

Règles avec négation: l'algorithme ASPERIX

Cours confiné GraphIK 2

21/04/2020

Dans les épisodes précédents:
application de règle

$$p(X), \text{ not } q(X, Y), p(Y), \text{ not } q(Y, X) \rightarrow r(X)$$

F { $p(a)$
 $q(a, b)$



$r(a) ?$

G { $p(a)$
 $q(b, a)$



$r(a) ?$

Dans les épisodes précédents:
dérivations persistantes et complètes

$(R_1) \quad p(X), \text{ not } q(X) \rightarrow r(X)$

$(R_2) \quad p(X) \rightarrow q(X)$

$p(a)$

application R_1 ,
 $q(a)$ absent
 $p(a) \longrightarrow p(a), r(a)$
application R_2 ,
 $q(a)$ apparaît
 $p(a), r(a) \longrightarrow p(a), r(a), q(a)$

On voudrait des applications
persistantes dans la dérivation: toute
application devrait rester applicable.

application R_1 ,
 $q(a)$ absent
 $p(a) \longrightarrow p(a), r(a)$
 R_2 est applicable,
mais on ferme les yeux

On voudrait des dérivations
complètes.

application R_2
 $p(a) \longrightarrow p(a), r(a)$
 R_1 n'est plus applicable,
on a bien fini

Bonne dérivation : persistante et
complète.

Dans les épisodes précédents:
existence et unicité

$p(X), \text{ not } q(X) \rightarrow q(X)$

$p(a)$

Pas de modèle stable

Implémentation possible de \perp

$p(X), \text{ not } q(X) \rightarrow r(X)$

$p(X), \text{ not } r(X) \rightarrow q(X)$

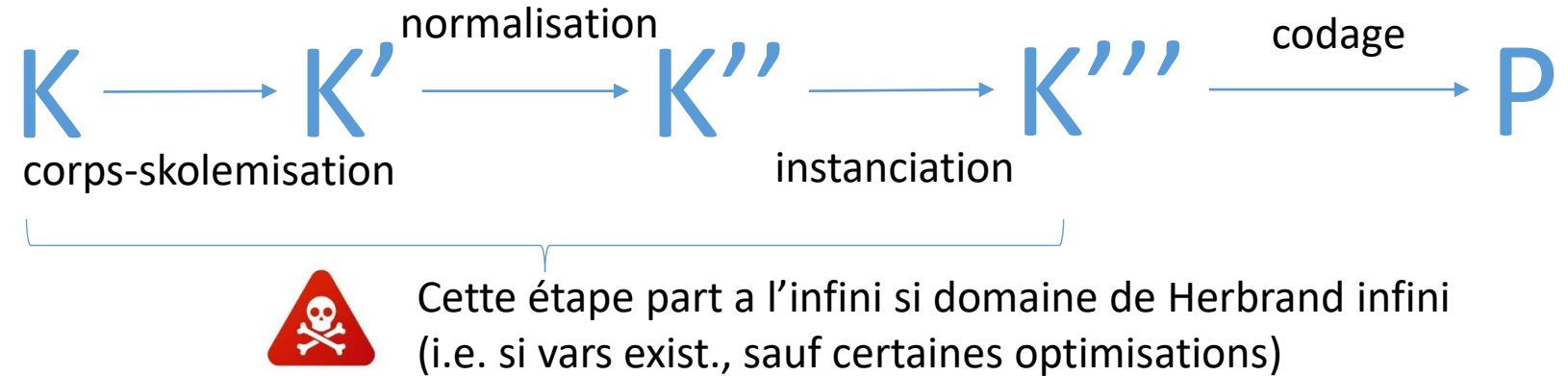
$p(a)$

Deux modèles stables

Implémentation possible de \vee

Dans les épisodes précédents: premier algorithme (définition usuelle)

1. Propositionalisation



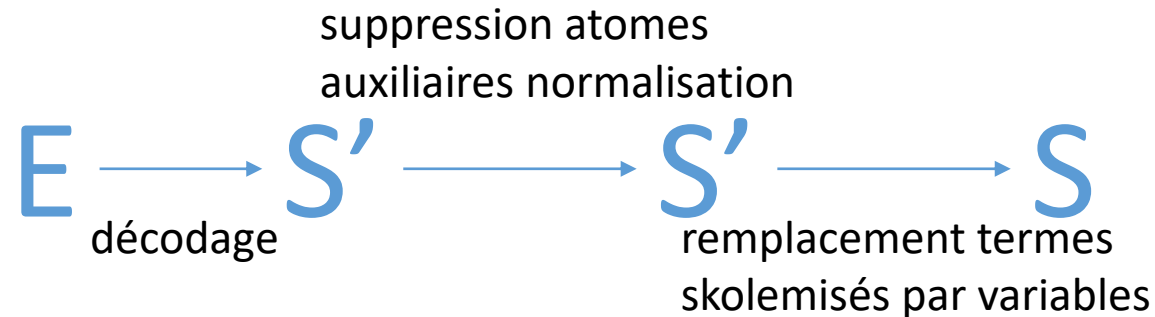
2. Générer

On devine un sous-ensemble E des atomes propositionnels

3. Tester

On teste si $E = P|_E^*$. Par définition (fin du suspense): E est un *modèle stable* de P .

4. Transformation inverse



S est le résultat d'une bonne dérivation de K ssi E est un modèle stable de P .

Echauffement

Maximalité des modèles stables

Soit K une KB (propositionnelle pour faire plus simple) et E_1 et E_2 2 modèles stables de K tels que $E_1 \subseteq E_2$



Pour simplifier la démo, j'utilise la DEF équivalente du programme réduit: « on garde les règles qui ne sont pas bloquées dans E »

1

Voir que $K_2 \subseteq K_1$ (avec $K_i = K_{|E_i}$)

Soit R une règle de K_2 . Elle provient d'une règle R non bloquée dans E_2 . Cette règle n'est pas non plus bloquée dans E_1 . Donc R est une règle de K_1 .

2

Voir que $E_2 \subseteq E_1$

De (1) on déduit que $K_2^* = E_2 \subseteq E_1 = K_1^*$.

CCL : Si $E_1 \subseteq E_2$, alors $E_1 = E_2$.

Les modèles stables sont maximaux.
(Ca aide un peu pour diminuer le nombre de MS à tester)



ASPERIX

ASPERIX: l'idée de base

On se place à la k-ième étape d'une bonne corps-skolem dérivation.

$$\textcircled{F} = F_0, F_1, F_2, \dots, F_i, F_{i+1}, \dots, F_k$$

h

On suppose une règle déclenchable à cette étape.

$$B^+, \text{ not } B_1^-, \dots, \text{ not } B_i^-, \dots, \text{ not } B_k^- \rightarrow H$$



Suspense...

Soit on l'applique

Soit on ne l'applique pas



ASPERIX: l'idée de base (application)

$$\textcircled{F} = F_0, F_1, F_2, \dots, F_i, F_{i+1}, \dots, F_k$$

\xrightarrow{h}

$$B^+, \text{ not } B_1^-, \dots, \text{ not } B_i^-, \dots, \text{ not } B_k^- \rightarrow H$$

Pour appliquer il faut qu'aucun des $h(B_i^-)$ ne soit déductible de F_k (pas bloqué), donc d'aucun des F_j , $j < k$ (monotonie), mais aussi d'aucun des F_p , $p > k$ (persistance). Comme on ne sait pas ce qui va se passer après F_k , il faudra garder cette info en mémoire (par exemple, si on met F_k dans un champs IN, on mettra chacun de ces $h(B_i^-)$ dans un champs OUT).



Importance de l'utilisation de règles **skolemisées**. Sinon il faudra « attacher les variables ». Par exemple, avec la règle $p(X), \text{ not } q(x) \rightarrow r(X)$, si elle est déclenchée (cas Skolem) sur $p(f(a))$, il faudra interdire $q(f(a))$, mais si elle est déclenchée (cas non Skolem) sur $p(Y127)$, il faudra interdire $q(Y127)$, mais cette variable là!

ASPERIX: l'idée de base (non application)

$$\textcircled{F} = F_0, F_1, F_2, \dots, F_i, F_{i+1}, \dots, F_k$$
$$B^+, \text{ not } B_1^-, \dots, \text{not } B_i^-, \dots, \text{not } B_k^- \xrightarrow{h} H$$

Pour ne pas appliquer il faut qu'au moins un des $h(B_i^-)$ soit déductible de F^* , sinon la dérivation ne serait pas complète. C'est-à-dire $h(B^-) = h(B_1^-) \vee \dots \vee h(B_k^-)$ doit être déductible de F^* .

Comme on ne sait pas ce qui va se passer après F_k , il faudra garder cette info en mémoire (par exemple, on mettra $h(B^-)$ dans un champs *MBT*, pour *Must Be True*).



Même remarque sur l'importance de l'utilisation de règles **skolemisées**, ou l'utilisation de variables attachées dans le cas contraire.

ASPERIX: l'évaluation, étape générale de l'algorithme

$B^+, \text{not } B_1^-, \dots, \text{not } B_i^-, \dots, \text{not } B_k^- \rightarrow H$

h ↗

IN	OUT	MBT
F_k	O_k	M_k

Si la règle est déclenchée par F_k , on va pouvoir l'évaluer.

On applique

On n'applique pas

IN	OUT	MBT
$F_k \cup h(H)$	$O_k \cup \{h(B_1^-), \dots, h(B_k^-)\}$	M_k

Ici $h(H)$ et pas $h^s(H)$
car Skolem

Ici on rajoute k éléments,
1 par corps négatif

IN	OUT	MBT
F_k	O_k	$M_k \cup \{h(B^-)\}$

Ici on rajoute 1 élément,
disjonction des k corps négatifs

Bonne branche: branche complète, dont tous les MBT sont satisfaits dans la branche, et dont aucun OUT n'est violé dans la branche.

Résultat d'une branche: union de tous les IN de la branche.

IN	OUT	MBT
F_i	$\{B_1, \dots, B_i, \dots, B_p\}$	$\{ \dots, (B'_1 \vee \dots \vee B'_j \vee \dots \vee B'_q), \dots \}$

Violation d'un OUT dans la branche \mathcal{B} .

\mathcal{B}

Satisfaction d'un MBT dans la branche \mathcal{B}' .

\mathcal{B}'

IN	OUT	MBT
F_j

IN	OUT	MBT
F_q

...

Branche complète: tous les déclenchements possibles ont été évalués.

ASPERIX: enfin l'algorithme...

IN	OUT	MBT
F	\emptyset	\emptyset

1

On part de la base de faits

2

On évalue tout ce qui est évaluable

...

3

Théorème: les résultats des bonnes branches sont exactement les résultats des bonnes dérivations.

Avantages:

- Pas besoin de normaliser
- Pas besoin d'instancier
- Generate and Test remplacé Par BackTrack

Problème : pas pratique quand l'arbre est infini.

Propriété: Si la forme positive des regles est corps-skolem-finite, alors l'algorithme termine (l'arbre est fini)

Question: et les autres dérivations?

Exemples bien connus revisités

Cas des règles existentielles

$$p(X) \rightarrow q(X)$$

$$p(a)$$

IN	OUT	MBT
$p(a)$	\emptyset	\emptyset

IN	OUT	MBT
$p(a)$ $q(a)$	\emptyset	\emptyset



IN	OUT	MBT
$p(a)$	\emptyset	$\{\text{FALSE}\}$

Car $\vee \emptyset = \text{FALSE}$



FALSE ne sera jamais
conséquence d'un IN
positif.

Simplification

IN	OUT	MBT
$p(a)$	\emptyset	\emptyset

IN	OUT	MBT
$p(a)$ $q(a)$	\emptyset	\emptyset

La petite KB stratifiable

$$p(X), \text{ not } q(X) \rightarrow r(X)$$

$$p(X) \rightarrow q(X)$$

$$p(a)$$

Plus malin d'appliquer les règles existentielles en premier (moins de sommets en général)

IN	OUT	MBT
p(a)	∅	∅

IN	OUT	MBT
p(a) r(a)	q(a)	∅

IN	OUT	MBT
p(a) r(a) q(a)	q(a)	∅



IN	OUT	MBT
p(a)	∅	q(a)

IN	OUT	MBT
p(a) q(a)	∅	q(a)



La petite KB stratifiable (2)

IN	OUT	MBT
p(a)	∅	∅

IN	OUT	MBT
p(a) q(a)	∅	∅

$p(X), \text{not } q(X) \rightarrow r(X)$

$p(X) \rightarrow q(X)$

p(a)

Simplification

Attention, même si on ne visualise pas la non application, il faut la compter comme évaluée!

IN	OUT	MBT
p(a)	∅	q(a)

IN	OUT	MBT
p(a) q(a)	∅	∅

IN	OUT	MBT
p(a) q(a) r(a)	q(a)	∅

Appliquer sur déjà bloqué: inutile car violation.



IN	OUT	MBT
p(a) q(a)	∅	q(a)

Ne pas appliquer sur déjà bloqué: inutile car n'ajoute rien.



Remarque: si on suit l'ordre de la stratification, une seule branche!

La petite KB absurde

$p(X), \text{ not } q(X) \rightarrow q(X)$
 $p(a)$

IN	OUT	MBT
$p(a)$	\emptyset	\emptyset

IN	OUT	MBT
$p(a)$ $q(a)$	$q(a)$	\emptyset



IN	OUT	MBT
$p(a)$	\emptyset	$q(a)$



Exercice : voir que si il y a aussi $q(a)$ dans la base de faits, il y a un modèle stable...

La petite KB disjonctive

$p(X), \text{ not } q(X) \rightarrow r(X)$
 $p(X), \text{ not } r(X) \rightarrow q(X)$

$p(a)$

IN	OUT	MBT
$p(a)$	\emptyset	\emptyset

IN	OUT	MBT
$p(a)$ $r(a)$	$q(a)$	\emptyset



IN	OUT	MBT
$p(a)$	\emptyset	$q(a)$

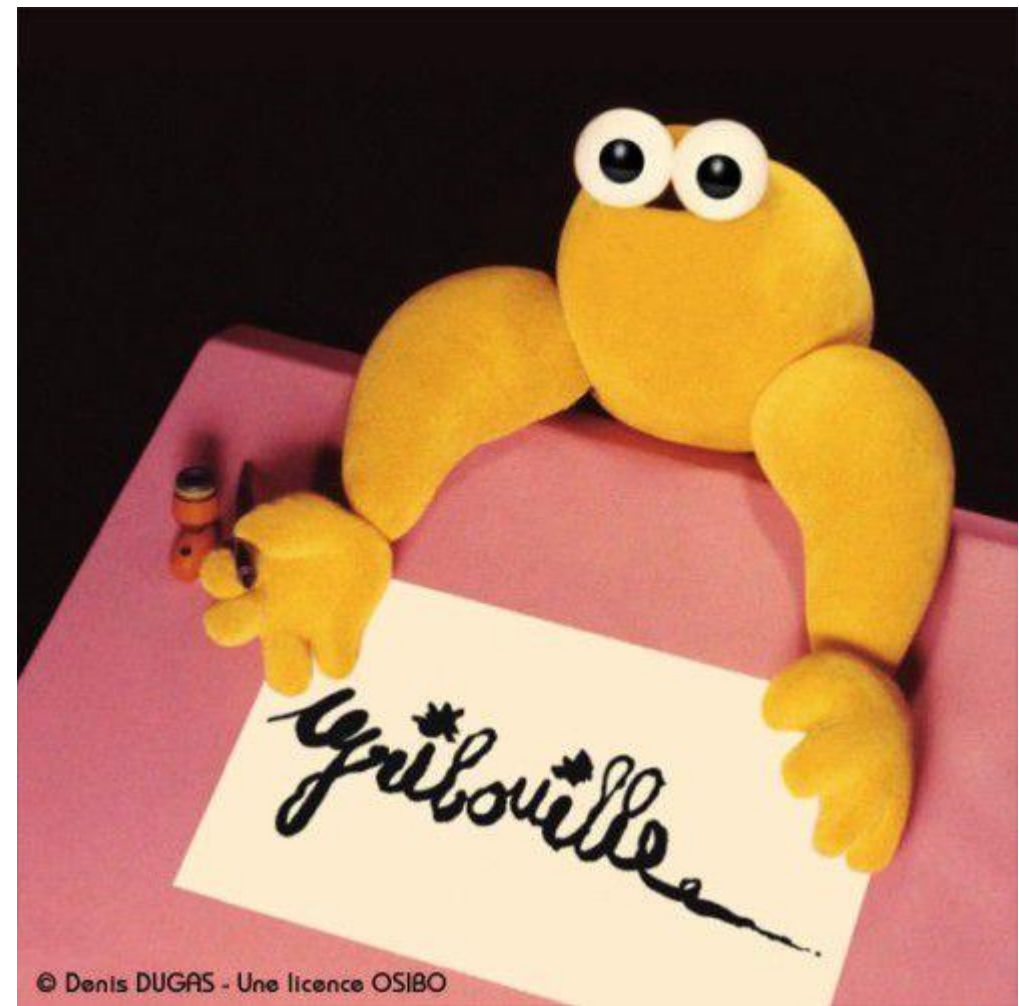
IN	OUT	MBT
$p(a)$ $q(a)$	$r(a)$	$q(a)$



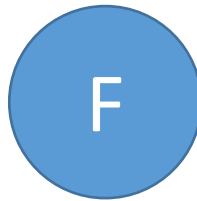
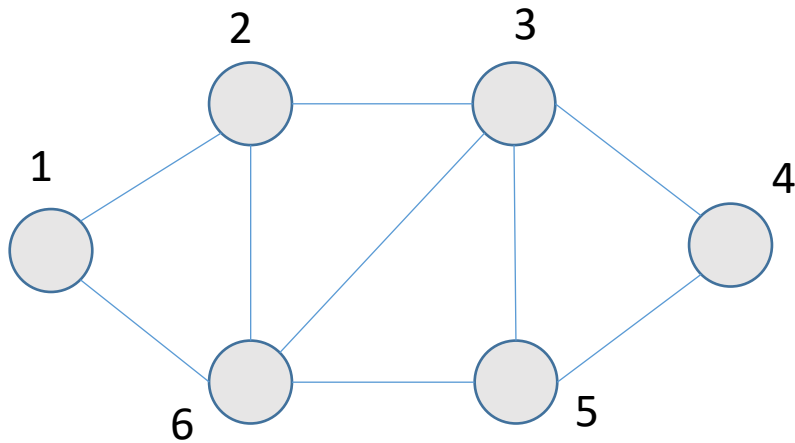
IN	OUT	MBT
$p(a)$	\emptyset	$q(a)$ $r(a)$



K-colorons un peu...



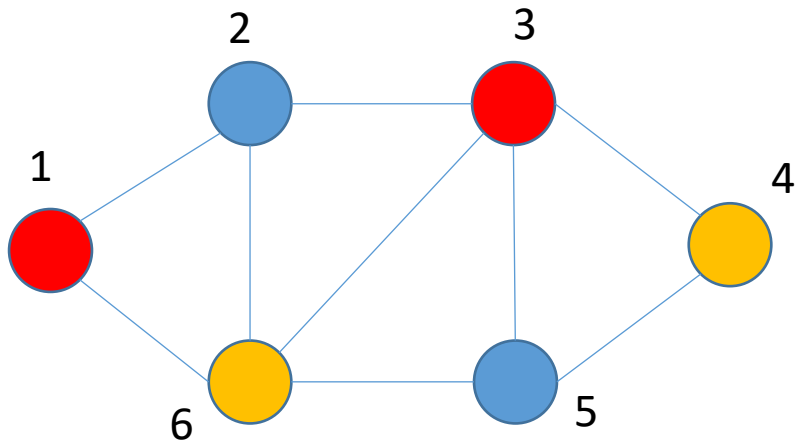
Codage d'une instance du problème « 3-coloration »



$s(1), s(2), s(3), s(4), s(5), s(6).$

$v(1, 2), v(1, 6), v(2, 3), v(2, 6),$
 $v(3, 4), v(3, 5), v(3, 6), v(4, 5).$

Ecrire un programme, qui calcule un modèle stable par 3-coloration du graphe (avec couleurs b, j, r)



Le modèle stable qui encodera cette 3-coloration devra contenir les atomes:

$c(1, r), c(2, b), c(3, r),$
 $c(4, j), c(5, b), c(6, j).$

Résolution du problème « 3-coloration »

$$v(X, Y) \rightarrow v(Y, X)$$

$$s(X), \text{ not } c(X, b), \text{ not } c(X, y) \rightarrow c(X, r)$$

$$s(X), \text{ not } c(X, b), \text{ not } c(X, r) \rightarrow c(X, y)$$

$$s(X), \text{ not } c(X, r), \text{ not } c(X, y) \rightarrow c(X, b)$$

$$v(X, Y), c(X, Z), c(Y, Z) \rightarrow \perp$$

Test de la base de règles « 3-coloration »

IN	OUT	MBT
$s(1)$	\emptyset	\emptyset

IN	OUT	MBT
$s(1)$ $c(1, r)$	$c(1, b)$ $c(1, j)$	\emptyset



$c(1, r)$ bloque l'application
des 2 autres règles.

$c(1, b)$ bloque l'application
des 2 autres règles.
 $c(1, b)$ est prouvé

IN	OUT	MBT
$s(1)$	\emptyset	$c(1, b) \vee c(1, j)$

IN	OUT	MBT
$s(1)$ $c(1, b)$	$c(1, j)$ $c(1, r)$	$c(1, b) \vee c(1, j)$



IN	OUT	MBT
$s(1)$	\emptyset	$c(1, b) \vee c(1, j)$ $c(1, j) \vee c(1, r)$

IN	OUT	MBT
$s(1)$ $c(1, j)$	$c(1, b)$ $c(1, r)$	$c(1, b) \vee c(1, j)$ $c(1, j) \vee c(1, r)$



$c(1, j)$ bloque l'application
des 2 autres règles.
 $c(1, j)$ est prouvé (2 fois)

IN	OUT	MBT
$s(1)$	\emptyset	$c(1, b) \vee c(1, j)$ $c(1, j) \vee c(1, r)$ $c(1, b) \vee c(1, r)$

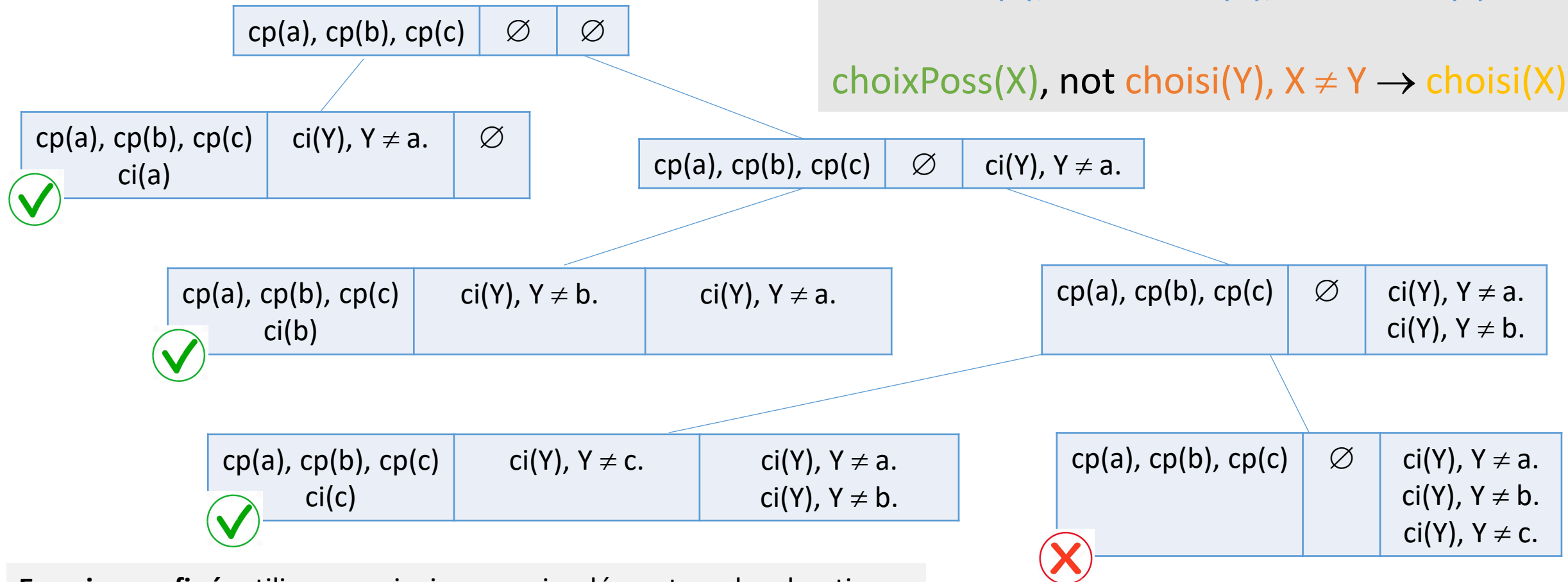


Choix multiples

Le problème de cette implémentation: ne fait que 3-coloration. On voudrait un programme qui fonctionne pour tous les k . Comme ça a été implémenté, ce n'est pas possible, il faudrait k règles, chacune excluant les $k-1$ autres choix...

$\text{choixPoss}(a), \text{choixPoss}(b), \text{choixPoss}(c).$

$\text{choixPoss}(X), \text{not } \text{choisi}(Y), X \neq Y \rightarrow \text{choisi}(X)$



Exercice confiné: utiliser ce principe pour implémenter « k -coloration »

Conclusion (et oui, je mange parfois a midi)

ASPERIX: un algorithme permettant de calculer les bonnes dérivations, sans normalisation, sans instanciation, via un BT

S'arrête si la forme positive des regles est corps-skolem-finite.

La sémantique dépend du type de chase utilisé (voir prochain cours)

On n'a pas encore abordé les pistes d'optimisation, ni le problème du requêtage.

Une première modélisation:

Comme dit l'OMS: « Testez, testez, testez »

Un principe de modélisation: générer toutes les solutions envisageables, utiliser \perp pour pruner.
Dans un deuxième temps, optimiser.

A suivre:

Moyen: le loup, la chevre, le choux, voire le Wumpus

Complicé: évolution d'un système complexe

Expert (voir avec Madalina): architecture agents rationnels/environnement