Modélisation, implémentation et vérification d'agents avec les GDT

G. SIMON

MASTER LID - Module SMA

Novembre 2008

Travaux de : M. Flouret, D. Fournier, B. Mermet, B. Zanutini

Plan du cours

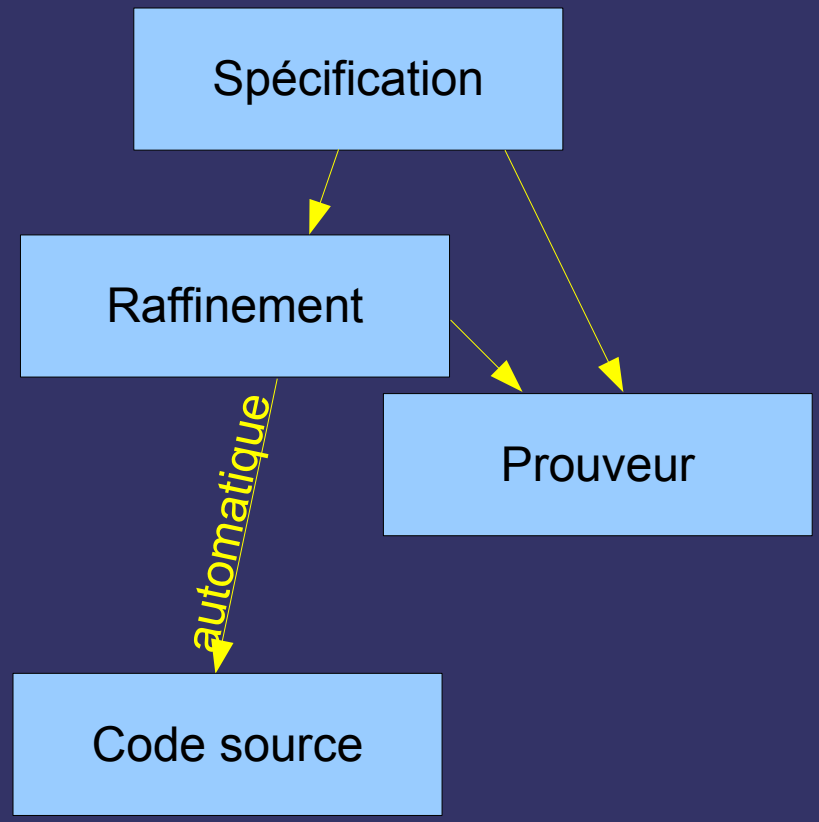
- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

Origines du modèle de GDT

- Pour quoi faire ? : concevoir, tester, implanter et vérifier le comportement d'un SMA
  - Étape d'agentification : agents + buts
  - Comment concevoir le comportement d'un agent devant satisfaire un but donné ?
- Postulats de base
  - Un agent évolue dans un environnement dynamique
  - L'environnement peut évoluer indépendamment d'un agent
    - Évolution propre
    - Actions des autres agents
  - Vérifier la correction du comportement d'un agent signifie vérifier que ce comportement vérifie un certain nombre de propriétés spécifiées.

Caractéristiques attendues du modèle

- Langage de spécification
- Système de preuve
- Compositionnalité du système de preuve
- Vérification de la spécification
- Génération automatique du programme implantant le SMA.



Plan du cours

- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

L'environnement

- Un ensemble V_e de variables typées avec
  - une propriété d'invariance I_e vraie en permanence
    - doit être préservée par les agents
    - permet de modéliser les domaines des var.
  - un ensemble de propriétés stables
    - si deviennent vraies, restent vraies
    - doivent être préservées par les agents
- Un ensemble de constantes

Un agent GDT

- Un sous-ensemble de variables de l'environnement (perception limitée)
- Un ensemble de variables typées dites internes avec :
  - une clause d'initialisation
  - une propriété d'invariance
  - des propriétés stables
- Un ensemble d'actions possibles (cf robots)
- Un comportement décrit par un GDT

Plan du cours

- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

Un GDT (Goal Decomposition Tree)

- Arbre de décomposition de buts
  - Noeuds = buts
  - Branches = opérateurs de décomposition
    - Racine = but principal de l'agent
    - Feuilles = actions
- Précondition
  - Évaluée en premier
  - Doit être préservée par le GDT si plusieurs exécutions possibles
- Triggering context : évalué en second

Notion de but dans le modèle de GDT

- But = but d'achèvement
- 2 types de buts :
  - Buts de progrès: "Je dois augmenter la valeur de x"
  - Buts d'état (permettant d'atteindre un état particulier): "x doit être au moins égal à 4"
- Spécification d'un but : triplet (nom, CS, GPF)
  - Nom: nom du but
  - CS: Condition de Satisfaction
    - A quelle condition le but est-il résolu ?
  - GPF: Guaranted Property in case of Failure
    - Une tentative de résolution d'un but peut échouer
    - Cette propriété reste vraie en cas d'échec

Spécification formelle d'un but

- Les utilisateurs de GDT peuvent choisir leur propre logique L :
  - Logique du premier ordre classiquement utilisée
  - Logiques multi-valuées peuvent aussi être utilisées (attention aux opérateurs !)
- Les CS et GPF sont exprimées en L' qui est une extension de L sur deux mondes :
  - Variables non primées : variables du premier monde
  - Variables primées : variables du second monde

Exemples de CS

- X doit être incrémenté de 1
  - but de progrès déterministe
  - x'=x+1
- X doit augmenter
  - but de progrès indéterministe
  - x' > x
- X doit être égal à 4
  - but d'état déterministe
  - x = 4
- X doit devenir plus grand que y et z doit être égal à 5
  - but de progrès indéterministe
  - $x' > y \wedge z' = 5$

Plusieurs buts principaux ?

- Buts de maintien sont gérés par l'invariant
- 2 buts d'achèvement A et B sont en fait un seul but d'achèvement (A ^ B)

Types de buts

- NS/NNS : nécessairement satisfiable ou non
  - Un but NS est un but dont chaque tentative de résolution va réussir.
  - Si la résolution d'un but peut échouer où si l'on ne sait pas si sa résolution peut échouer ou pas alors le but est NNS.
- L/NL : paresseux ou non paresseux
  - Un but paresseux est un but dont la CS est évaluée avant chaque tentative de résolution
  - Si la CS est vraie alors le but est considéré comme étant déjà résolu, sa décomposition n'est pas exécutée
  - Remarques
    - Les buts de progrès ne peuvent pas être paresseux
    - Les buts d'achèvement sont souvent paresseux

Buts feuilles

- Ne sont pas décomposés
- Sont associés à une action atomique spécifiée par
  - un nom
  - une précondition
  - une postcondition
  - une GPF (à partir de laquelle les GPF des parents sont inférées pendant le pss de preuve)
- L'action peut être :
  - Une simple affectation
  - L'appel à une fonction (cf effecteurs par exemple)

Buts feuilles : exemple de l'action «drop»

- Sémantique informelle
  - Contexte : étude de cas « Robots sur Mars »
  - Sémantique : déposer un déchet sur une cellule
- Modélisation :
  - Nom : drop
  - CS : $\neg busy' \wedge G'(x, y)$
  - Action :
    - Nom : drop
    - Pre : $busy \wedge \neg G(x, y)$
    - Post : $\neg busy' \wedge G'(x, y)$
    - GPF : false
  - But NS, NL

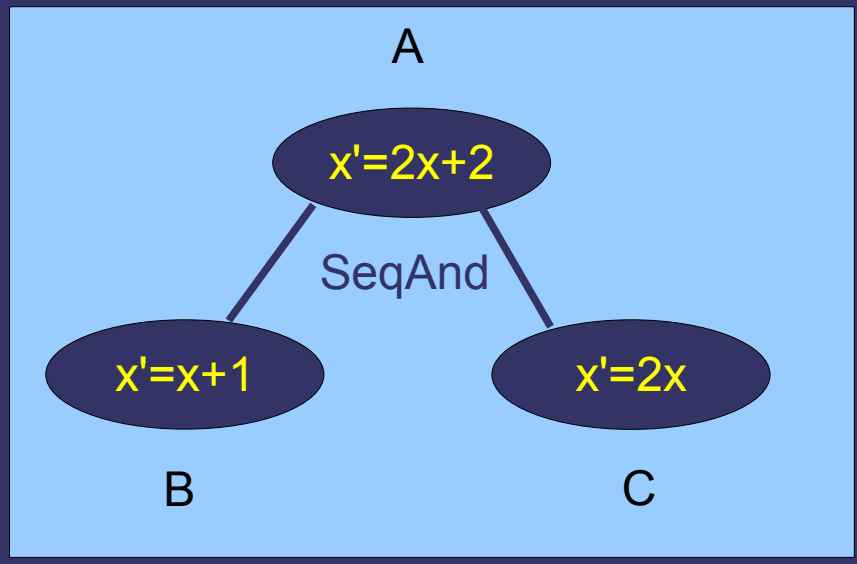
Arbre et décomposition de buts

- Un but est décomposé en sous-buts via des opérateurs de décomposition
- Opérateurs
  - SeqAnd, SeqOr (séquentiels)
  - And, Or (indéterministes)
  - Case (conditonnel)
  - Iter (itératif)
  - SyncSeqOr, SyncSeqAnd (synchronisés)
- Extensible moyennant de fournir
  - Une sémantique LTL
  - Un schéma de preuve, un pattern de composition d'automates
  - Le tout respectant la sémantique générale des opérateurs (voir plus loin)

Sémantique d'une décomposition

- Décomposition
  - But parent : A
  - Buts fils : B et C
  - Opérateur de décomposition : Op(B,C)
- Sémantique informelle : Op(B,C) $\Rightarrow SC_A$
- Que signifie « satisfaire Op(B,C) » ?
  - spécifié par la sémantique de chaque opérateur
  - Précise l'ordre d'exécution des sous-buts

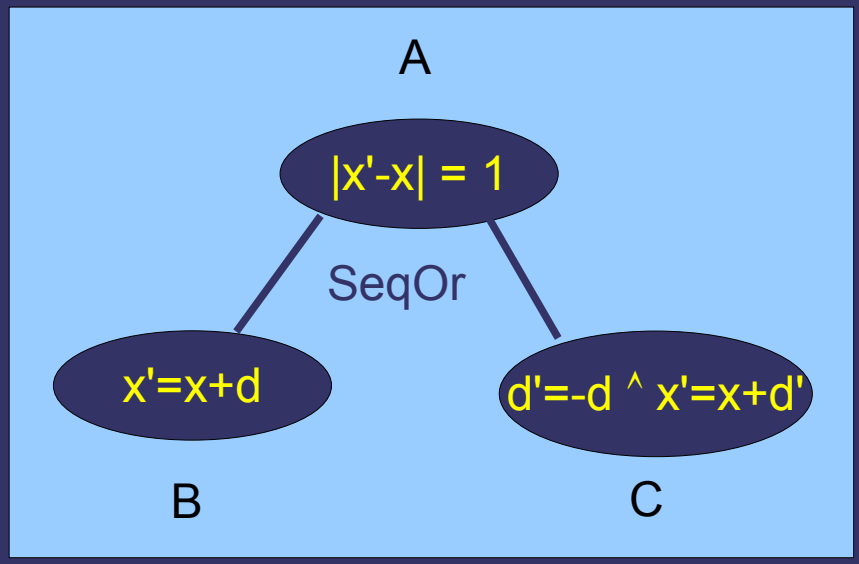
Exemple de décomposition : seqand



- Sémantique:
- Si B est résolu, essayer de résoudre C.
- Si C est résolu, A est résolu.
- Si B échoue, ne pas essayer de résoudre C, mais A peut être résolu quand-même.
- Si C échoue, A peut éventuellement être résolu.

Exemple x=2
en résolvant le but B, on exécute x:=x+4

Exemple de décomposition : seqor

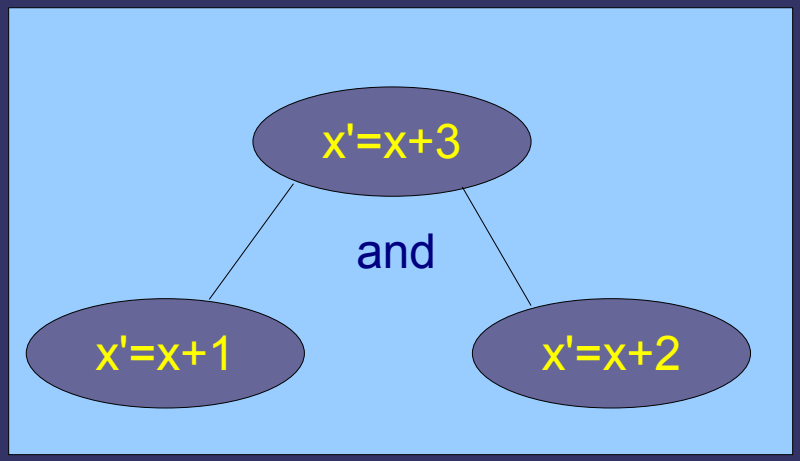


- Contexte:
- x est une variable qui doit croître de 1 à 10
- puis décroitre de 10 à 1
- d est une variable initialisée à 1
- La décomposition modélise un pas
- d'évolution de x.

Remarque : B doit être un but NNS

- Sémantique:
- Si B est résolu, alors A est résolu
- Si B échoue alors tenter de résoudre C
- Si C échoue, A est a priori non résolu
- mais peut être résolu quand-même par effet de bord.
- Si C est résolu, alors A est résolu.

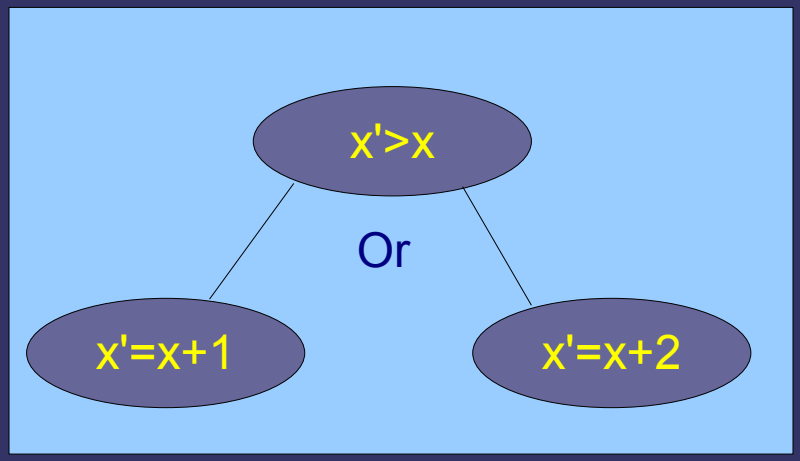
Exemple de décomposition : and



NB : opérateur indéterministe

- Sémantique:
- Choisir au hasard un des deux sous-buts : sb
- Essayer de résoudre sb
- Si sb est résolu alors le but parent est résolu
- Si sb n'est pas résolu alors tenter de résoudre l'autre sous-but
- Si le second sous-but est résolu alors le but parent l'est aussi
- Si le second sous-but échoue, le but parent peut être résolu par effet de bord

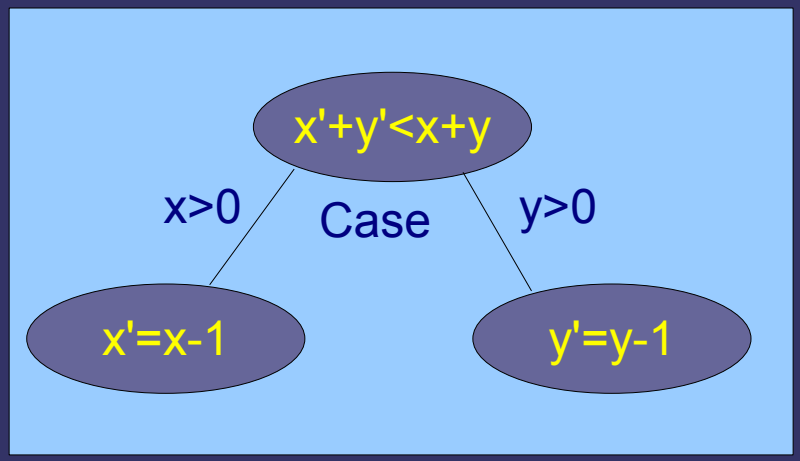
Exemple de décomposition : or



NB : opérateur indéterministe

- Sémantique:
- Choisir au hasard un des deux sous-buts : sb
- Essayer de résoudre sb
- Si sb est résolu alors le but parent est résolu
- Si sb n'est pas résolu alors le but parent peut être résolu par effet de bord

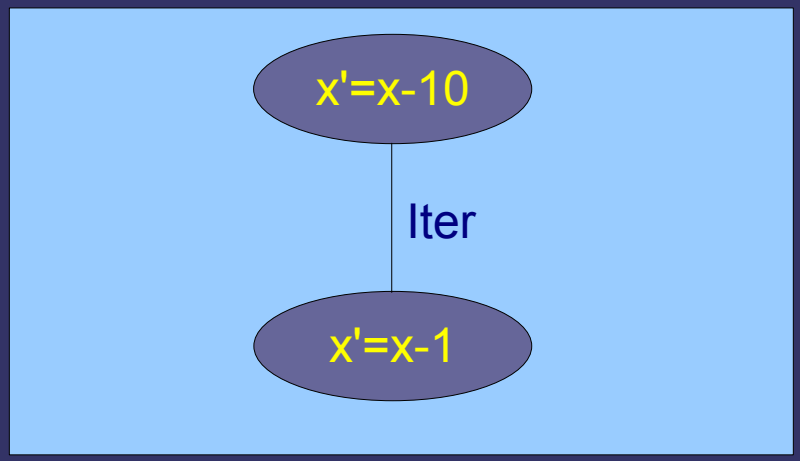
Exemple de décomposition : case



- la disjonction des conditions du case doit être vraie au moment de l'exécution du but père.
- au moins un des deux sous-buts doit être activable
- indéterminisme possible

- Sémantique:
- Evaluer les deux conditions
- Tenter de résoudre le sous-but sb dont la condition est vraie
- Si sb est résolu alors le but A est résolu
- Si sb échoue alors le but A peut éventuellement être résolu par effet de bord
- Si les deux conditions sont vraies, choisir un des sous-buts aléatoirement

Exemple de décomposition : Iter

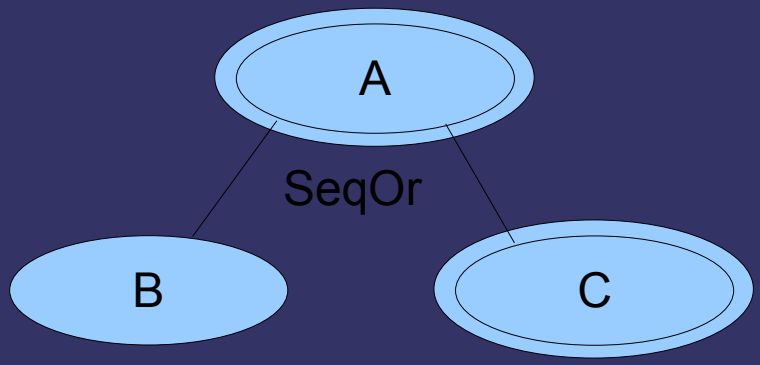


- Sémantique:
- Pour résoudre le but parent, il faut résoudre plusieurs fois le but fils
- Chaque résolution du but fils doit contribuer à la résolution du but parent
- La résolution du but fils peut échouer certaines fois
- Le processus de résolution s'arrête lorsque le but parent est résolu

Opérateurs synchronisés

- syncseqor et syncseqand
- Paramétrés par un ensemble de variables V
- Sémantique :
  - La même que seqor et seqand
  - En plus : entre la résolution des deux sous-buts, les variables de V sont verrouillées
- Application : V = variables d'environnement

L'opérateur seqOr pour ordonner des plans



- Pourquoi ne pas utiliser l'opérateur Case ?

  Car la réussite du plan dont B est la tête dépend de conditions sur l'environnement.

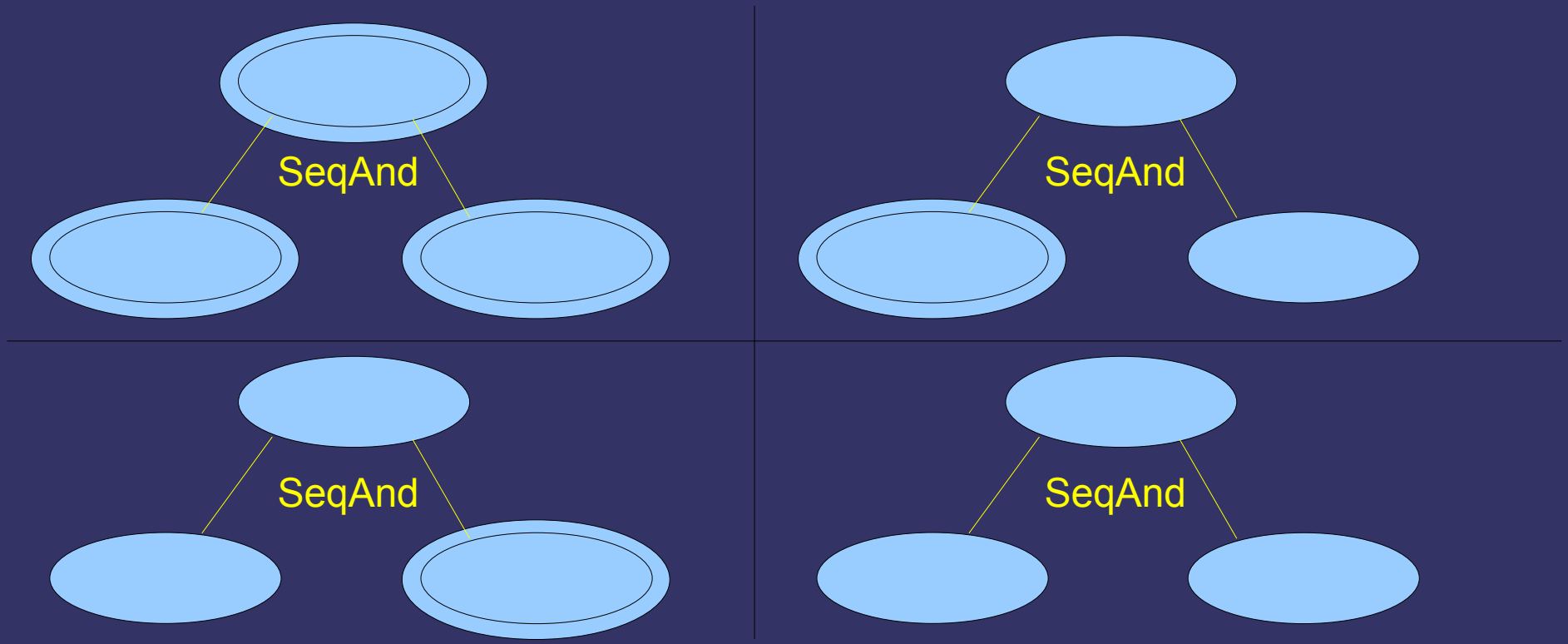
- B correspond à un plan qui permet de résoudre A avec un coût faible dans 80% des cas.
- C est un plan beaucoup plus couteux permettant de résoudre A dans tous les cas/

Quand stopper la décomposition ?

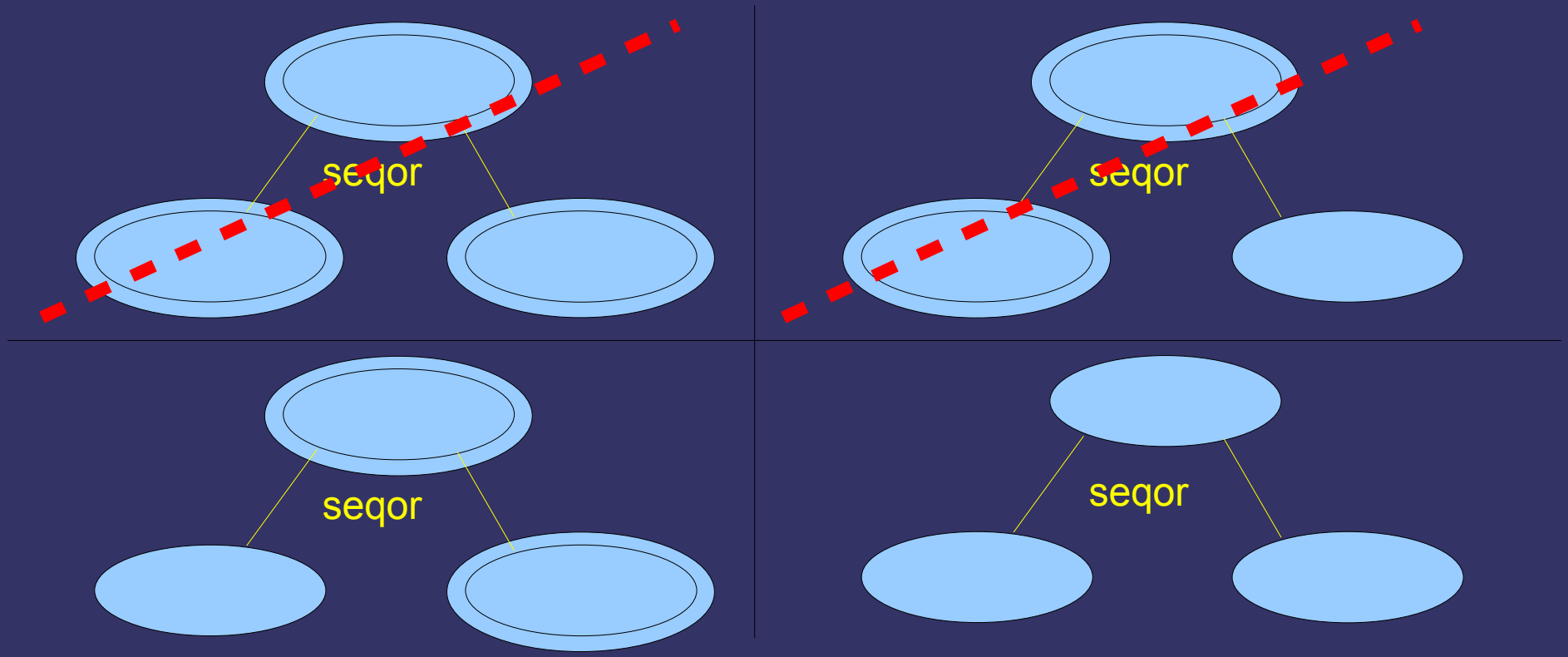
- Si l'agent est un robot, quand le but correspond à une action possible du robot
- Quand un but ne concerne que des variables d'environnement et il peut être résolu par une action possible de l'agent sur l'environnement
- Quand le but ne concerne que des variables internes et peut être résolu par de simples affectations
- Quand le but est encore complexe mais qu'une fonction d'une librairie du langage cible est capable de le résoudre.

Inférence de la « satisfiabilité »

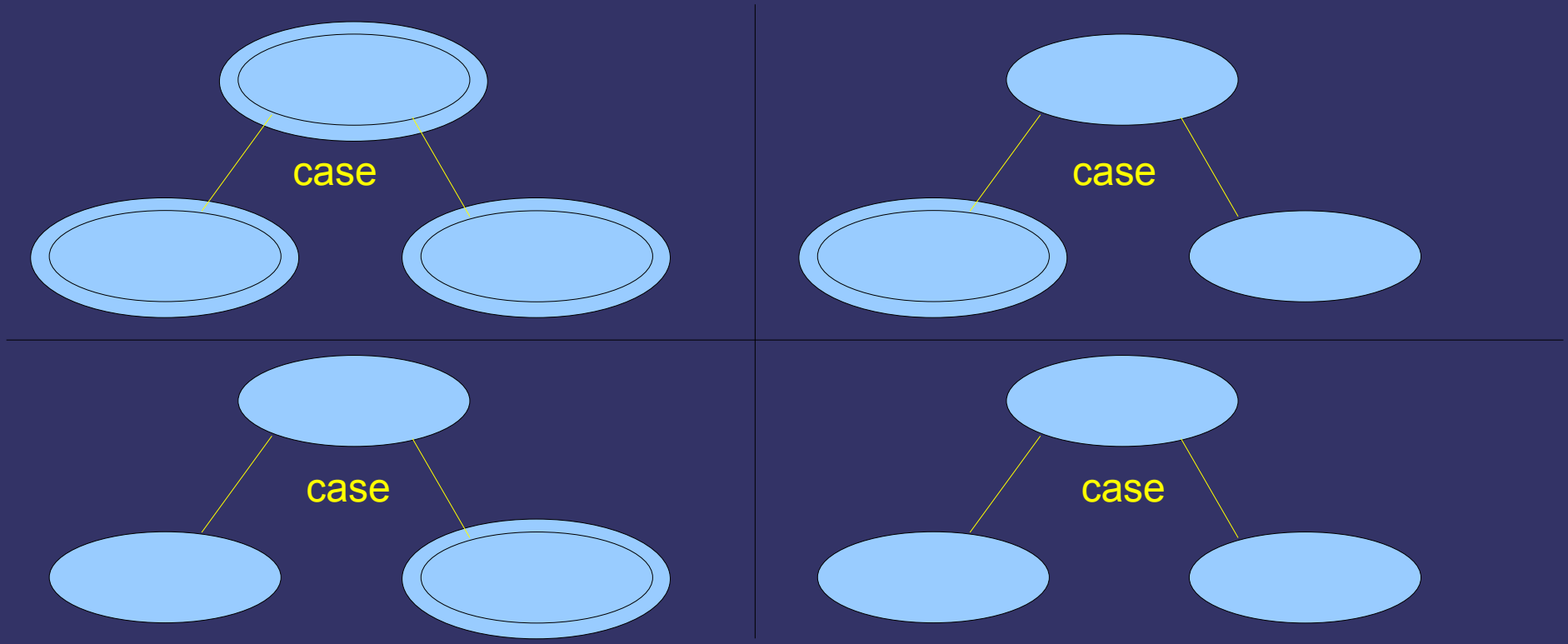
- Mécanisme s'appliquant des feuilles vers la racine
- Exemple du SeqAnd



Satisfiabilité : cas de seqor

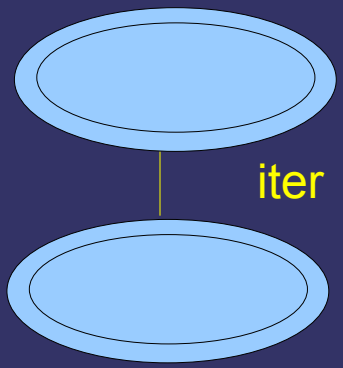


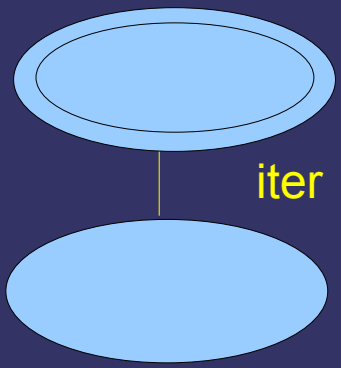
Satisfiabilité : cas de case



Satisfiabilité : cas de iter

- Le but parent d'une décomposition de type « iter » doit être NS (pour des raisons de terminaison)





Plan du cours

- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

Sémantique en LTL : introduction

- LTL = logique temporelle linéaire
- Exécution du GDT vue comme la production d'une trace composée d'états.
- Règles
  - Globales à tous les opérateurs
  - Propres à chaque opérateur de décomposition
- Opérateurs de base en LTL
  - Next ($\bigcirc$)
  - Toujours ($\Box$)
  - Un jour ($\Diamond$)
  - Opérateurs booléens classiques
  - NB : A$\rightarrow$B : dans un même état de la trace

Sémantique LTL : variables utilisées

- $Init_A$: vrai quand l'état courant correspond au début de la résolution de A
- $Init_A^{NL}$: vrai quand l'état courant correspond au début de l'exécution de la décomposition de A
- End_A : vrai quand l'état courant correspond à la fin de la résolution de A
- In_A : vrai quand l'état courant correspond à une étape de résolution du but A
- Sat_A : vrai quand dans l'état courant la SC du but A est satisfaite

Sémantique LTL : règles générales

- $\Box(\neg in_B \wedge \neg in_C)$ Les sous-buts sont mutuellement exclusifs
- $\Box((in_B \rightarrow in_A) \wedge (in_C \rightarrow in_A))$ Caractérisation de sous-buts
- $\Box(init_A \vee end_A)$ Résoudre un but prend au moins un pas d'exécution
- $\Box(in_A \wedge \neg end_A \rightarrow \bigcirc \neg init_A)$ Des tentatives de résolution d'un même but ne peuvent pas être imbriquées
- Noeud paresseux :
- $\Box(init_A \rightarrow \bigcirc(sat_A \rightarrow end_A))$ un noeud paresseux satisfait est aussitôt fini
- $\Box(init_A \rightarrow \bigcirc(sat_A \rightarrow init_A^{NL}))$ un noeud paresseux non satisfait va être traité comme un noeud non paresseux
- Noeud non paresseux :
- $\Box(init_A \rightarrow init_A^{NL})$

Sémantique LTL de SeqAnd

- Cas de A <= B SeqAnd C avec A NL
  - $\Box(sat_B \rightarrow \bigcirc init_C)$
  - $\Box(end_B \wedge \neg sat_B \rightarrow \bigcirc end_A)$
  - $\Box(sat_C \rightarrow (end_A \wedge sat_A))$
  - $\Box(end_C \wedge \neg sat_C \rightarrow \bigcirc end_A)$
  - $\Box(init_A{}^{NL} \rightarrow init_B)$

Sémantique LTL de And

- Cas de A <= B And C avec A NL

- $\Box(init_A^{NL} \rightarrow second_A)$
- $\Box(second_A \wedge in_A \rightarrow \bigcirc(\neg init_A \rightarrow second_A))$
- $\Box(sat_B \wedge \neg second_A \rightarrow \bigcirc(second_A \wedge init_C))$
- $\Box(sat_C \wedge \neg second_A \rightarrow \bigcirc(second_A \wedge init_B))$
- $\Box(end_B \wedge \neg sat_B \rightarrow \bigcirc end_A)$
- $\Box(end_C \wedge \neg sat_C \rightarrow \bigcirc end_A)$
- $\Box(sat_C \wedge second_A \rightarrow (end_A \wedge sat_A))$
- $\Box(sat_B \wedge second_A \rightarrow (end_A \wedge sat_A))$
- $\Box(init_A^{NL} \rightarrow init_B \vee init_C)$

Sémantique LTL de Or

- Cas de A <= B OR C avec A NL

  - $\Box(init_A^{NL} \rightarrow \neg second_A)$
  - $\Box(second_A \wedge in_A \rightarrow \bigcirc(init_A \rightarrow second_A))$
  - $\Box(sat_B \rightarrow (end_A \wedge sat_A))$
  - $\Box(sat_C \rightarrow (end_A \wedge sat_A))$
  - $\Box(end_B \wedge \neg sat_B \wedge \neg second_A \rightarrow \bigcirc(second_A \wedge init_C))$
  - $\Box(end_C \wedge \neg sat_C \wedge \neg second_A \rightarrow \bigcirc(second_A \wedge init_B))$
  - $\Box(end_B \wedge \neg sat_B \wedge second_A \rightarrow \bigcirc(end_A))$
  - $\Box(end_C \wedge \neg sat_C \wedge second_A \rightarrow \bigcirc(end_A))$
  - $\Box(init_A^{NL} \rightarrow init_B \vee init_C)$

Sémantique LTL du SeqOr

- Cas de A <= B SeqOr C avec A NL
  - $\Box(sat_B \rightarrow end_A \wedge sat_A)$
  - $\Box(end_B \wedge \neg sat_B \rightarrow \bigcirc init_C)$
  - $\Box(sat_C \rightarrow (end_A \wedge sat_A))$
  - $\Box(end_C \wedge \neg sat_C \rightarrow \bigcirc end_A)$
  - $\Box(init_A^{NL} \rightarrow init_B)$

Sémantique LTL de Case

- Cas de A <= case($cond_B$,B,$cond_C$,C) avec A NL
  - $\Box(sat_B \rightarrow end_A \wedge sat_A)$
  - $\Box(sat_C \rightarrow (end_A \wedge sat_A))$
  - $\Box(init_A^{NL} \rightarrow \bigcirc(cond_B \vee cond_C))$
  - $\Box(init_A^{NL} \rightarrow \bigcirc(\neg\, cond_B \rightarrow init_C))$
  - $\Box(init_A^{NL} \rightarrow \bigcirc(\neg\, cond_C \rightarrow init_B))$

Sémantique LTL de Iter

- Cas de A <= iter B avec A NL
  - $\Box(init_A^{NL} \rightarrow init_B)$
  - $\Box(end_B \rightarrow \bigcirc(\neg sc_A \rightarrow init_B))$
  - $\Box(end_B \rightarrow \bigcirc(sc_A \rightarrow (end_A \wedge sat_A))$

Principe de vérification

- La préservation de l'invariant est vérifiée au niveau des feuilles
- Chaque décomposition est vérifiée grâce au calcul d'un contexte qui est propagé de bas en haut et de gauche à droite
- Un schéma de preuve est associé à chaque opérateur
- Chaque schéma de preuve permet de générer des obligations de preuve

Plan du cours

- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

Génération de code : principe

- Un automate avec des transitions étiquetées est associé à chaque type de but feuille
- Un pattern de composition d'automates est associé à chaque opérateur de décomposition
- Des patterns de post-traitement sont spécifiés pour les buts paresseux et NNS

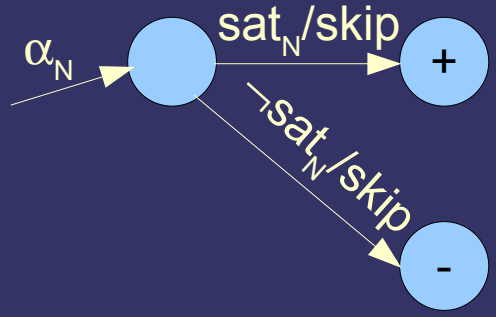
Automate de comportement : spécification

- Alphabet d'entrée :
  - sat_N, $\neg sat_N$ pour tout noeud N du GDT
  - $cond1_i$, $cond2_i$ pour tout opérateur $case_i$
  - true, $choice_1$, $\neg choice_2$
- Alphabet de sortie :
  - α_F pour tout but feuille F (action associée)
  - skip (ne rien faire)
  - sv_N pour tout noeud N
  - $lock_V$, $unlock_V$ pour toute variable V de l'environnement
- Deux états finaux : + (succès) et - (échec)

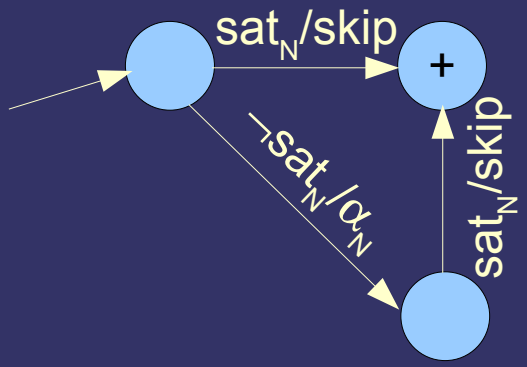
Automates pour les buts élémentaires

But NS et NL

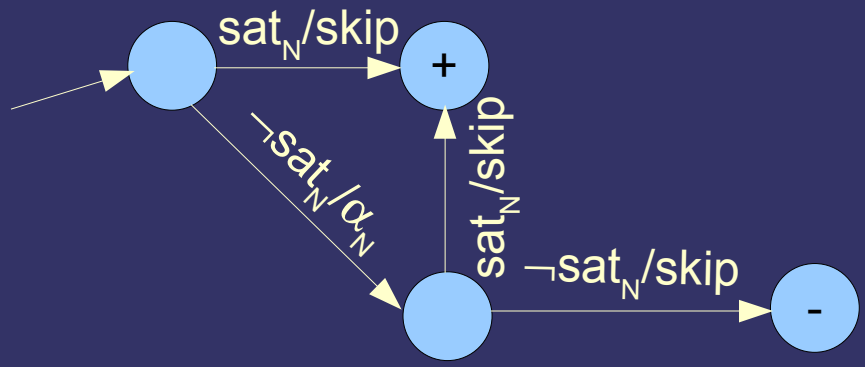
But NNS et NL



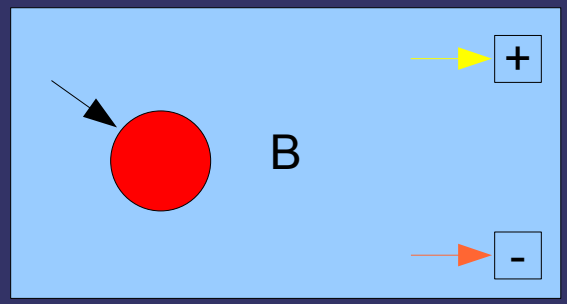
But NS et L

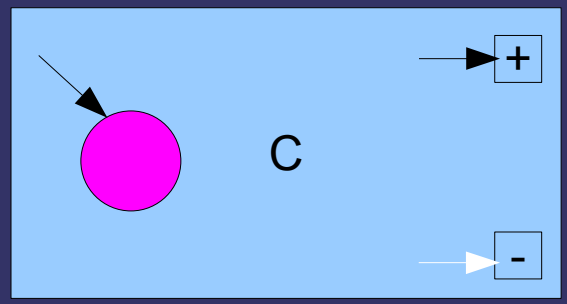


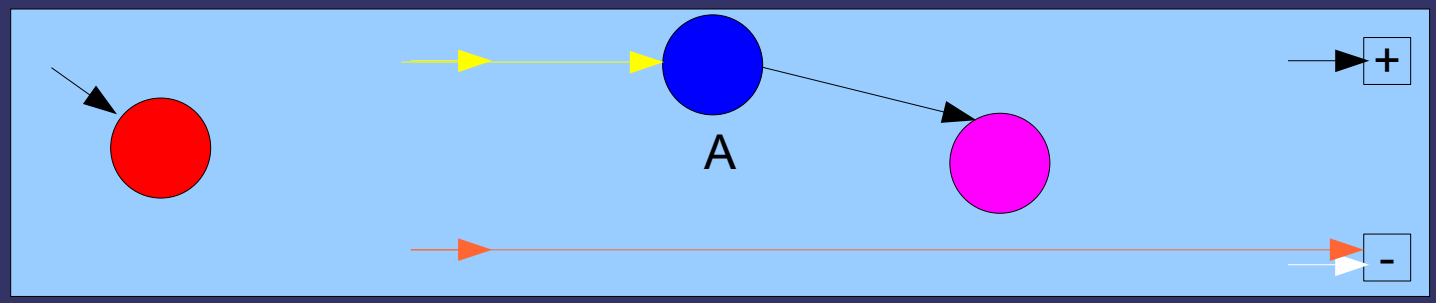
But NS et L



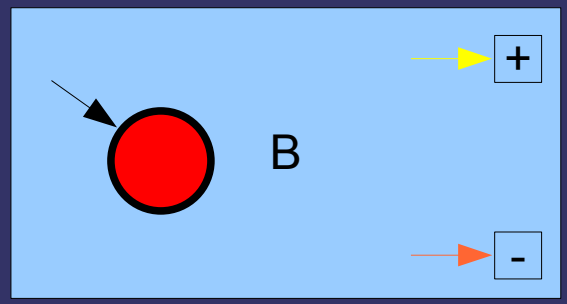
Pattern de composition pour SeqAnd

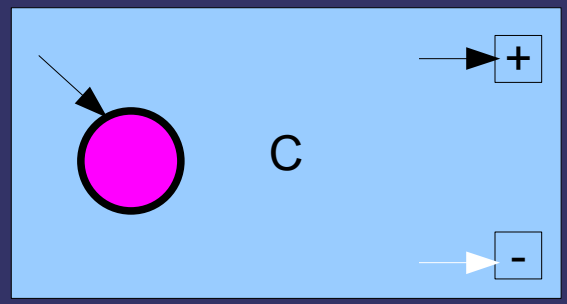


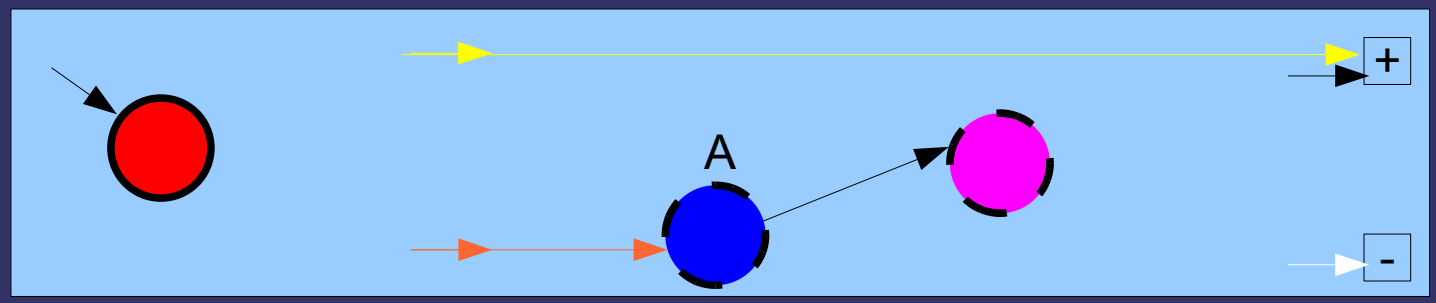




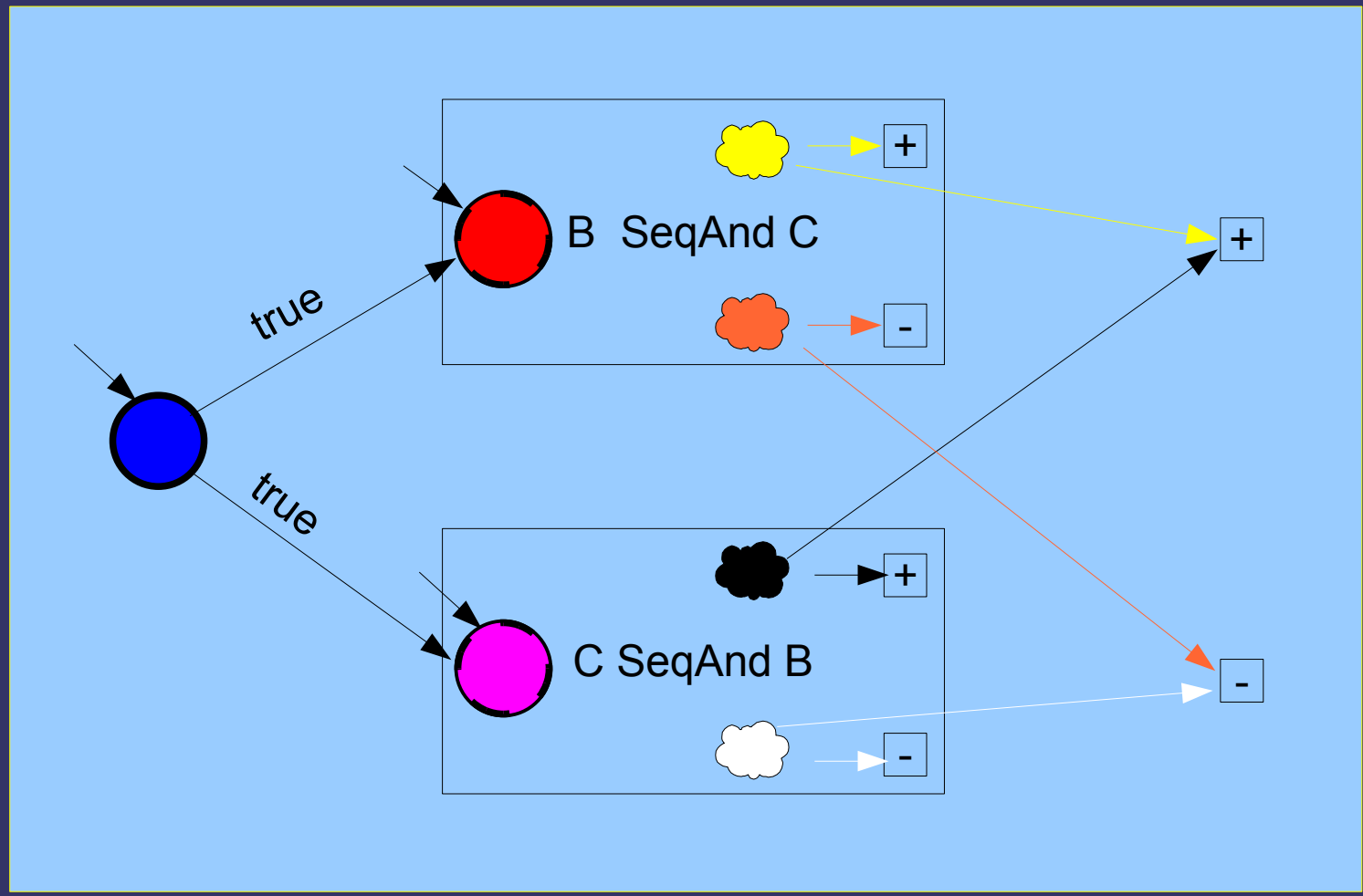
Animation SeqOr



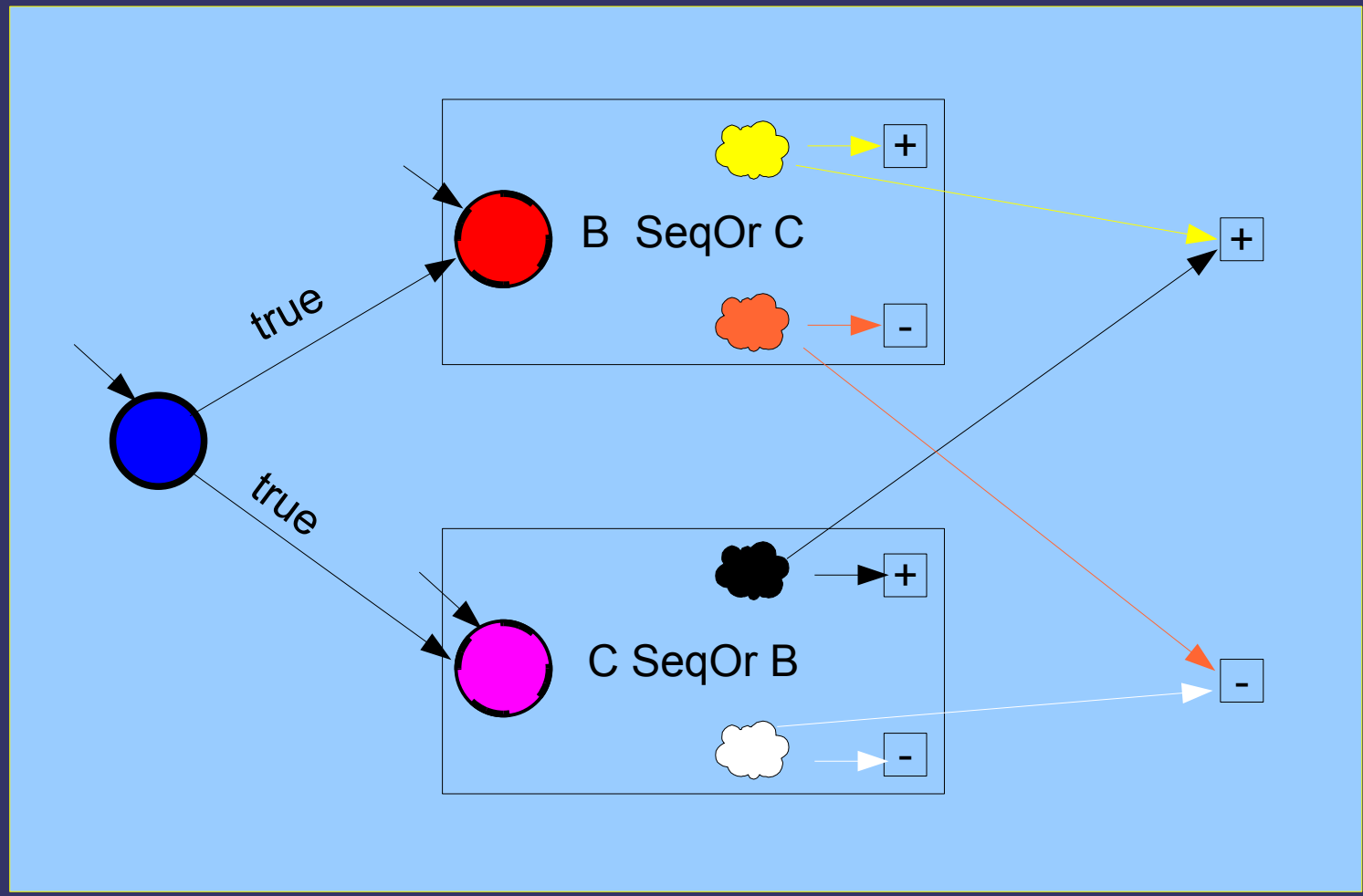




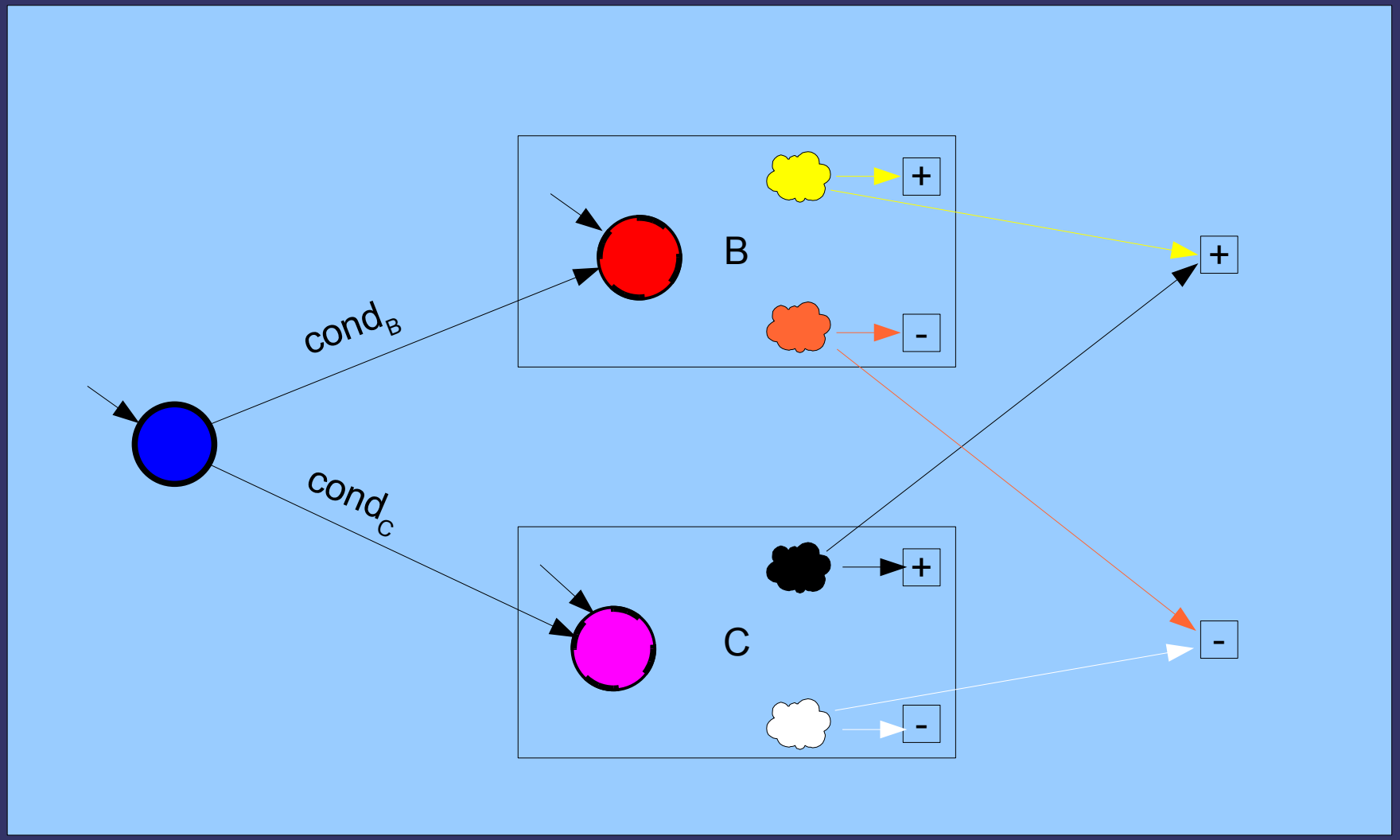
Animation And



Animation Or



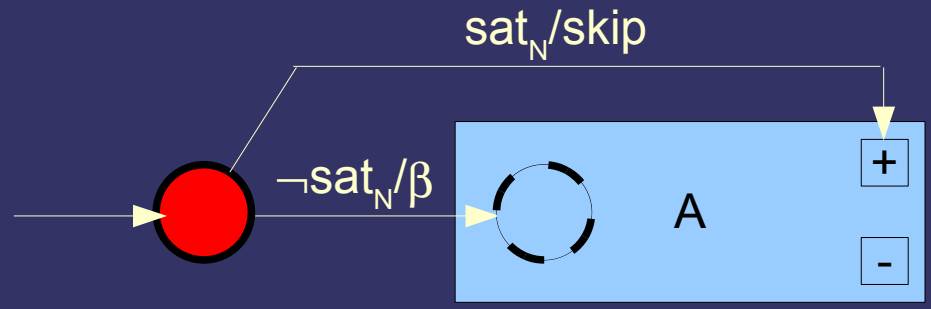
Animation Case



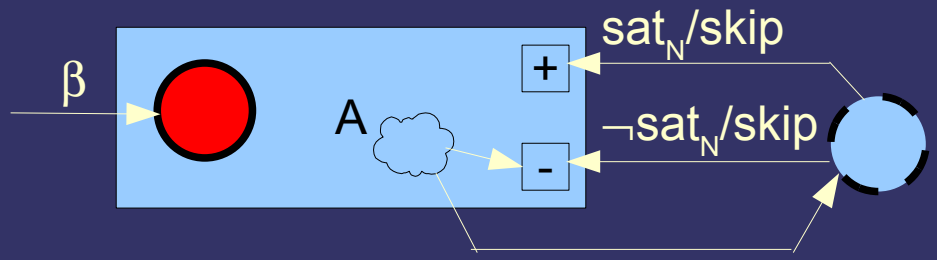
Paresse/ NNSatisfaisabilité : Pré et post-patterns

Soit N un but intermédiaire décomposé via un opérateur Op. Soit A l'automate généré en appliquant le pattern de composition d'automates associé à Op aux sous-buts de N.

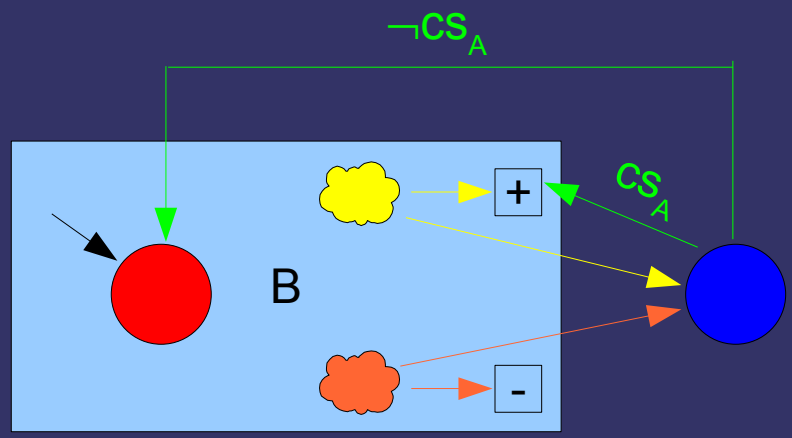
Paresse : Pré-pattern



NNSatisfaisabilité : Post-pattern



Animation Iter



Correction de l'approche

- Les schémas de preuve ont été validés grâce à la sémantique LTL
- Les automates de composition ont également été validés par cette sémantique
- L'inférence des contextes, des GPF et des types de buts a été également vérifiée
- Publication en cours de soumission à JAAMAS

Plan du cours

- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

Outils

- Un éditeur de GDT permettant de générer
  - La représentation conceptuelle et graphique du GDT
  - Une synthèse HTML et XML du GDT
  - Le code associé (en Java, via les automates)
- Pour l'instant pas de connexion directe avec un prouveur ☹

Perspectives sur les outils

- Un outil générique permettant de :
  - Spécifier n'importe quel GDT
  - Générer automatiquement le code via les automates
  - Générer automatiquement les obligations de preuve
- Connexion avec un prouveur
  - Krt est actuellement étudié
  - PVS pourrait aussi être envisagé
- Une connexion avec SPIN est actuellement étudiée
- Animation de GDT : plusieurs pistes
  - Utilisation de AgentSpeak et Jason
  - Utilisation de Promela et SPIN
  - Utilisation de JADEX

Animation de la spécification (prototypage)

- Possible grâce à une approche top-down
- Principe de l'animateur (à développer) :
  - Si un but est décomposé, l'exécuter à partir de la sémantique de la décomposition.
  - Si un but n'est pas décomposé :
    - Si c'est un but feuille avec une action associée, exécuter l'action.
    - Si le but n'est pas un but feuille :
      - Si la SC (ou la GPF si le but doit échouer) est déterministe et que des valeurs la satisfaisant peuvent être inférées, les inférer
      - Sinon laisser l'utilisateur proposer ses propres valeurs pour les variables modifiées par la résolution du but, vérifier la satisfaction de la SC (ou de la GPF) et continuer l'animation

Perspectives vs Sujet de stage de Master 2

- Génération et analyse de traces
  - Lien avec les travaux sur LEADSTO
  - Vérification de propriétés inter-traces

Plan du cours

- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

Etude de cas : RoM (Robots on Mars)

- Spécification initiale

  [1] R.H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. *Verifiable multi-agent programs*. In M. Dastani, J. Dix, and A. Seghrouchni, editors,ProMAS, 2003.

- Description informelle : 2 robots doivent collaborer pour nettoyer Mars (représentée par une grille)
  - Robot R2 : position fixe, peut brûler les déchets qui se trouvent sur sa cellule
  - Robot R1
    - explore Mars cellule par cellule
    - Quand il trouve un déchet, il le ramasse (au plus 3 fois), l'apporte à R2 puis reprend son exploration

AgentSpeak : quelques éléments

- Langage de programmation logique
- Permet de spécifier et implanter des agents BDI
- 2 notions fondamentales
  - Un ensemble de croyances : prédicats
  - Un ensemble de plans :
    - Évenement déclencheur : ajout/suppression de buts (prédicats) ou de croyances
    - Contexte : conjonction de croyances
    - Corps : séquence d'actions basiques (prédicats) et de buts (sous-buts)

Agentspeak : résumé de la syntaxe

- Buts :
  - But d'achèvement : *! prédicat*
  - But test : *? prédicat*
- Evenements (*+/- prédicat*) : ajout ou suppression de but ou de croyance
- Plan : $e : b_1 \wedge b_2 \wedge \ldots \wedge b_n \leftarrow h_1 \wedge h_2 \wedge \ldots \wedge h_n$
  - e : événement
  - b_i : croyance
  - h_i : action basique ou but

Code AgentSpeak pour le robot R1

- Croyances

  pos(r2,2,2).
  checking(slots).

- Plans

```
+pos(r1,X1,Y1) : checking(slots) & not(garbage(r1)) <- next(slot).
+garbage(r1) : checking(slots) <- !stop(check);!take(garb,r2);!continue(check).
+!stop(check) : true <- ?pos(r1,X1,Y1);+pos(back,X1,Y1);-checking(slots).
+!take(S,L) : true <- !ensure_pick(S);!go(L);drop(S).
+!ensure_pick(S) : garbage(r1)<- pick(garb);!ensure_pick(S).
+!ensure_pick(S) : true <- true.
+!continue(check) : true<- !go(back);-pos(back,X1,Y1);
      +checking(slots);next(slot).
+!go(L) : pos(L,Xl,Yl) & pos(r1,Xl,Yl) <- true.
+!go(L) : true <- ?pos(L,Xl,Yl); moveTowards(Xl,Yl);!go(L).
```

Code AgentSpeak pour le robot R2

- Croyances

  néant

- Plans

  +garbage(r2) : true <- burn(garb).

Modélisation de Rom avec les GDT
Les variables

- Environnement
  - Variables : G(x,y) : int × int → booléen
  - Constantes : xMin, yMin, xMax, Ymax, xR2, yR2
- Variables de l'agent R1
  - x, y, xSaved, Ysaved
  - NbAttempts, busy, clean
- Variables de l'agent R2
  - busyR2

Modélisation de RoM avec les GDT
Spécification des agents

- Invariant de l'environnement : true
- Agent R1 :
  - Invariant :
  - Clause d'initialisation : $clean=\emptyset \wedge busy=false \wedge pos=pos_{Min} \wedge pos_{Saved}=pos_{Min} \wedge nb_{Attempts} = 0$
  - Condition de déclenchement : true
  - Précondition : $clean=\emptyset \wedge \neg busy \wedge pos=pos_{Min}$
- Agent R2 :
  - Invariant : true
  - Clause d'initialisation : $busy_{R2}$ = false
  - Condition de déclenchement : $G(x_{R2},y_{R2})$
  - Précondition : $\neg busy_{R2}$

GDT R1 (1)

GDT R1 (2)

GDT R1 (3)

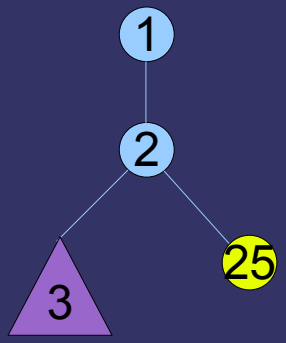
GDT R1 (4)

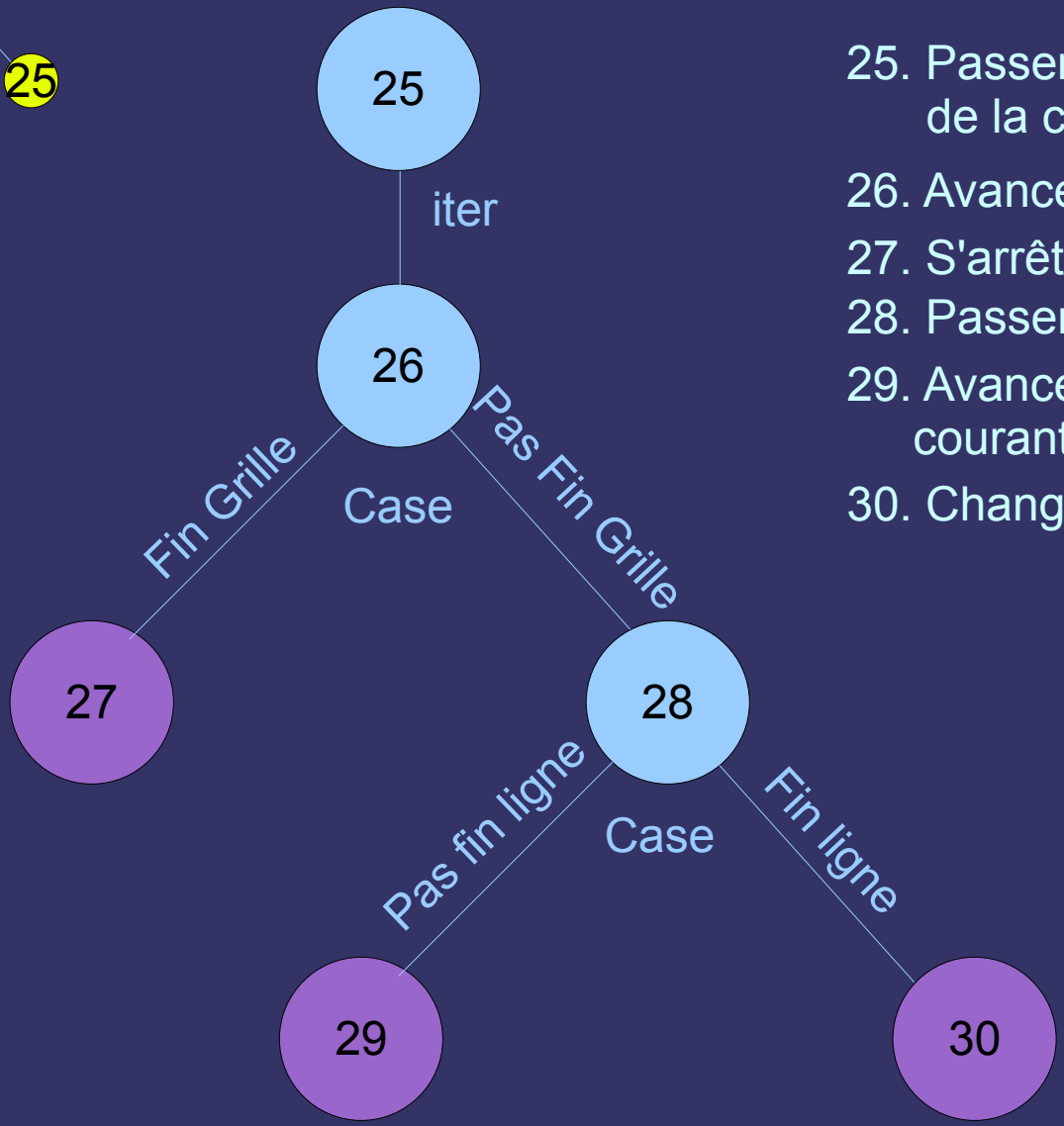
GDT R1 (5)

GDT R1 (6)

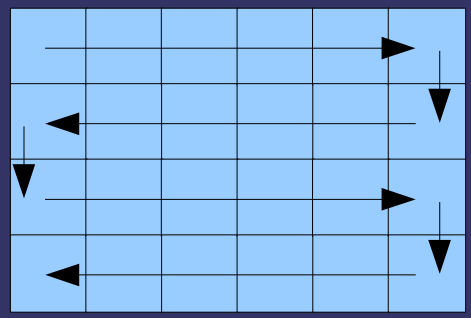
GDT R1 (7)

GDT R1 (8)

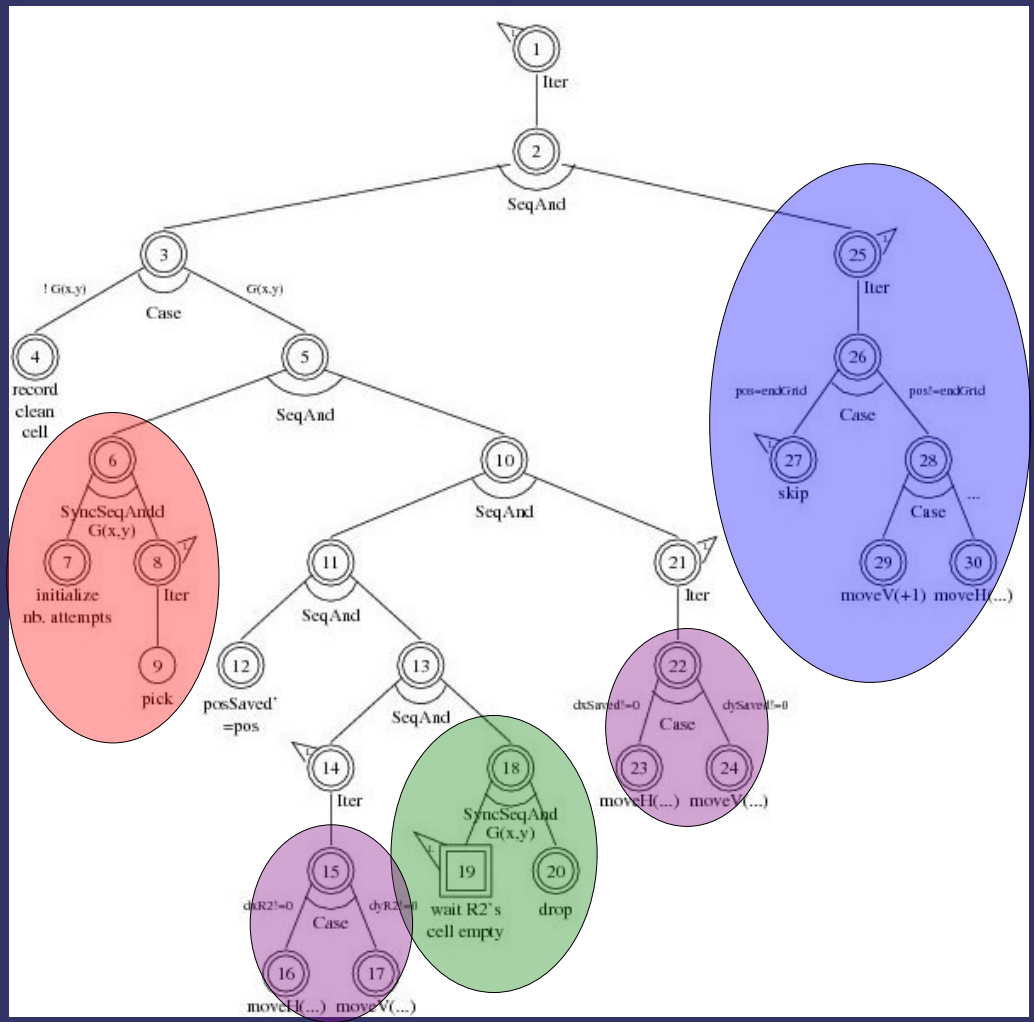




25. Passer à la case suivante différente de la cellule de R2
26. Avancer d'une case ou s'arrêter
27. S'arrêter
28. Passer à la case suivante
29. Avancer d'une case dans le sens courant
30. Changer de ligne et de sens



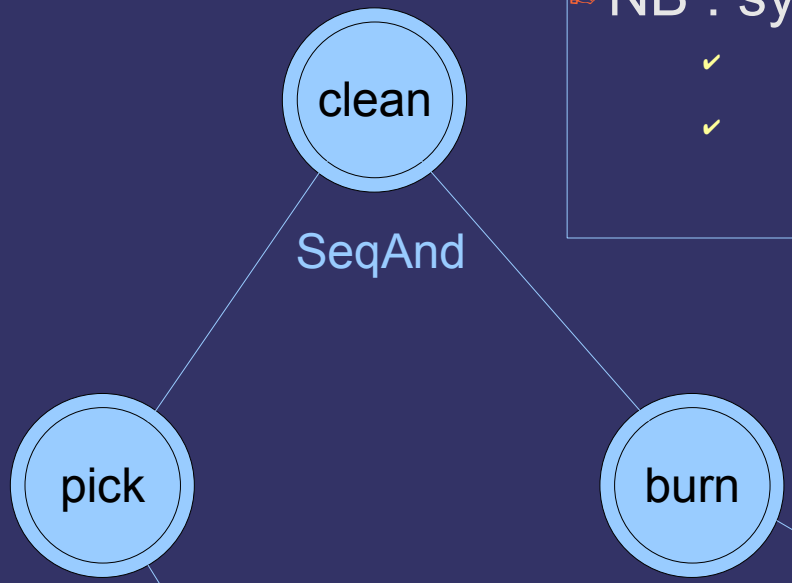
Le GDT proposé pour R1 : vue globale



- NextSlot()
- Pick()
- Drop()
- moveTowards()

Le GDT proposé pour R2

- NB : synchronisation entre R1 et R2 assurée par
  - Le but externe dans le GDT de R1
  - La condition de déclenchement du GDT de R2



CS_{clean} : $\neg busy_{R2} \wedge G(x_{R2},y_{R2})=clean$

CS_{pick} : $busy_{R2} \wedge G(x_{R2},y_{R2})=clean$

CS_{burn} : $\neg busy_{R2} \wedge G(x_{R2},y_{R2})=clean$

α_{pick} :

précondition : $\neg busy_{R2} \wedge G(x_{R2},y_{R2})=dirty$

postcondition : $busy_{R2} \wedge G(x_{R2},y_{R2})=clean$

α_{burn} :

précondition : $busy_{R2}$

postcondition : $\neg busy_{R2}$

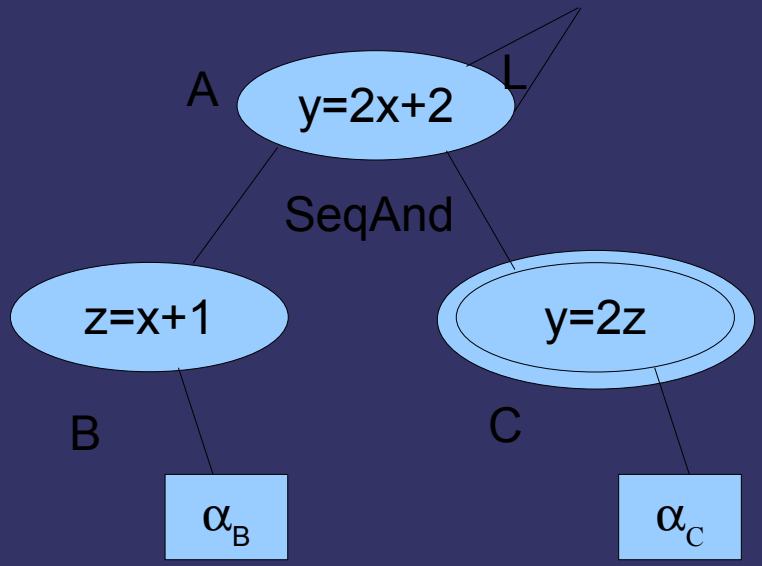
Comparaison des principes de vérification

- Techniques utilisées
  - Model Checking/Theorem proving
- Moyen
  - Outils (SPIN et JPF) / Manuel pour l'instant
- Quand ?
  - A la fin du développement / Progressivement
- But de la vérification
  - Le code / La spécification
- Quoi ?
  - 2 déchets sur une grille 5x5 / configuration quelconque

Agents GDT vs Agents BDI ?

- Croyances : variables du GDT
- Désirs : buts à résoudre après le but courant (selon un parcours infixe de l'arbre)
- Plans : décomposition de chaque but
- Intentions : plans associés aux buts ancêtres du but courant

Des GDT à AgentSpeak



+!A:true ← +!eval(a),+!doA.

+!doA: sat(a) ← true.
+!doA:¬sat(a) ← +!seqAndA1.

A est paresseux

+!seqAndA1: true ← +!doB,+!eval(b), +!seqAndA2.

+!doB: true ← ?val(x,V),-val(x,V),+val(x,V+1).

α_B

+!seqAndA2: ¬ sat(b)← +!eval(a),+!endA.

+!seqAndA2: sat(b)← +!doC,+sat(c),+sat(a).

α_C

C est NS

+!doC: true← ?val(x,V),-val(x,V),+val(x,2*V).

+!endA: sat(a) ← true.

+!endA: ¬ sat(a) ← failure.

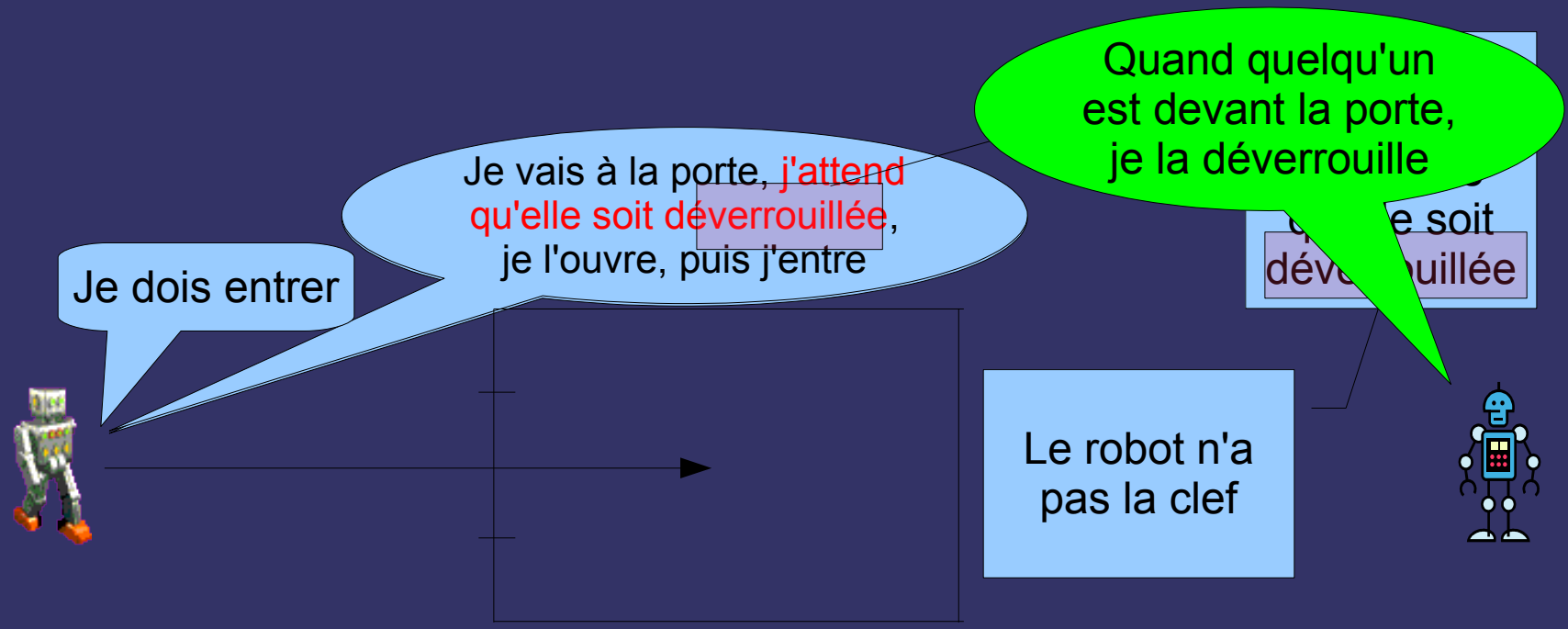
A est NNS

Plan du cours

- Contexte général de l'approche GDT
- Représentation des agents et de l'environnement
- Modélisation du comportement d'un agent
- Sémantique du modèle
- Génération de code
- Outils
- Étude de cas : robots sur Mars
- Extensions du modèle

But Externe

- Définition : un but feuille d'un GDT qui
  - Doit être résolu pour que l'agent puisse continuer
  - Ne peut pas être résolu par l'agent lui-même
  - Peut être résolu par un autre agent



But externe : spécification et remarques

- Spécification d'un but externe BE:
  - Nom + CS de BE
  - Type : NS + L
  - Un agent R devant résoudre le but BE
  - Le but BR de R permettant de satisfaire la CS de BE
- Améliorations :
  - Ne plus spécifier ni R ni BR
  - Nécessite d'associer aux GDT des propriétés LeadsTo

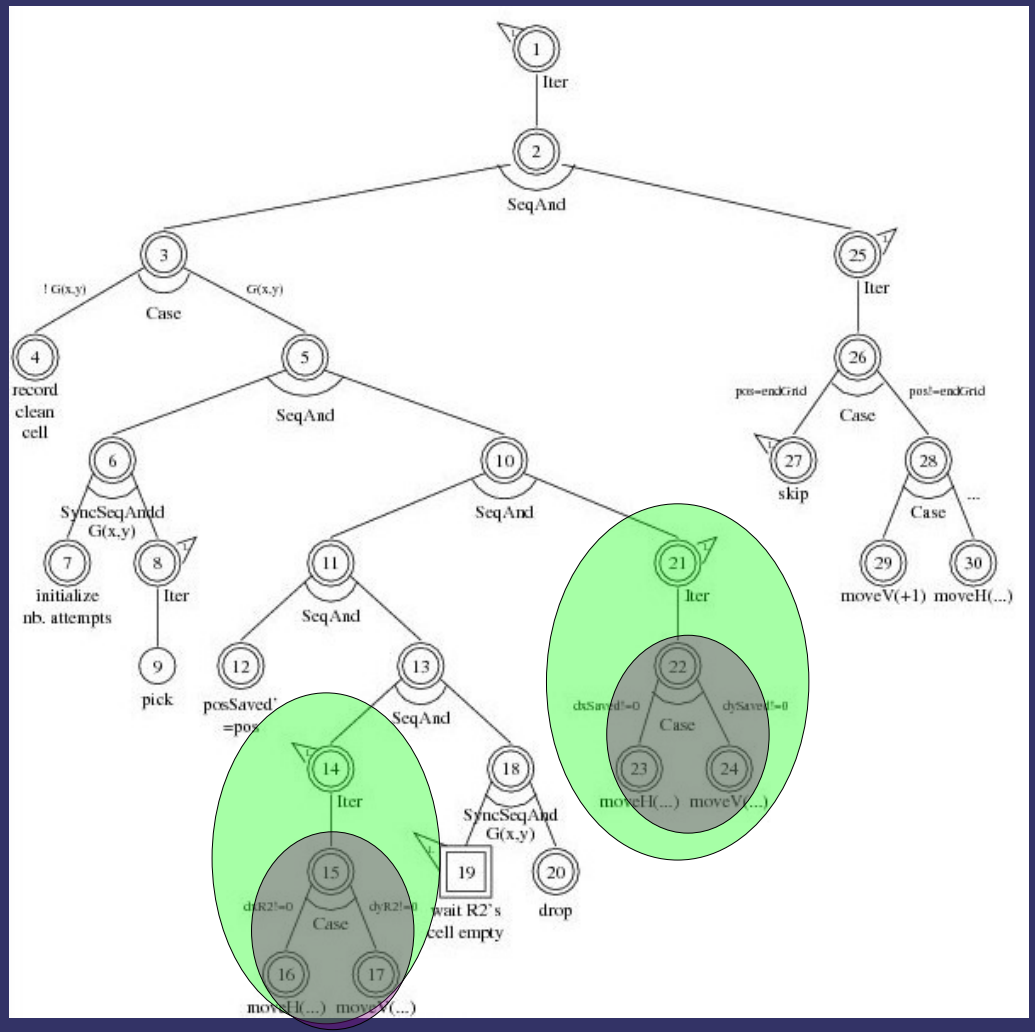
Exemple : pour le robot qui peut ouvrir la porte :

$\Box$ (porte verrouillée $\wedge$ quelqu'un devant la porte $\rightarrow \Diamond$ porte déverrouillée)

GDT paramétrés

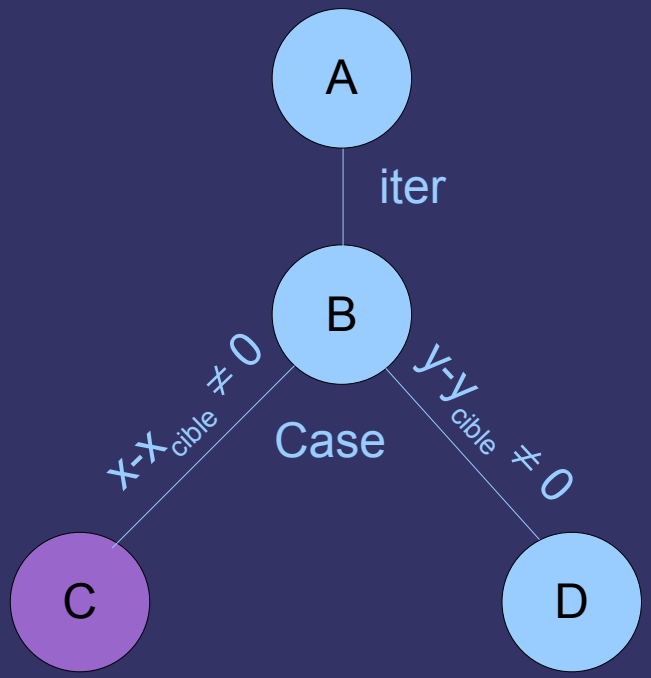
- Principe
  - Non associé à un agent
  - GDT réutilisable par instanciation des paramètres
  - Preuves de ces GDT réutilisables
- Schémas de preuve définis pour
  - Valider un GDT paramétré
  - Valider une utilisation dans un contexte précis
- Définition possible de propriétés stables
  - Propriétés maintenues par l'exécution du GDT
  - Propriétés pas forcément vraies au début
  - Augmente les possibilités de ré-utilisation
- Moyen d'échange de compétences entre agents
  - Librairie de plans

GDT paramétrés pour le cas RoM



- Sous-GDT : se déplacer d'une case
- Plus généralement: Aller vers une destination

Exemple de GDT paramétré « Aller de (x,y) à (x_{cible}, y_{cible}) »



Paramètres

$x, y, x_{cible}, y_{cible}$

Invariant :

$(x,y,x_{cible},y_{cible}) \in \mathbb{N}^4$

CS_A :

$x'=x_{cible} \wedge y'=y_{cible}$

CS_B :

$|x'-x_{cible}|+|y'-y_{cible}| < |x-x_{cible}|+|y-y_{cible}|$

CS_C :

$x'=x-(x-x_{cible})/|x-x_{cible}| \wedge y'=y$

CS_D :

$y'=y-(y-y_{cible})/|y-y_{cible}| \wedge x'=x$

Et le niveau SMA ?

- Travaux en cours
  - Définition d'outils de spécification au niveau système : a été appliqué à une extension du cas RoM avec plusieurs robots R2
  - Coordination entre agents via les buts externes
- A faire : modéliser les communications

Conclusion

- Caractéristiques intéressantes des GDT
  - Aspects méthodologiques
  - Validation de l'approche
  - Processus de vérification et de génération de code
  - Un modèle ouvert
  - Peut aider à gérer les différents aspects d'un agent
- Limites actuelles
  - Outillage limité
  - Communication non prise en compte

Perspectives

- Développement d'outils
  - Principalement :
    - Génération d'obligations de preuve
    - Connexion avec un prouveur
    - Génération effective de code au niveau SMA
  - Mais aussi
    - Animation de GDT
    - Connexion avec un model-checker
    - Explication d'échecs de preuve
- Modélisation de la communication
- Modélisation d'organisations d'agents
- Preuves de propriétés au niveau du système
- Aspects génie logiciel dans le développement des SMA
- Utiliser des logiques multi-valuées