

Planification d'actions

Philippe Morignot
pmorignot@yahoo.fr

Plan du cours

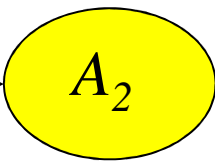
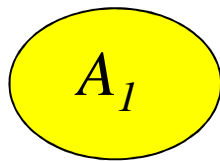
- Définition
- Modèle et Langages
- Exemple
- Algorithmes : espace des états, espace des plans, plan de graphe, par SAT, par PPC, par algorithmes évolutionnaires, HTN.

Enoncé

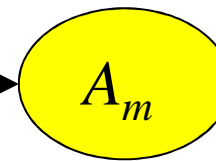
« Etant donnés des actions génériques possibles,
un état et des buts,
trouver une séquence d'actions instantiées,
menant l'état initial à un état (final)
contenant les buts. »

Etat

f_1
 f_2
...
 f_n



...

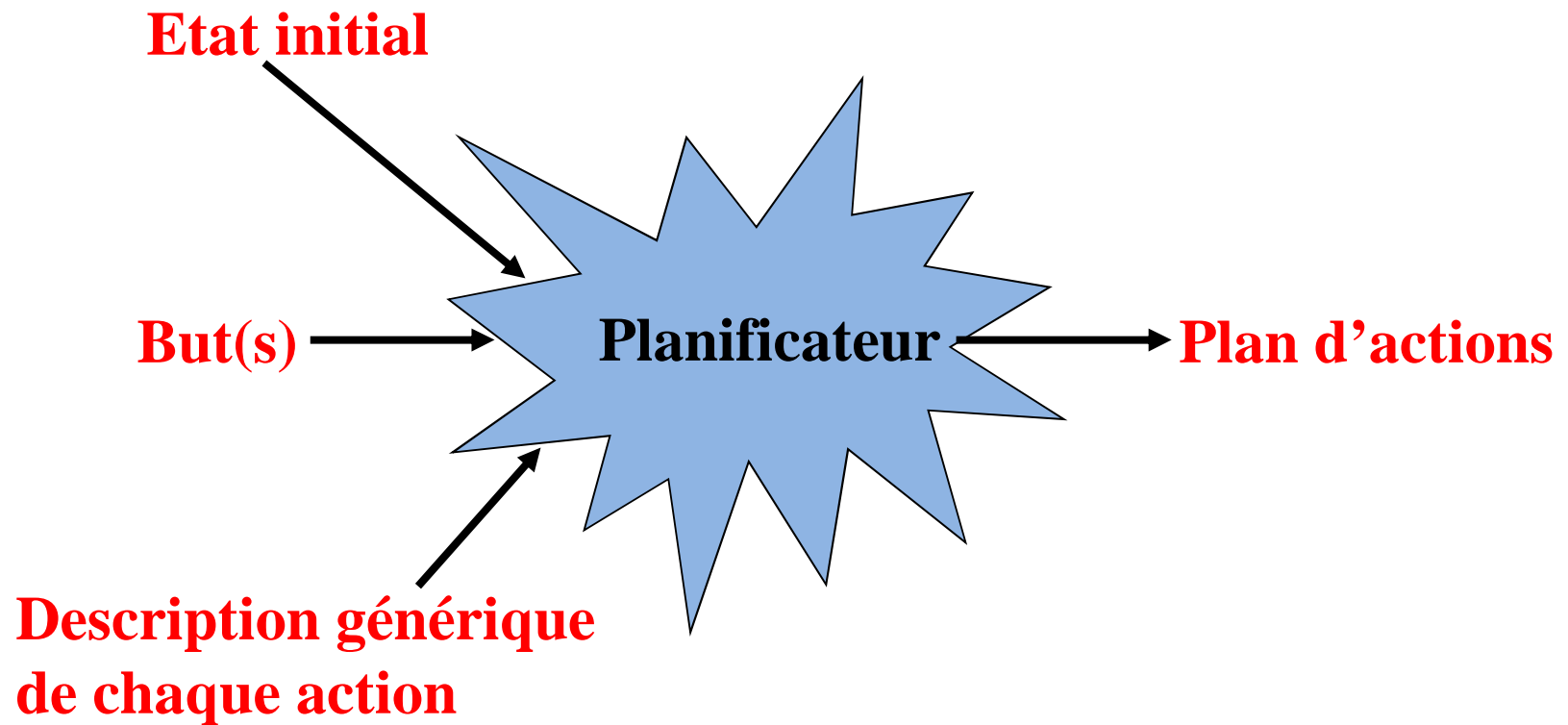


Buts

b_1
 b_2
...
 b_l

- « **Planification d'actions** » / « **synthèse de plan** » / « **génération de plans d'actions** » : activité de construction d'un plan.
- « **Planificateur** » / « **planificateur de tâches** » / « **planificateur d'actions** » : programme informatique qui résout ce problème.
 - Différent de « planificateur de chemin » en Robotique.

Un planificateur d'actions



Difficulté

- Domaine du grutier :
 - 1 grue, a lieux, b camions, c piles de conteneurs, d conteneurs.



- Si $a = 5$, $b = 3$, $c = 3$, $d = 100$, alors $\sim 10^{277}$ états.
- La planification classique est NP.
- **On ne peut pas expliciter tous les états.**

Hypothèses

- Hyp. 1 : *l'agent est la seule cause de changement dans l'environnement.*
 - Pas d'autre agent, artificiel ou humain.
- Hyp. 2 : *l'environnement est totalement observable, l'agent en a une connaissance parfaite.*
 - L'agent ne raisonne (e.g., planifie) pas sur des choses qu'il ne connaît pas.
- Hyp. 3 : *l'environnement est statique.*
 - Même si l'environnement peut avoir des lois de comportements, il ne bouge pas spontanément.
- Hyp. 4 : *le nombre d'objets considérés est fini.*
 - Logique des propositions.

Planning Domain Definition Language (PDDL) (1 / 2)

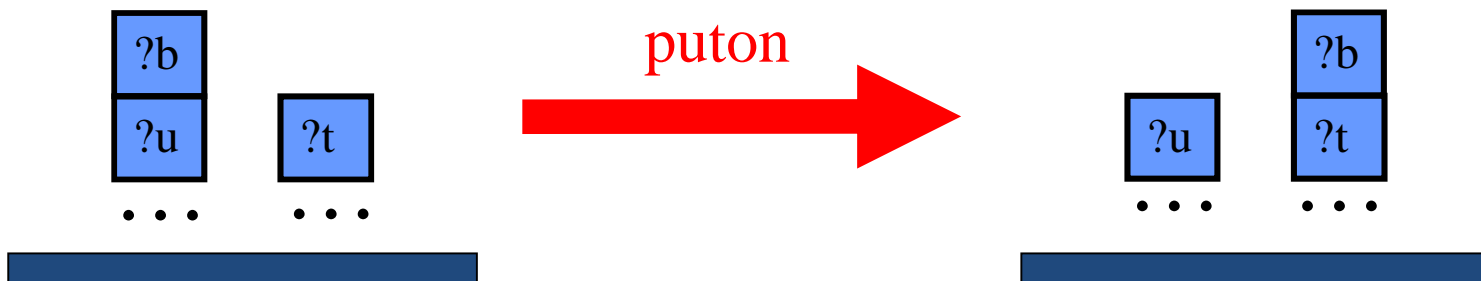
- Langage de représentation pour définir :
 - un domaine : opérateurs
 - un problème : état et buts.
- Un opérateur est composé de :
 - **Pré-conditions** : termes qui doivent être vrai pour que l'action puisse s'exécuter.
 - **Effets / post-conditions** : termes que l'exécution de l'action change par rapport à l'état entrant (liste d'ajouts ADD-LIST / de retraits DELETE-LIST).
 - Une post-condition peut être positive ou négative.
- Un terme peut être parfois vrai, parfois faux, suivant l'instant où on le considère dans le plan.
 - Connecteur « *not* ». Ex. : (**not** (SUR SOURIS TAPIS))
 - « *Fluent* » (*littéral*). Ex. : (SUR SOURIS TAPIS)

PDDL : exemple de domaine

Le monde des cubes

- Opérateur : **(:action puton**
:parameters (?b ?u ?t - block)
:precondition (and (clear ?b)
(on ?b ?u))
(clear ?t))
:effect (and (not (on ?b ?u)) (clear ?u)
(on ?b ?t) (not (clear ?t))))

puton ?b ?u ?t	
(clear ?b)	(not (on ?b ?u))
(on ?b ?u)	(clear ?u)
(clear ?t)	(on ?b ?t)
	(not (clear ?t))



- Et la table ? Et le bras ? Et si plusieurs bras ? Et si les cubes sont colorés ? Ou avec une encoche ? Ou de dimensions variables ?
- Conditionnelles ? Quantification universelle ?

PDDL : représentation

- Problème de la **qualification** : en pratique, on ne peut pas lister toutes les pré-conditions dans un opérateur.
- Problème de la **ramification** : en pratique, on ne peut pas lister toutes les post-conditions dans un opérateur.
- Exemple : opérateur « Démarrer une voiture »
 - Pré-conditions : clé-dans-le-barillet ET clé-tournée
 - Post-conditions : moteur-tourne

Planning Domain Definition Language (PDDL) (2 / 2)

- **Résolution du problème du cadre :** lors de l'exécution d'un opérateur, ce qui n'est pas explicitement changé par une post-condition est considéré comme inchangé.
- **Hypothèse du monde clos :** dans un état, un terme qui n'est pas mentionné est considéré comme étant *faux*.
 - Par opposition à *l'hypothèse du monde ouvert* (ontologies) : dans un état, ce qui n'est pas mentionné est considéré comme étant *inconnu*.

STRIPS et ADL

(Stanford Research Institute Planning System) (Action Description Language)

Littéraux positifs seulement dans les états : Riche \wedge Célèbre	Littéraux positifs et négatifs dans les états : \neg Riche \wedge \neg Célèbre
Hypothèse du monde fermé	Hypothèse du monde ouvert.
Effet $P \wedge \neg Q$ signifie ajoute P et détruit Q	Effet $P \wedge \neg Q$ signifie ajoute P et $\neg Q$ et détruit $\neg P$ et Q
Propositions seulement dans les buts Riche \wedge Célèbre	Variables quantifiées dans les buts : $\exists x, AT(Avion1, x) \wedge AT(Avion2, x)$
Les buts sont des conjonctions : Riche \wedge Célèbre	Les buts sont des conjonctions et disjonctions : \neg Pauvre \wedge (Riche \vee intelligent)
Les effets sont des conjonctions	Effets conditionnels : QUAND P : E
Pas d'égalité	Egalité ($x = y$) prédéfinie
Pas de types	Les variables peuvent être typées

PDDL : exemple de problème

(define (problem blocks-24-1)

(:domain blocks)

(:objects X W V U T S R Q P O N M L K J I H G F E D C A B)

(:init

(CLEAR K) (CLEAR I) (ONTABLE C) (ONTABLE O)

(ON K F) (ON F T) (ON T B) (ON B G) (ON G R)

(ON R M) (ON M E) (ON E J) (ON J V) (ON V N)

(ON N U) (ON U H) (ON H C) (ON I A) (ON A P)

(ON P Q) (ON Q D) (ON D W) (ON W X) (ON X S)

(ON S L) (ON L O) (HANDEEMPTY))

(:goal (and

(ON L C) (ON C P) (ON P Q) (ON Q M) (ON M B)

(ON B G) (ON G F) (ON F K) (ON K E) (ON E R)

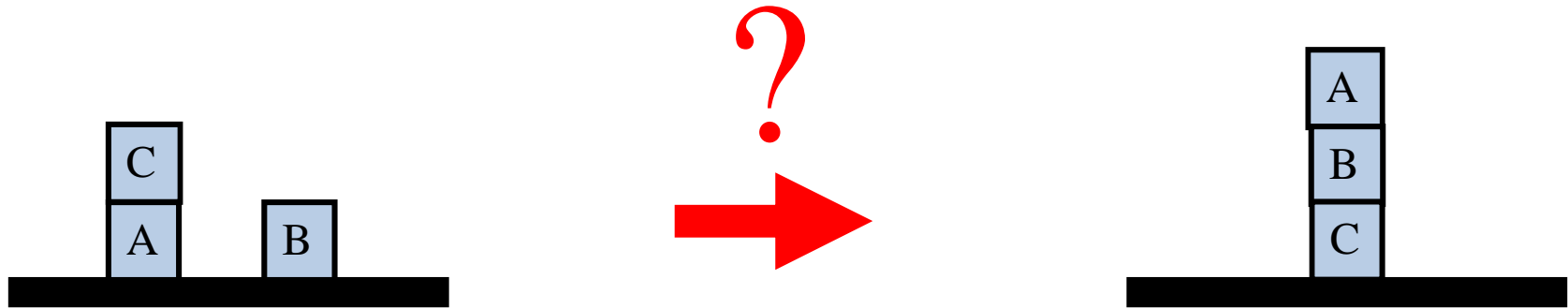
(ON R A) (ON A W) (ON W T) (ON T N) (ON N J)

(ON J U) (ON U S) (ON S D) (ON D H) (ON H V)

(ON V O) (ON O I) (ON I X))))

L'anomalie de Gerald Jay Sussman

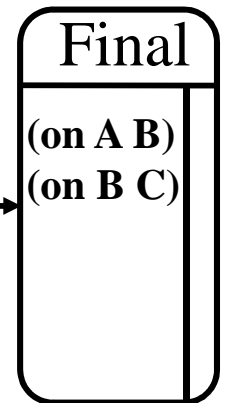
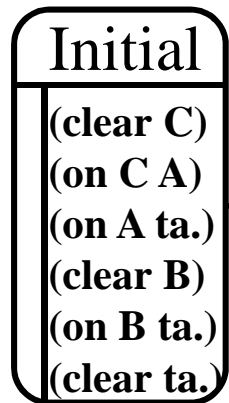
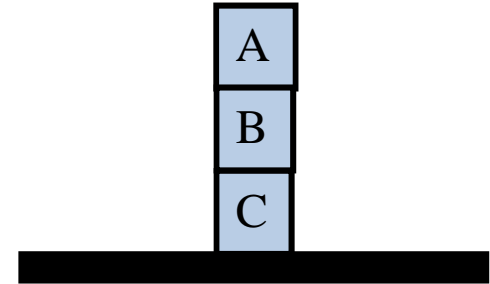
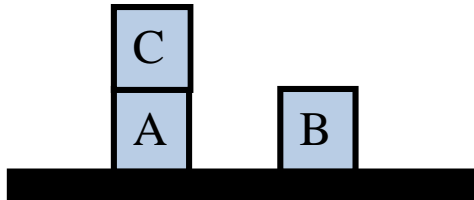
(1/16)



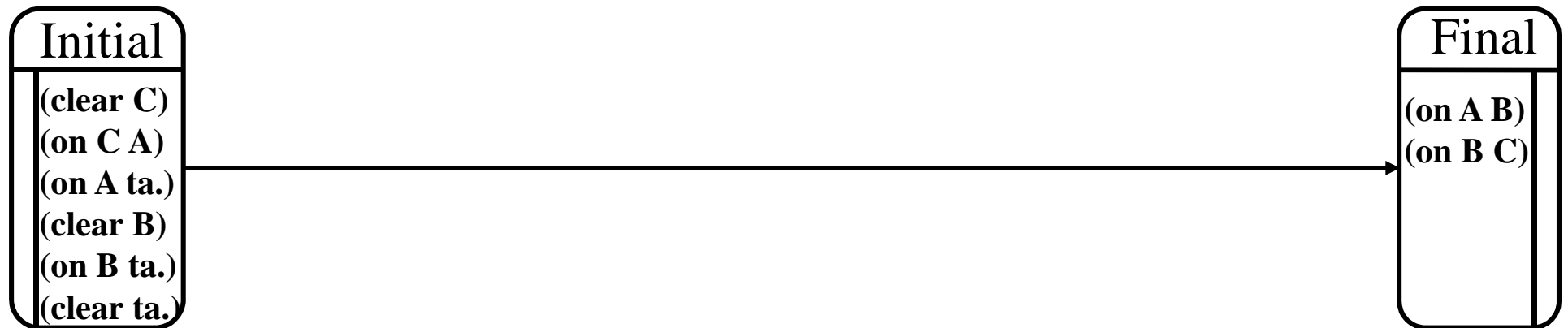
avec :

puton ?b ?u ?t	
(clear ?b)	(not (on ?b ?u))
(on ?b ?u)	(=> (<> ?u table)
(clear ?t)	(clear ?u)
	(on ?b ?t)
	(=> (<> ?t table)
	(not (clear ?t)))

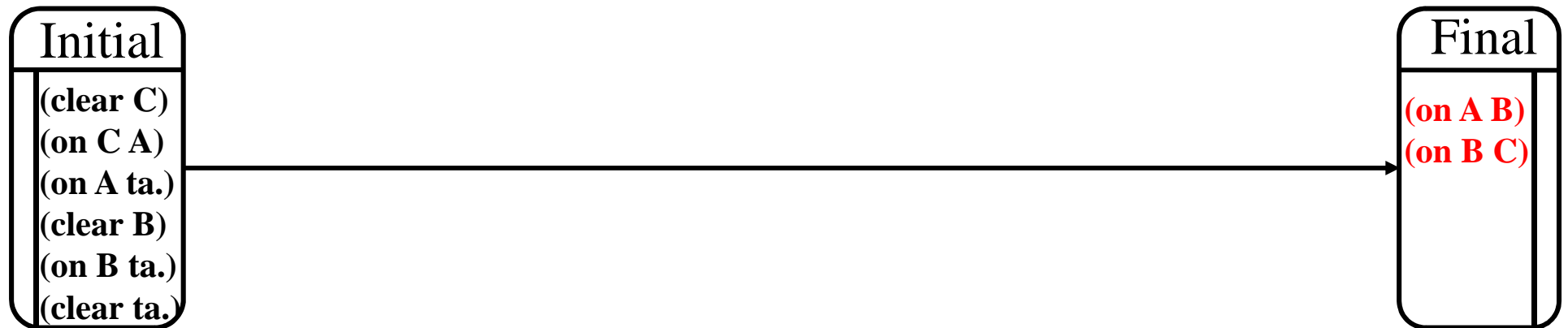
L'anomalie de Gerald Jay Sussman (2/16)



L'anomalie de Gerald Jay Sussman (2/16)

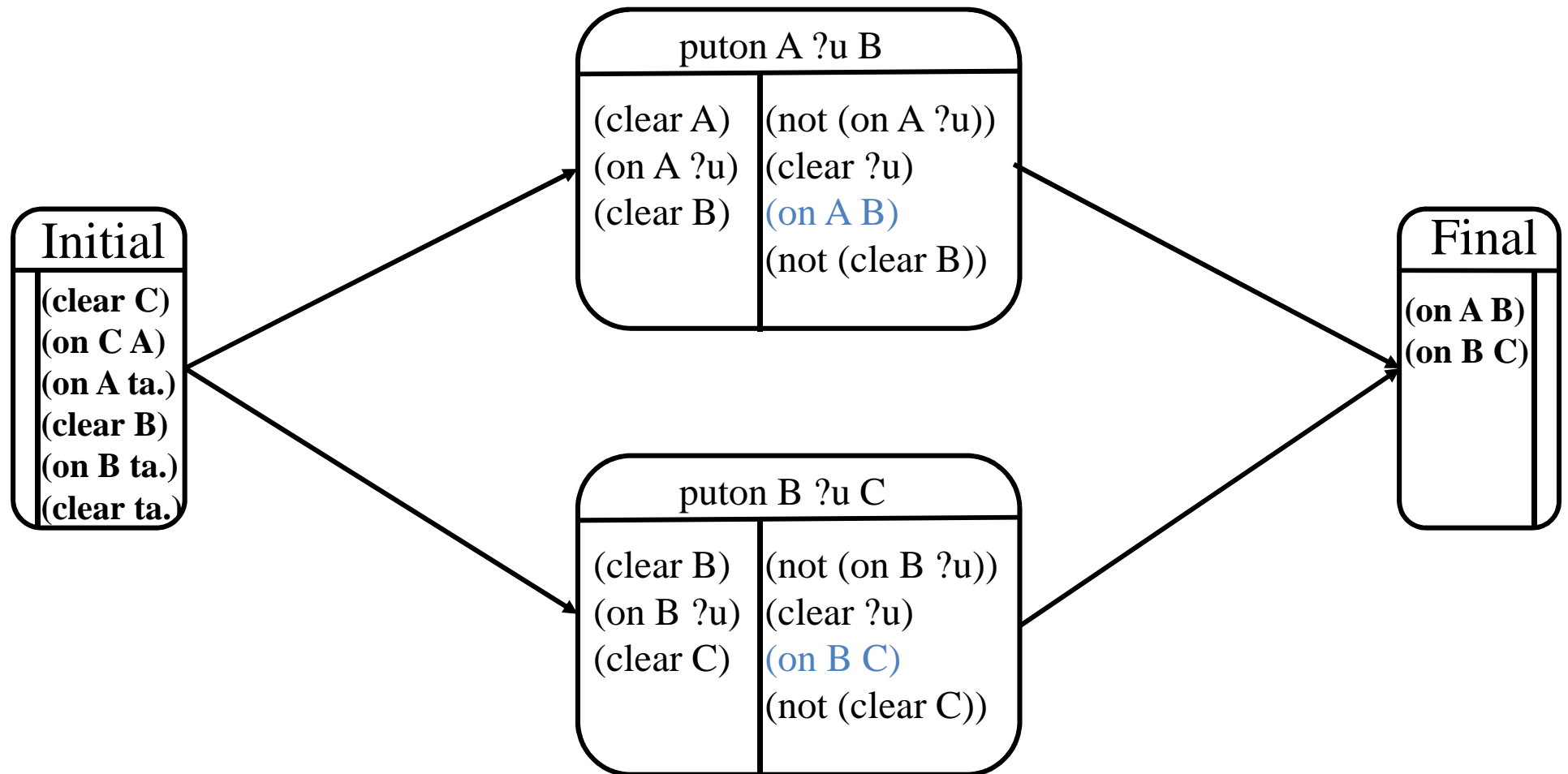


L'anomalie de Gerald Jay Sussman (3/16)



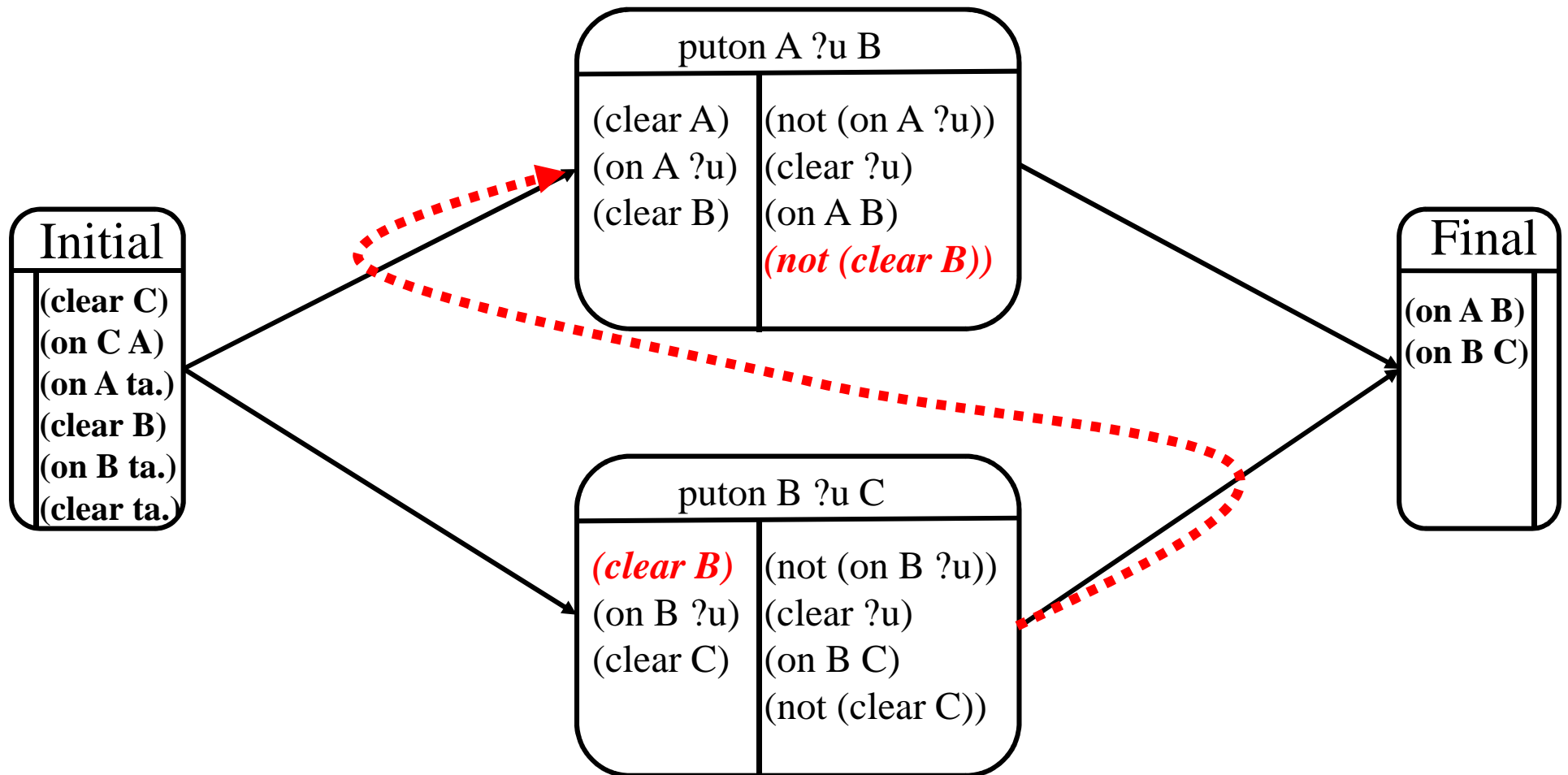
L'anomalie de Gerald Jay Sussman

(4/16)

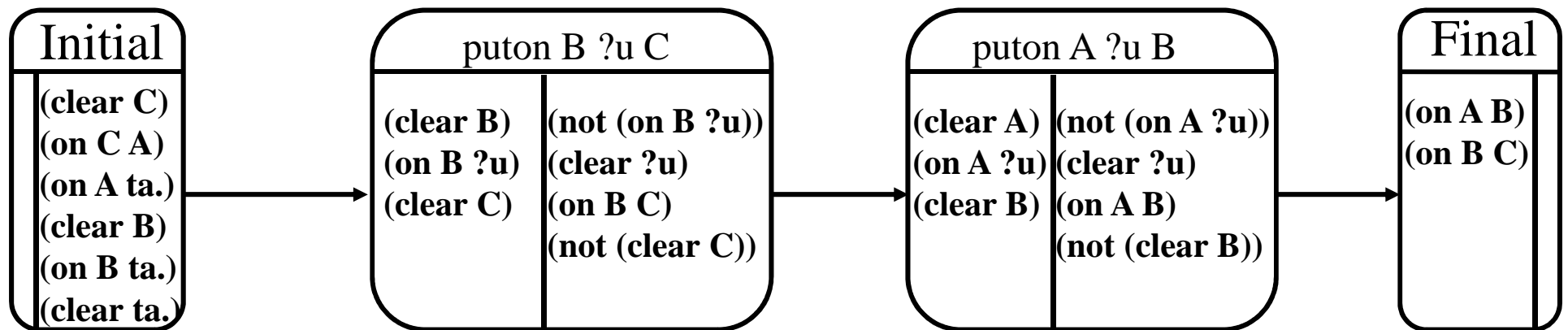


L'anomalie de Gerald Jay Sussman

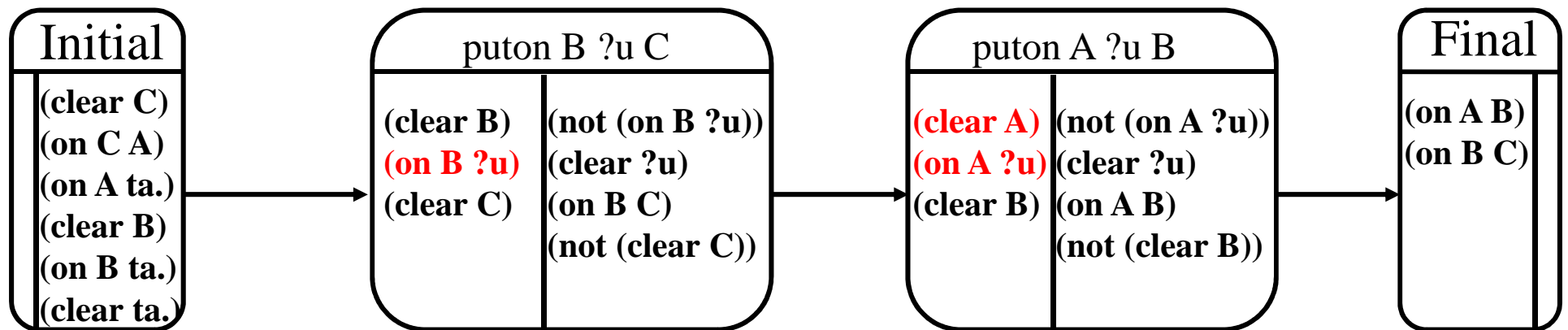
(5/16)



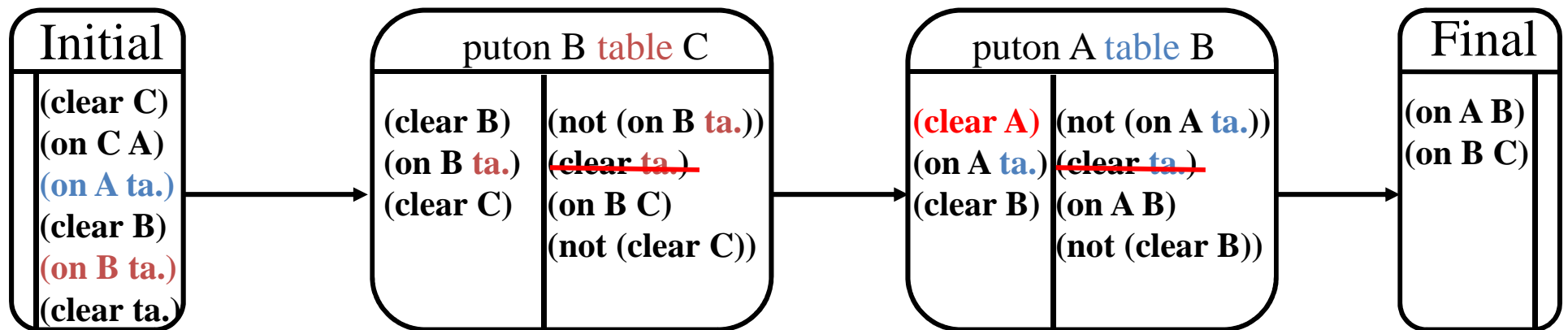
L'anomalie de Gerald Jay Sussman (6/16)



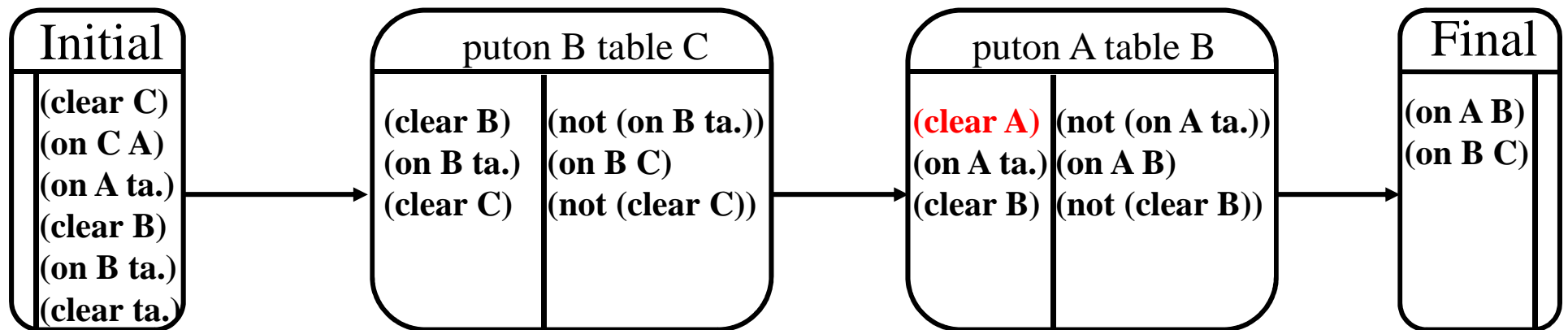
L'anomalie de Gerald Jay Sussman (7/16)



L'anomalie de Gerald Jay Sussman (8/16)

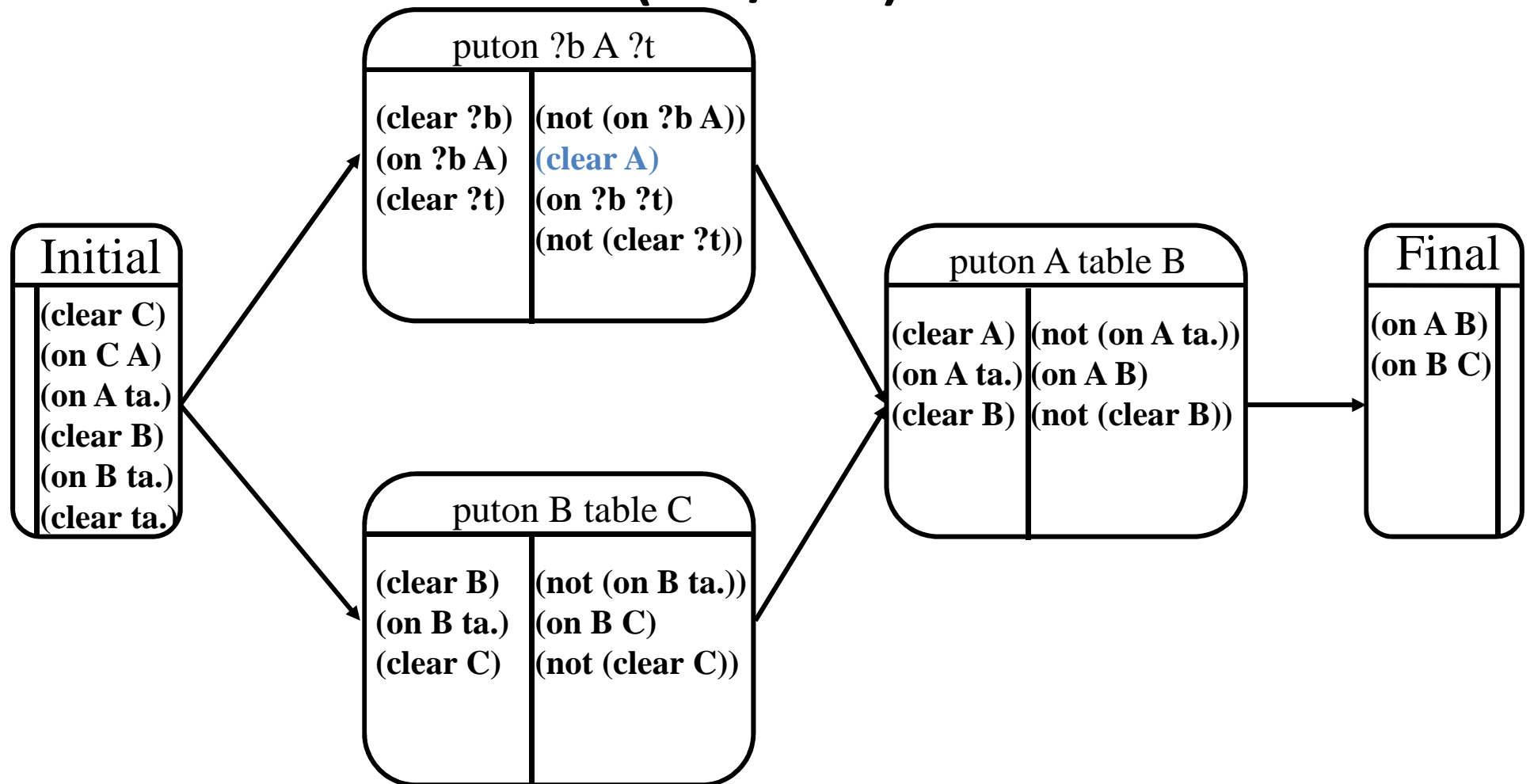


L'anomalie de Gerald Jay Sussman (9/16)



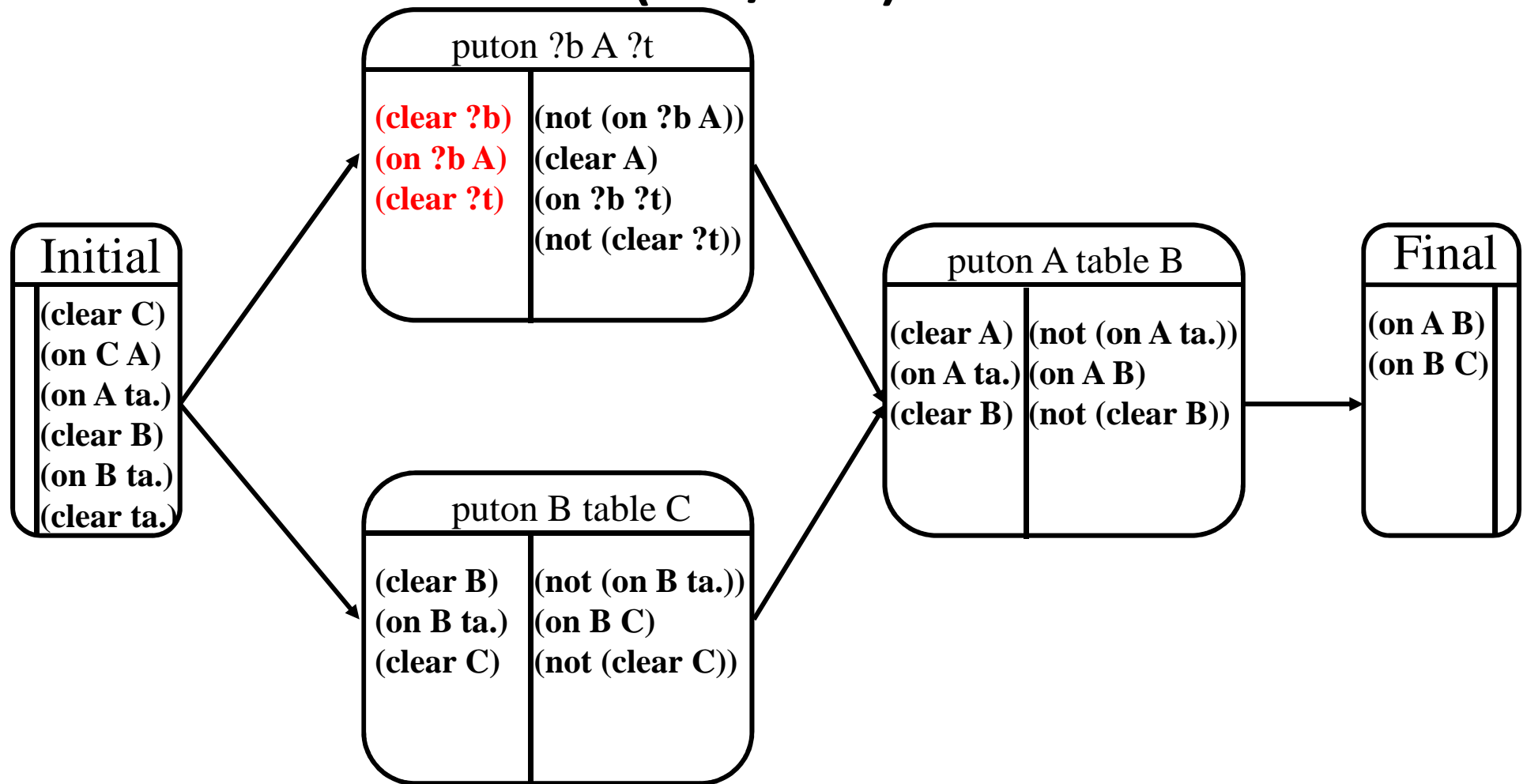
L'anomalie de Gerald Jay Sussman

(10/16)



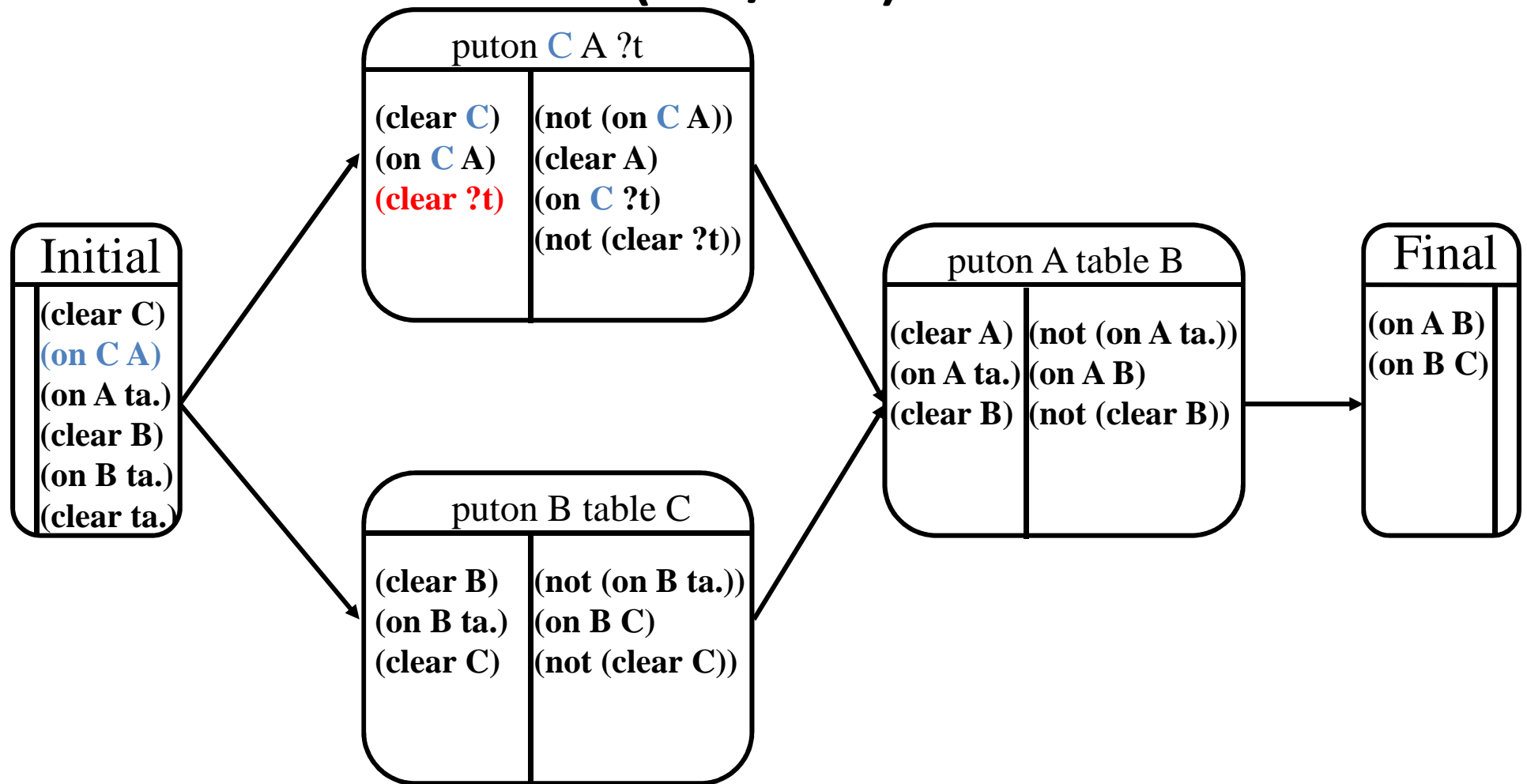
L'anomalie de Gerald Jay Sussman

(11/16)



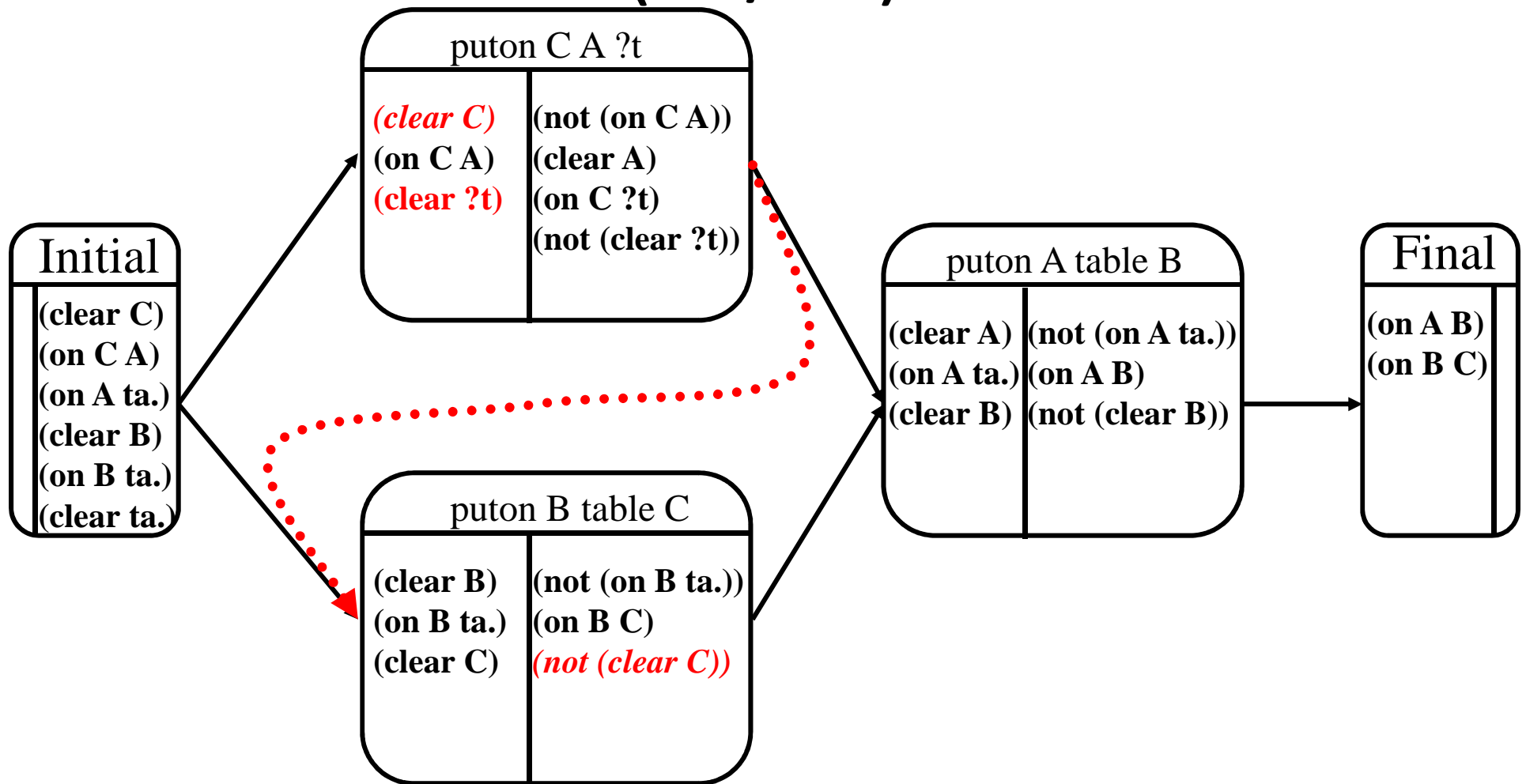
L'anomalie de Gerald Jay Sussman

(12/16)

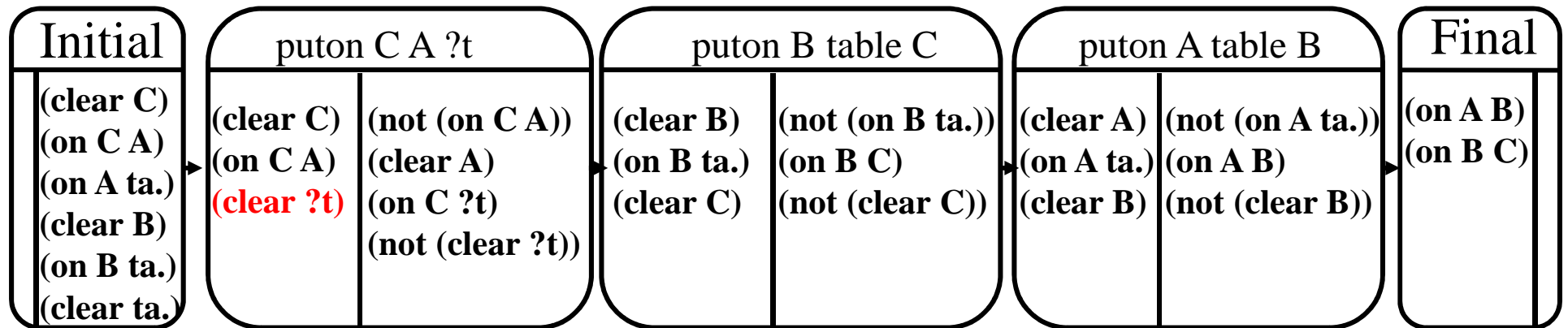


L'anomalie de Gerald Jay Sussman

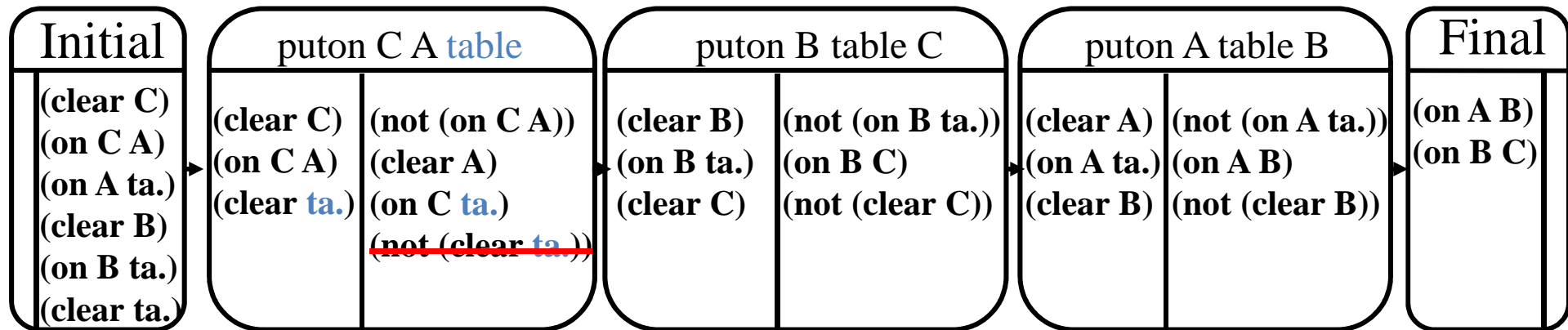
(13/16)



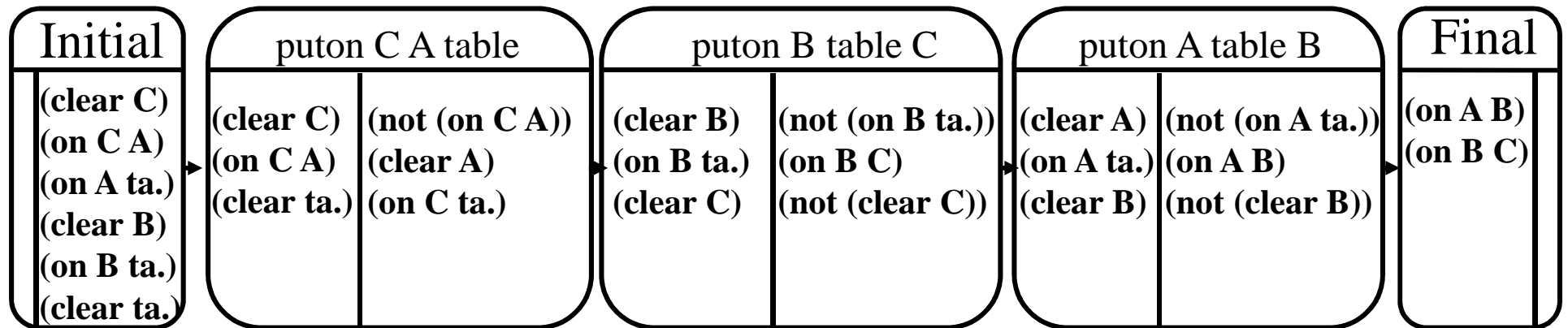
L'anomalie de Gerald Jay Sussman (14/16)



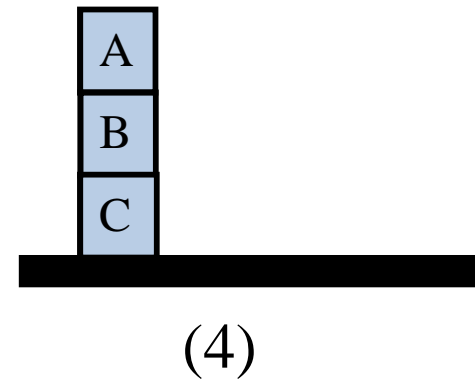
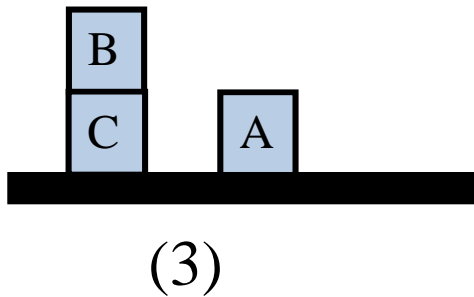
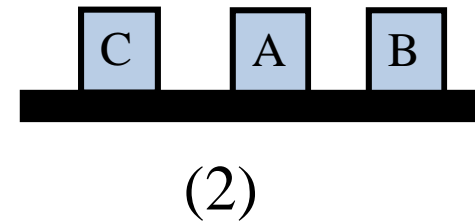
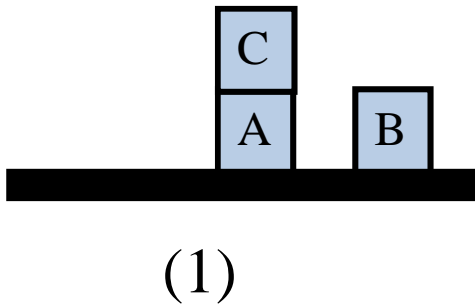
L'anomalie de Gerald Jay Sussman (15/16)



L'anomalie de Gerald Jay Sussman (16/16)



L'anomalie de Gerald Jay Sussman : solution



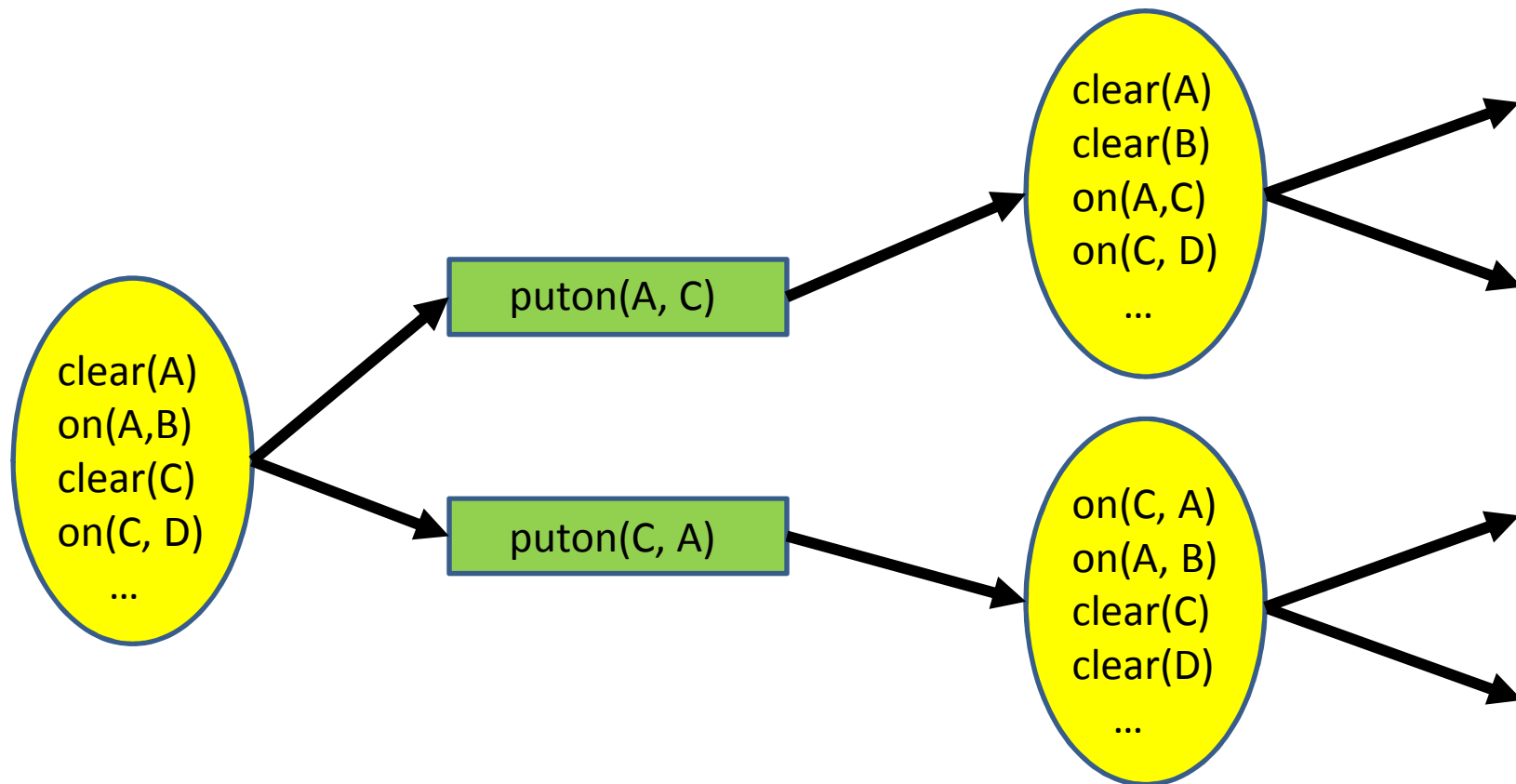
Algorithme dans un espace d'états :

Recherche avant (1 / 2)

- Un **état** S_i = un état de l'environnement.
 - S_0 = état initial = f_1, \dots, f_n .
 - Exemple : une configuration de cubes.
- **Successeurs(S_i)** = les N nouveaux états $\{ S_{i+1}, \dots, S_{i+N} \}$ 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, \dots, b_l .
- Algorithme : tout algorithme de recherche dans un espace d'états (par ex., A^*).

Algorithmes dans un espace d'états :

Recherche avant (2 / 2)



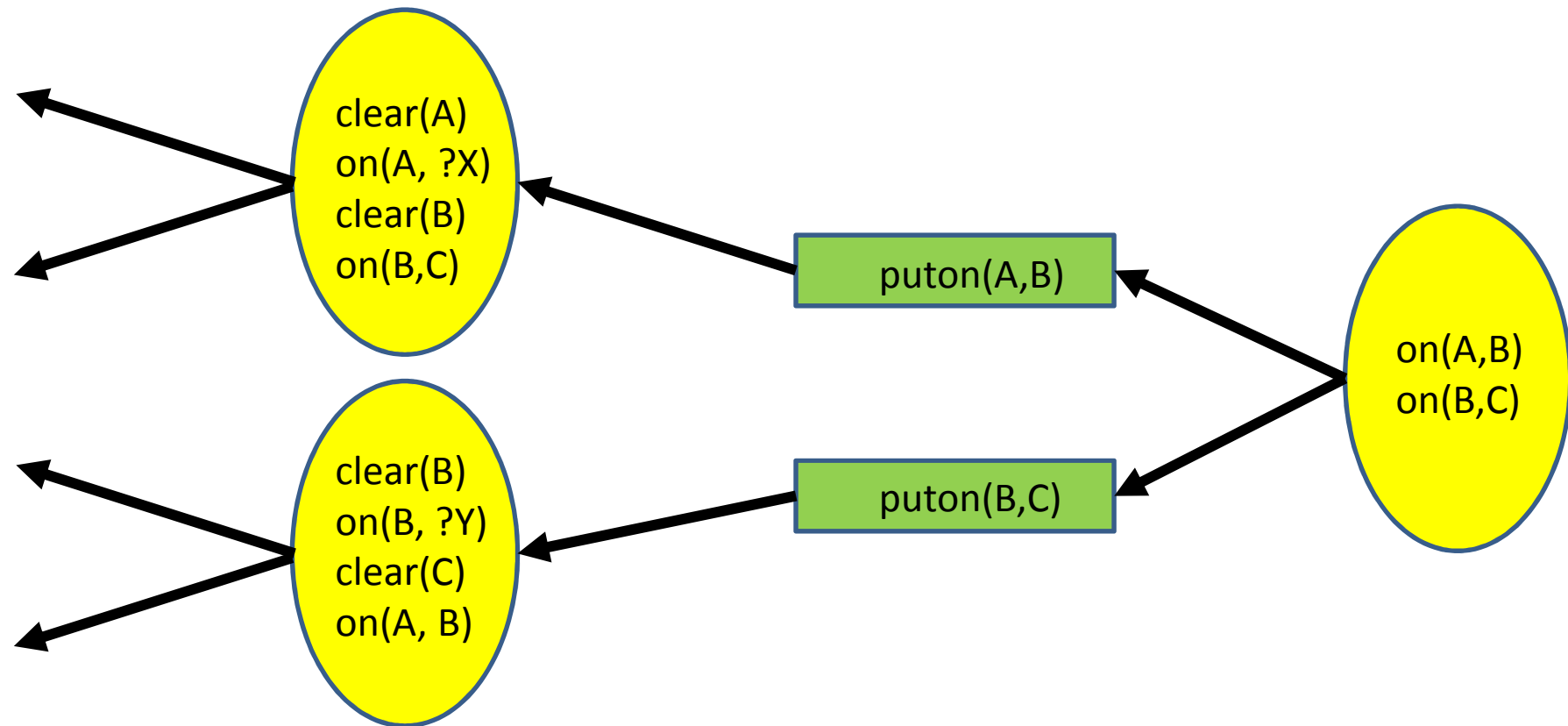
Algorithme dans un espace d'états :

Recherche arrière (1 / 2)

- Un **état** S_i = un état de l'environnement.
 - S_0 = les buts = b_1, \dots, b_l .
- **Prédécesseurs(S_i)** = N nouveaux états $\{ S_{i+1}, \dots, S_{i+N} \}$ obtenus en appliquant un opérateur en chaînage arrière à S_i .
 - Un opérateur est applicable ssi il possède une post-condition qui s'unifie avec un terme de S_i .
 - Ajouter les préconditions à S_i pour obtenir S_{i+j} .
- **bool Solution(S_i)** ssi S_i contient les littéraux f_1, \dots, f_n .
- Algorithme : tout algorithme de recherche dans un espace d'états (par ex., A^*).

Algorithme dans un espace d'états :

Recherche arrière (2 / 2)



Algorithmes dans un espace d'états :

Heuristiques

- Une fonction heuristique = une estimation de la distance à la solution + admissibilité.
- Trouver une bonne heuristique est difficile.
- Utiliser un problème relaxé : le coût optimal pour le problème relaxé est une heuristique du problème général.
 - Problème relaxé : Ne pas considérer les pré-conditions ; ne pas considérer les post-conditions négatives.
 - Exemple : buts = $A \wedge B \wedge C$
 $\text{Act}(X, \text{Eff.: } A \wedge P) \text{ Act}(Y, \text{Eff.: } B \wedge C \wedge Q) \text{ Act}(Z, \text{Eff.: } B \wedge P \wedge Q)$
Les actions X et Y suffisent (interaction positive), donc coût de 2.
Mieux que la résolution indépendante des 3 buts (coût 3).

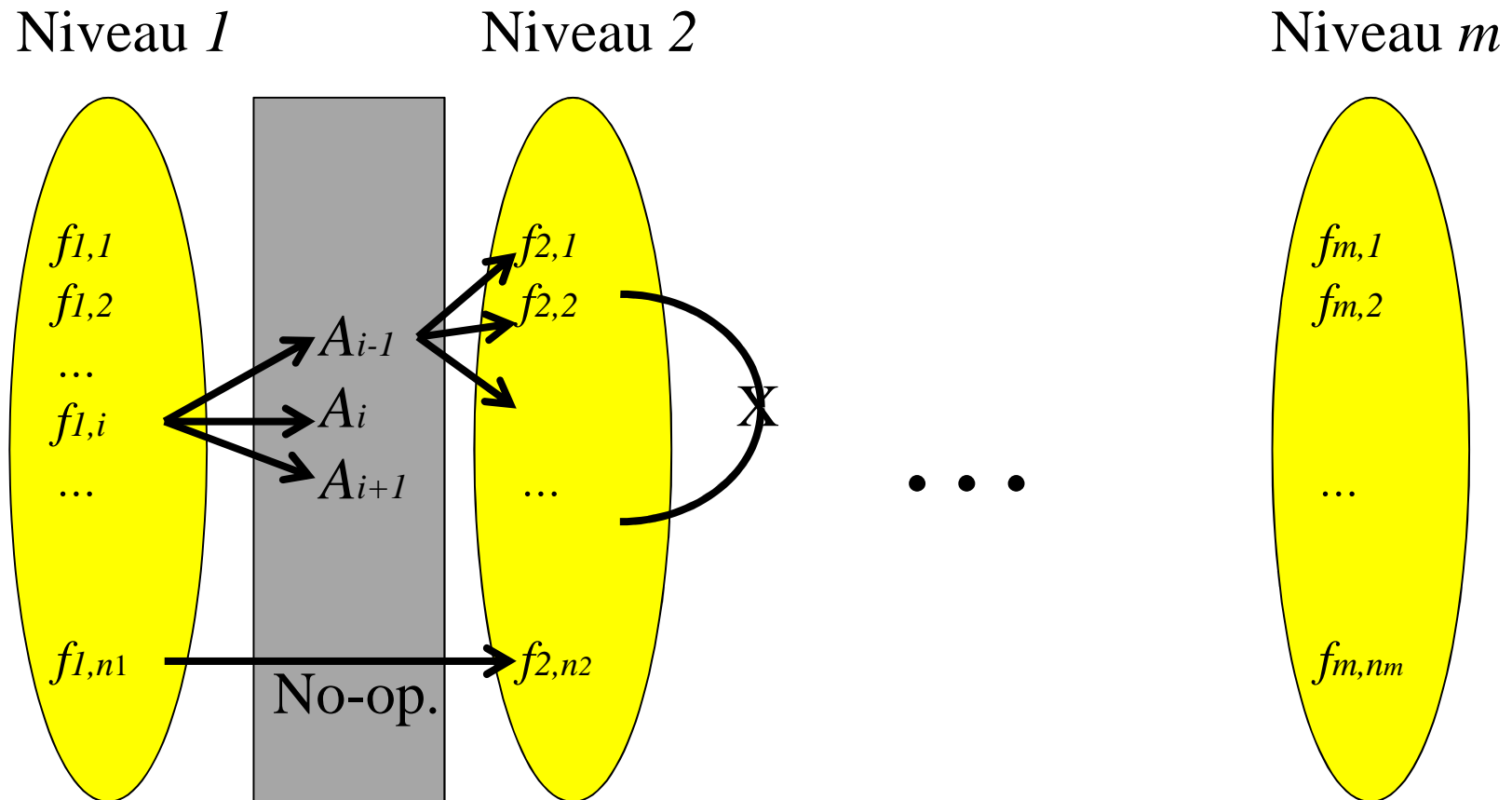
Algorithme dans un espace de plans partiels (1 / 2)

- Plan = (Descriptions T , Opérateurs Op , OrdrePartiel O , Unification U).
- Définitions :
 - Conflit : une post-condition qui peut détruire un lien causal (menace).
 - Satisfaire une pré-condition p : faire qu'il y ait un opérateur (ou pseudo opérateur initial) avant p , dont une post-condition q s'unifie avec p .
- $PLANIFICATEUR(T, Op., O, U)$:
TANTQUE conflit || au moins 1 pré-condition non satisfaite
 1. Résoudre les conflits ;
 2. Choisir une pré-condition p non satisfaite ;
 3. Satisfaire p :
 - Ajouter à U une contrainte d'unification / de non-unification;
 - Ajouter à O une contrainte de précédence ;
 - Ajouter à Op un opérateur de T partiellement instantié.

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

- Un état = un plan partiellement ordonné et partiellement instantié.
- *PLANIFICATEUR*(*T*, *Op.*, *O*, *U*) :
Algorithme de recherche dans un espace d'états (par ex., A^*) :
 - Un **état** = un plan partiel (*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 && toutes les pré-conditions sont satisfaites.
- Heuristique : nombre de préconditions non satisfaites, nombre de conflits

Plan de graphe (1 / 4)



- Niveaux avec *mutex* (relations d'exclusions mutuelles) : un niveau n'est pas un état.
- Plan de graphe avant, et recherche arrière.

Plan de graphe (2 / 4)

- Structure de données :
 - Plan de graphe = termes et opérateurs en niveaux alternés
 - No-op
- Algorithme GRAPHLAN(S_0 , Buts, T)
 - A. Poser le niveau initial égal à S_0
 - B. TANT QUE (!plan-solution || $S_i \neq S_{i-1}$)
 1. 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.
 2. Expansion du plan de graphe sur 1 niveau
 - a) Développer toutes les actions possibles
 - b) Relations *mutex*

Plan de graphe (3 / 4)

- Relations d'exclusion mutuelle (*mutex*) entre opérateurs :
 - Effets inconsistants : un opérateur possède un effet qui nie un effet d'un autre opérateur.
 - Interférence : un effet d'un opérateur est la négation d'une précondition d'un autre opérateur.
 - Besoins en compétition : une précondition d'un opérateur est mutex avec une précondition d'un autre opérateur.
- Mutex entre littéraux de même niveau :
 - L'un est la négation de l'autre.
 - Toute paire d'action possible, qui les établit, est mutex.

Plan de graphe (4 / 4)

- Recherche en arrière d'un plan-solution dans le plan de graphe :
 - Une recherche dans un espace d'états (ou CSP dynamique).
 - Etat initial = niveau S_n avec les buts
 - Prédécesseurs = choisir un sous-ensemble d'actions dans A_{i-1} sans conflit qui satisfasse les buts de S_i .
 - But = arriver à S_0 avec tous les buts satisfaits.

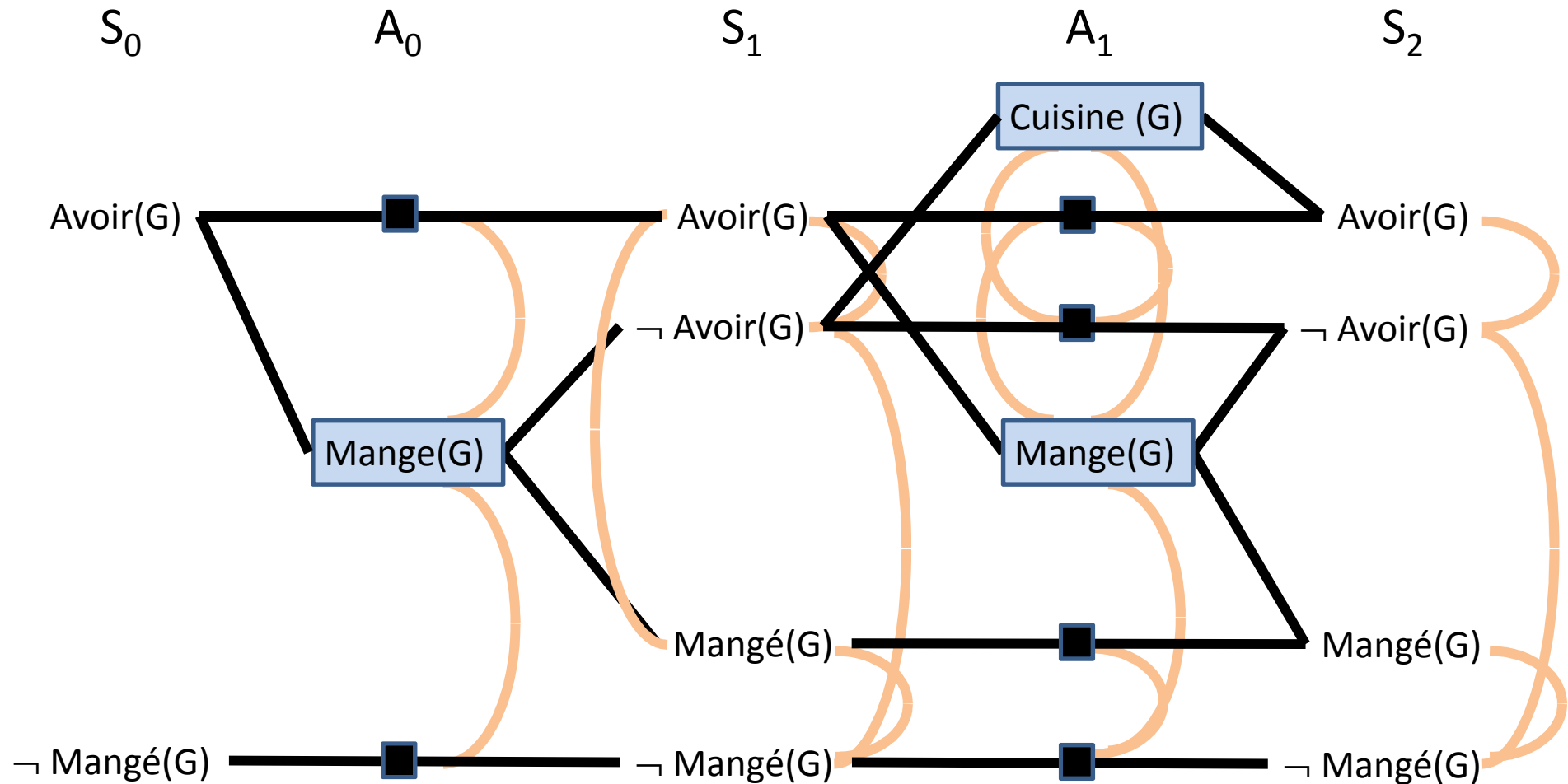
Plan de graphe

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

Terminaison

- Le nombre de littéraux par niveau s'accroît de façon monotone.
- Le nombre d'actions par niveau s'accroît de façon monotone.
- Le nombre de mutex par niveau décroît de façon monotone.
- Conclusion : éventuellement retrouver le même niveau deux fois ...

Planificateurs par solver SAT (1 / 4)

- Solvers SAT (rappel) :

- Problème de la satisfiabilité d'une formule logique : « *Trouver la valeur de vérité de chaque proposition qui, ensemble, satisfont une formule logique donnée* ».
- Exemple de formule : $(p \wedge q) \vee (r \wedge \neg s \wedge t) \vee (u \wedge v \wedge \neg w)$
- Logique des propositions.
- 3SAT est le 1^{er} problème prouvé NP-complet (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)

- 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 , ALLER-EN 2.

Planificateurs par solver SAT (3 / 4)

- Dans le monde des cubes, pour l'anomalie de Gerald Sussman :
 - Buts : $\text{on}(A,B)@T \wedge \text{on}(B,C)@T$
 - Etat initial : $\text{clear}(C)@0 \wedge \text{on}(C,A)@0 \wedge \text{clear}(B)@0$
 $(\wedge \neg \text{on}(A,C)@0 \wedge \neg \text{on}(A,B)@0 \wedge \neg \text{on}(B,C)@0 \wedge \neg \text{on}(B,A)@0$
 $\wedge \neg \text{on}(C,B)@0 \wedge \neg \text{clear}(A)@0)$ // *hypothèse du monde fermé*
 - Une proposition par opérateur instantié :
 $\forall x, \forall y, \forall z, \forall t :$
 $\text{on}(x,y)@t \wedge \text{clear}(x)@t \wedge \text{clear}(z)@t \wedge \text{puton}(x,y,z)@t \Rightarrow \text{clear}(y)@t+1 \wedge \text{on}(x,z)@t+1$
 - Axiomes sur les préconditions :
 $\forall x, \forall y, \forall z, \forall t :$
 $\text{puton}(x, y, z)@t \Rightarrow \text{on}(x,y)@t \wedge \text{clear}(x)@t \wedge \text{clear}(z)@t$
 - Un seul opérateur à la fois (contraintes d'état) :
 $\forall x, \forall y, \forall y', \forall z, \forall z', \forall t / y \neq y' \wedge z \neq z' :$
 $\neg (\text{puton}(x, y, z)@t \wedge \text{puton}(x, y', z')@t)$

Planificateurs par solver SAT (4 / 4)

- Complexité de cet encodage : $T \times |Act| \times |O|^P$ 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 2^e encodage : $T \times |Act| \times |O|$

Planificateurs par PPC (1 / 3)

- Programmation par contraintes (rappel) : variables, domaines, contraintes + retour-arrière, consistance d'arc.
- Principe :
 1. Estimer heuristiquement la longueur n du plan-solution
 2. Transformer le problème de planification en un CSP
 3. Résoudre cette formulation avec un solveur de CSP
 4. Si le solveur de CSP échoue, ALORS incrémenter n , 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)

- Canonicité : un opérateur instantié apparaît au plus une fois.
- Pseudo opérateurs : *Start* (sans pré-condition) et *End* (sans post-condition).
- Heuristique pour estimer la durée **B** du plan-solution.
- Puisqu'un plan-solution est linéaire, numéroter les instants.
- Actions duratives.
- Déclarer des contraintes redondantes
- Variables du CSP :
 - Date de début de l'opérateur o : $T(o) \in [0 ; B[$
 - Support de la précondition p dans l'opérateur o : $S(p, o) \in O(p)$
 - Date de début du $S(p, o)$: $T(p, o) \in [0 ; B[$
 - $InPlan(o) \in \{0, 1\}$ indique la présence de l'opérateur o dans le plan
 $InPlan(Start) = 1, InPlan(End) = 1$.

Planificateurs par PPC (3 / 3)

Contraintes

- Bornes : $T(Start) + dist(Start, o) \leq T(o)$ $T(o) + dur(o) + dist(o, End) \leq T(End)$
- Contraintes de support :

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

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

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

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

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

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

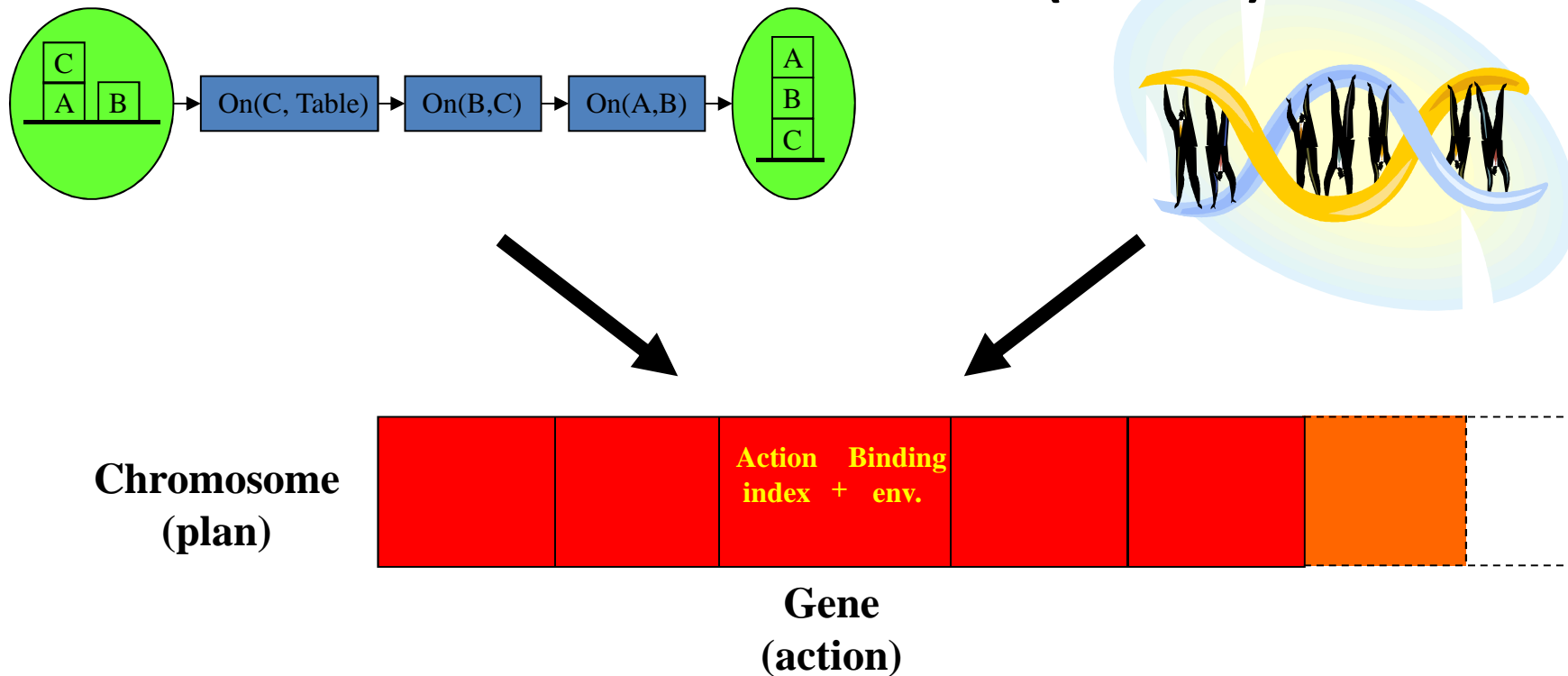
$$T(o) + dur(o) + dist(o, o') \leq T(o')$$
- Mutex : si les opérateurs o et o' sont mutex, pas de parallélisme

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

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 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 « bugs » dans l'individu-plan.

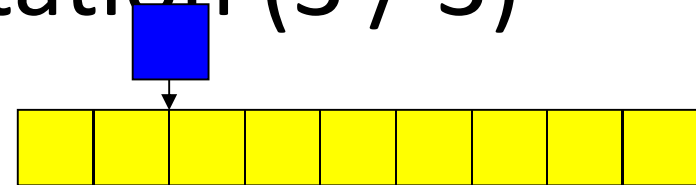
Planificateurs par algorithmes évolutionnaires (2 / 3)



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

Planificateurs par algorithmes évolutifs : mutation (3 / 3)

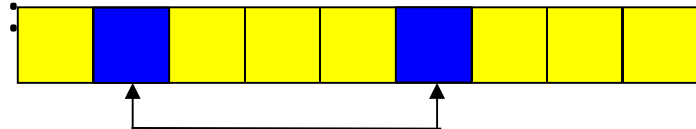
- Ajout aléatoire de gène :



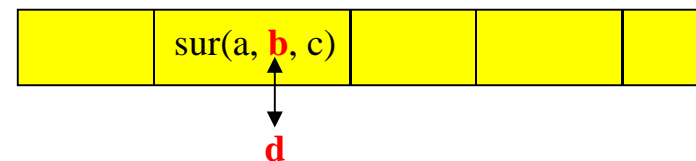
- Retrait aléatoire de gène :



- Permutation aléatoire de deux gènes :

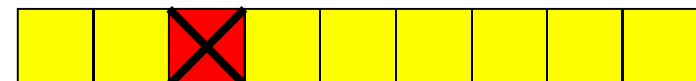


- Remplacement d'un gène :

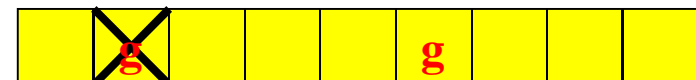


- Mutations heuristiques :

- Retrait d'un gène conflictuel :



- Retrait d'un gène dupliqué :



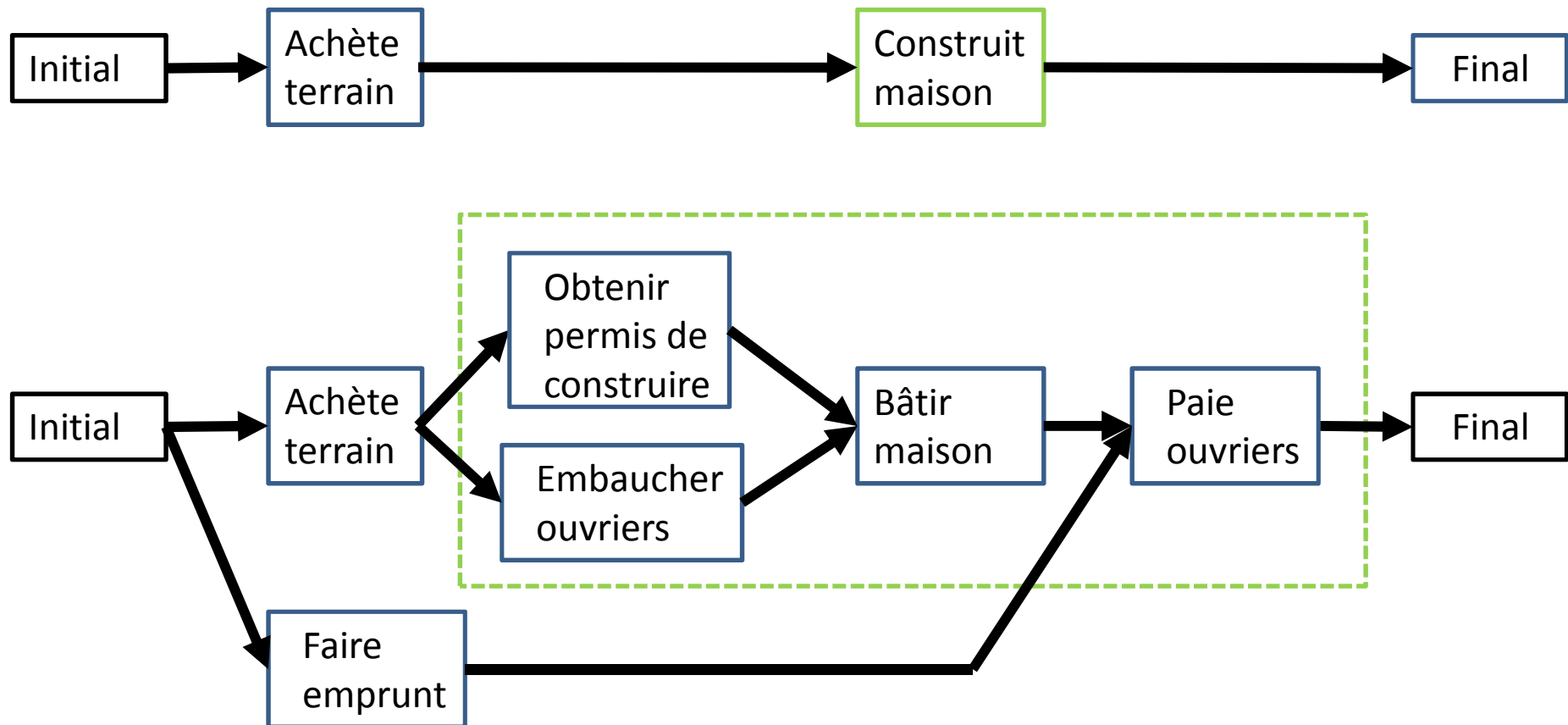
Réseaux de tâches hiérarchiques

(1 / 2)

- 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
- Planifier à un niveau de détails donné, se préoccuper des détails ensuite. Ne pas planifier d'entrée de jeu au plus grand niveau de détail, comme tous les autres planificateurs.
 - Diminuer la complexité.
- 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)



Planification & Exécution



Références

1. Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010, 3rd edition. Chapitre 11.
2. Malik Ghallab, Dana Nau, Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, San Mateo, CA, May 04, 635 pages.
3. Tutorials et cours sur le web (e.g., Joerg Hoffmann).

Conclusion

- La planification d'actions consiste à trouver une séquence d'opérateurs instantiés (un plan) menant un état initial à des buts.
 - Difficile parce que interaction entre sous buts.
- Les opérateurs sont exprimés en STRIPS / ADL / PDDL.
- Plusieurs types d'algorithmes pour construire un planificateur d'actions :
 - Dans l'espace des états (progression, régression).
 - 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.
- Non vus : planification probabiliste, planification conditionnelle, PERT, planification par logique modale (Patrick Doherty), FF et successeurs (Joerg Hoffmann).