Planification automatique

Boi Faltings

Laboratoire d'Intelligence Artificielle
boi.faltings@epfl.ch
http://moodle.epfl.ch/

Pourquoi planifier?

1. Optimisation de ressources:
    - mission spatiale: chaque seconde coûte
    - construction: éviter des temps morts
2. Interdépendences complexes entre tâches: éviter des erreurs!
3. Agents autonomes

Planification automatique

Etant donné:

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

trouver une séquence d'actions qui va assurer que l'état initial est transformé en un état final qui remplit les conditions.

Aucune limite supérieure à la complexité du plan nécessaire $\Rightarrow$ problème très difficile.

Eléments d'un système de planification

1. Modélisation symbolique
   - de l'état du monde: *calcul de situations*
   - des actions: *opérateurs*
   - des buts
2. Algorithme de recherche pour trouver une séquence d'actions.
3. Méthodes pour traiter l'interférence entre différents buts.

Le monde des blocs

Le monde des blocs est un exemple standard de la littérature 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 suffisament 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.

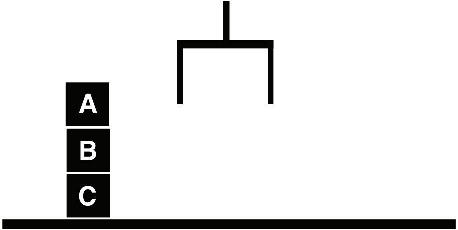
Modélisation du monde

- Le monde évolue d'un *état* à l'autre
- Etats logiquement contradictoires $\Rightarrow$ distinguer leur descriptions
- Modèle d'un état $=$ conjonction d'expressions en calcul de prédicats qui doivent être vrais.
- Toute autre proposition est supposée fausse.

Exemple d'un état

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

pourrait décrire l'état suivant du monde des blocs:



Situations et états

- Etat doit décrire même les positions de blocs qui n'ont aucune importance: plus complexe que nécessaire!
  $\Rightarrow$ représenter des *situations*: pas toutes les propositions vraies sont incluses.
- Différence: les propositions non representées peuvent être vraies ou fausses.
- Les situations permettent la représentation du but, où normalement pas toutes les propriétés de l'état sont specifiées.
- Le *calcul de situations* représente la transformation des situations par les opérateurs.

Actions

- Un plan se traduit en une séquence d'*actions*.
- Action = transformation d'un état en un autre.
- *Opérateur* = action au niveau 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 illimité, 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* P: des propositions qui doivent être vraies dans S_i pour que l'opérateur puisse s'appliquer.
- ses *postconditions* A(add): les propositions qui sont réalisées dans S_{i+1} par l'application de l'opérateur, et donc ajoutées.
- ses *suppressions* D(delete): les propositions qui ne seront plus vraies dans S_{i+1}, et donc supprimées.

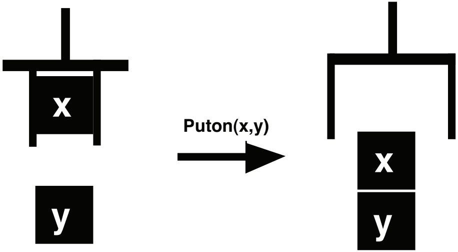
On suppose que toutes les propositions non-mentionnées dans les suppressions restent valables (axiome de cadre).
Les propositions doivent faire partie d'un ensemble fini de possibilités.

Schéma d'opérateurs

- Les opérateurs sont souvent très similaires
  $\Rightarrow$ exprimer par un *schéma* avec variables
- Instantiation d'un schéma:
  substituer les variables par des symboles;
  chaque substitution donne un autre opérateur.
- Ensemble fini de symboles $\Rightarrow$ ensemble fini de propositions.

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:

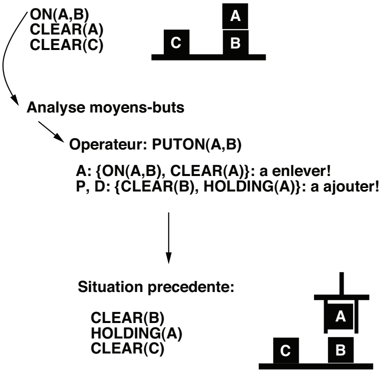
- préconditions (P): CLEAR(y), HOLDING(x)
- postconditions (A): ON(x,y), CLEAR(x), HANDEMPTY
- suppressions (D): CLEAR(y), HOLDING(x)

Schémas d'opérateurs du monde des blocs

- `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)`

Planification linéaire: STRIPS

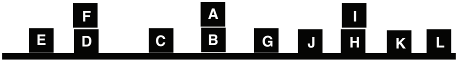
Chaînage arrière à partir du but:



$\Rightarrow$ arbre de recherche.

Chaînage avant/arrière?

Chaînage avant: beaucoup d'opérateurs!



$\Rightarrow$ chaînage arrière mieux adapté!

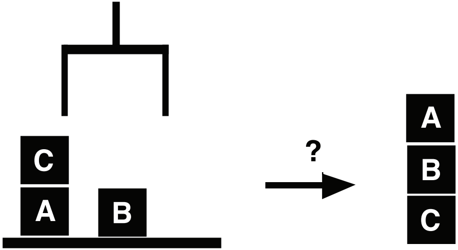
Exemple

Considérez le but de construire une 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`



Demandons à ChatGPT (GPT4)

Si bloc C est au-dessus de bloc A, et bloc B se trouve à coté sur la table, pourriez-vous me dire comment les arranger pour faire un tour de A sur B sur C? On ne peut déplacer qu'un seul bloc á la fois.

ChatGPT:

> *Initialement, nous avons cette disposition :*
> *C sur A*
> *B à côté sur la table*
> *Voici les étapes à suivre :*
> *Déplacez le bloc C sur la table (à côté de B).*
> *Déplacez le bloc A sur B.*
> *Déplacez finalement le bloc C sur A, qui est déjà sur B.*
> *A la fin de ces étapes, vous aurez réussi à empiler les blocs dans l'ordre demandé : A sur B sur C.*

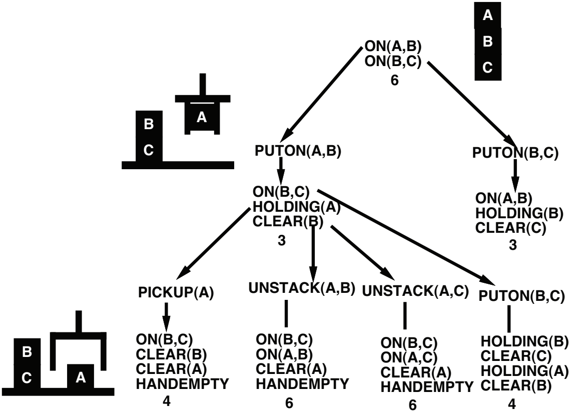
Planification avec A*

- 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`)

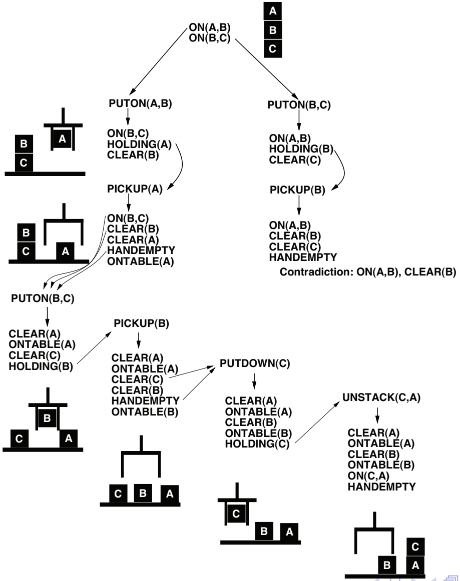
⇒ assure que le résultat est le plan qui exige le moins d'efforts.

- La recherche est terminée quand on arrive à une situation où toutes les propositions sont satisfaites par la situation initiale.

Espace de recherche



L'ordre temporel est implicite dans la génération de l'arbre.


ON(A,B)
ON(B,C)
A
B
C
PUTON(A,B)
PUTON(B,C)
ON(B,C)
HOLDING(A)
CLEAR(B)
ON(A,B)
HOLDING(B)
CLEAR(C)
PICKUP(A)
PICKUP(B)
ON(B,C)
CLEAR(B)
CLEAR(A)
HANDEMPTY
ONTABLE(A)
ON(A,B)
CLEAR(B)
CLEAR(C)
HANDEMPTY
Contradiction: ON(A,B), CLEAR(B)
PUTON(B,C)
CLEAR(A)
ONTABLE(A)
CLEAR(C)
HOLDING(B)
PICKUP(B)
CLEAR(A)
ONTABLE(A)
CLEAR(C)
CLEAR(B)
HANDEMPTY
ONTABLE(B)
PUTDOWN(C)
CLEAR(A)
ONTABLE(A)
CLEAR(B)
ONTABLE(B)
HOLDING(C)
UNSTACK(C,A)
CLEAR(A)
ONTABLE(A)
CLEAR(B)
ONTABLE(B)
ON(C,A)
HANDEMPTY

Problèmes de chaînage arrière

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

  But: `CLEAR(A)`

  Opérateur: `UNSTACK(x,y)` ⇒ `CLEAR(y)`, ...

  ⇒ `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 éliminer les situations inconsistents au plus tôt!

Problèmes de l'analyse moyens-buts

L'analyse moyens-buts ne trouve pas tous les plans possibles.
Pour le but:

`ABOVE(A,C)` $\wedge$ `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 supplémentaires (par exemple, des tampons) est nécessaire, la planification peut devenir impossible à calculer.

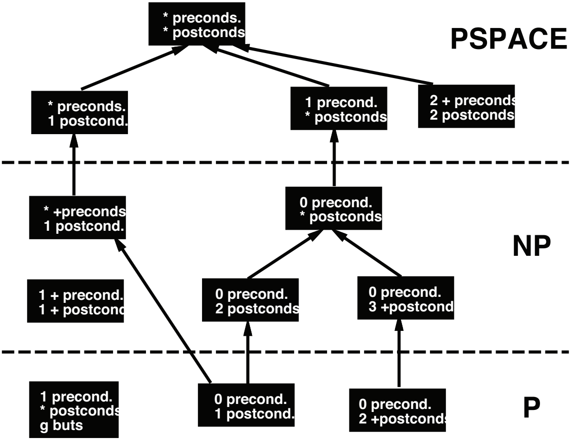
Planning Domain Definition Language (PDDL)

- Langage pour représenter des problèmes de planification.
- Exemple: opérateur `unstack`

```
(:action unstack
    :parameters (?x ?y - block)
    :precondition (and (on ?x ?y) (clear ?x) (empty))
    :effect (and (hold ?x) (clear ?y) (not (clear ?x))
                        (not (empty)) (not (on ?x ?y))))
```

- `PDDL 3.0` = version étendue du calcul de situations.
- Permet d'utiliser de nombreuses logiciels de planification.

Complexité de la planification (PLANMIN)



Comment planifier avec GPT

- formuler le problème sous format PDDL.
- entrainer un modèle pour générer des plans sur la base de situation initiale et but.
- soit in-context ou comme modèle spécifique.

Aucune recherche de la meilleure solution, mais capable de reproduire des variantes de plans connus.

Performance de GPT

Table 1: Results of GPT-4, GPT-3.5 (popularly known as ChatGPT), Instruct-GPT3.5, Instruct-GPT3 (text-davinci-002) and GPT3 (davinci) for the Plan Generation task with prompts in natural language.

Domain	Method	Instances correct				
		GPT-4	GPT-3.5	I-GPT3.5	I-GPT3	GPT-3
Blocksworld (BW)	One-shot	206/600 (34.3%)	37/600 (6.1%)	54/600 (9%)	41/600 (6.8%)	6/600 (1%)
	Zero-shot	210/600 (34.6%)	8/600 (1.3%)	-	-	-
	COT	214/600 (35.6%)	-	-	-	-
Logistics Domain	One-shot	28/200 (14%)	1/200 (0.5%)	6/200 (3%)	3/200 (1.5%)	-
	Zero-shot	15/200 (7.5%)	1/200 (0.5%)	-	-	-

Source: Subbarao Kambhampati

Performance de GPT (2)

Avec les noms des opérateurs echangés à d'autres noms:

Table 1: Results of GPT-4, GPT-3.5 (popularly known as ChatGPT), Instruct-GPT3.5, Instruct-GPT3 (text-davinci-002) and GPT3 (davinci) for the Plan Generation task with prompts in natural language.

Domain	Method	Instances correct				
		GPT-4	GPT-3.5	I-GPT3.5	I-GPT3	GPT-3
Mystery BW (Deceptive)	One-shot	26/600 (4.3%)	0/600 (0%)	4/600 (0.6%)	14/600 (2.3%)	0/600 (0%)
	Zero-shot	1/600 (0.16%)	0/600 (0%)	-	-	-
	COT	54/600 (9%)	-	-	-	-
Mystery BW (Randomized)	One-shot	12/600 (2%)	0/600 (0%)	5/600 (0.8%)	5/600 (0.8%)	1/600 (0.1%)
	Zero-shot	0/600 (0%)	0/600 (0%)	-	-	-

Source: Subbarao Kambhampati

Planification non-linéaire

- Considérons plusieurs buts qu'on peut atteindre dans n'importe quel ordre.
- La planification par A^* effectue, en parallèle, une recherche entre toutes les ordres des buts $\Rightarrow$ explosion combinatoire

$\Rightarrow$ planification *non-linéaire*:

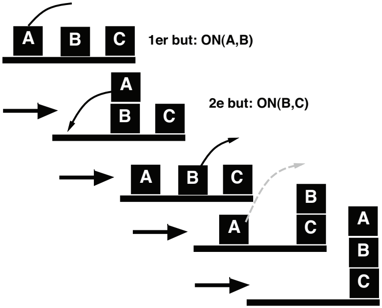
trouver d'abord les sous-plans pour chaque but en soi, et ensuite trouver l'ordre qui convient.

Buts multiples

L'ordre des buts peut être important. Par exemple:

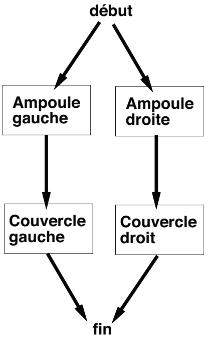
`ON(A,B)` $\wedge$ `ON(B,C)`

Mauvais ordre $\Rightarrow$ anomalie de Sussman: un plan pour atteindre un sous-but est défait pour atteindre un autre sous-but



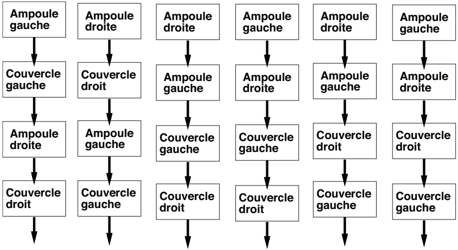
Représentation de plans non-linéaires

- Les plans représentent des *ordres partiels*.
- Si certaines actions peuvent être exécutées dans n'importe quel ordre: plus compacte.
- *Least committment*: ne pas faire des choix avant que ce soit vraiment nécessaire.



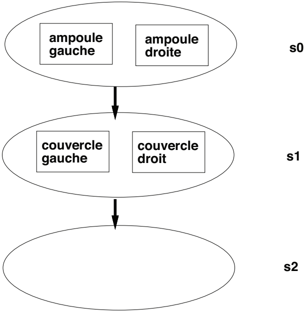
Exemple:

Changer deux ampoules:
un seul plan non-linéaire représente 6 plans linéaires:



Une représentation alternative

S'il existe une manière d'ordonner les actions, il existe une manière de les ordonner comme séquence d'actions parallèles:



Actions parallèles peuvent être executées dans n'importe quel ordre.

Planification ⇒ satisfaction de contraintes

- On fixe le nombre l d'actions parallèles (étapes) (si aucune solution n'existe, on augmente l)
- Pour chaque action possible et chaque étape ($S_i, i \in 1..l$), on introduit une variable booléenne:
  - vrai: l'action est appliquée dans cette étape
  - faux: l'action n'est pas appliquée dans cette étape

⇒ assignation à toutes les variables = plan

Exemple

- 4 actions:
  poser ampoule gauche/droite $\simeq$ `ag/ad`
  poser couvercle gauche/droite $\simeq$ `cg/cd`
- 3 étapes: s0,s1,s2
  $\Rightarrow$ variables:
  `ag(s0),ad(s0),cg(s0),cd(s0)`
  `ag(s1),ad(s1),cg(s1),cd(s1)`
- Dernier étape = état final: aucune action possible

Variables d'état

- Pour formuler les contraintes entre actions, il faut des variables qui décrivent la situation au début de l'étape.
- Pour chaque proposition possible de la représentation STRIPS et chaque étape S_i, il existe une variable booléenne
- Exemple:
  posée ampoule gauche/droite $\simeq$ `pag/pad`
  posé couvercle gauche/droite $\simeq$ `pcg/pcd`

Préconditions, conséquences $\Rightarrow$ contraintes

Contrainte logique entre action et ses préconditions:

$$\textit{action(s)} \Rightarrow \textit{précondition(s)}$$

Exemple: `ag(s0)` $\Rightarrow \neg$ `pcg(s0)`

Valeurs admissibles:

	pcg	
ag	vrai	faux
vrai	0	1
faux	1	1

Postconditions/suppressions:
contraintes avec variables de l'étape suivante:

$$\textit{ag(s0)} \Rightarrow \textit{pag(s1)}$$

Buts, conditions initiales ⇒ contraintes

Conditions initales ⇒ contraintes sur les variables de la première étape.

Exemple:

`C1:pag(s0)=faux;C2:pad(s0)=faux;`

`C3:pcg(s0)=faux;C4:pcd(s0)=faux;`

Buts = contraintes sur les variables de la dernière étape

Exemple:

`C1:pag(s2)=vrai;C2:pad(s2)=vrai;`

`C3:pcg(s2)=vrai;C4:pcd(s2)=vrai;`

Axiomes de cadre $\Rightarrow$ contraintes

- Variable d'état pas touchée par une action
  $\Rightarrow$ valeur à la prochaine étape reste identique.
- Equivalent:
  si la valeur change, il y a une action qui en est responsable.

$\Rightarrow$ pour toutes paires de variable d'état d'étapes successives, ajouter une contrainte:

> *si la valeur de la variable change, il doit y avoir une action qui l'a comme postcondition.*

qui implique les 2 variables et une ou plusieurs actions.

- Exemple:
  $\neg\ \texttt{pcg(s0)} \wedge \texttt{pcg(s1)} \Rightarrow \texttt{cg(s0)}$
  $\texttt{pcg(s0)} \wedge \neg\ \texttt{pcg(s1)} \Rightarrow \perp$

Contraintes d'exclusion

- Considérons deux actions a_1 et a_2 qui ont une précondition p et une postcondition $\neg\ p$
- a_1 et a_2 ne peuvent pas s'executer tous les deux sur la base de p, car p sera rendu fausse par la première action!
- $\Rightarrow$ contrainte d'exclusion mutuelle (mutex) entre des actions telle que l'une affecte la précondition de l'autre.

Planification = solution du PSC

Les contraintes n'admettront que des plans valables:

```
pag(s0) = faux; pad(s0) = faux; pcg(s0) = faux; pcd(s0) = faux;
ag(s0) = vrai; ad(s0) = vrai; cg(s0) = faux; cd(s0) = faux;
pag(s1) = vrai; pad(s1) = vrai; pcg(s1) = faux; pcd(s1) = faux;
ag(s1) = faux; ad(s1) = faux; cg(s1) = vrai; cd(s1) = vrai;
pag(s2) = vrai; pad(s2) = vrai; pcg(s2) = vrai; pcd(s2) = vrai;
```

Il peut y avoir plusieurs solutions ⇒ plusieurs plans.

Efficacité

- Solveurs efficaces permettent de résoudre des PSC booléens (SAT) de millions de variables
  $\Rightarrow$ applicable à des problèmes de planification très grands!
- La planification par CSP est la technique la plus efficace pour des plans complexes.

Planification et raisonnement temporel

- Dernière étape: l'ordonnancement des opérations.
  $\Rightarrow$ satisfaire deux types de contraintes:
  - précédence: 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.

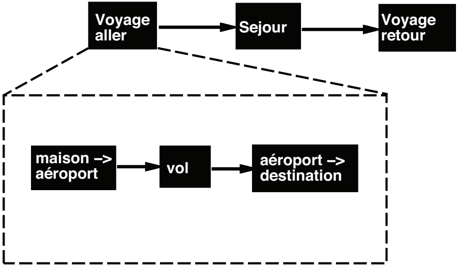
  $\Rightarrow$ trouver des intervalles de temps pour exécuter les opérateurs de façon à respecter toutes les contraintes.
- Résolu par des logiques temporelles.

Planification heuristique

- Pour des problèmes peu contraints, des méthodes linéaires avec des heuristiques peuvent s'avérer efficaces.
- Exemple: `FF` (fast forward) enchaîne des actions en chaînage avant.
- Fonctionne bien quand il y a peu d'interactions entre buts.

Planification hiérarchique

- Réduire la complexité par plusieurs niveaux d'abstraction:



- Solution du niveau supérieur
  $\Rightarrow$ définition de sous-problèmes au niveau inférieur
- Peut cacher des possibilités: impossible de spécifier qu'on veut utiliser la même compagnie à l'aller et au retour pour un tarif moins cher.

La planification en pratique

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

- planification de missions spatiales.
- opérations d'un porte-avions.
- opération d'un ensemble d'ascenseurs.
- robots autonomes.

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

Application: Planification des mouvements d'ascenseurs

- Ascenseurs réglés individuellement $\Rightarrow$ vont tous sur le même étage.
- Schindler a mis au point un système de planification basé sur le formalisme STRIPS et traduction en PSC.
- Le système coordonne les ascenseurs.
- Résultat: temps de parcours réduit de $10 - 50\%$.
- Permettrait aussi de réduire le nombre d'ascenseurs.

Résumé

- Représentation du monde, buts et opérateurs: Calcul de Situations
- Planification linéaire: STRIPS
- Planification non-linéaire
- Planification non-linéaire par satisfaction de contraintes