad s_1 \wedge s_2 \wedge \dots \wedge s_n$$

- Séparation de la description des états et de la représentation la pile de buts
- La recherche se fait en chaînage avant (progression) ou en chaînage arrière (régression)
- La version initiale de STRIPS est une recherche en profondeur d'abord sur la pile des buts: satisfaire d'abord s_1 et ses sous buts, puis s_2 , etc...



Algorithme STRIPS

1. Ajouter la conjonction des buts à la pile
2. Tant que la pile n'est pas vide faire :
 - 2.1 si le but supérieur filtre vers l'état courant, alors dépiler
 - 2.2 sinon si le but composé ne filtre pas l'état courant, alors ajouter chacun des buts simples qui composent le but composé dans un ordre nouveau sur la pile
 - 2.3 sinon si le but est simple, trouver des règles dont les listes d'addition contiennent le but et
 1. Remplacer le but par la règle instanciée
 2. Placer les préconditions de la règle instanciée sur le sommet de la pile de buts
 - 2.4 enfin, si c'est une règle, l'enlever de la pile, la déclencher et mettre à jour à l'état





Propriétés de STRIPS

- Peut être progressif (chaînage avant) ou régressif (chaînage arrière) selon l'ordre dans lequel les buts et les règles sont ajoutées à la pile
- Toutes les séquences sont pas optimales
- STRIPS n'est pas complet: tous les plans possibles ne sont pas constructibles à l'aide de STRIPS
- STRIPS impose un ordre total sur les plans – cela peut accroître considérablement l'espace recherche

L

I

P

6

C

N

F

S

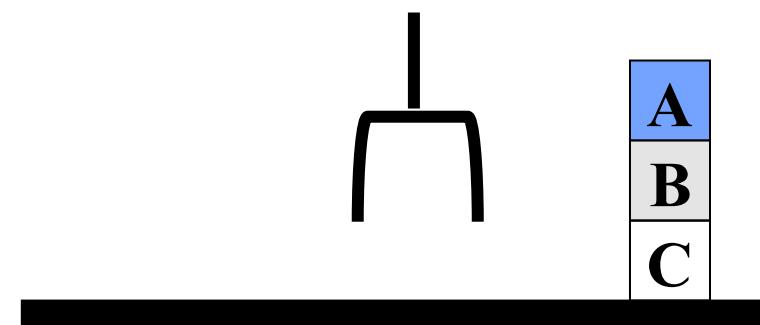
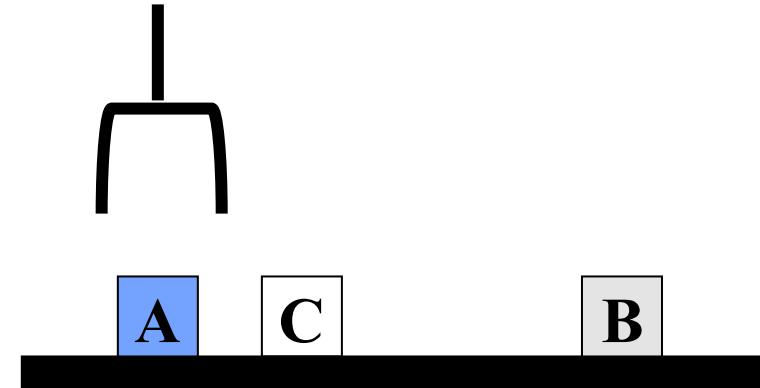
Plan typique

État initial:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Pile de buts:

on(b,c)
on(a,b)
ontable(c)



Plan:

pickup(b)
stack(b,c)
pickup(a)
stack(a,b)



Construction du plan

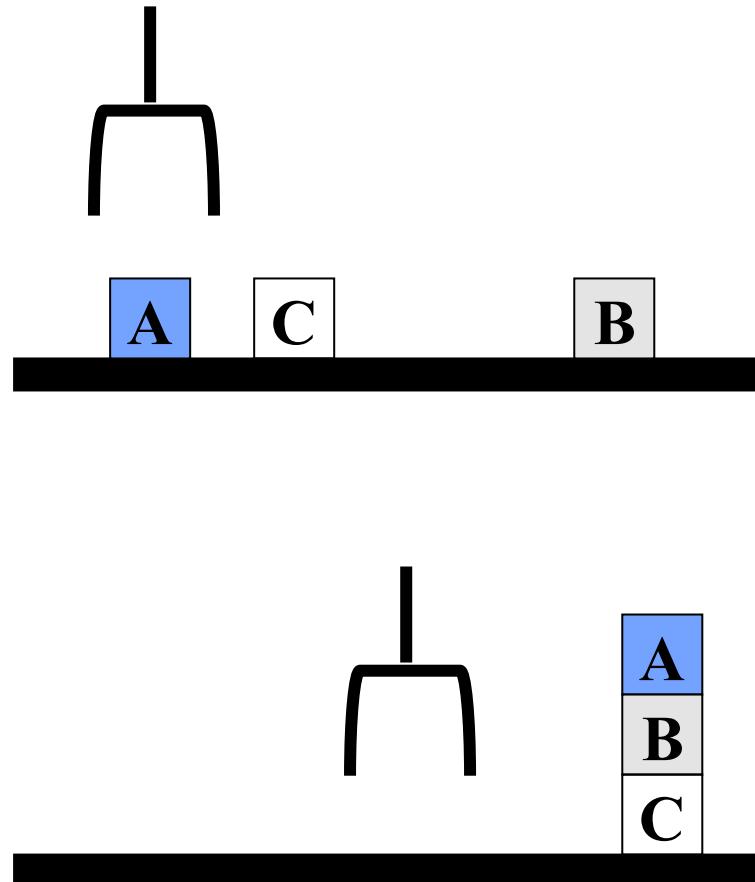
I
État initial:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Pile de buts:

on(b,c)
on(a,b)
ontable(c)

S



Plan:

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

État 1:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Pile de buts:

pickup(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
on(a,b)
ontable(c)



Jean-Gab

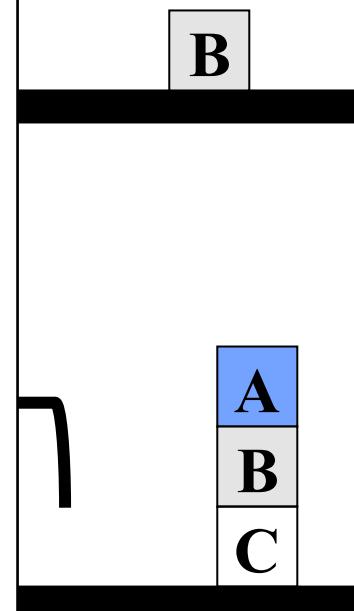
État 1:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Pile de buts:

ontable(b)
clear(b)
handempty
pickup(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
on(a,b)
ontable(c)

Exécution du plan



Plan:

```
operator(stack(X,Y),  
        Precond [holding(X),clear(Y)],  
        Ajout [handempty,on(X,Y),clear(X)],  
        Destr [holding(X),clear(Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(pickup(X),  
        Precond [ontable(X), clear(X), handempty],  
        Ajout [holding(X)],  
        Destr [ontable(X), clear(X), handempty],  
        Constr [X==table]).
```

```
operator(unstack(X,Y),  
        Precond [on(X,Y), clear(X), handempty],  
        Ajout [holding(X),clear(Y)],  
        Destr [handempty,clear(X),on(X,Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(putdown(X),  
        Precond [holding(X)],  
        Ajout [ontable(X),handempty,clear(X)],  
        Destr [holding(X)],  
        Constr [X==table]).
```

État 1:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Pile de buts:

ontable(b)
clear(b)
handempty
pickup(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
on(a,b)
ontable(c)

Construction du plan

État 2:

holding(b)
clear(a)
clear(c)
ontable(a)
ontable(c)

Pile de buts:

holding(b)
clear(c)
stack(b, c)
on(b,c)
on(a,b)
ontable(c)

État 3:

handempty
on(b, c)
clear(b)
clear(a)
ontable(a)
ontable(c)

Pile de buts:

on(a,b)
ontable(c)

Plan:

pickup(b)
stack(b, c)

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty, on(X,Y), clear(X)],
Destr [holding(X), clear(Y)],
Constr [X==Y, Y==table, X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X), clear(Y)],
Destr [handempty, clear(X), on(X,Y)],
Constr [X==Y, Y==table, X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).



État 3:

handempty
on(b, c)
clear(b)

Construction du plan

État 4:

handempty

clear(a)
ontable(a)
ontable(a)

Pile de buts:

on(a)
ontable(a)
ontable(c)

Pile de buts:

pickup(a)
holding(a)
clear(b)
stack(a, b)
on(a, b)
ontable(c)

État 5:

handempty
on(b, c)
clear(b)
clear(a)
ontable(a)
ontable(c)

Pile de buts:

holding(a)
clear(b)
stack(a, b)
on(a, b)
ontable(c)

Plan:

pickup(b)
stack(b, c)

operator(stack(X, Y),
Precond [holding(X), clear(Y)],
Ajout [handempty, on(X, Y), clear(X)],
Destr [holding(X), clear(Y)],
Constr [X==Y, Y==table, X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X, Y),
Precond [on(X, Y), clear(X), handempty],
Ajout [holding(X), clear(Y)],
Destr [handempty, clear(X), on(X, Y)],
Constr [X==Y, Y==table, X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

N
R
S



État 6:

handempty
on(b, c)
clear(b)
clear(a)
ontable(a)
ontable(c)

Pile de buts:

pickup(a)
holding(a)
clear(b)
stack(a, b)
on(a,b)
ontable(c)

R

S



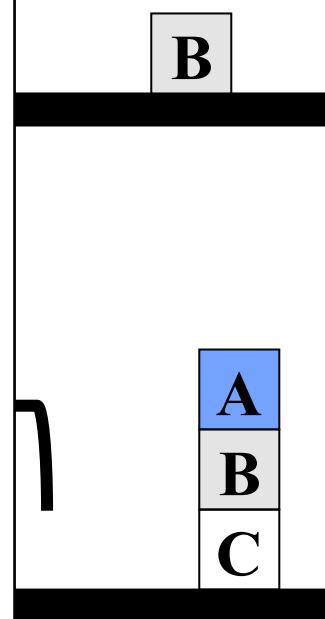
État 7:

handempty
on(b, c)
clear(b)
clear(a)
ontable(a)
ontable(c)

Pile de buts:

ontable(a)
clear(a)
handempty
pickup(a)
holding(a)
clear(b)
stack(a, b)
on(a,b)
ontable(c)

Exécution du plan



Plan:

pickup(b)
stack(b, c)

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X),clear(X),handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État 6:

handempty
on(b, c)
clear(b)
clear(a)
ontable(a)
ontable(c)

Pile de buts:

pickup(a)
holding(a)
clear(b)
stack(a, b)
on(a,b)
ontable(c)

R

S



État 7:

handempty
on(b, c)
clear(b)
clear(a)
ontable(a)
ontable(c)

Pile de buts:

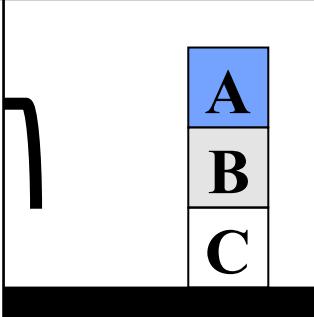
ontable(a)
clear(a)
handempty
pickup(a)
holding(a)
clear(b)
stack(a, b)
on(a,b)
ontable(c)

État 8:

holding(a)
on(b, c)
clear(b)
ontable(c)

Pile de buts:

holding(a)
clear(b)
stack(a, b)
on(a,b)
ontable(c)



Plan:

pickup(b)
stack(b, c)
pickup(a)

```
or(stack(X,Y),
  second [holding(X),clear(Y)],
  first [handempty,on(X,Y),clear(X)],
  third [holding(X),clear(Y)],
  fourth [X==Y,Y==table,X==table]).
```

```
or(pickup(X),
  second [ontable(X), clear(X), handempty],
  first [holding(X)],
  third [ontable(X),clear(X),handempty],
  fourth [X==table]).
```

```
or(unstack(X,Y),
  second [on(X,Y), clear(X), handempty],
  first [holding(X),clear(Y)],
  third [handempty,clear(X),on(X,Y)],
  fourth [X==Y,Y==table,X==table]).
```

```
operator(putdown(X),
  Precond [holding(X)],
  Ajout [ontable(X),handempty,clear(X)],
  Destr [holding(X)],
  Constr [X==table]).
```

État 8:

holding(a)
on(b, c)
clear(b)
ontable(c)

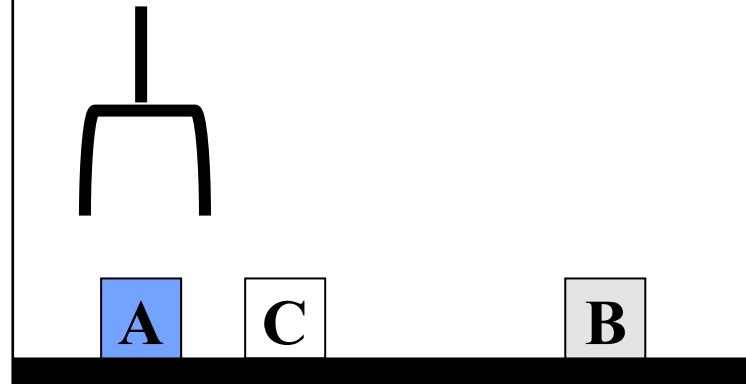
Pile de buts:

holding(a)
clear(b)
stack(a, b)
on(a,b)
ontable(c)

N
R
S



Construction du plan



Plan:

pickup(b)
stack(b, c)
pickup(a)

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destru [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destru [ontable(X),clear(X),handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destru [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destru [holding(X)],
Constr [X==table]).

État 8:

holding(a)
on(b, c)
clear(b)
ontable(c)

Pile de buts:

holding(a)
clear(b)
stack(a, b)
on(a,b)
ontable(c)

N
R
S

Construction du plan

État 9:

handempty
on(a, b)
clear(a)
on(b, c)
ontable(c)

Pile de buts:

on(a,b)
ontable(c)

État 10:

handempty
on(a, b)
clear(a)
on(b, c)
ontable(c)

Pile de buts:

Plan:

pickup(b)
stack(b, c)
pickup(a)
stack(a, b)

stack(X,Y),
bnd [holding(X),clear(Y)],
t [handempty,on(X,Y),clear(X)],
r [holding(X),clear(Y)],
str [X==Y,Y==table,X==table]).

pickup(X),
bnd [ontable(X), clear(X), handempty],
t [holding(X)],
r [ontable(X),clear(X),handempty],
str [X==table]).

unstack(X,Y),
bnd [on(X,Y), clear(X), handempty],
t [holding(X),clear(Y)],
r [handempty,clear(X),on(X,Y)],
str [X==Y,Y==table,X==table]).

putdown(X),
bnd [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

ARRÊT



État initial:

```
clear(a)  
clear(b)  
clear(c)  
ontable(a)  
ontable(b)  
ontable(c)  
handempty
```

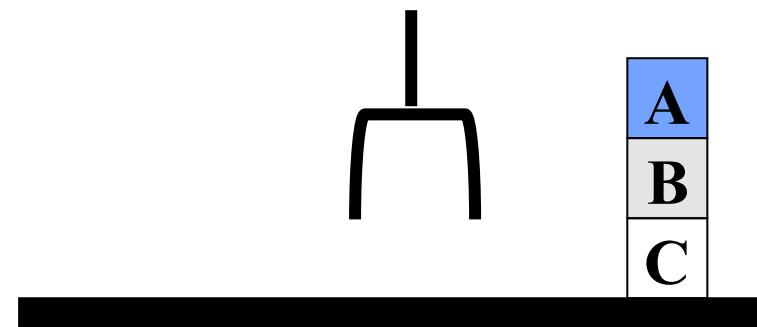
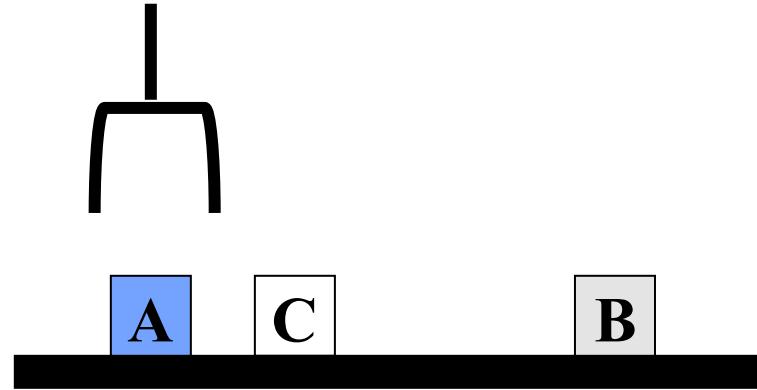
Pile de buts:

```
on(a,b)  
on(b,c)  
ontable(c)
```

R
S



Un autre problème



État initial:

```
clear(a)  
clear(b)  
clear(c)  
ontable(a)  
ontable(b)  
ontable(c)  
handempty
```

Pile de buts:

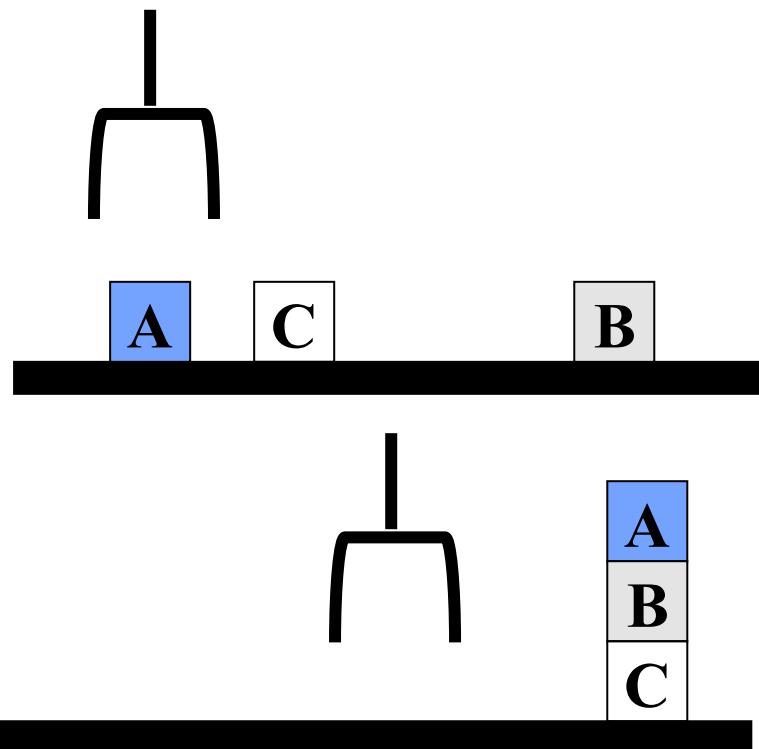
```
on(a,b)  
on(b,c)  
ontable(c)
```

R
S



Plan:

Un autre problème



```
operator(stack(X,Y),  
        Precond [holding(X),clear(Y)],  
        Ajout [handempty,on(X,Y),clear(X)],  
        Destr [holding(X),clear(Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(pickup(X),  
        Precond [ontable(X), clear(X), handempty],  
        Ajout [holding(X)],  
        Destr [ontable(X), clear(X), handempty],  
        Constr [X==table]).
```

```
operator(unstack(X,Y),  
        Precond [on(X,Y), clear(X), handempty],  
        Ajout [holding(X),clear(Y)],  
        Destr [handempty,clear(X),on(X,Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(putdown(X),  
        Precond [holding(X)],  
        Ajout [ontable(X),handempty,clear(X)],  
        Destr [holding(X)],  
        Constr [X==table]).
```

État initial:

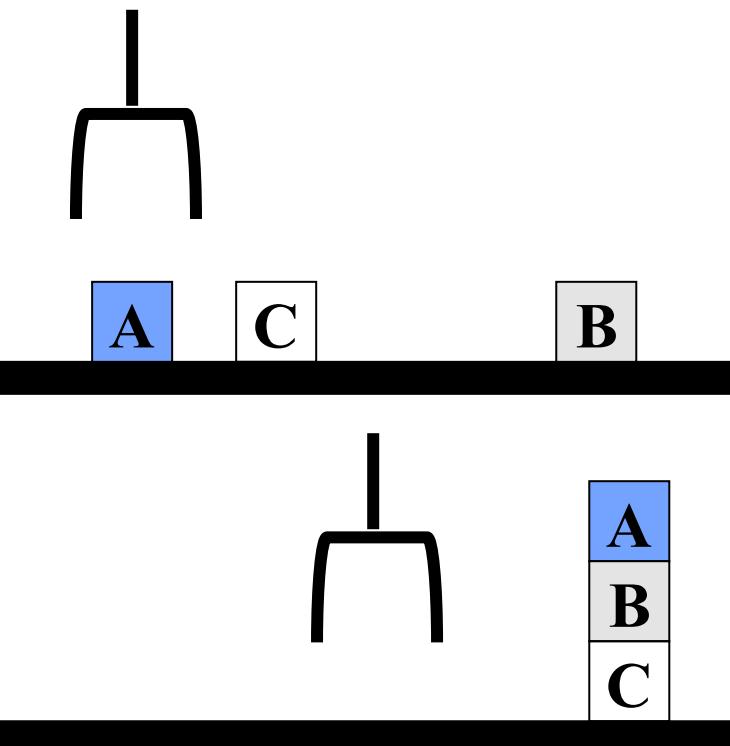
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Pile de buts:

holding(a)
clear(b)
stack(a,b)
on(a,b)
on(b,c)
ontable(c)

Plan:

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

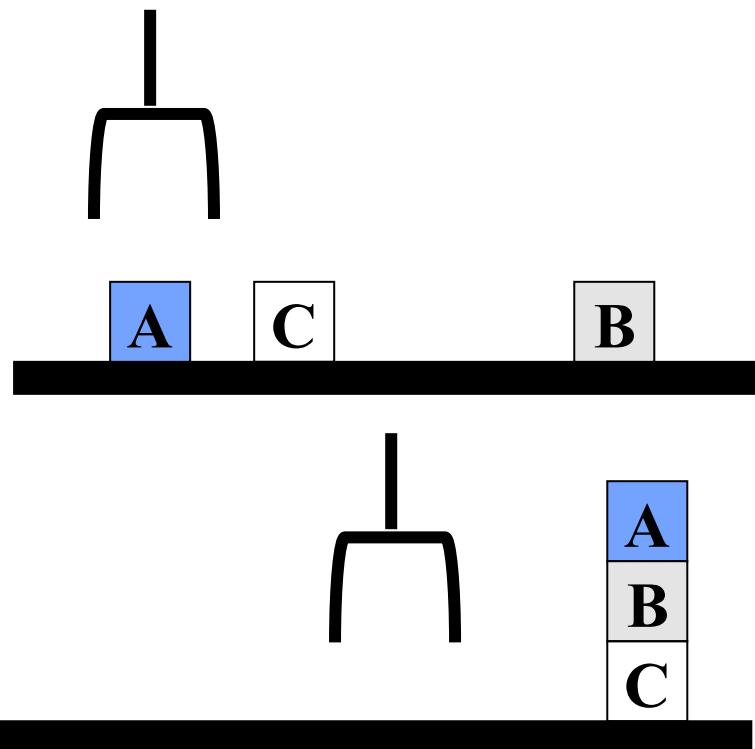
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Pile de buts:

ontable(a)
clear(a)
handempty
pickup(a)
holding(a)
clear(b)
stack(a,b)
on(a,b)
on(b,c)
ontable(c)

Plan:

Un autre problème



operator(stack(X,Y),
Precond [holding(X), clear(Y)],
Ajout [handempty, on(X,Y), clear(X)],
Destr [holding(X), clear(Y)],
Constr [X==Y, Y==table, X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X), clear(Y)],
Destr [handempty, clear(X), on(X,Y)],
Constr [X==Y, Y==table, X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

holding(a)
clear(b)
clear(c)
ontable(b)
ontable(c)

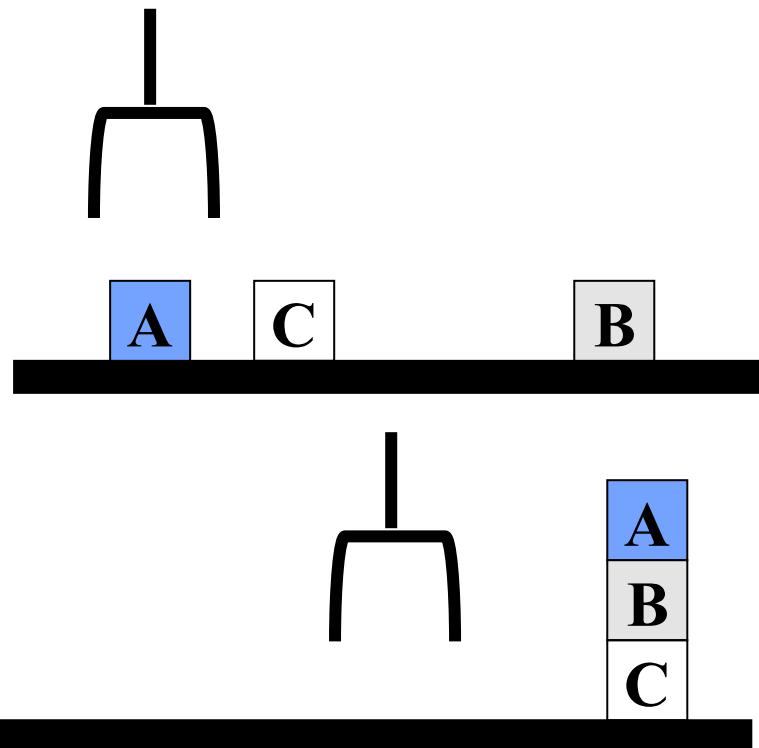
Pile de buts:

holding(a)
clear(b)
stack(a,b)
on(a,b)
on(b,c)
ontable(c)

Plan:

pickup(a)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

handempty
on(a, b)
clear(a)
clear(c)
ontable(b)
ontable(c)

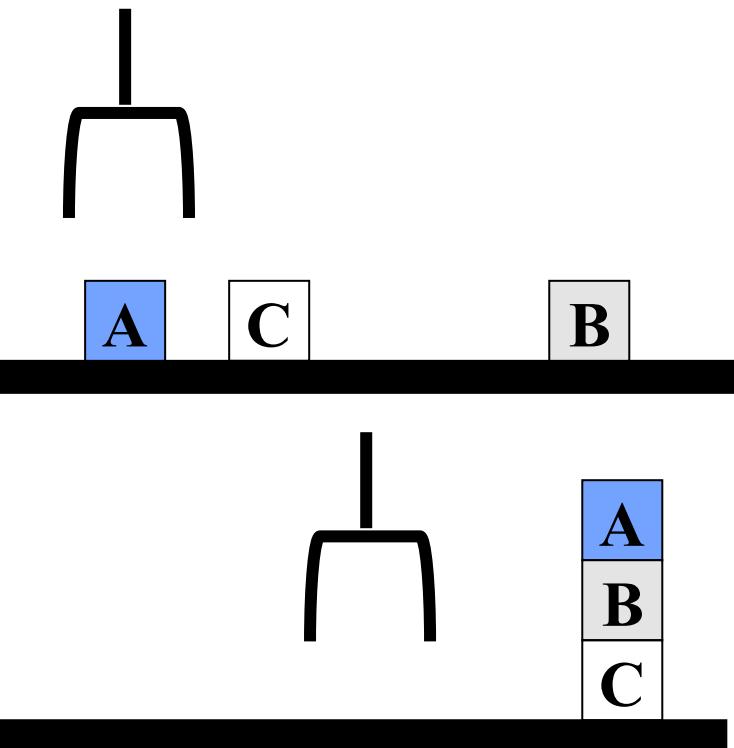
Pile de buts:

on(a,b)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)

Un autre problème



operator(stack(X,Y),
 Precond [holding(X),clear(Y)],
 Ajout [handempty,on(X,Y),clear(X)],
 Destr [holding(X),clear(Y)],
 Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
 Precond [ontable(X), clear(X), handempty],
 Ajout [holding(X)],
 Destr[ontable(X),clear(X),handempty],
 Constr [X==table]).

operator(unstack(X,Y),
 Precond [on(X,Y), clear(X), handempty],
 Ajout [holding(X),clear(Y)],
 Destr [handempty,clear(X),on(X,Y)],
 Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
 Precond[holding(X)],
 Ajout [ontable(X),handempty,clear(X)],
 Destr [holding(X)],
 Constr [X==table]).

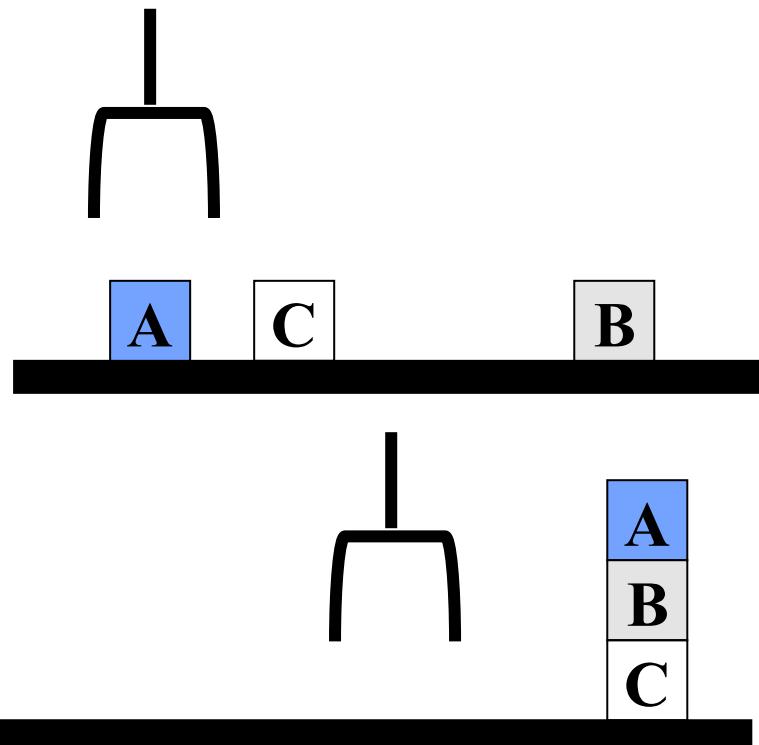


État initial:

handempty
on(a, b)
clear(a)
clear(c)
ontable(b)
ontable(c)

Pile de buts:

on(b,c)
ontable(c)



Plan:

pickup(a)
stack(a,b)

Un autre problème

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

handempty
on(a, b)
clear(a)
clear(c)
ontable(b)
ontable(c)

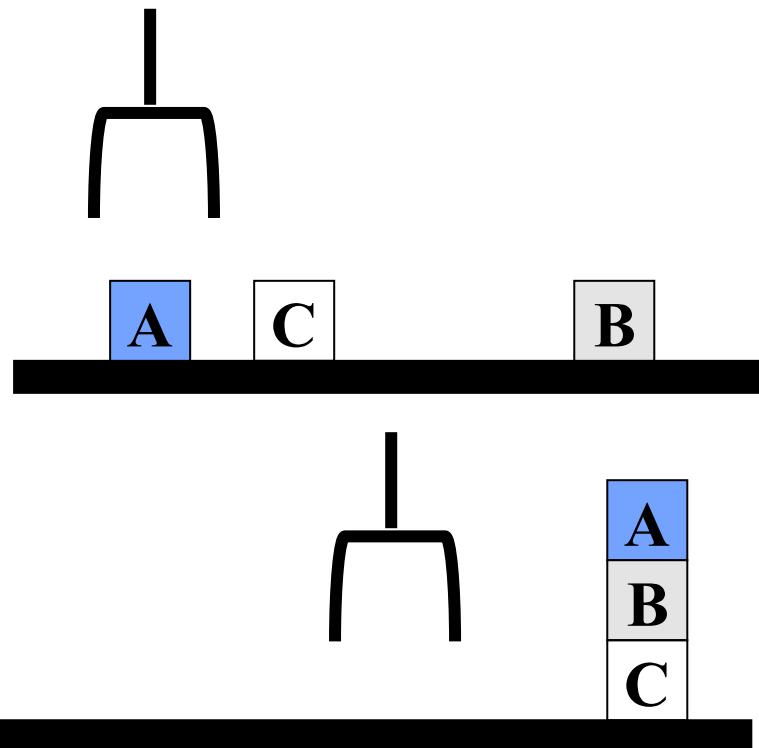
Pile de buts:

holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

handempty
on(a, b)
clear(a)
clear(c)
ontable(b)
ontable(c)

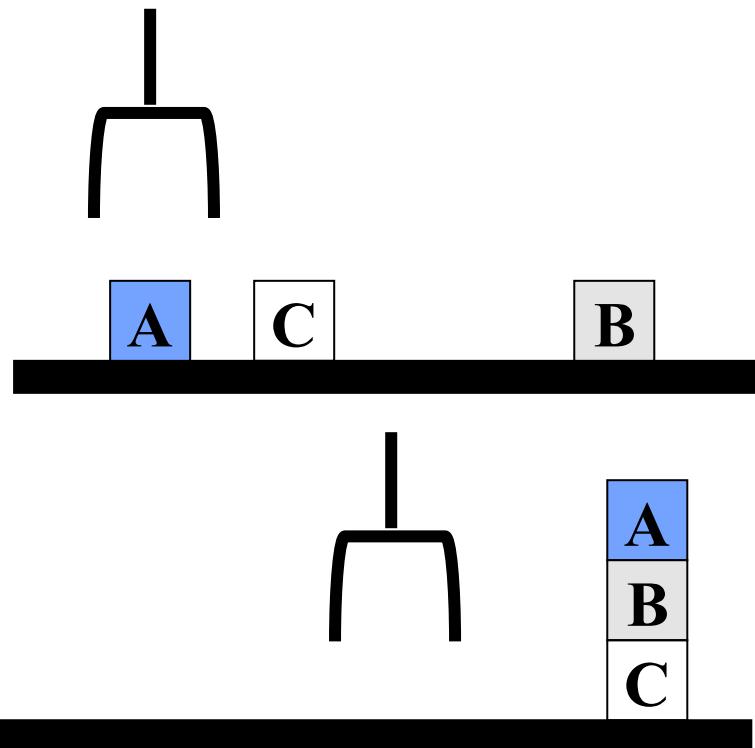
Pile de buts:

ontable(b)
clear(b)
handempty
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

handempty
on(a, b)
clear(a)
clear(c)
ontable(b)
ontable(c)

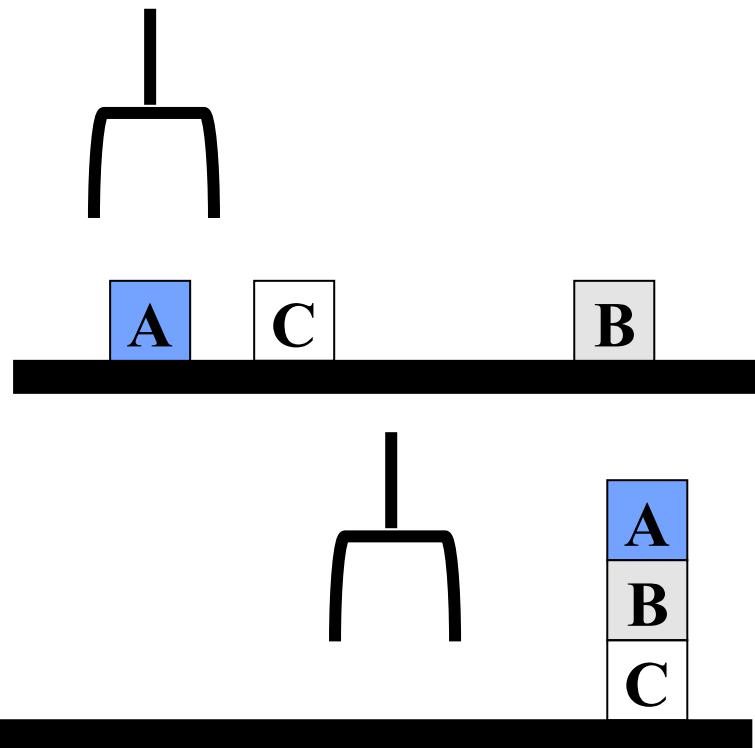
Pile de buts:

unstack(a, b)
ontable(b)
clear(b)
handempty
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).



État initial:

handempty
on(a, b)
clear(a)
clear(c)
ontable(b)
ontable(c)

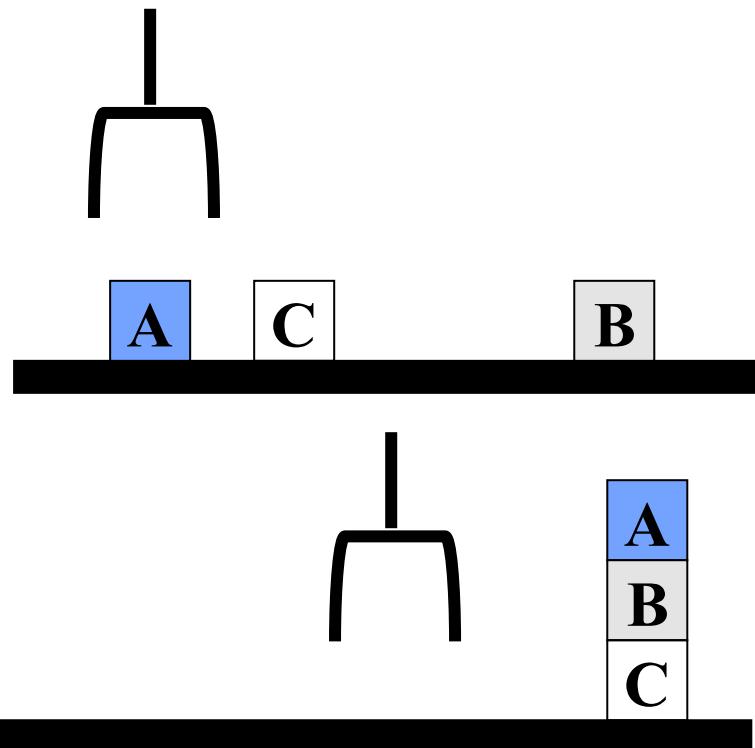
Pile de buts:

on(a, b)
clear(a)
handempty
unstack(a, b)
ontable(b)
clear(b)
handempty
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

holding(a)
clear(b)
clear(c)
ontable(b)
ontable(c)

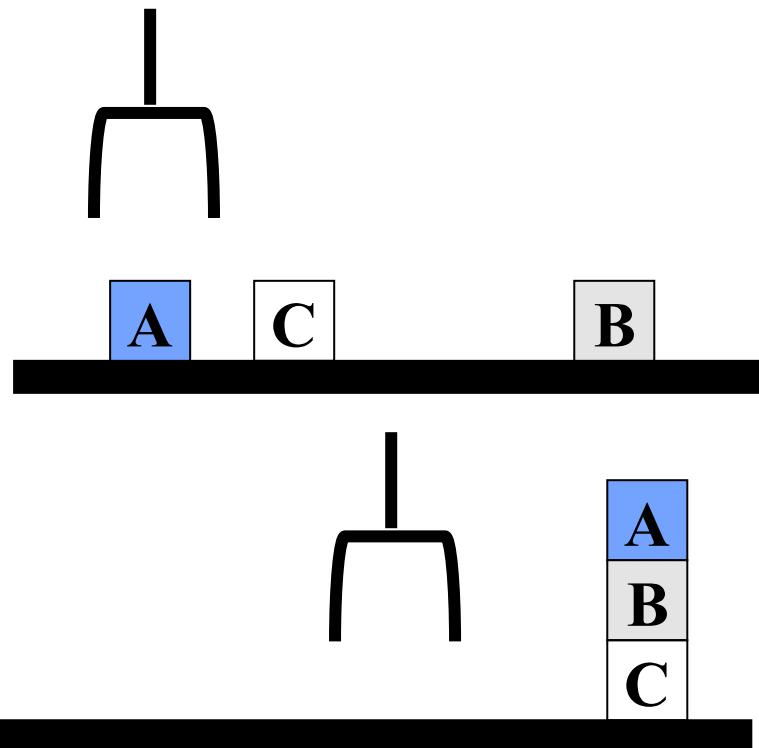
Pile de buts:

ontable(b)
clear(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

holding(a)
clear(b)
clear(c)
ontable(b)
ontable(c)

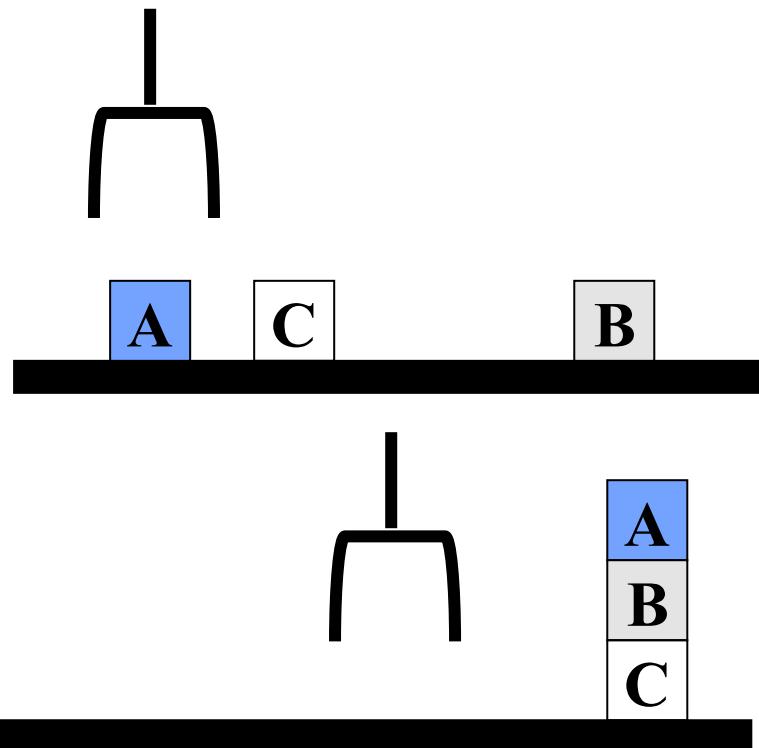
Pile de buts:

putdown(a)
ontable(b)
clear(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

holding(a)
clear(b)
clear(c)
ontable(b)
ontable(c)

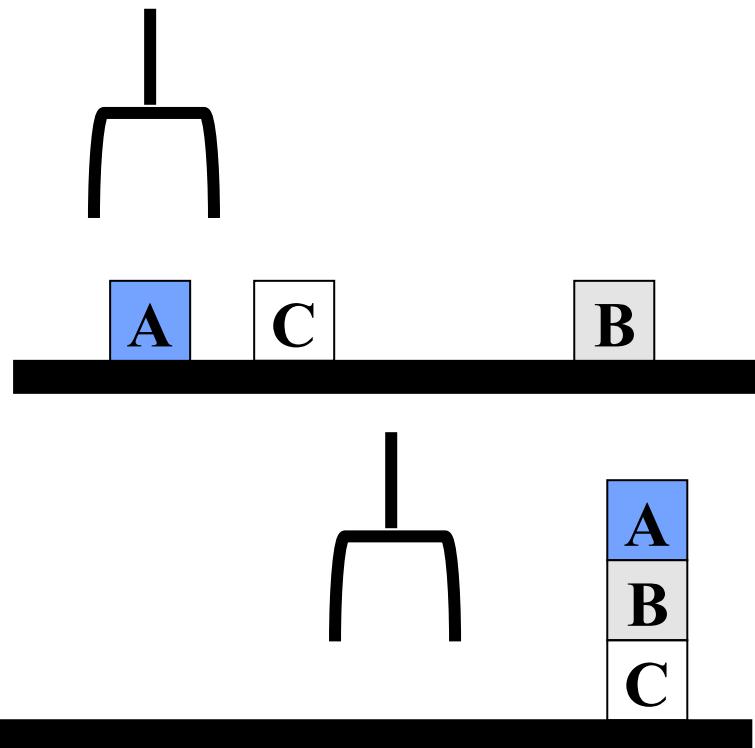
Pile de buts:

holding(a)
putdown(a)
ontable(b)
clear(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

ontable(a)
handempty
clear(a)
clear(b)
clear(c)
ontable(b)
ontable(c)

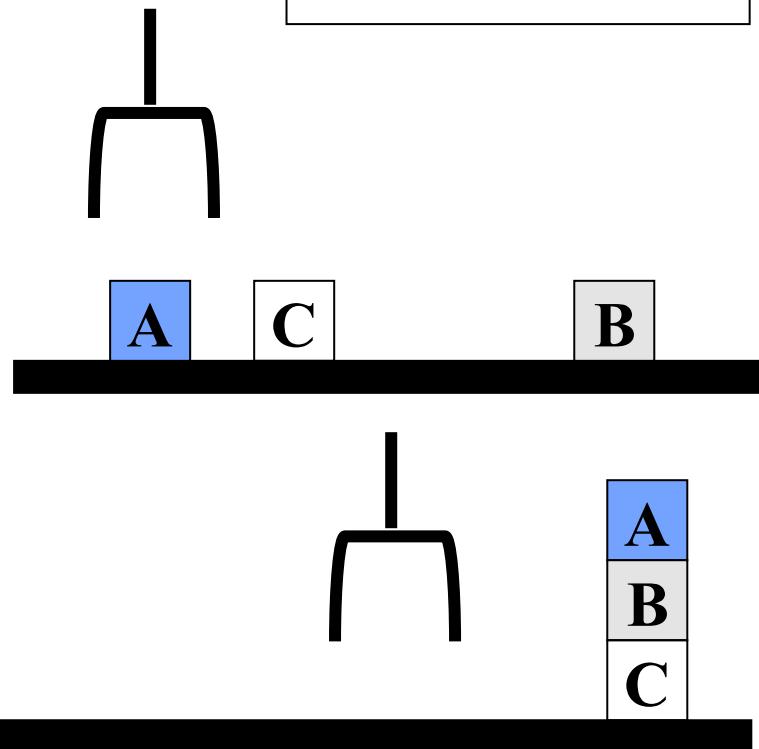
Pile de buts:

holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

ontable(a)
handempty
clear(a)
clear(b)
clear(c)
ontable(b)
ontable(c)

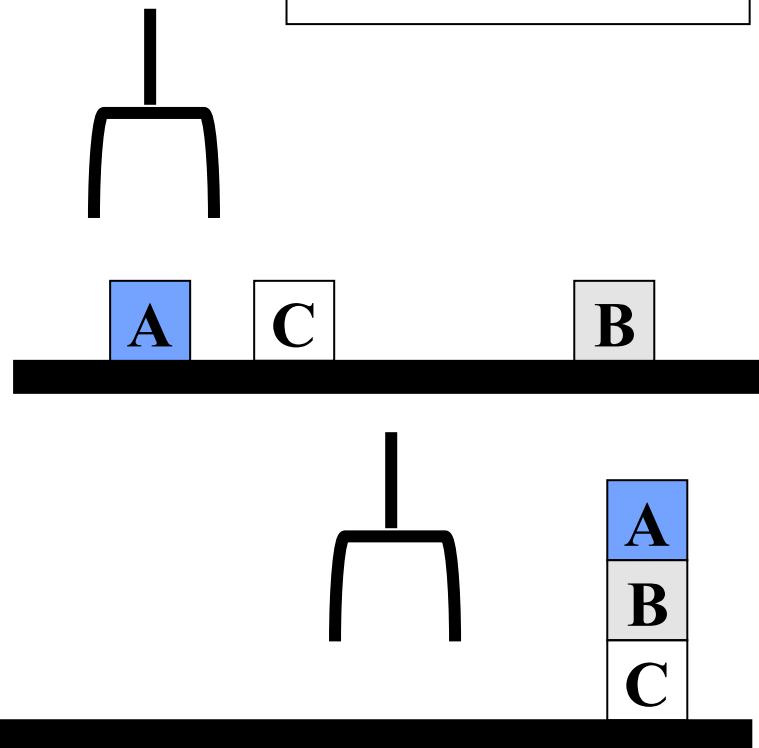
Pile de buts:

pickup(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

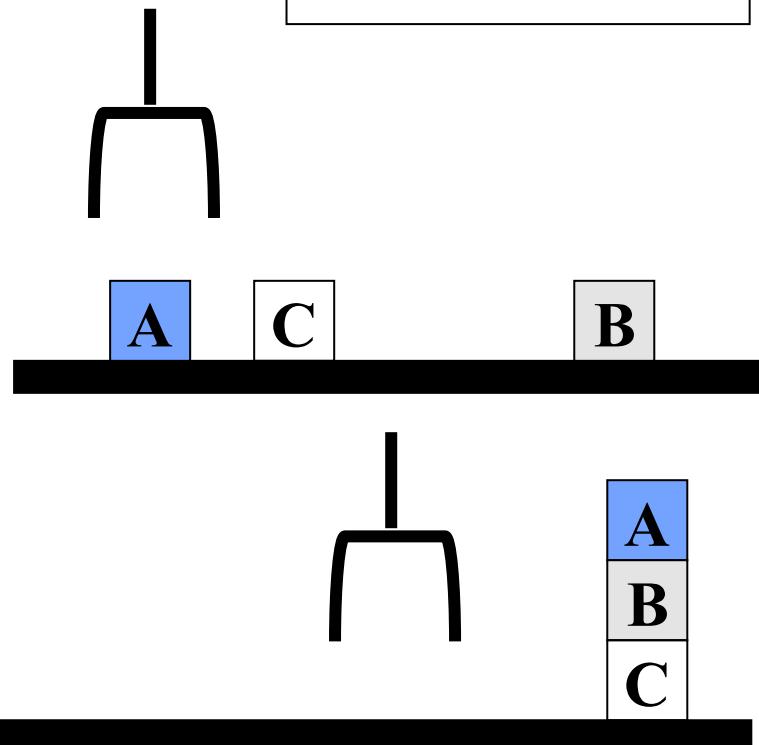
ontable(a)
handempty
clear(a)
clear(b)
clear(c)
ontable(b)
ontable(c)

Pile de buts:

clear(b)
handempty
ontable(b)
pickup(b)
holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)



Un autre problème

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

holding(b)
ontable(a)
clear(a)
clear(c)
ontable(c)

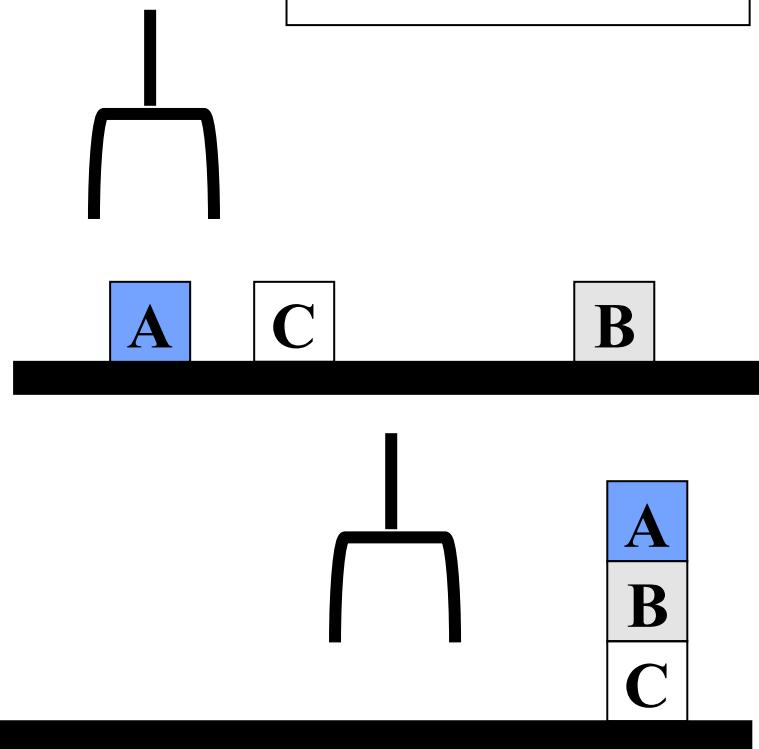
Pile de buts:

holding(b)
clear(c)
stack(b, c)
on(b,c)
ontable(c)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)

Un autre problème



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr[ontable(X),clear(X),handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

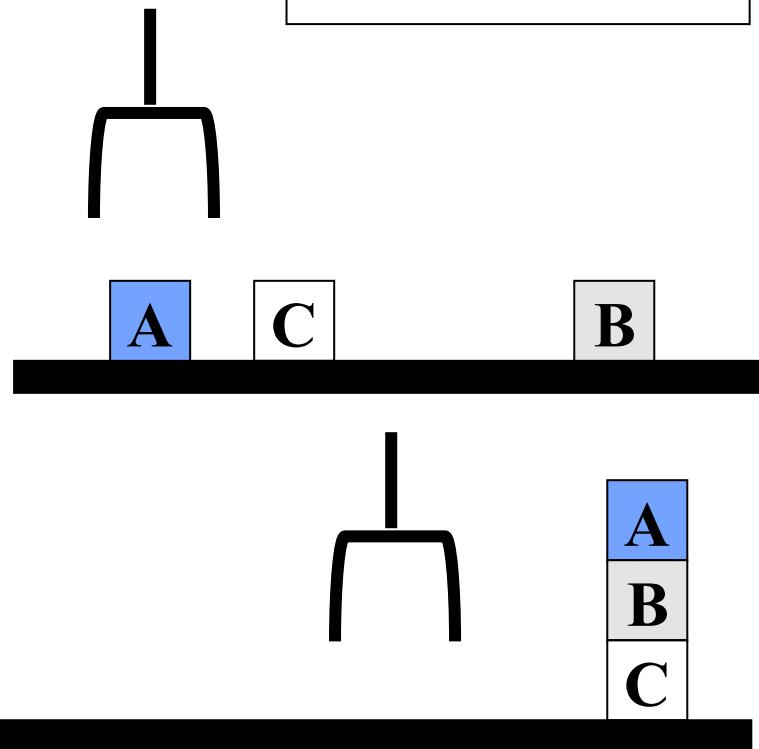
operator(putdown(X),
Precond[holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

handempty
on(b, c)
clear(b)
ontable(a)
clear(a)
ontable(c)

Pile de buts:

on(b,c)
ontable(c)



Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)

Un autre problème

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

handempty
on(b, c)
clear(b)
ontable(a)
clear(a)
ontable(c)

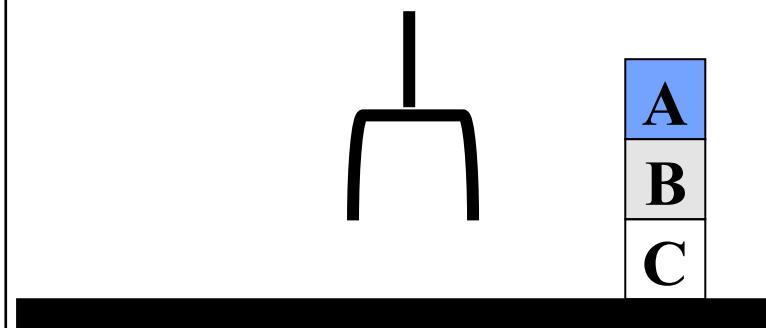
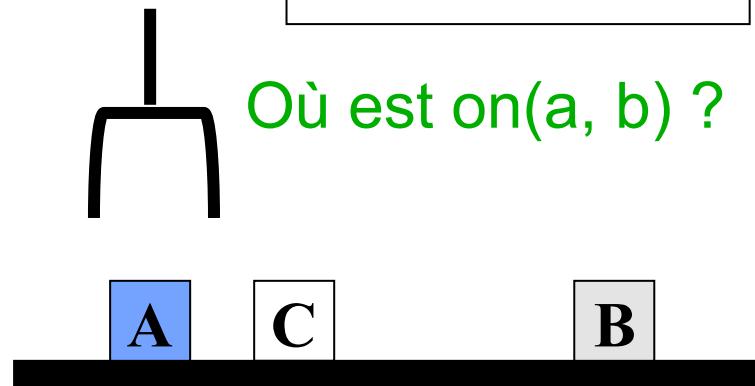
Pile de buts:

?

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)

Un autre problème



```
operator(stack(X,Y),  
        Precond [holding(X),clear(Y)],  
        Ajout [handempty,on(X,Y),clear(X)],  
        Destr [holding(X),clear(Y)],  
        Constr [X==Y,Y==table,X==table]).
```

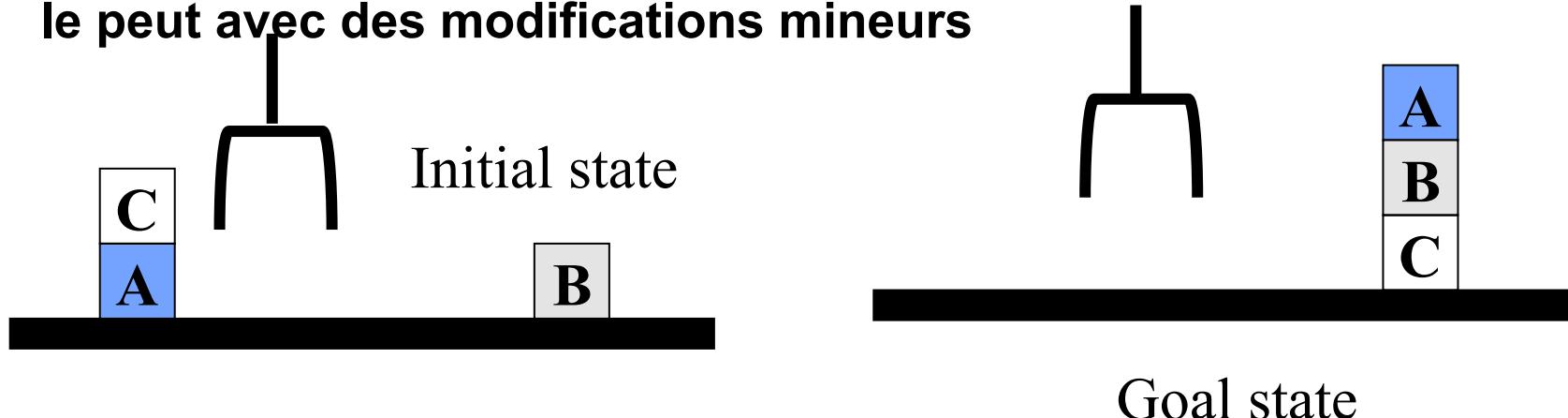
```
operator(pickup(X),  
        Precond [ontable(X), clear(X), handempty],  
        Ajout [holding(X)],  
        Destr [ontable(X), clear(X), handempty],  
        Constr [X==table]).
```

```
operator(unstack(X,Y),  
        Precond [on(X,Y), clear(X), handempty],  
        Ajout [holding(X),clear(Y)],  
        Destr [handempty,clear(X),on(X,Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(putdown(X),  
        Precond [holding(X)],  
        Ajout [ontable(X),handempty,clear(X)],  
        Destr [holding(X)],  
        Constr [X==table]).
```

Interaction des buts

- Les algorithmes de planification simples supposent que les buts qui doivent être réalisés sont indépendants
 - Chacun peut être réalisé indépendamment et ensuite ils sont concaténés dans la solution
- Ce problème appelé “l'anomalie de Sussman” est une exemple classique d’interaction de buts:
 - Résoudre $\text{on}(A,B)$ d’abord (en faisant $\text{unstack}(C,A)$, $\text{stack}(A,B)$) sera défaire en résolvant le second but $\text{on}(B,C)$ (en faisant $\text{unstack}(A,B)$, $\text{stack}(B,C)$).
 - Résoudre $\text{on}(B,C)$ sera défaire en résolvant $\text{on}(A,B)$
- Le système classique STRIPS ne peut résoudre ce problème. On le peut avec des modifications mineurs



État initial:

handempty
on(b, c)
clear(b)
ontable(a)
clear(a)
ontable(c)

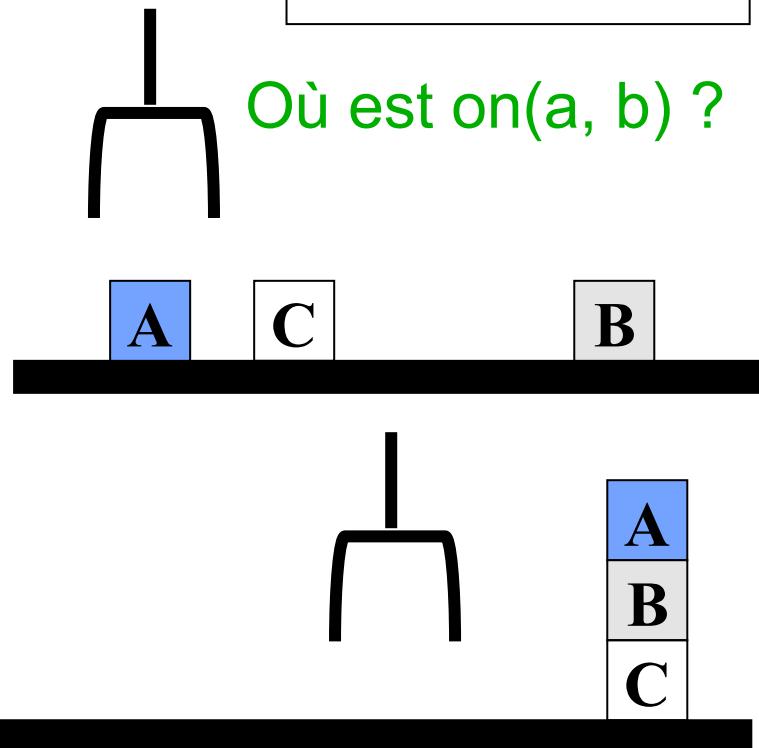
Pile de buts:

on(a, b) &
on(b, c) &
ontable(c) &
clear(a)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)

Solution:
ajouter au
fond de la
pile de but,
le but
conjoint...



```
operator(stack(X,Y),  
        Precond [holding(X),clear(Y)],  
        Ajout [handempty,on(X,Y),clear(X)],  
        Destr [holding(X),clear(Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(pickup(X),  
        Precond [ontable(X), clear(X), handempty],  
        Ajout [holding(X)],  
        Destr [ontable(X), clear(X), handempty],  
        Constr [X==table]).
```

```
operator(unstack(X,Y),  
        Precond [on(X,Y), clear(X), handempty],  
        Ajout [holding(X),clear(Y)],  
        Destr [handempty,clear(X),on(X,Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(putdown(X),  
        Precond [holding(X)],  
        Ajout [ontable(X),handempty,clear(X)],  
        Destr [holding(X)],  
        Constr [X==table]).
```

État initial:

handempty
on(b, c)
clear(b)
ontable(a)
clear(a)
ontable(c)

Pile de buts:

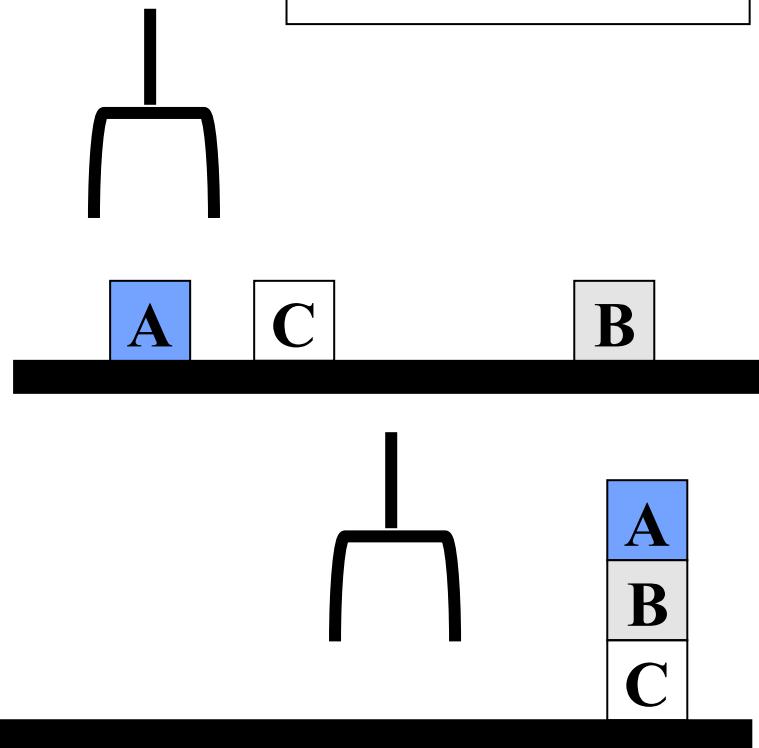
on(a, b)
on(b, c)
ontable(c)
clear(a)

on(a, b) &
on(b, c) &
ontable(c) &
clear(a)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)

Poursuite planif!



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

handempty
on(b, c)
clear(b)
ontable(a)
clear(a)
ontable(c)

Pile de buts:

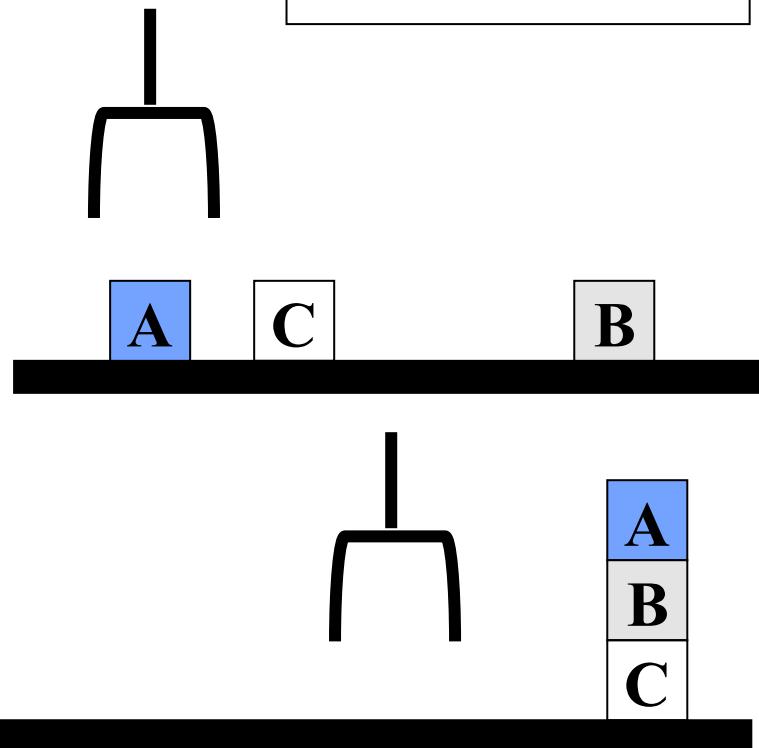
stack(a, b)
on(a, b)
on(b, c)
ontable(c)
clear(a)

on(a, b) &
on(b, c) &
ontable(c) &
clear(a)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)

Poursuite planif!



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

handempty
on(b, c)
clear(b)
ontable(a)
clear(a)
ontable(c)

Pile de buts:

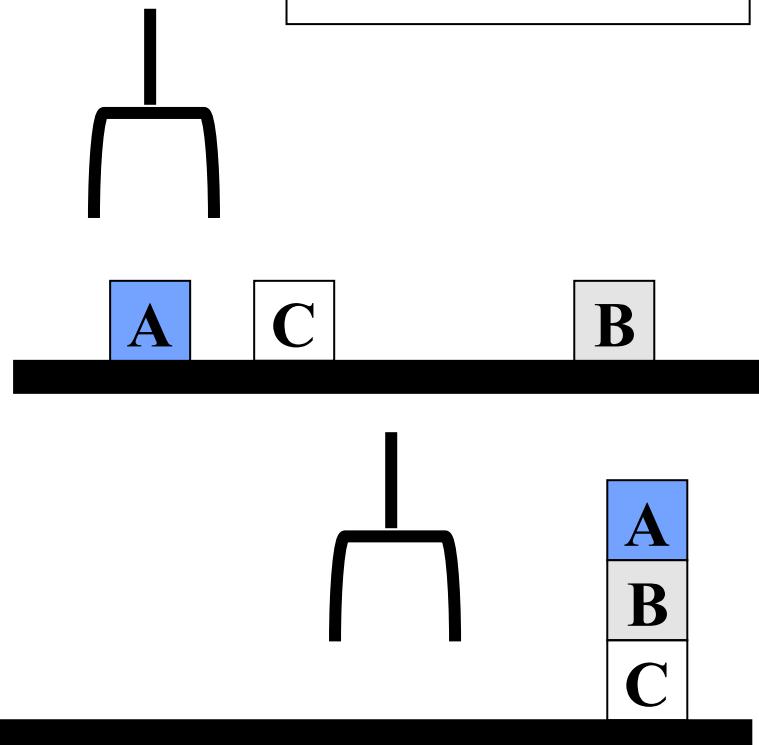
holding(a)
clear(b)
stack(a, b)
on(a, b)
on(b, c)
ontable(c)
clear(a)

on(a, b) &
on(b, c) &
ontable(c) &
clear(a)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)

Poursuite planif!



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

handempty
on(b, c)
clear(b)
ontable(a)
clear(a)
ontable(c)

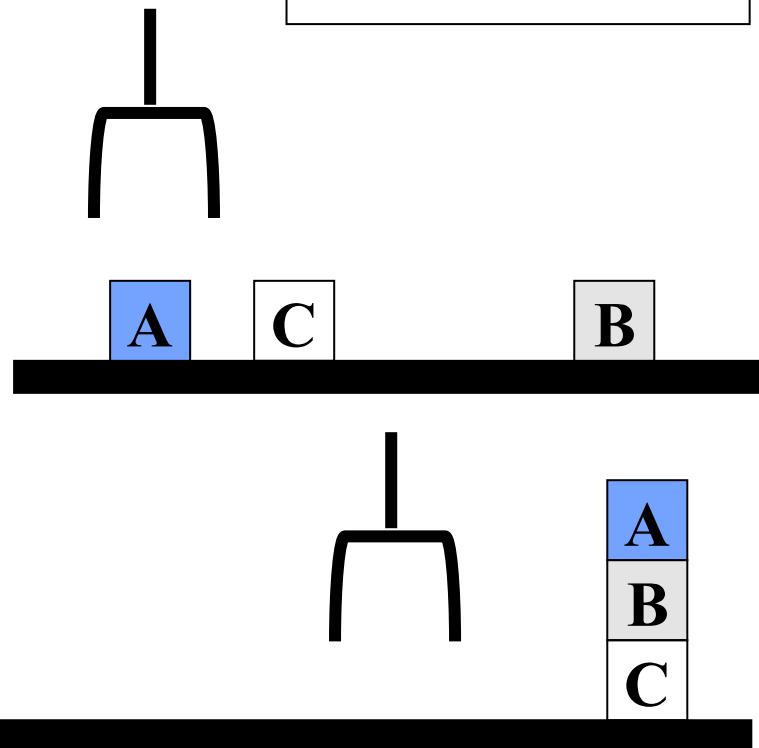
Pile de buts:

pickup(a)
holding(a)
clear(b)
stack(a, b)
on(a, b)
on(b, c)
ontable(c)
clear(a)
on(a, b) &
on(b, c) &
ontable(c) &
clear(a)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)

Poursuite planif!



operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

holding(a)
on(b, c)
clear(b)
ontable(c)

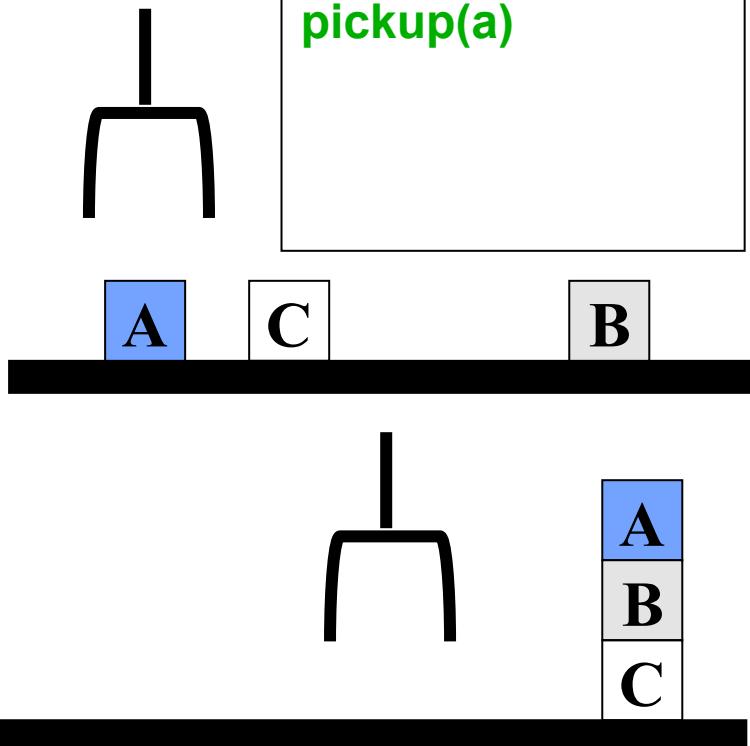
Pile de buts:

holding(a)
clear(b)
stack(a, b)
on(a, b)
on(b, c)
ontable(c)
clear(a)

on(a, b) &
on(b, c) &
ontable(c) &
clear(a)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)
pickup(a)



Poursuite planif!

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty,on(X,Y),clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y,Y==table,X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr[ontable(X),clear(X),handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X),clear(Y)],
Destr [handempty,clear(X),on(X,Y)],
Constr [X==Y,Y==table,X==table]).

operator(putdown(X),
Precond[holding(X)],
Ajout [ontable(X),handempty,clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

on(a, b)
clear(a)
handempty
on(b, c)
ontable(c)

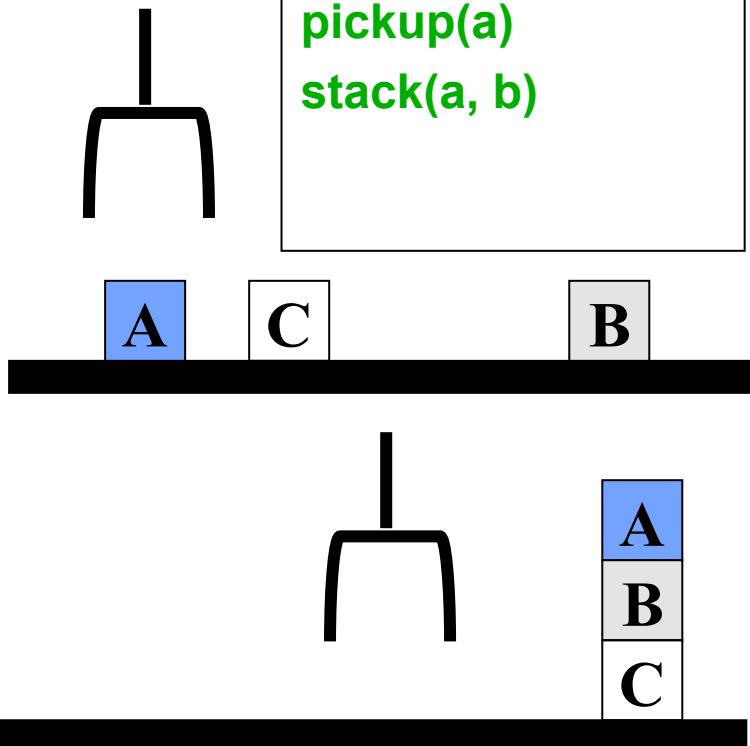
Pile de buts:

on(a, b)
on(b, c)
ontable(c)
clear(a)

on(a, b) &
on(b, c) &
ontable(c) &
clear(a)

Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)
pickup(a)
stack(a, b)



Poursuite planif!

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty, on(X,Y), clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y, Y==table, X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X), clear(Y)],
Destr [handempty, clear(X), on(X,Y)],
Constr [X==Y, Y==table, X==table]).

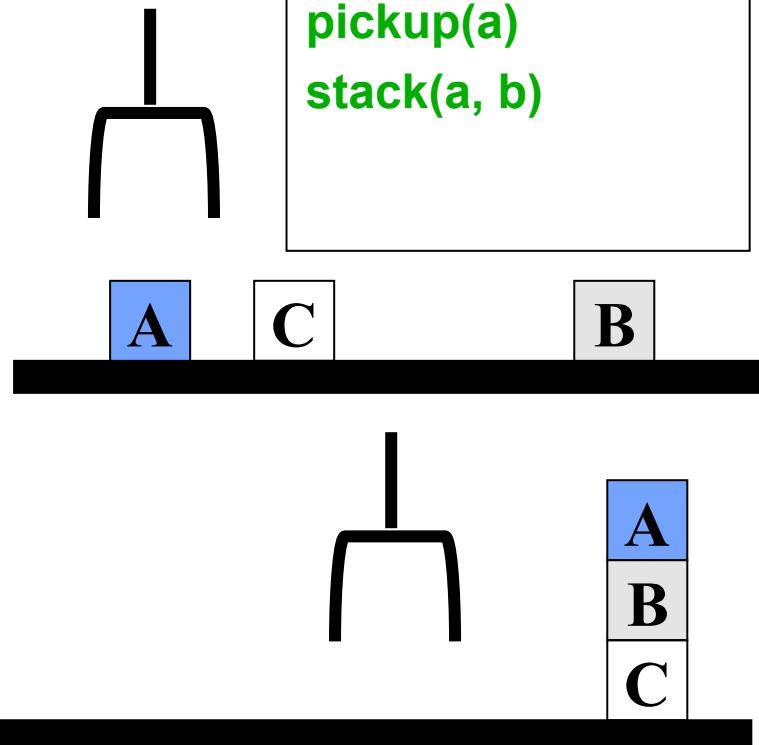
operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

État initial:

on(a, b)
clear(a)
handempty
on(b, c)
ontable(c)

Pile de buts:

on(a, b) &
on(b, c) &
ontable(c) &
clear(a)



Plan:

pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)
pickup(a)
stack(a, b)

Poursuite planif!

```
operator(stack(X,Y),  
        Precond [holding(X),clear(Y)],  
        Ajout [handempty,on(X,Y),clear(X)],  
        Destr [holding(X),clear(Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(pickup(X),  
        Precond [ontable(X), clear(X), handempty],  
        Ajout [holding(X)],  
        Destr[ontable(X),clear(X),handempty],  
        Constr [X==table]).
```

```
operator(unstack(X,Y),  
        Precond [on(X,Y), clear(X), handempty],  
        Ajout [holding(X),clear(Y)],  
        Destr [handempty,clear(X),on(X,Y)],  
        Constr [X==Y,Y==table,X==table]).
```

```
operator(putdown(X),  
        Precond[holding(X)],  
        Ajout [ontable(X),handempty,clear(X)],  
        Destr [holding(X)],  
        Constr [X==table]).
```

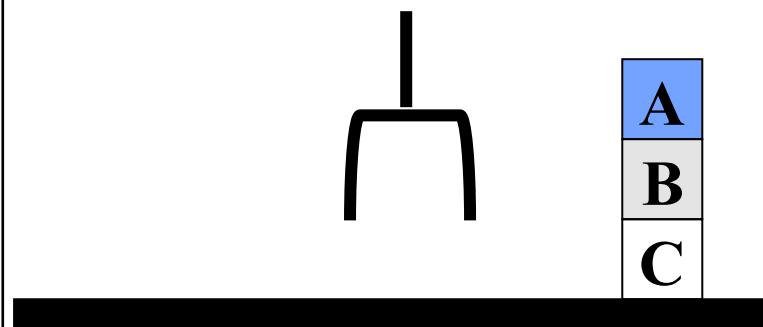
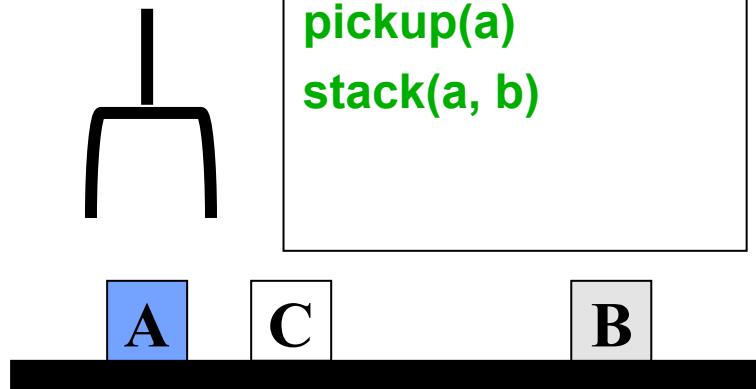
État initial:

on(a, b)
clear(a)
handempty
on(b, c)
ontable(c)

Pile de buts:

Pile vide!
Fin planif

Plan:
pickup(a)
stack(a,b)
unstack(a, b)
putdown(a)
pickup(b)
stack(b, c)
pickup(a)
stack(a, b)



Poursuite planif!

operator(stack(X,Y),
Precond [holding(X),clear(Y)],
Ajout [handempty, on(X,Y), clear(X)],
Destr [holding(X),clear(Y)],
Constr [X==Y, Y==table, X==table]).

operator(pickup(X),
Precond [ontable(X), clear(X), handempty],
Ajout [holding(X)],
Destr [ontable(X), clear(X), handempty],
Constr [X==table]).

operator(unstack(X,Y),
Precond [on(X,Y), clear(X), handempty],
Ajout [holding(X), clear(Y)],
Destr [handempty, clear(X), on(X,Y)],
Constr [X==Y, Y==table, X==table]).

operator(putdown(X),
Precond [holding(X)],
Ajout [ontable(X), handempty, clear(X)],
Destr [holding(X)],
Constr [X==table]).

Programmation de STRIPS en PROLOG

```
test(Plan) :- etat_initial(I),  
            etat_final(F),  
            resoudre(I, [F], Plan).  
  
resoudre(_, [], []).  
resoudre(Etat, [L|Buts], Plan) :- is list(L),  
    subset(L, Etat), !, resoudre(Etat, Buts, Plan).  
resoudre(Etat, [L|Buts], Plan) :- is list(L), !,  
    subtract(L, Etat, LL), append(LL, TL|Buts), NButs,  
    resoudre(Etat, NButs, Plan).  
resoudre(Etat, [operateur(Op)|Buts], [Op|Plan]) :- !,  
    appliquer operateur(Etat, Op, NEtat),  
    resoudre(NEtat, Buts, Plan).  
resoudre(Etat, [But|Buts], Plan) :- member(But, Etat), !,  
    resoudre(Etat, Buts, Plan).  
resoudre(Etat, [But|Buts], Plan) :-  
    trouver_operateur(Etat, But, Op, P),  
    intersection(P, Buts, []),  
    operateur(Op, Preconds, _, _),  
    resoudre(Etat, [Preconds, opérateur(Op), But|Buts], Plan).
```



Programmation STRIPS - suite

```
trouver_operateur(Etat, But, Op, []) :-  
    clause_operateur(Op, Preconds, _, AListe), Q,  
    member(But, AListe), soustraction(Preconds, Etat, []),  
    call(Q), term_variables(Op, []).
```

```
trouver_operateur(Etat, But, Op, [T|S]) :-  
    clause_operateur(Op, Preconds, _, AListe), Q,  
    member(But, AListe),  
    soustraction(Preconds, Etat, [T|S]), call(Q),  
    term_variables(Op, []).
```

```
appliquer_operateur(Etat, Op, NEtat) :-  
    operateur(Op, _, Dliste, AListe),  
    subtract(Etat, Dliste, DEtat),  
    union(DEtat, AListe, NEtat).
```

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

```
soustraction([X|L], D, LL) :- member(X, D),  
    soustraction(L, D, LL).
```

```
soustraction([X|L], D, [X|LL]) :- soustraction(L, D,  
    LL).
```





Monde des blocs - opérateurs

```

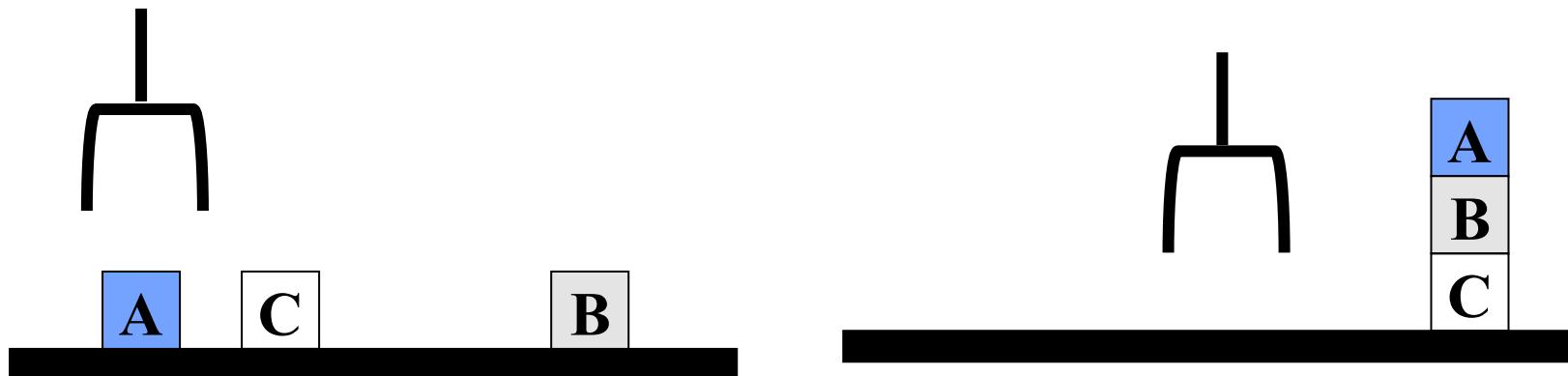
operateur(deposer(B),
           [poignet_prise(B)], [poignet_prise(B)],
           [poignet_vide, sur_table(B), libre(B)]).

operateur(depiler(B, C),
           [poignet_vide, bloc(B), libre(B), sur(B, C)],
           [poignet_vide, libre(B), sur(B, C)],
           [poignet_prise(B), libre(C)]) :- \+ B == C.

operateur(prendre(B),
           [poignet_vide, bloc(B), libre(B), sur_table(B)],
           [poignet_vide, libre(B), sur_table(B)],
           [poignet_prise(B)]).

operateur(empiler(B,C),
           [poignet_prise(B), bloc(B), bloc(C), libre(C)],
           [poignet_prise(B), libre(C)],
           [poignet_vide, sur(B, C), libre(B)]) :- \+ B == C.

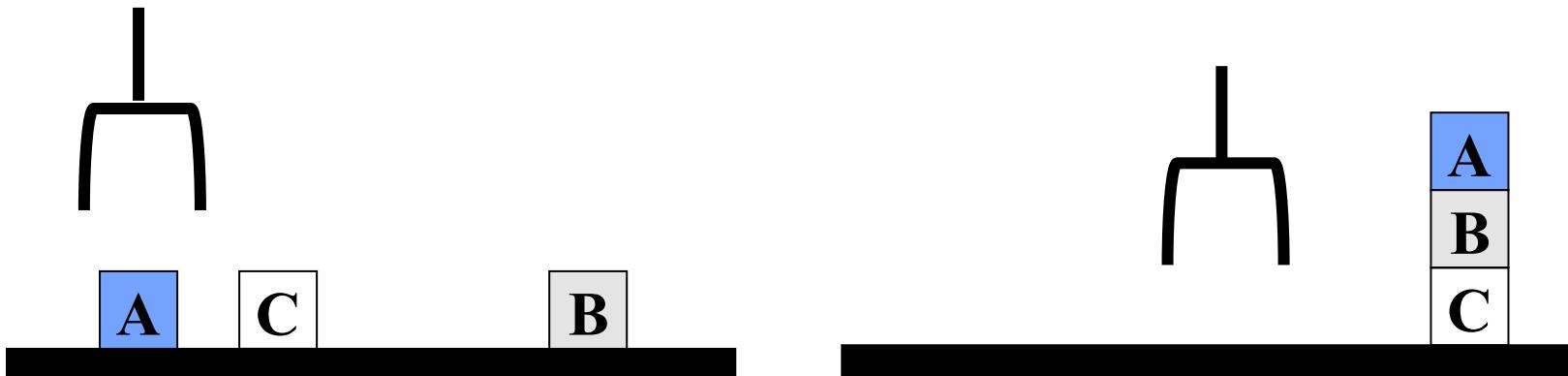
```



Programmation CLIPS - lancement

```
etat_initial([sur_table(a), sur_table(b),  
             sur_table(c), libre(a), libre(b),  
             libre(c), poignet_vide, bloc(a),  
             bloc(b), bloc(c)]).
```

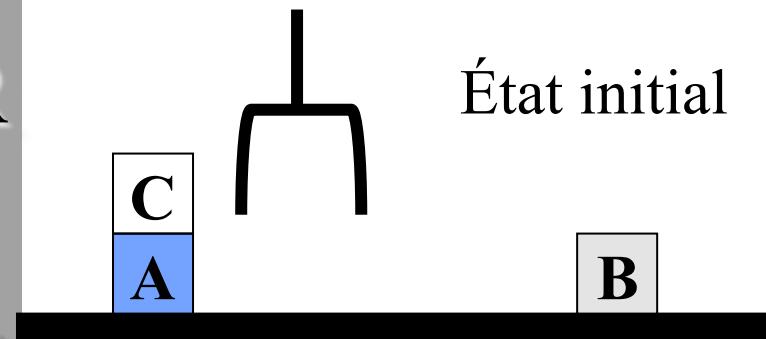
```
etat_final([sur_table(a), sur(b, a),  
            sur(c, b), libre(c), poignet_vide]).
```



```

Achieve on(a,b) via stack(a,b) with preconds: [holding(a),clear(b)]
|Achieve holding(a) via pickup(a) with preconds: [ontable(a),clear(a),handempty]
||Achieve clear(a) via unstack(_1584,a) with preconds:
[on(_1584,a),clear(_1584),handempty]
||Applying unstack(c,a)
||Achieve handempty via putdown(_2691) with preconds: [holding(_2691)]
||Applying putdown(c)
|Applying pickup(a)
Applying stack(a,b)
Achieve on(b,c) via stack(b,c) with preconds: [holding(b),clear(c)]
|Achieve holding(b) via pickup(b) with preconds: [ontable(b),clear(b),handempty]
||Achieve clear(b) via unstack(_5625,b) with preconds:
[on(_5625,b),clear(_5625),handempty]
||Applying unstack(a,b)
||Achieve handempty via putdown(_6648) with preconds: [holding(_6648)]
||Applying putdown(a)
|Applying pickup(b)
Applying stack(b,c)
Achieve on(a,b) via stack(a,b) with preconds: [holding(a),clear(b)]
|Achieve holding(a) via pickup(a) with preconds: [ontable(a),clear(a),handempty]
|Applying pickup(a)
Applying stack(a,b)

```



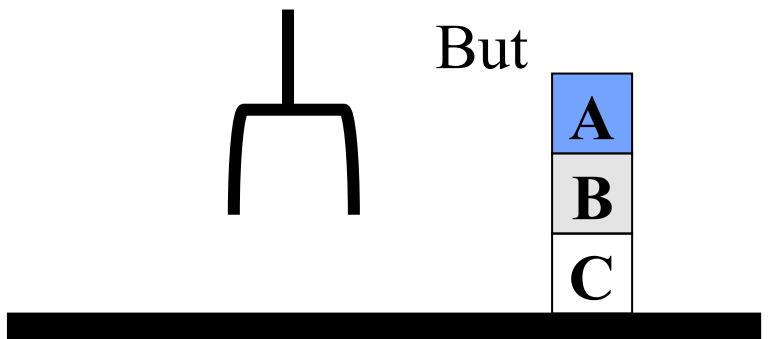
État initial

Anomalie de Sussman

```

From
[clear(b),clear(c),ontable(a),ontable(b),on(c,a),handempty]
To [on(a,b),on(b,c),ontable(c)]
Do:
  unstack(c,a)
  putdown(c)
  pickup(a)
  stack(a,b)
  unstack(a,b)
  putdown(a)
  pickup(b)
  stack(b,c)
  pickup(a)
  stack(a,b)

```



But

Anomalie de Sussman - lancement

```
etat_initial([sur_table(a), sur_table(b),  
    sur(c, a), libre(b), libre(c),  
    poignet_vide, bloc(a), bloc(b),  
    bloc(c)]).
```

```
etat_final([sur_table(a), sur(b, a),  
    sur(c, b), libre(c), poignet_vide]).
```





Un problème que STRIPS ne peut résoudre

“Échanger le contenu de deux registres mémoires $r1$ et $r2$ dans les contenus initiaux sont respectivement a et b . On suppose un troisième registre $r3$ est disponible.”

Etat initial: $\text{contents}(r1,a)$, $\text{contents}(r2,b)$, $\text{contents}(r3,0)$

But: $\text{contents}(r1,b)$, $\text{contents}(r2,a)$

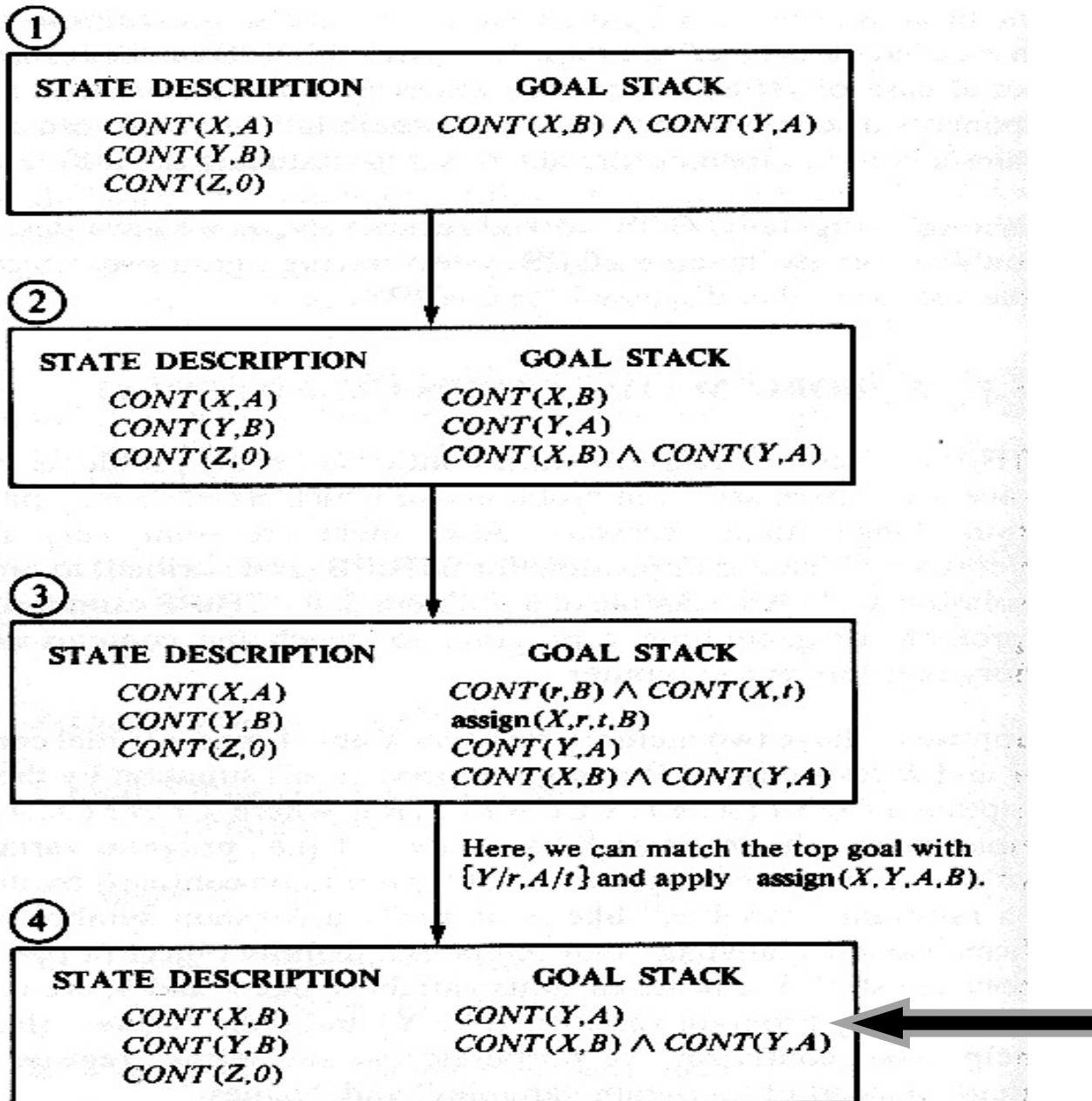
Opérateurs:

$\text{assign}(X,A,Y,B)$

P : $\text{contents}(X,A)$, $\text{contents}(Y,B)$

D : $\text{contents}(Y,B)$

A : $\text{contents}(Y,A)$



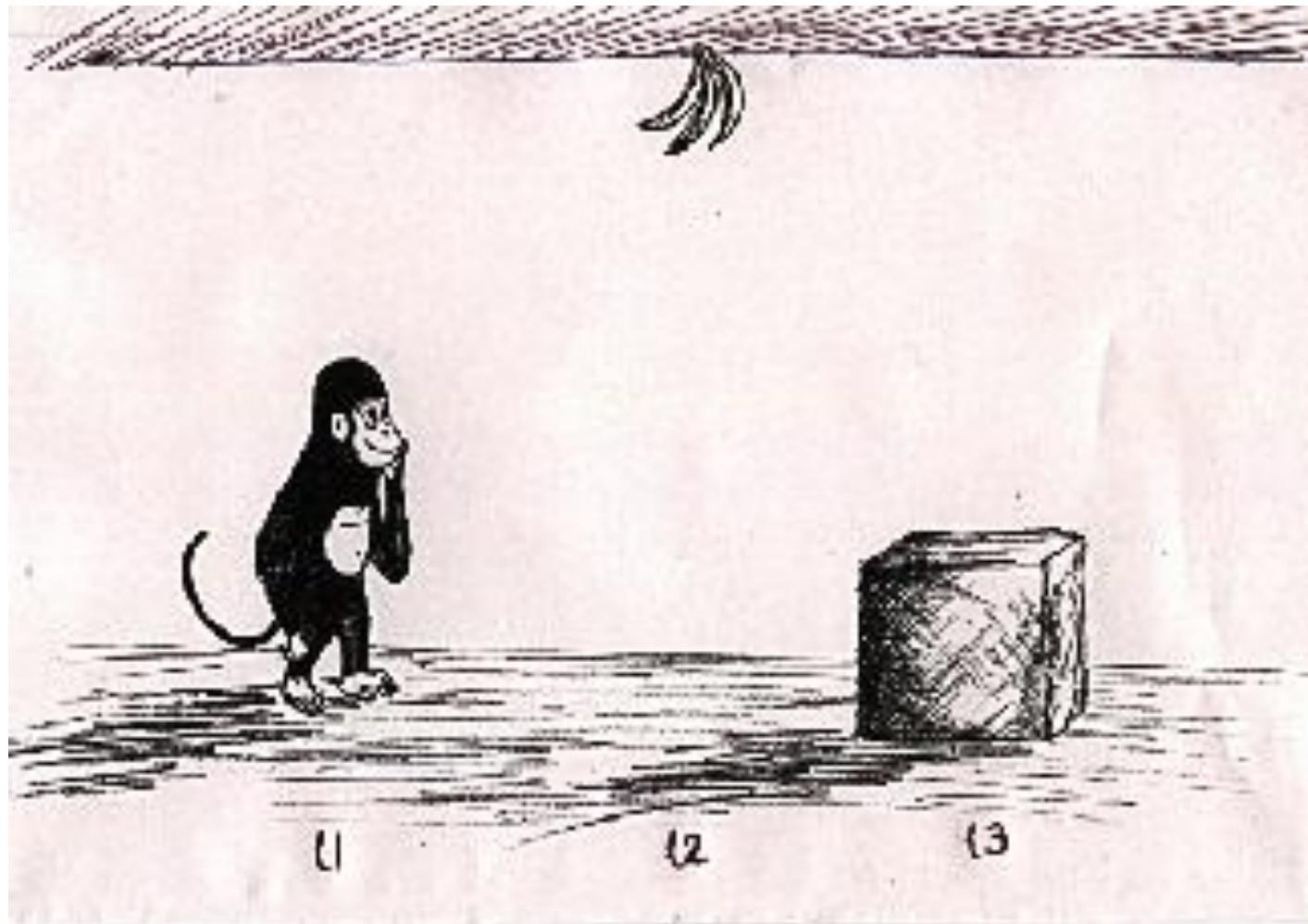
Echec!

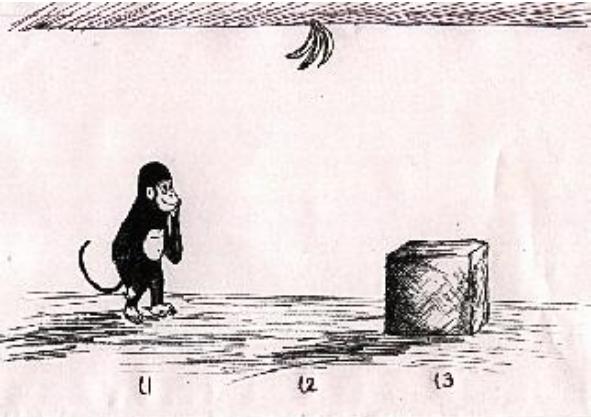


L
I
P
6

C
N
R
S

« Le singe et les bananes »





« Le singe et les bananes » états et opérateurs

```
etat_initial([position(caisse, u1), position(bananes, u2),  
position(singe,u3)]).  
  
etat_final([possede(singe, bananes)]).  
  
opereur(deplacer(P, Q), [position(singe, P)],  
[position(singe, P)], [position(singe, Q)]) :- \+ P == Q.  
  
opereur(pousser(U, V), [position(singe, U),  
position(caisse, U)], [position(singe, U), position(caisse,  
U)], [position(singe, V), position(caisse, V)]) :- \+ U == V.  
  
opereur(cueillir_bananes(U), [position(singe, U),  
position(caisse, U), position(bananes,U)], [],  
[possede(singe, bananes)]).
```

