# IFT608 / IFT 702
# Planification en intelligence artificielle

## Contrôle de la recherche
## avec des réseaux de tâches hiérarchiques

Froduald Kabanza
Département d'informatique
Université de Sherbrooke

# Sujets

- Introduction
- *Total-Order STN Planning (*Total-Order Forward decomposition - TFD)
- *Partial-Order STN Planning (*Partial-Order Forward decomposition - PFD)

Malik Ghallab, Dana Nau & Paolo Traverso. *Automated Planning and Acting*. http://projects.laas.fr/planning/book.pdf

Malik Ghallab, Dana Nau & Paolo Traverso. *Automated Planning: Theory and Practice.*

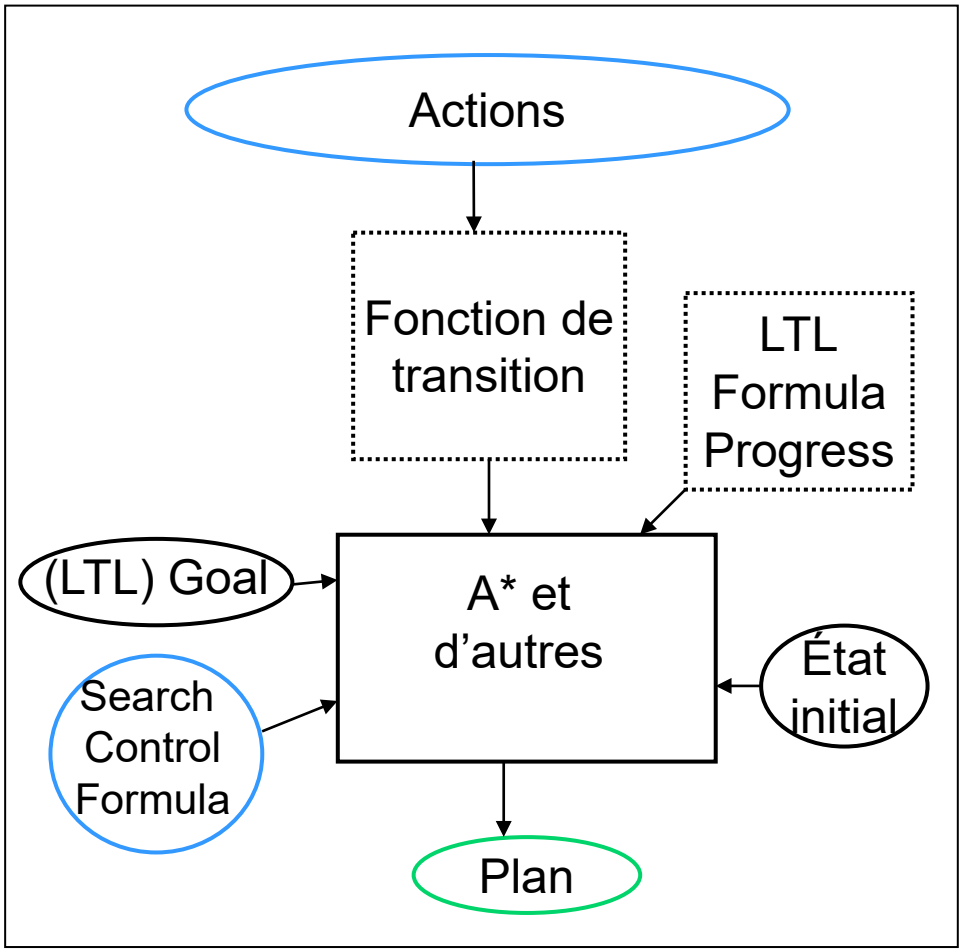
# Introduction

- Principe: spécifier des recettes pour contrôler l'exploration de l'espace des solutions pour un planificateur.

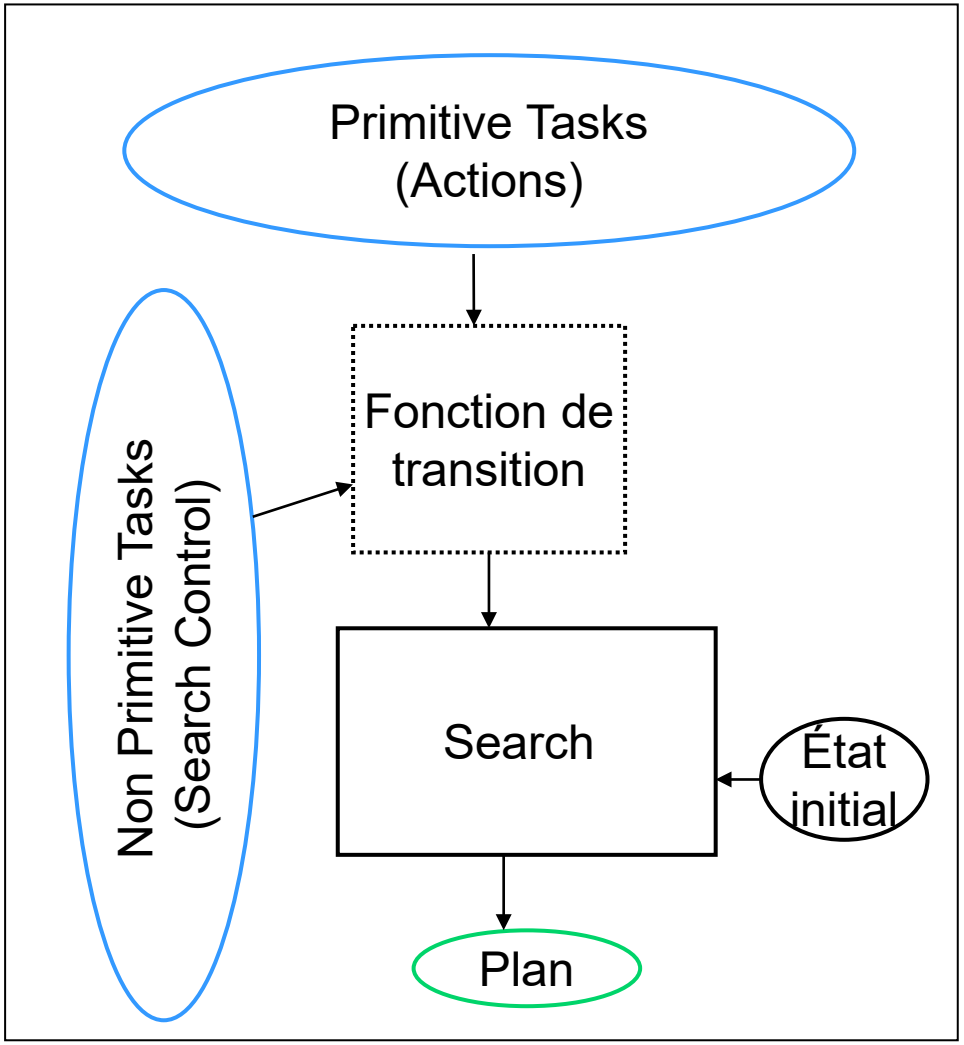
  - » Spécifier les chemins à éviter: approche TLPLAN:
    - *TLPLAN: Temporal Logic Planner*

  - » Spécifier les sous-buts (sous tâches) à suivre: approche HTN:
    - *HTN: Hierarchical Task Network (HTN)*

# TLPLAN


Actions
Fonction de transition
LTL Formula Progress
(LTL) Goal
A* et d'autres
État initial
Search Control Formula
Plan

# HTN


Primitive Tasks
(Actions)
Non Primitive Tasks
(Search Control)
Fonction de
transition
Search
État
initial
Plan

# Vue d'ensemble de l'approche HTN

**Éntrées**: état initial, réseau de tâches initial (à effectuer), opérateurs, méthodes

**Procédure**:

- Décomposer les tâches récursivement, jusqu'à trouver des tâches primitives directement exécutables par les opérateurs
- Rebrousser chemin (*backtracking*) et essayer d'autres décompositions si nécessaire
  - » Ex., si aucune méthode n'est applicable, ou pas moyen de satisfaire une contrainte

**Observations :**

- Ressemble à la réduction de problèmes propre à la programmation dynamique (mais avec du backtracking si ça échoue)
- Effectuer un ensemble de tâches plutôt qu'atteindre un « but » comme en planification classique

# Élements de base d'un HTN

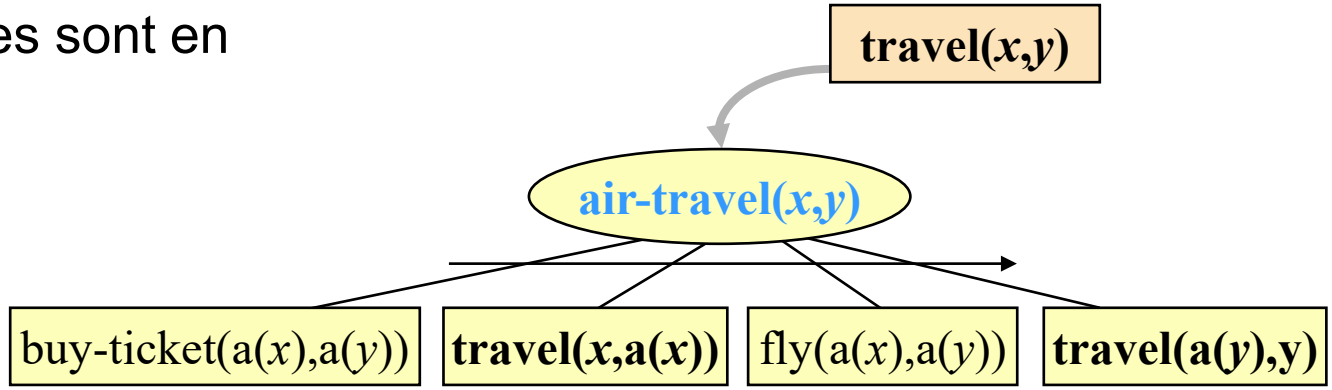
- *États, opérateurs*: comme dans les approches précédentes.

- *Tâches (non primitives)*: une tâche représente une activité complexe décomposable en sous-activités.
  - » Elle a une *règle (méthode)* de décomposition de tâches associée.

- *Tâches Primitives* : une tâche primitive représente une activité non décomposable
  - » C'est une action
  - » Elle a des préconditions et des effets comme dans PDDL

# Exemple

Les tâches non primitives sont en gras




1) **air-travel(*x*,*y*)**

*Task:* **travel(*x*,*y*)**

*Pre*: long-distance(*x*,*y*)

*Subtasks*:

buy-ticket(a(*x*),a(*y*)), **travel(*x*,a(*x*))**, fly(a(*x*),a(*x*)), **travel(a(*x*),*y*)**

2) **taxi-travel(*x*,*y*)**

*Task:* **travel(*x*,*y*)**

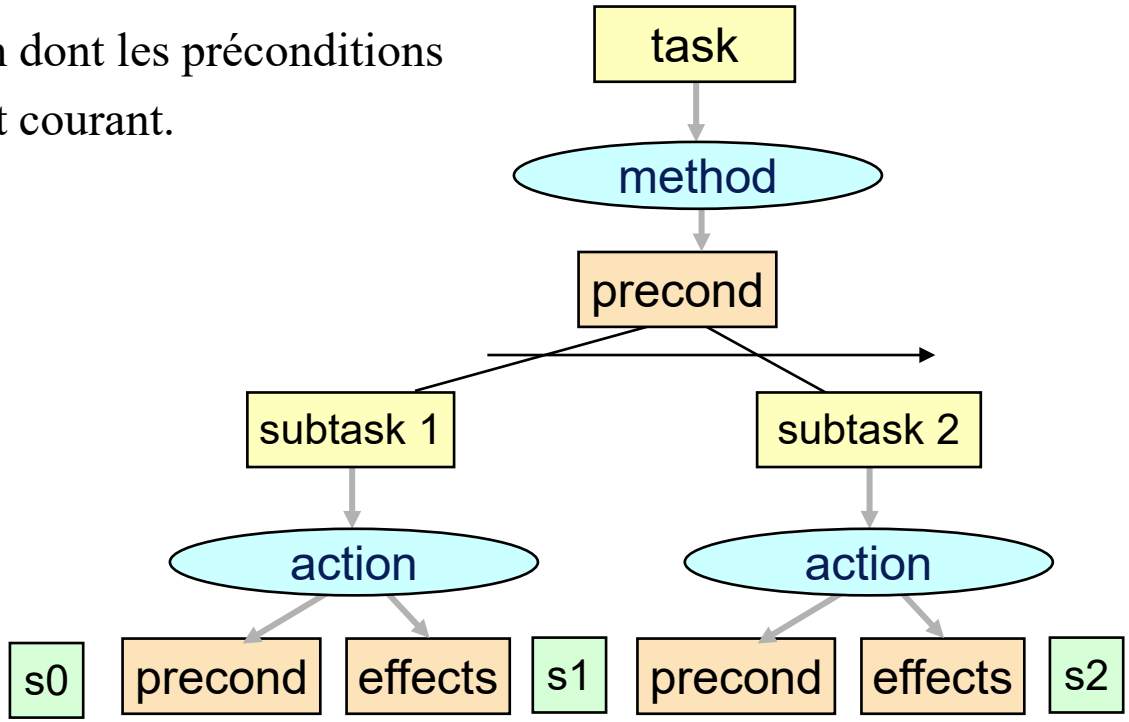
*Pre*: short-distance(*x*,*y*)

*Subtasks*:

get-taxi, ride(*x*,*y*), pay-driver

# Formes des problèmes/solutions

- *Définition du domaine* : méthodes & actions
- *Problème* : état initial, *(un but)*, liste de tâches, et la définition du domaine
- *Solution*: plan *(satisfaisant le but)* obtenu en
  - » Applicant récursivement les méthodes aux tâches non primitives (en respectant les préconditions)
  - » Applicant une action dont les préconditions satisfaites dans l'état courant.

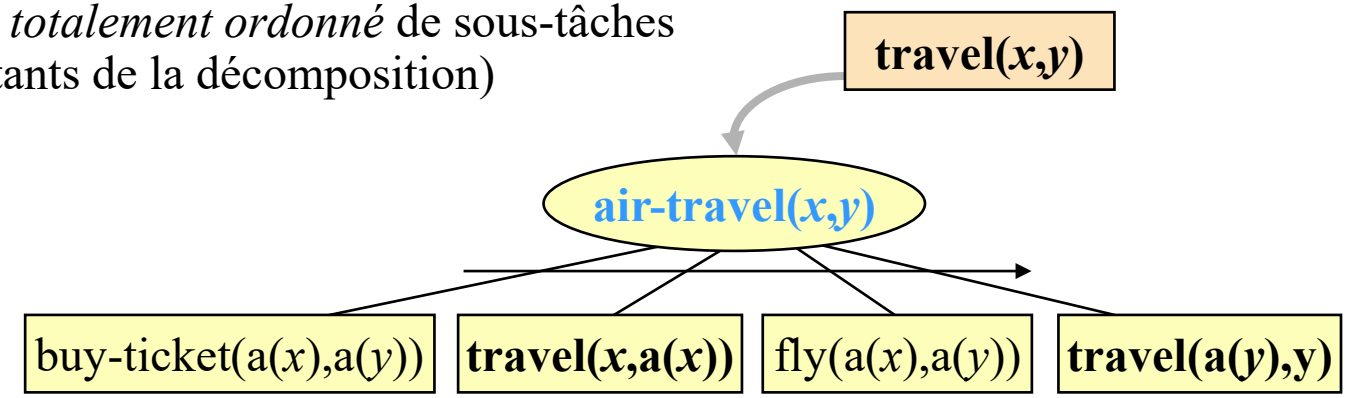


# Plan

- Introduction
- *Total-Order STN Planning*
- *Partial-Order STN Planning*

# Total-Order STN Planning

- Une méthode pour décomposer une tâche non primitive est décrite par
  - » La tâche à décomposer
  - » Une *précondition* (ensemble de littéraux: condition pour que la décomposition soit applicable)
  - » Un *ensemble totalement ordonné* de sous-tâches (tâches résultants de la décomposition)




1) **air-travel(*x*,*y*)**

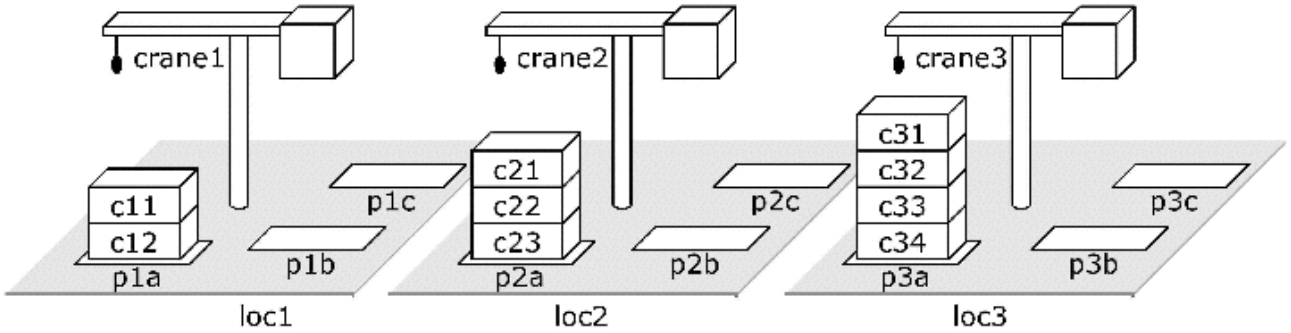
*Task:* **travel(*x*,*y*)**

*Pre*: long-distance(*x*,*y*)

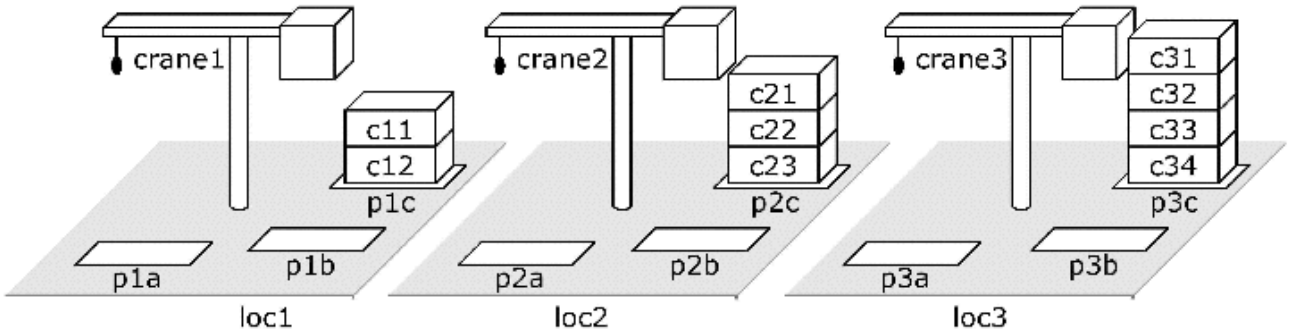
*Subtasks*: buy-ticket(a(*x*),a(*y*)), **travel(*x*,a(*x*))**,
fly(a(*x*),a(*x*)), **travel(a(*x*),*y*)**

# Exemple (Ghallab et al., 2004)

- Supposons qu'on veut déplacer trois piles de conteneurs en respectant l'ordre d'empilement  pour chaque pile

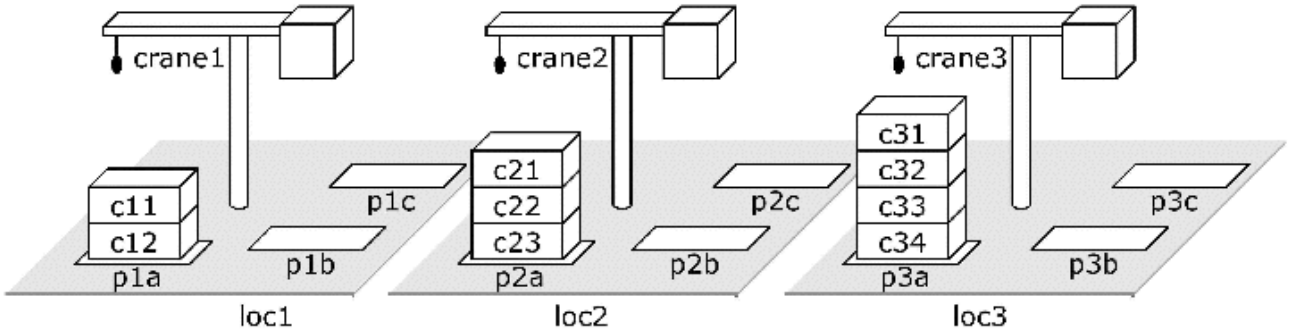



(a) initial state

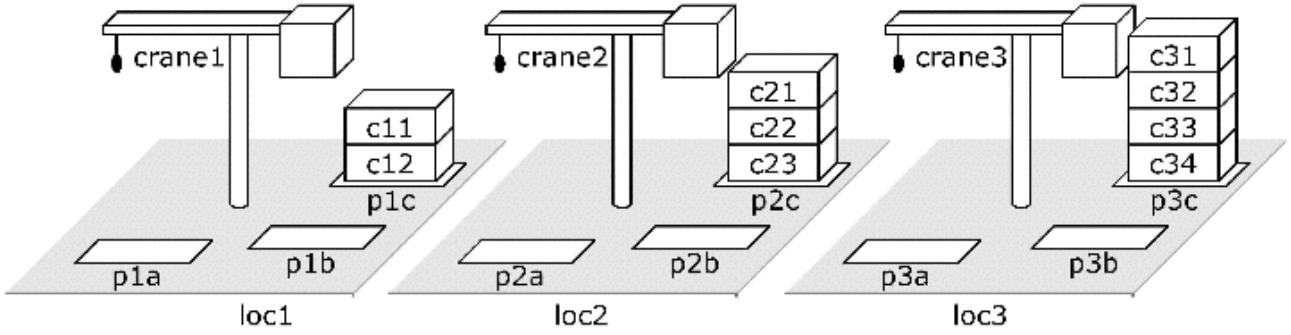



(b) goal

- Une stratégie pour déplacer les containeurs d'une pile *p* à une pile *r* :
  - » déplacer les containeurs de la pile *p* vers une pile intermédiaire *r* (inversée)
  - » ensuite les déplacer de la pile *r* à la pile *q*




(a) initial state




(b) goal

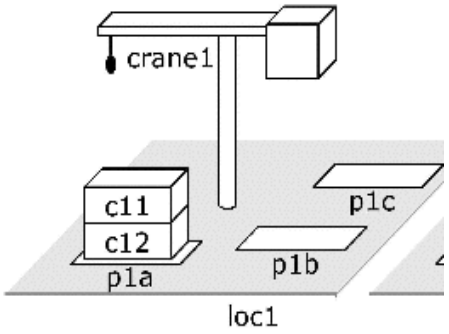
take-and-put($c, k, l_1, l_2, p_1, p_2, x_1, x_2$):
  task: move-topmost-container($p_1, p_2$)
  precond: top($c, p_1$), on($c, x_1$), *; true if $p_1$ is not empty*
    attached($p_1, l_1$), belong($k, l_1$), *; bind $l_1$ and $k$*
    attached($p_2, l_2$), top($x_2, p_2$) *; bind $l_2$ and $x_2$*
  subtasks: ⟨take($k, l_1, c, x_1, p_1$), put($k, l_2, c, x_2, p_2$)⟩

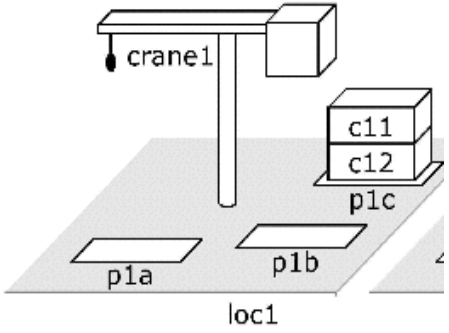
recursive-move($p, q, c, x$):
  task: move-stack($p, q$)
  precond: top($c, p$), on($c, x$) *; true if $p$ is not empty*
  subtasks: ⟨move-topmost-container($p, q$), move-stack($p, q$)⟩
    *;; the second subtask recursively moves the rest of the stack*

do-nothing($p, q$)
  task: move-stack($p, q$)
  precond: top($pallet, p$) *; true if $p$ is empty*
  subtasks: ⟨⟩ *; no subtasks, because we are done*

move-each-twice()
  task: move-all-stacks()
  precond: *; no preconditions*
  subtasks: *; move each stack twice:*
    ⟨move-stack(p1a,p1b), move-stack(p1b,p1c),
    move-stack(p2a,p2b), move-stack(p2b,p2c),
    move-stack(p3a,p3b), move-stack(p3b,p3c)⟩

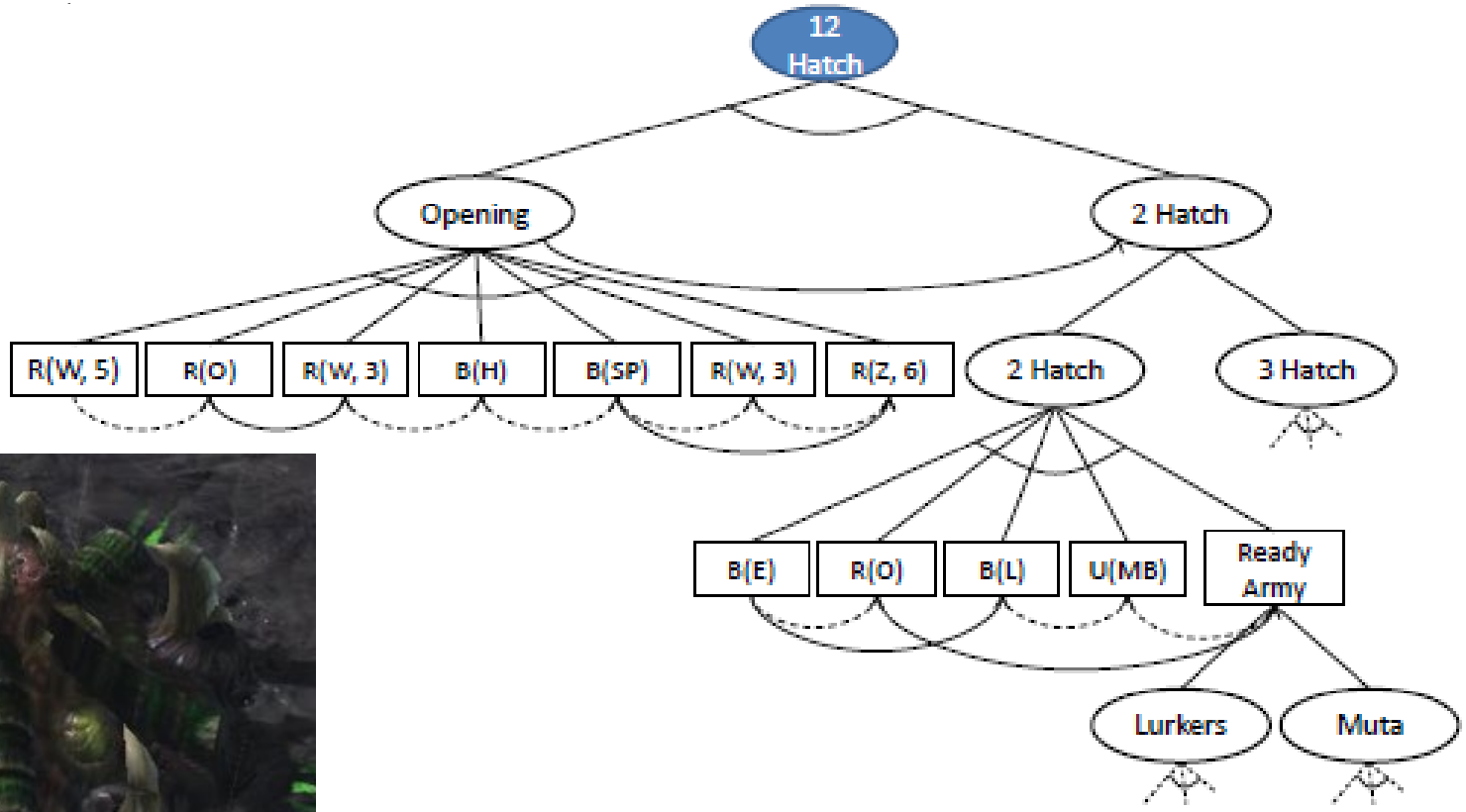
## Spécification avec ordre total



# Exemple – StrarCraft Macromanagement

- http://planiart.usherbrooke.ca/kabanza/publications/10/pair10-opponent.pdf
  Opponent Behaviour Recognition for Real-Time Strategy Games
  Kabanza

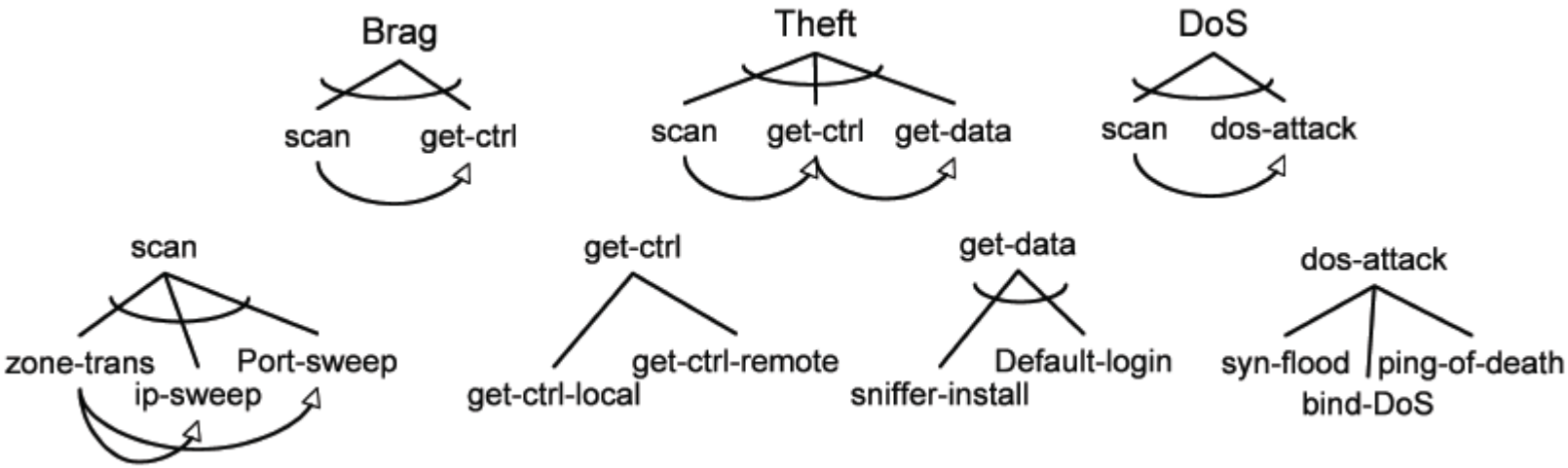


# Exemple – Network Hackers

- http://www.sciencedirect.com/science/article/pii/S0004370209000459

  A probabilistic plan recognition algorithm based on plan tree grammars

  Geib & Goldman, Artificial Intelligence, 2009.



# Algorithme : Total-Order Forward decomposition (p. 239)

TFD$(s, \langle t_1, \ldots, t_k \rangle, O, M)$
  if $k = 0$ then return $\langle\rangle$ (i.e., the empty plan)
  if $t_1$ is primitive then
    $active \leftarrow \{(a, \sigma) \mid a$ is a ground instance of an operator in $O$,
      $\sigma$ is a substitution such that $a$ is relevant for $\sigma(t_1)$,
      and $a$ is applicable to $s\}$
    if $active = \emptyset$ then return failure
    nondeterministically choose any $(a, \sigma) \in active$
    $\pi \leftarrow$ TFD$(\gamma(s, a), \sigma(\langle t_2, \ldots, t_k \rangle), O, M)$
    if $\pi =$ failure then return failure
    else return $a.\pi$
  else if $t_1$ is nonprimitive then
    $active \leftarrow \{m \mid m$ is a ground instance of a method in $M$,
      $\sigma$ is a substitution such that $m$ is relevant for $\sigma(t_1)$,
      and $m$ is applicable to $s\}$
    if $active = \emptyset$ then return failure
    nondeterministically choose any $(m, \sigma) \in active$
    $w \leftarrow \text{subtasks}(m).\sigma(\langle t_2, \ldots, t_k \rangle)$
    return TFD$(s, w, O, M)$

state $s$; task list T=( $\mathbf{t_1}$ ,$t_2$,…)

action $a$

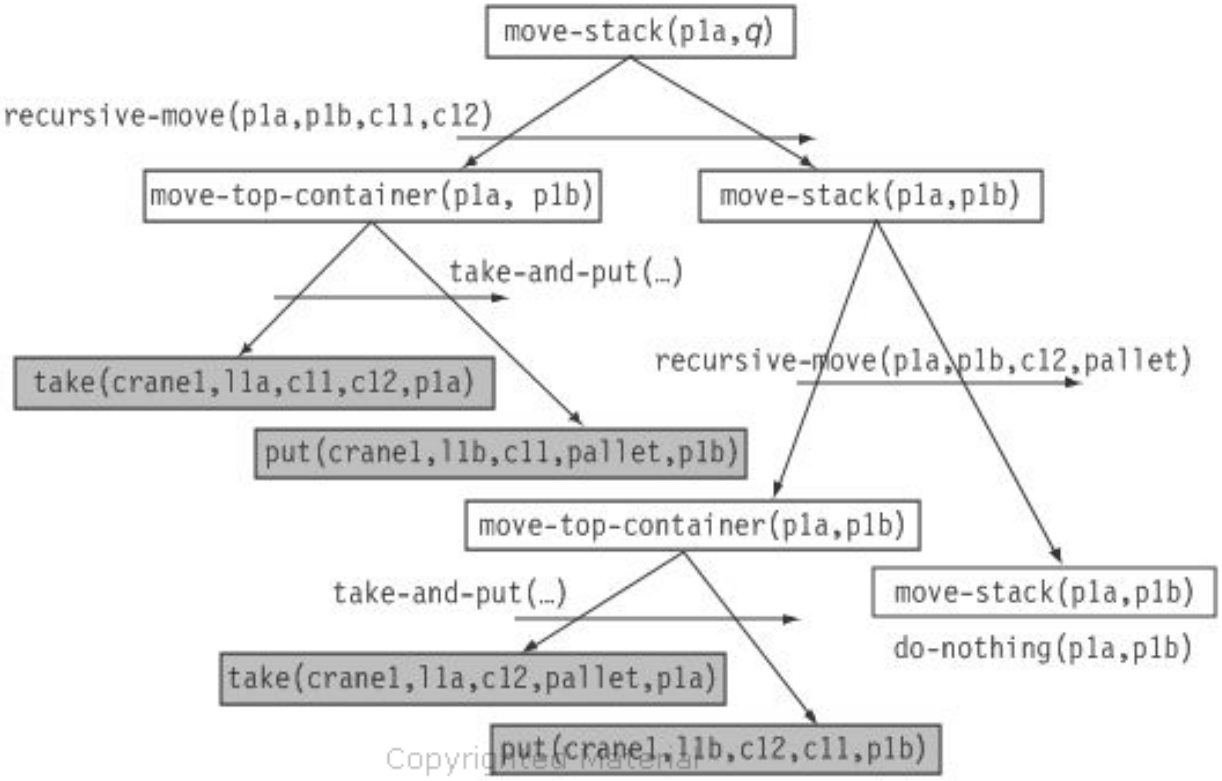
state $\gamma(s,a)$ ; task list T=($t_2$, …)

task list T=( $\mathbf{t_1}$ ,$t_2$,…)

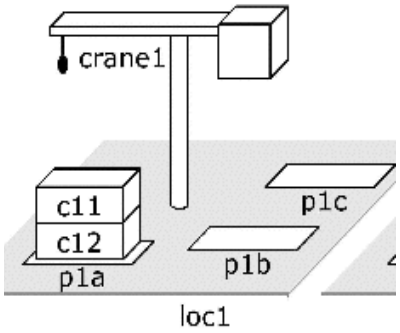
method instance $m$

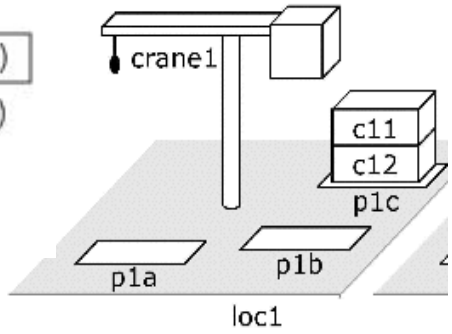
task list T=( $\mathbf{u_1,\ldots,u_k}$ ,$t_2$,…)

# Plan obtenu par TFD





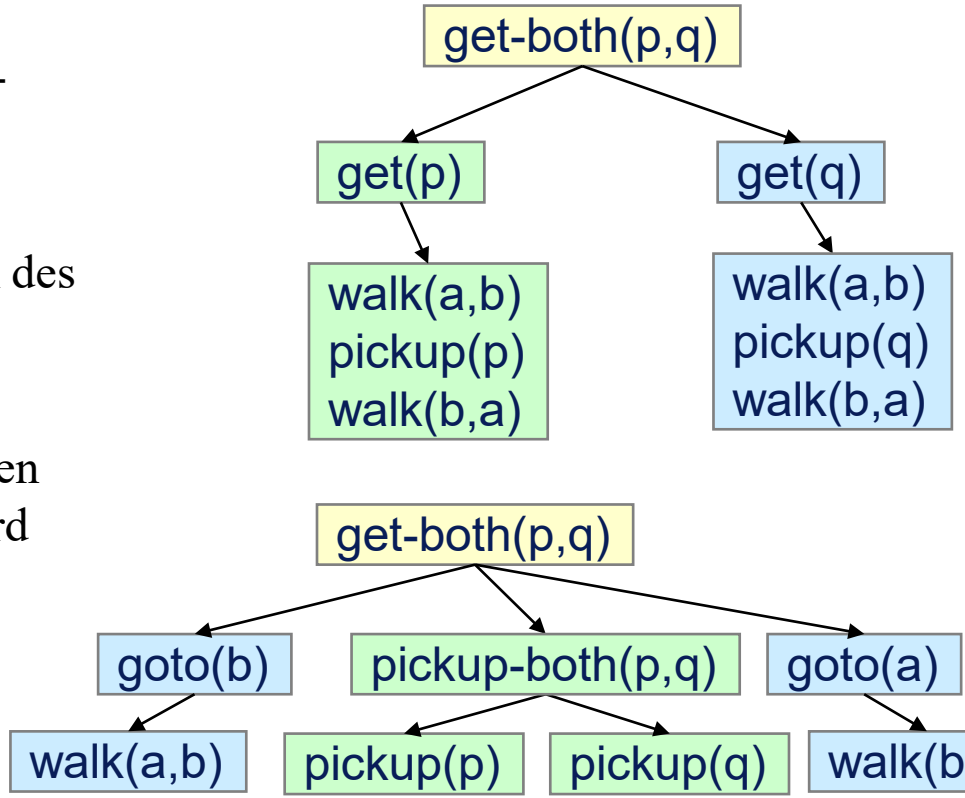



# Limites des tâches complètement ordonnées

- On ne peut pas entrelacer des sous-tâches de tâches différentes
  - » Inefficacité du planificateur
  - » Complexité de la spécification des domaines

- Théoriquement on peut toujours s'en sortir (puisque Total-Order-Forward est *Turing-complete*), mais au prix d'une spécification
  - » Peu naturelle
    - On veut des méthodes qui procèdent globalement plutôt que localement
  - » Peu efficace

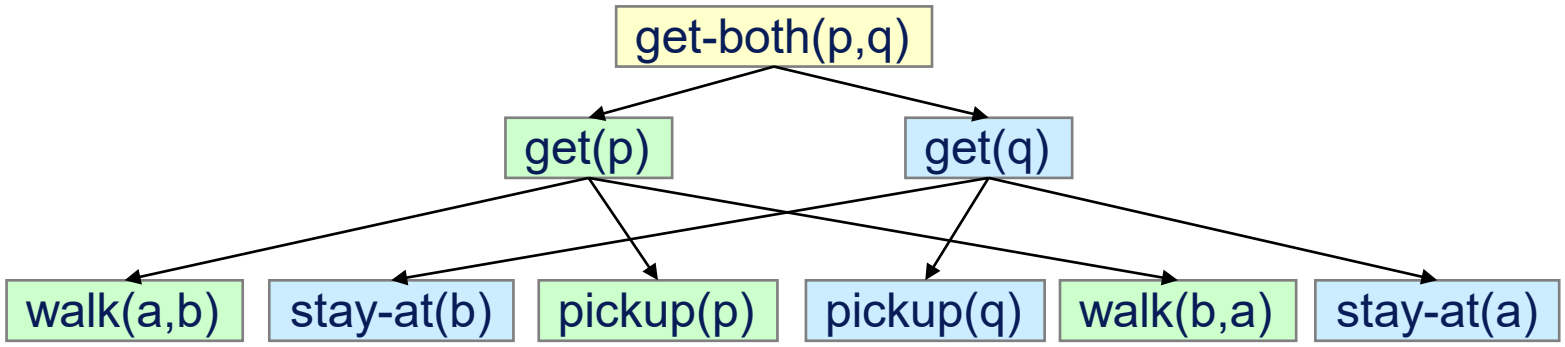


# Plan

- Introduction
- *Simple Task Network (STN) Planning*
- *Total-Order STN Planning*
- *Partial-Order STN Planning*

# Partial-Order STN Planning

- "Partial-Order STN Planning" généralise "Total-Order STN Planning"
  - » Les sous-tâches ne sont pas totalement ordonnées
  - » Au contraire, elles sont partiellement ordonnées
  - » Par conséquent, on peut  entrelacer des sous-tâches de tâches différentes

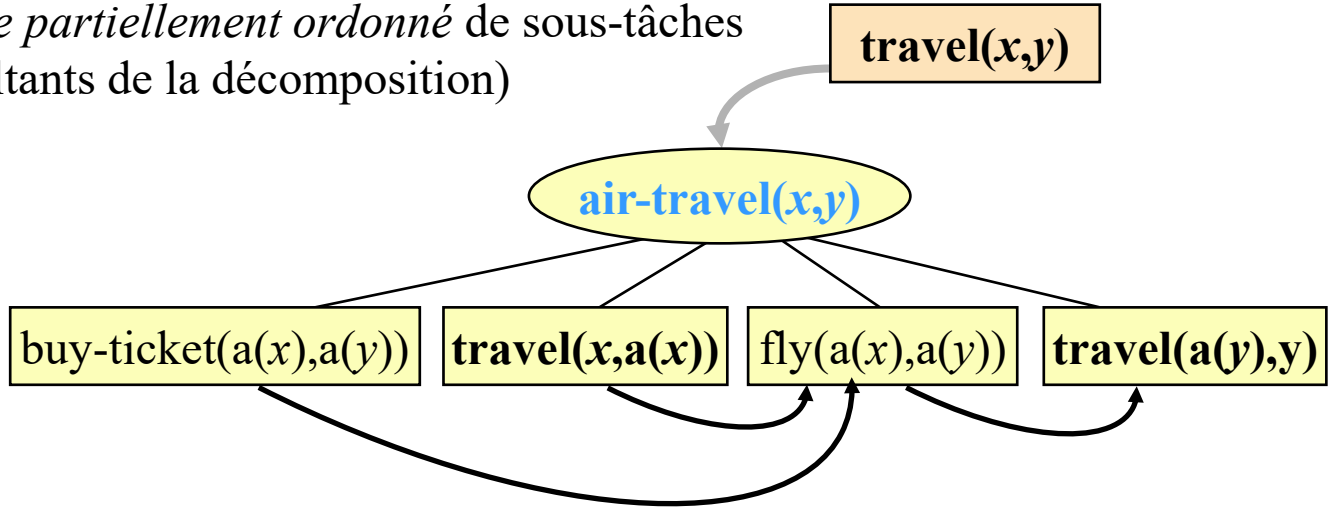



- L'algorithme de planification devient plus complexe
  - » Exemple: réparer les interactions (*add/delete* des effets) entre actions

- Certaines approches utilisent les lien causaux vue dans l'approche PSP (Planification par recherche dans l'espace de plans)

# Partial-Order STN Planning

- Une méthode est décrite par
  - » Une tâche non primitive (la tâche à décomposer par la méthode)
  - » Une *précondition* (ensemble de littéraux: condition pour que la décomposition soit applicable)
  - » Un *ensemble partiellement ordonné* de sous-tâches (tâches résultants de la décomposition)




1) **air-travel(*x*,*y*)**

*Task:* **travel(*x*,*y*)**

*Pre*: long-distance(*x*,*y*)

*Subtasks*: $u_1$=buy-ticket(a(*x*),a(*y*)), $u_2$=**travel(*x*,a(*x*))**, $u_3$=fly(a(*x*),a(*x*)),
$u_4$=**travel(a(*x*),*y*)**, $\{(u_1,u_3), (u_2,u_3), (u_3,u_4)\}$

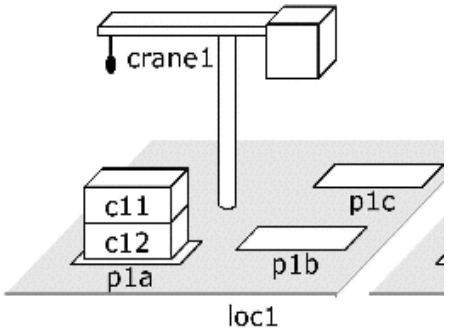
## Spécification avec ordre partiel

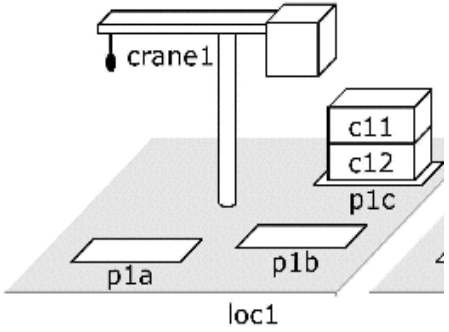
take-and-put($c, k, l_1, l_2, p_1, p_2, x_1, x_2$):
task: move-topmost-container($p_1, p_2$)
precond: top($c, p_1$), on($c, x_1$), *; true if $p_1$ is not empty*
attached($p_1, l_1$), belong($k, l_1$), *; bind $l_1$ and $k$*
attached($p_2, l_2$), top($x_2, p_2$) *; bind $l_2$ and $x_2$*
subtasks: ⟨take($k, l_1, c, x_1, p_1$), put($k, l_2, c, x_2, p_2$)⟩

recursive-move($p, q, c, x$):
task: move-stack($p, q$)
precond: top($c, p$), on($c, x$) *; true if $p$ is not empty*
subtasks: ⟨move-topmost-container($p, q$), move-stack($p, q$)⟩
*;; the second subtask recursively moves the rest of the stack*

do-nothing($p, q$)
task: move-stack($p, q$)
precond: top($pallet, p$) *; true if $p$ is empty*
subtasks: ⟨⟩ *; no subtasks, because we are done*

move-each-twice()
task: move-all-stacks()
precond: *; no preconditions*
network: *; move each stack twice:*
$u_1$ =move-stack(p1a,p1b), $u_2$ =move-stack(p1b,p1c),
$u_3$ =move-stack(p2a,p2b), $u_4$ =move-stack(p2b,p2c),
$u_5$ =move-stack(p3a,p3b), $u_6$ =move-stack(p3b,p3c),
$\{(u_1, u_2), (u_3, u_4), (u_5, u_6)\}$




<- sous-tâches partiellement ordonnées

# Algorithme: Partial-Order Forward Decomposition (p. 243)

PFD$(s, w, O, M)$
  if $w = \emptyset$ then return the empty plan
  nondeterministically choose any $u \in w$ that has no predecessors in $w$
  if $t_u$ is a primitive task then
    $active \leftarrow \{(a, \sigma) \mid a$ is a ground instance of an operator in $O$,
      $\sigma$ is a substitution such that $\text{name}(a) = \sigma(t_u)$,
      and $a$ is applicable to $s\}$
    if $active = \emptyset$ then return failure
    nondeterministically choose any $(a, \sigma) \in active$
    $\pi \leftarrow$ PFD$(\gamma(s, a), \sigma(w - \{u\}), O, M)$
    if $\pi =$ failure then return failure
    else return $a.\pi$
  else
    $active \leftarrow \{(m, \sigma) \mid m$ is a ground instance of a method in $M$,
      $\sigma$ is a substitution such that $\text{name}(m) = \sigma(t_u)$,
      and $m$ is applicable to $s\}$
    if $active = \emptyset$ then return failure
    nondeterministically choose any $(m, \sigma) \in active$
    nondeterministically choose any task network $w' \in \delta(w, u, m, \sigma)$
    return(PFD$(s, w', O, M)$)

$\pi=\{p_1,\ldots,p_k\}$; $w=\{$ **$t_1$** $,t_2,\ldots\}$

operator instance **o**

P$=\{p_1 \ldots,p_k,$ **o** $\}$; $w'=\{t_2, \ldots\}$

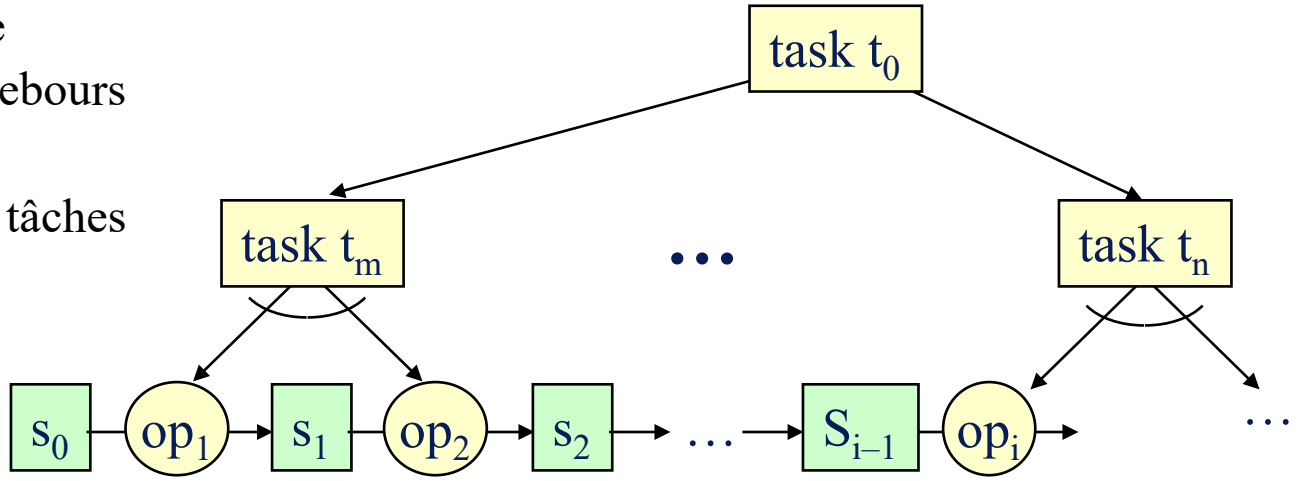
$w=\{$ **$t_1$** $,t_2,\ldots\}$

method instance **m**

w'$=\{$ **$t_{11},\ldots,t_{1k}$** $,t_2,\ldots\}$

# Comparaison à recherche dans l'espace d'états

- *Goal-oriented* comme dans une recherche à rebours
  - » Les buts correspondent aux tâches




- Génère les actions dans l'ordre de leur exécution comme une recherche en avant

- Lorsqu'on veut planifier la prochaine tâche
  - » Toutes les tâches qui la précède sont déjà planifiées
  - » Ainsi, on connaît l'état courant!

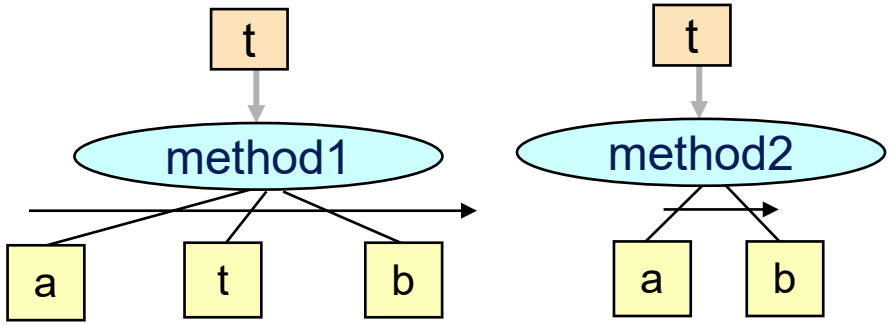
# Comparaison avec la planification classique

- Tout problème de planification classique peut être traduit en un problème HTN, en temps polynomial
  - » Une des façons de le faire:
    - Pour chaque but ou précondition $e$, crée une tâche $t_e$
    - Pour chaque action $o$ et effet $e$, crée une méthode $m_{o,e}$
      - › La tâche effectuée par la méthode est $t_e$
      - › Les sous tâches sont $t_{p1}, t_{p2}, \ldots, t_{pn}, o$ où $p_1, p_2, \ldots, p_n$ sont les préconditions de $o$
      - › Ordre partiel sur les contraintes: chaque $t_{pi}$ précède $o$

- Par contre, il y a des problèmes HTN qui ne peuvent être traduits dans des problèmes de planification classique.

# Comparaison avec la planification classique

- Deux méthodes:
  - » Aucun argument
  - » Aucune précondition
- Deux opérateurs, a et b
  - » Sans argument ni précondition




- État initial : vide; tâche initiale t
- Ensemble de solutions $\{a^n b^n \mid n > 0\}$

- Aucun problème de planification classique n'a cette solution
  - » Le système de transitions est une machine à état fini
  - » Aucune machine à état fini ne peut reconnaître le language $\{a^n b^n \mid n \geq 0\}$

# Implémentations

- L'approche TFD est implémenté dans SHOP
- L'approche PFD est implementé dans SHOP2
- http://www.cs.umd.edu/projects/shop/

# Vers l'apprentissage automatique?

- Shah & Srivastava (AAMAS, 2022). Using Deep Learning to Bootstrap Abstractions for Hierarchical Robot Planning. https://arxiv.org/abs/2202.00907

# Références


- Malik Ghallab, Dana Nau & Paolo Traverso. *Automated Planning: theory and practice.* Morgan Kaufmann, 2004. ISBN: 1-55860-856. http://www.laas.fr/planning/ (Chapitre 11)

- http://www.cs.umd.edu/projects/shop/