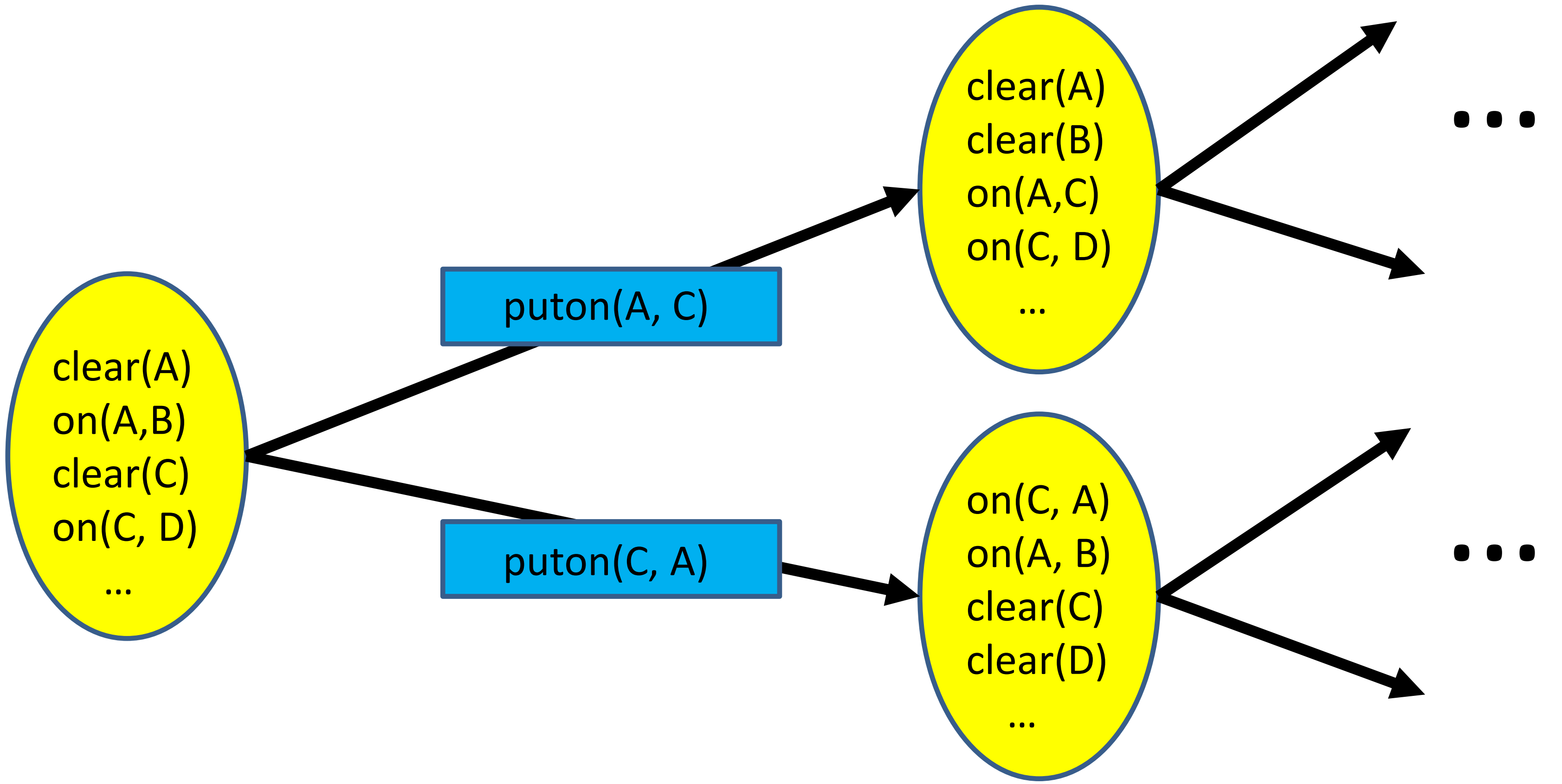
Planification d’actions
Heure 2 : algorithmes pour planifier

Philippe Morignot
pmorignot@yahoo.fr

Histoire des planificateurs

- 1971: STRIPS par Richard Fikes.
- 1977: NOAH par Earl Sacerdoti
- 1981: MOLGEN par Mark Stefik
- 1986: IxTeT par Malik Ghallab.
- 1986: SIPE par David Wilkins.
- 1987: TWEAK par David Chapman.
- 1991: SNLP par Mac Allister & Rosenblitt.
- 1992: UCPOP par Anthony Barrett & Daniel Weld.
- 1992: BLACKBOX/SATPLAN par Henry Kautz & Bart Selman.
- 1997: GRAPHPLAN par Avrim Blum & Merrick Furst.
- 2000: HSP par Hector Geffner.
- 2000: YAHSP par Vincent Vidal.
- 2001: FF par Jörg Hoffmann,
- 2005: CPT par Vincent Vidal.
- 2007: DAE par Marc Schoenauer.
- …

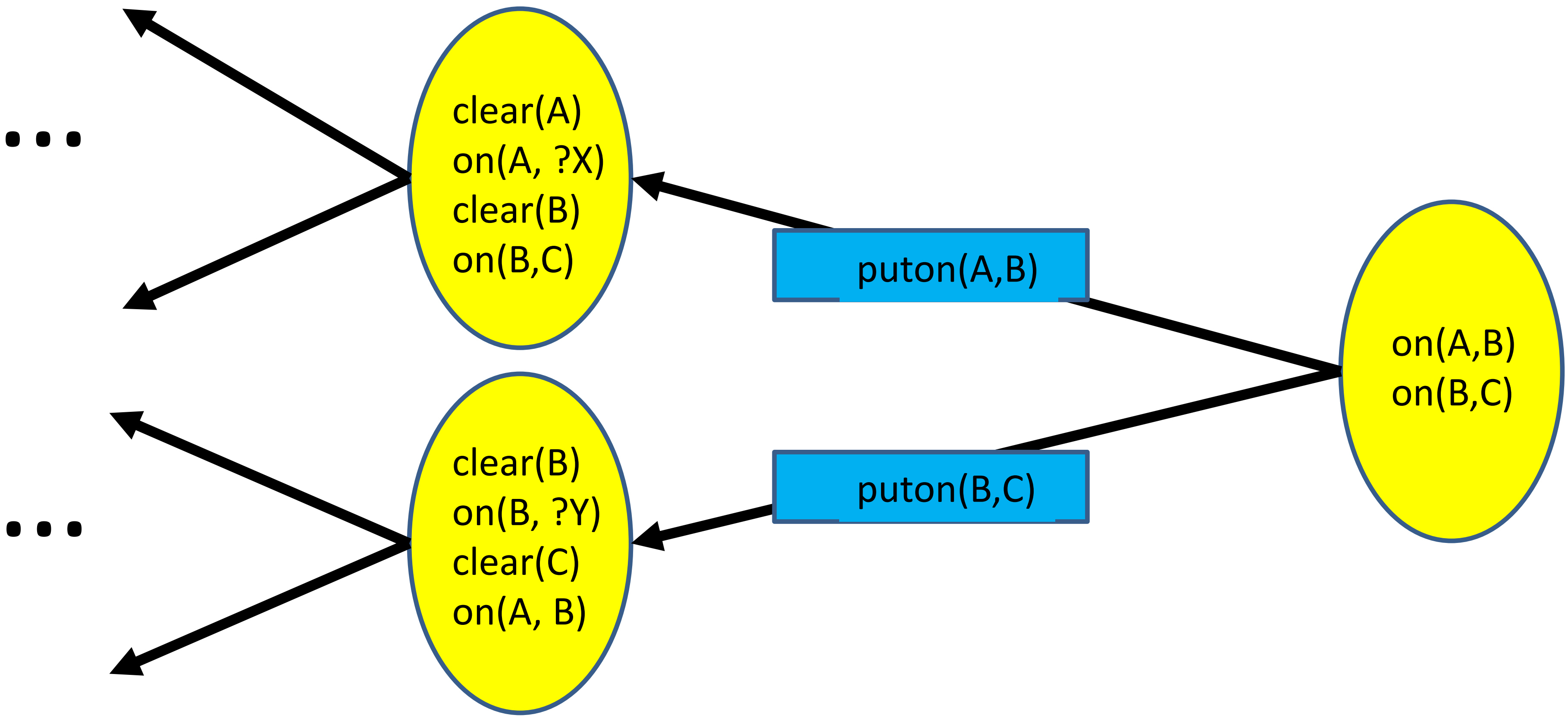
Algorithmes dans un espace d'états : Recherche avant (1 / 2)



Algorithmes dans un espace d'états : Recherche avant (2 / 2)

- Un **état** S_i = un état de l'environnement.
  - S_0 = état initial = $f_1, \ldots, f_n$.
  - Exemple : une configuration de cubes.
- ***Successeurs(S_i)*** = les *N* nouveaux états { $S_{i+1}, \ldots, S_{i+N}$ } atteignables par un opérateur applicable en chaînage avant à S_i :
  - Toutes les préconditions sont présentes dans S_i ; ajouter les effets positifs à S_i et retrancher les effets négatifs de S_i pour obtenir S_{i+j}.
- ***bool Solution(S_i)*** ssi S_i contient les buts $b_1, \ldots, b_l$.
- Algorithme : tout algorithme de recherche dans un espace d'états (par ex., A*).

Algorithmes dans un espace d'états : Recherche arrière (1 / 2)



Algorithmes dans un espace d'états : Recherche arrière (2 / 2)

- Un **état** $\boldsymbol{S_i}$ = un état de l'environnement.
  - $\boldsymbol{S_0}$ = les buts = $\boldsymbol{b_1}$, …, $\boldsymbol{b_l}$.
- ***Prédécesseurs(S_i)*** = *N* nouveaux états { $\boldsymbol{S_{i+1}}$, …, $\boldsymbol{S_{i+N}}$ } obtenus en appliquant un opérateur en chainage arrière à $\boldsymbol{S_i}$.
  - Un opérateur est applicable ssi il possède une post-condition qui s'unifie avec un terme de $\boldsymbol{S_i}$.
  - Ajouter les préconditions à $\boldsymbol{S_i}$ pour obtenir $\boldsymbol{S_{i+j}}$.
- ***bool Solution(S_i)*** ssi $\boldsymbol{S_i}$ contient les littéraux $\boldsymbol{f_1}$, …, $\boldsymbol{f_n}$.

- Algorithme : tout algorithme de recherche dans un espace d'états (par ex., A*).

Algorithmes dans un espace d'états : Heuristiques

- Une fonction heuristique = une estimation de la distance à la solution + admissibilité.
- Mais trouver une bonne heuristique est difficile …
- **Utiliser un problème relaxé** : le coût optimal pour un problème relaxé est une heuristique du problème général.

  Par exemple, ne pas considérer les pré-conditions ; ne pas considérer les post-conditions négatives.

- <u>Exemple :</u> Soit les buts $\boldsymbol{A \wedge B \wedge C}$ et les actions sans préconditions

  Action(X, A ∧ P) ***Action(Y, B ∧ C ∧ Q)*** ***Action(Z, B ∧ P ∧ Q)***

  Les actions ***X*** et ***Y*** suffisent (interaction positive), donc coût de 2.

  Mieux que la résolution indépendante des 3 buts (coût 3) avec préconditions.

Algorithme dans l’espace des plans partiels (1 / 2)

- Plan = (Descriptions ***T***, Opérateurs ***Op***, OrdrePartiel ***O***, Unification ***U***)
  - Conflit = une post-condition qui peut détruire un *lien causal* (menace)
  - Satisfaire une pré-condition ***p*** = faire qu’il y ait un opérateur avant ***p***, dont une post-condition ***q*** est telle que ***p = q***

*PLANIFICATEUR(**T**, **Op**, **O**, **U**)*

TANTQUE ∃ conflit OU ∃ une pré-condition non satisfaite FAIRE

1. Choisir un conflit dans le plan partiel
2. Le résoudre
3. Choisir une pré-condition ***p*** non satisfaite dans le plan partiel
4. La satisfaire
   - En ajoutant une contrainte d’unification ou de non-unification à ***U***
   - En ajoutant une contrainte de précédence à ***O***
   - En ajoutant un opérateur de ***T*** partiellement instantié à ***Op***

Algorithme dans l'espace des plans partiels (2 / 2)

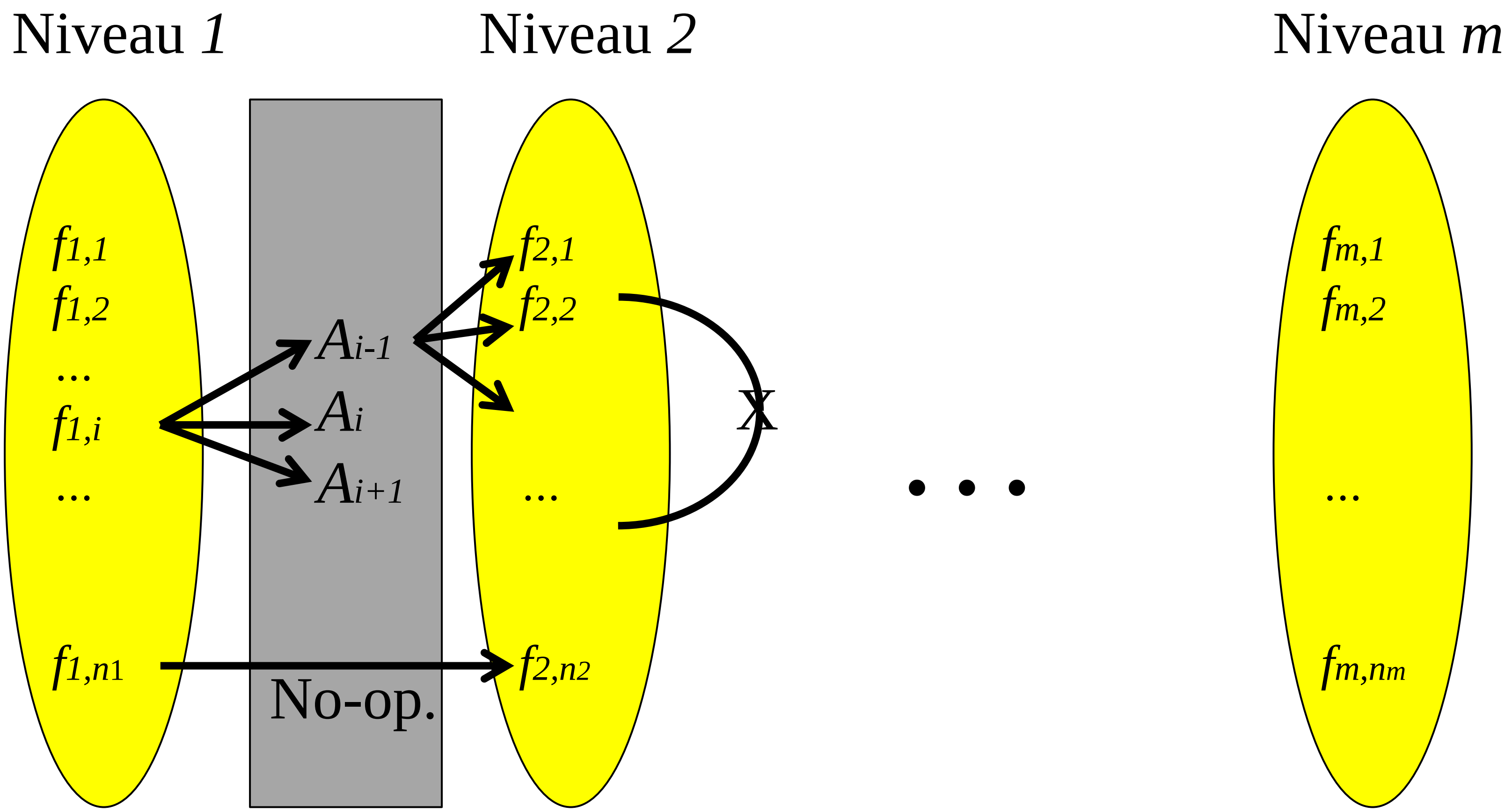
PLANIFICATEUR(T*, Op, O, U)*

Algorithme de recherche dans un espace d'états (par ex., A*) avec :

- Un **état** = un plan partiellement ordonné partiellement instantié(***T***, ***Op***, ***O***, ***U***).
- Un **successeur** = obtenu par résolution de conflit, ou satisfaction d'une pré-condition.
  - Ajout d'une contrainte d'unification / non-unification à ***U***
  - Ajout d'une contrainte de précédence à ***O***
  - Ajout d'un opérateur de ***T*** à ***Op***
- Une **fonction solution** = aucun conflit ET toutes les pré-conditions sont satisfaites.

- Heuristique : nombre de préconditions non satisfaites, nombre de conflits

Plan de graphe (1 / 4)



- Développer un plan de graphe en avant, et recherche arrière.
- Niveaux avec relations d'exclusions mutuelles (mutex)
- Un niveau n'est pas un état, mais des littéraux possibles.

Plan de graphe (2 / 4)

- **Mutex entre opérateurs d'un même niveau :**
  - Un opérateur possède un effet qui nie un effet d'un autre opérateur (effets inconsistants)
  - Un effet d'un opérateur est la négation d'une précondition d'un autre opérateur (interférence).
  - Une précondition d'un opérateur est mutex avec une précondition d'un autre opérateur (besoins en compétition).
- **Mutex entre littéraux d'un même niveau :**
  - L'un est la négation de l'autre.
  - Toute paire d'action possible, qui les établit, est mutex.

Plan de graphe
Exemple (1 / 2)

- **Etat initial :** avoir(Gateau)
- **Buts :** avoir(Gateau) $\wedge$ mangé(Gateau)

- **Action :** mange(Gateau)

  Pré-conditions : avoir(Gateau)

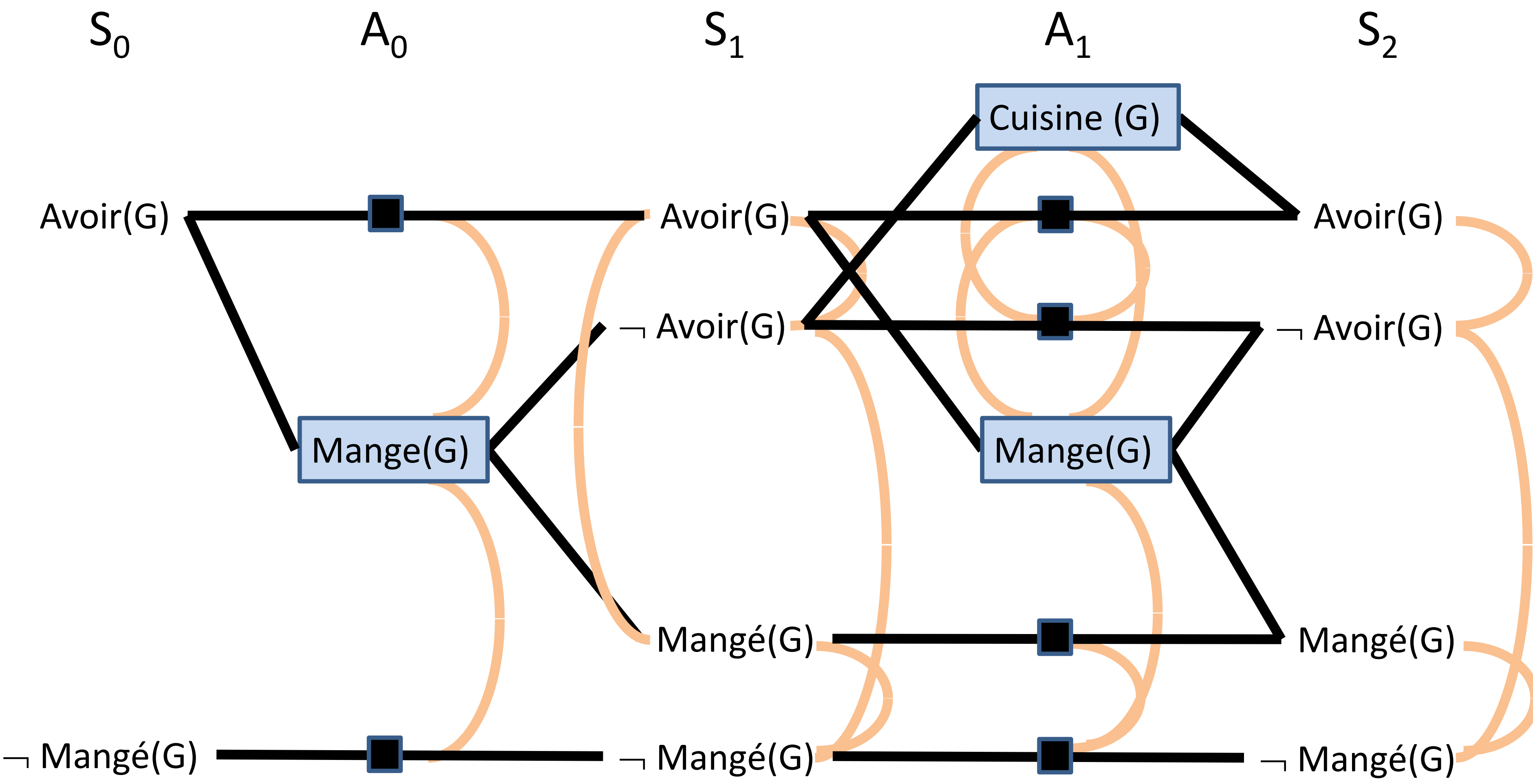
  Post-conditions : $\neg$ avoir(Gateau) $\wedge$ mangé(Gateau)

- **Action :** cuisine(Gateau)

  Pré-conditions : $\neg$ avoir(Gateau)

  Post-conditions : avoir(Gateau)

Plan de graphe
Exemple (2 / 2)



Plan de graphe (3 / 4)

- Structure de données :
  - Plan de graphe = termes et opérateurs en niveaux alternés
  - No-op

- GRAPHLAN($\boldsymbol{N_0}$, Buts, T)

  Poser le niveau initial égal à $\boldsymbol{N_0}$

  TANTQUE !plan-solution OU $\boldsymbol{N_i}$!= $\boldsymbol{N_{i-1}}$ FAIRE

  - SI les buts sont dans le dernier niveau et ne sont pas mutex

    ALORS rechercher un plan-solution depuis le dernier niveau en arrière.

    SI un tel plan-solution est trouvé, ALORS SUCCES.

  - Expansion du plan de graphe sur un niveau :

    Développer toutes les actions possibles

    Ecrire les relations d'exclusion mutuelle *mutex*

Plan de graphe (4 / 4)

- Recherche d'un plan-solution en arrière dans le plan de graphe :
  - Une recherche dans un espace d'états.

  - **Etat initial** = niveau $\boldsymbol{S_n}$ avec les buts
  - **Prédécesseurs** = choisir un sous-ensemble d'actions dans $\boldsymbol{A_{i-1}}$ sans conflit qui satisfasse les buts de $\boldsymbol{S_i}$.
  - **But** = arriver à $\boldsymbol{S_0}$ avec tous les buts satisfaits.

Planificateurs par solver SAT (1 / 4) Rappels sur les solveurs SAT

- Problème de la **sat**isfiabilité d'une formule logique : *Trouver la valeur de vérité de chaque proposition qui, ensemble, satisfont une formule logique donnée*.
- Exemple de formule : $(\boldsymbol{p} \wedge \boldsymbol{q}) \vee (\boldsymbol{r} \wedge \neg \boldsymbol{s} \wedge \boldsymbol{t}) \vee (\boldsymbol{u} \wedge \boldsymbol{v} \wedge \neg \boldsymbol{w})$
- Logique des propositions.
- 3SAT est le 1er problème prouvé NP-complet en 1971.
- Conférence Internationale SAT : http://www.satisfiability.org/
- Applications : preuves de la correction de circuits intégrés dans un chip, preuve de non risque d'accident dans le programme qui pilote une ligne de métro (par ex., ligne 1 du métro parisien), ….

Planificateurs par solver SAT (2 / 4) Algorithme

- En logique des propositions, essayer de prouver la formule : *etat-initial* $\wedge$ *tous-les-plans-possibles* $\wedge$ *buts*

- **Principe de l'algorithme :**
  1. *n = 1 // Longueur du plan-solution.*
  2. Transformer le problème de longueur *n* en une formule en logique des propositions.
  3. Utiliser un solveur SAT pour essayer de prouver la formule.
     Si prouvé, ALORS extraire le plan-solution de la formule ET SUCCES.
  4. SI le solveur SAT échoue, ALORS incrémenter *n* et GOTO 2.

Planificateurs par solver SAT (3 / 4) Encodage de l'anomalie de Sussman

- **Buts :** on(A,B)@T $\wedge$ on(B,C)@T
- **Etat initial :** clear(C)@0 $\wedge$ on(C,A)@0 $\wedge$ clear(B)@0
  ($\wedge \neg$ on(A,C)@0 $\wedge \neg$ on(A,B)@0 $\wedge \neg$ on(B,C)@0 $\wedge \neg$ on(B,A)@0
  $\wedge \neg$ on(C, B)@0 $\wedge \neg$ clear(A)@0) *// hypothèse du monde fermé*
- **Schémas d'axiome, sur les préconditions :**

$\forall$ ***x***, $\forall$ ***y***, $\forall$ ***z***, $\forall$ ***t*** :

puton(***x***, ***y***, ***z***)@***t*** $\Rightarrow$ on(***x***,***y***)@***t*** $\wedge$ clear(***x***)@***t*** $\wedge$ clear(***z***)@***t***

- **Schémas d'axiome, sur les effets :**

$\forall$ ***x***, $\forall$ ***y***, $\forall$ ***z***, $\forall$ ***t*** :

on(***x***,***y***)@***t*** $\wedge$ clear(***x***)@***t*** $\wedge$ clear(***z***)@***t*** $\wedge$ puton(***x***,***y***,***z***)@***t*** $\Rightarrow$ clear(***y***)@***t***+1 $\wedge$ on(***x***,***z***)@***t***+1

- **Un opérateur à la fois :**

$\forall$ ***x***, $\forall$ ***y***, $\forall$ ***y'***, $\forall$ ***z***, $\forall$ ***z'***, $\forall$ **t** / ***y*** <> ***y'*** $\wedge$ ***z*** <> ***z'*** :

$\neg$ (puton(***x***, ***y***, ***z***)@***t*** $\wedge$ puton(***x*** ,***y'*** ,***z'***)@***t***)

- **Schémas d'axiome, pour le problème du cadre:**

$\forall$ p, $\forall$ t: $\quad p@(t+1) \Rightarrow (\quad p@t \vee a_1{}^p@t \quad \vee \ldots \vee a_n{}^p@t \ )$

$\neg p@(t+1) \Rightarrow (\neg p@t \vee a_1{}^{\neg p}@t \vee \ldots \vee a_n{}^{\neg p}@t \ )$

Planificateurs par solver SAT (4 / 4) Amélioration de l'encodage

- **Complexité de cet encodage :** T x $|Act|$ x $|O|^P$ avec :

  T = taille limite du plan

  $|Act|$ = nombre d'opérateurs

  $|O|$ = nombre d'objets

  P = arité maximum d'un opérateur

- Séparation des symboles : certains symboles séparés seront inutiles dans les axiomes

  puton1(A) = cube qui est bougé

  puton2(B) = cube destination

  puton3(C) = cube du dessous

- Exemple : contrainte d'état :

  puton1(A)@t $\wedge$ (puton2(B)@t $\wedge \neg$ puton2(C)@t $\vee$ puton2(C)@t $\wedge \neg$ puton2(B)@t)

- **Complexité de ce 2e encodage :** T x $|Act|$ x $|O|$

Planificateurs par PPC (1 / 3)

- Programmation par contraintes (rappel) : variables, domaines, contraintes + heuristiques + solveur.
- Principe :
  1. Estimer une borne inférieure de la longueur *n* du plan-solution
  2. Transformer le problème de planification de longueur *n* en un CSP
  3. Résoudre cette formulation avec un solver de CSP. SI trouvé, SUCCES.
  4. Si le solver de CSP échoue, ALORS incrémenter *n* et GOTO 2.
- Exemples :
  - IxTeT du LAAS à Toulouse [Laborie 95].
    http://spiderman-2.laas.fr/RIA/IxTeT/ixtet-planner.html
  - Constraint Programming Temporal planner (CPT) de l'ONERA Toulouse [Vidal 06].
    http://v.vidal.free.fr/onera/#cpt

Planificateurs par PPC (2 / 3)

- Un opérateur instantié apparait au plus une fois (canonicité).
- Pseudo opérateurs *Start* (sans pré-condition) et *End* (sans post-condition).
- Heuristique pour estimer la longueur ***B*** du plan-solution.
- Puisqu'un plan-solution est linéaire, numéroter les instants.
- Une action peut durer plus que 1 unité de temps (actions duratives).
- Déclarer des contraintes redondantes.

- Variables du CSP :
  - Date de début de l'opérateur o : $T(\boldsymbol{o}) \in [\ 0\ ;\ \boldsymbol{B}\ [$
  - Support de la précondition ***p*** dans l'opérateur ***o*** : $S(\boldsymbol{p}, \boldsymbol{o}) \in O(\boldsymbol{p})$
  - Date de début du $S(\boldsymbol{p}, \boldsymbol{o})$: $T(\boldsymbol{p}, \boldsymbol{o}) \in [\ 0\ ;\ \boldsymbol{B}\ [$
  - $InPlan(\boldsymbol{o}) \in \{0, 1\}$ indique la présence de l'opérateur ***o*** dans le plan
    InPlan(Start) = 1, *InPlan(End)* = 1.

Planificateurs par PPC (3 / 3) Contraintes

- **Bornes :** $T(Start) + dist(Start, \mathbf{o}) \leq T(\mathbf{o})$ $\quad$ $T(\mathbf{o}) + dur(\mathbf{o}) + dist(\mathbf{o}, End) \leq T(End)$
- **Contraintes de support :**

  $S(\mathbf{p}, \mathbf{o}) = \mathbf{o'} \Rightarrow T(\mathbf{p}, \mathbf{o}) = T(\mathbf{o'})$

  $\min_{\mathbf{o'} \text{ in } S(\mathbf{p}, \mathbf{o})} (T(\mathbf{o'})) \leq T(\mathbf{p}, \mathbf{o}) \leq \max_{\mathbf{o'} \text{ in } S(\mathbf{p}, \mathbf{o})} (T(\mathbf{o'}))$
- **Préconditions :** Les supports ***o'*** de la précondition ***p*** de ***o*** doivent précéder o selon *dist(**o'**, **o**)*

  $T(\mathbf{o}) \geq \min_{o' \in D(S(\mathbf{p}, \mathbf{o}))} (T(\mathbf{o'}) + dur(\mathbf{o'}) + dist(\mathbf{o'}, \mathbf{o}))$

  $T(\mathbf{o}) \geq T(\mathbf{p}, \mathbf{o}) + \min_{o' \in D(S(\mathbf{p}, \mathbf{o}))} (dur(\mathbf{o'}) + dist(\mathbf{o'}, \mathbf{o}))$

  $T(\mathbf{o'}) + dur(\mathbf{o'}) + dist(\mathbf{o'}, \mathbf{o}) > T(\mathbf{o}) \Rightarrow S(\mathbf{p}, \mathbf{o}) \neq \mathbf{o'}$
- **Liens causaux :** Pour toute précondition ***p*** de ***o*** et pour tout ***o'*** qui détruit ***p***, ***o'*** est avant *S(**p**, **o**)* ou suit ***o***

  $T(\mathbf{o'}) + dur(\mathbf{o'}) + \min_{o'' \in D(S(\mathbf{p}, \mathbf{o}))} (dist(\mathbf{o'}, \mathbf{o''})) \leq T(\mathbf{p}, \mathbf{o}) \quad \vee$

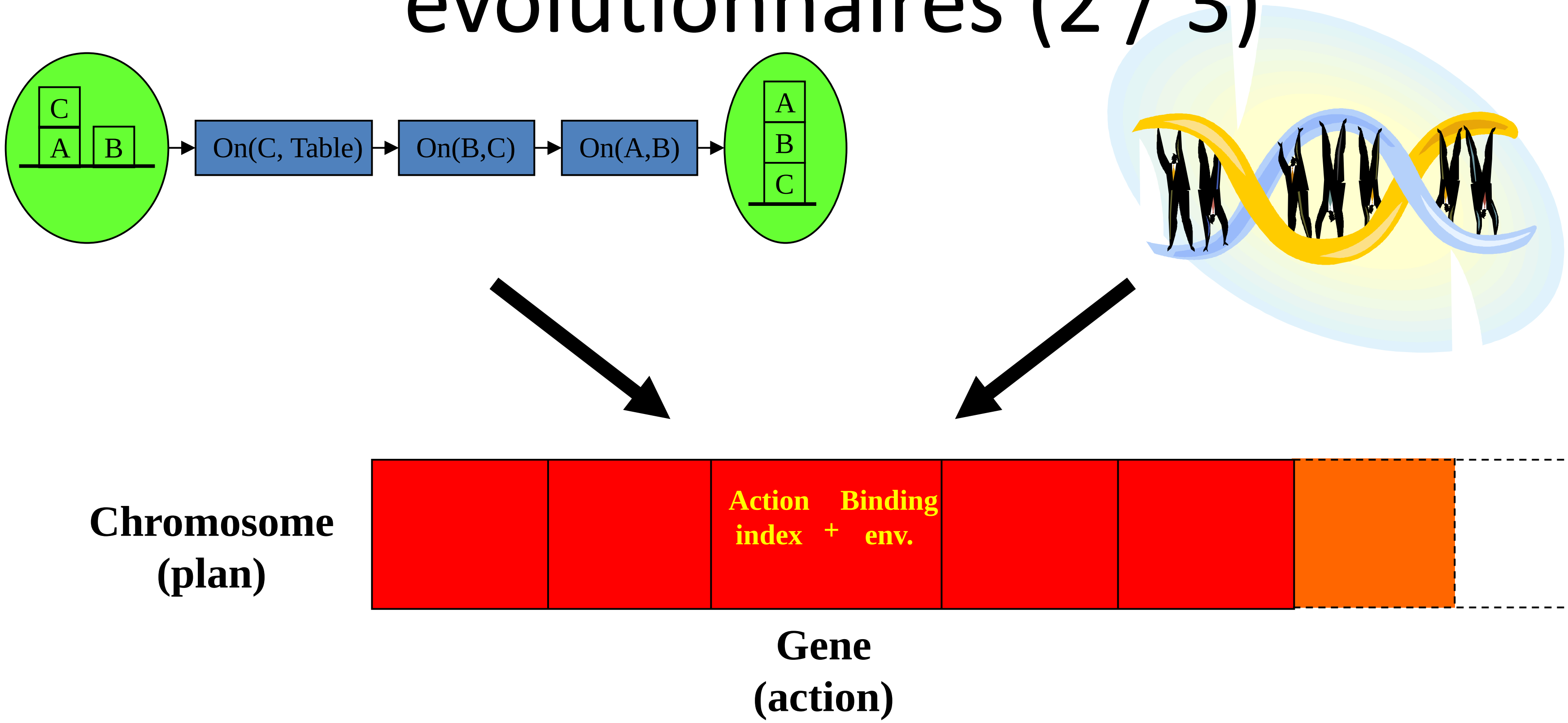
  $T(\mathbf{o}) + dur(\mathbf{o}) + dist(\mathbf{o}, \mathbf{o'}) \leq T(\mathbf{o'})$
- **Mutex :** si les opérateurs ***o*** et ***o'*** sont mutex, pas de parallélisme

  $T(\mathbf{o}) + dur(\mathbf{o}) + dist(\mathbf{o}, \mathbf{o'}) \leq T(\mathbf{o'}) \vee T(\mathbf{o'}) + dur(\mathbf{o'}) + dist(\mathbf{o'}, \mathbf{o}) \leq T(\mathbf{o})$

Planificateurs par algorithmes évolutionnaires (1 / 3)

- Algorithmes évolutionnaires (rappels) :
  - Un individu est représenté par un chromosome (une séquence de gènes).
  - Opérateurs de croisement entre 2 chromosomes et de mutation d'un chromosome.
  - Fonction d'adaptation à l'environnement.
  - Emergence espérée d'une solution après N générations de la population d'individus.
- Encodage :
  - Un plan partiel séquentiel est un individu, un gène est un opérateur instantié.
  - Fonction d'adaptation = nombre de conflits dans l'individu.

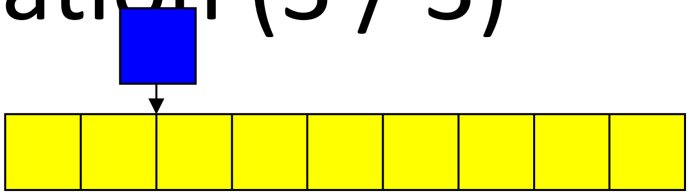
Planificateurs par algorithmes évolutionnaires (2 / 3)



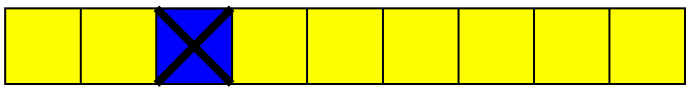
$$C = (a_i, (p_{i,j}, o_j)_{j \in Param(i)})_{i \in [1,N]} \quad \text{where} \begin{cases} a_i & : \text{index of } i\text{-th action} \\ p_{i,j} & : \text{index of } j\text{-th parameter in } i\text{-th action} \\ o_j & : \text{index of } j\text{-th object in the parameter list} \end{cases}$$

Planificateurs par algorithmes évolutionnaires : mutation (3 / 3)

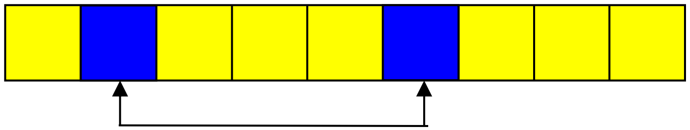
- Ajout aléatoire de gène :



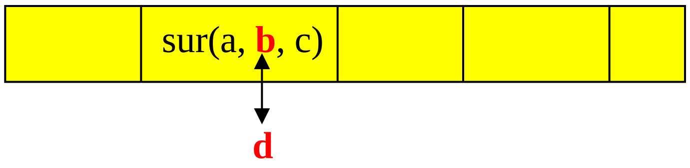
- Retrait aléatoire de gène :



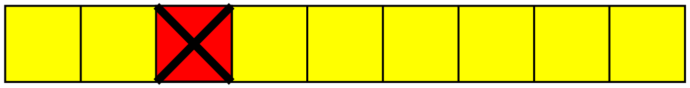
- Permutation aléatoire de deux gènes:



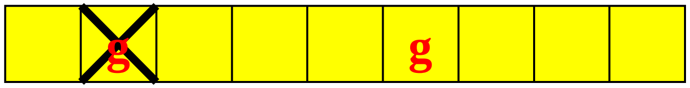
- Remplacement d’un gène :



- Mutations heuristiques :
  - Retrait d’un gène conflictuel :



  - Retrait d’un gène dupliqué :



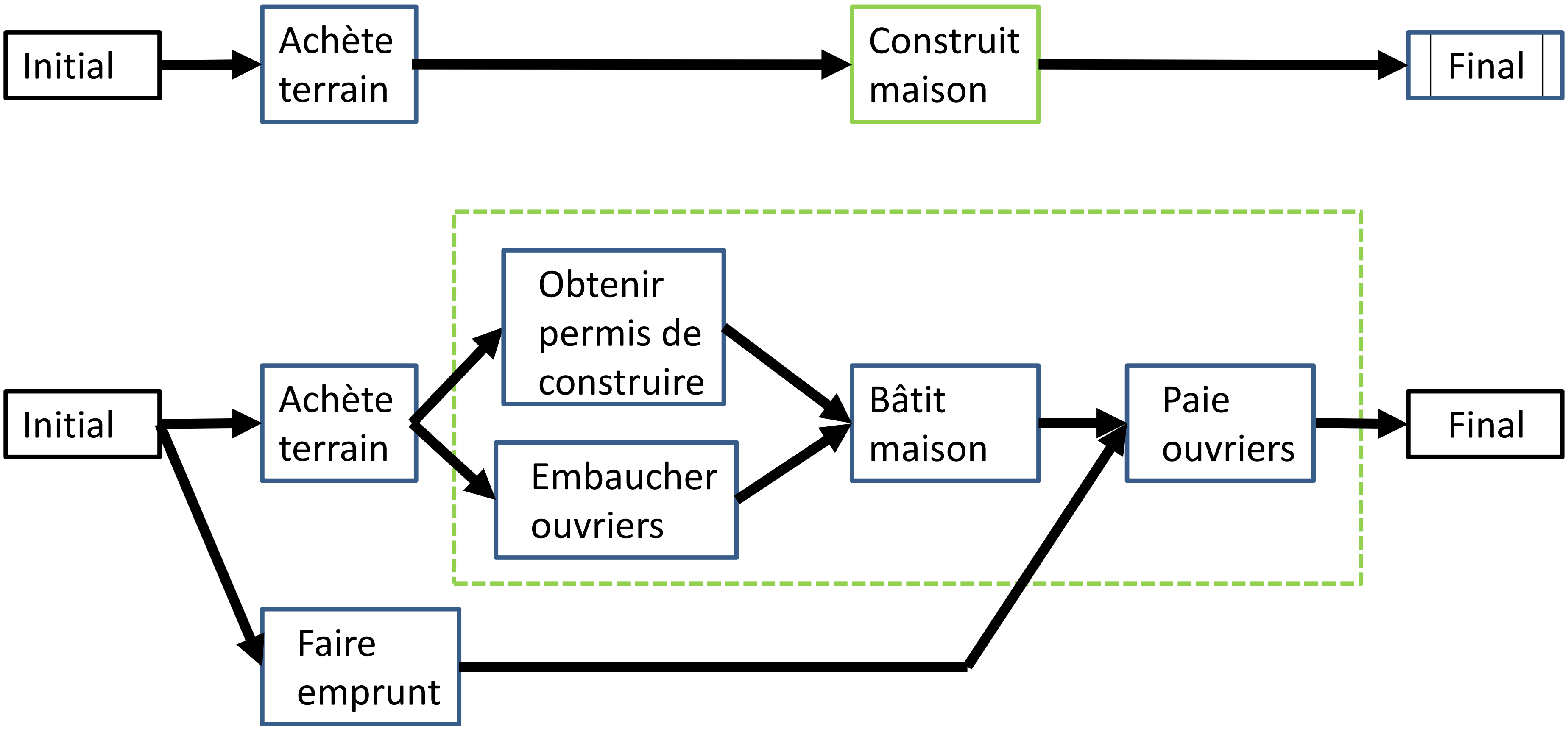
Réseaux de tâches hiérarchiques (1 / 2)

- Planifier à un niveau d'abstraction donné, se préoccuper des détails ensuite. Ne pas planifier d'entrée de jeu au niveau de détail le plus grand .
  - Diminuer la complexité.

- Un opérateur se décompose en d'autres opérateurs (décomposition d'actions).
  - Connaissance supplémentaire.
  - Bibliothèque de plans.
  - Décomposition potentiellement récursive : marcher → faire-un-pas PUIS marcher

- Algorithme : cas particulier des planificateurs dans l'espace des plans partiels : la fonction *Successeur()* contient en plus le fait de décomposer une action.

Réseaux de tâches hiérarchiques (2 / 2)



Les planificateurs

- Quelle est le meilleur algorithme ?
  - Chaque planificateur est en général testé sur des benchmarks en PDDL.
  - Il n’y en a pas de meilleur absolument sur tous les problèmes à la fois …
  - International Planning Competition depuis 1998

  http://ipc.icaps-conference.org/

Références

- **[Blum 97]** A. Blum, M. Furst. *Fast Planning through Planning Graph Analysis*. Artificial Intelligence, 90:281-300, 1997.
- **[Bonet 98]** B. Bonet, H. Geffner. *HSP: Heuristic Search Planner*. In Proceedings of Artificial Intelligence Planning Systems (AIPS), 1998.
- **[Kautz 92]** H. Kautz, B. Selman. *Planning as Satisfiability*. In Proceedings of ECAI'92.
- **[Laborie 95]** P. Laborie, M. Ghallab. *Planning with Sharable Resource Constraints*. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1995, pages 1643 – 1651.
- **[Penberthy 92]** J. S. Penberthy, D. Weld. *UCPOP: A Sound, Complete, Partial-Order Planner for ADL*. In Proceedings of 3rd International Conference on Knowledge Representation and Reasoning (KR'92), Cambridge, MA, 1992.
- **[Vidal 06]** V. Vidal, H. Geffner. *Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming*. Artificial Intelligence, 170(3): 298-335, 2006.

Conclusion

- Plusieurs types d'algorithmes pour construire un planificateur d'actions :
  - Dans l'espace des états.
  - Dans l'espace des plans partiels.
  - Plan de graphe.
  - Basé sur l'utilisation d'un solver SAT.
  - Basé sur la programmation par contraintes.
  - Basé sur les algorithmes évolutionnaires.
  - Réseaux de tâches hiérarchiques.
- <u>Non vu :</u> planification par logique temporelle, planification probabiliste, ...