Planification Automatique

Représentation du monde: Calcul de Situations

Opérateurs

Planification linéaire: STRIPS

Interaction entre buts et planification non-linéaire

Pourquoi planifier?

Optimization des ressources:

- mission spatiale: chaque seconde coûte
- construction: éviter des temps morts

Interdépendences complexes entre tâches:

éviter des erreurs!

Trop de choix:

résoudre des casse-têtes

Agents autonomes:

prise de décisions

Planification automatique

Etant donné:

- un ensemble d’opérateurs (actions)
- un état initial
- des conditions qui doivent être satisfaites dans l’état final

trouver une séquence d’opérateurs qui vont assurer que l’état intial est transformé en un état final qui remplit les conditions.

La planification est un problème très difficile, car il n’y a pas de limite supérieure à la complexité du plan nécessaire pour réaliser un but donné.

Eléments d’un système de planification

Modélisation symbolique

- de l’état du monde: *calcul de situations*
- des actions: *opérateurs*
- des buts

Algorithme de recherche pour trouver une séquence d’opérateurs.

Méthodes pour traiter l’interférence entre différents buts.

Le monde des blocs

Le monde des blocs est un exemple standard de la litérature sur les systèmes de planification. Il comporte:

- un ensemble de blocs, tous de même taille.
- la main d'un robot, capable de tenir et de déplacer un bloc à la fois.
- une table, qui a suffisement de place pour placer tous les blocs qui existent.

Le monde des blocs est décrit par les prédicats:

- `ON(x,y)` : bloc `x` se trouve au-dessus du bloc `y`.
- `ONTABLE(x)` : bloc `x` se trouve sur la table.
- `CLEAR(x)` : il n'y a rien sur bloc `x`.
- `HOLDING(x)` : la main tient bloc `x`.
- `HANDEMPTY`: la main est vide.

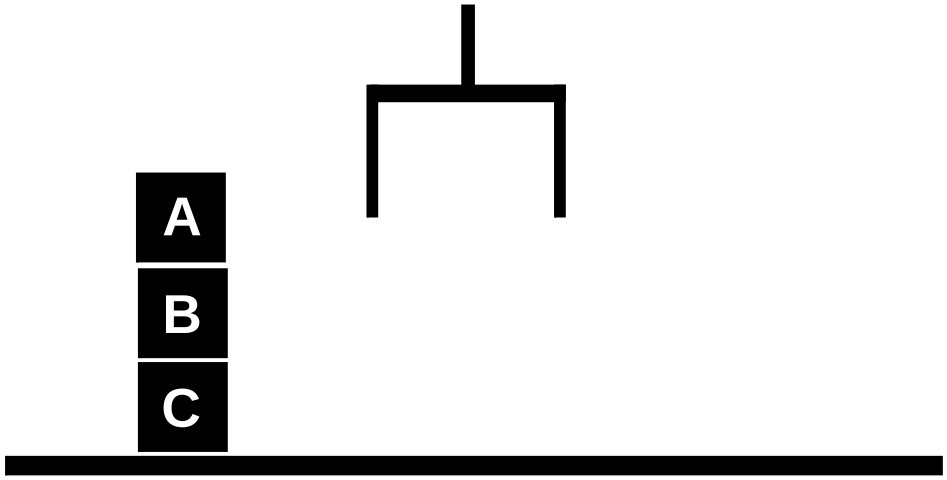
Modélisation du monde

Pendant l'éxécution d'un plan, le monde évolue d'un *état* à un autre. Il faut distinguer les états car la description de deux états est le plus souvent contradictoire.

Chaque état peut être modélisé par une conjonction d'expressions en calcul de prédicats:

```
ON(A,B)
ON(B,C)
CLEAR(A)
HANDEMPTY
ONTABLE(C)
```

pourrait décrire la situation suivante du monde des blocs:


A
B
C

Situations et Etats

La modélisation d’un état par une conjonction de prédicats est *partielle*: la relation `ONTABLE(C)` n’est pas incluse.

On appelle une telle description partielle d’un état une *situation*.

Les situations permettent la représentation du but, où normalement pas toutes les proprietés de l’état sont specifiées.

Les opérateurs transforment une situation en une autre. Le calcul qui décrit cette suite de transformations s’appelle *calcul de situations*.

Complexité de la planification

En général, un opérateur effectuera une transformation quelconque de S_i en S_{i+1}: $S_{i+1} = f(S_i)$

Chapman:

> si l'ensemble de situations est illimitée, la planification n'est pas décidable

$\Rightarrow$ les opérateurs doivent opérer sur un ensemble fini

Exemple: limiter f à la suppression/ajout de propositions

Opérateurs STRIPS

Un opérateur permet une transformation $S_i \Rightarrow S_{i+1}$ et se caractérise par:

- ses *préconditions*: des propositions qui doivent être vraies dans S_i pour que l'opérateur puisse s'appliquer.
- ses *postconditions*: les propositions qui sont réalisées dans S_{i+1} par l'application de l'opérateur, et donc ajoutées.
- ses *suppressions*: les propositions qui ne seront plus vraies dans S_{i+1}, et donc supprimées.

On suppose que tous les propositions non-mentionnées dans les suppressions restent valables (axiome de cadre).

Les propositions doivent faire partie d'un ensemble fini de possibilités.

Schema d’opérateurs

Les opérateurs sont souvent très similaires

$\Rightarrow$ exprimer par un *schema* avec variables

Instantiation d’un schema:

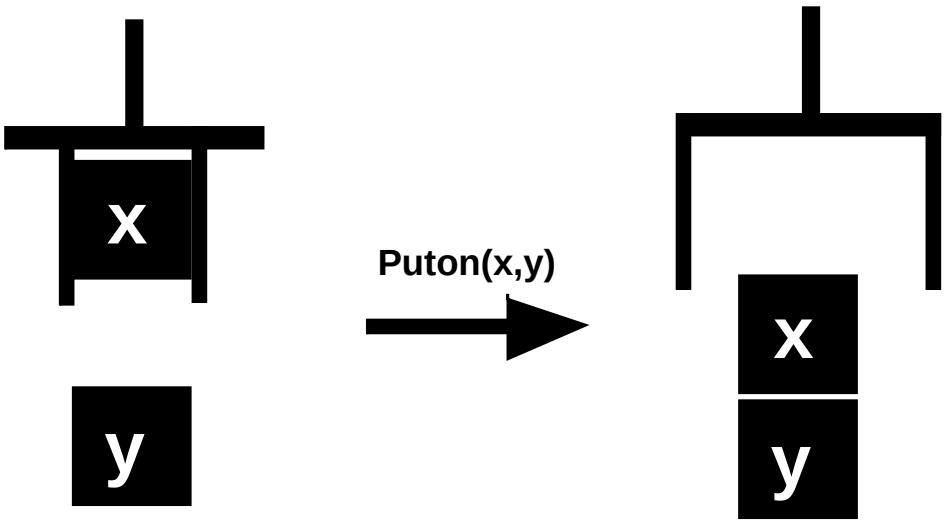
substituer les variables par individus;

chaque substitution donne un autre opérateur.

Ensemble limité d’individus!

Exemple: l'opérateur PUTON

PUTON(x,y) modélise l'action de mettre un bloc x au-dessus un autre bloc y:



Il est caractérisé comme suit:

- préconditions (P): CLEAR(y), HOLDING(x)
- postconditions (A): ON(x,y), CLEAR(x), HANDEMPTY
- suppressions (D): CLEAR(y), HOLDING(x)

Opérateurs du monde des blocs

Le monde des blocs comporte quatre opérateurs:

- `PICKUP(x)`:

  P= `ONTABLE(x)`, `CLEAR(x)`, `HANDEMPTY`

  D = P, A = `HOLDING(x)`

- `PUTDOWN(x)`:

  P= `HOLDING(x)`, D = P,

  A= `ONTABLE(x)`, `CLEAR(x)`, `HANDEMPTY`

- `PUTON(x,y)`:

  P = `HOLDING(x)`, `CLEAR(y)`, D = P,

  A = `HANDEMPTY`, `ON(x,y)`, `CLEAR(x)`

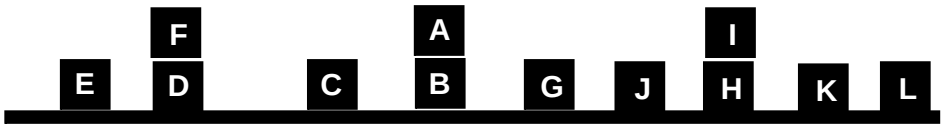
- `UNSTACK(x,y)`:

  P = `HANDEMPTY`, `ON(x,y)`, `CLEAR(x)`,D = P,

  A = `HOLDING(x)`, `CLEAR(y)`

Chaînage avant/arrière?

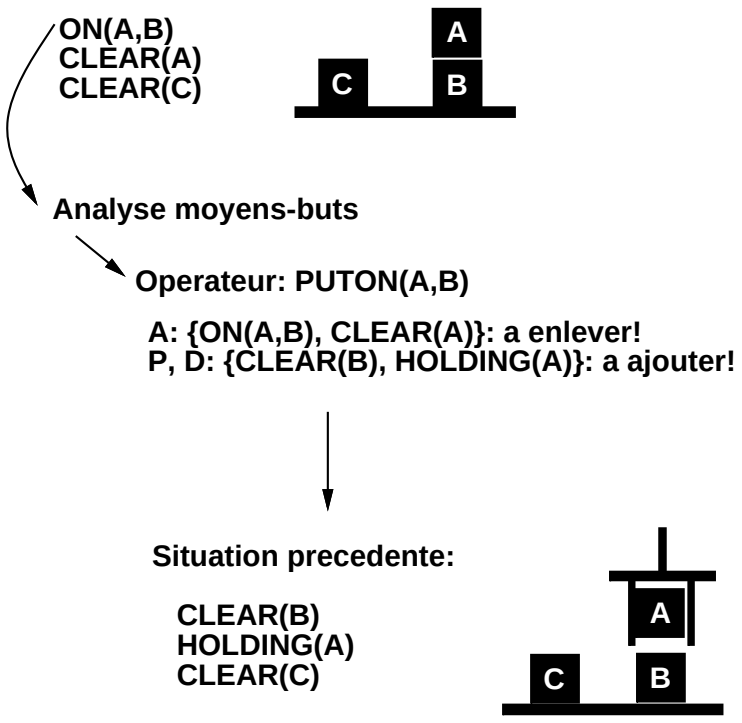
Chaînage avant: beaucoup d’opérateurs!



$\Rightarrow$ chaînage arrière mieux adapté!

Suivant le mécanisme de l'analyse moyens-buts établi par GPS, la planification peut se faire en chaînage arrière à partir du but.

Les opérateurs qui entrent en ligne de compte sont ceux qui ont le potentiel de réaliser un but ou sous-but. Leurs préconditions définissent les sous-buts du prochain niveau:



$\Rightarrow$ arbre de recherche.

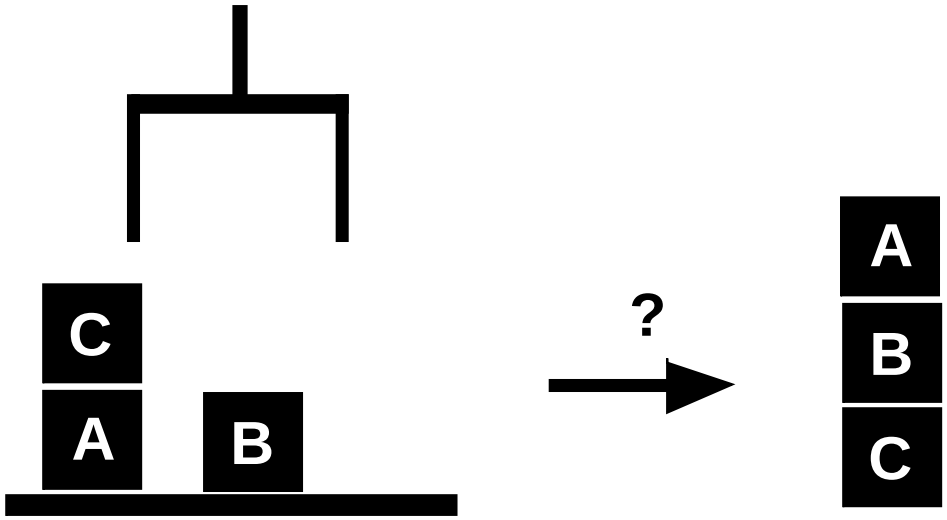
Exemple

Considérez le but de construire un tour de 3 blocs:

`ON(A,B), ON(B,C)`

à partir de la situation initiale:

`ON(C,A), CLEAR(B), CLEAR(C),`
`ONTABLE(A),`
`ONTABLE(B), HANDEMPTY`



Planification avec A*

L'espace défini par le chaînage des opérateurs est traité avec un algorithme de recherche heuristique comme A*:

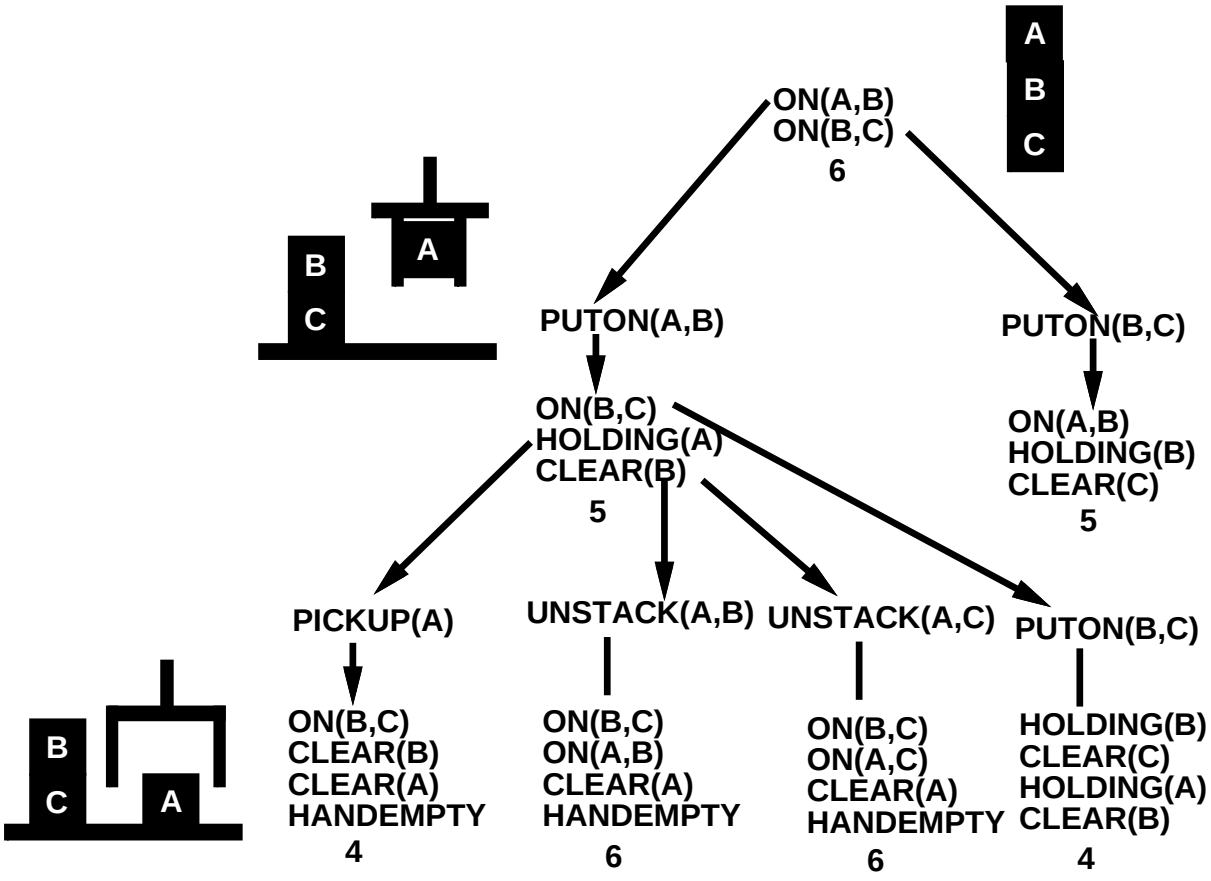
- noeuds de recherche = situations
- fonction `succ(n)` = chaînage des sous-buts
- coût d'un chemin = somme des coûts d'application des opérateurs
- heuristique = 2*d(ON) + d(HOLDING)

$\Rightarrow$ assure que le résultat est le plan qui exige le moins d'efforts.

La recherche est terminée quand on arrive à une situation ou tous les propositions sont satisfaits par la situation initiale.

Espace de recherche

Recherche en chaînage arrière:



L’ordre temporel des actions est implicite dans la génération de l’arbre.

Le problème de la consistance

Un etat peut être inconsistant:

`HOLDING(B)`

`CLEAR(C)`

`HOLDING(A)`

`CLEAR(B)`

mais se transformer en un etat consistant:

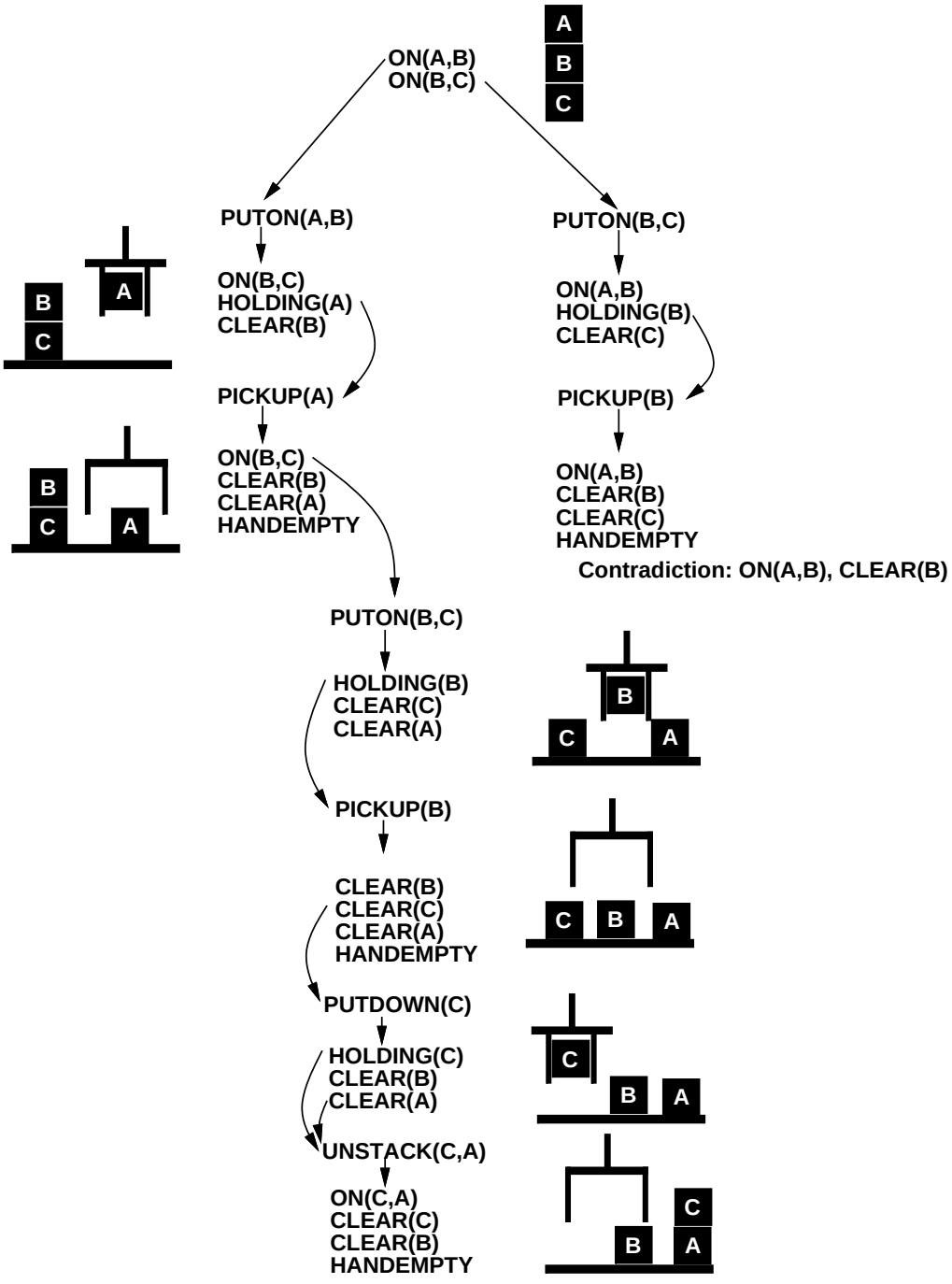
`PICKUP(B),PICKUP(A)` $\Rightarrow$

`CLEAR(B),CLEAR(A),HANDEMPTY,CLEAR(C)`

$\Rightarrow$ il faut les detecter!

Solution par A*

Par chaînage arrière, on trouve l'arbre de recherche suivant:



Problèmes de Chaînage arrière

Parfois on ne connaît pas toutes les variables pour complètement instantier un opérateur en chaînage arrière:

But: `CLEAR(A)`

Opérateur: `UNSTACK(x,y)` $\Rightarrow$

`CLEAR(y)`, ...

$\Rightarrow$ `y=A, x=?`

Il faut soit maintenir les variables (mécanisme compliqué à cause des tests de circularité) ou faire une recherche entre les instantiations de l'opérateur pour chaque objet existant dans le monde.

Problèmes de Chaînage arrière (2)

Considérez l'opérateur `PUTON(A,B)`, qui a la précondition `HOLDING(A)`. Si on applique cet opérateur à la situation:

`ON(C,A), ON(A,B)`

on obtient le sous-but:

`ON(C,A), HOLDING(A),`
`CLEAR(B)`

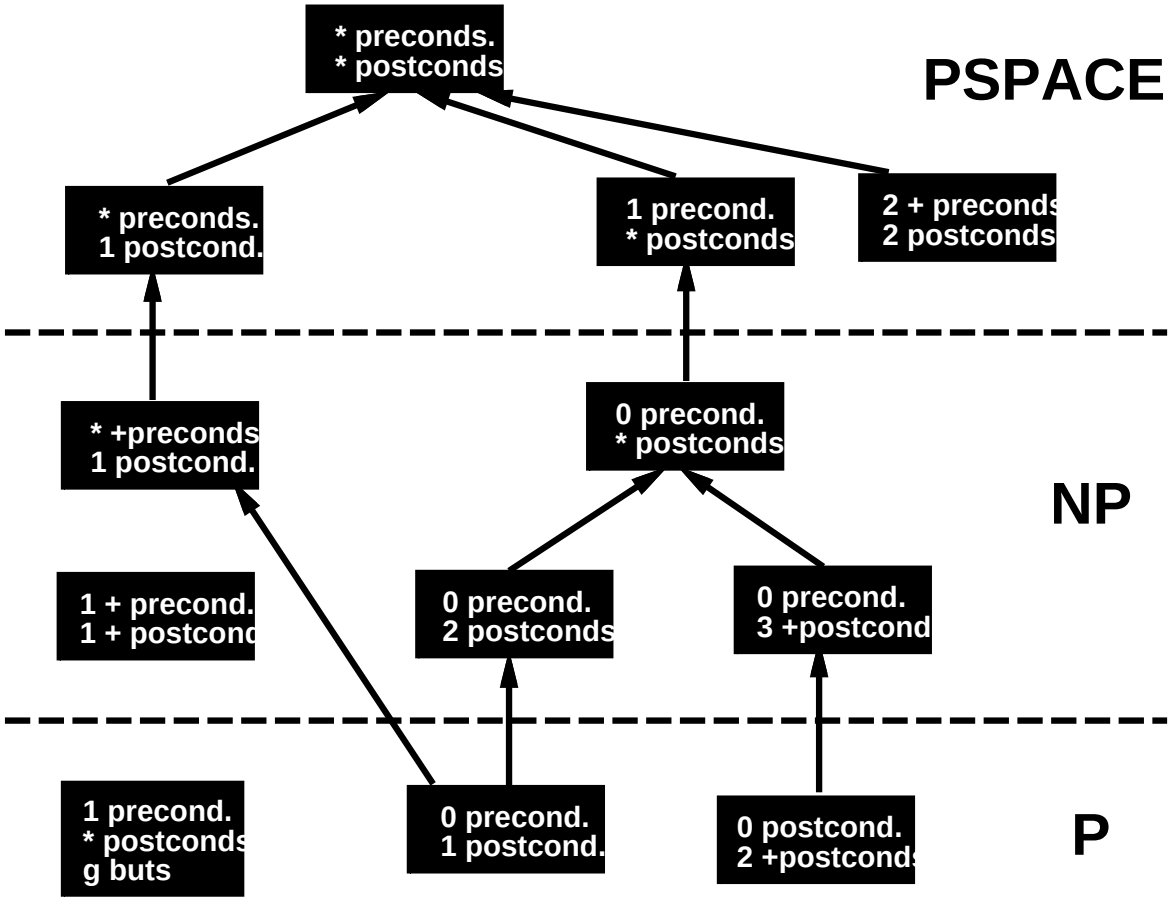
auquel on peut appliquer les opérateurs `PICKUP(A)`, `PUTON(C,A)` et `PICKUP(C)` pour arriver à la situation initiale:

`CLEAR(A), CLEAR(B), CLEAR(C)`

Cependant, le plan pour arriver au but passe par des situations inconsistentes, et n'est donc pas valable!

$\Rightarrow$ il faut détecter et éliminer les états inconsistents au plus tôt!

Complexité de la planification



La planification comme déduction logique

La planification peut être comprise comme une déduction logique en ajoutant à chaque prédicat un symbole qui décrit la situation:

$$\texttt{ON(A,B,}S_1\texttt{)}$$

en définissant une fonction d'évolution des états:

$$S_1 = \texttt{apply(PUTON(A,B),}\ S_0\texttt{)}$$

et en formulant des règles d'inférence:

$$\texttt{(CLEAR(x,s)} \wedge \texttt{HOLDING(y,s))}$$
$$\Rightarrow$$
$$\texttt{(ON(y,x,apply(PUTON(y,x),s))}$$
$$\wedge\ \texttt{....}$$

$\Rightarrow$ la solution est donnée par la séquence de fonctions imbriquées de la dernière situation.

Le problème des cadres (”Frame Problem”)

La formulation logique pose cependant un problème: pour tous les prédicats qui ne sont pas affectés par une action, il faut explicitement faire l’inférence qu’ils sont vrais dans la situation suivante.

Pour chaque opérateur et pour chaque proprieté qui n’est pas affectée, il faut donc écrire une règle d’inférence $\Rightarrow$ nombre énorme de règles: frame problem.

Solution 1: règles de 2ème ordre:

$$(\forall\ \mathrm{P})\ (\mathrm{P} \neq \mathrm{CLEAR}) \wedge (\mathrm{P} \neq \ldots)$$
$$\Rightarrow$$
$$(\mathrm{P(s)} \Rightarrow$$
$$\mathrm{P(apply(PUTON(x,y),s))})$$

mais calcul de prédicats de 2ème ordre pose des problèmes algorithmiques.

Solution 2: formuler les propositions comme individus:

$$\mathrm{HOLDS(ON(A,B),S_0)}$$

$\Rightarrow$ les propositions deviennent des individus!

Planification logique (exemple)

Situation initiale:

$\texttt{CLEAR(A,}S_0\texttt{)}$, $\texttt{CLEAR(B,}S_0\texttt{)}$,

$\texttt{ON(B,C,}S_0\texttt{)}$,

$\texttt{HANDEMPTY(}S_0\texttt{)}$

But: `ON(A,B,x)`

La planification donne comme résultat:

```
x = apply(PUTON(A,B),
          apply(PICKUP(A),S0))
```

Resumé

Représentation du monde, buts et opérateurs:

Calcul de Situations

Planification linéaire: STRIPS

Planification par preuve de théorèmes

Planification non-linéaire

Interaction entre buts

Planification non-linéaire

Planification réactive

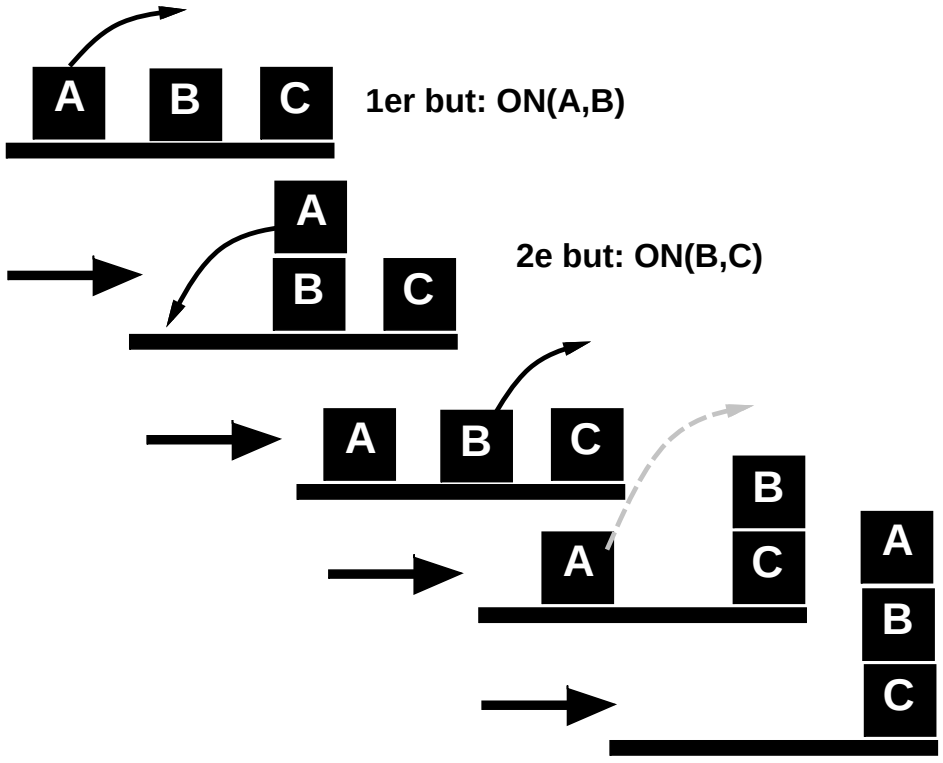
Buts multiples

Quand il y a plusieurs buts, il faut les attaquer dans le bon ordre. Par exemple, si on a les buts:

$$ON(A,B) \wedge ON(B,C)$$

on ne peut les achever que dans la séquence `ON(B,C)`, `ON(A,B)`.

Un système de planification qui n'effectue pas ce tri peut conduire à l'anomalie de Sussman:



Solution 1: Protections

Idée: éviter que le programme donne un plan au cours duquel on achève un but qui est défait par la suite.

Une fois achevé, chaque but est protegée pour ne plus être défait. Par example, on ne permettrait plus jamais de défaire `ON(A,B)`.

$\Rightarrow$ backtrack si la solution comporte un but qui est défait.

Cependant, ceci peut être très inefficace si le nombre de buts est grand.

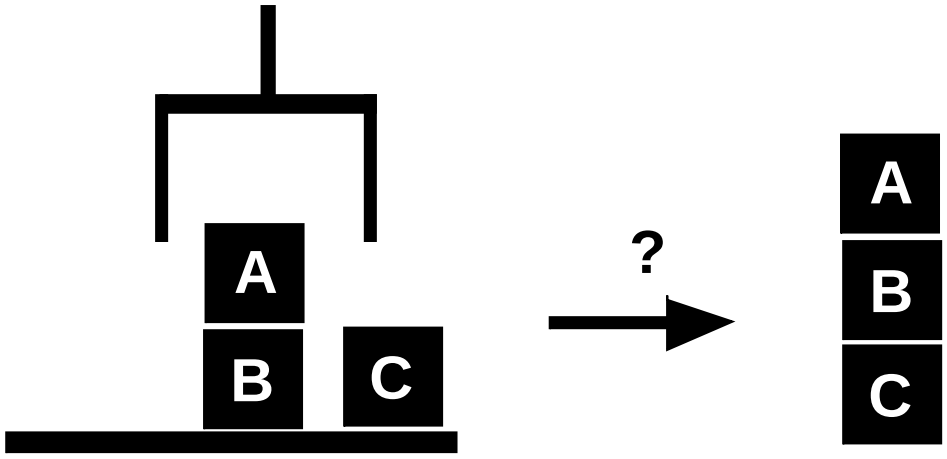
Un contre-exemple pour les protections

Construire la tour:

`ON(A,B)`, `ON(B,C)`

à partir de la situation initiale:

`ON(A,B)`, `ONTABLE(C)`, `ONTABLE(B)`, `HANDEMPTY`



$\Rightarrow$ le plan implique de défaire `ON(A,B)`!

Solution 2: Planification non-linéaire

Le vrai problème posé par l'interférence entre buts ne se trouve pas au niveau des actions elles-mêmes, mais de leur *ordre*.

Solution: trouver d'abord les sous-plans pour chaque but en soi, et ensuite trouver l'ordre qui convient: planification non-linéaire.

Dans notre exemple:

a) `ON(A,B)` $\Rightarrow$ `PUTON(A,B)`

b) `ON(B,C)` $\Rightarrow$ `PUTON(B,C)`

sont d'abord générées comme sous-plans indépendants pour réaliser les deux buts.

Plans alternatifs pour réaliser ON(A,B) :

1) plan vide, précondition = ON(A,B)

2) (PUTON(A,B)), préconditions =
CLEAR(B),HOLDING(A))

....

i)
(UNSTACK(A,?x),PUTDOWN(A),PICKUP(A),PUTON(A,
préconditions = ON(A,?x),CLEAR(B),HANDEMPTY

....

Plans alternatifs pour achever ON(B,C) :

....

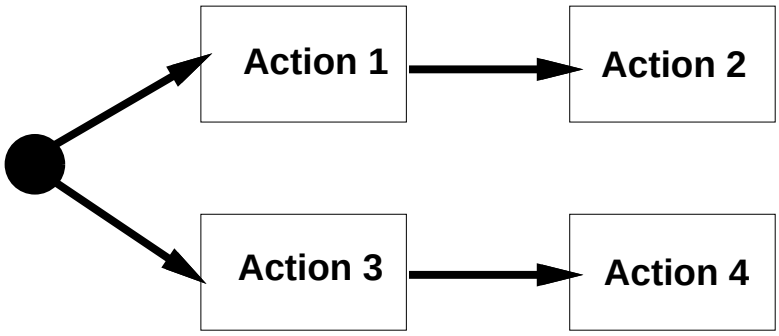
k) (PICKUP(B),PUTON(B,C)),
préconditions = CLEAR(B),CLEAR(C),HANDEMPTY

....

Seul plan i) peut se combiner avec plan k)!

Représentation de plans non-linéaires

Dans la planification non-linéaire, les plans représentent des *ordres partiels*:



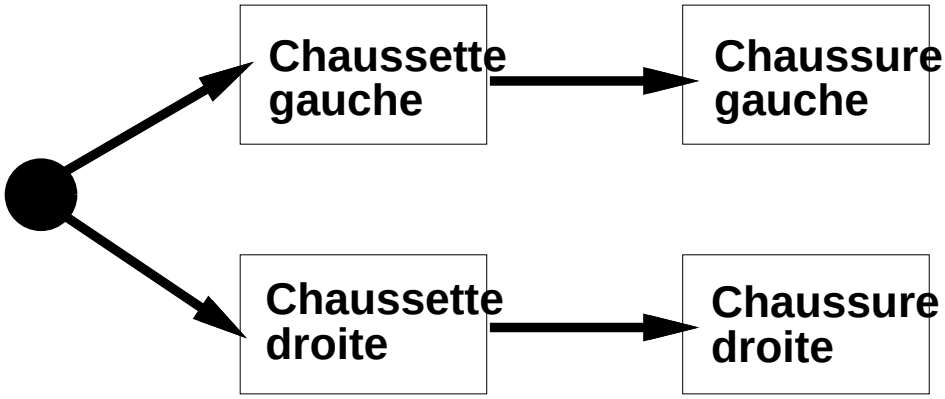
Si certaines actions peuvent être executées dans n’importe quel ordre, la planification non-linéaire permet une représentation plus compacte.

Idée générale: *least committment*, c’est-à-dire ne pas faire des choix avant que ce soit vraiment nécessaire.

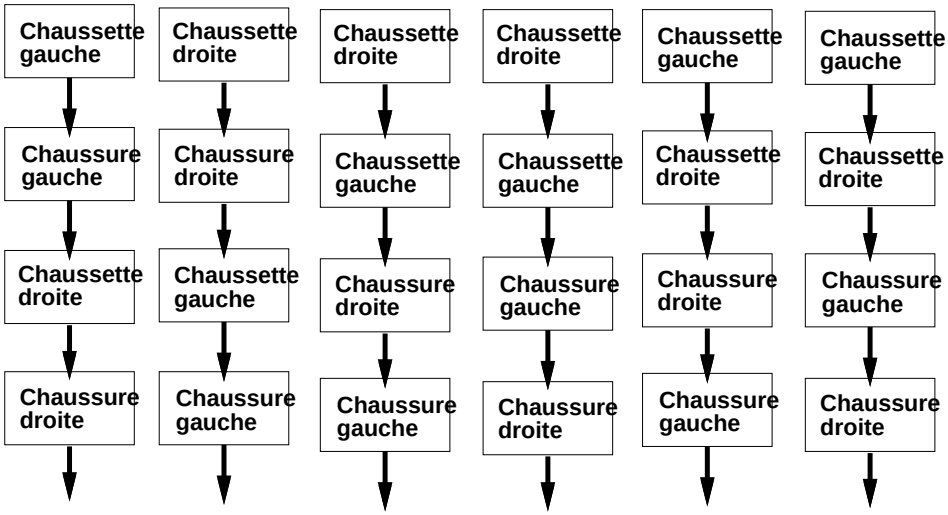
Exemple:

Mettre les chaussettes et les chaussures:

un seul plan non-linéaire:

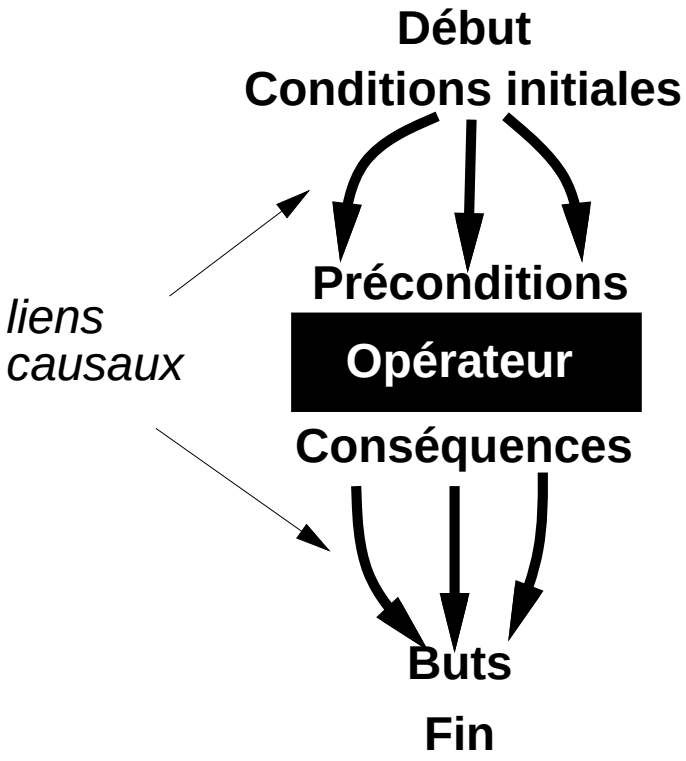


représente 6 plans linéaires:



Notation graphique pour la planification non-linéaire

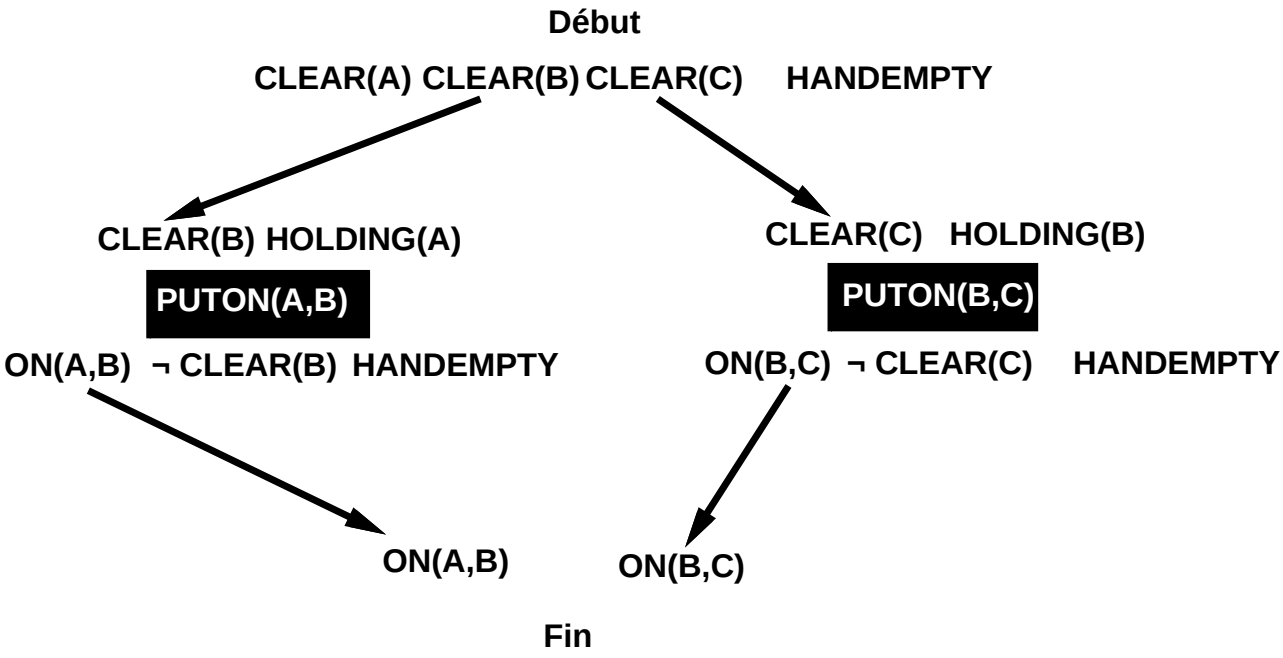
Pour chaque opérateur, on note les préconditions en dessus et les conséquences en dessous:



Les liens causaux mettent en évidence la manière dont le plan satisfait les buts à partir des conditions initiales.

Espace de plans partiels

On représente l'ensemble de tous les plans partiels comme un espace:

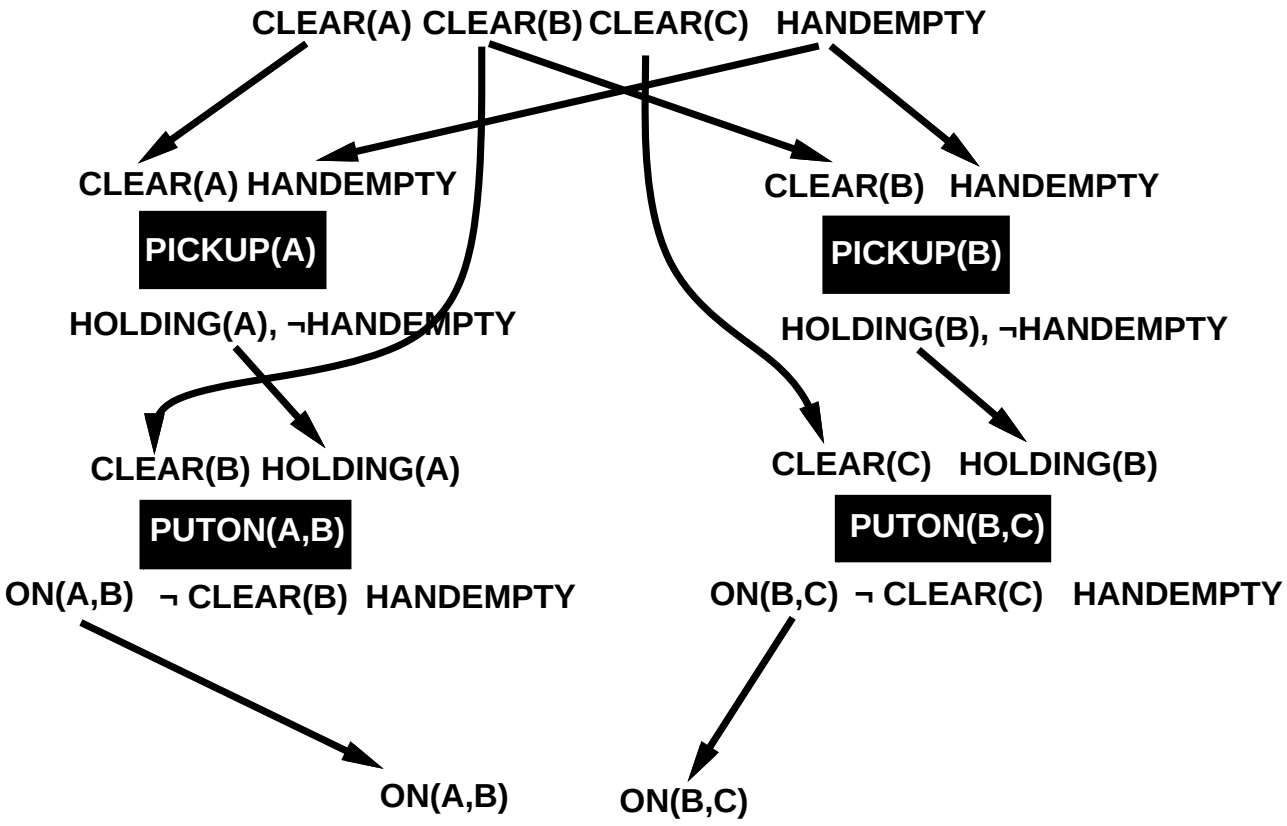


Liens causaux entre buts et actions, actions et état initial.

Compléter les plans partiels

Les préconditions qui ne sont pas satisfaites par un lien causal exigeront des opérations supplémentaires.

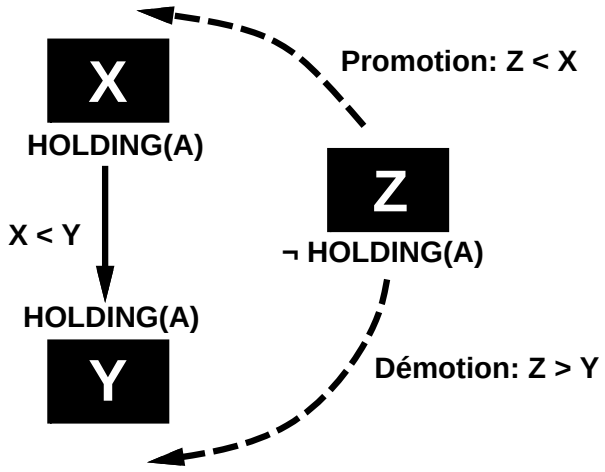
Par recherche récursive, on construit un ensemble d'opérateurs tel que tous les buts et sous-buts sont satisfaits par un lien causal:



Contraintes d'ordre

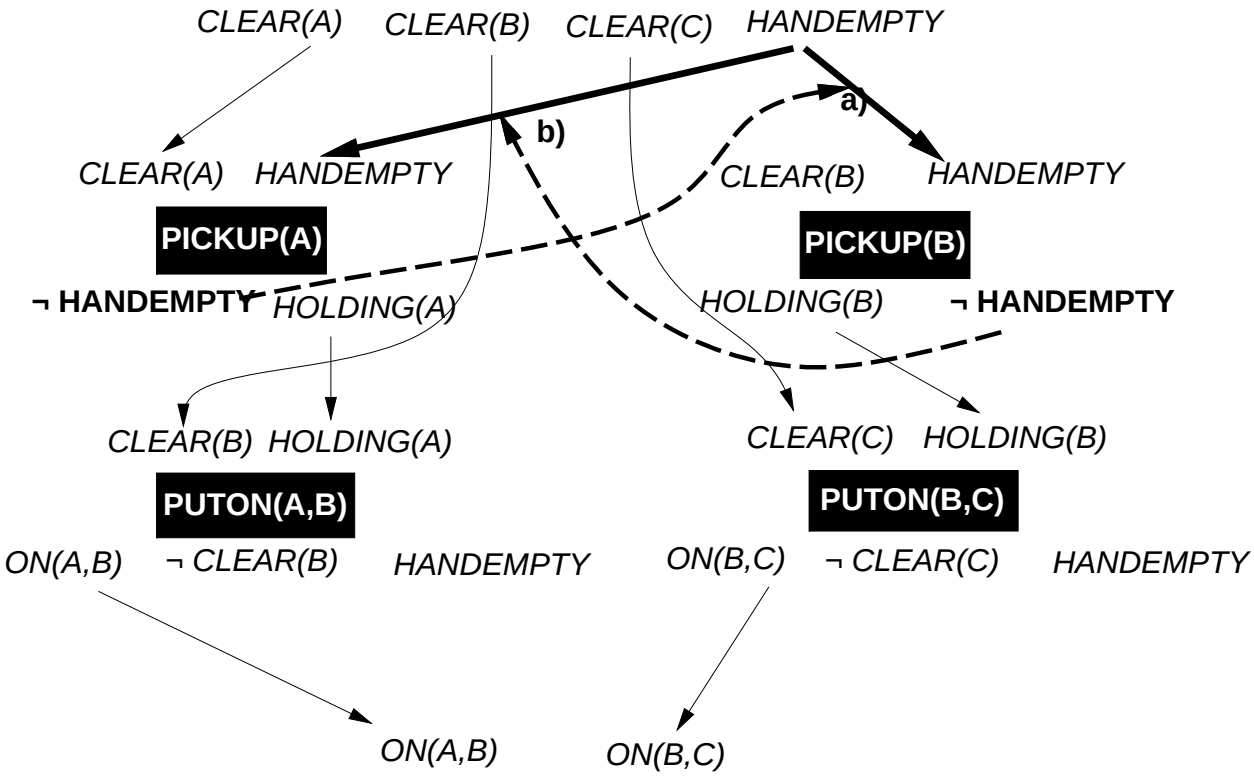
Chaque lien causal entre une précondition P d'un opérateur Y et une postcondition P d'un opérateur X impose une contrainte de *promotion* X $<$ Y (X avant Y).

Si une action Z réalise la condition $\neg P$, elle *menace* tout lien causal $Y \rightarrow P$ menant à une condition P. $\Rightarrow$ on établit une contrainte de *demotion* Z $>$ Y, ou
on établit une contrainte de *promotion* Z $<$ X



S'il n'y a pas d'ordre qui respecte toutes les contraintes, backtrack pour choisir d'autres opérateurs ou liens causaux.

Exemple:



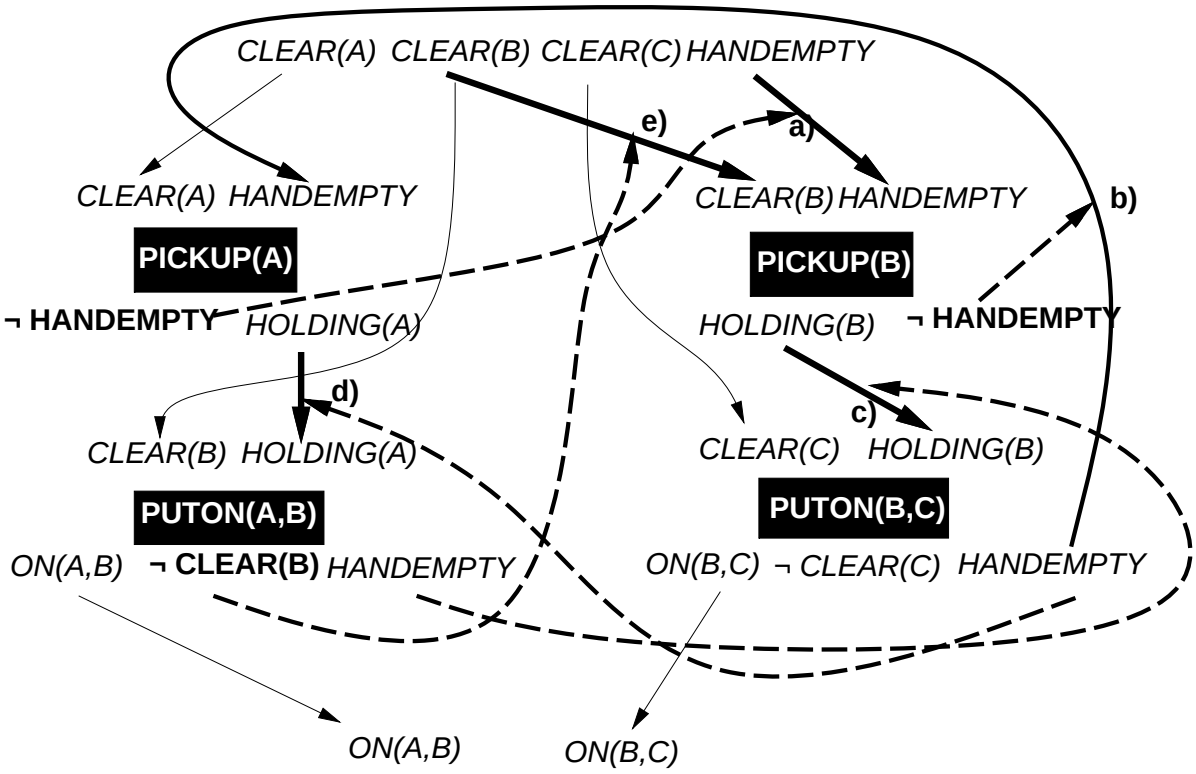
Menaces:

- a) $\Rightarrow$ PICKUP(A) $>$ PICKUP(B) ou
  PICKUP(A) $<$ START
- b) $\Rightarrow$ PICKUP(B) $>$ PICKUP(A) ou
  PICKUP(B) $<$ START

$\Rightarrow$ Contraintres contradictoires!

$\Rightarrow$ retour-arrière

Exemple:



Menaces:

a) $\Rightarrow$ PICKUP(A) $>$ PICKUP(B)

b) $\Rightarrow$ PICKUP(B) $<$ PICKUP(A)

c) $\Rightarrow$ PUTON(A,B) $>$ PUTON(B,C)

d) $\Rightarrow$ PUTON(B,C) $<$ PICKUP(A)

e) $\Rightarrow$ PUTON(A,B) $>$ PICKUP(B)

+ Contraintes de précédence causales $\Rightarrow$ solution:

PICKUP(B),PUTON(B,C),PICKUP(A),PUTON(A,B)

Planification et raisonnement temporel

Souvent, on peut exécuter plusieurs tâches en parallèle. Dans ce cas, il n'y a pas une précédence simple entre opérateurs, mais un ordonnancement temporel à faire.

Il s'agit de satisfaire deux types de contraintes:

- précedence: un opérateur qui satisfait la précondition d'un autre doit venir avant.
- ressource: deux opérations qui utilisent la même ressource ne peuvent pas être executées en même temps.

$\Rightarrow$ trouver des intervalles de temps pour exécuter les opérateurs de facon à respecter toutes les contraintes.

Résolu par des logiques temporelles.

Opérateurs avec variables

Pour éviter de faire des choix trop tôt, on pourrait utiliser des opérateurs contenant des variables, par exemple:

```
PUTON(A,?x)
... UNSTACK(A,?x)
```

pour entreposer `A` sur un autre bloc et libérer la main

Ceci ne marchera pas:

- si `?x = A`, ni
- si `?x = B` et `B` est utilisé dans une autre opération, ni
- si `?x = ?y` et `?y = B` qui est utilisé ailleurs
- ...

Disjonctions

Disjonction de préconditions:

= plusieurs opérateurs à choix

Disjonction de postconditions:

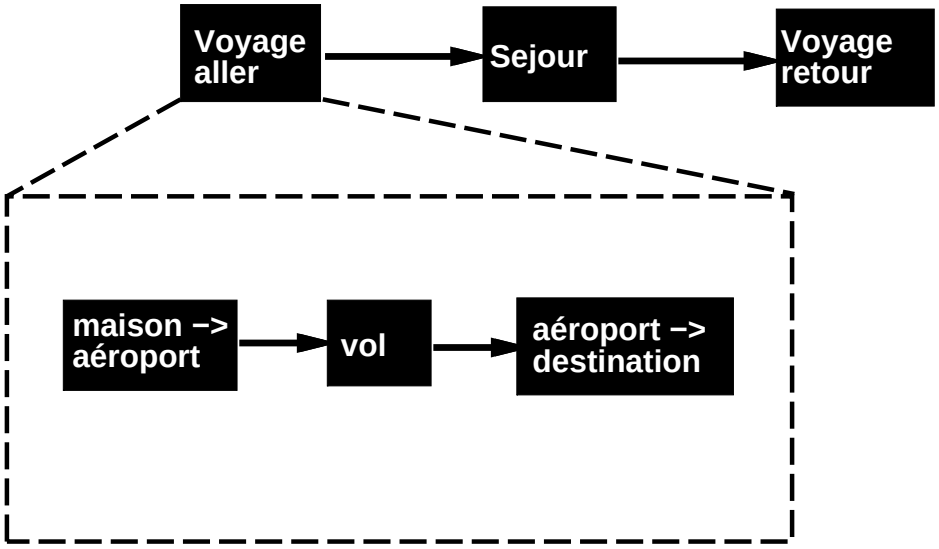
= plusieurs contextes pour poursuivre la planification

$\Rightarrow$ explosion combinatoire des possibilités

Exemple: mesures dont le résultat est incertain

Planification hiérarchique

Souvent, un plan peut être vu à plusieurs niveaux d'abstraction.



Résolution du niveau supérieur

$\Rightarrow$ définition de sous-problèmes au niveau inférieur

Plus efficace, mais peut cacher des possibilités

Planification réactive

En réalité, un agent ne peut jamais être absolument sûr que chaque opérateur produit le résultat voulu.

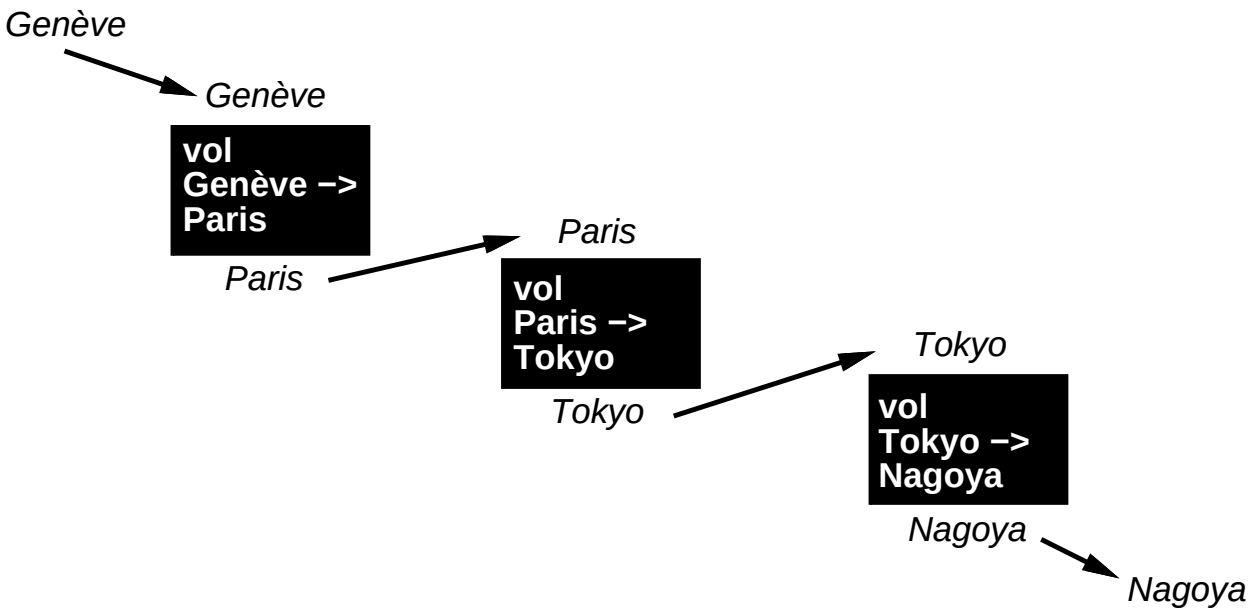
Il peut arriver qu'un agent extérieur change des conditions et oblige à adapter le plan.

En cas de conséquences disjonctives, prévoir tous les cas de figure pourrait être trop coûteux.

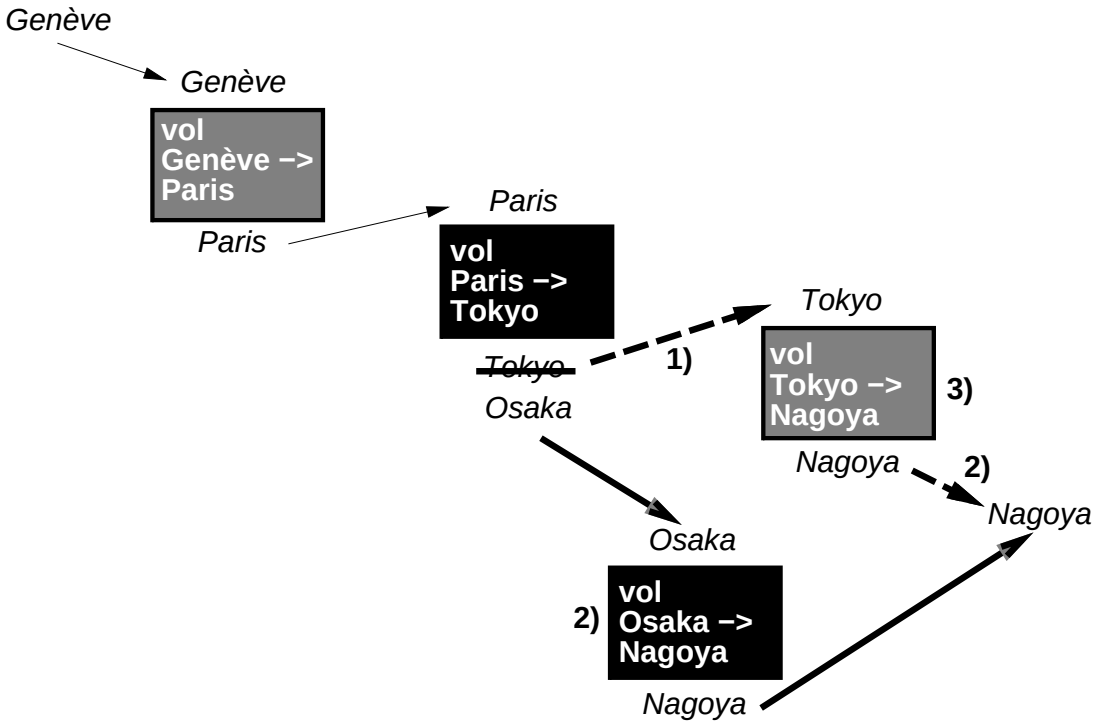
$\Rightarrow$ il faudrait une planification *réactive*

Algorithme de planification réactive

Supposons qu’une voyage de Genève à Nagoya implique de changer d’avion à Paris et Tokyo, par le plan suivant:



Si, par mauvais temps, l'avion atterit à Osaka, il faudra alors modifier le plan:



Etapes:

1. Biffer les liens causals qui ne sont plus valables.
2. Ajouter des nouvelles unifications et actions pour compléter le réseau causal.
3. Eliminer les actions qui ne sont plus utiles.

Problèmes de l'Analyse Moyens-Buts

L'analyse moyens-buts ne trouve pas tous les plans possibles. Par exemple, si le but est:

`ABOVE(A,C) ∧ ON(A,B)`

si le système commence par le but `ABOVE(A,C)`, il va mettre `A` sur `C`, et ensuite échouer, car on ne peut pas mettre `A` sur `B` sans défaire le premier but.

si le système s'attaque d'abord à `ON(A,B)`, il va ensuite échouer car on ne peut pas mettre `A` au dessus de `C` sans d'abord défaire `ON(A,B)`.

Autre problème: si la *création* d'éléments supplementaires (par exemple, des tampons) est nécessaire, la planification peut devenir impossible à calculer.

La planification en pratique

Les systèmes de planification automatique ont été utilisés dans les applications suivantes:

- planification de missions spatiales (DEVISER)
- réparation de machines (NONLIN)
- opérations d’un porte-avions (SIPE)
- assemblage robotique (Spar)
- traffic aérien

Mais un grand nombre d’applications n’est jamais publié pour des raisons stratégiques.

Résumé

Interaction entre buts et planification non-linéaire

Contraintes d’ordonnancement

Planification réactive