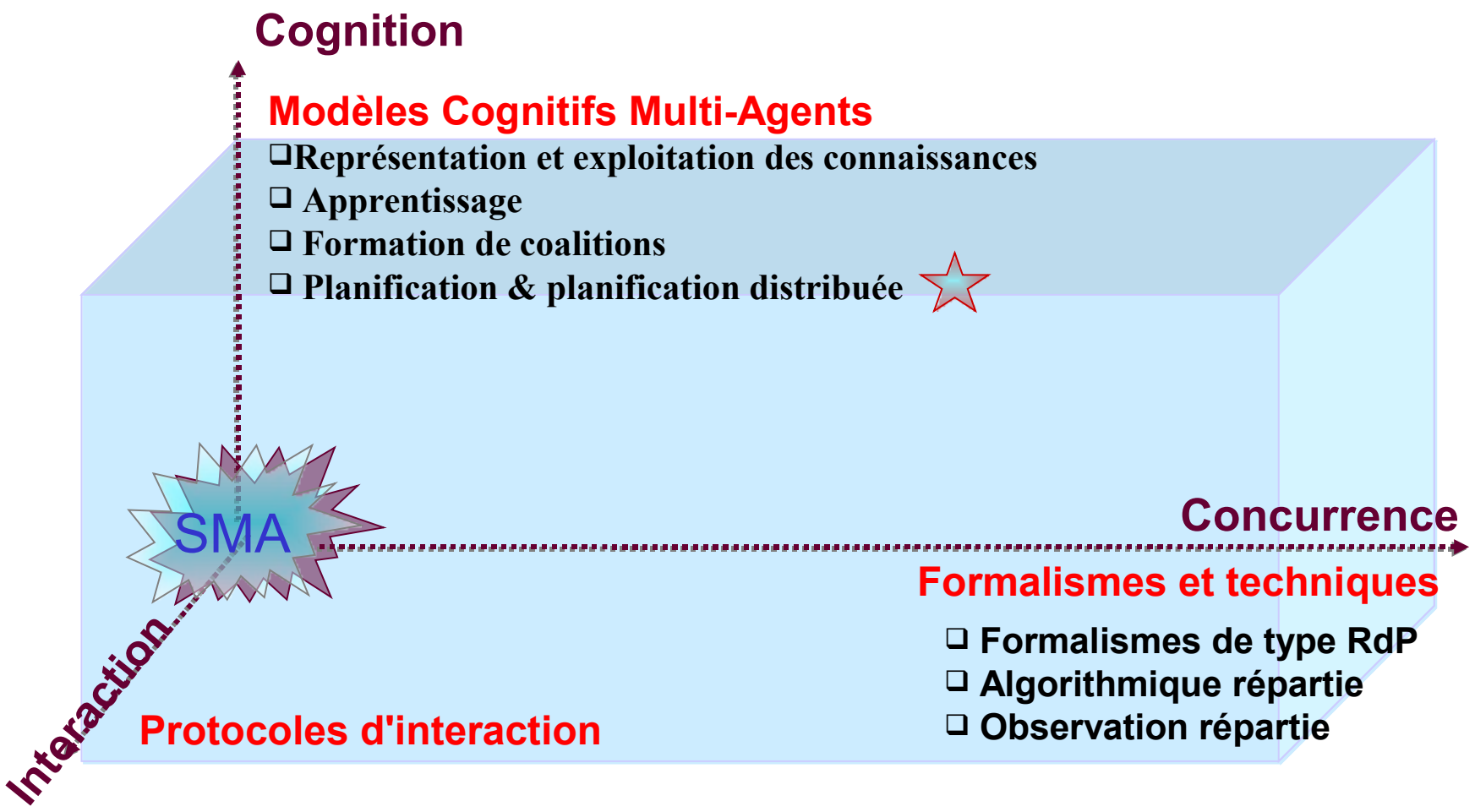
SMA et techniques

Tarek Melliti
Laboratoire IBISC
(Informatique Biologie Intégrative et Systèmes Complexes)
tarek.melliti@ ibisc.univ-evry.fr

SMA et techniques : Les trois dimensions [Amal el Fallah]

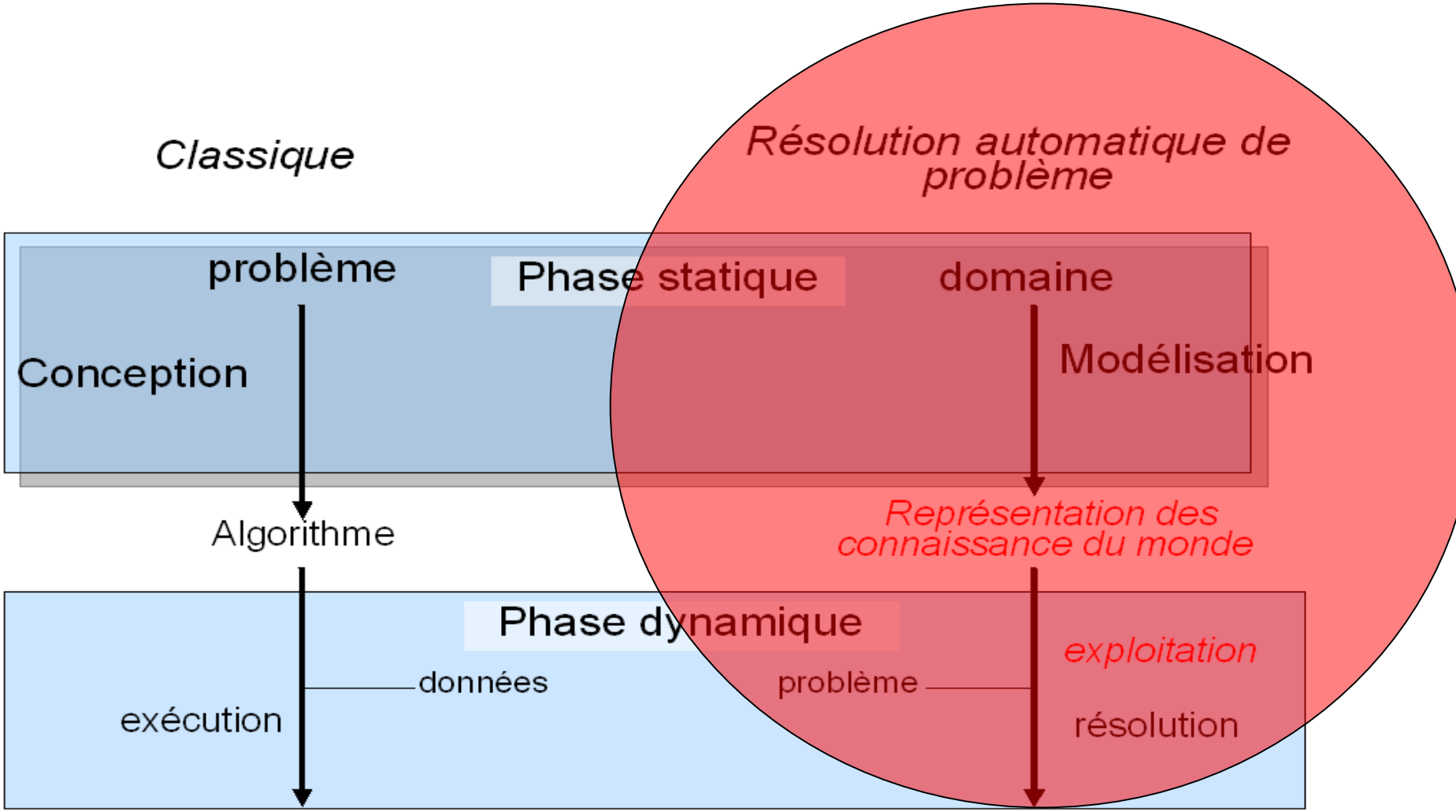


- ❑ Modélisation cognitive des interactions
- ❑ Ingénierie des protocoles d'interaction

Planification : cas d'un mono-agent

Tarek Melliti
Laboratoire IBISC
(Informatique Biologie Intégrative et Systèmes Complexes)
tarek.melliti@ ibisc.univ-evry.fr

Démarche générale pour la résolution automatique de problème (1)



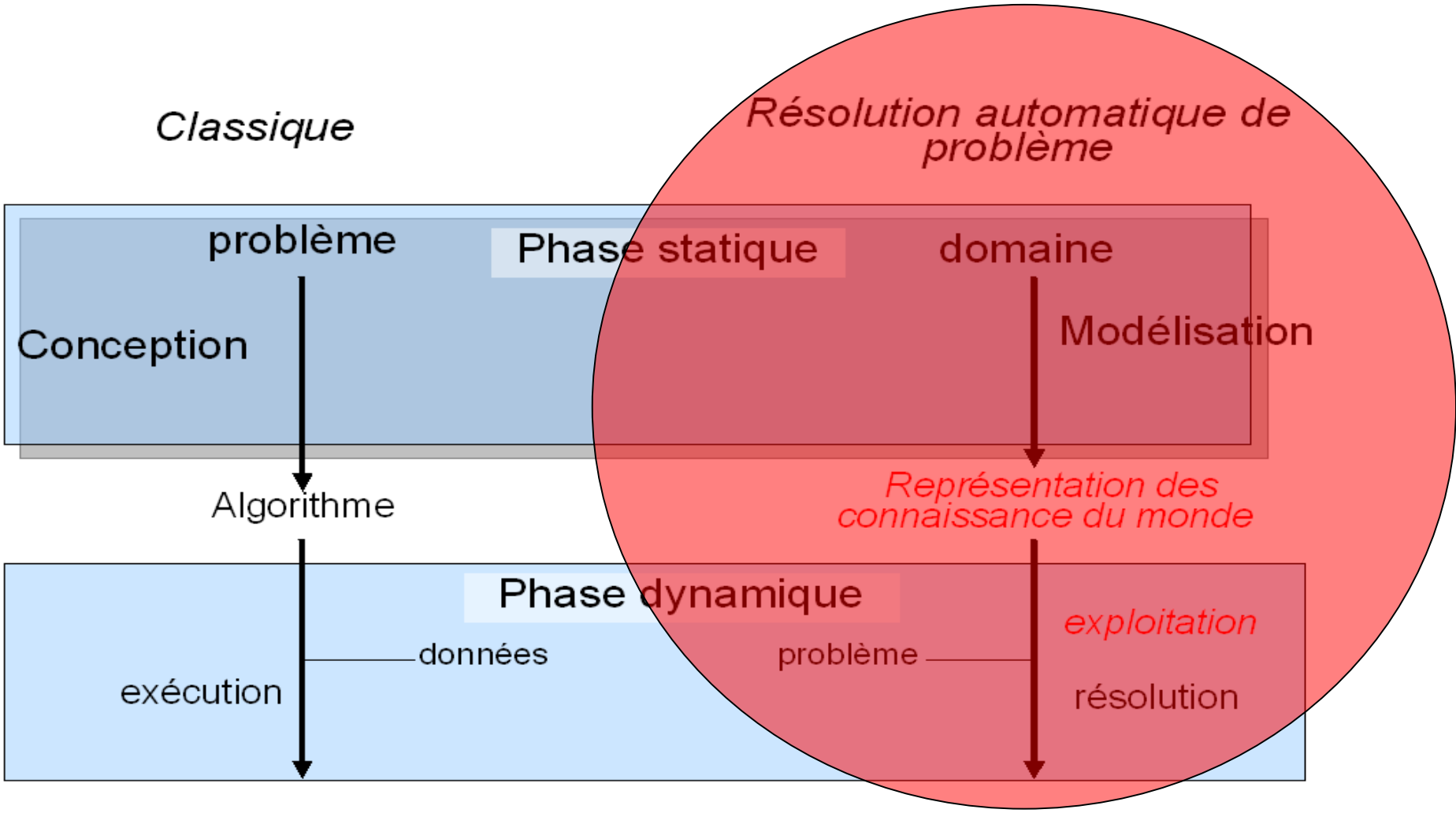
Ce qu'on a vu et ce qui reste à voir

- Dans cette partie du cours on va voir deux types d'approches de résolution pour deux classes de problèmes différents.

- Le premier : Systèmes experts
  - Type de problème: classification, diagnostic, réponse à des questions.
  - Exemple: est ce que la “baleine bleue” est un mammifère ?

- Le deuxième : planification
  - Définir la suite d'actions pour atteindre un état donné
  - Exemple: comment aller à la fac ?

Démarche générale pour la résolution automatique de problème (1)

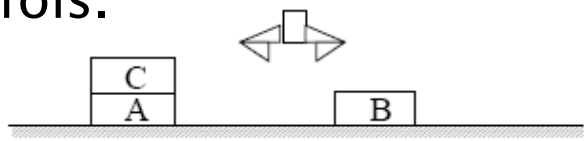


Problématique de la planification

- Contrairement à un système Expert
  - La planification n'a pas pour but la production de nouvelles connaissances à partir de règles de déduction.
  - Est ce que x est un Y ?
  - Que peut on déduire de X et Y sachant ....???
- Un système de planification :
  - se situe dans un monde, le fait évoluer par des actions,
  - agit en fonction d'un ou plusieurs buts exprimés comme des propriétés de l'état du monde.

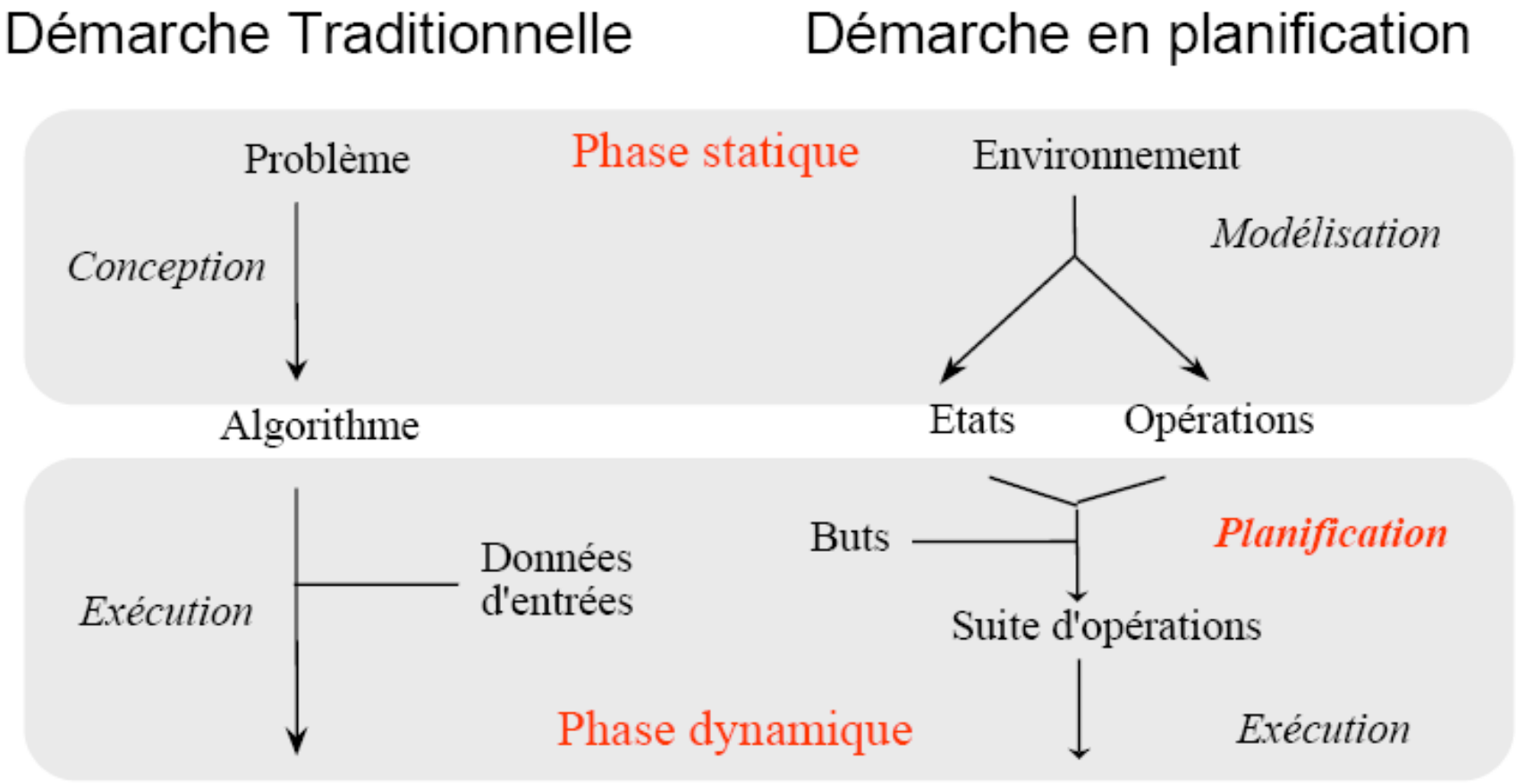
Un exemple : le monde des cubes

- Définition du monde des cubes :
  - L'univers est composé d'un ensemble de blocs cubiques et d'une table.
  - Les blocs sont mobiles, la table est immobile.
  - Un agent effectue des actions primitives sur les blocs pour changer leur état.
  - Cet agent est un bras avec une pince mobile.
  - Un bloc peut reposer sur la table, sur un autre bloc ou être dans la pince.
  - Il ne peut pas y avoir plus d'un bloc sur un autre bloc.
  - La table est assez grande pour que tous les blocs puissent y prendre place
  - La pince ne peut déplacer qu'un bloc à la fois.



Démarche de la planification (2)

- La planification entre dans le cadre des méthodes de résolution automatique de problème



Les questions aux quelles il faut répondre

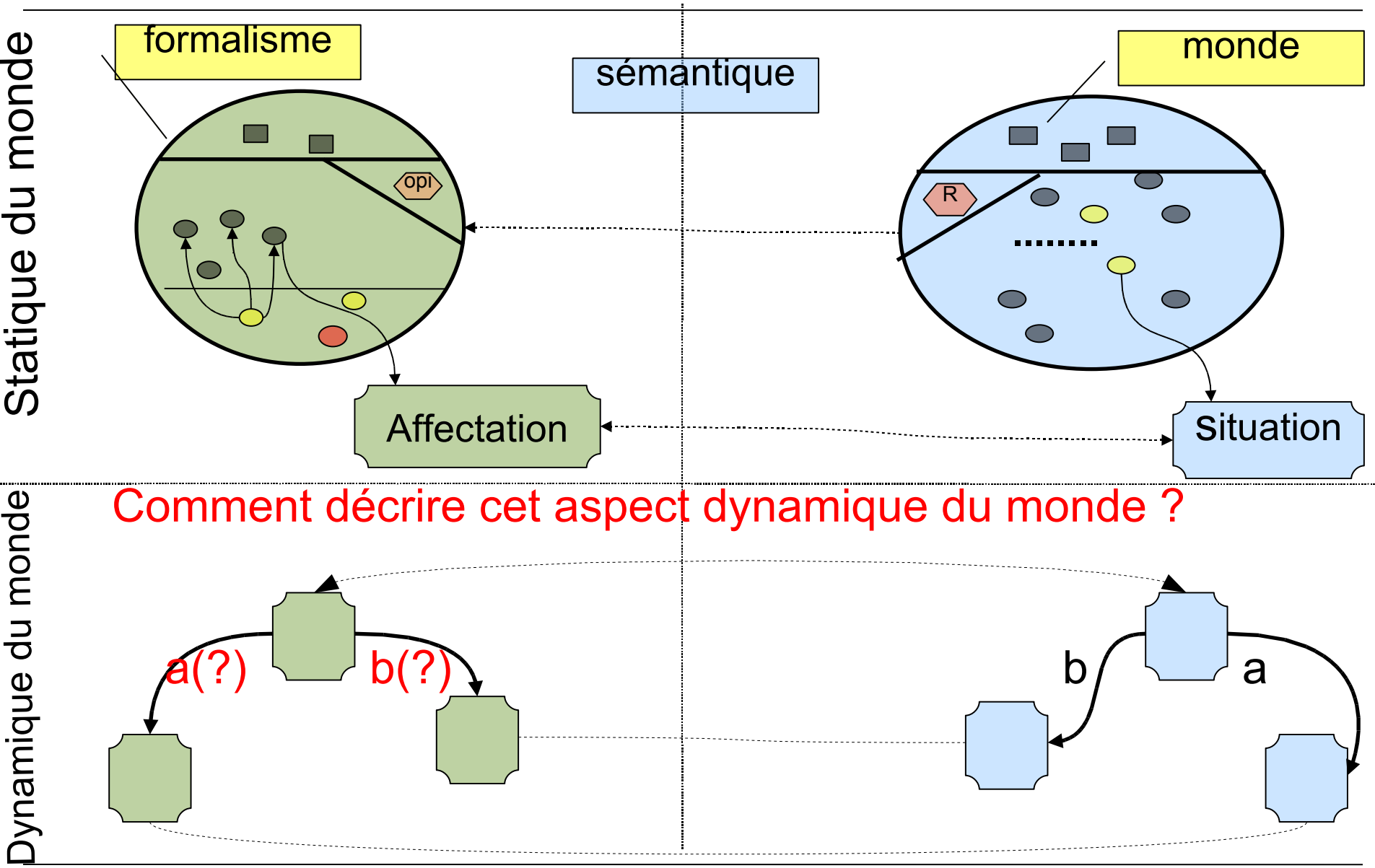
1) La représentation du monde et des capacités de l'agent
   - les actions, les plans, le changement, le temps, les
   - objectifs à atteindre

2) L'algorithmique :
   - comment réaliser une synthèse automatique de plans
   - à partir d'un but à atteindre et d'un état du monde

3) Le suivi de l'exécution
   - le contrôle, la réactivité aux évolutions, la révision des plans déjà produits

La représentation

Représentation de connaissances pour la planification



Un monde dynamique ?

- Un monde
  - On sait le faire avec un formalisme (syntaxe et sémantique)
  - Pour décrire une situation particulière
  - Une formule ou phrase du formalisme + une affectation
  - Description d'une situation d'un monde decrit par une logique propositionnelle :

    $$il\_fait\_beau\{Vrai\} \wedge bonne\_humeur\{Vrai\}$$

- Des actions:
  - Transforme un état en un autre état
  - Rajouter de nouvelles formules et changer les affections
  - **Une façon de voir consiste à considérer une action comme une règle**
    - Intérêt évident : on connait les règles
    - Executer une action est alors réduit à l'application d'une ou plusieurs règles de déduction du formalisme considéré

Action comme un ou +ieurs règles de déduction : exemple

- table_case_Milieu, table_case_Gauche, Table_case_Droit
- Déplacer vers la gauche la table est une action qui déplace la table vers la case à gauche de sa case actuelle
- Sachant que la table est actuellement au milieu :
  - *table_case_Milieu*
- Et que j'ai déplacé la table vers la gauche
- Quelle est sa position?
  - *table_case_Gauche*

- Sachant que ma théorie ou mon monde est :
  - A
  - A->B
- Et que j'ai appliqué l'action de déduction *modus ponens*
- Qu'elle est l'état de mon monde?

Mais ..... la monotonie?

- Ce qu'on a vu jusque là
- Dans notre démarche de preuve et de déduction
- e.g
  - A
  - Ai
  - Application du *modus ponens* produire Aj

  => Ai reste valide
- Dans notre exemple précédent que devient table_au_Milieu

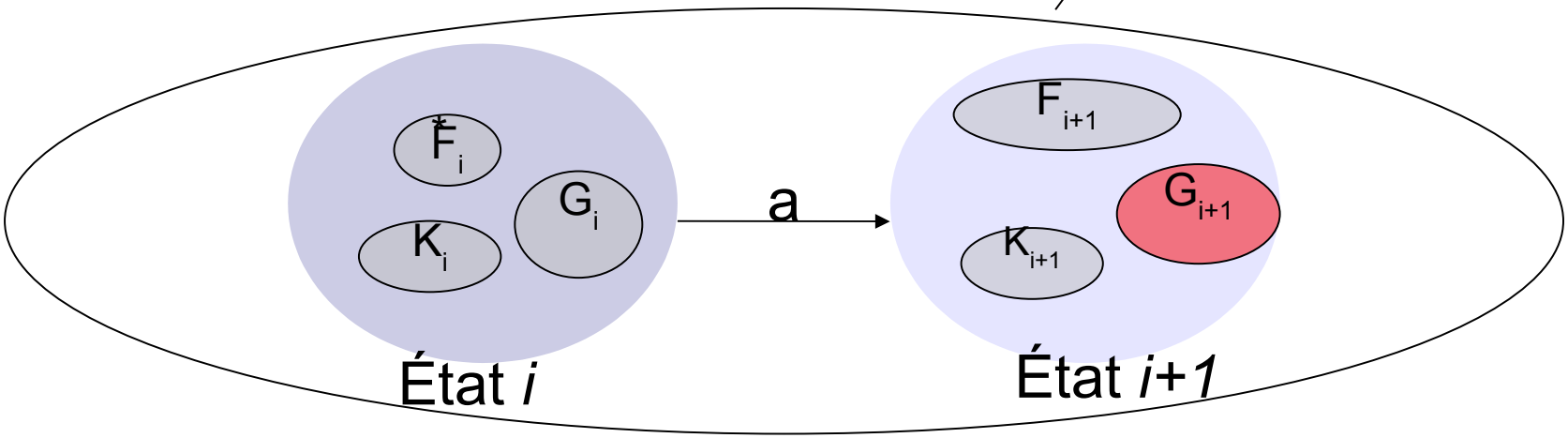
Les états du monde : le problème de contexte

- Avant –dans le cadre du raisonnement– on se donnait un état du monde sur le quel on veut inférer de nouvelles connaissances :
  - Divise les mondes (les formules) en :
    - Celles à qui on peut affirmer la valeur de vérité (vrai et faux)
    - Celles dont la valeur de vérité est inconnu
- En introduisant la dynamique:
  - Les valeurs de vérité d'une formule n'est pas universelle mais dépendent de l'état.
- Comment représenter les états du monde dans le système déductif ?

Deux possibilité: expliciter les états

- Représentation explicite des états
  - F_i : la formule *F* dans la situation *i*
  - *Ici du monde i au i+1, y a que G qui chage de valeur*
  - *À chaque monde on décrit ce qui change mais aussi ce qui change pas.*

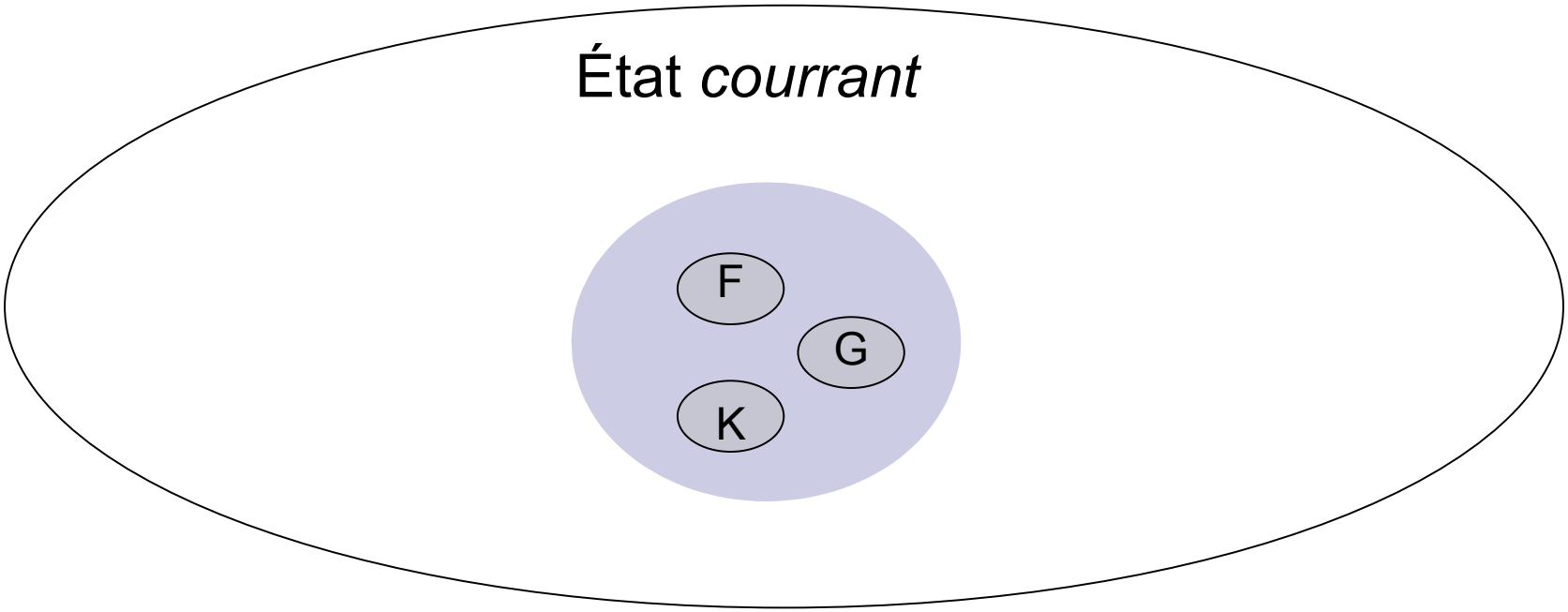
Très coûteuse



On reste dans un système déductif Monotone :
une formule garde toujours sa valeur de vérité

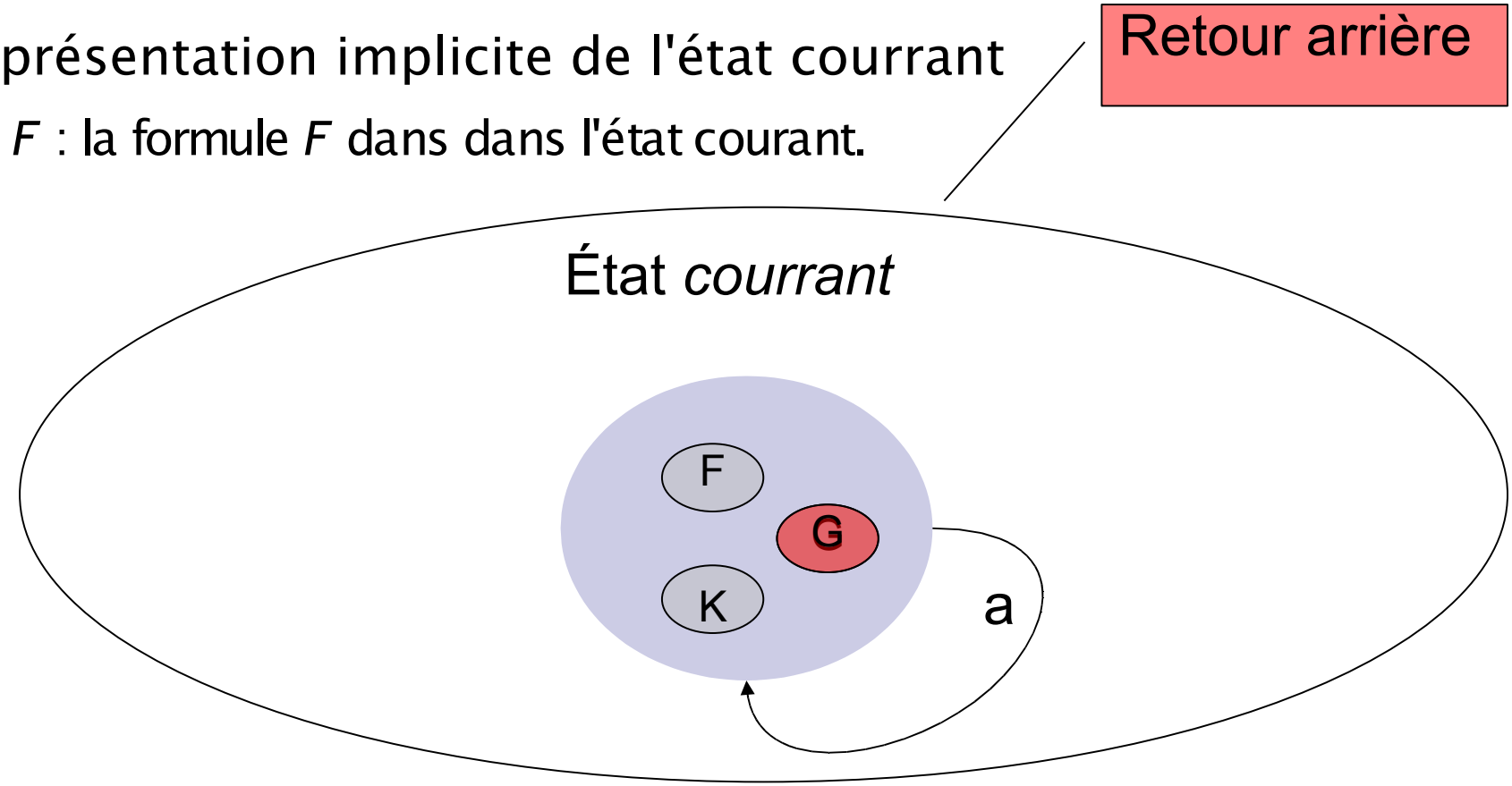
Deux possibilités : explicité les états

- Représentation implicite de l'état courrant
  - *F* : la formule *F* dans dans l'état courant.



Deux possibilités : explicité les états

- Représentation implicite de l'état courrant
  - *F* : la formule *F* dans dans l'état courant.



Représentation à la STRIPS :

- C'est le formalisme considéré dans ce cours
- Utilise la deuxième approche
- Représentation du monde
  - Les formules décrivent le monde courant
  - Des mecanismes de déduction classique sur le monde courant
- Représentation des actions
  - Comme des règles:
  - Quel sera l'état du monde courant une fois cette action exécutée ?

STRIPS : Représentation du monde

- On va utiliser la logique des prédicats pour deux avantages :
  - une représentation uniforme
  - une application des systèmes d'inférence logique.
- Dans le monde des cubes on a des objets qui
  - possèdent des attributs (objet, attribut, valeur)
  - Des relations R entre objets
- Modélisation:
  - (objet, attribut, valeur) -> *attribut(objet,valeur)*
  - Une relation entre deux objets: *Relation(Objet1,objet2)*

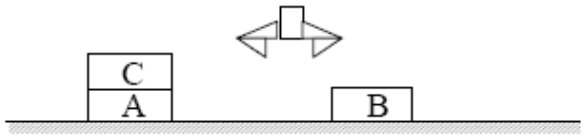
STRIPS : Représentation du monde
application au monde des cubes

- Formalisme logique de premier ordre

- Dans le monde des cubes, nous avons les prédicats suivants classés par arité :
  - ***HANDEMPTY*** : le bras du robot est libre,
  - ***HOLDING(x)*** : le bras du robot tient le cube x,
  - ***CLEAR(x)*** : le cube x ne supporte aucun cube,
  - ***ONTABLE(x)*** : le cube x est posé sur la table,
  - ***ON(x,y)*** : le cube x est posé sur le cube y.

- Un modèle d'état est la donnée d'un ensemble de formules (le plus souvent des littéraux sans variable) qui sont connues comme étant vraies par l'agent. On emploiera par abus de langage, le terme état pour désigner un modèle d'état.

- Exemple: ***ONTABLE(A) , ONTABLE(B) , ON(C,A) , HANDEMPTY , CLEAR(B) , CLEAR(C)***



STRIPS : Représentation des actions

- Une action est définie par un n-uplet
  - Un {} de variables qui designe les paramètres de l'action
  - Une contrainte de distinction entre les variables (utilisant le AND; le OR et le NOT)
  - PRE: tient pour la pré-condition c'est une conjonction de formules qui doivent être valides dans l'état courant pour assurer le succès de l'action.
  - DEL : les formules qui ne sont plus connus par l'agent pour être vrai (attention !!!!! cela ne signifie pas qu'elles soient fausses)
  - POST : les faits qui deviendront vrai une fois l'action exécutée.

STRIPS : Représentation des actions
application au monde des cubes

Le robot et les cubes

- pickup(x) : le robot attrape le cube x posé sur la table
  PRE & DEL : ONTABLE(x), CLEAR(x), HANDEMPTY
  ADD : HOLDING(x)

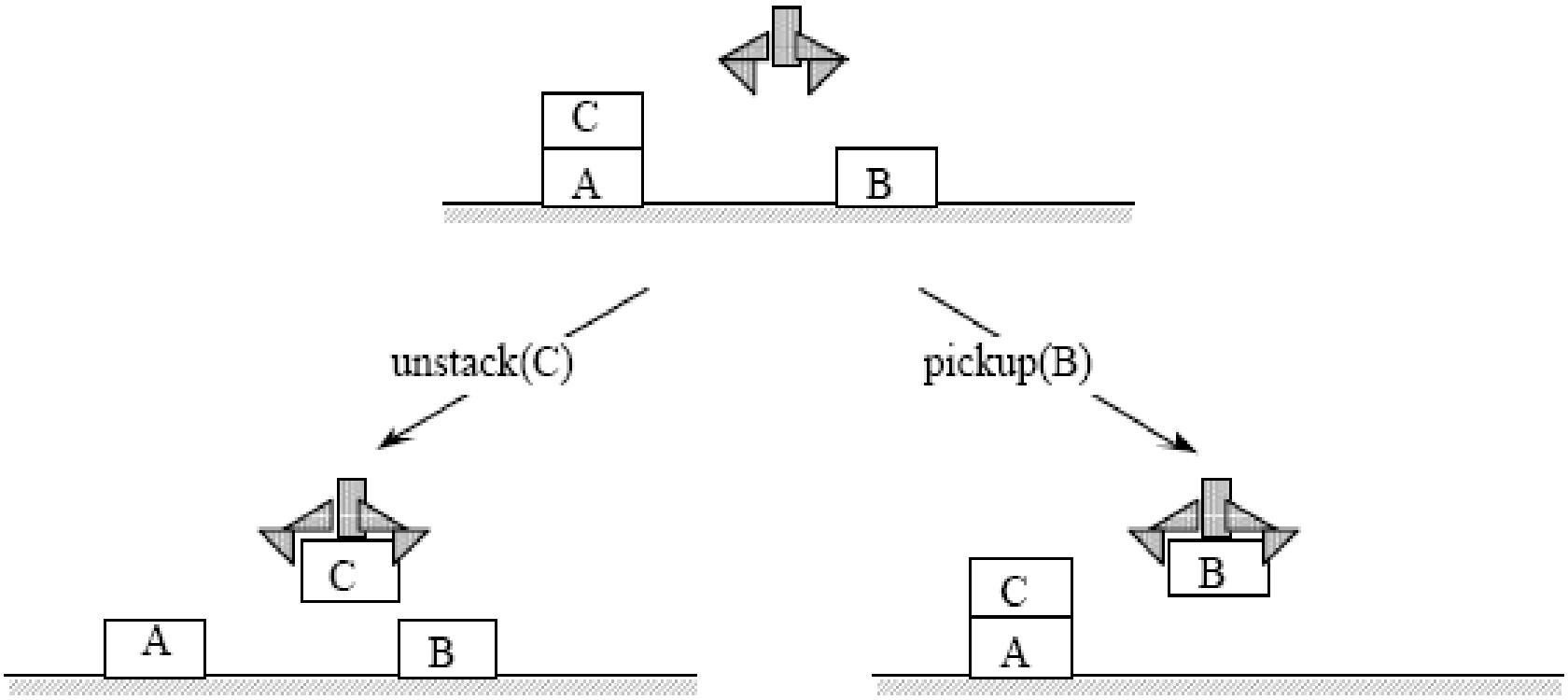
- putdown(x) : le robot dépose le cube x sur la table
  PRE & DEL : HOLDING(x)
  ADD : ONTABLE (x), CLEAR(x), HANDEMPTY

- stack(x,y) : le robot dépose le cube x sur le cube y
  PRE & DEL : HOLDING(x), CLEAR(y)
  ADD : HANDEMPTY, ON(x,y), CLEAR(x)

- unstack(x,y) : le robot attrape le cube x posé sur le cube y
  PRE & DEL : HANDEMPTY, CLEAR(x), ON (x,y)
  ADD : HOLDING(x), CLEAR(y)

Exemple

- Soit l'état du monde précédemment considéré comme état courant du monde.
  - ***ONTABLE(A) , ONTABLE(B) ON(C,A) , HANDEMPTY , CLEAR(B) , CLEAR(C)***



Sémantique de la présentation STRIPS

- Définition Un système de planification à la STRIPS est la donnée :
  - d'un ensemble de formules essentielles
  - d'un ensemble de règles "à la STRIPS" dont PRE, DEL et ADD sont des conjonctions de formules essentielles
  - d'un ensemble de formules d'interprétation sémantique
  - d'un modèle d'état initial composé de formules essentielles

Un système de planification STRIPS

Formules essentielles

HANDEMPTY
HOLDING,
CLEAR
ONTABLE
ON

Formules interprétation sémantique

- L'état du robot
  HANDEMPTY XOR ∃ x HOLDING(x)
  ∀ x,y HOLDING(x) AND HOLDING(y) ⇒ x=y
- Le sommet d'un cube
  ∀ x HOLDING(x) XOR CLEAR(x) XOR ∃ y ON(y,x)
  ∀ x,y,z ON(y,x) AND ON(z,x) ⇒ y=z
- La base d'un cube
  ∀ x HOLDING(x) XOR ONTABLE(x) XOR ∃ y ON(x,y)
  ∀ x,y,z ON(x,y) AND ON(x,z) ⇒ y=z
- La cohérence d'un empilement (à l'aide d'un prédicat auxiliaire)
  ∀ x,y ON(x,y) ⇒ ON*(x,y)
  ∀ x,y,z ON(x,y) AND ON*(y,z) ⇒ ON*(x,z)
  ∀ x NOT ON*(x,x)

Règles ou actions

Le robot et les cubes

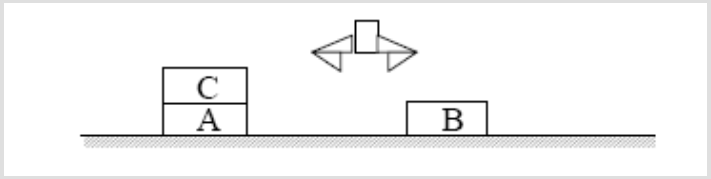
- pickup(x) : le robot attrape le cube x posé sur la table
  PRE & DEL : ONTABLE(x), CLEAR(x), HANDEMPTY
  ADD : HOLDING(x)

- putdown(x) : le robot dépose le cube x sur la table
  PRE & DEL : HOLDING(x)
  ADD : ONTABLE (x), CLEAR(x), HANDEMPTY

- stack(x,y) : le robot dépose le cube x sur le cube y
  PRE & DEL : HOLDING(x), CLEAR(y)
  ADD : HANDEMPTY, ON(x,y), CLEAR(x)

- unstack(x,y) : le robot attrape le cube x posé sur le cube y
  PRE & DEL : HANDEMPTY, CLEAR(x), ON (x,y)
  ADD : HOLDING(x), CLEAR(y)

Modèle de l'état initial

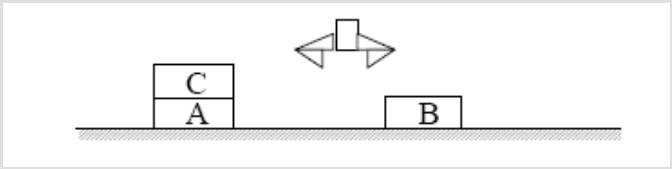


ONTABLE(A) , ONTABLE(B) ,ON(C,A) , HANDEMPTY , CLEAR(B) , CLEAR(C)

Consistance d'un modèle d'état : pour STRIPS (2)

-Soit FE une fonction qui assouci à chaque état du monde l'ensemble

de formules essentielles qui la compose

Formules essentiel: **F**	Formules interprétation sémantique: **IS**
HANDEMPTY ***HOLDING,*** ***CLEAR*** ***ONTABLE*** ***ON***	• L'état du robot HANDEMPTY XOR ∃ x HOLDING(x) ∀ x,y HOLDING(x) AND HOLDING(y) ⇒ x=y • Le sommet d'un cube ∀ x HOLDING(x) XOR CLEAR(x) XOR ∃ y ON(y,x) ∀ x,y,z ON(y,x) AND ON(z,x) ⇒ y=z • La base d'un cube ∀ x HOLDING(x) XOR ONTABLE(x) XOR ∃ y ON(x,y) ∀ x,y,z ON(x,y) AND ON(x,z) ⇒ y=z • La cohérence d'un empilement (à l'aide d'un prédicat auxiliaire) ∀ x,y ON(x,y) ⇒ ON*(x,y) ∀ x,y,z ON(x,y) AND ON*(y,z) ⇒ ON*(x,z) ∀ x NOT ON*(x,x)
Règles ou actions : R	**Modèle de l'état initiale : M**
Le robot et les cubes • pickup(x) : le robot attrape le cube x posé sur la table PRE & DEL : ONTABLE(x), CLEAR(x), HANDEMPTY ADD : HOLDING(x) • putdown(x) : le robot dépose le cube x sur la table PRE & DEL : HOLDING(x) ADD : ONTABLE (x), CLEAR(x), HANDEMPTY • stack(x,y) : le robot dépose le cube x sur le cube y PRE & DEL : HOLDING(x), CLEAR(y) ADD : HANDEMPTY, ON(x,y), CLEAR(x) • unstack(x,y) : le robot attrape le cube x posé sur le cube y PRE & DEL : HANDEMPTY, CLEAR(x), ON (x,y) ADD : HOLDING(x), CLEAR(y)	 ***ONTABLE(A) , ONTABLE(B) ,ON(C,A) , HANDEMPTY , CLEAR(B) , CLEAR(C),***

FE

Consistance d'un modèle d'état : pour STRIPS (2)

Définition :

Soit M un modèle soit L(M) ensemble de formules déductibles de M et IS

$$IS, M \models L(M) \text{ on a}$$

- $FE(L(M)) \subseteq FE(M)$ *: aucune nouvelle formule essentielle n'est déductible (e.g Holding)*
- $Faux \notin L(M)$ *(e.g ON(A,A) ne doit pas appartenir à L(M))*

Consistance d'un système de planification à la STRIPS

Définition :

Un système de planification à la STRIPS est sémantiquement consistant ssi

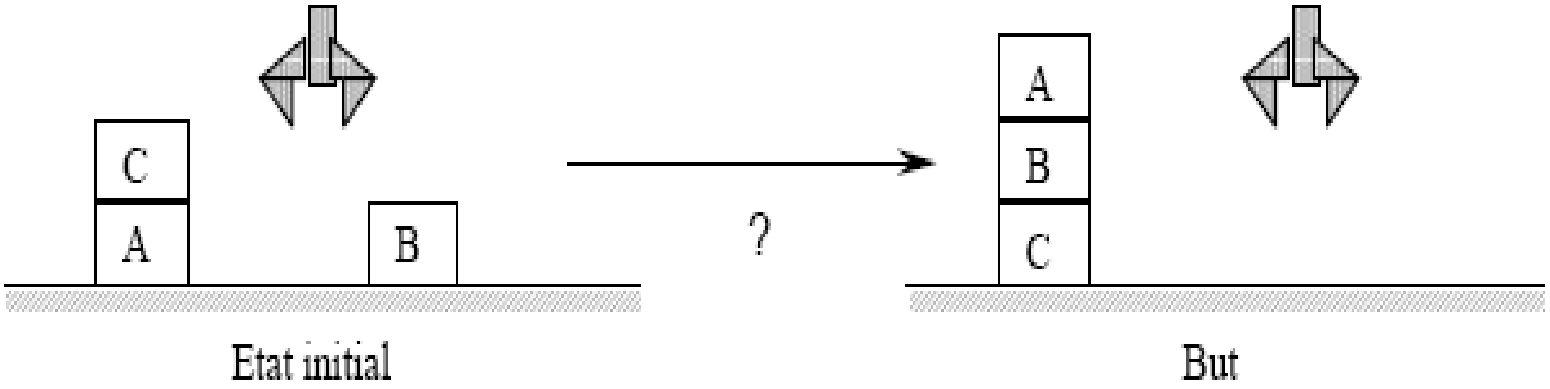
- ***Son modèle d'état initial est consistant***
- ***Et quelque soit les états consistant toute action sur un état consistant donne un état consistant***

Problème de planification

- Soit E la représentation du monde actuel
- Et soit G la représentation du monde qu'on veut atteindre
- Un problème de planification noté (E;G) consiste à trouvé la suite d'actions $a_1;...;a_n$
- Tels que :

- Un plan est une suite d'actions qui appliquées sur l'état initial E mène au but G.

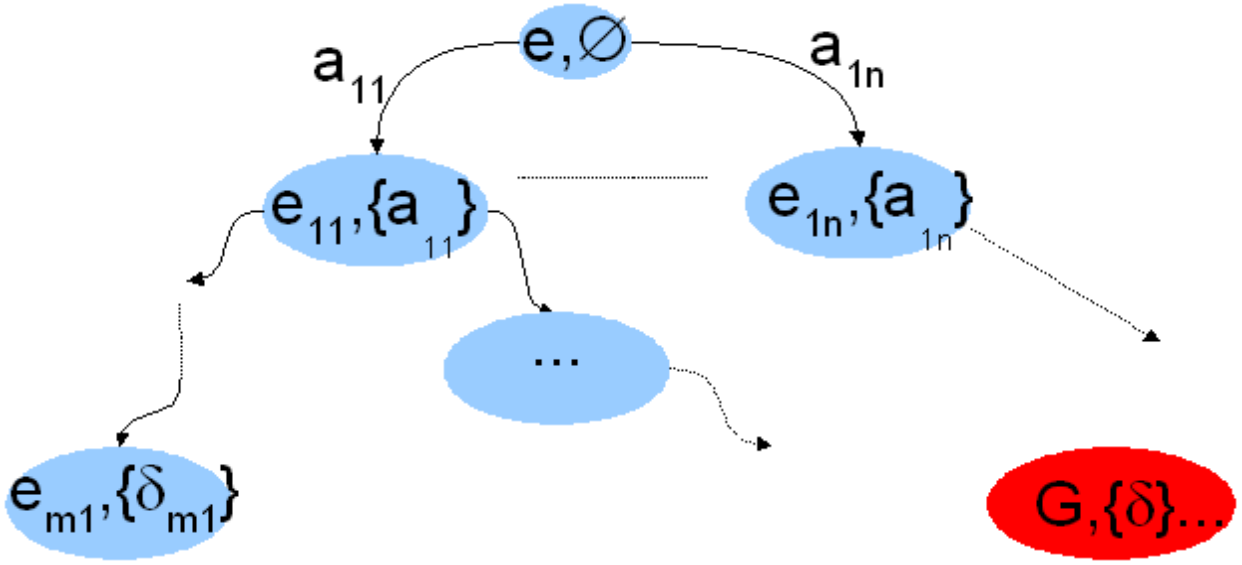
Problème de planification : exemple des cubes



pickup(C) ; putdown(C) ; pickup(B) ; stack(B,C) ; pickup(A) ; stack(A,B)

Planification et exploration

- Étant donné un état E
- Nous pouvons construire un graphe d'états où chaque état est un couple : Etat du monde et liste des actions qui mène de l'état du monde initial vers l'état en question.



Quelques propriétés d'un algorithme de planification

- ***Definition 1***: Un algorithme de planification est consistant ssi tout plan produit par le planificateur est correct.

- ***Definition 2***: Un planificateur est complet ssi lorsqu'un problème a au moins une solution alors le planificateur produit un plan.
  - ***Pour que cette propriété ait un sens il faut que le planificateur soit consistent***

- ***Définition 3***: Un planificateur se termine ssi :

  1. soit le problème a au moins une solution et le planificateur produit un plan,
  2. soit le problème n'a pas de solution et le planificateur s'arrête au bout d'un
  - ***Mais aussi très important la complexité***

Un algorithme générique

- La plupart des algorithmes se décrivent selon un schéma générique.
- Principes :
  - Un planificateur manipule à chaque instant une solution partielle. Une solution partielle contient souvent :
    - un plan partiel
    - des structures de contrôle additionnelles qui guident sa résolution.
  - il dispose d'un fonction *Succès* qui lui permette de savoir que la solution obtenue convient.

Un algorithme générique

Si *Succès*(solution initiale)
    Renvoyer(solution initiale)
Sinon
    S_{cons} = {solution initiale}
    S_{aex} = {solution initiale}
    Tant que $S_{aex} \neq \emptyset$ **Faire**
        Choisir(solution) dans S_{aex}
        Si (solution_suivante = *Nouveau_successeur*(solution)) $\neq \emptyset$ **Alors**
            Si $\nexists$ sol_vue $\in S_{cons}$ t.q. sol_vue *Subsume* solution_suivante **Alors**
                Si *Succès*(solution_suivante) **Alors**
                    Renvoyer(solution_suivante)
                Fsi
                $S_{cons} = S_{cons} \cup$ {solution_suivante}
                $S_{aex} = S_{aex} \cup$ {solution_suivante}
            Fsi
        Sinon
            $S_{aex} = S_{aex} \setminus$ {solution}
        Fsi
    FinTantque
    Renvoyer(échec)
Fsi

Cet algorithme se termine si l'espace des états est fini

Les spécificités d'un algorithme

- Chaque algorithme définit :
  - sa propre structure de solution,
  - le moyen de construire la solution initiale à partir du problème,
  - le test d'une solution (i.e. la fonction *Succès*),
  - le moyen de produire les successeurs d'une solution (i.e. la fonction *Nouveau_successeur*)
  - le fait qu'une solution subsume une autre c'est à dire que la première est assurée de conduire à une solution complète si la deuxième y conduit. La fonction *Subsume* la plus simple est le test d'égalité.
  - La variable `Scons` représente les solutions déjà construites
  - La variable `Saex` les solutions dont tous les successeurs n'ont pas été déterminés.

Les spécificités d'un algorithme

- la fonction *Choisir* détermine en fait la stratégie de l'exploration du graphe des solutions.

- Nous connaissons déjà pas mal de stratégie:
  - En Profondeur
  - en largeur...

- Cependant il est plus intéressant de choisir une stratégie qui puisse dépendre de la structure de la solution et éventuellement du problème à résoudre.
  - Une heuristique gagnante

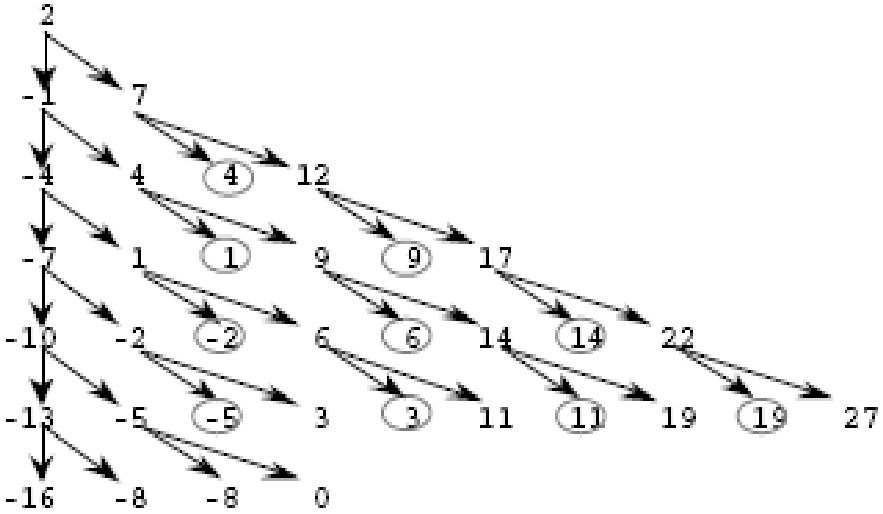
Example de problème d'exploration

- on démarre avec un entier (ici 2) et on dispose de règles pour modifier cet entier :
  - soit soustraire 3,
  - soit ajouter 5.
- On cherche à atteindre une valeur donnée (ici 0).
- Pour simplifier on appliquera toujours une soustraction avant un ajout dans la fonction *Nouveau_successeur*.

Parcours en profondeur

- Le parcours en profondeur ne se termine pas

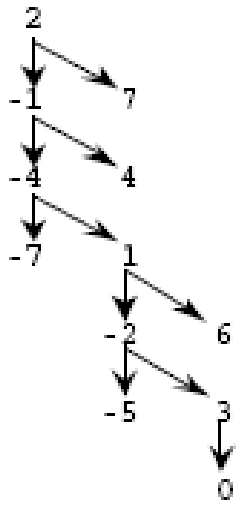
Le parcours en largeur se termine toujours (si une solution existe) mais la taille de l'arbre construit reste impressionnante par rapport au problème.



Une heuristique

- C'est une stratégie d'exploration qui tient compte de nature du problème
- Une idée peut être ???

Choisir de développer le successeur le plus proche du but (0)



Les planificateurs

Faisons le point....

- On a réduit les actions sur le monde à des règles de déduction. Chose qu'on savait déjas faire
- D'où un porblème de planification est une suite de règles appliquées de telle façon on arrive à déduire le monde but.
- On a des règles, on a un fait qui est monde initial, et on va déduire ou essayer de prouver un fait (le monde but).
- Comme un problème de systèmes expert.

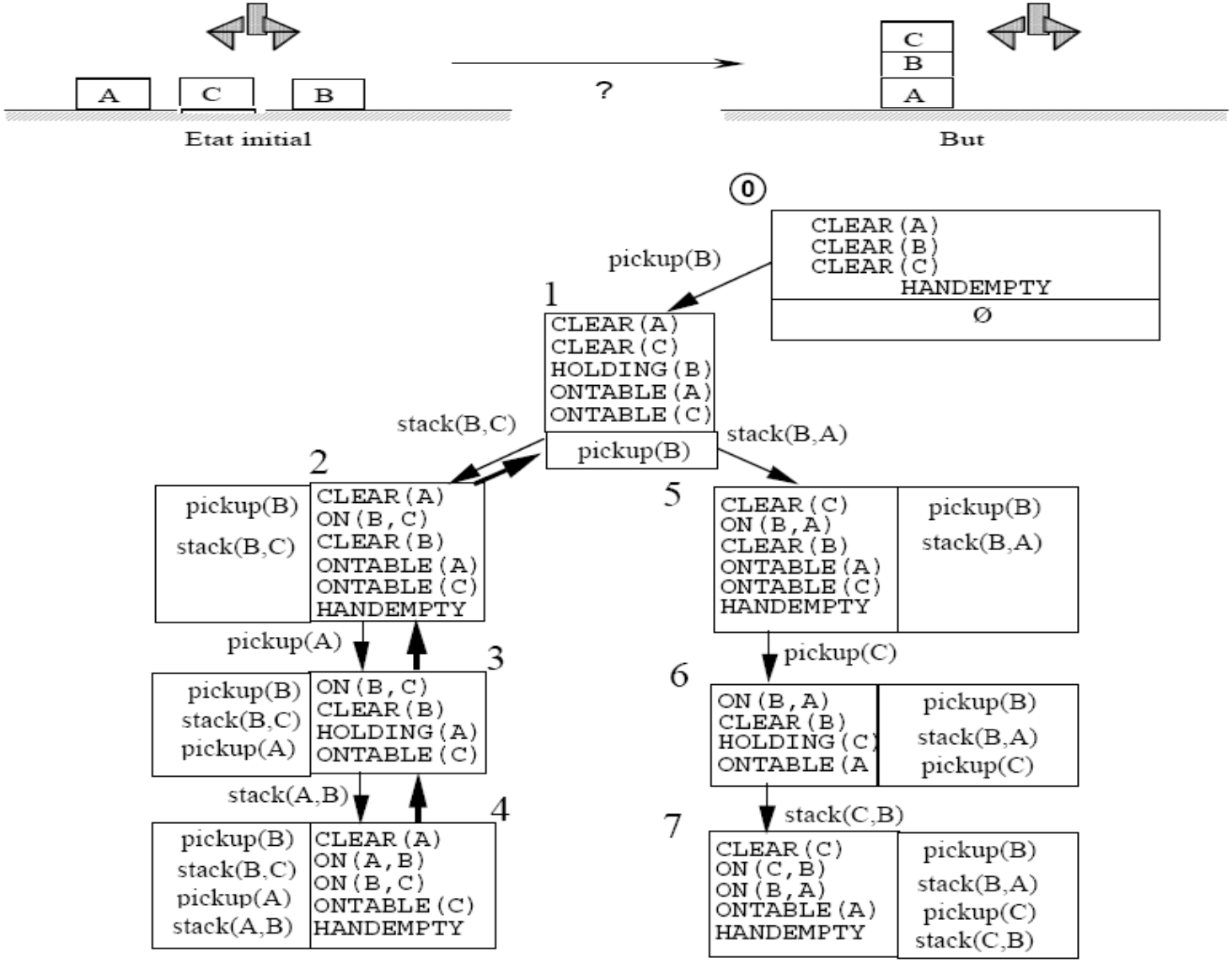
Et ça on sait faire : chaînage avant et chaînage arrière.

attention on est pas dans un cas monotone (retour arrière)

CAV : chaînage avant

- On commence par l'état initial
- La condition de succès consiste à vérifier que le but est satisfaisable dans le modèle d'état associé à la solution.
- Le test d'équivalence est l'égalité des modèles d'états : s=(e,π) subsume s'=(e',π') $\Leftrightarrow$ e = e' (ou mieux e $\supset$ e').
- Une stratégie possible est de choisir la solution dont l'état du monde satisfait le plus de sous-buts.

CAV: example


A
C
B
?
C
B
A
Etat initial
But
0
CLEAR(A)
CLEAR(B)
CLEAR(C)
HANDEMPTY
Ø
pickup(B)
1
CLEAR(A)
CLEAR(C)
HOLDING(B)
ONTABLE(A)
ONTABLE(C)
pickup(B)
stack(B,C)
stack(B,A)
2
pickup(B)
stack(B,C)
CLEAR(A)
ON(B,C)
CLEAR(B)
ONTABLE(A)
ONTABLE(C)
HANDEMPTY
5
CLEAR(C)
ON(B,A)
CLEAR(B)
ONTABLE(A)
ONTABLE(C)
HANDEMPTY
pickup(B)
stack(B,A)
pickup(A)
3
pickup(B)
stack(B,C)
pickup(A)
ON(B,C)
CLEAR(B)
HOLDING(A)
ONTABLE(C)
pickup(C)
6
ON(B,A)
CLEAR(B)
HOLDING(C)
ONTABLE(A
pickup(B)
stack(B,A)
pickup(C)
stack(A,B)
4
pickup(B)
stack(B,C)
pickup(A)
stack(A,B)
CLEAR(A)
ON(A,B)
ON(B,C)
ONTABLE(C)
HANDEMPTY
stack(C,B)
7
CLEAR(C)
ON(C,B)
ON(B,A)
ONTABLE(A)
HANDEMPTY
pickup(B)
stack(B,A)
pickup(C)
stack(C,B)

Propriété du CAV

- Théoriquement CAV a de nombreux avantages :
  - consistant ,
  - complet,
  - se termine (si les modèles du monde sont finis ce qui est souvent le cas).
  - Complexité élevé, au pire des cas le nombre des modèle du monde (la cause est les retours arrières)

CAR: Chaînage arrière

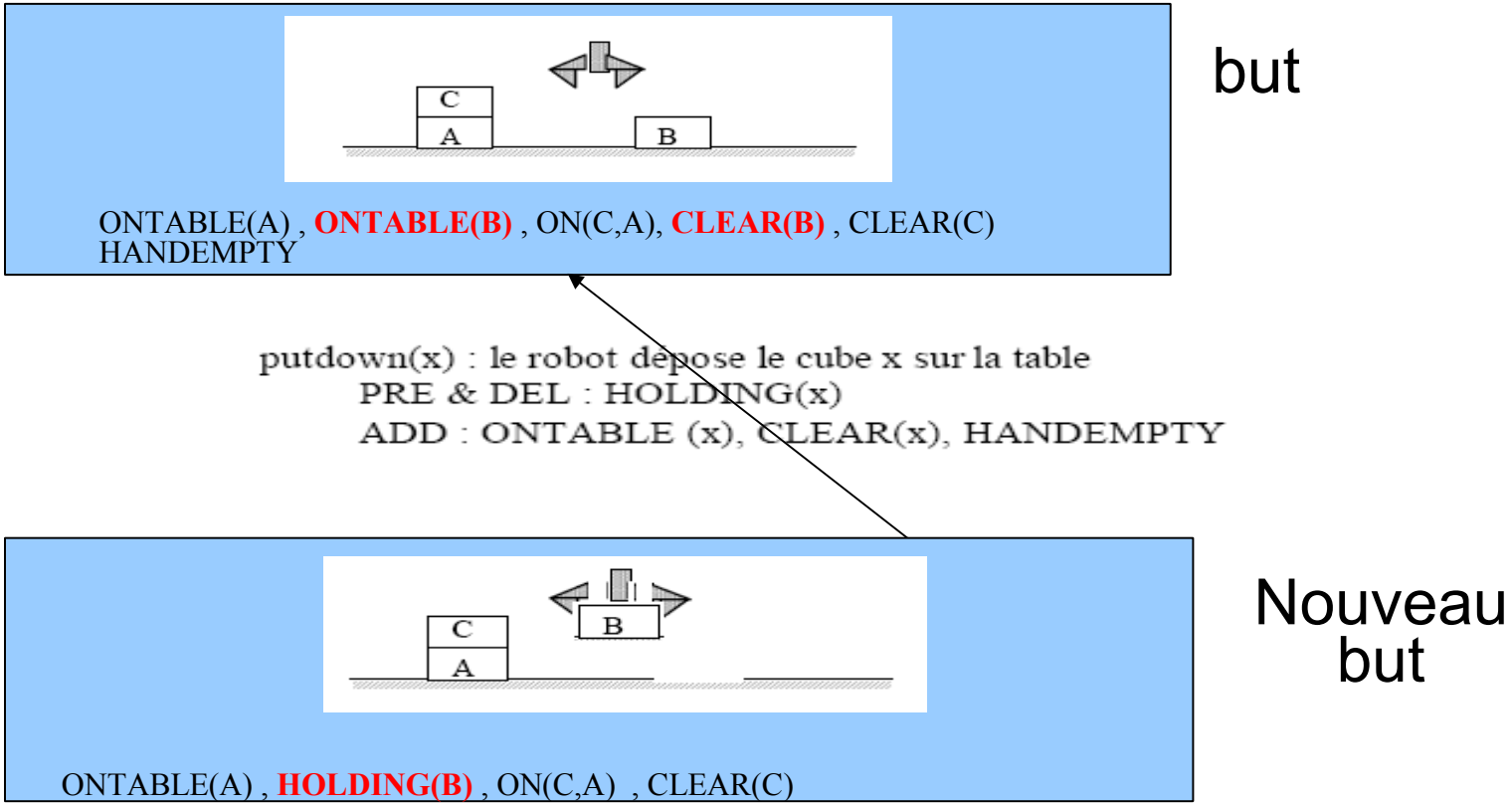
- L'algorithme CAR a un principe inverse de celui de CAV : il part du but qu'il "régresse" en buts intermédiaires par "régression" d'actions.
- La régression d'une action est une application qui permet de rendre vrai l'un des sous-buts du but considéré sans invalider les autres sousbuts.
- Lorsque l'un des buts intermédiaires (i.e. une solution courante du planificateur) est satisfaisable dans l'état initial, le plan est obtenu en appliquant les actions régressées en ordre inverse du chemin qui a conduit à cette solution.
- On escompte un grand gain en complexité car une solution est un ensemble de sous-buts régressés qui correspond implicitement à un ensemble d'états du monde : il y a factorisation des états à travers les buts qu'ils satisfont.

La régression vision intuitive

Ques> *Quelles actions aurait-elle put causer le fait que **B** est sur la table?*

Rep> ***putdown(B)*** *est la seul qui produit **B** sur la table et aussi le faite que **B** soit libre aussi.*

Ques> *quel été alors l'état du monde avant d'executer* ***putdown(B)***



Rappel sur l'unification restreintes

- Elle s'applique sur deux littéraux du même prédicat
  - $L = Q(\alpha_1, \ldots, a_n)$, $L' = Q(\alpha'_1, \ldots, \alpha'_n)$
  - La *Signature* de l'unification est un ensemble de variables et on désigne par *Cste* l'ensemble
  - des constantes de l'univers.
  - On requiert que $\alpha i , \alpha' i \in$ Cste $\cup$ Signature
  - Une unification de L et L' est une relation d'équivalence U sur {Signature $\cup$ Cste}qui vérifie :
  - $\forall$ i, αi U $\alpha' i$ (i.e. U doit identifier les arguments des littéraux)
  - $\forall$ C, classe d'équivalence de U, | C $\cap$Cste |1
  - Le résultat d'une unification consiste à remplacer les variables de la signature par de nouvelles variables qui représentent les classes d'équivalence. Plus précisément si C est une classe d'équivalence :
    - Si | C $\cap$ Cste | = 0 alors pour x $\in$ C, U(x) = X_c une nouvelle variable (ou une variable arbitraire déjà existante dans C)
    - Si | C $\cap$ Cste | = 1 alors pour x $\in$ C, U(x) = c $\in$ C l'unique constante de C.

Exemple d'unification

- Soient Q(x,a,y) et Q(b,z,t) à unifier, cela conduit à x U b, a U z et y U t.
- Soient Q(x,a,y) et Q(b,x,t) à unifier, cela conduit à x R b, a R z et y R t.
- Ici R désigne une relation d'équivalence qui ne conduit pas à une unification.
- On appellera instanciation, une unification qui fait disparaître toutes les variables

La négation

- Lorsqu'une unification est possible, on choisit la plus grande unification i.e. la relation d'équivalence la plus fine (celle qui produit le plus de classes d'équivalence).
- L'intérêt de l'unification est d'associer un littéral sous-but avec un littéral d'une liste ADD d'une règle lors de réfression de but
- Mais...
- Pour que le plan soit consistent il faut que cette unification ne doit pas permettre une possible unification des DEL de l'action avec l'un des autres sous but.

Contrainte de distinction

- A une unification, on associe une contrainte de distinction appelée Neg(U). Il s'agit d'un prédicat qui empêche l'unification.
- Neg(U) vrai $\Leftrightarrow$ U impossible.
- Pour empêcher qu'il y ait unification, il faut :
- Neg(U) = $OR_{\alpha U \beta} (\alpha \neq \beta)$ ou encore = $OR_{i=1\ldots n} (\alpha_i \neq \alpha_i')_i$
- dans le cas d'une unification de deux littéraux
- Exemple:
- Soient Q(x,a,z) et Q(y,x,t) à distinguer, cela conduit à $x \neq y \vee a \neq x \vee z \neq t$.

Solution CAR

- Une solution courante de CAR se compose de :
- un ensemble de variables, la signature
- une conjonction g =i = 1...n And i g de littéraux dont les seules variables appartiennent à Signature qui représentent les buts à satisfaire dans la solution courante
- une contrainte de distinction entre les variables de Signature qui interdisent certaines unifications
- une suite d'actions dont les variables non instanciées sont dans la signature (et représentent le plan partiel) Implicitement, toutes les variables des buts sont quantifiées existentiellement.

Le succès d'une solution

- Comme on l'a précisé le succès d'une solution est la possibilité de satisfaire les sousbuts.
- courants dans l'état initial.
- Succès(sol)
- ∃ U unification (en fait une instanciation) des variables telle que :
  - l'état initial du monde $e_0 \models g^U$ (équivalent à vérifier que tous les littéraux de g^U apparaissent dans e0)
  - la contrainte de distinction est vraie.

Successeur d'une solution : régression (1)

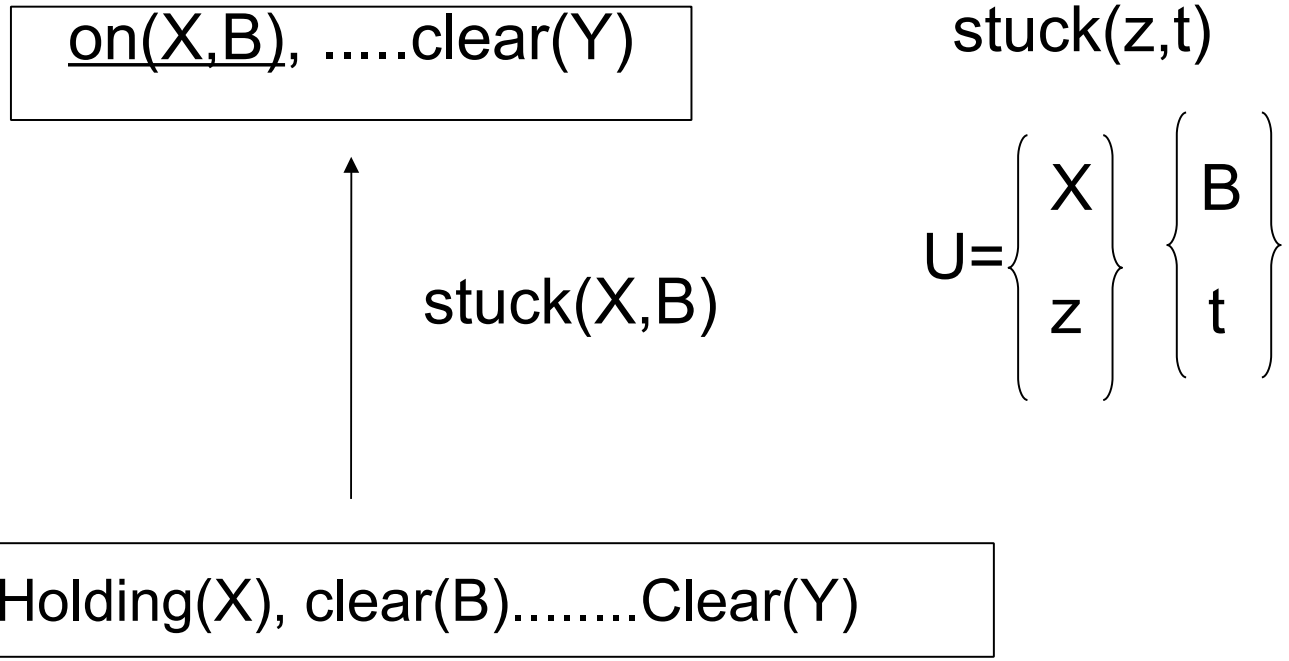
- Soit une solution courante *sol* définie par :
  - le but courant noté g = { g_i }
  - la contrainte de distinction
  - la signature de la solution notée *sig*
  - la suite d'actions à effectuer
- Soit une règle ou action *R* définie par
  - ADD={Add_i}, PRE{Pre_i}, DEL= {Del_i}
- Il y a régression de sol par *R* via l'unification *U* pour obtenir *sol'* ssi

Successeur d'une solution : régression(2)

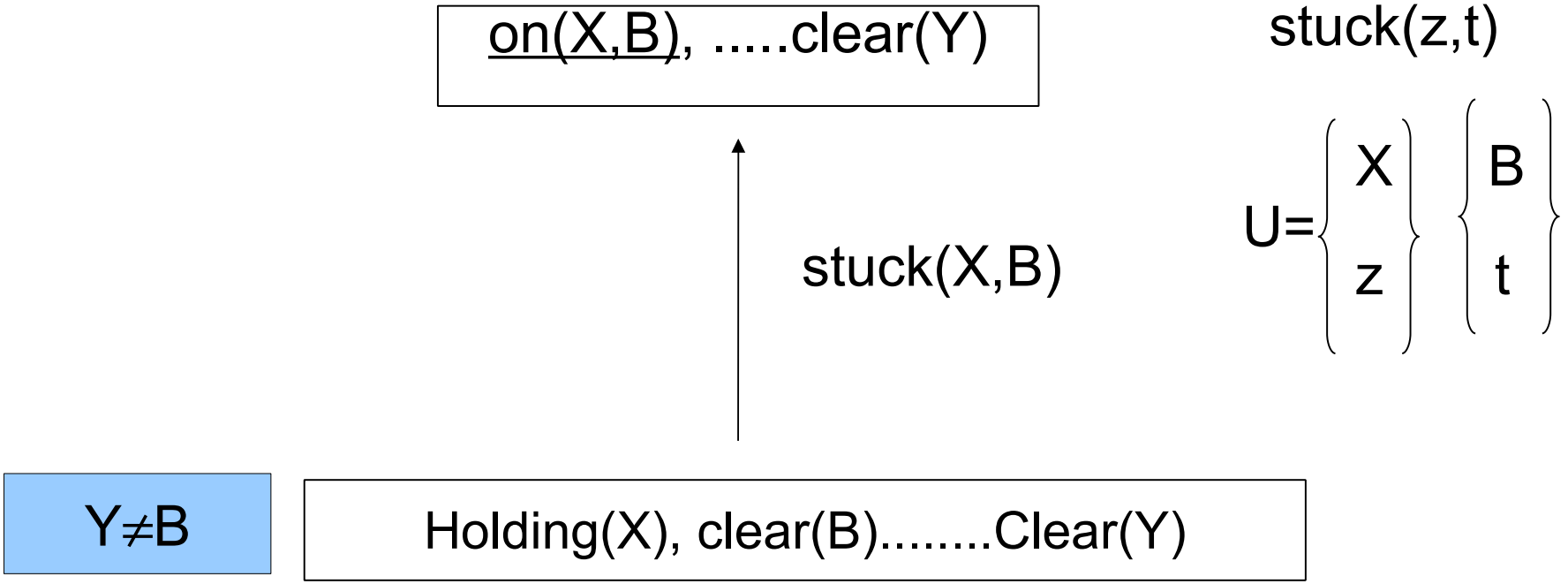
- Il y a régression de *sol* par l'action *A* via l'unification *U* pour obtenir *sol'* ssi
  - $\exists\ g_i \in g, \exists\ Add_j \in$ ADD tq g_i et Add_j s'unifient par U (plusieurs buts peuvent être unifiés simultanément avec des éléments de la liste Add
  - Sol' est :
    - $g' = \{ Pre_i^U \} \cup (\{ g_i^U \} / \{ Add_i^U \})$
    - $sig' = sig^U \cup signature(R)^U$
    - $contrainte' = contrainte^U \underset{U' \text{ destructrice}}{\text{And}} Neg(U')$

      où U' est destructrice ssi $\exists\ g_i \in g, \exists\ Del_j \in$ DELL tq g_i^U s'unifie avec Del_j^U par U'.
    - la suite d'actions est unifiée par U puis préfixée par A^U .

Example : unification



Example : négation



Explication:

Puisque stuck(X,B) détruit clear(B) il faut alors empêcher que stuck detruit tous les buts clear autre que B, Ici clear (Y). Ceci se fait en interdisant toutes unification de Y à B.

exercice

- Le monde est définie par des variables
- Chaque variable est définie par :
  - son nom
  - sa valeur (parmi un ensemble donné).
- L'agent est capable d'effectuer une seule action, l'affection
  - qui transfert la valeur d'une des variables vers l'autre en écrasant sont contenue.
- Question (10 mn)
- Modélisez ce problème en :
  - définissant les prédicats qui peuvent décrire notre monde des variables
  - La règle associer à l'action affectation (PRE, POST, DELL)

Solution:

- Pour decrire le monde il faut decrire une association entre le nom de la varaible et sa valeur :
  - c(X,Y) // la varaible de nom X contient la valeur Y

La seul action de l'agent est l'affection elle prend deux états de deux variable.

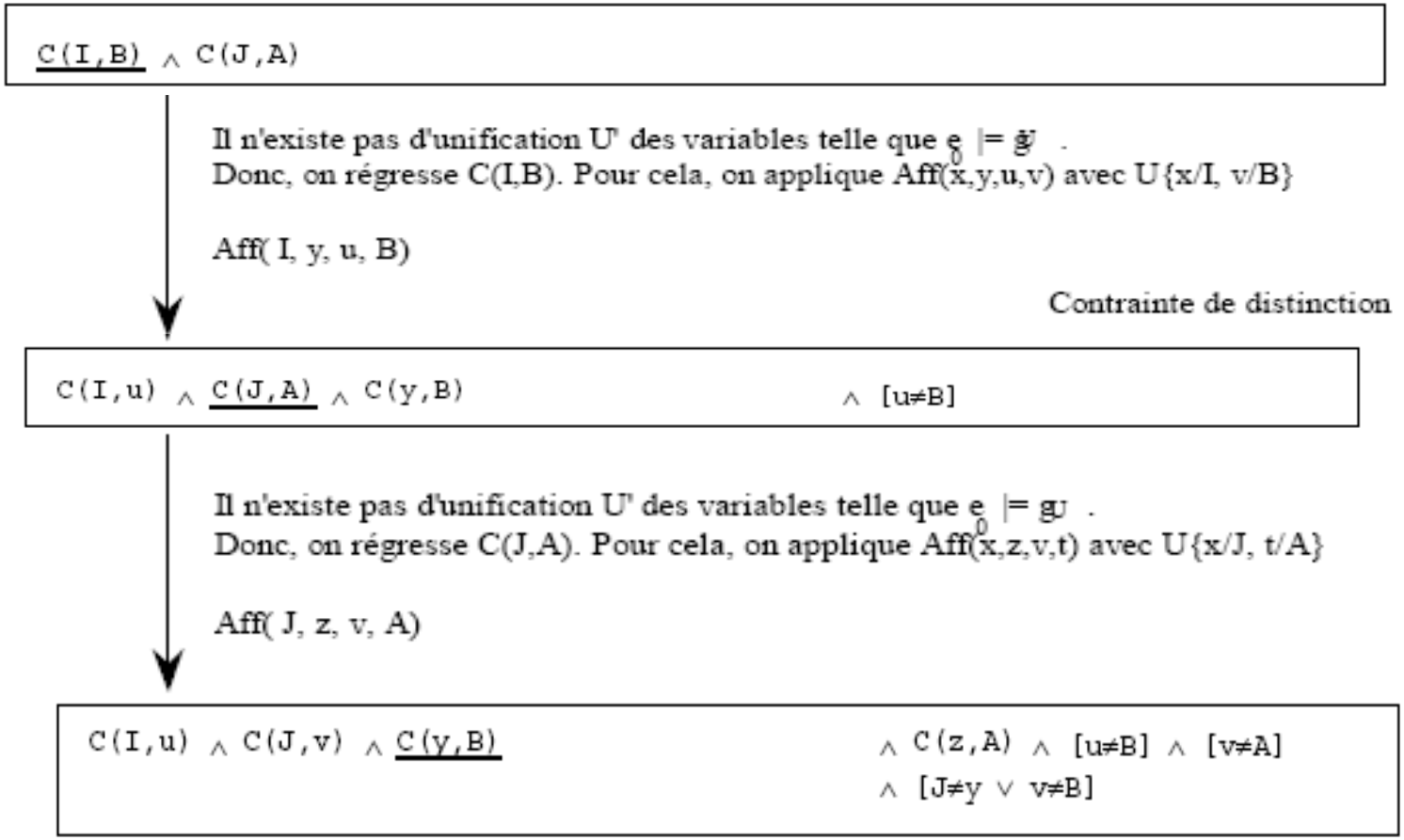
- Affect(X,Y,Z,T) // affecte la valeur de la variable Y qui contient T dans la variable X qui contient actuellement Z:
  - PRE : C(X,Z), C(Y,T)
    il faut que les valeurs correspondent vraiment
  - DELL: c(X,Z)
    Le contenu de X est écrasé
  - ADD: C(X,T)
    la nouvelle valeur de X est celle de Y

Excercice (Suite)

- Soit le monde suivant :
  - Trois variable I J K qui prenne leur valeur dans {A,B,C}.
  - Initialement I contient A, J contient B, et K contient C.
- On veux permuter le contenue de I et J.
- Question :
  1. Modélisez le monde initial et le but? En utilisant la modélisation de l'exercice précédent.
  2. Sans planification dite comment faire ?
- Solution:
  1. initial: *c(I,A), c(J,B), c(K,C)* BUT: *c(I,B), c(J,A)*
  2. la solution passe par l'utilisation d'une varaible intermidiare ici *K*

Solution avec CAR

- On démarre avec le but. L'heuristique choisie pour le choix du littéral à régresser est le littéral le plus ancien dont le second terme est une constante.



Suite et fin

La seule unification U' possible des variables telle que $e_0 \models g$ est {u/A, v/B, y/J, z/I}.
Mais, ceci fausse la contrainte de distinction. Donc, on régresse C(y,B). Pour cela, on applique Aff(x,t,w,r) avec U{x/y, r/B}

Aff(y, t, w, B)

C(I,u) ∧ C(J,v) ∧ C(y,w) ∧ C(z,A) ∧ C(t,B)
∧ [u≠B] ∧ [v≠A] ∧ [w≠B]
∧ [J≠y ∨ v≠B] ∧ [y≠I ∨ u≠w]
∧ [y≠J ∨ v≠w] ∧ [y≠z ∨ A≠w]

~~{u/A, v/B, z/I, t/J, y/I, w/A}~~

Violation [y=z=I et w=A]

~~{u/A, v/B, z/I, t/J, y/J, w/B}~~

Violation [y=J et v=B] et B=W

U={u/A, v/B, z/I, t/J, y/K, w/C}

$\text{Aff(y, t, w, B),Aff(J, z, v, A),Aff(I, y, u, B)}^{U} =$

Aff(K, J, C, B) ; Aff(J, I, B, A) ; Aff(I, K, A,B)