Intelligence Artificielle

Planification

Stephane Marchand-Maillet

Contenu

- Planification par représentation logique
- Formalisation
- Méthodes de planification

Planification

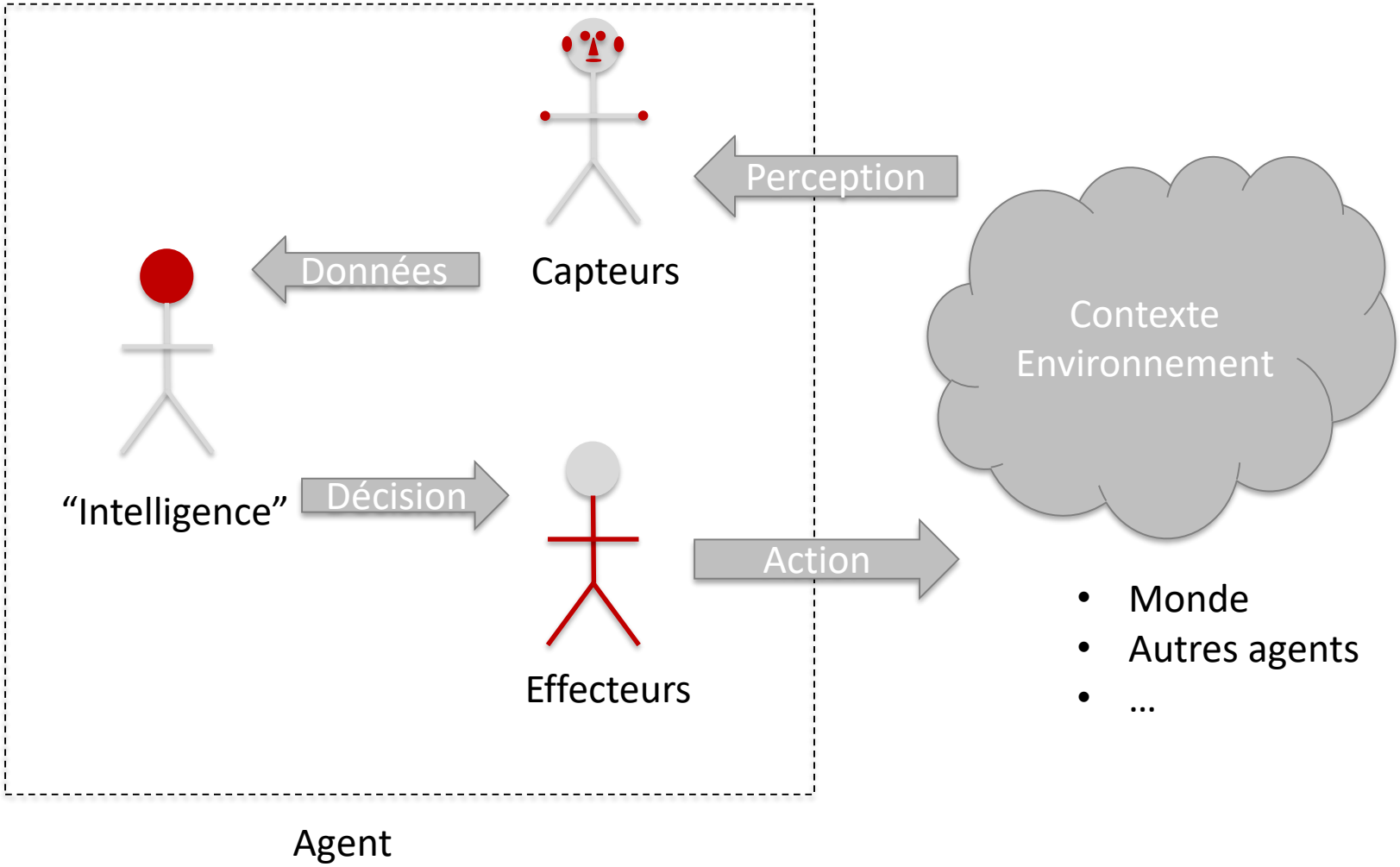
Position du problème:

La planification est le problème de produire une séquence d'actions (un plan) menant à un objectif fixé

Les méthodes recherche sont un cas particulier de planification où les outils utilisés sont des structures de données (graphe, arbres) et des algorithmes de recherche. Le plan est le chemin de s_I à s_G

Ici on va utiliser les outils de la logique pour la planification

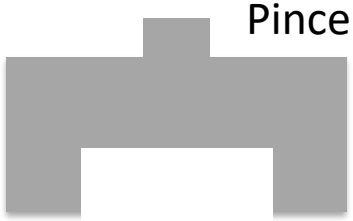
Rappel: Boucle Action-Perception



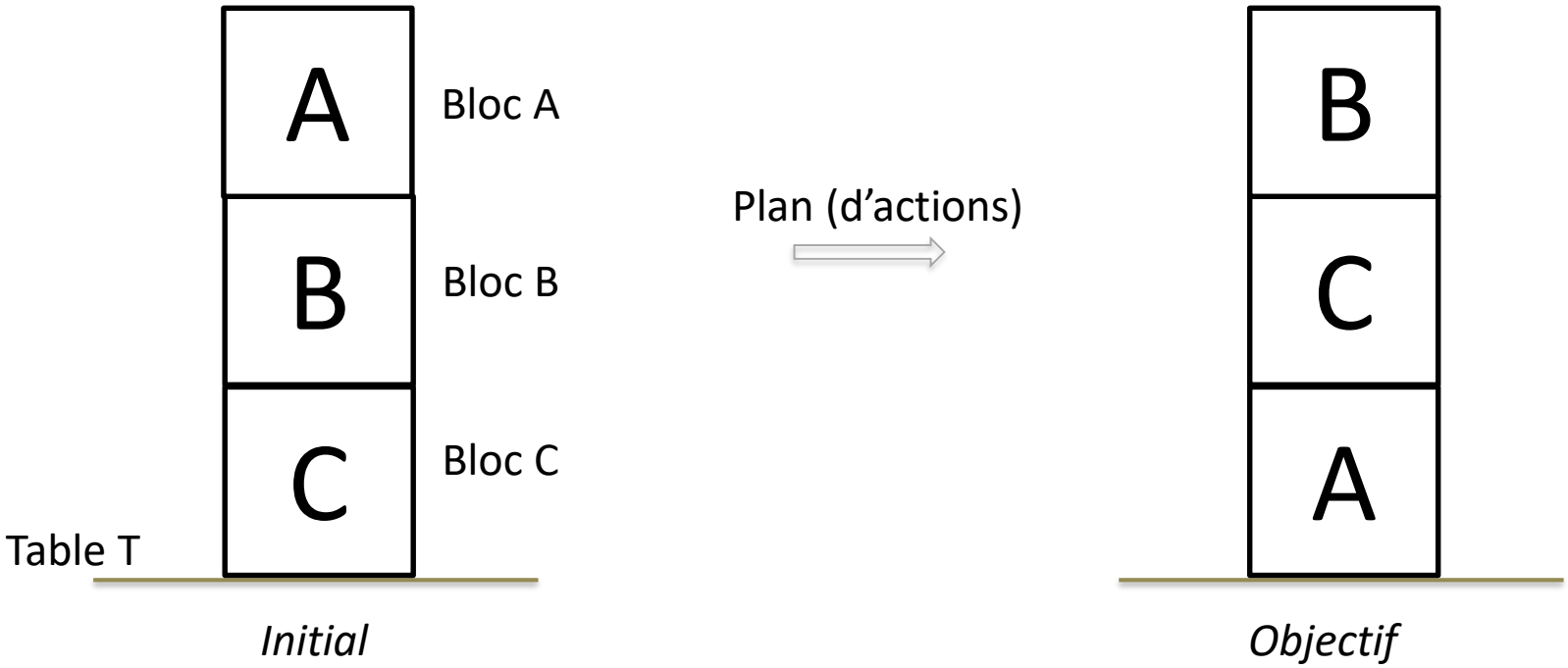
L'agent perçoit l'état actuel, et agit vers l'objectif

Exemple

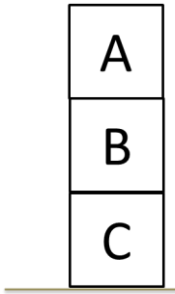
Blocs:



Pince P (effecteur)



Formalisation: faits, états



Observer les faits et les réifier:

- Un fait est une proposition (vraie):

  sur(A,B): le cube A est sur le cube B (vrai)

- On généralise (réifie) en fonction de l'état *s*:

  sur(*x*,*y*,*s*): à l'état *s*, l'objet (cube, table) *x* est sur l'objet *y*

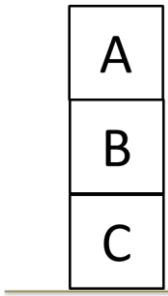
- On peut écrire:

  sur(A,B,s_I) ou sur(B,C, s_G)

  ou

  $\forall\, x,y,s$ sur(x,y,s) $\Rightarrow$ $\neg$ libre(y,s)

Formalisation: action



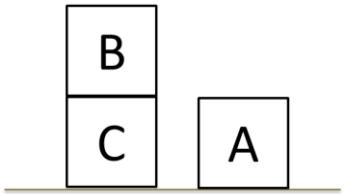
On fait de meme pour les actions:

- L'action deplacer(*x*,*y*,*z*) est l'opération de déplacer l'objet *x* (qui est sur *y*) vers *z*
- On applique cette operation sur l'état *s* pour générer un nouvel état *s'*

  s' = faire(action, s)

- On peut écrire

  libre(B,faire(deplacer(A,B,T),*s*))



ou

sur(*x*,*y*,*s*) $\wedge$ libre(*x*,*s*) $\wedge$ (z $\neq$ y) $\Rightarrow$ libre(*y*,faire(deplacer(*x*,*y*,*z*),*s*)

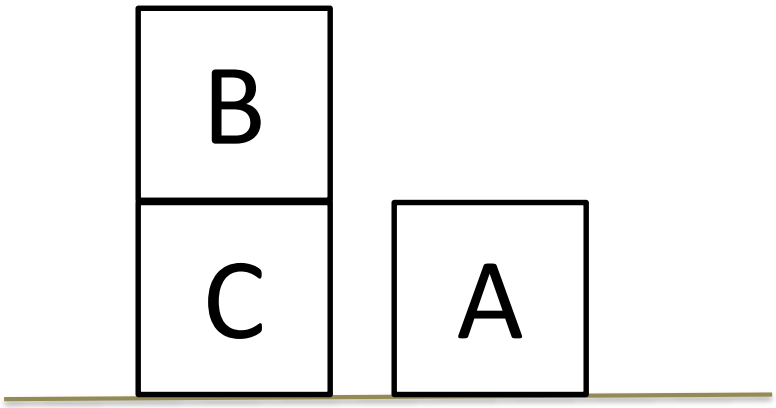
Actions

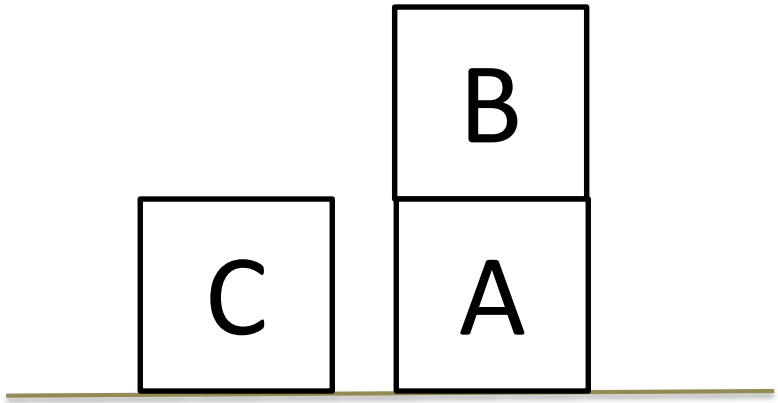
$sur(x,y,s) \wedge libre(x,s) \wedge libre(z,s) \wedge (z \neq y) \Rightarrow libre(y,faire(deplacer(x,y,z),s)$

Pré-conditions à l'action

Effet de l'action

$sur(x,y,s) \wedge libre(x,s) \wedge libre(z,s) \wedge (z \neq y) \Rightarrow sur(x,z,faire(deplacer(x,y,z),s)$

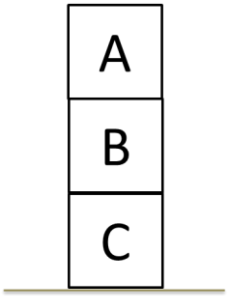




Formalisation: axiomes

On inclut des axiomes vrais pour tous les états comme règles d'inférence

$$\forall\, x,y,s \quad \text{sur}(x,y,s) \Rightarrow \neg\, \text{libre}(y,s)$$



Question: comment arriver à représenter le monde dans son ensemble?

$$\forall\, x,y,s \quad \text{sur}(x,y,s) \wedge (x \neq u) \Rightarrow \text{sur}(x,y,\text{faire}(\text{deplacer}(u,v,w),s)$$

(déplacer *u* n'affecte pas les positions relatives de *x* et *y*)

«*Frame problem*»: envelopper le monde

- Effets locaux
- Axiomes globaux

Enveloppe du monde

https://www.wired.com/2011/06/yes-a-laundry-folding-robot/

Créer un environnement local maitrisable

https://abcnews.go.com/GMA/Living/machine-fold-laundry-debuts-ces/story?id=60260894

Exemple pratique: langage STRIPS

STRIPS: STanford Research Institute Planning System (70's)

→ Créé pour la robotique (robot Harvey)

Principe: Basé sur la réduction de problèmes:

Décomposer un problème en sous-problèmes solubles

- A l'état *s*, chercher une différence avec s_G (proposition pas directement prouvable)
- Chercher une action *a* permettant de réduire cette différence et dont les préconditions sont satisfaites en *s*
- Appliquer l'action *a* vers *s'* et chercher à atteindre s_G de *s'*

STRIPS

- Pas de proposition négative

- Hypothèse du monde fermé:

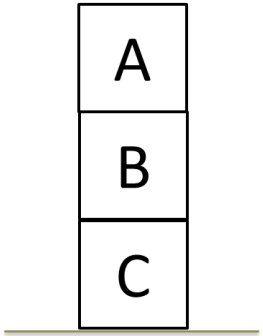
  Ce qui n'est pas exprimé comme vrai est faux

- Pas de disjonction (OR)
- Pas de variable

Exemple: Blocs

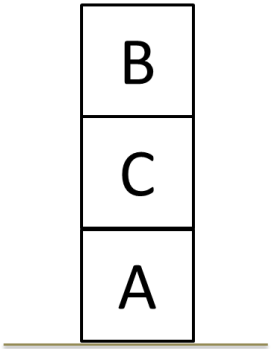
- Etat: conjonction de propositions

  cube(A) ∧ cube(B) ∧ cube(C) ∧
  sur(A,B) ∧ sur(B,C) ∧ sur(C,T) ∧
  libre(A) ∧ vide(P)



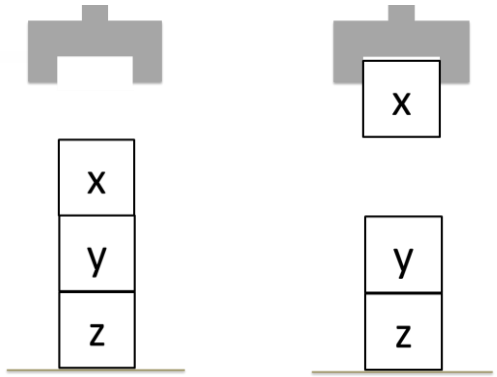
- Objectif (état): conjonction de propositions

  cube(A) ∧ cube(B) ∧ cube(C) ∧
  sur(C,A) ∧ sur(B,C) ∧ sur(A,T) ∧
  libre(B)



L'objectif est réalisé quand toutes les propositions le décrivant font partie de l'état courant

Représenter une action



action(sujets)

Préconditions: doivent être vraies

Effets:

Add-list: propositions ajoutées

Delete-list: propositions supprimées

Résultat: conjonction de propositions (nouvel état)

Exemple:

depiler(*x*,*y*)

P: vide(*P*), cube(*x*), cube(*y*), libre(*x*), sur(*x*,*y*)

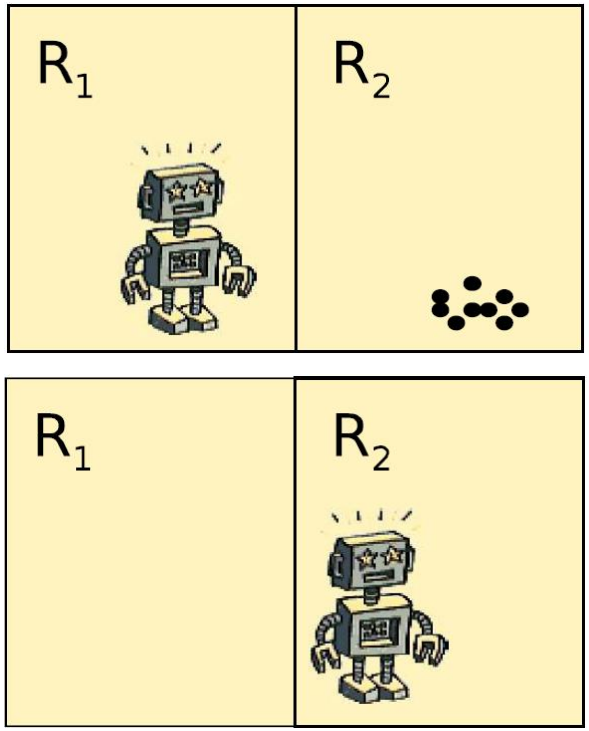
A: tenir(*x*), libre(*y*)

D: libre(*x*), vide(*P*), sur(*x*,*y*)

Exemple à développer

- Un robot aspirateur A
- 2 pièces R1, R2
- De la poussière

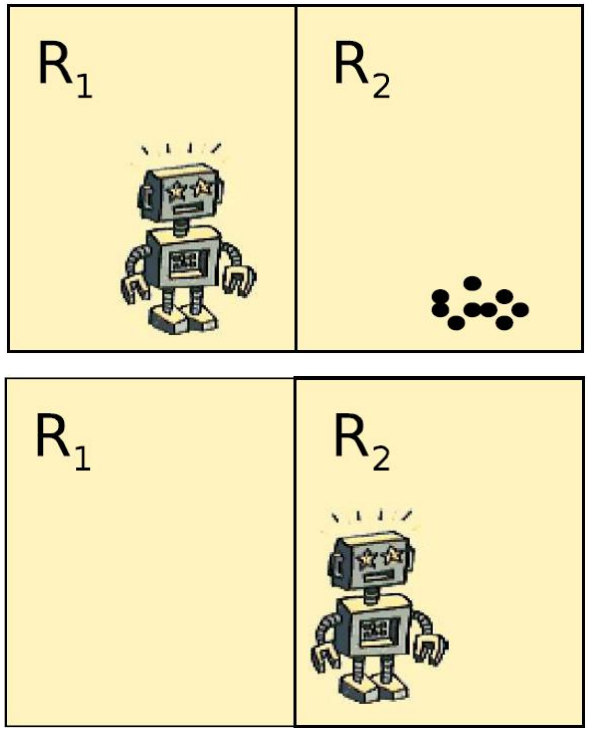
→ But: 2 pièces propres



- Agent? Etat?
- Actions?
- Etat initial? Etat final?

Exemple à développer

- Un robot aspirateur A
- 2 pièces R1, R2
- De la poussière

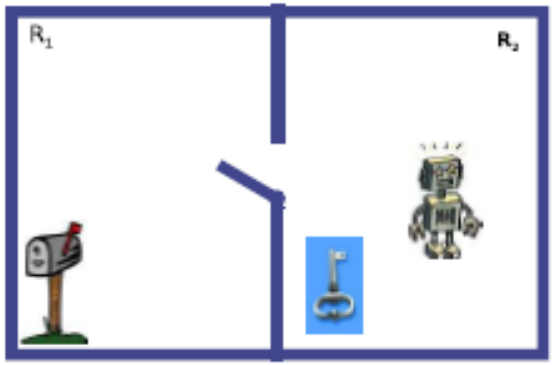


→ But: 2 pièces propres

- Agent? Etat? dans(A,R1), propre(R1)
- Actions? deplace(A,R1), aspire(R1)
- Etat initial? Etat final?

Exemple contraint

- Le robot doit fermer la porte à clé et mettre celle-ci dans la boîte
- Il faut la clé pour ouvrir et fermer la porte
- Une fois la porte fermée, le robot ne peut plus la rouvrir
- Une fois la clé dans la boîte, le robot ne peut plus la reprendre



Grasp-Key-in-R_2
P = In(Robot,R_2) ^ In(Key,R_2)
D = ∅
A = Holding(Key)

Lock-Door
P = Holding(Key)
D = Unlocked(Door)
A = Locked(Door)

Move-Key-from-R_2-into-R_1
P = In(Robot,R_2) ^ Holding(Key) ^ Unlocked(Door)
D = In(Robot,R_2), In(Key,R_2)
A = In(Robot,R_1), In(Key,R_1)

Put-Key-Into-Box
P = In(Robot,R_1) ^ Holding(Key)
D = Holding(Key), In(Key,R_1)
A = In(Key,Box)

Chaînage avant

- On développe un graphe pour explorer les états en appliquant les actions possible
- L'exploration s'arrête quand toutes les propositions du but sont présentes dans l'état courant

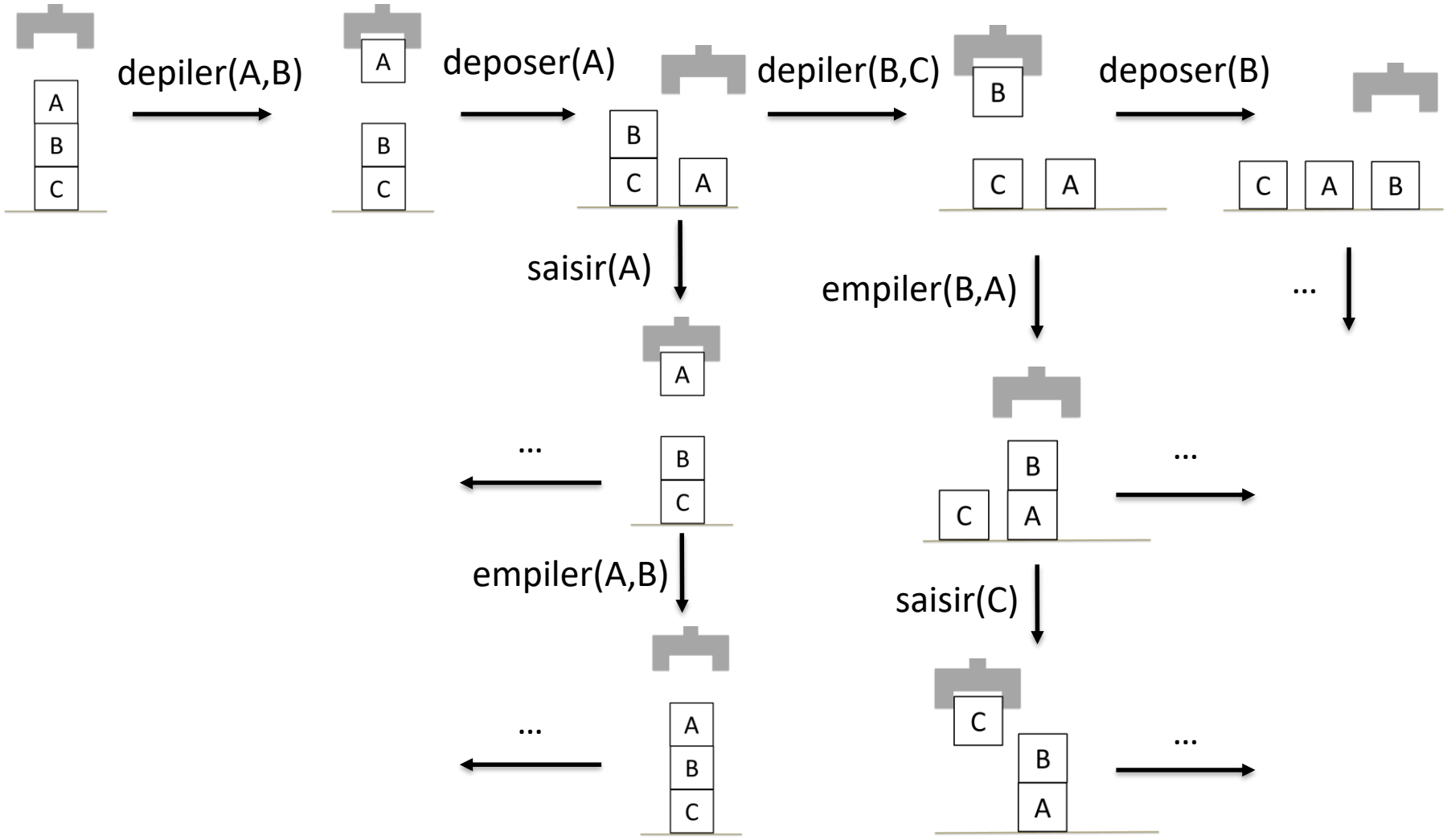
→Niveau de branchement élevé

→Pas de garantie de complétude (cycle)

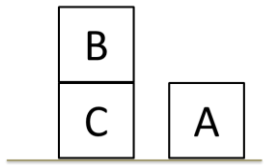
→Pas de garantie d'optimalité

→On cherche une heuristique pour guider la construction de la solution

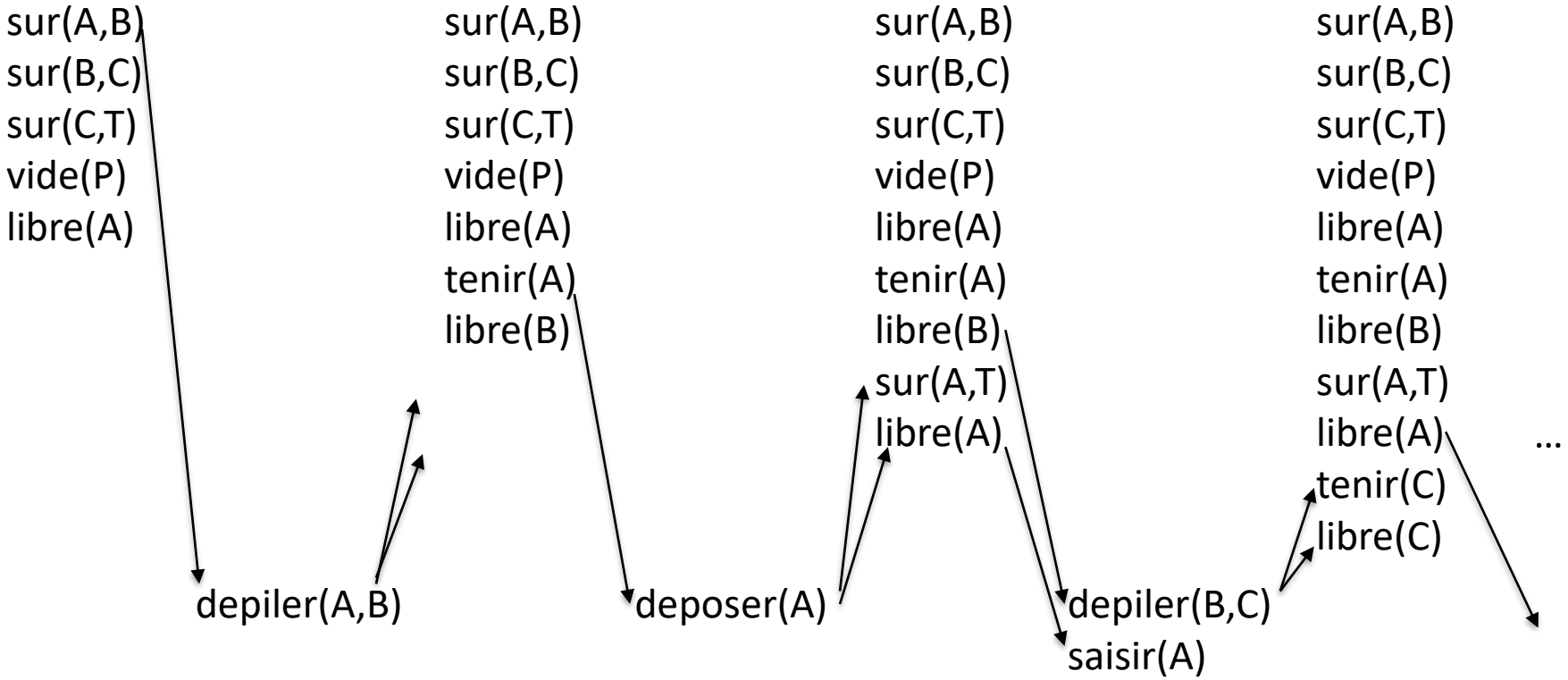
Chaînage avant



Graphe de planification



- On construit un graphe connectant les états par des actions applicables



- La profondeur des propositions dans ce graphe sert d'heuristique (approximation) admissible et consistante
- A* avec cette heuristique produit une solution avec un minimum d'actions (optimale)

Chaînage arrière

On part de l'objectif et on régresse par actions pertinentes pour retourner à l'état initial

→ Niveau de branchement réduit

Def: une action est pertinente pour un état si une proposition de sa ADD-LIST est contenue dans l'état

Exemple:

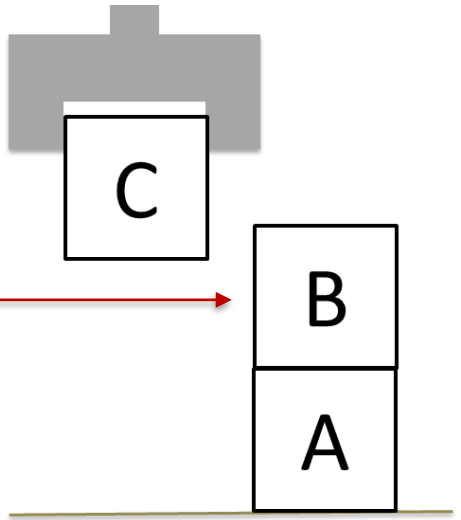
empiler(*x*,*y*)

P: tenir(*x*), cube(*x*), cube(*y*), libre(*y*)

A: sur(*x*,*y*), libre(*x*), vide(*P*)

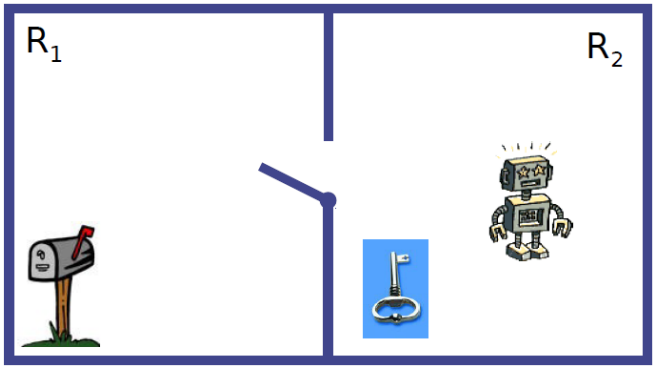
D: libre(*y*), tenir(*x*)

est pertinente (empiler(*C*,*B*)) pour l'état:



Planification non-linéaire

Certains sous-objectifs peuvent être incompatibles si ils ne sont pas atteints dans le bon ordre.



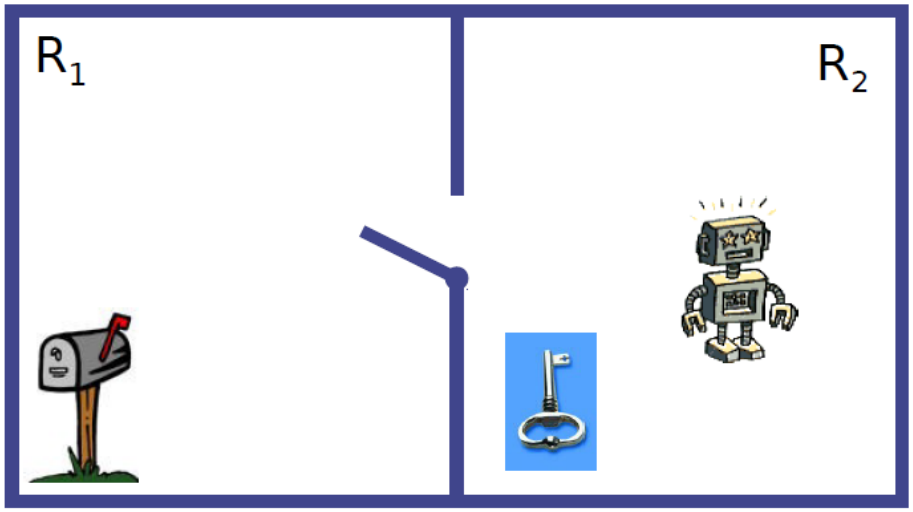
Exemple:

- Le robot doit (1) fermer la porte à clé et (2) mettre celle-ci dans la boîte

- Si le robot met la clé dans la boîte (2) avant de fermer la porte (1), l'objectif ne peut pas être atteint

→La planification non-linéaire opère par ordonnancement partiel des taches en gérant les conflits (cycles et menaces)

Planification non-linéaire



P = ⊘
D = ⊘
A = In(Robot,R_2)
In(Key,R2)
Unlocked(Door)

P = Locked(Door)
In(Key,Box)
D = ⊘
A = ⊘

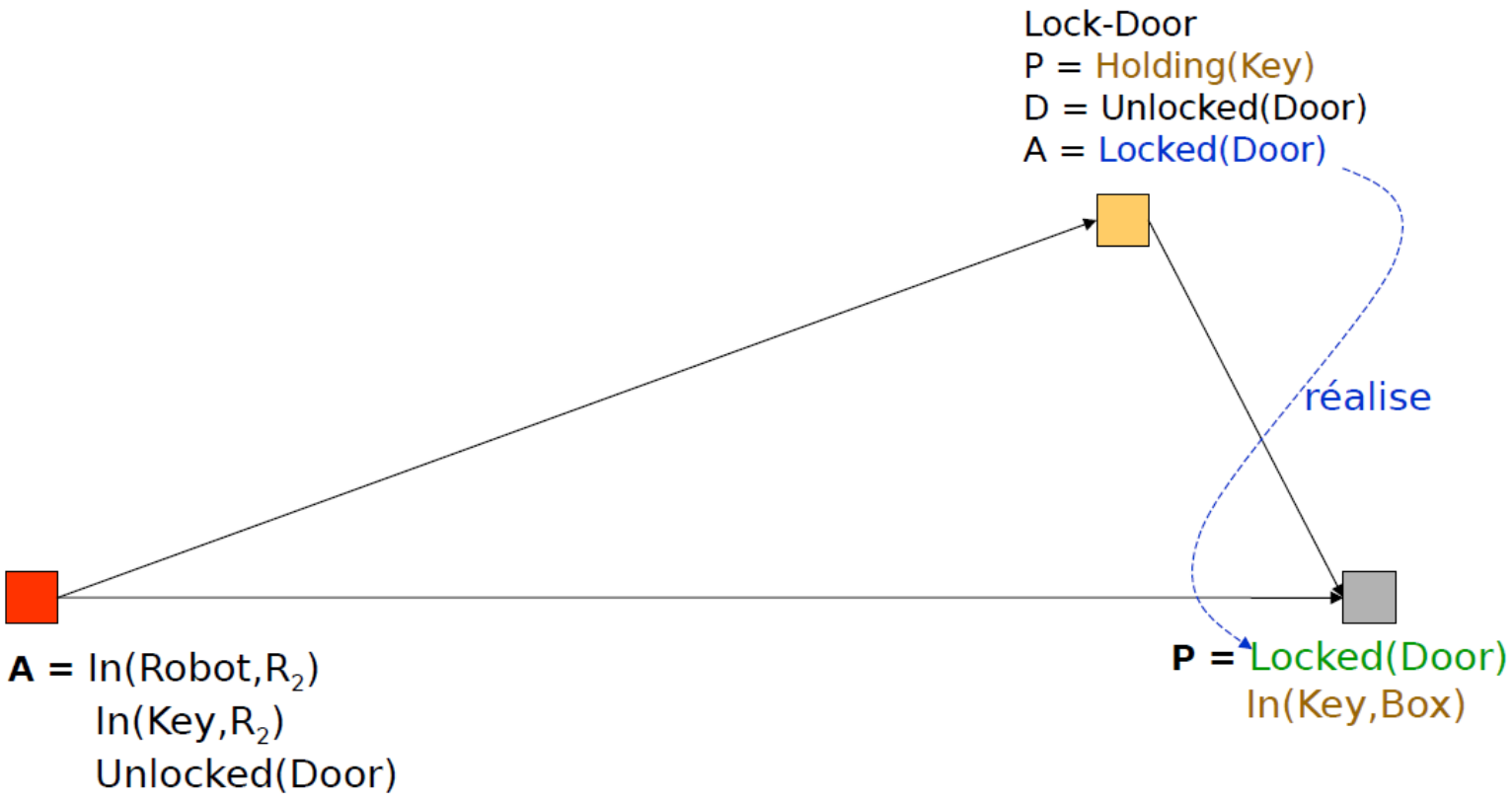
Planification non-linéaire

Lock-Door
P = Holding(Key)
D = Unlocked(Door)
A = Locked(Door)

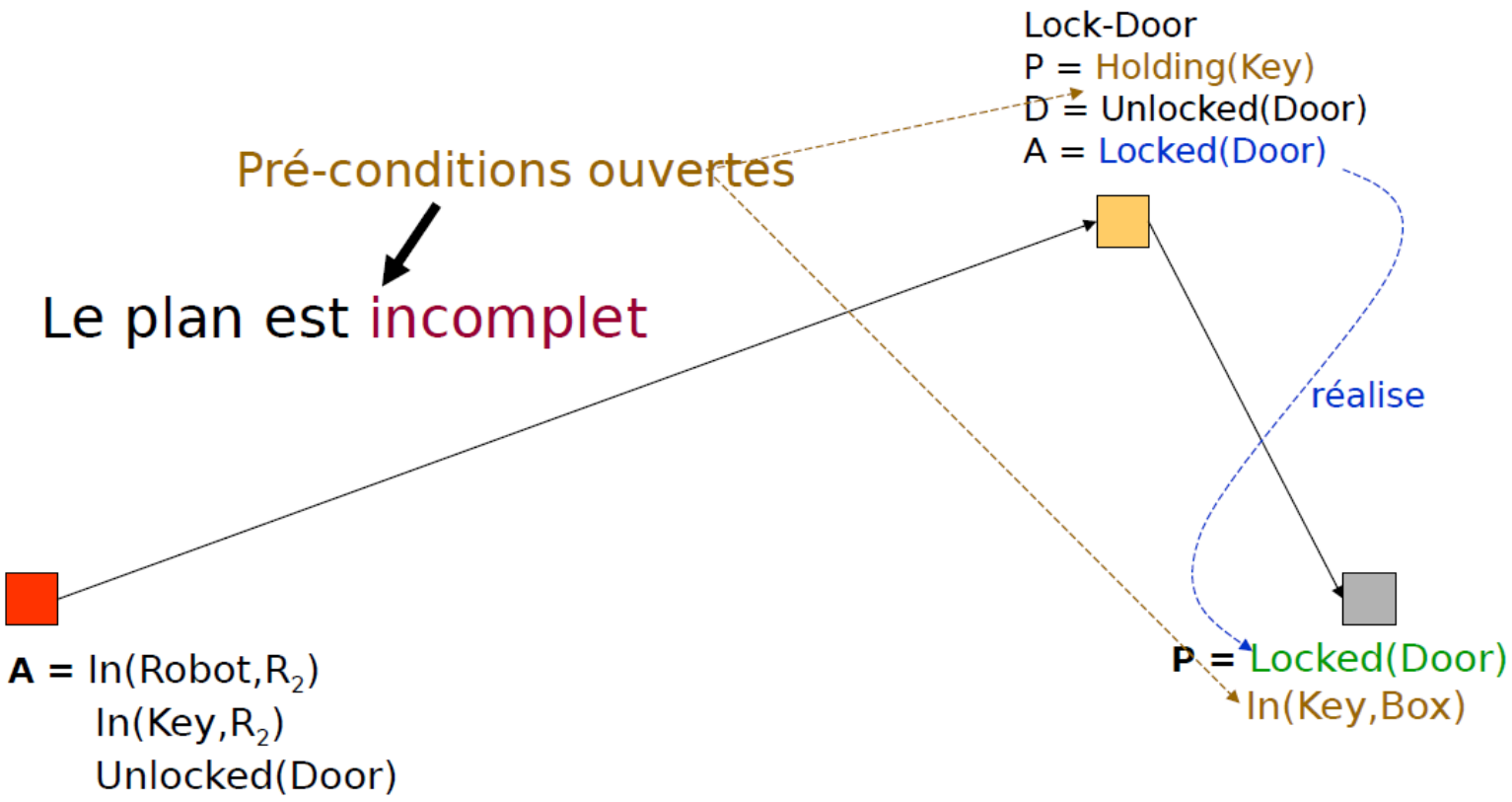
A = In(Robot,R_2)
In(Key,R_2)
Unlocked(Door)

P = Locked(Door)
In(Key,Box)

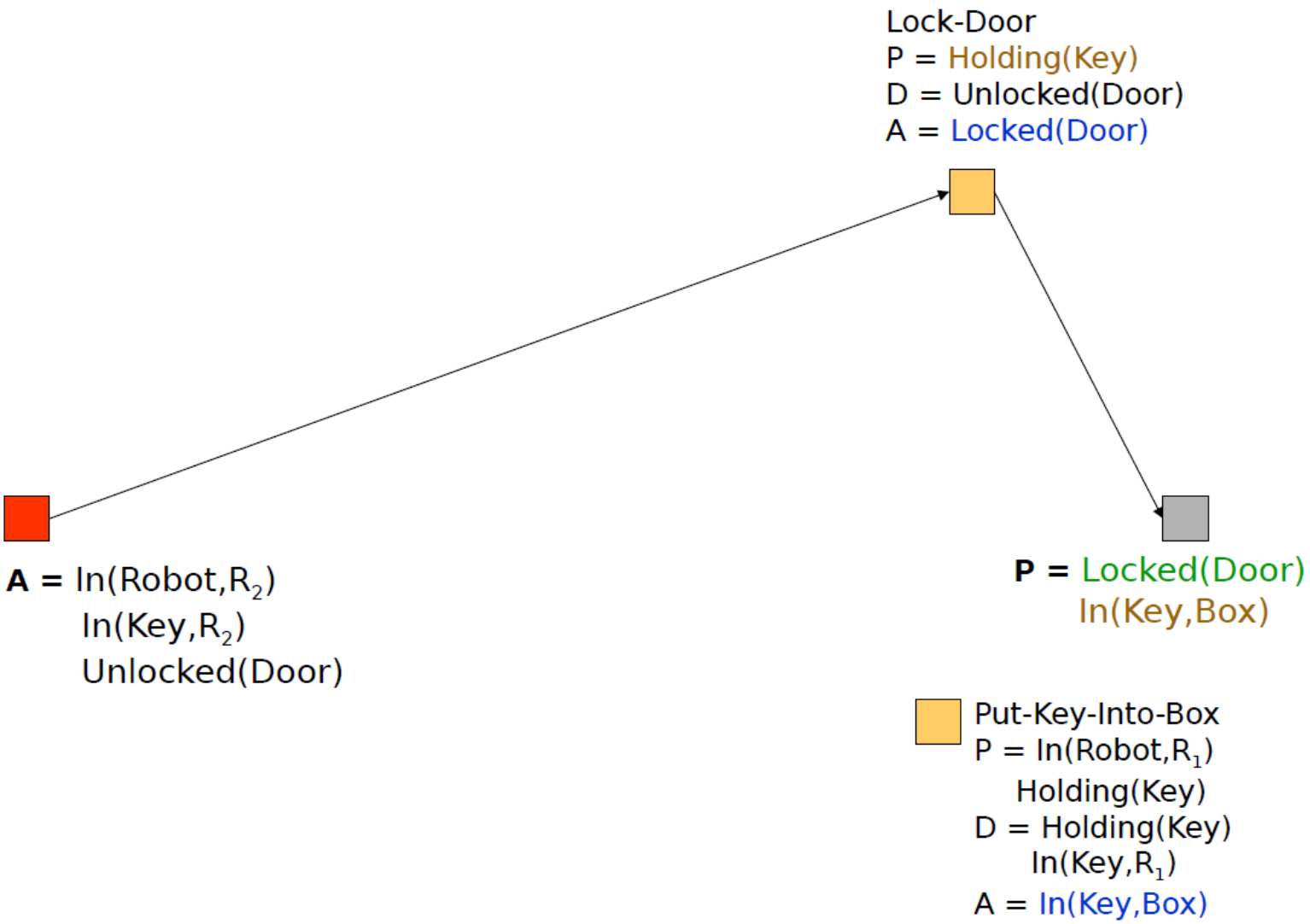
Planification non-linéaire



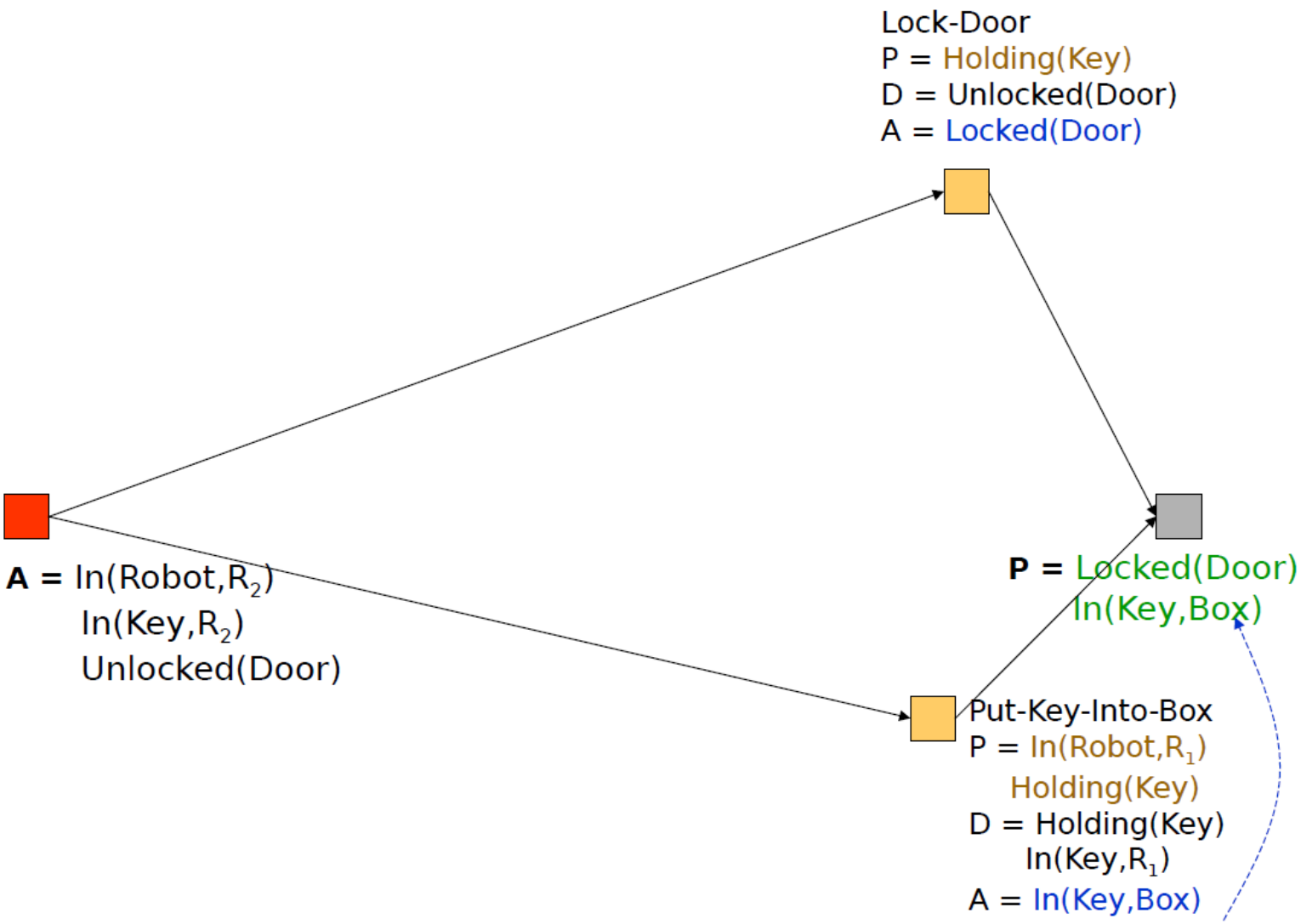
Planification non-linéaire



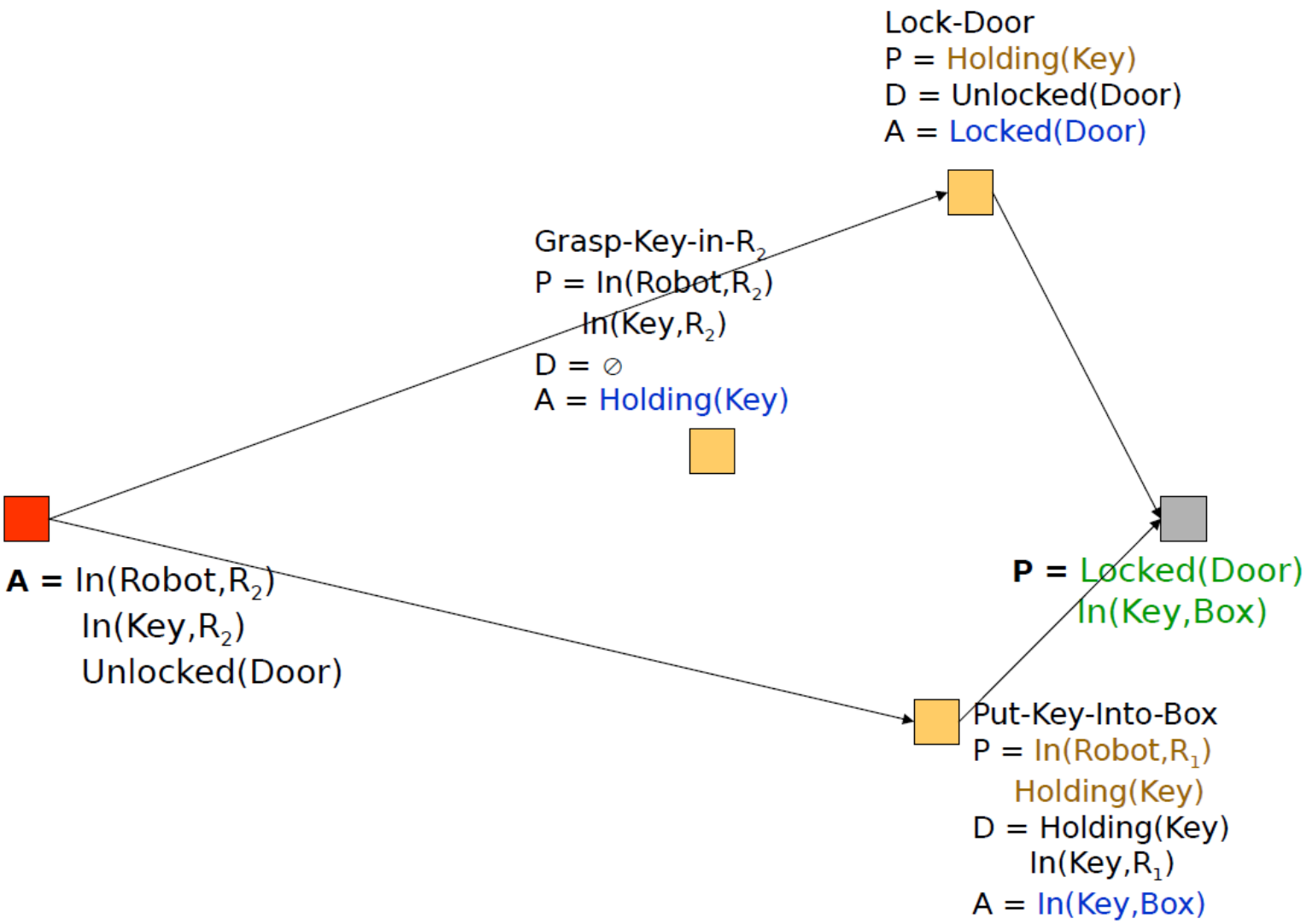
Planification non-linéaire



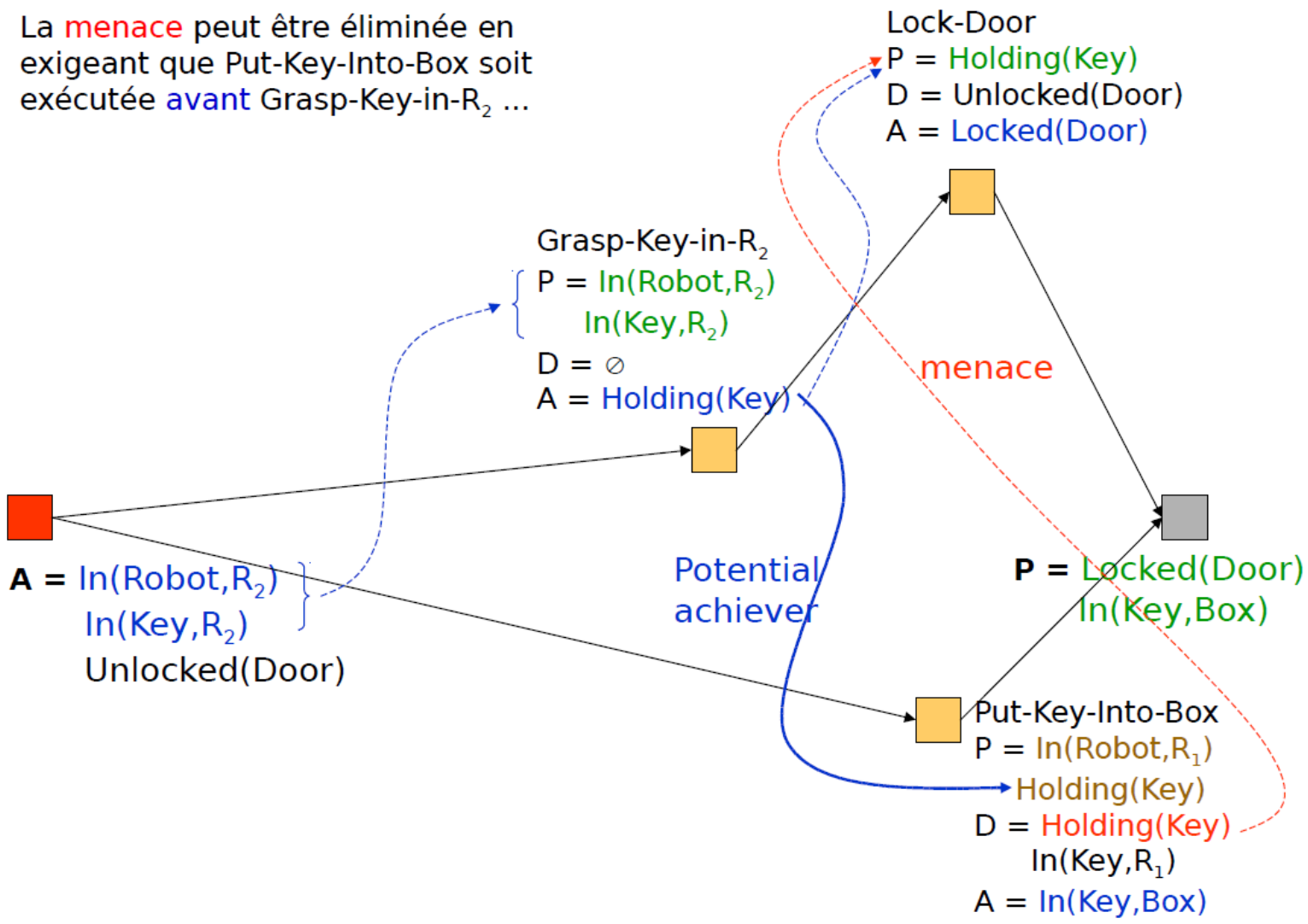
Planification non-linéaire



Planification non-linéaire

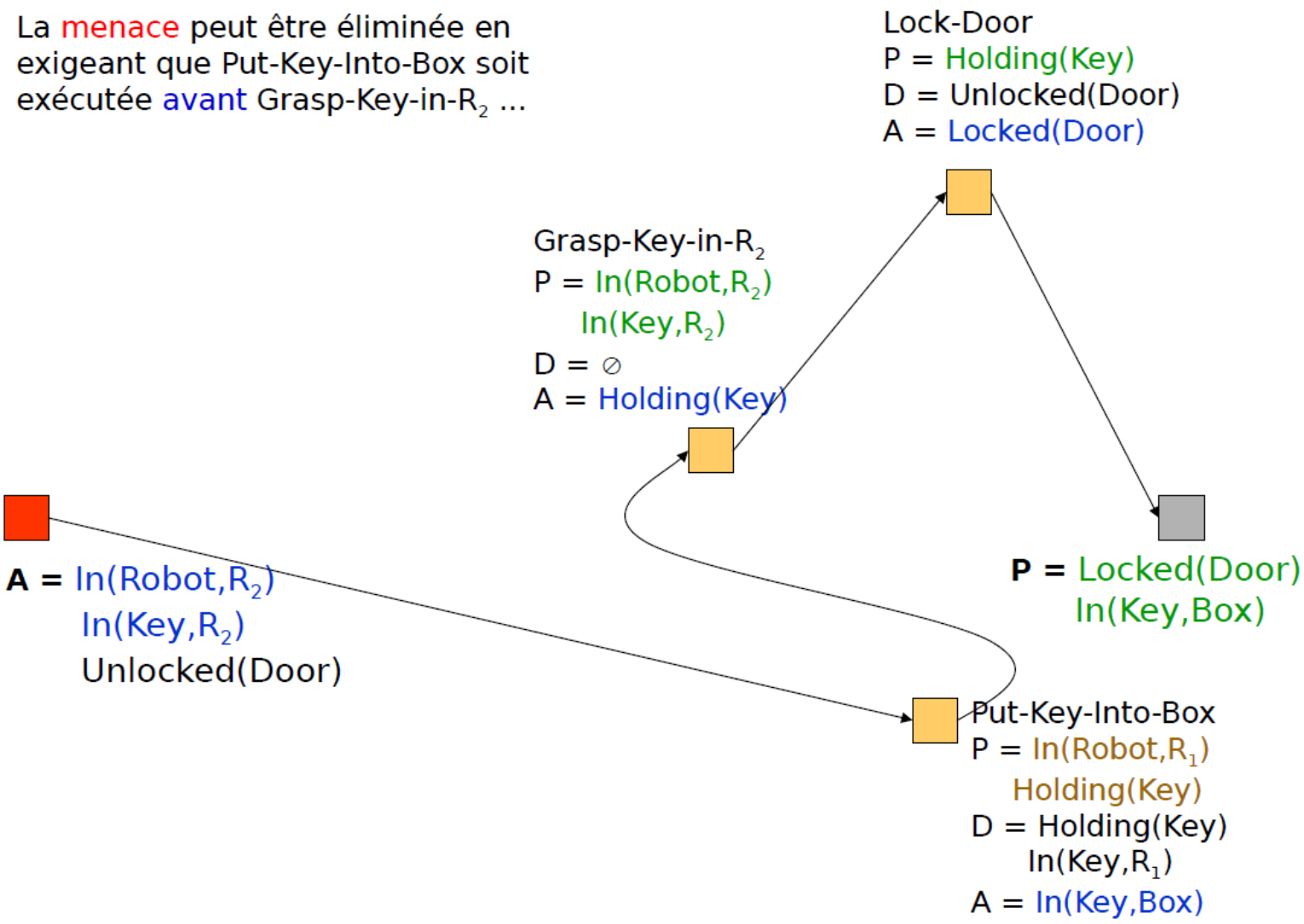


Planification non-linéaire

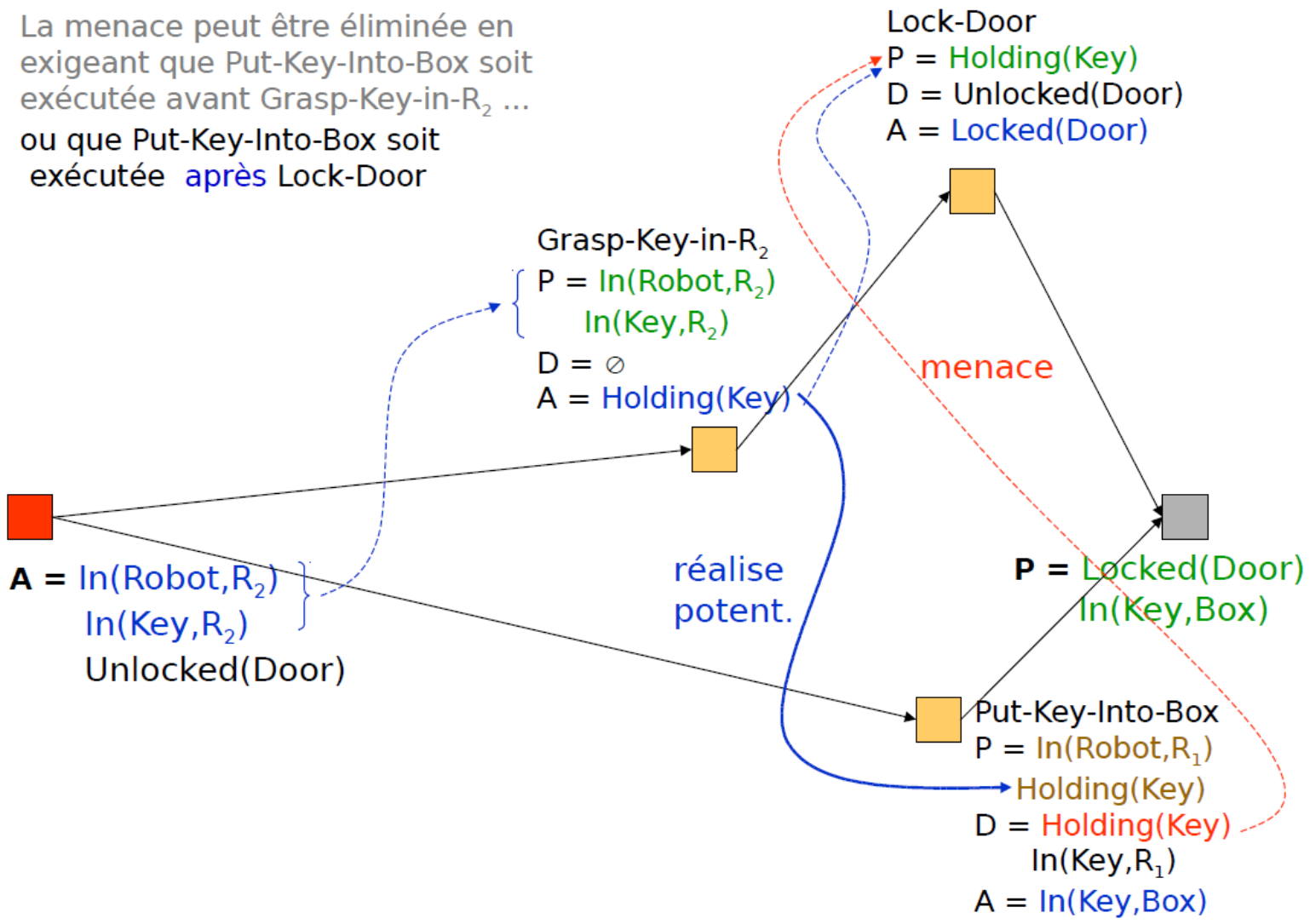


Planification non-linéaire

La menace peut être éliminée en exigeant que Put-Key-Into-Box soit exécutée avant Grasp-Key-in-R_2 ...



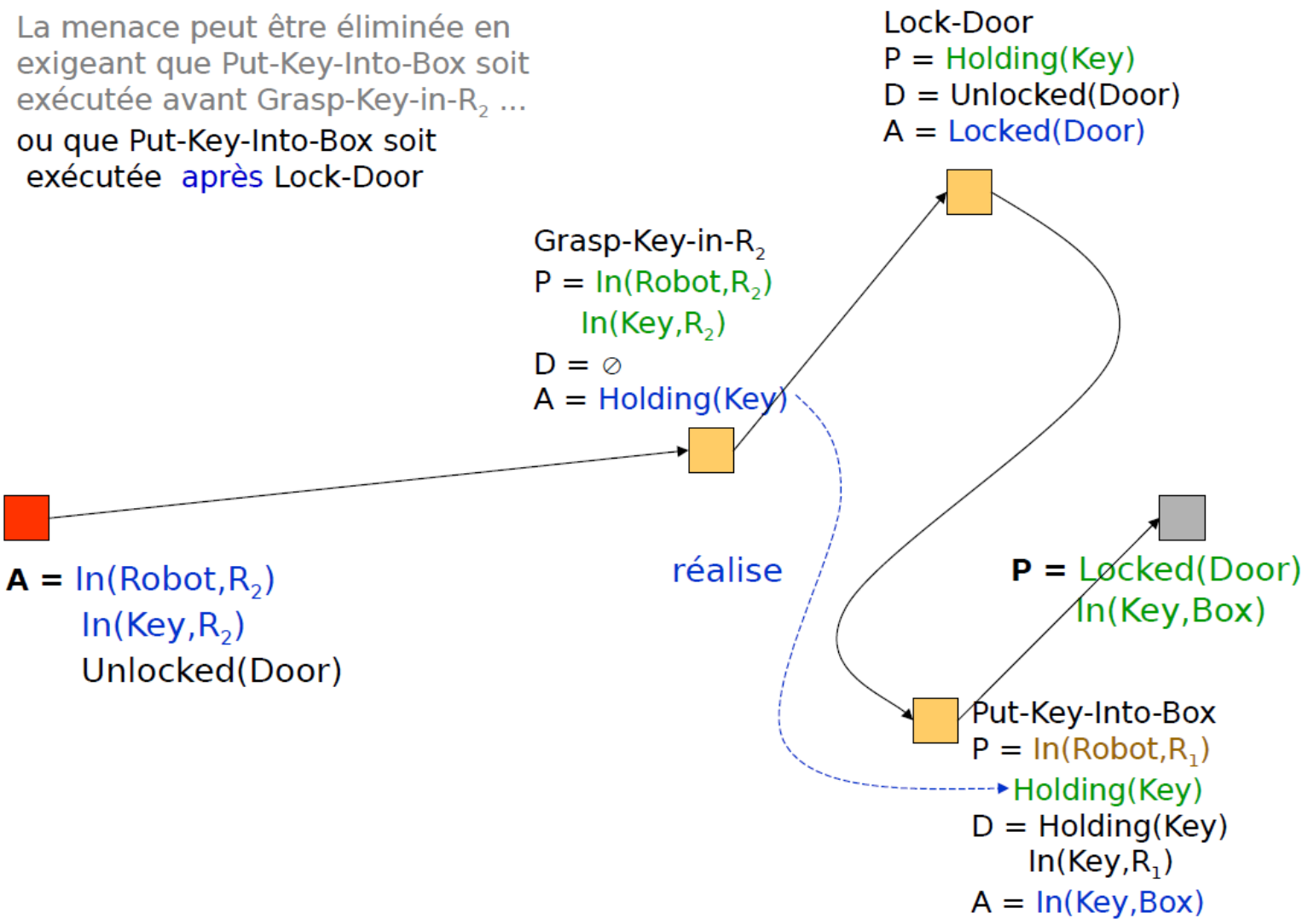
Planification non-linéaire



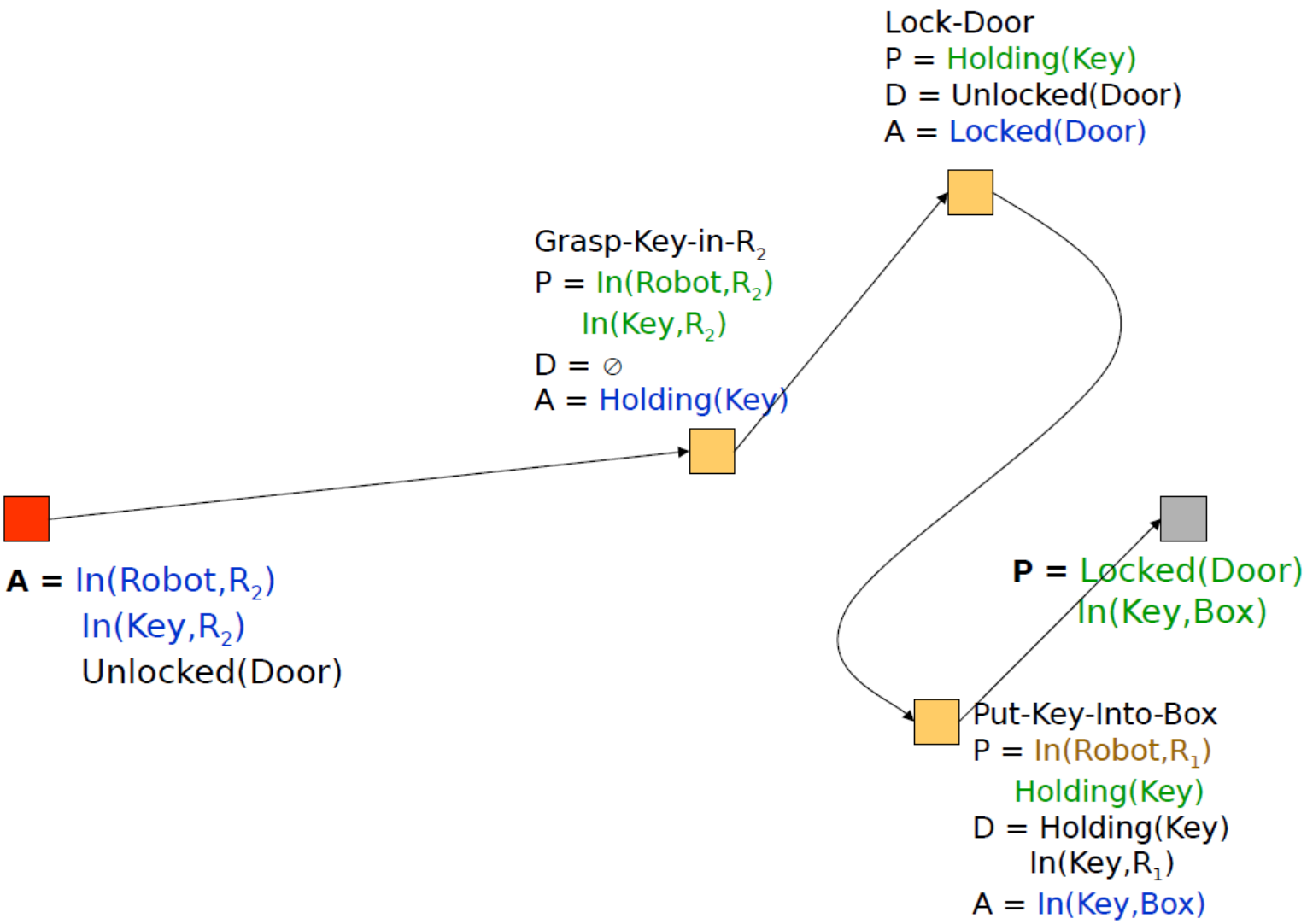
Planification non-linéaire

La menace peut être éliminée en exigeant que Put-Key-Into-Box soit exécutée avant Grasp-Key-in-R_2 ...

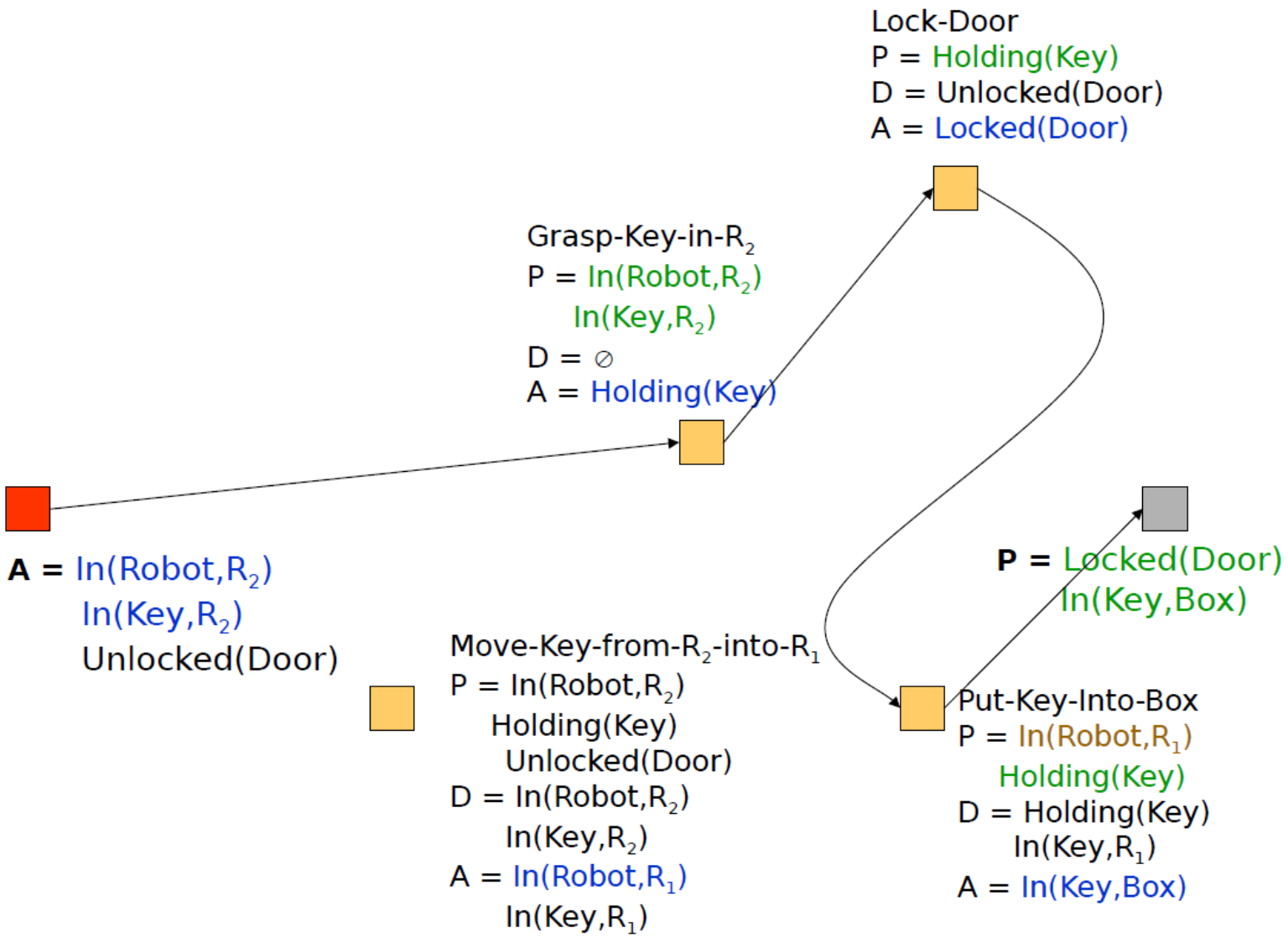
ou que Put-Key-Into-Box soit exécutée après Lock-Door



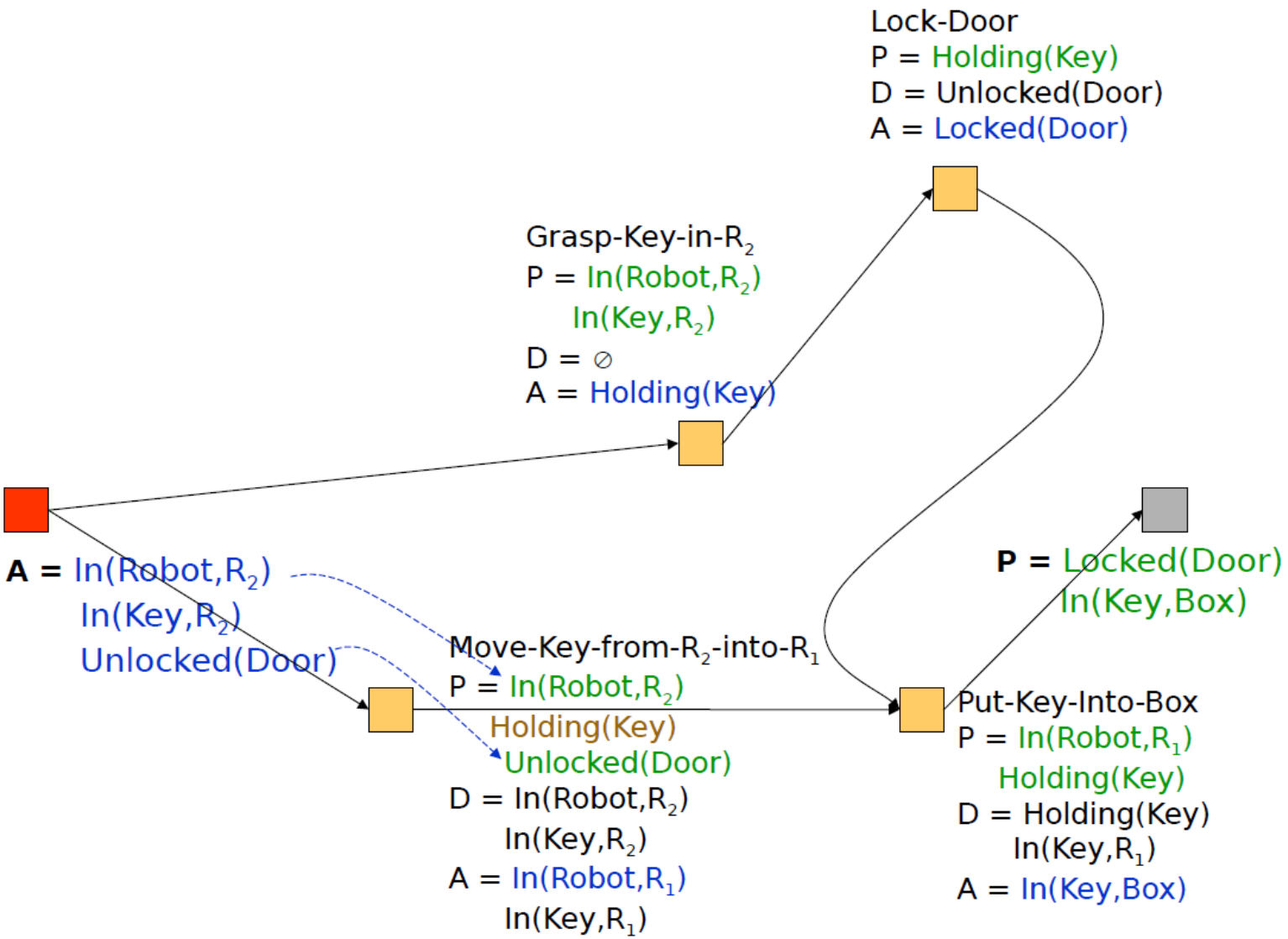
Planification non-linéaire



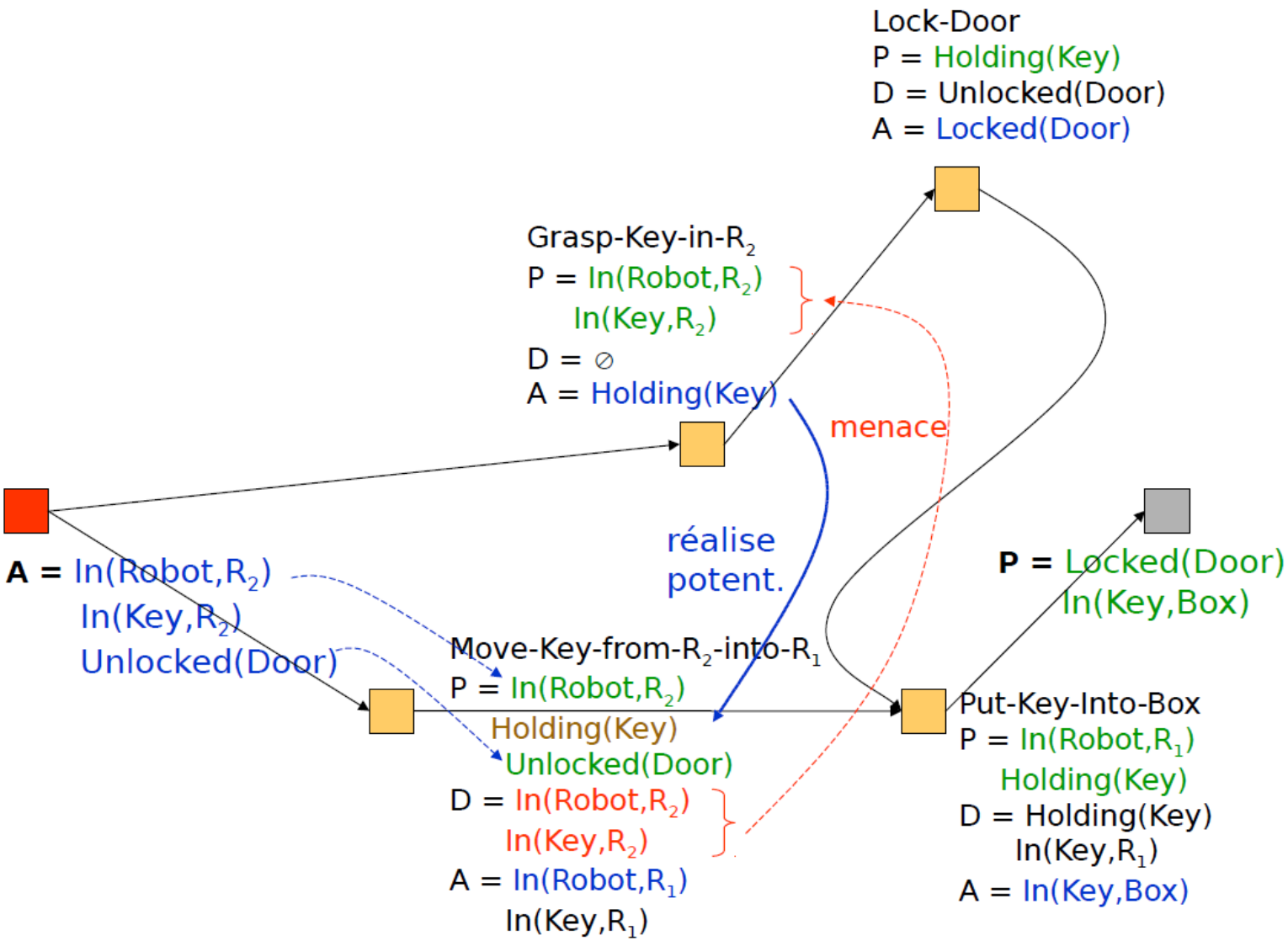
Planification non-linéaire



Planification non-linéaire

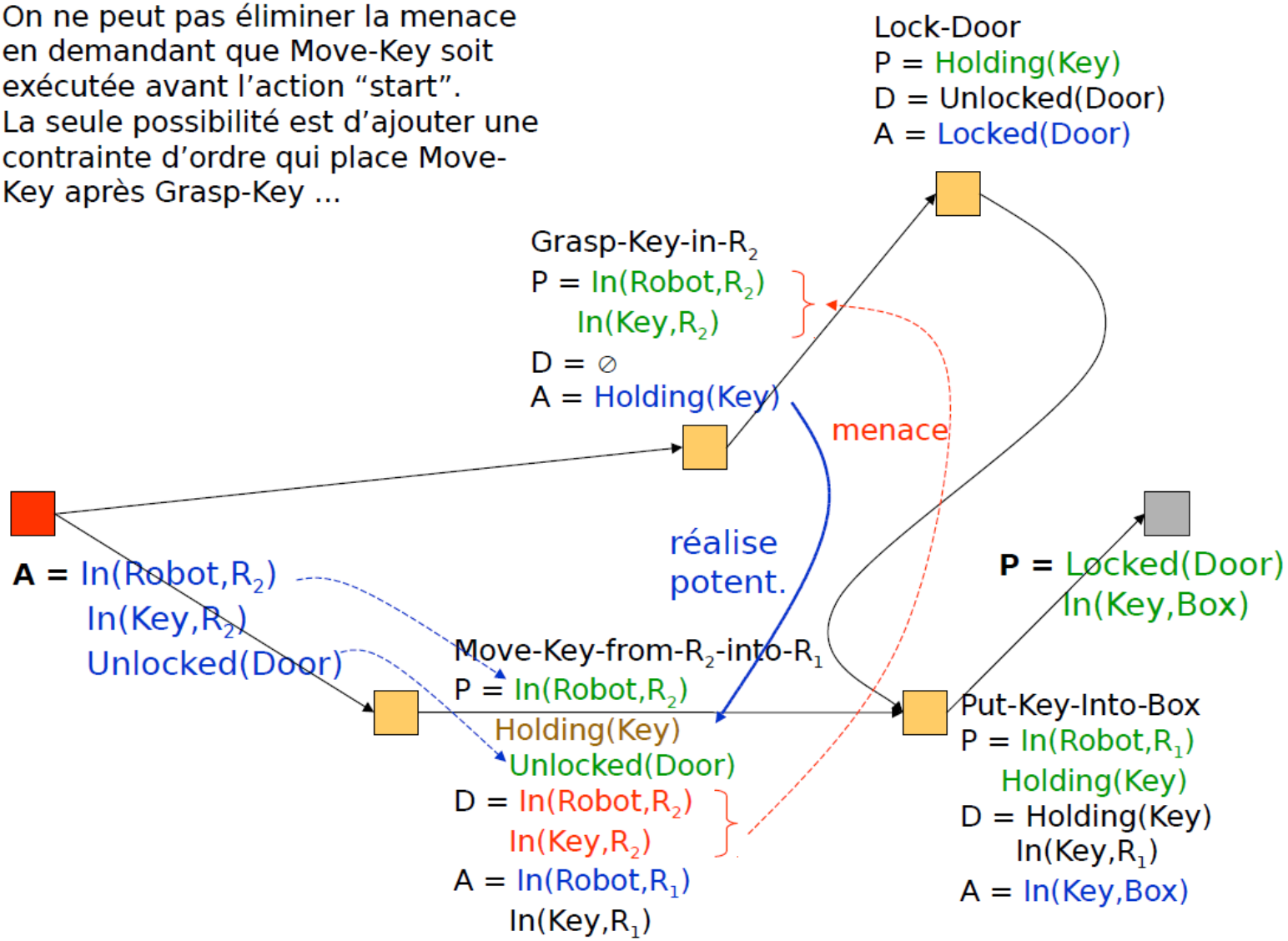


Planification non-linéaire

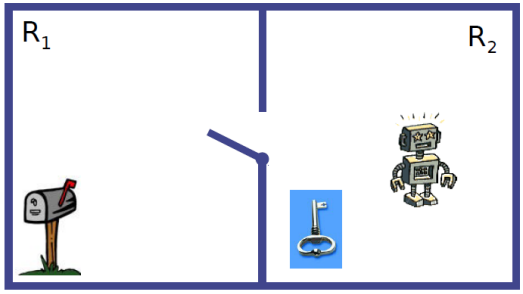


Planification non-linéaire

On ne peut pas éliminer la menace en demandant que Move-Key soit exécutée avant l'action "start".
La seule possibilité est d'ajouter une contrainte d'ordre qui place Move-Key après Grasp-Key ...



Planification non-linéaire



On ne peut pas éliminer la menace en demandant que Move-Key soit exécutée avant l'action "start".
La seule possibilité est d'ajouter une contrainte d'ordre qui place Move-Key après Grasp-Key ...
Mais il y a une autre menace ...
La seule possibilité d'éliminer les deux menaces est de placer Move-Key après Grasp-Key et avant Lock-Door

Lock-Door
P = Holding(Key)
D = Unlocked(Door)
A = Locked(Door)

Grasp-Key-in-R_2
P = In(Robot,R_2)
In(Key,R_2)
D = ⊘
A = Holding(Key)

menace

réalise potent.

A = In(Robot,R_2)
In(Key,R_2)
Unlocked(Door)

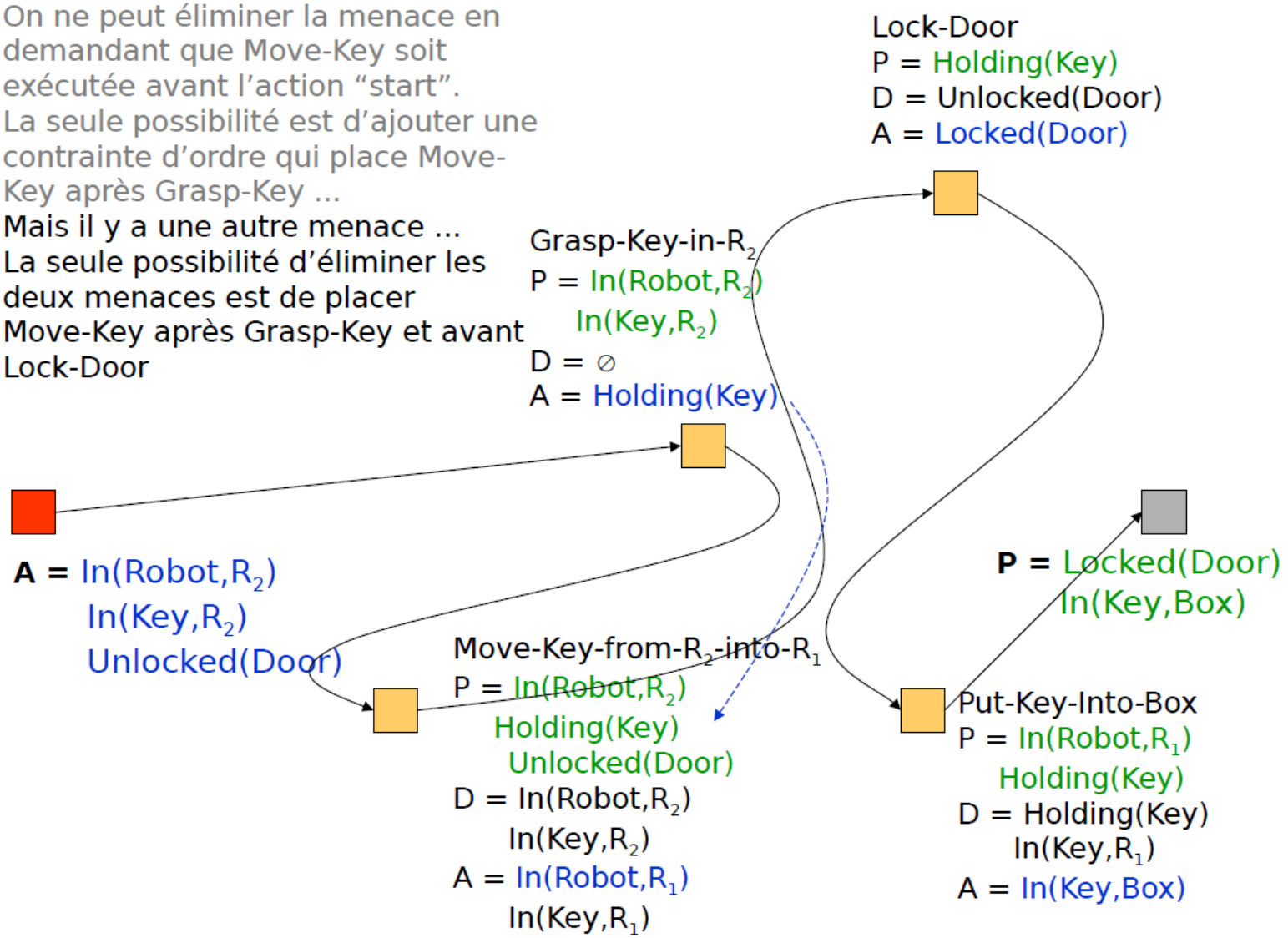
Move-Key-from-R_2-into-R_1
P = In(Robot,R_2)
Holding(Key)
Unlocked(Door)
D = In(Robot,R_2)
In(Key,R_2)
A = In(Robot,R_1)
In(Key,R_1)

P = Locked(Door)
In(Key,Box)

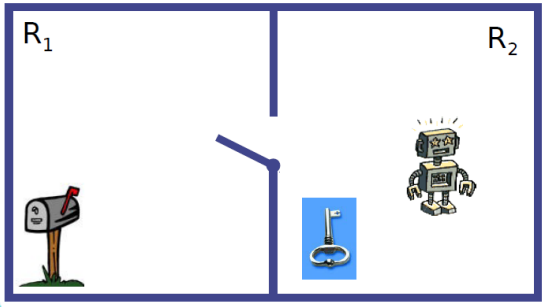
Put-Key-Into-Box
P = In(Robot,R_1)
Holding(Key)
D = Holding(Key)
In(Key,R_1)
A = In(Key,Box)

Planification non-linéaire

On ne peut éliminer la menace en demandant que Move-Key soit exécutée avant l'action "start".
La seule possibilité est d'ajouter une contrainte d'ordre qui place Move-Key après Grasp-Key ...
Mais il y a une autre menace ...
La seule possibilité d'éliminer les deux menaces est de placer Move-Key après Grasp-Key et avant Lock-Door



Planification non-linéaire



Le plan est maintenant **complet**: toutes les pré-conditions sont satisfaites et il n'y aucune menace

Lock-Door
P = Holding(Key)
D = Unlocked(Door)
A = Locked(Door)

Grasp-Key-in-R_2
P = In(Robot,R_2)
In(Key,R_2)
D = ⊘
A = Holding(Key)

A = In(Robot,R_2)
In(Key,R_2)
Unlocked(Door)

Move-Key-from-R_2-into-R_1
P = In(Robot,R_2)
Holding(Key)
Unlocked(Door)
D = In(Robot,R_2)
In(Key,R_2)
A = In(Robot,R_1)
In(Key,R_1)

P = Locked(Door)
In(Key,Box)

Put-Key-Into-Box
P = In(Robot,R_1)
Holding(Key)
D = Holding(Key)
In(Key,R_1)
A = In(Key,Box)

Résumé

- La planification représente les états comme des conjonctions de propositions logiques
- Les actions sont des modificateurs
  - Précondition
  - Add-/Delete-lists
- Le chainage avant souffre d'un facteur de branchement trop important
- Le chainage arrière régresse de l'objectif vers l'état initial
- La planification non-linéaire explore un espace de plans partiels (ordre partiel)