Correction Partie "Modèles stables" - HAI933I

Exercice 6 – Modèles stables (4 points)

Question 1 Utilisez la méthode vue en cours pour mettre le programme suivant sous forme propositionnelle. Vous détaillerez soigneusement les étapes de cette transformation.

$$p(a), q(a,b).$$
$$p(X), \text{not } q(X,Y) \to r(X).$$

Étape 1 : le programme est déjà skolemisé (pas de variable existentielle). *Étape 2 :* on doit normaliser la règle car toutes les variables de son corps négatif ne sont pas dans le corps positif. On obtient :

$$(F) \quad p(a), q(a,b).$$
$$(R) \quad p(X), \text{not } s(X) \to r(X).$$
$$(S) \quad q(X,Y) \to s(X).$$

Étape 3 (grounding) : le domaine de Herbrand est $\mathcal{H} = \{a, b\}$, on instancie les règles de toutes les façons possibles avec les constantes de $\mathcal{H}$.

$$(F) \quad p(a), q(a,b).$$
$$(R_1) \quad p(a), \text{not } s(a) \to r(a).$$
$$(R_2) \quad p(b), \text{not } s(b) \to r(b).$$

$$(S_1) \quad q(a,a) \to s(a).$$
$$(S_2) \quad q(a,b) \to s(a).$$
$$(S_3) \quad q(b,a) \to s(b).$$
$$(S_4) \quad q(b,b) \to s(b).$$

Dans le cours, j'avais donné une étape 4 de propositionalisation, qui consiste par exemple à donner le nom a_1 à l'atome $p(a)$, a_2 à l'atome $q(a,b)$ ...Mais j'avais également indiqué à l'oral que cette étape est facultative, puisqu'on peut déjà considérer que $p(a)$ est un atome propositionnel : il est juste écrit de façon un peu bizarre.

Détaillez le calcul d'une dérivation persistante et complète utilisant les règles que vous avez obtenues.

Remarque : si le résultat de cette dérivation n'est pas conforme à votre intuition, peut-être avez vous oublié une étape lors de votre transformation.

On part de $F = F_0 = \{p(a), q(a,b)\}$, on applique (S_2) pour obtenir $F_1 = \{p(a), q(a,b), s(a)\}$. Cette dérivation est bien complète (plus aucune règle n'est applicable – voir que (R_1) est bloquée par $s(a)$) et elle ne peut être que persistante puisqu'aucune règle avec corps négatif n'a été appliquée.

La deuxième partie (facile) de la question a souvent été oubliée : pensez à lire les sujets jusqu'au bout !

Question 2 Nous considérons maintenant le programme Π défini ci-dessous (7 lignes au total), et souhaitons savoir si les ensembles d'atomes $E_1 = \{a, c, d, f\}$ et $E_2 = \{a, e, f\}$ sont des modèles stables de ce programme.

$a.$

$a, \text{not } c \to b.$ $\quad$ $b \to c.$

$a \to f.$ $\quad$ $f, \text{not } e \to d.$

$f, \text{not } d \to e.$ $\quad$ $d \to c.$

Vous construirez les programmes réduits associés à ces deux ensembles d'atomes et les utiliserez pour répondre à la question en utilisant la définition par point fixe.

Le programme Π_1 obtenu par la réduction de Π par E_1 est :

$a.$ $\quad$ $b \to c.$

$a \to f.$ $\quad$ $f \to d.$

$d \to c.$

La saturation de Π_1 donne l'ensemble d'atomes $\Pi_1^* = \{a, f, d, c\}$. On a bien $\Pi_1^* = E_1$ donc E_1 est un modèle stable de Π.

De même, le programme Π_2 obtenu par la réduction de Π par E_2 est :

$a.$ $\quad$ $b \to c.$

$a \to b.$ $\quad$ $d \to c.$

$a \to f.$

$f \to e.$

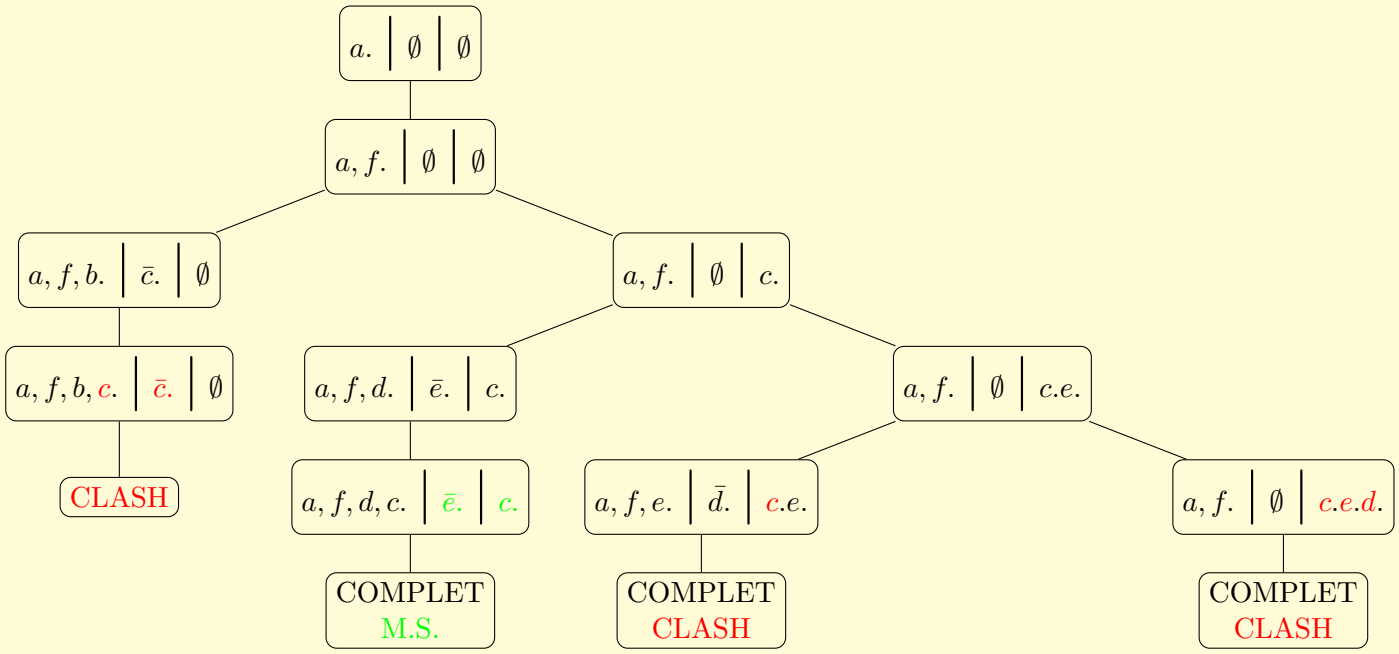
La saturation de Π_2 donne l'ensemble d'atomes $\Pi_2^* = \{a, b, f, e, c, d\}$. On n'a pas $\Pi_2^* = E_2$ donc E_2 n'est pas un modèle stable de Π.

Question 3 Reprenons le programme Π de la question 2. Les ensembles d'atomes $E_4 = \{a, c, d, e, f\}$, et $E_5 = \{c, d, e, f\}$ sont-ils des modèles stables de ce programme ? Votre réponse devra être argumentée, mais n'utilisera pas nécessairement la définition par point fixe.

L'ensemble d'atome E_4 ne peut pas être un modèle stable car on a $E_1 \subset E_4$: or E_1 est un modèle stable, et ceux-ci sont maximaux par inclusion.

De même l'ensemble d'atomes E_5 n'est pas un modèle stable : il ne contient pas a qui est un fait et doit donc être dans tous les modèles stables.

Question 4 Utilisez l'algorithme ASPERIX vu en cours pour trouver tous les modèles stables du programme Π de la question 2. *Vous vérifierez que vos réponses aux questions précédentes sont cohérentes.*



La seule branche complète sans clash nous donne le modèle stable E_1 de la question précédente.

Comme très souvent, vous vous êtes perdus dans des arbres ASPERIX beaucoup trop grands ! Un peu de stratégie : appliquez d'abord les règles positives, qui ne créent pas de nouvelles branches, puis des règles dont la conclusion permettra d'appliquer de nouvelles règles positives. Et c'est encore mieux quand ces applications permettent la violation d'une contrainte négative...

Considérant le nombre de modèles stables que vous avez trouvé, pouvez-vous en déduire si le programme Π est stratifiable ?

On sait que si Π est stratifiable, alors il y aura un et un seul modèle stable. Mais la réciproque n'est pas vraie : on ne peut pas déduire que le programme est stratifiable du fait qu'il admette un et un seul modèle stable.

Exercice 7 – Modélisation en ASP – 4 pts

Les points sont accordés si au moins la moitié de l'exercice est correctement traitée.

Dans le jeu “Le compte est bon”, il faut réussir à obtenir un nombre cible à partir de 6 nombres initiaux, en utilisant les 4 opérations arithmétiques élémentaires : addition, soustraction, multiplication, division. Les conditions initiales du jeu seront codées comme dans l'exemple suivant :

$$\begin{aligned}&\text{disponible}(a, 25, 0).\\ &\text{disponible}(b, 10, 0).\\ &\text{disponible}(c, 10, 0).\\ &\text{disponible}(d, 7, 0).\\ &\text{disponible}(e, 3, 0).\\ &\text{disponible}(f, 4, 0).\\ &\text{cible}(284).\end{aligned}$$

L'atome $\text{disponible}(a, 25, 0).$ se lit *“Le nombre qui a pour identifiant a, dont la valeur est 25, est disponible à l'étape 0”*. Coder uniquement le fait que *“le nombre 25 est disponible à l'étape 0”* ne permettrait pas d'avoir

des occurences multiples d'un même nombre. En supposant qu'on ait pu choisir 2 nombres et une opération à chaque étape, 4 règles conçues sur le modèle de la règle suivante (qui concerne l'addition) permettent de générer un nouveau nombre à l'étape N+1 :

$$\text{choisi1}(X1, N), \text{choisi2}(X2, N), \text{opchoisie}(\text{addition}, N),$$
$$\text{disponible}(X1, V1, N), \text{disponible}(X2, V2, N) \rightarrow \text{disponible}(Y, V1 + V2, N + 1).$$

Le but de l'exercice est d'écrire un programme dont les modèles stables encodent toutes les manières d'arriver au nombre cible. Il s' agit d'une version simplifiée du jeu, en particulier, on ne cherche pas à générer un nombre “le plus proche possible” de la cible.

Question 1 Ecrire une règle permettant de générer l'atome $fini()$ dès que la valeur cible a été trouvée. Complétez ensuite le programme pour que les modèles stables contiennent uniquement les cas où cette valeur cible a été trouvée.

```
fini(N) :- disponible(X, V, N), cible(V).
% Deuxieme partie de la question
autredisponible(X, N) :- disponible(X, V, N), disponible(Y, W, N), X != Y.
:- disponible(X, V, N), not fini(N), not autredisponible(X, N).
```

Question 2 L'atome $choisi1(X, N)$ signifie que le nombre d'identifiant X a été choisi comme premier argument de notre calcul à l'étape N. Ecrire la (ou les) règle(s) permettant de générer cet atome. Vous vous inspirerez du choix multiple vu en cours, et prendrez en compte les points suivants :

- il n'y a plus besoin de faire de choix quand le jeu est fini ;
- on ne peut pas faire de choix quand on n'a pas au moins deux identifiants de nombres disponibles ;
- un seul identifiant de nombre peut être choisi.

```
choisi1(X, N) :- disponible(X, V, N), not fini(N), not autrechoisi1(X, N).
autrechoisi1(X, N) :- disponible(X, V, N), choisi1(Y, N), X != Y.
```

C'est exactement le choix n-aire vu en cours. Voir qu'il y a nécessairement deux disponibles sinon la contrainte de la réponse à la question 1 serait déclenchée.

Question 3 L'atome $choisi2(X, N)$ signifie que le nombre d'identifiant X a été choisi comme deuxième argument de notre calcul à l'étape N. Ecrire la (ou les) règle(s) permettant de générer cet atome. Vous vous inspirerez du choix multiple vu en cours, et prendrez en compte les points suivants :

- on ne fait le choix 2 qu'après le choix 1 ;
- on ne peut pas faire le même choix en choix2 qu'en choix1 ;
- un seul identifiant de nombre peut être choisi.

```
choisi2(X, N) :- disponible(X, V, N), choisi1(Y, N), X!= Y, not autrechoisi2(X, N).
autrechoisi2(X, N) :- disponible(X, V, N), choisi2(Y, N), X != Y.
```

Question 4 L'atome $opchoisie(X, N)$ signifie que l'opération X a été choisie à l'étape N. Ecrire la (ou les) règle(s) permettant de générer cet atome. Vous vous inspirerez du choix multiple vu en cours, et prendrez en compte les points suivants :

- on ne peut choisir que parmi les opérations codées dans les atomes suivants : $op(addition), op(soustraction), op(multiplication), op(division)$;
- il n'y a plus besoin de faire de choix quand le jeu est fini ;
- une seule opération peut être choisie à chaque étape ;
- (OPTIONNEL) on ne peut choisir la division que si son résultat est un nombre entier : on pourra tester dans le corps de la règle $X1\%X2 == 0$ pour savoir si la valeur $X1$ du premier nombre choisi est divisible par la valeur $X2$ du second.

```
opchoisie(O, N) :- op(O), chosi1(X, N), choisi2(Y, N), not autreop(O, N), not impossible(O, N).
valeurs(V, W, N) :- choisi1(X, N), choisi2(Y, N), disponible(X, V, N), disponible(Y, W, N).
impossible(division, N) :- valeurs(V, W, N), V \ W != 0.
```

Question 5 Ecrire la ou les règles permettant d'assurer que les nombres qui n'ont pas été choisis à une étape sont encore disponibles à l'étape suivante.

```
disponible(X, V, N+1) :- disponible(X, V, N), not choisi1(X, N), not choisi2(X, N).
```

Question 6 L'algorithme ASPERIX s'arrête-t-il sur votre programme ? Justifiez votre réponse. Montrez que le grounding de ce programme est infini. Identifiez-en toutes les causes. Proposez une modification de votre programme pour obtenir un grounding fini.

L'algorithme ASPERIX s'arrête sur ce programme car à chaque étape on choisit 2 nombres. Les autres nombres restent disponibles, ainsi que celui généré par l'opération. Le nombre de dispobibles décroît donc à chaque étape et donc pour chaque branche, au bout d'un nombre fini d'opérations, soit on aura généré `fini` et plus aucun choix ne sera effectué, soit la contrainte de la question 1 sera déclenchée.

Par contre, le grounding de CLINGO est infini. Les règles responsables de ceci sont :

1. la règle de calcul donnée dans l'énoncé qui :
   (a) crée une nouvel identificateur `Y`
   (b) crée un nouveau nombre `V1 + V2`
   (c) crée une nouvelle étape `N+1`
2. la règle de la question 5 qui crée une nouvelle étape `N+1`

Le problème 1.a peut être facilement réglé. Il suffit d'écrire la règle sous la forme :

```
disponible(X, V+W, N+1) :- opchoisie(addition, N), chois1(X, V, N), valeurs(V, W, N).
```

En gardant l'identifiant du choix 1 pour a valeur obtenue, on n'a pas besoin de générer de nouvel identificateur.

Pour les problèmes 1.c et 2, la solution vue en cours (borner le nombre d'étapes) peut être utilisée. On peut rajouter dans le programme :

```
#const maxetape = 5
etape(0..maxetape).
```

Puis modifier la règle précédente et la règle de la question 5 en utilisant cette borne :

```
disponible(X, V+W, N+1) :- opchoisie(addition, N), chois1(X, V, N), valeurs(V, W, N), etape(N+1).
disponible(X, V, N+1) :- disponible(X, V, N), not choisi1(X, N), not choisi2(X, N), etape(N+1).
```

On remarque que si l’instance du problème “Le compte est bon” commence avec N identificateurs de nombres, on reste complet en prenant `const maxetape = N`.

Curieusement, lorsqu’on effectue ces modifications, le grounding de CLINGO (qui est plus intelligent que la version naïve vue en cours) s’arrête maintenant si on ne considère que les opérations `addition`, `multiplication` et `soustraction`. Mais dès qu’on rajoute `division`, le grounding redevient infini.

Une solution consiste en la génération initiale de toutes les valeurs possibles pour aider le grounding de CLINGO. Par exemple :

```
possible(V, N):- disponible(X, V, 0).
possible (V+W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1).
possible (V-W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1).
possible (V*W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1).
possible (V/W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1), V\W == 0.
```

Puis l’utilisation de ces possibles dans les règles générant de nouveaux nombres :

```
disponible(X, V+W, N+1) :- opchoisie(addition, N), chois1(X, V, N), valeurs(V, W, N), etape(N+1),
    possible(V+W, M).
```

La le grounding s’arrête, mais l’espace de recherche est tellement énorme que CLINGO plante à partir de `maxetape = 3`.