CNRS - Toulouse INP - UT3 - UTC - UT2
Institut de Recherche en Informatique de Toulouse

Introduction à la planification

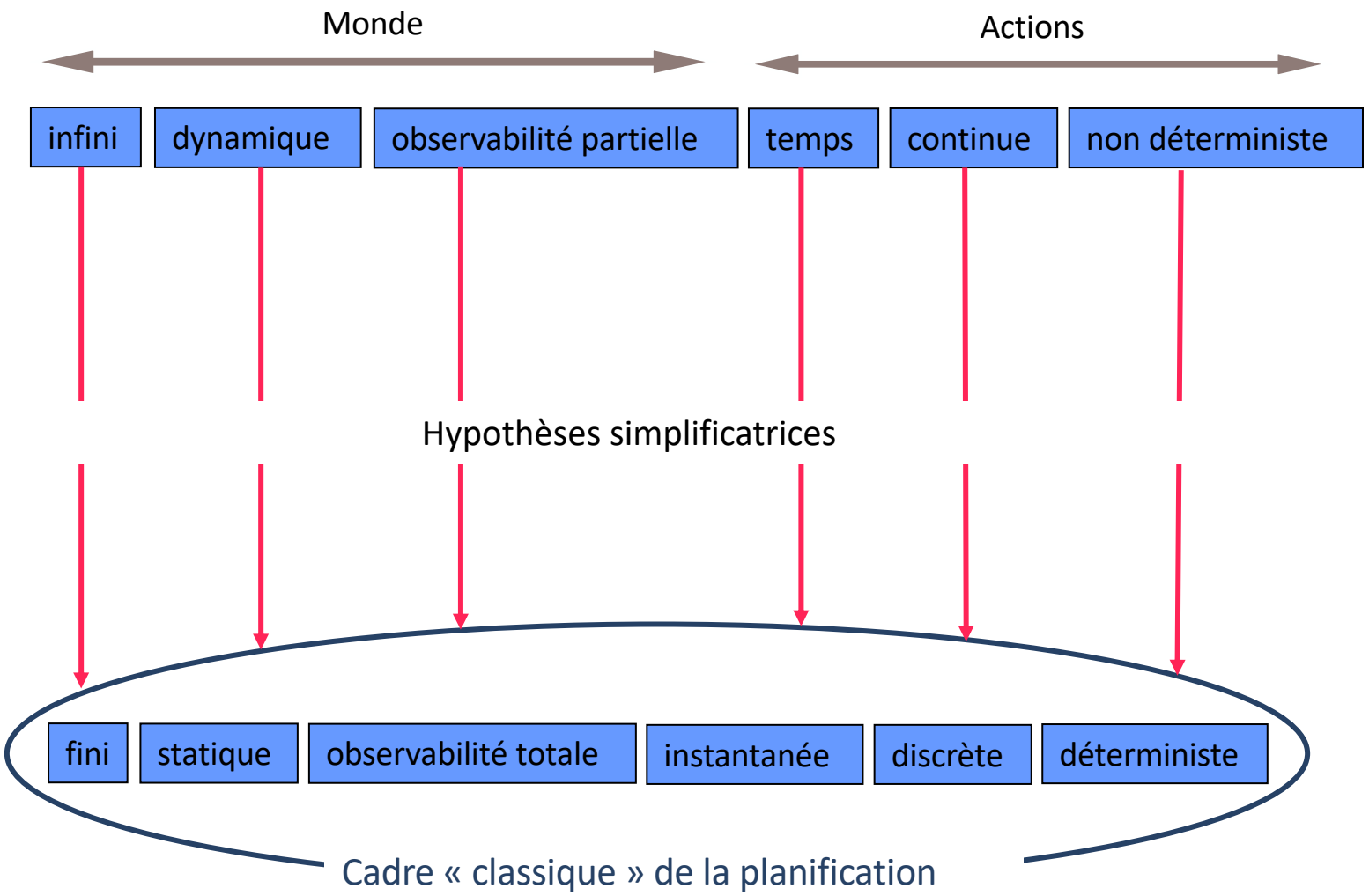
Partie 1 : Un peu d'histoire, planification classique

Tiago de Lima, Frédéric Maris, Ajdin Sumic, Thierry Vidal, Bruno Zanuttini

Le cadre classique

- Le problème général posé par la synthèse d'un plan-solution est très complexe car la planification implique trois étapes :
  - la sélection d'**actions applicables** (parmi les très nombreuses actions disponibles)
  - le choix parmi elles, d'**actions pertinentes** pour se diriger vers le but (qui nécessite de raisonner sur leurs dépendances causales)
  - un **raisonnement sur leurs interactions** pour obtenir un ordonnancement exécutable de ces actions

Le cadre classique



Le cadre classique

- Le problème général posé par la synthèse d'un plan-solution est très complexe car la planification implique trois étapes :
  - la sélection d'**actions applicables** (parmi les très nombreuses actions disponibles)
  - le choix parmi elles, d'**actions pertinentes** pour se diriger vers le but (qui nécessite de raisonner sur leurs dépendances causales)
  - un **raisonnement sur leurs interactions** pour obtenir un ordonnancement exécutable de ces actions

Le cadre classique

- Le problème général posé par la synthèse d'un plan-solution est très complexe car la planification implique trois étapes :
  - la sélection d'**actions applicables** (parmi les très nombreuses actions disponibles)
  - le choix parmi elles, d'**actions pertinentes** pour se diriger vers le but (qui nécessite de raisonner sur leurs dépendances causales)
  - un **raisonnement sur leurs interactions** pour obtenir un ordonnancement exécutable de ces actions

- Dans le cadre classique, le problème consistant à…
  - prouver l'existence d'un plan-solution de taille polynomiale est NP-complet
  - Prouver l'existence d'un plan-solution est PSPACE-complet

Le cadre classique

- Pourquoi travailler dans ce cadre restrictif ?
  - développer des algorithmes indépendants du domaine qui soient performants pour des représentations simples de l'action
  - tirer ensuite parti des progrès effectués pour enrichir cette représentation (prise en compte de ressources, du temps, de l'incertain…) tout en restant efficace
  - objectif : finir par pouvoir traiter des problèmes réels

Le cadre classique

- Développement important de la discipline
  - Conférences ICAPS (International Conference on Automated Planning and Scheduling) http://www.icaps-conference.org/
  - Compétitions IPC (International Planning Competition) https://ipc2023.github.io/
  - Développement de langages de représentation de domaines / problèmes de planification : STRIPS, ADL, PDDL
  - Benchmarks de plus en plus inspirés de problèmes réels (prise en compte de ressources, du temps…)

Langages pour la planification

- Le **langage STRIPS** : exemple du « domaine des cubes »
  - Représentation STRIPS du problème : état initial et but

Etat initial But

```
État init : {sur(A,B), surTable(B), sur(C,D), surTable(D),libre(A), libre(C)}
But : {sur(B, A), surTable(A), sur(D, C), surTable(C)}
```

  - Représentation STRIPS des opérateurs (deux sont nécessaires)

```
# prendre le cube ?x qui est sur le cube ?y, le poser sur le cube ?z
Poser-sur-cube(?x, ?y, ?z) :
    Prec = {sur(?x, ?y), libre(?x), libre(?z)}
    Add = {sur(?x, ?z), libre(?y)}
    Del = {sur(?x, ?y), libre(?z)}

Poser-sur-table (…) -> À VOUS DE LE COMPLÉTER
```

Langages pour la planification

- Le langage ADL
  - Sous-ensemble de la logique du premier ordre : un opérateur o est représenté par son nom et un doublet <préconditions, effets>. Ajouts et retraits sont groupés dans les effets (ajouts : littéraux positifs, retraits : littéraux négatifs). ADL permet l'utilisation de connecteurs logiques et de quantificateurs

  - dans Pre(o) et Eff(o), $\wedge$ représente une conjonction de formules
  - dans Eff(o), $\rightarrow$ permet de représenter un effet conditionnel
  - dans Pre(o) et dans les antécédent des effets conditionnels, $\vee$ permet de représenter une précondition disjonctive
  - dans Pre(o) et Eff(o), $\forall$ et $\exists$ représentent la quantification universelle et la quantification existentielle

Langages pour la planification

- Le langage ADL : exemple du « domaine des cubes »
  - Représentation ADL des opérateurs (un seul suffit)

```
    # prendre le cube ?x qui est sur ?y (cube, table), le poser sur ?z (cube,
table)
    Poser-sur :
        Nom(poser-sur) = poser-sur(?x, ?y, ?z)
         Pre(poser-sur) = sur(?x, ?y) ∧ libre(?x) ∧ libre(?z) ∧
             ≠(?x, ?z) ∧ ≠ (?y, ?z)
         Eff(poser-sur) = sur(?x, ?z) ∧ ¬sur(?x, ?y) ∧
               (≠(?y, Table) → libre(?y)) ∧ (≠(?z, Table) → ¬libre(?z))
```

Langages pour la planification

- Le langage PDDL
  - Prise en compte : de durées, d'effets dépendants du temps, de ressources continues…
  - typage
  - contraintes d'égalité
  - effets conditionnels
  - préconditions disjonctives
  - quantification universelle
  - mise à jour de variables d'état…

Langages pour la planification

- Le **langage PDDL** : exemple du « domaine des cubes »
  - Représentation PDDL des opérateurs (un seul suffit)

```
(define (domain blocksword)
     (: requirements :strips :equality :conditional-effects)
     (: predicates (on ?x ?y) (clear ?x))

     # prendre le cube ?x qui est sur ?y (cube, table), le poser sur ?z (cube, table)
     (: action puton
         : parameters (?x ?y ?z)
            : precondition
                (and (on ?x ?y)(clear ?x)(clear ?z)
                   (not (= ?y ?z)) (not (= ?x ?y))
                          (not (= ?x ?z)) (not (= ?x Table)))
            : effect
              (and (on ?x ?z) (not (on ?x ?y))
                     (when (not (eq ?y Table)) (clear ?y))
                        (when (not (eq ?z Table)) (not (clear ?z)))))
```

Langages pour la planification

- Le langage PDDL : exemple du « domaine satellite »
  - Observations par des satellites équipés de différents instruments

    - version **Strips** : relativement simple
    - version **numérique** : gestion de l'énergie (ressource consommable), capacité de stockage d'information limitée (plans pour acquérir toute l'info avec minimum d'énergie)
    - **temporelle simple** : utilisation simultanée possible de plusieurs satellites pour acquérir les informations recherchées (plans pour acquérir l'information au plus vite)
    - **temporelle** : temps critiques pour l'acquisition, temps de calibration différents…
    - **complexe** : version temporelle combinée avec la version numérique

Langages pour la planification

- Le langage PDDL : exemple du « domaine satellite »
  - Observations par des satellites équipés de différents instruments. Version Strips

```
(define (domain satellite)
    (:requirements :strips :equality :typing)
    (:types satellite direction instrument mode)
    (:predicates
        (on-board ?i - instrument ?s - satellite)
        (supports ?i - instrument ?m - mode)
        (pointing ?s - satellite ?d - direction)
        (power-avail ?s - satellite)
        (power-on ?i - instrument)
        (calibrated ?i - instrument)
        (have-image ?d - direction ?m - mode)
        (calibration-target ?i - instrument ?d - direction))
```

Langages pour la planification

- Le langage PDDL : exemple du « domaine satellite »
  - Observations par des satellites équipés de différents instruments. Version Strips

```
(:action turn-to
   :parameters    (?s - satellite ?d-new - direction ?d-prev -
direction)
   :precondition (and (pointing ?s ?d-prev)
                      (not (= ?d-new ?d-prev)))
   :effect        (and (pointing ?s ?d-new)
                      (not (pointing ?s ?d-prev))))

(:action switch-on
   :parameters    (?i - instrument ?s - satellite)
   :precondition (and (on-board ?i ?s)
                      (power-avail ?s))
   :effect        (and (power-on ?i)
                      (not (calibrated ?i))
                      (not (power-avail ?s))))…
```

Langages pour la planification

- Le langage PDDL : exemple du « domaine satellite »
  - Observations par des satellites équipés de différents instruments. Version Temporelle

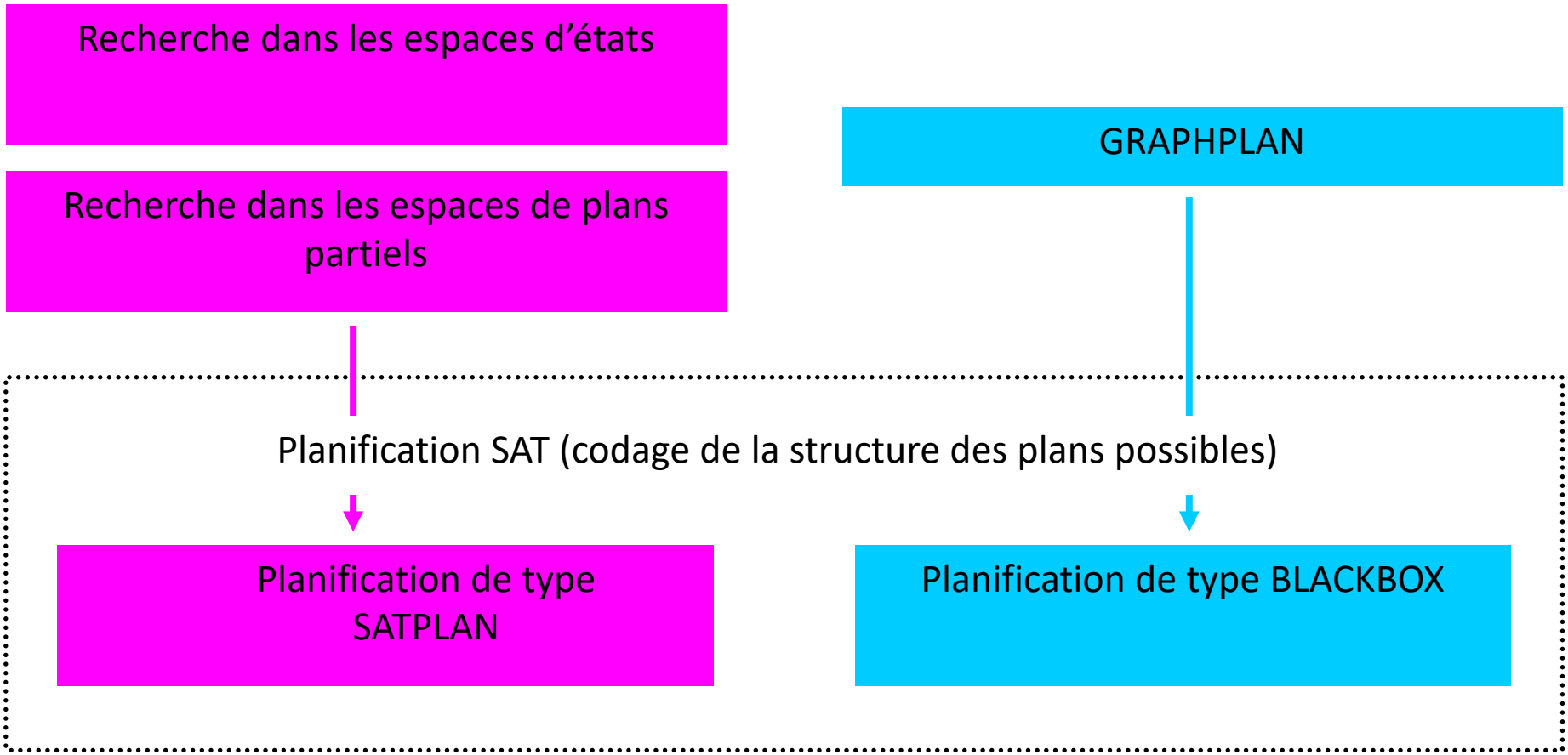
```
(:durative-action turn_to
    :parameters (?s - satellite ?d-new - direction ?d-prev - direction)
    :duration   (= ?duration (slew-time ?d-prev ?d-new))
    :condition  (and (at start (pointing ?s ?d-prev))
                     (over all (not (= ?d-new ?d-prev))))
    :effect     (and (at end (pointing ?s ?d_new))
                          (at start (not (pointing ?s ?d-prev)))))
```

Langages pour la planification

- Le langage PDDL : exemple du « domaine satellite »
  - Observations par des satellites équipés de différents instruments. Version numérique

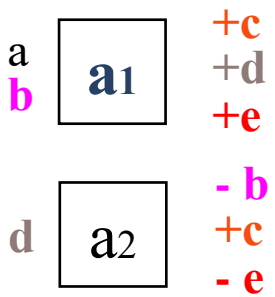
```
(:action turn_to
    :parameters    (?s - satellite ?d-new - direction ?d-prev - direction)
     :precondition (and (pointing ?s ?d-prev)
                        (not (= ?d_-new ?d-prev))
                        (>= (fuel ?s)(slew-time ?d-new ?d-prev)))
     :effect       (and (pointing ?s ?d-new)
                        (not (pointing ?s ?d-prev))
                        (decrease (fuel ?s)(slew-time ?d-new ?d-prev))
                        (increase (fuel-used)(slew-time ?d-new ?d_prev))))
```

Algorithmes pour la synthèse de plans



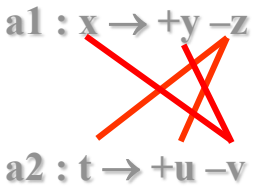
Classification des interactions

- **Interactions positives :**
  - **Effets multiples :** action qui produit plusieurs fluents : **action a1**
  - **Add/Add :** ∃ f, f ∈ Add(a1) ∩ Add(a2) : **fluent c**
  - **Add/Prec** : ∃ f, f ∈ Add(a1) ∩ Prec(a2) : **fluent d**
- **Interactions négatives :**
  - **Effets antagonistes :** ∃ f, f ∈ Add(a1) ∩ Del(a2) : **fluent e**
  - **Interactions croisées :** ∃ f, f ∈ Del(a2) ∩ Prec(a1) : **fluent b**



Actions indépendantes

- Deux **actions** a1, a2 sont **indépendantes** (noté a1 # a2) ssi elles n'ont pas d'interactions négatives càd :
  - Del(a1) ∩ (Prec(a2) ∪ Add(a2)) = ∅ et
  - Del(a2) ∩ (Prec(a1) ∪ Add(a1)) = ∅



- **Ensemble** d'actions **indépendantes :**
  - Q est un ensemble d'actions indépendantes ou **ensemble indépendant** ssi toutes les actions Ai qui le composent sont indépendantes 2 à 2 ;
  - **Application d'un ensemble indépendant (chaînage avant) :**
    - un ensemble indépendant Q est applicable sur un état E ssi : $\cup Prec(Ai) \subseteq E$
    - l'état résultant est l'ensemble de fluents :

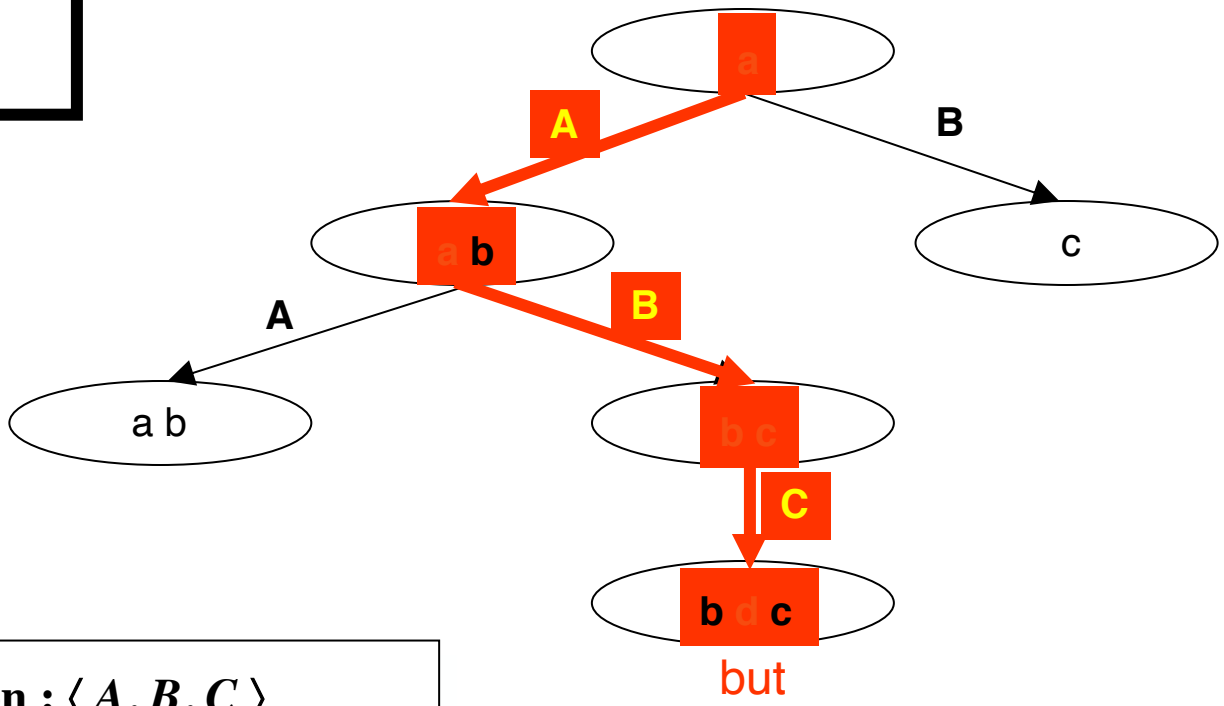
$$E \uparrow Q = (E - \cup Del(Ai)) \cup (\cup Add(Ai))$$

Recherche dans les espaces d'Etats

$A : a \rightarrow +b$

$B : a \rightarrow +c\ -a$

$C : b\ c \rightarrow +d$



Plan solution : $\langle A, B, C \rangle$

Recherche dans les espaces de plans partiels

$A : a \rightarrow +b$

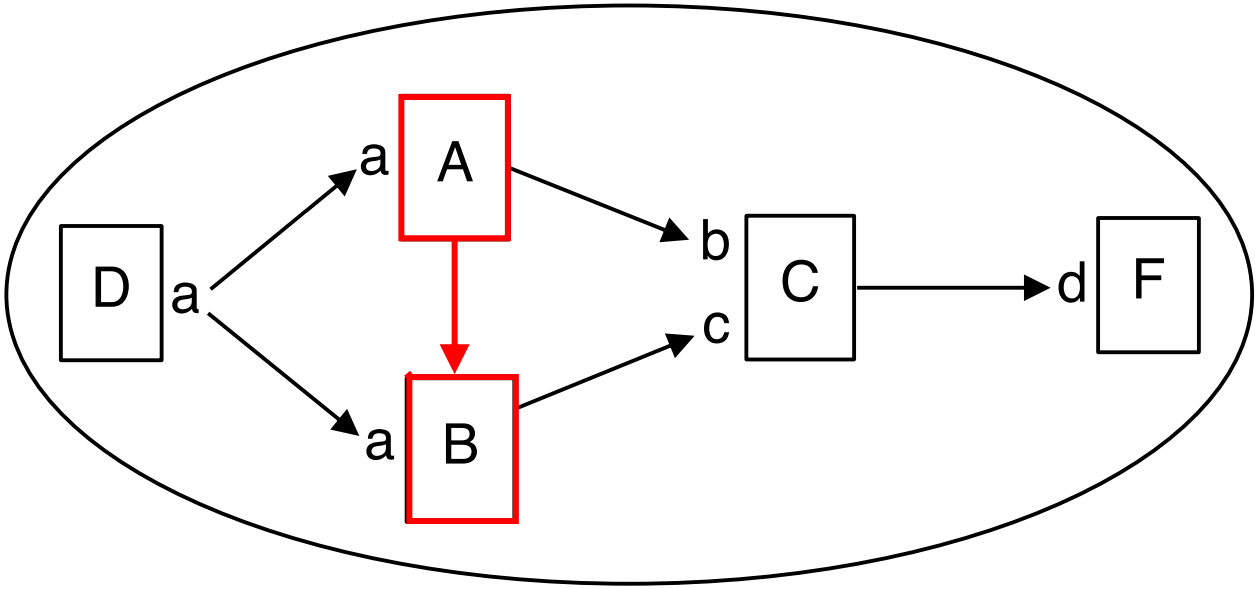
$B : a \rightarrow +c\ -a$

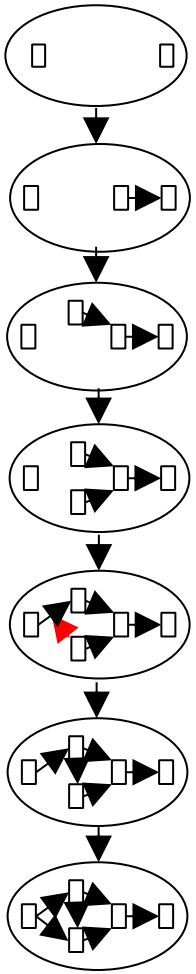
$C : b\ c \rightarrow +d$

Plan solution = {*Actions*, *Contraintes*}

- *Actions* = {A, B, C}
- *Contraintes* = {(A, B), (A, C), (B, C)}

post-traité, donne : <A, B, C>





Principes du planificateur GRAPHPLAN

- GRAPHPLAN **sépare la planification en deux procédures** :
  - la construction (complexité polynomiale en temps et en espace par rapport à la taille des données du problème) du graphe de planification
  - la recherche d'une solution potentielle dans le sous-arbre extrait de ce graphe (NP), qui peut être réalisée par différentes méthodes
- **Le graphe fournit de nombreuses informations** susceptibles de fournir des **heuristiques indépendantes du domaine** pour les méthodes classiques (recherche dans les espaces d'états…), il peut aussi être adapté pour la prise en compte de ressources et du temps

Définitions

- Dans GRAPHPLAN, deux actions d'un même niveau dans le graphe sont **mutuellement exclusives** (mutex) ssi :
  - elles ne sont pas indépendantes ou,
  - elles ont des préconditions mutex au niveau précédent (elles ne peuvent donc pas être déclenchées en même temps) :

    $\exists$ (p,q) $\in$ Prec(a1) x Prec(a2), telles que p et q sont mutex.

- Deux **fluents** p et q sont **mutex** au niveau i ssi tous les couples d'actions qui les produisent à ce même niveau sont mutex (il n'existe pas de couple d'actions non mutex qui les produise à ce niveau) :

  $\forall$ a1, a2 / p $\in$ Add(a1), q $\in$ Add(a2), a1 et a2 mutex.

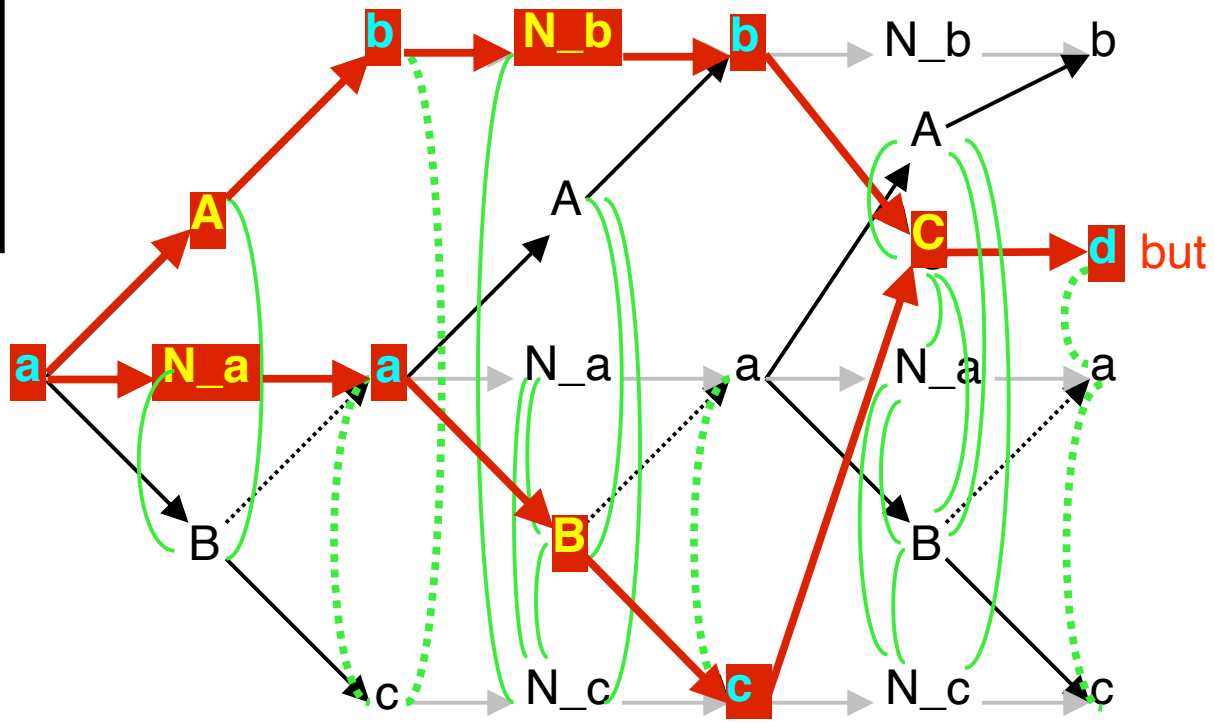
Déroulement de l'algorithme

$A : a \rightarrow +b$

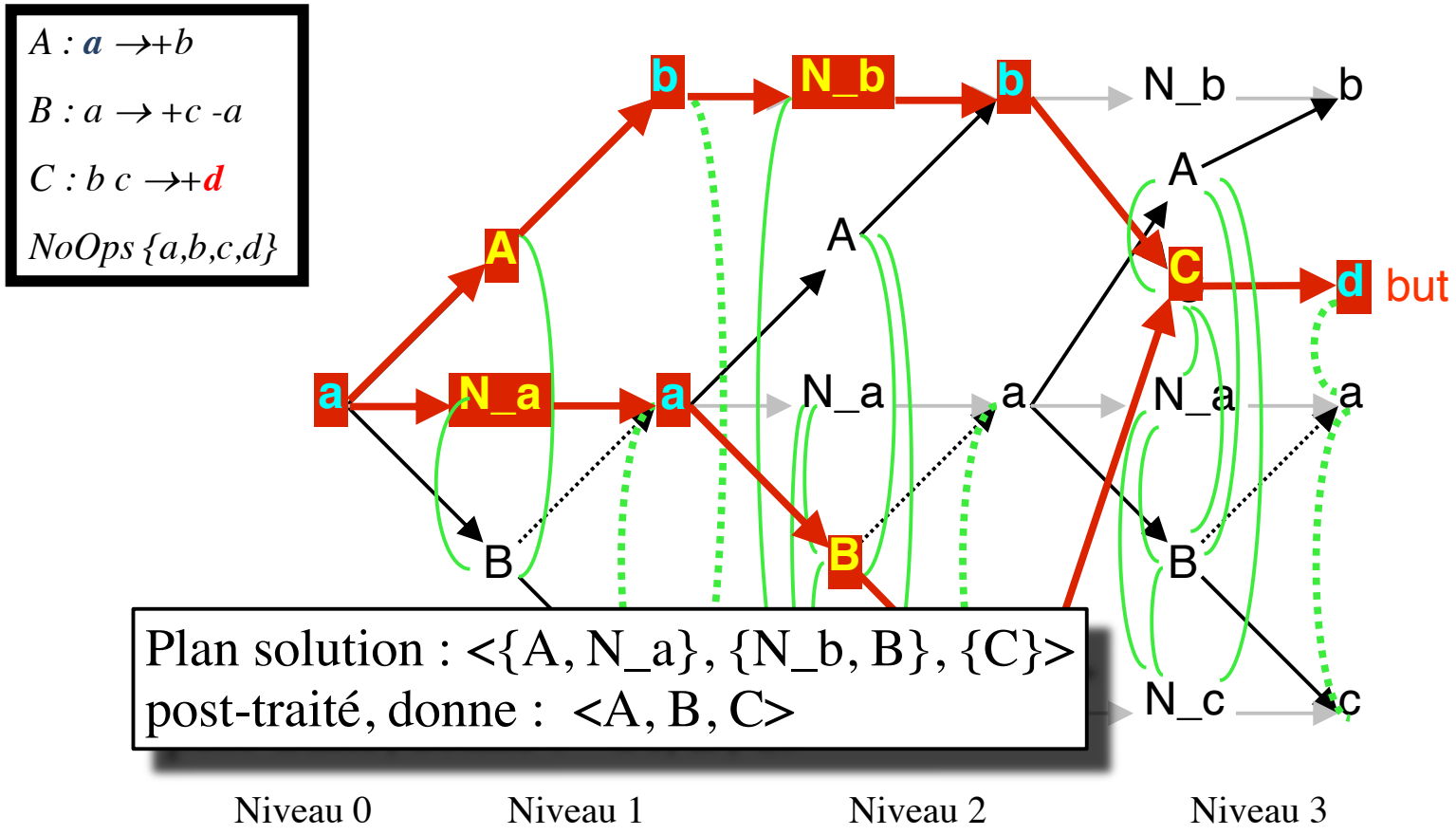
$B : a \rightarrow +c\ -a$

$C : b\ c \rightarrow +d$

$NoOps\ \{a,b,c,d\}$



Déroulement de l'algorithme



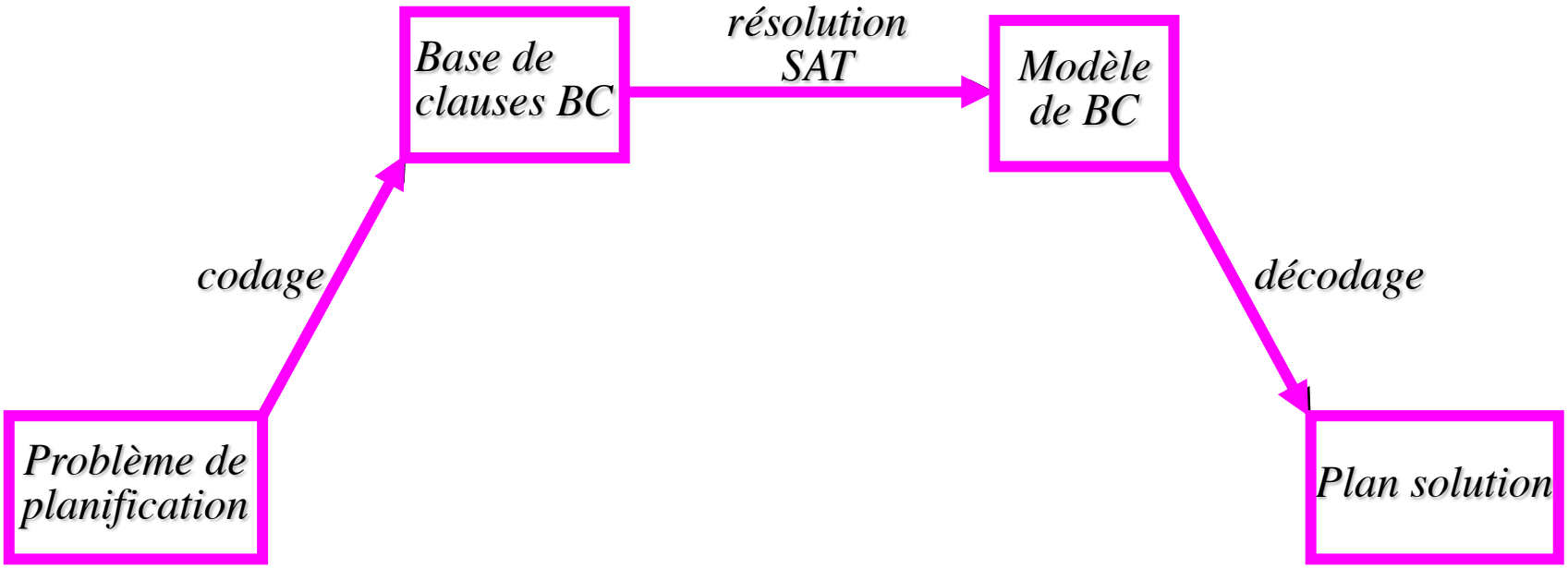
Importance de GRAPHPLAN

- Optimalité des plans-solutions en nombre de niveaux (d'ensembles d'actions indépendantes)

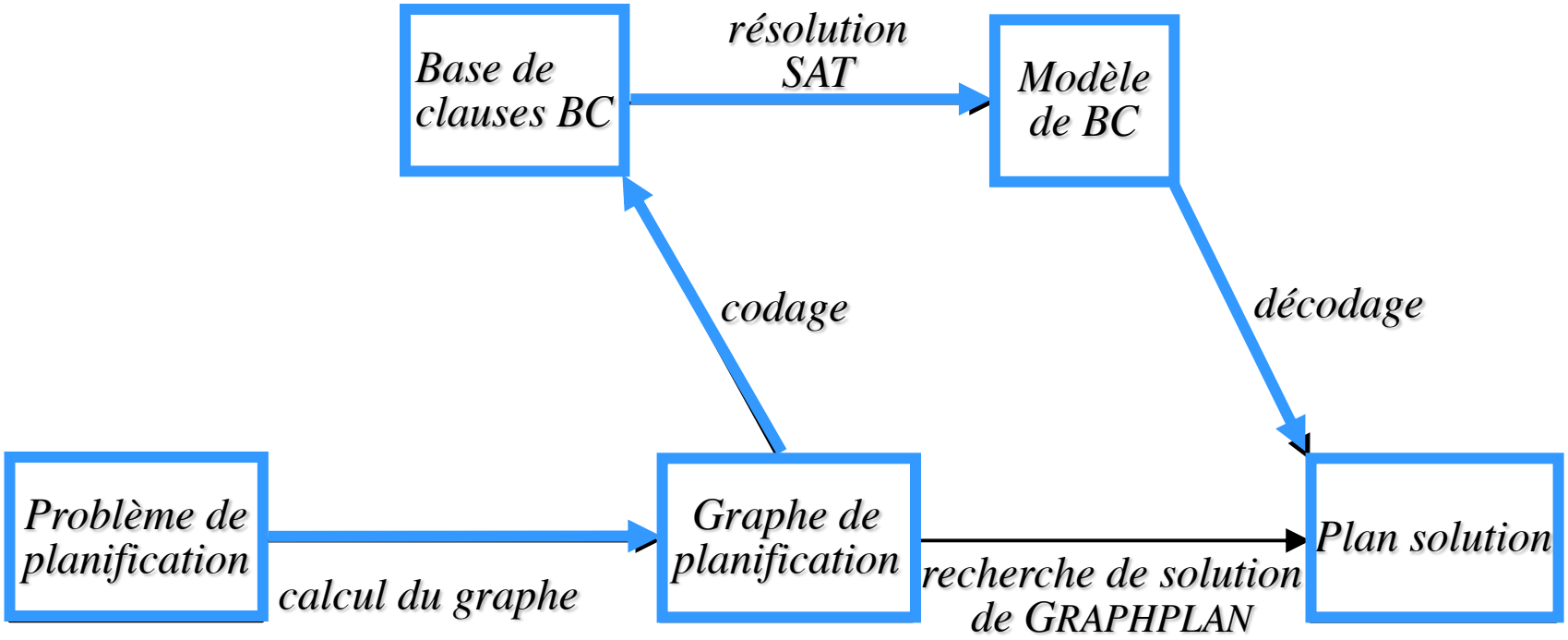
- Utilisation du graphe de planification pour la recherche heuristique dans les espaces d'états (HSP, FF, ALTALT, YASHP…)

- Flexibilité du graphe de planification :
  - Graphe numérique
  - Temporel…

Planification SAT (SATPLAN)



Planification SAT (BLACKBOX)



Exemple de codage

1. $$\left(\bigwedge_{f\in I} f(0)\right)\wedge\left(\bigwedge_{f\in(F-I)} \neg f(0)\right)\wedge\left(\bigwedge_{f\in G} f(k)\right)$$

2. $$\bigwedge_{i\in[1,k]}\bigwedge_{a\in O}\left(a(i)\Rightarrow\left(\bigwedge_{f\in \mathrm{Prec}(a)} f(i-1)\right)\wedge\left(\bigwedge_{f\in Add(a)} f(i)\right)\wedge\left(\bigwedge_{f\in Del(a)} \neg f(i)\right)\right)$$

3. $$\bigwedge_{i\in[1,k]}\bigwedge_{f\in((I\cup F_a)\cap F_d)}\left(\left(f(i-1)\wedge\neg f(i)\right)\Rightarrow\bigvee_{a\in O/\, f\in Del(a)} a(i)\right)$$

$$\bigwedge_{i\in[1,k]}\bigwedge_{f\in(((F-I)\cup F_d)\cap F_a)}\left(\left(\neg f(i-1)\wedge f(i)\right)\Rightarrow\bigvee_{a\in O/\, f\in Add(a)} a(i)\right)$$

4. $$\bigwedge_{i\in[1,k]}\ \bigwedge_{(a_m,a_n)\in O^2/\, m<n}\left(\neg a_m(i)\vee\neg a_n(i)\right)$$
$$\wedge\left(\left(\mathrm{Prec}(a_m)\cap Del(a_n)\neq\varnothing\right)\vee\left(Del(a_m)\cap\mathrm{Prec}(a_n)\neq\varnothing\right)\right)$$
$$\wedge\left(\left(Add(a_m)\cap Del(a_n)=\varnothing\right)\wedge\left(Del(a_m)\cap Add(a_n)=\varnothing\right)\right)$$