Planification

1

Plan

- Introduction à la planification
- La langage STRIPS
- Représentation des plans
- Planification partiellement ordonnée
- Graphes de panification

2

Planification

- Un agent planificateur construit des plans pour atteindre ses buts pour ensuite les exécuter.
- Il est semblable aux agents de résolution de problèmes des chapitres 3 et 4.
- Il est différent au niveau de ses représentations de buts, d'états, d'actions et de solutions.
- Il est aussi différent dans sa manière de trouver des solutions: l'usage de représentations explicites et logiques permet au processus de planification de mieux diriger la recherche.

3

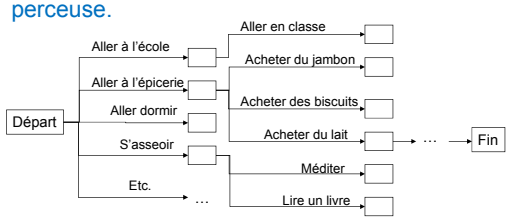
Environnement de planification

- Les environnements considérés dans ce chapitre sont:
  - Complètement observables,
  - Déterministes,
  - Statiques,
  - Discrets.

4

Limites de la résolution de problèmes

- Tâche: Aller chercher du lait, des bananes et une perceuse.



- Algorithmes de recherche: les heuristiques et les tests de buts sont inadéquats.

5

3 avantages de la planification

- 1- Ouvrir la représentation des actions et des buts afin de permettre une sélection plus fine.
  - États et buts: logique du premier ordre
  - Actions: description logique de pré-conditions et d'effets.
  - Connexion directe entre état et actions: permet de guider et contraindre la recherche au lieu de faire des expansions aveugles d'alternatives, comme *Acheter*(*Beurre*) ou *Manger*(*Pommes*).

6

3 avantages de la planification

- 2- Relâcher les contraintes pour la construction séquentielle de solutions.
  - Le processus de planification peut ajouter des actions quand elles sont requises.
  - Pas d'obligation de le faire à partir de l'état initial de manière incrémentale.
  - Réduit le facteur de branchement et le nombre de retour en arrière en prenant des décisions simples et évidentes en premier lieu.
  - Exemple : on peut décider d'acheter du lait sans savoir où en acheter, comment s'y rendre et quoi faire après. Il n'y a pas nécessairement de lien entre l'ordre de planification et l'ordre d'exécution.

7

3 avantages de la planification

- 3- Appliquer une approche *diviser-pour-régner* en définissant des sous-buts.
  - Les buts peuvent être des conjonctions: aller chercher du lait ET aller chercher des bananes ET aller chercher une perceuse.
  - On peut faire des sous-plans pour chaque sous-buts.
  - Ceci fonctionne tant que le coût pour combiner les sous-buts n'est pas trop grand.

8

STRIPS (1)

- C'est un langage de représentation de base pour les planificateurs.
- Représentation des états
  - Conjonction de littéraux positifs [$Pos = \grave{A} = At$]
  - Ex : $Pos(Avion_1, Québec) \wedge Pos(Avion_2, Montréal)$
  - Les littéraux ne peuvent pas contenir de variables ou de fonctions.
    - Ex: littéraux non permis: $Pos(x,y)$ ou $Pos(Père(Fred),Montréal)$
  - Assomption du monde clos: tout ce qui n'est pas mentionné est supposé faux.

9

STRIPS (2)

- Représentation des buts
  - Conjonction de littéraux positifs
  - Un état satisfait un but s'il contient tous les littéraux du but (et peut-être plus).
  - Ex: L'état *Riche ET Populaire ET Triste* satisfait le but *Riche ET Populaire*.

10

STRIPS (3)

- Représentation des actions
  - Une action est définie en fonction de ses préconditions et de ses effets.

$Action(Vol(a, dep, dest)$,
PRECOND: $Pos(a, dep) \wedge Avion(a) \wedge Aéroport(dep) \wedge Aéroport(dest)$
EFFETS: $\neg Pos(a, dep) \wedge Pos(a, dest)$

  - ajout des littéraux positifs : $Pos(a,dest)$
  - retrait des littéraux négatifs : $= Pos(a,dep)$

11

STRIPS (4)

- Préconditions: conjonction de littéraux sans fonctions spécifiant ce qui doit être vrai avant de pouvoir exécuter l'action. Toutes les variables dans les préconditions doivent aussi apparaître dans la liste de paramètres de l'action.
- Effets: conjonction de littéraux sans fonction représentant comment l'état change après l'exécution de l'action.

12

Fonctionnement

- Une action est applicable dans tout état qui satisfait la précondition, sinon l'action n'a aucun effet.
- Pour établir l'applicabilité, il va falloir trouver une substitution valable pour les variables dans la précondition, ex:

L'état $Pos(A_1, JFK) \wedge Pos(A_2, SFO) \wedge Avion(A_1) \wedge Avion(A_2) \wedge Aéroport(JFK) \wedge Aéroport(SFO)$

Satisfait la précondition

$Pos(a, dep) \wedge Avion(a) \wedge Aéroport(dep) \wedge Aéroport(dest)$

Avec la substitution $\{a/A_1, dep/JFK, dest/SFO\}$.
Donc, l'action $Vol(A_1, JFK, SFO)$ est applicable.

13

Fonctionnement

- L'état résultant de l'exécution d'une action applicable est identique à l'état précédent, mais avec
  - l'ajout des littéraux positifs de l'effet de l'action
  - et le retrait des littéraux négatifs de l'effet de l'action.
- L'état résultant de l'exemple est:

$Pos(A_1, SFO) \wedge Pos(A_2, SFO) \wedge Avion(A_1) \wedge Avion(A_2) \wedge Aéroport(JFK) \wedge Aéroport(SFO)$

- Assomption de STRIPS: Tous les littéraux qui ne sont pas mentionnés dans l'effet ne changent pas.

14

Exemple (1)

- Transport de fret d'avion:
  - 3 actions: ***Charger*** $(Load)$, ***Décharger*** $(Unload)$, ***Voler*** (Fly)
  - Les actions ont un effet sur 2 prédicats: $In(c,p)$ signifie que la cargaison c est dans l'avion p; $At(x,p)$ signifie que l'objet x est à l'aéroport p.
- Le plan suivant est une solution au problème:

Goal(At(C_1; JFK) ^ At(C_2; SFO))

$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO)]$

15

Exemple (2)

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Problème de transport par avion.

16

Shakey1

11.13 The original STRIPS program was designed to control Shakey the robot. Figure 11.17 shows a version of Shakey's world consisting of four rooms lined up along a corridor, where each room has a door and a light switch.

The actions in Shakey's world include moving from place to place, pushing movable objects (such as boxes), climbing onto and down from rigid objects (such as boxes), and turning light switches on and off. The robot itself was never dextrous enough to climb on a box or toggle a switch, but the STRIPS planner was capable of finding and printing out plans that were beyond the robot's abilities. Shakey's six actions are the following:

- $Go(x, y)$, which requires that Shakey be at x and that x and y are locations in the same room. By convention a door between two rooms is in both of them.
- Push a box b from location x to location y within the same room: $Push(b, x, y)$. We will need the predicate Box and constants for the boxes.

17

Shakey(2)

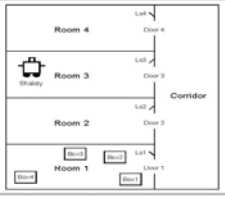


Figure 11.17 Shakey's world. Shakey can move between landmarks within a room, can pass through the door between rooms, can climb climbable objects and push pushable objects, and can flip light switches.

- Climb onto a box: $ClimbUp(b)$; climb down from a box: $ClimbDown(b)$. We will need the predicate On and the constant $Floor$.
- Turn a light switch on: $TurnOn(s)$; turn it off: $TurnOff(s)$. To turn a light on or off, Shakey must be on top of a box at the light switch's location.

Describe Shakey's six actions and the initial state from Figure 11.17 in STRIPS notation. Construct a plan for Shakey to get $Box2$ into $Room2$.

Solution exercice

Action(Go(x, y),
PRECOND : At(Shakey, x) ∧ In(x, r) ∧ In(y, r),
EFFECT: At(Shakey, y) ∧ ¬(at(Shakey, x)))

Action(Push(b, x, y),
PRECOND: At(Shakey, x) ∧ At(b, x) ∧ Pushable(b) ∧ In(x, r) ∧ In(y, r),
EFFECT: At(b, y) ∧ At(Shakey, y) ∧ ¬At(b, x) ∧ ¬At(Shakey, x))

Action(ClimbUp(b),
PRECOND: At(Shakey, x) ∧ At(b, x) ∧ Climbable(b),
EFFECT: On(Shakey, b) ∧ ¬On(Shakey, Floor))

19

Solution exercice

Action(ClimbDown(b),
PRECOND: On(Shakey, b),
EFFECT: On(Shakey, Floor) ∧ ¬On(Shakey, b))

Action(TurnOn(l),
PRECOND: On(Shakey, b) ∧ At(Shakey, x) ∧ At(l, x),
EFFECT: TurnedOn(l))

Action(TurnOff(l),
PRECOND: On(Shakey, b) ∧ At(Shakey, x) ∧ At(l, x),
EFFECT: ¬TurnedOn(l))

20

Heuristiques pour la recherche dans un espace d'états (2)

- Supposition de l'indépendance des sous-buts
  - Optimiste : s'il y a des interactions négatives. Par ex: une action dans un sous plan peut supprimer un but atteint par un autre sous-plan.
  - Pessimiste: s'il y a des actions redondantes dans les sous-plans: par ex, 2 actions qui peuvent être remplacées par une seule dans le plan final.

21

Heuristiques pour la recherche dans un espace d'états

- « Simplifier » le problème
  - Enlever toutes les préconditions: Toute action est alors exécutable
  - Enlever les effets négatifs: Si une action a pour effet *A* — =*B* dans le problème d'origine alors elle aura pour effet *A* dans le problème relaxé.
    - Donc il est inutile de se soucier des effets négatifs entre sous-plans, car aucune action ne peut supprimer les actions accomplies par une autre action.

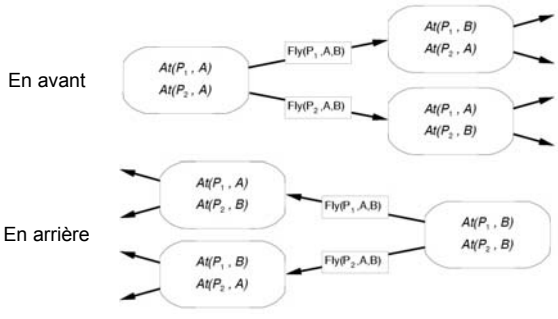
22

Espaces d'états

- Les nœuds sont des états du monde et le chemin est le plan.
  - Recherche progressive (en avant): procède de l'état initial vers l'état final en utilisant les techniques de recherche. Le facteur de branchement est élevé.
  - Recherche régressive (en arrière): procède de l'état final vers l'état initial. Limite le facteur de branchement, car habituellement le but a peu d'opérateurs applicables.
    - Il y a des complications s'il y a des conjonctions de buts.

23

Recherche en avant et en arrière



24

Espaces de plans

- Les nœuds sont des plans partiels, les opérateurs s'appliquent sur des plans.
  - Le chemin pour obtenir un plan n'est pas important.
  - On démarre avec un plan incomplet et on applique des opérateurs afin d'obtenir un plan complet et consistant.
  - Opérateurs :
    - Ajouter un lien d'une action vers une précondition non résolue.
    - Ajouter une étape pour résoudre une précondition non résolue.
    - Ordonner une étape par rapport à une autre.

25

Représentation de plans

- Engagement minimum (least commitment): s'engager uniquement sur les aspects pertinents des actions, en laissant les autres aspects à plus tard.
- Ordre partiel: le planificateur représente un ordre entre certaines étapes.
- Ordre total: le planificateur représente un ordre entre toutes les étapes: linéarisation du plan.

26

Composantes d'un plan

- Un ensemble d'actions constituant les étapes du plan.
  - Un plan vide contient uniquement les actions *Départ* et *Fin*.
- Un ensemble de contraintes d'ordonnancement. Chaque contrainte spécifie qu'une action doit survenir avant une autre.
  - Ex: $A \prec B$ A doit être exécutée avant B :

27

Composantes d'un plan

- Un ensemble de liens causaux.
  - Un lien causal entre deux actions *A* et *B* est noté:

$$A \xrightarrow{p} B$$

  - Il signifie que A accomplit p pour B.
  - Le lien causal est un intervalle de protection, il interdit de placer une autre action C, entre A et B, qui a comme effet $\neg p$.
- Un ensemble de préconditions non résolues.
  - Le but du planificateur est de réduire cet ensemble à l'ensemble vide.

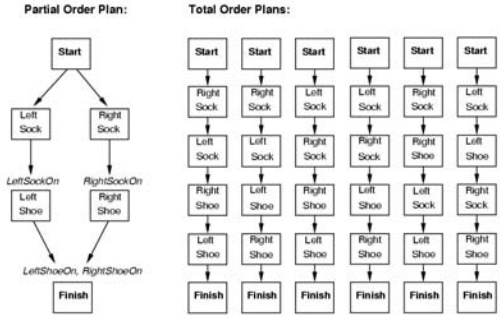
28

Solution

- Une solution est un plan complet et consistant que l'agent peut exécuter et qui garantit l'accomplissement du but.
  - Complet : toutes les préconditions sont satisfaites par des actions précédentes et elles ne sont pas annulées par d'autres actions.
  - Consistant : il n'y a pas de contradictions entre les contraintes d'ordonnancement des actions et pas de conflits entre les liens causaux.

29

Exemple



30

Exemple

- Les composantes du plan:

$$\text{Actions:}\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$$
$$\text{Orderings:}\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$$
$$\text{Links:}\{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe,$$
$$RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}$$
$$\text{Open Preconditions:}\{\}$$

- Il manque les contraintes d'ordonnancement pour les actions *Start* et *Finish*.

$$Start \prec Finish$$

31

Algorithme POP

- Le plan initial contient:
  - Les actions $Départ$ et Fin,
  - La contrainte d'ordonnancement $Départ \prec Fin$
  - Aucun lien causal
  - Les préconditions non résolues sont toutes les préconditions de *Fin*.
- Le test de but retourne que le plan est une solution si l'ensemble des préconditions non résolues est vide.

32

Algorithme POP

- La fonction successeur choisit arbitrairement une précondition non résolue p d'une action B et génère tous les plans consistants où une action A permet d'accomplir p.
  - Le lien causal et la contrainte d'ordonnancement suivants sont ajoutés au plan:

$$A \xrightarrow{p} B \quad A \prec B$$

  - Si *A* est une nouvelle action, on ajoute les contraintes:

$$Départ \prec A \text{ et } A \prec Fin$$

  - On résout les conflits entre le nouveau lien causal et toutes les actions existantes et entre l'action A (si c'est une nouvelle action) et tous les liens causaux existants.
    - On résout les conflits en ajoutant des contraintes.

33

Exemple : le pneu de secours

- But est d'avoir une roue de secours correctement gonflé sur l'essieu, alors que l'état initial était une roue à plat sur l'essieu et une roue gonflée dans le coffre.
- 4 actions : ôter la roue du coffre ($Remove(Spare, Trunck)$); ôter la roue à plat de l'essieu($Remove(Flat, Axle)$, placer la roue de secours sur l'essieu($MoveOn(Spare, Axle)$ et laisser la voiture sans surveillance toute la nuit($LeaveOverNight$). On suppose que la voiture est tombée en panne dans un lieu malfamé.

34

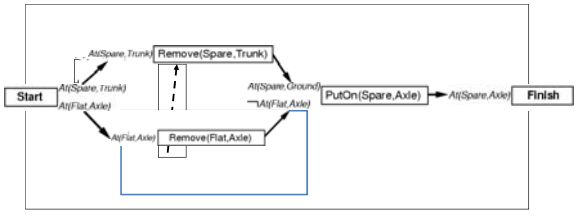
Exemple

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
PRECOND: $At(Spare, Trunk)$
EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
PRECOND: $At(Flat, Axle)$
EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

Mettre un pneu de secours.

35

Exemple



36

POP avec logique du premier ordre

- Exemple, si on a $On(A,B)$ comme précondition non résolue et l'action suivante (« $move\ b\ de\ x\ à\ y$ »):

$Action(Move(b,x,y),$
PRECOND:$On(b,x) \wedge Clear(b) \wedge Clear(y),$
EFFECT:$On(b,y) \wedge Clear(x) \wedge \neg On(b,x) \wedge \neg Clear(y))$.

- On peut unifier $On(A,B)$ avec $On(b,y)$, ce qui donne:

$Action(Move(A,x,B),$
PRECOND:$On(A,x) \wedge Clear(A) \wedge Clear(B),$
EFFECT:$On(A,B) \wedge Clear(x) \wedge \neg On(A,x) \wedge \neg Clear(B))$.

37

POP avec logique du premier ordre

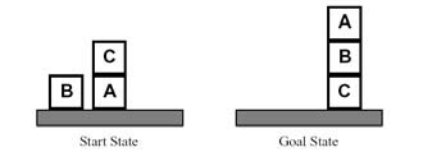
- La présence de variables dans les actions complexifie la détection et la résolution des conflits.
  - Par exemple, lorsque l'on ajoute l'action $Move(A,x,B)$ dans le plan, on doit ajouter le lien causal suivant:

$$Move(A,x,B) \overset{On(A,B)}{\rightarrow} Finish$$

  - S'il y a une autre action M_2 qui a comme effet $\neg On(A,z)$, alors M_2 est en conflit seulement si $z = B$.
  - Pour résoudre cela, on ajoute un ensemble de contraintes d'inégalités de la forme $x \neq z$, où x peut être une constante ou une variable.

38

Exercice



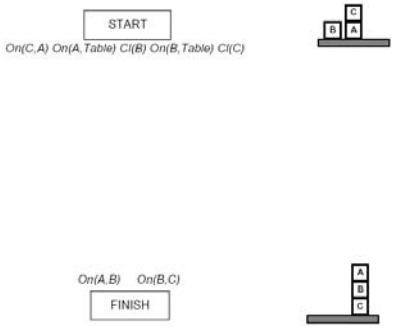
Clear(x) On(x,z) Clear(y)
PutOn(x,y)
~On(x,z) ~Clear(y) Clear(z) On(x,y)

Clear(x) On(x,z)
PutOnTable(x)
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

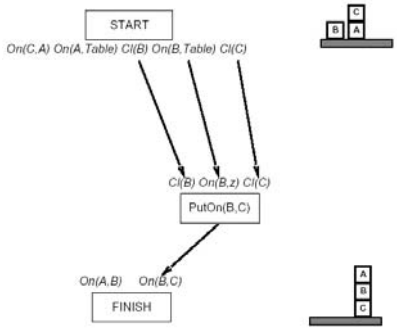
39

Exercice



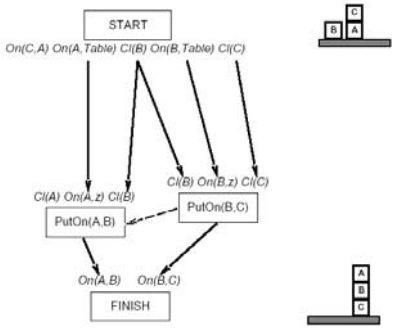
40

Exercice



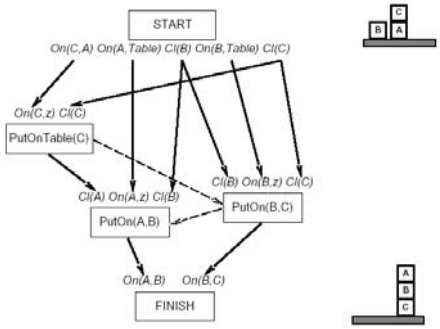
41

Exercice



42

Exercice



43

Heuristique pour la POP

- L'algorithme de planification doit choisir une précondition non résolue et tenter de la satisfaire.
- Pour choisir cette précondition, il peut utiliser une heuristique semblable à celle de la variable la moins contrainte pour les CSP.
  - C'est-à-dire, choisir la précondition qui peut être satisfaite par le plus petit nombre d'actions.
    - Si une précondition ouverte n'est satisfaite par aucune action, alors on va la choisir et l'algorithme va se rendre compte d'une impossibilité plus rapidement.
    - Si une précondition ouverte est satisfaite par juste une action, en ajoutant celle-ci, on peut ajouter des contraintes qui vont nous aider plus tard.

44

Graphe de planification

- Un graphe de planification est une structure de donnée permettant d'obtenir des heuristiques plus efficaces.
- Ces heuristiques peuvent être appliquées à n'importe quelle méthode de recherche vue jusqu'à maintenant.
- On peut même extraire le plan directement à partir du graphe de planification.

45

Graphe de planification

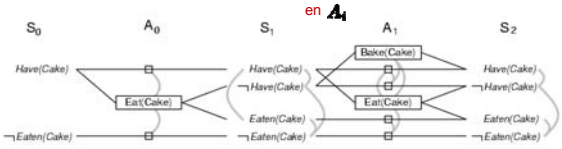
- Un graphe de planification consiste en une séquence de niveaux correspondant à des étapes dans le temps, où le niveau 0 est l'état initial.
- Chaque niveau contient:
  - Tous les littéraux pouvant être vrai à un temps donné
  - Toutes les actions qui pourraient avoir leurs préconditions satisfaites à un temps donné.
- Les graphes de planification ne fonctionnent que pour des problèmes de planification en logique propositionnelle.

46

Exemple

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake)$

Les rectangles représentent les actions et les petits carrés indiquent les actions persistantes. Les lignes droites représentent les préconditions et les effets. Les lignes grises incurvées indiquent les exclusions mutuelles « Mutex ». Si 2 littéraux sont mutex en S_i alors les actions persistantes sont mutex en A_i



47

Condition d'arrêt

- On arrête la construction du graphe lorsque l'on obtient deux niveaux identiques.
- À la fin on obtient une structure où:
  - Chaque niveau A_i contient toutes les actions qui peuvent être appliquées en S_i, avec des contraintes indiquant quelles paires d'actions ne peuvent pas être exécutées en même temps.
  - Chaque niveau S_i contient tous les littéraux qui peuvent être le résultat d'une action en A_{i-1}, avec des contraintes indiquant les paires impossibles.
- Un littéral qui n'apparaît pas au niveau final du graphe, ne peut pas être accompli par aucun plan.

48

Liens « mutex »

- Mutex signifie mutuellement exclusif.
- Il y a un lien mutex entre deux actions si:
  - Effets inconsistants: une action rend faux un effet d'une autre.
  - Interférence: Un des effets d'une action est la négation d'une précondition d'une autre action.
  - Compétition: Une précondition d'une action est mutuellement exclusive avec une précondition d'une autre action.

49

Littéral (from Wikipedia)

- En logique mathématique, un **littéral** est un atome (aussi appelé **littéral positif**) ou la négation d'un atome (aussi appelé **littéral négatif**).
- En logique propositionnelle, une variable *P* est un littéral, de même que sa négation ¬*P* ;

50

Liens « mutex »

- Il y a un lien mutex entre deux littéraux si au même niveau,
  - un littéral est la négation d'un autre, ou si
  - toutes les paires d'actions qui peuvent avoir généré les deux littéraux sont mutuellement exclusives.

51

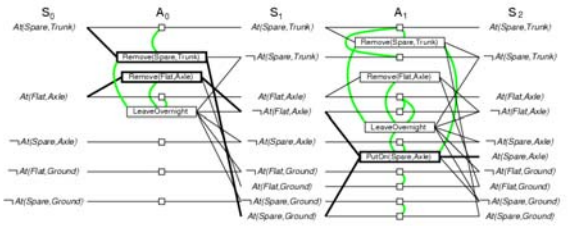
GRAPHPLAN

- GRAPHPLAN est un algorithme pour extraire un plan à partir du graphe de planification.

```
function GRAPHPLAN(problem) returns solution or failure
  graph ← INITIAL-PLANNING-GRAPH(problem)
  goals ← GOALS[problem]
  loop do
    if goals all non-mutex in last level of graph then do
      solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
      if solution ≠ failure then return solution
      else if NO-SOLUTION-POSSIBLE(graph) then return failure
    graph ← EXPAND-GRAPH(graph, problem)
  end
```

52

Exemple GRAPHPLAN



Problème du pneu de secours, le but est $At(Spare, Axle)$.

53

Exemple GRAPHPLAN

- Une fois arrivée en S_2, le but est présent, donc on essaie d'extraire la solution.
  - On recherche par en arrière, avec comme état initial le niveau S_n.
  - À chaque niveau S_i, on choisit toutes les actions sans conflits dans A_{i-1} qui accomplissent les buts.
    - Sans conflits veut dire qu'il n'y a pas de liens mutex entre les actions choisies ou entre leurs préconditions.
    - Le niveau S_{i-1} a comme buts les préconditions des actions choisies en A_{i-1}.
  - Le but est d'atteindre l'état S_0 avec tous les buts accomplis.

54