SYSTEMES ELEMENTAIRES DE GENERATION DE PLANS D'ACTIONS EN ROBOTIQUE

0. Introduction Les problèmes où les buts sont atteints au moyen de **séquences d'actions** sont des problèmes de l'I.A. La résolution des problèmes en robotique est un domaine dans lesquels ces types de problèmes peuvent se poser.

1. Position des problèmes en Robotique: Un robot a un repertoire d'actions qu'il peut accomplir dans un monde facile à comprendre. Par exemple un véhicule qui accomplit des taches, déplacer des objets dans un environnement contenant d'autres objets etc….

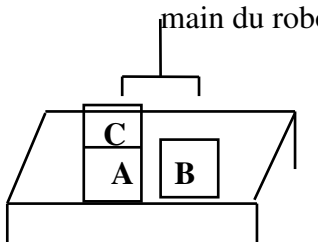
Exemple: Dans le monde des cubes, on a plusieurs cubes sur une table ou les uns sur les autres et un robot ayant une main capable de ramasser et de déplacer des cubes.

La programmation d'un robot nécessite l'intégration de nombreuses fonctions :
- la perception du monde qui l'entoure (le monde où il évolue, **les états**),
- les **actions** qu'il peut accomplir (**les règles**),
-la formulation de plans d'actions (**séquence de règles**) et le **contrôle de son exécution**.

Problème: Synthétiser une séquence d'actions, qui si elle est exécutée convenablement, le robot atteindra **le but** fixé à partir d'un **état initial**. La synthèse d'actions d'un tel problème peut être résolue par un système de production (SP) où la BDG décrit l'état du monde dans lequel évolue le robot et les règles représentent les actions du robot.

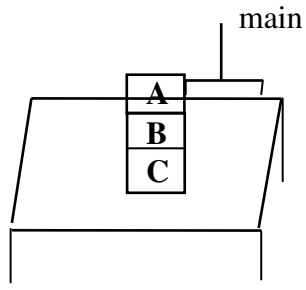
1.1 Description d'états du monde et de buts: Les descriptions d'états du monde et de buts des problèmes de robotique peuvent être construites à l'aide **de fbfs:**

Exemple :



Cet **état initial** peut être représenté par la conjonction:
découvert(B) $\wedge$ *découvert(C)* $\wedge$ *sur(C,A)* $\wedge$ *surtable(A)* $\wedge$ *surtable(B)* $\wedge$ *mainvide*

But:



but par la conjonction: *sur(B,C)* $\wedge$ *sur(A,B)*

1.2 Les Actions du robot: Les actions d'un robot transforment un état en un autre, on les modélise par des règles-av. Une technique simple et efficace pour représenter ces actions a été utilisée par un système de résolution de problème en robotique appelé **STRIPS**. Lorsqu'une règle-av est appliquée à un état, on change l'état en ajoutant une structure supplémentaire et en supprimant des expressions qui ne sont plus vraies.

Exemple: Si le robot ramasse B, alors *surtable(B), decouvert(B)* et *mainvide* ne sont plus vraies, on les supprime de la description d'état et on ajoute *tenu(B)* dans la nouvelle description qui est: *découvert(C)* $\wedge$ *sur(C,A)* $\wedge$ *surtable(A)* $\wedge$ ***tenu(B)***

Les règles-av de format STRIPS spécifient les expressions à supprimer et les expressions à ajouter au moyen d'une liste. Ces règles sont des quadruplés de la forme:
(<Nom_de_l'action (Paramètres)>, < Preconditions> , <Ajouts> , <Retraits>)
où : **<Nom_de_l'action>:** est le nom de la règle.

<Paramètres>: arguments de la règle, éventuellement des variables existentielles.

<Pré condition>: une expression qui doit être une Conséquence Logique de la description d'état, via une substitution σ, pour que la règle-av puisse être appliquée.

<Retraits>: une liste de littéraux. Lorsque la règle-av est appliquée à un état, la substitution est appliquée à cette liste et les instances obtenues sont supprimées de l'ancien état.

<Ajouts>: Une conjonction de littéraux, semblable à une consequence d'une regle-av. Lorsqu'une regle-av est appliquée à un état, la substitution est appliquée à cette liste et l'instance résultante est ajoutée à l'ancienne description.

On obtient ainsi un nouvel état.

Rmq: Les variables des Retraits et Ajouts doivent apparaitre dans la pré condition.

Résumé: Soit un état E_i composé de L1∧...Ln∧B1∧...∧Bm, soit une règle R ayant une pré condition P qui s'unifie avec L1,...,Ln par σ, ayant une liste de retrait LR et d'ajout LA. Après application de la règle R à E_i, on obtient l'état $E_{i+1}=\sigma(E_i - LR + LA)$

Exemple: *Ramasser(x)*

Pré condition: surtable(x) ∧ mainvide ∧ découvert(x)
Retraits: surtbale(x), mainvide, découvert(x)
Ajouts: tenu(x)

Avec l'état initial « *découvert(B)∧découvert(C)∧sur(C,A)∧surtable(A)∧surtable(B)∧ mainvide* » et en appliquant cette règle via la substitution x/B, le nouvel état est (Ancien état - Retraits + Ajouts): *découvert(C)∧ sur(C,A) ∧ surtbale(A) ∧* ***tenu(B)***.

2. Comme Un Système de Production en Chainage-Av:

On part de l'état initial, on sélectionne des règles-av applicables (espace de recherche) jusqu'à ce qu'on produise un état qui s'unifie avec le but.

Les règles de formats STRIPS correspondantes aux actions du robot de l'exemple sont

1) ramasser(x): Pré-condition & Retrait: surtable(x), découvert(x), mainvide
Ajout: tenu(x)
2) poser(x): Pré condition & Retrait: tenu(x)
Ajout: surtable(x)∧découvert(x)∧ mainvide
3) empiler(x, y): Pré condition & Retrait: tenu(x), découvert(y)
Ajout: mainvide∧ sur(x, y)∧decouvert(x)
4) désempiler(x, y): Pré condition & Retrait: mainvide, sur(x, y), decouvert(x)
Ajout: tenu(x)∧ découvert(y).

Soit le but à atteindre précédent, en appliquant ces règles suivant une certaine stratégie, on obtient un espace de recherche. On cherche un chemin de l'état initial vers l'état but.

Exercice Donner l'espace de recherche

La séquence d'actions trouvée est la suivante, (appelée plan d'actions):
désempiler(C,A), poser(C), ramasser(B), empiler(B,C), ramasser(A), empiler(A,B).

3. Comme Un Système de Production en Chaînage arrière :

On veut définir un système qui travaille en chaînage arrière (allant du but vers l'état initial). On applique des règles-ar pour produire des sous-buts. Il s'achève avec succès lorsqu'on produit un sous-buts qui s'unifie avec l'état initial. Il faut avoir des règles-ar qui transforment des buts en sous-buts. Ces règles-ar reposent sur les règles-av. Une règle-ar qui transforme un but B en sous-but B' repose d'un point de vue logique sur la règle-av qui lorsqu'elle est appliquée à un état filtrée par B' donne un état filtrée par B.

On remarque que lorsqu'une règle-av est appliquée à un état (sa pré-condition est vérifiée), on produit un état qui s'unifie avec les littéraux de la liste d'ajouts (puisqu'on ajoute la liste d'ajuts). C'est pourquoi si une expression de but contient un littéral L qui s'unifie avec un des littéraux de la liste d'ajouts d'une règle-av, alors nous savons que si nous produisons un état filtré par des instances adéquates des préconditions de cette règle-av, elle peut être appliquée pour produire un état filtré par L. Par conséquent, l'expression de sous but produite par une application en chaînage arrière d'une règle-av doit contenir des instances des préconditions de cette règle-av. Mais si le but contient des littéraux autre que L alors le sous-but doit aussi contenir des littéraux qui, après l'application de la règle-av, sont transformés en ces littéraux autres que L.

Formellement soit un but *(L∩B1∩... ∩Bn).* On veut utiliser une règle-av (en chaînage arrière) pour produire un sous-but. Soit une règle-av F avec une pré condition P, une liste de retrait S et une liste d'ajouts A contenant un littéral L' qui s'unifie avec L par σ. Les littéraux dans σP forment un sous-ensemble des littéraux du sous-but recherché. Nous devons inclure dans le sous but les expressions B1', B2',....,Bn' qui doivent être telles que l'application de l'instance σF de la règle-av à n'importe quelle description d'état filtrée par ces expressions produise une description d'état filtrée par B1, ..., Bn (pour avoir *L∩B1∩... ∩Bn).* Le sous but obtenu est donc: σP, B1', B2',....,Bn'.

Définition: Chaque Bi' est un littéral appelé la régression de Bi à travers l'instance σF de la règle-av F, obtenue par l'algorithme suivant:

Soit *R(Q; σF)* la régression d'un littéral Q à travers une instance close σF d'une règle-av F dont la pré condition est P, la liste des retraits S et la liste des ajouts A. Alors:

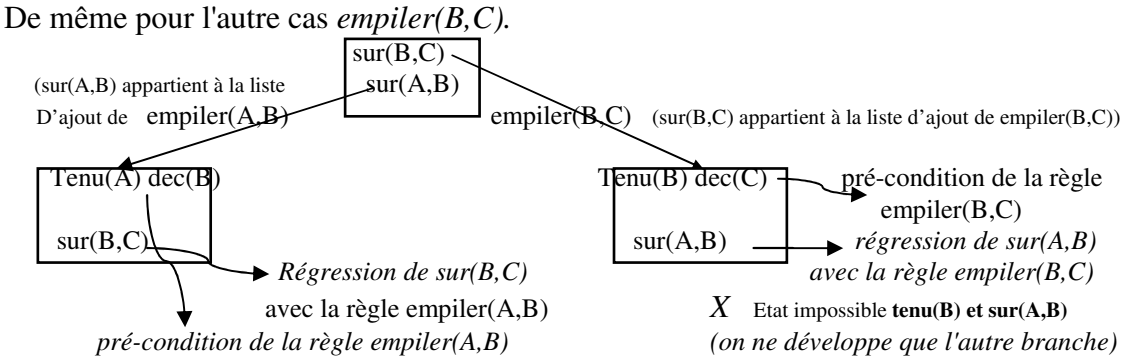
R(Q; σF): = Si σQ est un littéral dans σA */*dans liste d'Ajouts de la règle* σF **/*
Alors Vraie
Sinon Si σQ est un littéral dans σS */*dans liste de Retraits de la règle*σF **/*
Alors FauxSinon σQ */*ni dans liste d'Ajouts ni liste de Retraits */*

En résumé: Une règle-av peut être utilisée comme règle-ar de la manière suivante:

- La condition d'applicabilité de la règle-ar est que l'expression de but contienne un littéral qui s'unifie par σ avec l'un des littéraux de la liste des ajouts de la règle-av .
- L'expression de sous-buts est créée **en régressant les autres littéraux** (non unifiés) de l'expression de but à travers l'instance par σ de la règle-av et en connectant ceux-ci au moyen d'une **conjonction à l'instance de pré-condition de la règle-av**.

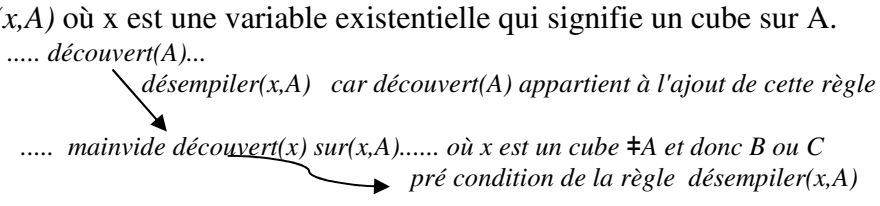
Elagage des nœuds: On ne développe pas les sous/buts dans lesquels on a une situation impossible comme par exemples: *Tenu(B)∧ sur(A,B)..,mainvide∧Tenu(A)..., tenu(A)∧tenu(B)....., toute description contenant Faux (produit par la régression),* etc...
Remarque: Ce processus est valable pour une liste de littéraux au lieu d'un littéral L.

Exemple 1: Soit le but: *sur(A, B) ∧ sur(B, C).* Il y'a 2 façons d'utiliser *empiler(x,y)* comme règle-ar sur ce but, car ces 2 littéraux appartiennent à la liste d'ajout de la règle via les substitutions {*x/A,y/B}* et *{x/B,y/C}*. Soit la 1ere *empiler(A,B*). La description de sous but est : Régresser les expressions (non unifiés) *sur(B,C)* à travers *empiler(A,B)* ce qui donne *sur(B,C) (algorithme précédent).* Ajouter la pré-condition de la règle *Tenu(A), découvert(B)* pour avoir le sous but : *sur(B,C) ∧Tenu(A)∧ découvert(B).*
De même pour l'autre cas *empiler(B,C).*

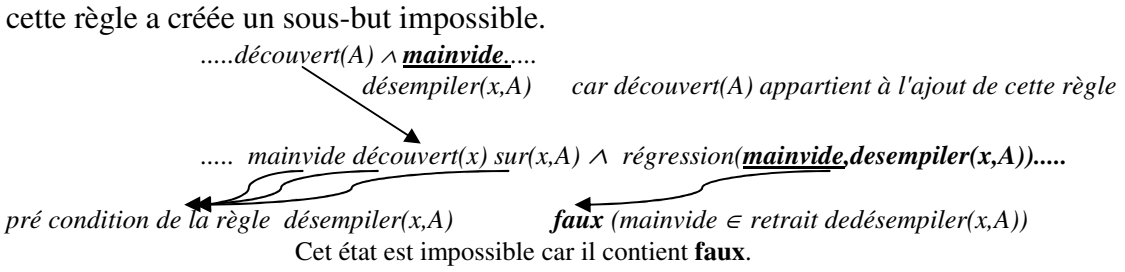


Exercice: Compléter l'espace de recherche jusqu'à l'obtention de l'état initial

Exemple 2 : Soit un but contenant *découvert(A).* 3 règles ont ce littéral dans leur liste d'ajouts. Soit *désempiler(x,y*) par {y/A}, l'expression de sous but créée est {*mainvide∧ découvert(x)∧sur(x,A)* où x est une variable existentielle qui signifie un cube sur A.

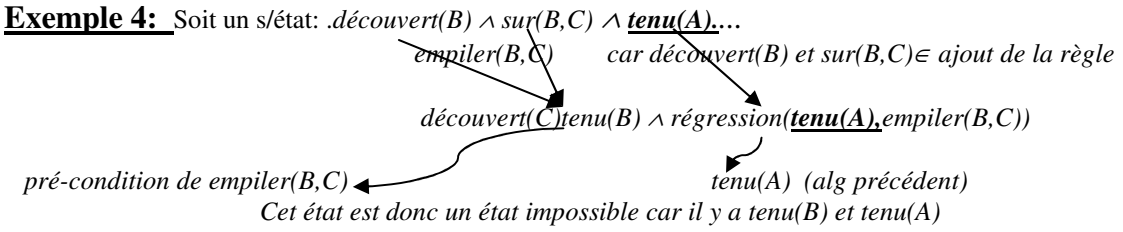


Exemple 3 : Supposons que nous voulions appliquer la règle *désempiler* à une expression de but *découvert(A) ∧ Mainvide.....*, via la substitution {*y/A}*. La régression de *Mainvid*e à travers *désempiler(x,A)* est *Faux.* Nous voyons donc que l'application de cette règle a créée un sous-but impossible.



Cet état est impossible car il contient **faux**.

Exemple 4: Soit un s/état: *.découvert(B) ∧ sur(B,C) ∧ tenu(A).…*

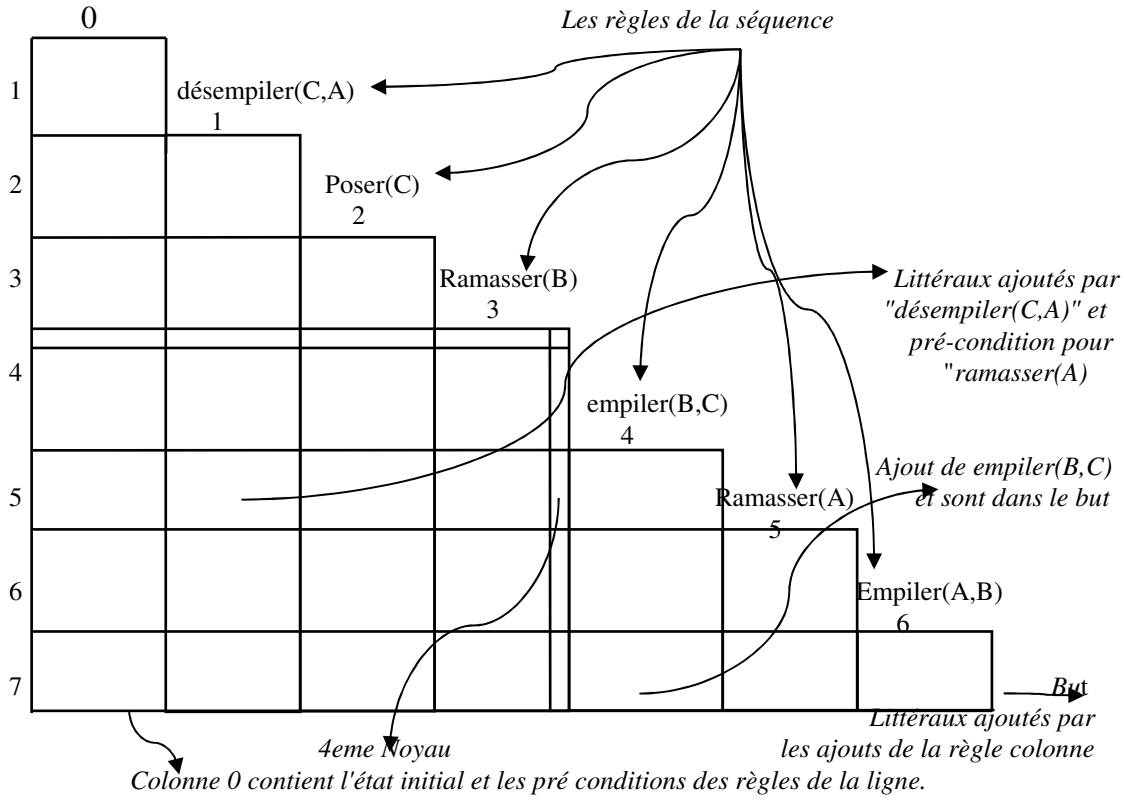


Cet état est donc un état impossible car il y a tenu(B) et tenu(A)

3. Table Triangulaire (Représentation des plans)

Il est utile d'avoir des informations supplémentaires incluses dans la spécification d'un plan trouvé; Il se peut par exemple que nous souhaiterions connaître les relations entre les règles, les pré-conditions qu'elles fournissent pour d'autres règles. Ces informations peuvent être données par une table triangulaire.

Soit N le nombre de règles du plan d'actions (pour l'exemple précédent N=6 règles)
- Les noms des colonnes 1-N représentent les règles de la séquence.
- La colonne 0 contient les littéraux de l'état initial
- La dernière ligne (N+1) contient les littéraux du but.
- Les cases (i,j) de la table pour i<N+1 et j>0 sont les littéraux ajoutés à la description d'états par la j^{eme} règle qui se retrouvent comme pré-condition de la i^{eme} règle.
- Les cases (i,0), pour i<N+1 contiennent la description de l'état initial et qui se retrouvent comme pré-condition de la i^{eme} règle.
- Les rubriques de la N+1eme ligne sont les littéraux de l'état initial et ceux ajoutés (appartiennent à la liste d'ajouts) par les règles et qui sont des composantes du But.
- Les rubriques à gauche de la i^{eme} règle sont ses pré-conditions.



Exercice : Compléter cette table triangulaire de la séquence précédente.

Définition : On définit le i^{eme} noyau comme l'intersection de toutes les lignes sous la i^{eme} ligne (i^{eme} comprise) avec toutes les colonnes à gauche de la i^{eme} colonne.

Ces rubriques du noyau correspondent aux conditions qui doivent être unifiées avec une description d'états afin que la séquence composée par la i^{eme} règle et les règles ultérieures soit applicable et atteigne le but.

Cas particuliers: Le 1^{er} noyau correspond à l'état initial, le $(N+1)^{eme}$ est le but.

Ces tables sont utilisées pour contrôler l'exécution des plans d'actions, et prendre en compte les effets imprévus. Par exemple le robot a subi un effet indésirable lors d'une action, on s'écarte alors du but: On peut produire un nouveau plan mais cette solution est coûteuse, Ou on cherche un moyen intelligent pour continuer le travail, qui est l'utilisation du noyau. On cherche alors le plus grand noyau qui filtre dans la table.

Pour trouver le noyau adéquat qui s'unifie avec une description d'état, nous vérifions chacun des noyaux en commençant par celui qui porte le numéro le plus élevé:

- Si la dernière ligne (but) filtre alors arrêt
- Si le noyau qui filtre est le i^{eme} alors on sait que la i^{eme} règle est applicable. On exécute la règle et les suivantes.
- S'il n'y a pas de noyau qui filtre, on génère un nouveau plan.