Onzième partie XI

Planification

En bref ...

Qu'est ce que la planification ?

Représentation classique en planification

La recherche dans un espace d'états

La recherche dans un espace de plans

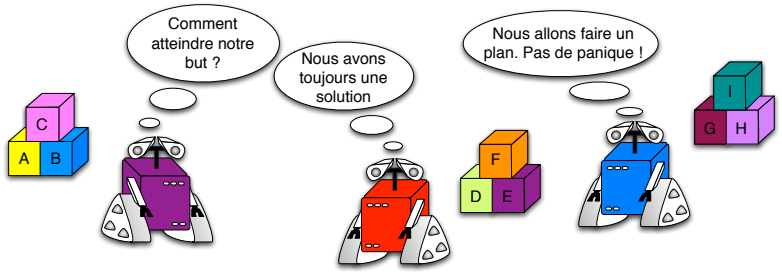
Technique de planification dans les graphes

Plan

Qu'est ce que la planification ?

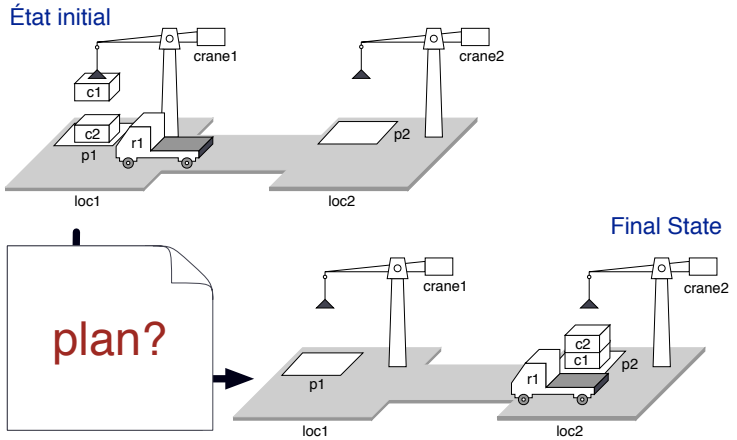
Pour débuter une définition...

"Planning is the reasoning side of acting. It is an abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes. This deliberation aims at acheiving as best as possible some prestated objectives. Automated planning is an area of Artificial Intelligence that studies this deliberation process computationaly.[1]*"*

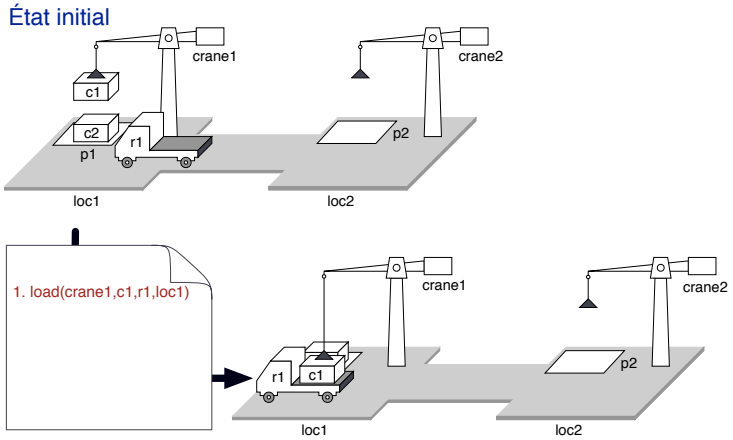


1. Automated Planning, M. Ghallab et al., Morgan Kaufmann 2004.

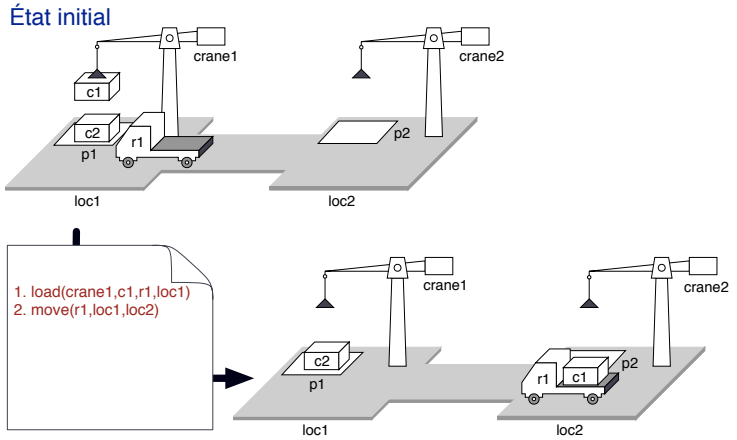
Exemple : les robots " Dockers "



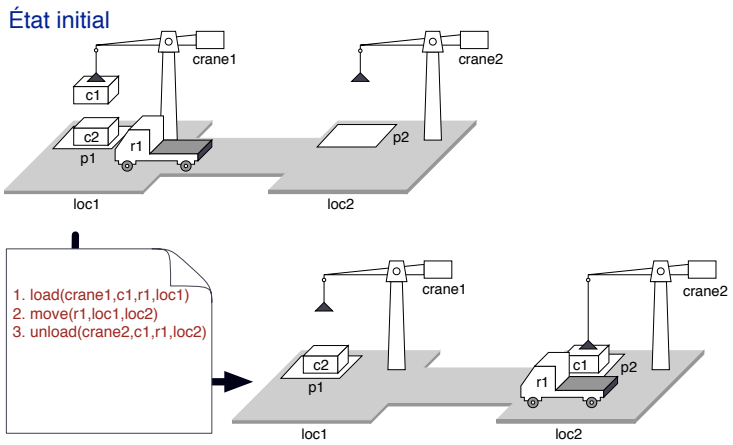
Exemple : les robots " Dockers "



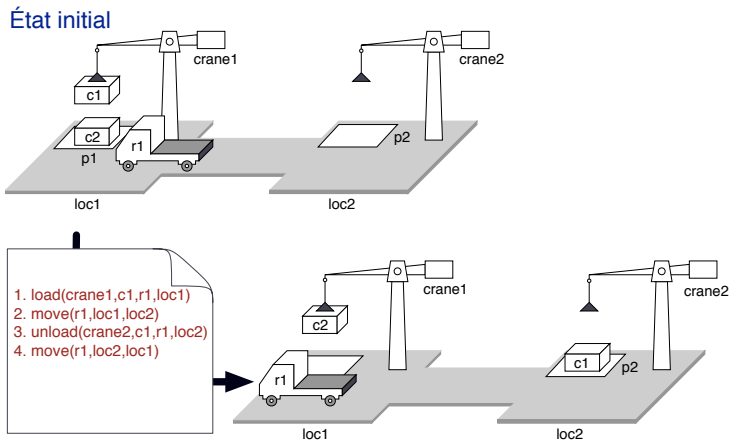
Exemple : les robots " Dockers "



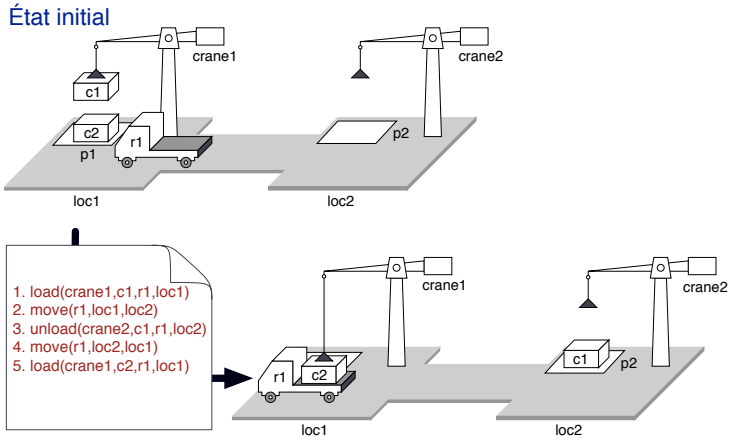
Exemple : les robots " Dockers "



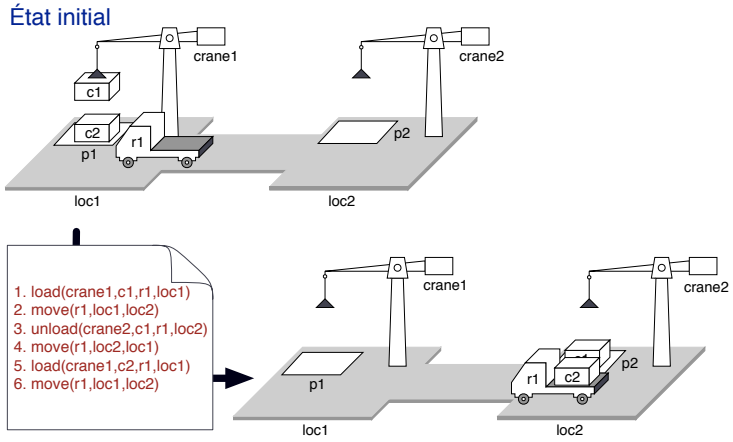
Exemple : les robots " Dockers "



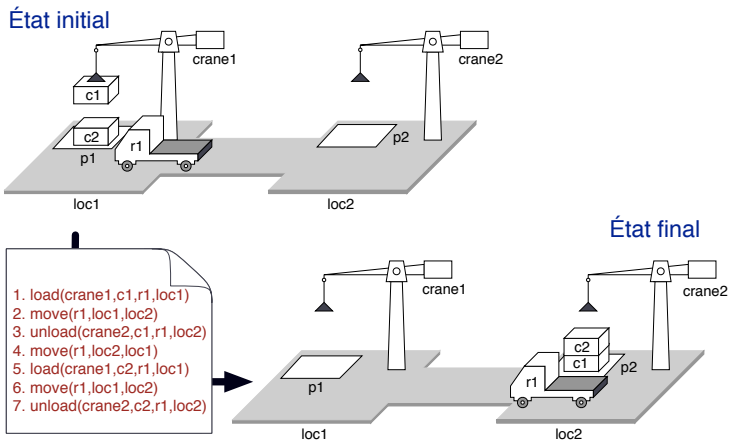
Exemple : les robots " Dockers "



Exemple : les robots " Dockers "



Exemple : les robots " Dockers "

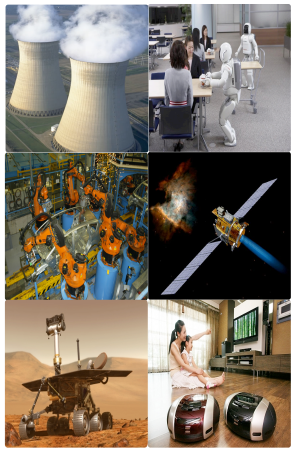


Motivations

- D'un point de vue pratique
  1. Concevoir des systèmes adaptables et autonomes capable de raisonner sur leurs capacités et les contraintes provenant de l'environement
  2. Définir des mécanismes génériques permettant à des agents autonomes de se coordoner et de coopérer afin de d'accomplir des tâches complexes qu'ils ne peuvent pas réaliser seuls
  3. Définir des mécanismes abstraits pour permettre la composition automatique de fonctionnalités
- D'un point de vue théorique
  1. Maîtriser les différents aspects de l'intelligence en voyant la planification comme un composant important de la rationnalité individuelle et collective
  2. Étudier la planification non seulement comme un processus abstrait et indépendent mais aussi comme un composant intégré à un mécanisme de délibération plus global

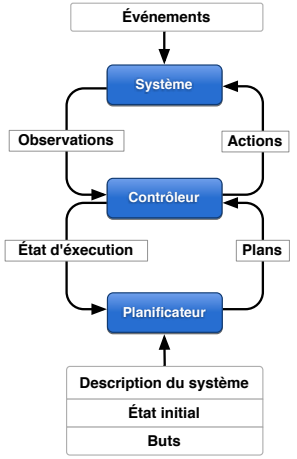
Quelques domaines d'applications

- Aérospaciale
  - Deep Space One, Prises de vues satellitaires
  - Mars Exploration Rover
- Militaire
  - Mission de déminage navale, Mission d'hélicopters de combats
- Robotique industrielle
  - Ligne d'assemblage, transport
- Vie de tous les jours
  - Aspirateur automatique, la tondeuse automatique
- Informatique
  - Jeux vidéo
  - Composition de services web, Informatique ambiente



Représentation graphique du modèle de la planification

- Le modèle conceptuel de la planification est défini par l'interaction de 3 composants
  1. Un système à transition d'états Σ qui décrit l'évolution de l'envionnement extérieur (le problème à résoudre) selon les événements et les actions qu'il reçoit
  2. Un contrôleur qui fournit une action à exécuter selon l'état courant
  3. Un planificateur génère un plan d'action pour atteindre le but spécifié



Système état-transition

Définition (Système état-transition)

Un système *état-transition* Σ est un tuple $\Sigma = (S, A, E, \gamma)$ tel que :

1. $S = \{s_0, s_1, s_2, \ldots\}$ est un ensemble d'états
2. $A = \{a_1, a_2, \ldots\}$ est un ensemble d'actions que le contrôleur peut exécuter
3. $E = \{e_1, e_2, \ldots\}$ est un ensemble d'évènements qui peuvent agir sur l'environnement
4. $\gamma : S \times (A \cup E) \rightarrow 2^S$ est une fonction de transition entre états

Représentation grâce à un graphe orienté :

- chaque état est représenté par un noeud
- si $s' \in \gamma(s, u)$ où $u \in A \cup E$, une transition existe entre les états s et s'

$\rightarrow$ un arc portant l'étiquette u relie les noeuds s et s'

Représentation en planification

Restrictions du modèle pour la planification classique

- Le modèle classique de la planification formule plusieurs hypothèses restrictives :
  - Σ est fini : le système Σ possède un nombre fini d'états
  - Σ est complètement observable : le controleur a une complète connaissance des états du système Σ.
  - Σ est déterministe : pour chaque état s et pour chaque événement et action u, $|\gamma(s, u)| \leq 1$.
  - Σ est statique : l'ensemble des événements E est vide. Σ ne possède pas de dynamique interne.
  - Les plans sont séquentielle : un plan solution est une séquence finie d'actions.
  - Le temps est implicite : les actions et les événements n'ont pas de durée.
  - La planificateur est "offline" : le planificateur ne prend pas en compte les changements pouvant survenir à Σ pendant qu'il planifie.

Représentation classique en planification

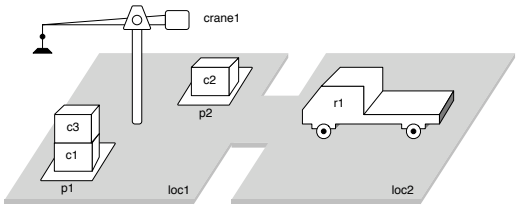
- La représentation classique (appelée STRIPS) s'appuie sur une notation dérivée de la logique du premier ordre.
  - Soit $\mathcal{L}$ un langage de la logique du premier ordre qui possède un nombre fini de symboles de prédicat et aucun symbole de fonction.
  - Les états sont représentés par un ensemble de formules logiques atomiques de $\mathcal{L}$
  - Les actions sont représentées par des opérateurs de planification qui expriment les changements de vérité des formules atomiques contenues dans les états
- La représentation logique des états et actions permet de raisonner sur les relations entres états

$\rightarrow$ La planification peut expoiter ces relations pour trouver un plan

Représentation des états

Définition(État)

Un état est un ensemble complètement instancié de formules atomiques (aussi appelé propositions) construites sur $\mathcal{L}$. $\mathcal{L}$ ne possède pas de symboles de fonction. Par conséquent, l'ensemble S de tous les états possibles est assuré d'être fini. Une formule atomique p est vraie dans un état$s$ ssi $p \in s$. Si g est un ensemble de formules atomiques, nous disons que s satisfait g (noté $s \models g$) lorsque qu'il existe une substitution σ telle que pour chaque formules positive $g' \in \sigma(g)$, $g' \in s$ et aucune formule négative est dans s.



$$s = \left\{ \begin{array}{l} attached(p1, loc1), \\ attached(p2, loc1), \\ in(c1, p1), in(c3, p1), \\ on(c3, c1), on(c1, pallet), \\ top(c3, p1), in(c2, p2), \\ top(c2, p2), \ldots \end{array} \right\}$$

Opérateur de planification

Les opérateurs de planification définissent la fonction de transition γ entre les états du système d'états transitions Σ

Définition (Opérateur de planification)

Un opérateur de planification est un triplet o =(name(o), precond(o), effects(o)) dont les éléments sont définis comme suit :

- name(o), le nom de l'opérateur, est une expression de la forme $n(x_1, \ldots, x_k)$ où n est un symbole appelé le nom de l'opérateur (n est unique dans $\mathcal{L}$) et $x_1, \ldots, x_k$ sont les symboles des variables qui apparaîssent dans les autres parties de o.
- precond(o) et effects(o), les préconditions et les effets de o, définissent respectivement les propriétés qui doivent être vérifiées pour appliquer l'opérateur et les propriétés modifiées par l'application de l'opérateur.

Exemple d'opérateur de planification

Exemple (Operateur Take)

L'opérateur de planification take(k,l,c,d,p) peut être défini comme suit :

; ; crane k at location l takes c off of d in pile p

take(k,l,c,d,p)

precond : belong(k,l), attached(p,l), empty(k), top(c,p), on(c,d)

effects : holding(k,c), ¬empty(k), ¬in(c,p), ¬top(c,p), ¬on(c,d), top(d,p)

Actions

Définition (Action)

Une action est une instance d'un opérateur de planification. Si a est une action et s est un état tel que $\text{precond}^+(a) \subseteq s$ et $\text{precond}^-(a) \cap s = \emptyset$, alors a est applicable dans s, et l'état résultant de l'application de a à l'état s est l'état :

$$\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$$

Domaine de planification

Définition (Domaine de planification)

Un domain de planification définit sur $\mathcal{L}$ est un système d'états transition $\Sigma = (S, A, \gamma)$ tel que :

1. $S \subseteq 2^n$, avec n le nombre d'atomes complètement instanciés de $\mathcal{L}$
2. $A = \{$toutes les instances complètement instanciées des opérateurs dans $\mathcal{O}\}$ où $\mathcal{O}$ est l'ensemble des opérateurs définis précédemment
3. $\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$ si $a \in A$ est applicable dans $s \in S$, sinon $\gamma(s, a)$ n'est pas définie
4. S est clos sous γ, i.e., si $s \in S$, alors pour chaque a qui est applicable dans s, $\gamma(s, a) \in S$.

Problème de Planification

Définition (Problème de planification)

Un problème de planification est un triplet $\mathcal{P} = (\mathcal{O}, s_0, g)$ où :

- $\mathcal{O}$ est un ensemble d'opérateurs de planification
- s_0, l'état initial, i.e., un état de S
- g, le but, est un ensemble de propositions

Extension de la représentation classique

- La représentation classique STRIPS est extrêmement limitée. Des extensions sont nécessaires pour décrire des problèmes intéressants.
- La représentation ADL ajoute les extensions suivantes :
  - Le typage des variables
  - Les opérateurs de planification conditionnels
  - Les expressions quantifiées
  - Les préconditions disjonctives
  - L'axiomatique et l'inférence
- Aujourd'hui, les langages de planification utilisent une syntaxe standard appelée PDDL (Planning Domain Description Langage).
  - Les représentations STRIPS et ADL sont des sous-langages de PDDL

Exemple PDDL

Exemple (Crane robot transportation domain)

```
(define (domain dwr)
    ( :requirements :strips :typing)
    ( :types location pile robot crane container)
    ( :predicates
        (adjacent ?l1 ?l2 - location) (attached ?p - pile ?l -location)
        (belong ?k - crane ?l - location) (at ?r - robot ?c container)
        (occupied ?l - location) etc. )
    ( :action move
        ( :parameters ( ?r - robot ?from ?to - location))
        ( :precondition (and (adjacent .from ?to) (at ?r ?from)
              (not (occupied ?to))))
        ( :effect (and (at ?r ?to) (not (occupied ?from)) (occupied ?to)
              (not (at ?r ?from)))))
    ( :action load ... ))
```

Les espaces d'états

Qu'est-ce que la planification dans les espaces d'états ?

Qu'est ce que la planification dans les espaces d'états ?

- Les algorithmes les plus simples pour la planification.
- Les algorithmes de recherche dans lesquels l'espace de recherche est un sous ensemble d'un espace d'états :
  - Chaque nœud correspond à un état du monde
  - Chaque arc correspond à une transition, i.e., une action
  - Le plan courant correspond à un chemin dans l'espace d'états de l'état initial à un état but

Recherche en chaînage avant

- L'algorithme de recherche en chaînage avant est deterministe
- L'algorithme de recherche en chaînage avant est correct et complet
  - L'algorithme de recherche en chaînage avant prend en entrée un problème de planification $\mathcal{P} = (\mathcal{O}, s_0, g)$. Si $\mathcal{P}$ est résoluble, alors Forward-search($\mathcal{O}$, s_0, g) retourne un plan solution. Sinon, il retourne échec.
- Le plan retourné par chaque invocation récursive de l'algorithme est appelé solution partielle car c'est une partie de la solution finale retournée par l'invocation de haut niveau

L'algorithme de recherche en chaînage avant

Algorithme (ForwardSearch($\mathcal{O}$, s_0, g))

if s_0 *satisfies* g **then return** *an empty plan* π ;
active $\leftarrow \{a \mid a$ *is a ground instance of an operator* $\mathcal{O}$
and precond(a) is true in $s_0\}$;
if active $= \emptyset$ **then return** Failure ;
nondeterministically choose an action $a_1 \in$ active ;
$s_1 \leftarrow \gamma(s, a_1)$;
$\pi \leftarrow$ `ForwardSearch`$(\mathcal{O}, s_1, g)$;
if $\pi \neq$ Failure **then return** $a_1 \cdot \pi$;
else return Failure ;

- Tout repose sur le choix (non déterministe) de l'action
- Si l'on choisit la meilleur action estimée selon une heuristique, cet algorithme revient à faire une recherche gloutonne (en profondeur)

Principe du planificateur "*Fast Forward*

- *Fast Forward (FF)* est un algorithme de chaînage avant très performant
- *Fast Forward* applique une recherche gloutonne et s'appuie sur :
  - une heuristique basée sur une formulation relaxée du problème
  - un algorithme spécifique de gradient appelé Enforced Hill Climbing
  - une heuristique qui s'appuie sur étude d'atteignabilité
  - un agenda de buts
  - une heuristique pour élaguer les successeurs non prometteurs
- Si la recherche n'aboutit pas à un plan, *FF* applique A*

Enforced Hill Climbing

Algorithme (EHC($\mathcal{O}$, s_0, g))

add s_0 *to* open ;
best_heuristic $\leftarrow$ `heuristic`(s_0) ;
while open $\neq \emptyset$ **do**
    state $\leftarrow$ `pop`(open) ;
    successors $\leftarrow$ `successors`(s) ;
    while successors $\neq \emptyset$ **do**
        next_state $\leftarrow$ `pop`(successors) ;
        if $g \in$ next_state **then return** `reconstruct_plan`(next_state) ;
        $h \leftarrow$ `heuristic`(next_state) ;
        if $h <$ best_heuristic **then**
            clear successors ;
            clear open ;
            best_heuristic $\leftarrow h$;
        add next_state *to the end of* open ;

Recherche en chaînage arrière

- L'idée est de débuter la recherche en partant du but et d'appliquer de manière inverse les opérateurs de planification pour produire des sous buts
- L'algorithme s'arrête s'il produit un ensemble de formules atomiques qui satisfait l'état initial
- L'algorithme de chaînage arrière est correct et complet

Algorithme (BackwardSearch($\mathcal{O}$, s_0, g))

if s_0 *satisfies* g **then return** *an empty plan* π ;
revelant $\leftarrow \{a \mid a$ *is a ground instance of an operator* $\mathcal{O}$
that is revelant for $g\}$;
if revelant $= \emptyset$ **then return** Failure ;
nondeterministically choose an action $a_1 \in$ revelant ;
$g_1 \leftarrow \gamma^{-1}(g, a_1)$;
$\pi \leftarrow$ `BackwardSearch`($\mathcal{O}$, s_0, g_1) ;
if $\pi \neq$ Failure **then return** $\pi \cdot a_1$;
else return Failure ;

L'algorithme STRIPS

- Le plus important problème de l'approche précédente est sa lenteur
  - $\Rightarrow$ Comment augmenter l'efficacité de l'algorithme en réduisant la taille de l'espace d'états construit ?
- STRIPS est un algorithme qui tente de répondre à cette question
- Il diffère de l'algorithme en chaînage arrière de deux manières :
  1. À chaque appel récurcif de l'algorithme STRIPS, seuls les préconditions du dernier opérateur sont utilisées comme sous-but. Cette modificiation réduit le coefficient de branchement de l'espace d'états de manière sinificative, mais rend l'algorithme STRIPS incomplet.
  2. Si l'état courant satisfait toutes les préconditions d'un opérateur, STRIPS s'engage de manière gloutonne à exécuter cet opérateur, i.e., il n'y aura aucun retour arrière sur ce choix. Cette technique élague une importante partie de l'espace d'états mais rend également STRIPS incomplet.

L'algorithme STRIPS

Algorithme (STRIPS($\mathcal{O}$, s, g))

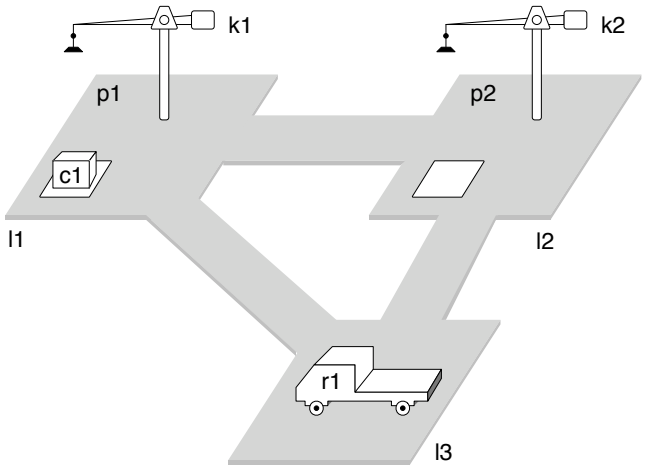
$\pi \leftarrow$ *the empty plan ;*
while true **do**
  if s *satisfies* g **then return** π ;
  revelant $\leftarrow \{a \mid a$ *is an ground instance of* $\mathcal{O}$ *revelant for* $g\}$;
  if revelant $= \emptyset$ **then return** Failure ;
  nondeterministically choose an action $a \in$ revelant ;
  $\pi' \leftarrow$ STRIPS($\mathcal{O}$, s, *precond*(a)) ;
  if $\pi' =$ Failure **then return** Failure ;
  $s \leftarrow \gamma(s, \pi')$; ; *if we get here, then* π' *achieves precond*(a) *from* s ;
  $s \leftarrow \gamma(s, a)$; ; s *now satisfies precond*(a) ;
  $\pi \leftarrow \pi \cdot \pi' \cdot a$;

Les espaces de plans

Qu'est-ce que la recherche dans un espace de plans ?

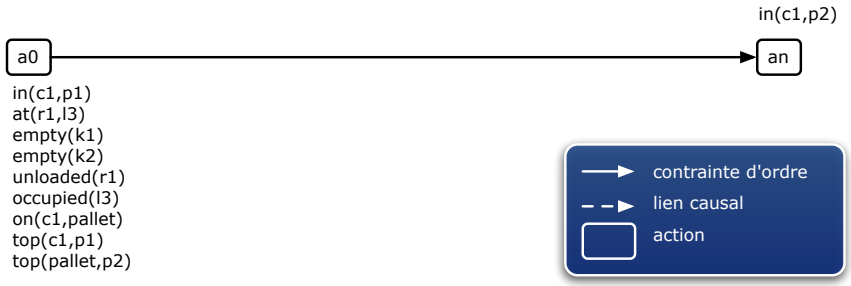
- L'espace de recherche n'est plus un espace d'états mais un espace de plans
  - Les nœuds sont des plans partiellement spécifiés
  - Les arcs sont des opérateurs de rafinement permettant de compléter le plan partiel, e.g., en permettant d'atteindre un but encore non atteint, en supprimant une incohérence dans le plan, etc.
- La solution d'un plan change, on parle de plan partiel
- La planification est considérée comme un processus de rafinement de plans qui peut :
  1. ajouter des actions
  2. ajouter des contraintes d'ordre entre les actions
  3. ajouter des liens causaux
  4. ajouter des contraintes d'instantiation sur les variables

Exemple



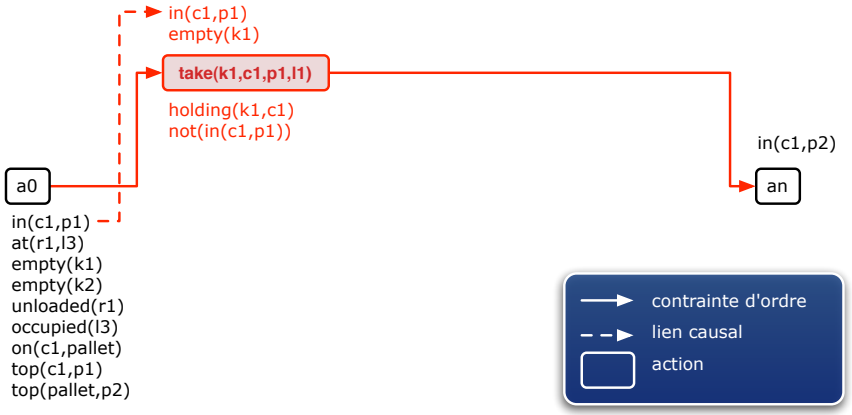
Exemple

Plan partiel initial



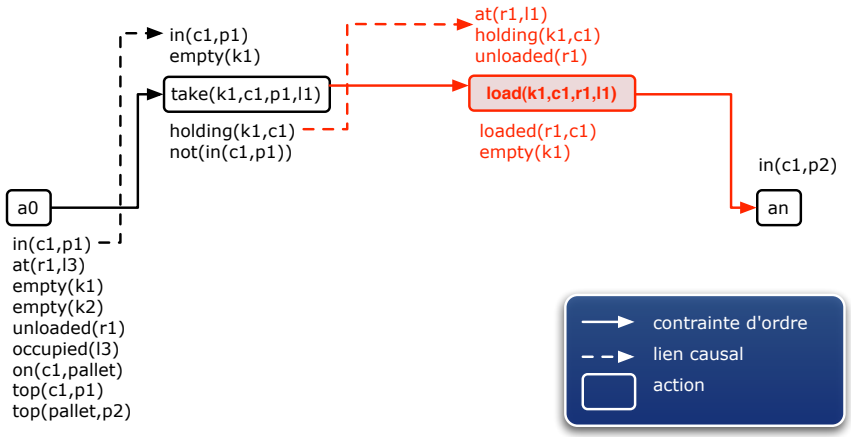
Exemple

Ajout de l'action take



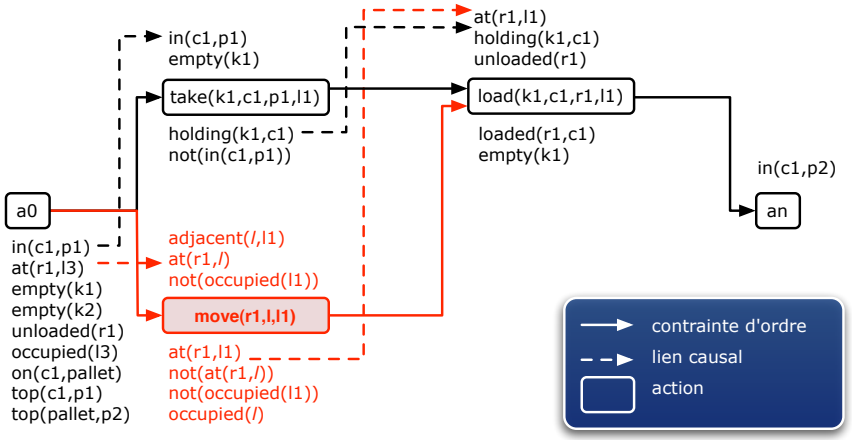
Exemple

Ajout de l'action load



Exemple

Ajout de l'action move



Plan partiel

Définition (Plan partiel)

Un plan partiel est un tuple $(A, \prec, B, L)$ où :

- $A = \{a_1, \ldots, a_k\}$ est un ensemble d'opérateurs de planification partiellement instantiées
- $\prec$ est un ensemble de contraintes d'ordre sur A de la forme $(a_i \prec a_j)$.
- B est un ensemble de contraintes d'instantiation sur les variables des actions de A de la forme $x = y$, $x \neq y$, ou $x \in D_x$, D_x est un sous ensemble du domaines de définition de la variable x.
- L est un ensemble de liens causaux de la forme $\langle a_i \xrightarrow{p} a_j \rangle$, tels que a_i et a_j sont des actions de A, les contraintes d'ordre $(a_i \prec a_j)$ sont définis dans $\prec$, la proposition p est un effet de a_i et une précondition de a_j, et les contraintes d'instantiation pour les variables de a_i et a_j apparaissant dans p sont dans B.

Plan solution

Définition (Plan solution)

Un plan partiel $\pi = (A, \prec, B, L)$ est un plan solution d'un problème de planification $P = (\Sigma, s_0, g)$ si :

- l'ensemble des contraintes d'ordre $\prec$ et l'ensemble des contraintes d'instantiation B doivent être conhérents
- toutes les séquence d'actions totalement ordonées et totalement instantiées de A satisfont $\prec$ et définissent un chemin dans le système d'états transitions Σ partant de l'état initial s_0 (correspondant aux effets de l'action a_0) à un état contenant toutes les propositions du but g (donné par les préconditions de l'action a_n).

Défaut et menace

Définition (Menace)

Une action a_k d'un plan π est une menace pour un lien causal $(a_i \xrightarrow{p} a_j)$ ssi :

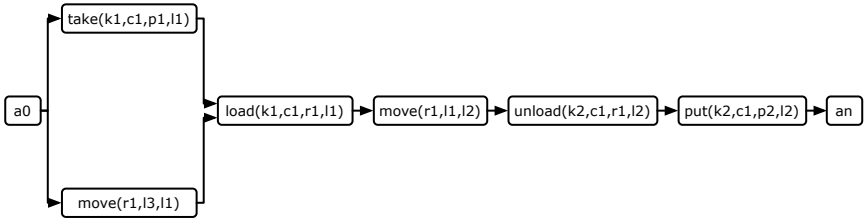
- a_k a un effet $\neg q$ qui est potentiellement inconhérent avec p
- les contraintes d'ordre $(a_i \prec a_k)$ et $(a_k \prec a_j)$ sont cohérentes avec B
- les contraintes d'instantiation résultant de l'unification de q et p sont cohérentes avec B

Définition (Défaut)

Un défaut dans un plan $\pi = (A, \prec, B, L)$ est soit :

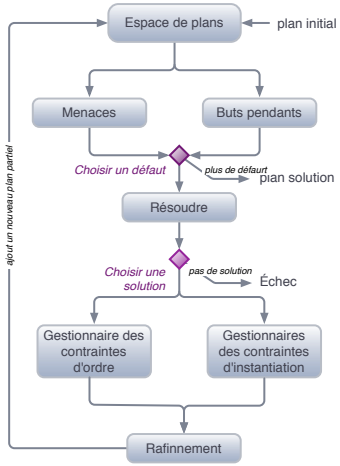
- un sous-but, i.e., la précondition d'une action de A n'est pas supportée par un lien causal
- une menace, i.e., une action qui interfère avec un lien causal

Exemple : plan solution



L'algorithme PSP

- Un plan π est solution lorsqu'il ne possède plus de défaut. Le principe de l'algorithme est de rafiner π, en maintenant $\prec$ et B cohérents, tant que π contient des défauts
- L'étape des l'algorithme PSP pour obtenir un plan solution sont les suivantes :
  1. Calculer les défauts du plan π, i.e., ses sous-buts pendants et ses menaces
  2. Choisir un défaut
  3. Calculer une manière de le résoudre
  4. Choisir une solution pour remédier au défaut
  5. Rafiner π, i.e., appliquer la solution choisie



L'algorithme PSP

Algorithme (PSP(π))

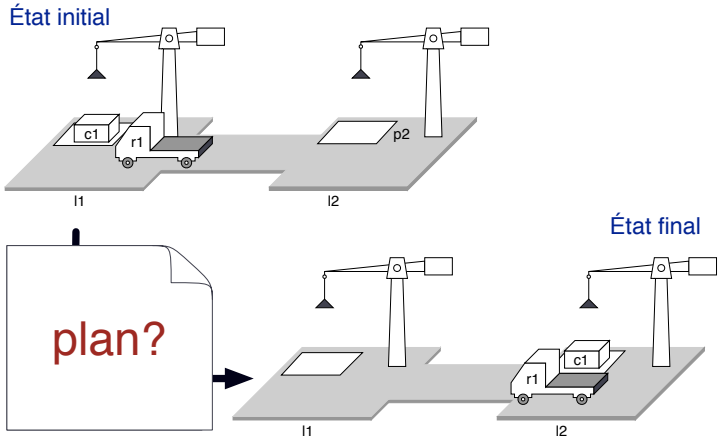
flaws $\leftarrow$ `OpenGoals`(π) $\cup$ `Threat`(π) ;
if flaws $= \emptyset$ **then return** π ;
select any flaw $\sigma \in$ flaws ;
resolvers $\leftarrow$ `Resolve`(σ,π) ;
if resolvers $= \emptyset$ **then return** Failure ;
nondeterministically choose a resolver $\rho \in$ resolvers ;
$\pi' \leftarrow$ `Refine`(ρ, π) ;
return PSP(π') ;

Les graphes de planification

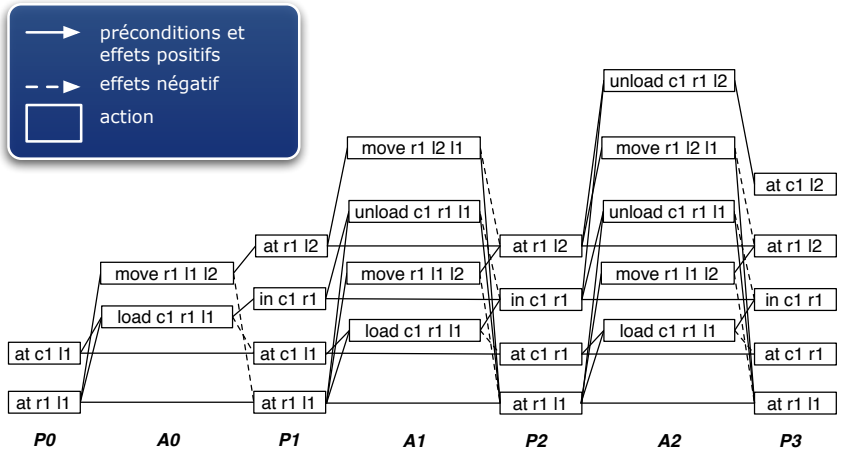
Qu'est-ce que la planification dans les graphes ?

- La planification dans les graphes s'appuie sur une représentation classique
- Cette technique introduit un nouvel espace de recherche appelé graphe de planification
- La planification dans les graphes produit des plans solutions qui sont des séquences d'ensembles d'actions
  - La planification dans les espaces d'états produit des séquences d'actions
  - La planification dans les espaces de plans produit des plans partiellement ordonnés
  - La planification dans les graphes est moins expressive que dans les espaces de plan mais plus que dans les espaces d'états

Exemple : Graphe de planification



Exemple : Graphe de planification



Indépendance entre actions

Définition (Indépendance)

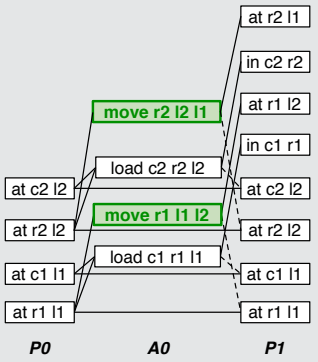
Deux actions (a, b) sont independantes ssi :

- $\text{effects}^-(a) \cap [\ \text{precond}(b) \cup \text{effect}^+(b)\ ] = \emptyset$
- $\text{effects}^-(b) \cap [\ \text{precond}(a) \cup \text{effect}^+(a)\ ] = \emptyset$

Remarques

1. L'indépendance entre actions n'est pas spécifique à un problème particulier
2. C'est une propriété intrinsèque des actions du domaine de planification qui peut être calculée *a priori*

Exemple (Indépendance)



Indépendance entre actions

Définition (Indépendance)

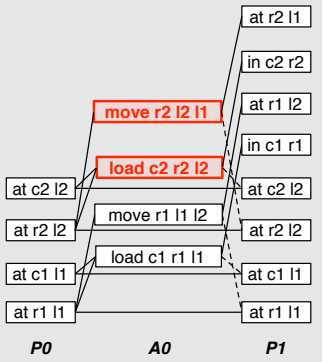
Deux actions (a, b) sont independantes ssi :

- $\text{effects}^-(a) \cap [\ \text{precond}(b) \cup \text{effect}^+(b)\ ] = \emptyset$
- $\text{effects}^-(b) \cap [\ \text{precond}(a) \cup \text{effect}^+(a)\ ] = \emptyset$

Remarques

1. L'indépendance entre actions n'est pas spécifique à un problème particulier
2. C'est une propriété intrinsèque des actions du domaine de planification qui peut être calculée *a priori*

Exemple (Dependance)



Les relations d'exclusions mutuelles (1/2)

- Constat :
  - Les ensembles de propositions à un même niveau du graphe ne préservent pas la cohérence (un niveau du graphe encode plusieurs états)
  - Certaines actions ne sont pas indépendantes
- Problème :
  - comment capturer les incompatibilités entre les actions et propositions dans le graphe de planification ?

Solution

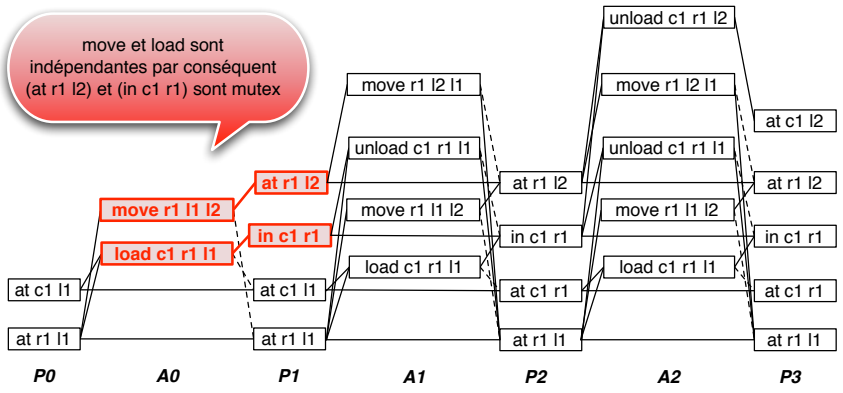
La solution est de conserver une trace des incompatibilités entre actions et propositions aussi appelées relations d'exclusions mutuelles en s'appuyant sur le critère de dépendance entre actions.

Les relations d'exclusions mutuelles (2/2)

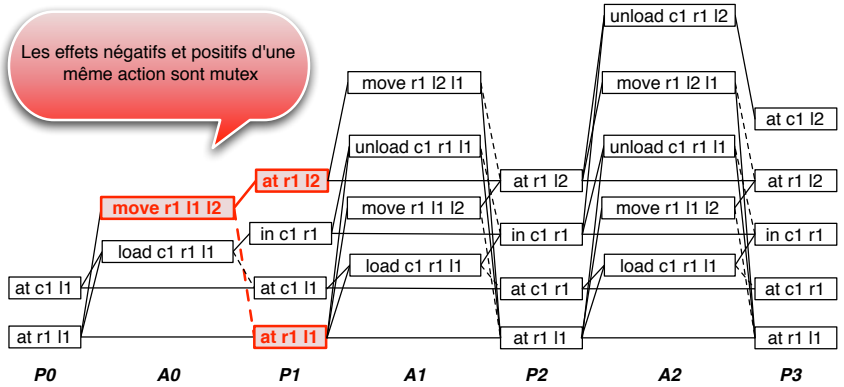
Définition (Exclusion mutuelle)

- Deux actions a et b d'un niveau A_i sont mutex si :
  1. a et b sont dependantes ou
  2. une précondition de a est mutex avec une précondition de b
- Deux propositions p et q d'un niveau P_i sont mutex si :
  1. toutes les actions de A_i qui produisent p comme effet (en incluant les no-op) sont mutex avec toutes les actions qui produisent q au même niveau et
  2. il n'existe pas d'action au niveau A_i qui produise à la fois p et q

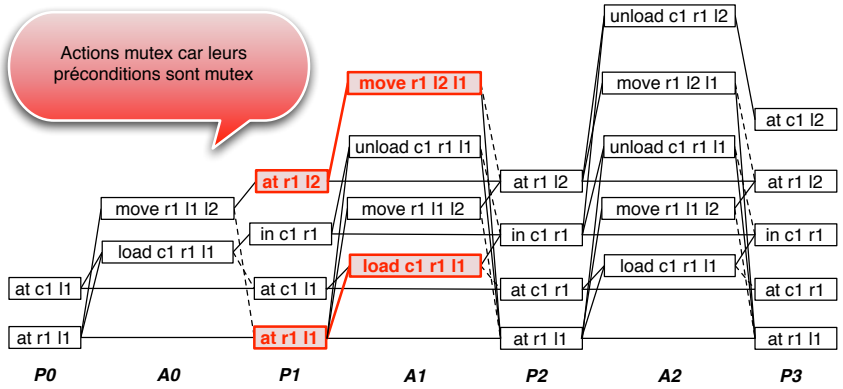
Exemple : Propagation des relations d'exclusions



Exemple : Propagation des relations d'exclusions



Exemple : Propagation des relations d'exclusions



Le planificateur Graphplan

- L'algorithme du planificateur Graphplan réalise une recherche proche de la recherche itérative en profondeur, découvrant une nouvelle partie de l'espace de recherche à chaque itération. Il exécute itérativement les traitements suivants :
  1. Étend le graphe de planification d'un niveau supplémentaire
  2. Extrait en chaînage arrière un plan solution en partant du dernier niveau du graphe
- La première extraction a lieu à un niveau P_i dans lequel toutes les propositions du but sont inclues et sont non mutex
  - cela n'a aucun sens de débuter la recherche avant, i.e., la condition ci-dessus est une condition nécessaire mais pas suffisante
- La boucle d'expansion et d'extraction se poursuit jusqu'à ce qu'un plan solution soit trouvé ou que la condition de terminaison soit atteinte

Expansion du graphe de planification

Algorithme (Expand($\langle P_0, A_1, \mu A_1, \ldots, A_{i-1}, \mu A_{i-1}, P_{i-1}, \mu P_{i-1}\rangle$))

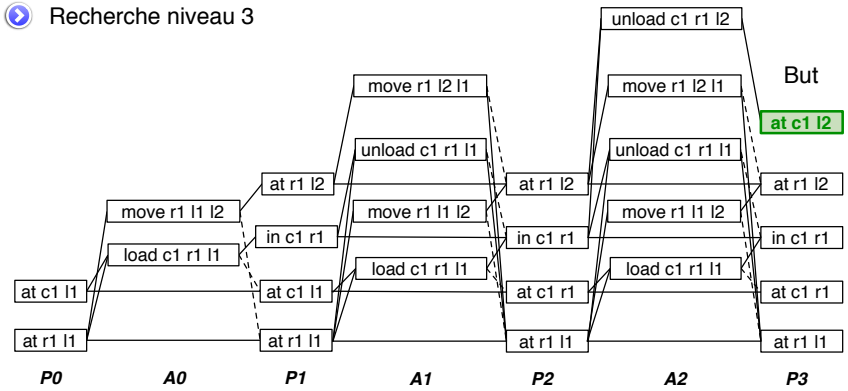
$A_i \leftarrow \{a \in A \mid precond(a) \in P_{i-1} \text{ and } precond(a) \cap \mu P_{i-1} = \emptyset\}$
$P_i \leftarrow \{p \mid \exists a \in A_i : p \in effects^+(a)\}$
$\mu A_i \leftarrow \{(a,b) \in A_i, a \neq b \mid a, b \text{ are dependent}$
$\quad$ or $\exists (p,q) \in \mu P_{i-1} : p \in precond(a) \text{ and } q \in precond(b)\}$
$\mu P_i \leftarrow \{(p,q) \in P_i, p \neq q \mid \forall a, b \in A_i, a \neq b :$
$\quad p \in effects^+(a) \text{ and } q \in effects^+(b) \Rightarrow (a,b) \in \mu A_i\}$
foreach $a \in A_i$ **do**
  link a with a precondition arcs to $precond(a)$ in P_{i-1}
  link a with a positive arcs to $effects^+(a)$ in P_i
  link a with a negative arcs to $effects^-(a)$ in P_i
return $\langle P_0, A_1, \mu A_1, \ldots, P_{i-1}, \mu P_{i-1}, A_i, \mu A_i, P_i, \mu P_i\rangle$

Extraction d'un plan solution

- L'extraction d'un plan solution du graphe de planification correspond à la recherche dans un arbre ET/OU :
  - En partant d'une proposition du but g, les branches-OU sont les arcs qui partent vers les actions au niveau précédent et qui produisent la proposition
  - En partant d'une action, les branches-ET sont les préconditions de l'action
- La recherche peut échouer $\Rightarrow$ un ensemble de propositions sont alors considérées comme inatteignables
- Idée : enregistrer ces ensembles de propositions qui ne sont pas atteignables pour éviter de les chercher à nouveau
  - Cet enregistrement est réalisé dans une table de hâchage notée ∇
  - Cette table de hâchage est indexée par le niveau dans lequel l'ensemble des propositions est considéré comme inatteignable

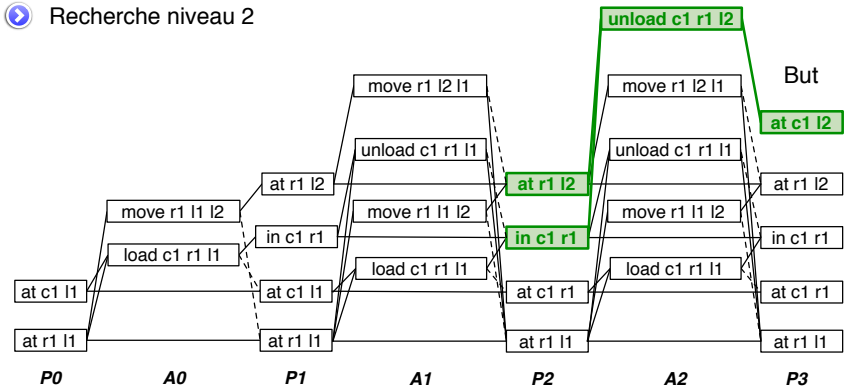
Un exemple de recherche

Recherche niveau 3



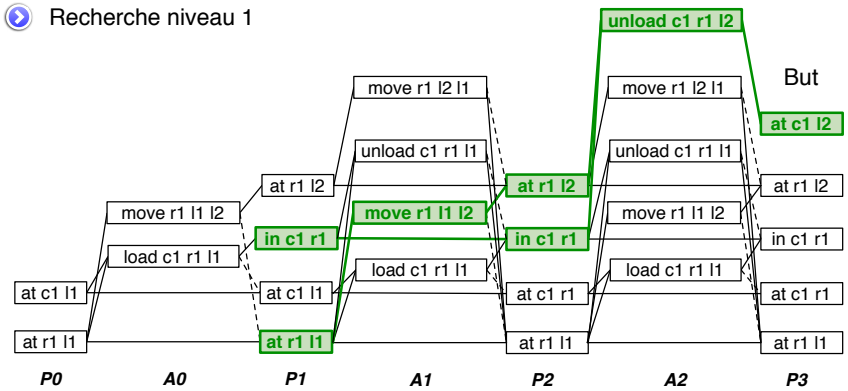
Un exemple de recherche

Recherche niveau 2



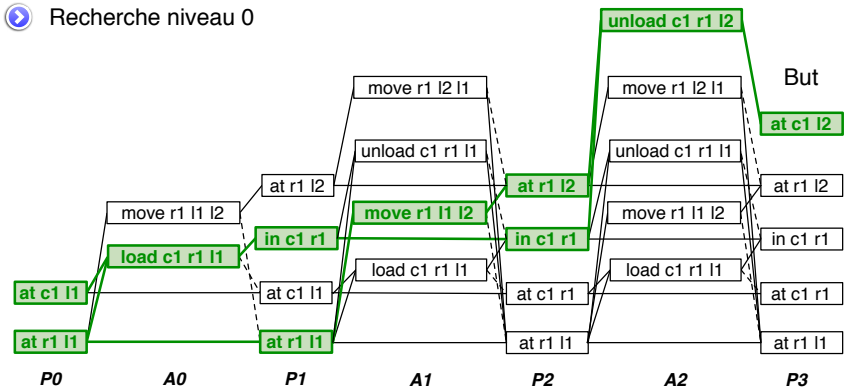
Un exemple de recherche

Recherche niveau 1



Un exemple de recherche

Recherche niveau 0



Les procédures de recherche

Algorithme (Extract(G, g, i))

if $i = 0$ **then return** $\langle\rangle$
if $g \in \nabla(i)$ **then return failure**
$\pi \leftarrow$ `GP-Search`$(G, g, \emptyset, i)$
if $\pi \neq$ **failure then return** π
$\nabla(i) \leftarrow \nabla(i) \cup \{g\}$
return failure

Les procédures de recherche

Algorithme (GP-Search(G, g, π, i))

if $g = \emptyset$ **then**
  $\Pi \leftarrow$ `Extract`$(G, \bigcup\{precond(a) \mid \forall a \in \pi\}, i-1)$
  if $\Pi =$ **failure then return failure**
  return $\Pi.\langle\pi\rangle$
else
  select any $p \in g$
  resolvers $\leftarrow \{a \in A_i \mid p \in effects^+(a)$ *and* $\forall b \in \pi : (a, b) \notin \mu A_i\}$
  if resolvers $= \emptyset$ **then return failure**
  nondeterministically choose $a \in$ resolvers
  return `GP-Search`$(G, g - effects^+(a), \pi \cup \{a\}, i)$

L'algorithme de Graphplan

Algorithme (Graphplan(A, s_0, g))

$i \leftarrow 0$, $\nabla \leftarrow \emptyset$, $P_0 \leftarrow s_0$, $G \leftarrow \langle P_0\rangle$
repeat
  $i \leftarrow i+1$, $G \leftarrow$ `Expand`(G, g, i)
until $[g \subseteq P_i$ *or* $g \cap \mu P_i = \emptyset]$ *or* `Fixedpoint`(G)
if $g \not\subseteq P_i$ *or* $g \cap P_i \neq \emptyset$ **then return failure**
$\Pi \leftarrow$ `Extract`(G, g, i)
if `Fixedpoint`(G) **then** $\eta \leftarrow |\nabla(\kappa)|$ **else** $\eta \leftarrow 0$
while $\Pi =$ **failure do**
  $i \leftarrow i+1$, $G \leftarrow$ `Expand`(G, g, i), $\Pi \leftarrow$ `Extract`(G, g, i)
  if $\Pi =$ **failure** *and* `Fixedpoint`(G) **then**
    if $\eta = |\nabla(\kappa)|$ **then return failure**
  $\eta \leftarrow |\nabla(\kappa)|$
return Π

Conclusion

- Les systèmes de planification sont des solveurs qui opèrent dans des représentations logiques explicites des états et actions
- Différentes approches existent :
  - Aucune n'est au dessus des autres, chacune a ses adeptes
  - The International Planning Competition (IPC) : une compétition annuelle internationale
- Le cadre de la planification évolue dans de nombreuses directions :
  - Prise en compte du temps,
  - Prise en compte du coût et de la limitation des ressources
  - Incertitudes dans la perception ou dans l'action
  - Non déterminisme de l'environnement
  - Planification multi-agents
  - ...