Projet modèles stables janvier 2021 -- correction

Exercice 1

Question 1

Le code suivant exprime que toute affirmation est soit véridique, soit un mensonge.

```
lie(C) :- say(A, C), not truth(C).
truth(C) :- say(A, C), not lie(C).
```

Il ne fallait surtout pas oublier d'exprimer qu'une affirmation ne peut pas être à la fois véridique et un mensonge (oublié dans beaucoup trop de copies). Je donne ici 3 façons différentes de le coder:

Méthode 1 (celle vue en cours)

```
abs() :- truth(C), lie(C).
absaux() :- abs(), not absaux(). % codage de abs() par un predicat absaux()
utilisé nulle part ailleurs
```

Méthode 2 Simplification de 1, mais il faut recoder `abs()` à chaque règle où on souhaite l'utiliser.

```
abs() :- truth(C), lie(C), not abs().
```

Méthode 3 Utilisation de la syntaxe ASP, plusieurs atomes en tête signifient une disjonction, et une disjonction vide veut dire absurde.

```
:- truth(C), lie(C). % tête vide = absurde
```

Question 2

Ce programme n'est pas stratifiable: on pouvait soit faire le dessin du graphe de dépendance des prédicats et voir qu'il y a un circuit négatif entre truth et lie, soit remarquer que, si on ajoute par exemple l'atome `say(A, C)` c'est exactement le petit programme disjonctif introductif du cours, qui a 2 modèles stables (et donc n'est pas stratifiable).

Question 3

On va transformer le programme suivant (méthode 2 de la question 1):

```

```

```
say(bob, c).
lie(c).
lie(C) :- say(A, C), not truth(C).
truth(C) :- say(A, C), not lie(C).
abs() :- truth(C), lie(C), not abs().
```

Suivons exactement les étapes vues en cours...

Normalisation Pour chacune des règles, on vérifie que toutes les variables apparaissant à la fois dans la tête et un corps négatif apparaissent déjà dans le corps positif. Le programme est déjà normal, donc pas besoin de normaliser.

Skolémisation Il n'y a aucune variable existentielle, donc pas besoin de skolémiser.

Grounding Le domaine de Herbrand est HB = {bob, c}. Attention, il est bien composé de constantes et pas d'atomes ground comme je l'ai vu trop souvent. Le programme ground est donc:

```
% les faits sont déjà ground, et font partie du programme
say(bob, c).
lie(c).
% Grounding de la première règle
lie(c) :- say(bob, c), not truth(c). % avec C <- c et A <- bob
lie(c) :- say(c, c), not truth(c).
lie(bob) :- say(c, bob), not truth(bob).
lie(bob) :- say(bob, bob), not truth(bob).
% Grounding de la deuxième règle
truth(c) :- say(bob, c), not lie(c). % avec C <- c et A <- bob
truth(c) :- say(c, c), not lie(c).
truth(bob) :- say(c, bob), not lie(bob).
truth(bob) :- say(bob, bob), not lie(bob).
% Grounding de la troisième règle
abs() :- truth(c), lie(c), not abs().
abs() :- truth(bob), lie(bob), not abs().
```

Propositionalisation Avec les conventions proposées dans le sujet on obtient:

```
% Faits
sbc.
lc.
% Grounding de la première règle
lc :- sbc, not tc.
lc :- scc, not tc. % pas declenchable
lt :- scb, not tb. % pas declenchable
lt :- sbb, not tb. % pas declenchable
% Grounding de la deuxième règle
tc :- sbc, not lc.
tc :- scc, not lc. % pas declenchable
tb :- scb, not lb. % pas declenchable
```

```
tb :- sbb, not lb. % pas declenchable
% Grounding de la troisième règle
abs :- tc, lc, not abs.
abs :- tb, lb, not abs. % pas declenchable
```

Les règles non déclenchables ne servent à rien mais sont quand même obtenues par grounding/propositionalisation. J'aurai voulu que vous me montriez que vous avez vu que ces règles sont obtenues par grounding. Si on enlève ces règles qui ne servent à rien, on obtient exactement le programme propositionnel que je vous ai donné.

Question 4

On calcule le programme réduit par E = {sbc, lc}. J'utilise le programme simplifié donné dans le sujet.

```
sbc. lc. % les atomes de la base de faits (je ne parle pas de E) sont toujours
présents
lc :- sbc. % car tc pas dans E, et on enleve le corps négatif.
% on supprime tc :- sbc, not lc. car lc dans E
abs :- tc, lc. % car abs pas dans E, et on enleve le corps négatif.
```

En saturant ce programme réduit, on obtient les atomes {sbc, lc}, c'est à dire E, donc E est un modèle stable du programme.

Question 5

La réponse la plus rapide (qui répond en même temps au début de la question 6) me semble: la base de faits étant {sbc, lc}, tout modèle stable contient {sbc, lc}. Or {sbc, lc} est un modèle stable (question 4), et tout modèle stable est maximal (cours). Donc E = {sbc, lc} est le seul modèle stable possible, ceux proposés dans la question ne le sont donc pas.

Question 6

Nous avons vu à la question 6 que le programme n'avait qu'un modèle stable. Le théorème vu en cours dit que si le programme est stratifiable, alors il n'y a qu'un modèle stable, mais la réciproque n'est pas vraie. On ne peut donc pas conclure.

Exercice 2

Question 7

```
truth(C) :- claim(C, S, P, O), triple(S, P, O).
lie(C) :- claim(C, S, P, O), not triple(S, P, O).
triple(C, S, P, O) :- claim(C, S, P, O), truth(C).
:- claim(C, S, P, O), lie(C), triple(S, P, O).
```

Question 8

Question 9

Exercice 3

Question 10

Comme l'un d'entre vous me l'a fait remarqué, j'ai oublié la règle permettant de déduire le mensonge sur la conjonction... On devrait avoir:

```
lie(C) :- conjonction(C, C1, C2), lie(C1).
lie(C) :- conjonction(C, C1, C2), lie(C2).
```

Question 11

```
truth(C) :- disjonction(C, C1, C2), truth(C1).
truth(C) :- disjonction(C, C1, C2), truth(C2).
lie(C) :- disjonction(C, C1, C2), lie(C1), lie(C2).
lie(C1) :- disjonction(C, C1, C2), lie(C).
lie(C2) :- disjonction(C, C1, C2), lie(C).
truth(C2) :- disjonction(C, C1, C2), truth(C), lie(C1).
truth(C1) :- disjonction(C, C1, C2), truth(C), lie(C2).
```

Question 12

```
lie(D) :- negation(C, D), truth(C).
truth(D) :- negation(C, D), lie(C).
lie(C) :- negation(C, D), truth(D).
truth(C) :- negation(C, D), lie(D).
```

Exercice 4

Question 13

```
say(affiche1, a1).
conjonction(a1, b1, b2).
claim(b1, princess, in, cell1).
claim(b2, tiger, in, cell2).
```

Question 14

```
say(affiche2, a2).
disjonction(a2, c1, c2).
```

```
conjonction(c1, d1, d2).
claim(d1, princess, in, cell1).
claim(d2, tiger, in, cell2).
conjonction(c2, e1, e2).
claim(e1, princess, in, cell2).
claim(e2, tiger, in, cell1).
```

Question 15

```
lie(a2) :- truth(a1).
truth(a2) :- lie(a1).
```

Pour aller plus loin

Question 16

```
lie(C) :- conjunction(C), argument(C, A), lie(A).
hasLie(C) :- conjunction(C), argument(C, A), lie(A).
truth(C) :- conjunction(C, A), not hasLie(C).
truth(A) :- conjunction(C, A), truth(C).
hasLieOtherThan(C, A) :- conjunction(C, A), argument(C, A), argument(C, B), A !=
B, lie(B).
lie(A) :- conjunction(C, A), argument(C, A), lie(C), not hasLieOtherThan(C, A).
```

Question 17

```
% les 2 règles de la question 8 fonctionnent encore mais il faut changer les
données
cell(cell1;cell2;cell3;cell4;cell5;cell6;cell7;cell8;cell9). % Il y a 9 cellules
entity(tiger;princess;void) % Une cellule peut être vide

% codage de la disjonction n-aire
truth(C) :- disjunction(C), argument(C, A), truth(A).
hasTruth(C) :- disjunction(C), argument(C, A), truth(A).
lie(C) :- disjunction(C, A), not hasTruth(C).
lie(A) :- disjunction(C, A), lie(C).
hasTruthOtherThan(C, A) :- disjunction(C, A), argument(C, A), argument(C, B), A !=
B, truth(B).
truth(A) :- disjunction(C, A), argument(C, A), truth(C), not hasTruthOtherThan(C,
A).

% assertion de l'affiche 1 (avec disjonction n-aire)
say(affiche1, c1).
disjunction(c1). argument(c1, cp1). argument(c1, cp3). argument(c1, cp5).
argument(c1, cp7). argument(c1, cp9).
claim(cp1, princess, in, cell1). claim(cp3, princess, in, cell3). claim(cp5,
```

```
princess, in, cell5).
claim(cp7, princess, in, cell7). claim(cp9, princess, in, cell9).

% assertion de l'affiche 2
say(affiche2, c2).
claim(c2, void, in, cell2).

% introduction du judgement: un claim parle du réel, un judgement parle d'un
judgement ou d'un claim
truth(JC) :- judgement(J, true, JC), truth(J).
lie(JC) :- judgement(J, false, JC), truth(J).
lie(JC) :- judgement(J, true, JC), lie(J).
truth(JC) :- judgement(J, false, JC), lie(J).
truth(J) :- judgement(J, true, JC), truth(JC).
lie(J) :- judgement(J, false, JC), truth(JC).
lie(J) :- judgement(J, true, JC), lie(JC).
truth(J) :- judgement(J, false, JC), lie(JC).
lie(J) :- judgement(J, TV, JC), not truth(J).
truth(J) :- judgement(J, TV, JC), not lie(J).

% assertion de l'affiche 3
say(affiche3, c3).
disjunction(c3, c31, c32). % avec le binaire
judgement(c31, true, c5).
judgement(c32, false, c7).

% assertion de l'affiche 4
say(affiche4, c4).
judgement(c4, false, c1).

% assertion de l'affiche 5
say(affiche5, c5).
disjunction(c5, c51, c52).
judgement(c51, true, c2).
judgement(c52, true, c4).

% assertion de l'affiche 6
say(affiche6, c6).
judgement(c6, false, c3).

% assertion de l'affiche 7
say(affiche7, c7).
negation(c7, c70).
claim(c70, princess, in, cell1).

% assertion de l'affiche 8
say(affiche8, c8).
conjunction(c8, c81, c82).
claim(c81, tiger, in, cell8).
claim(c82, void, in, cell9).

% assertion de l'affiche 9
say(affiche9, c9).
conjunction(c9, c91, c92).
```

```
claim(c91, tiger, in, cell9).
judgement(c92, false, c6).

% indices du roi
truth(c1) :- triple(princess, in, cell1).
truth(c2) :- triple(princess, in, cell2).
truth(c3) :- triple(princess, in, cell3).
truth(c4) :- triple(princess, in, cell4).
truth(c5) :- triple(princess, in, cell5).
truth(c6) :- triple(princess, in, cell6).
truth(c7) :- triple(princess, in, cell7).
truth(c8) :- triple(princess, in, cell8).
truth(c9) :- triple(princess, in, cell9). % trop de ligne, mauvais codage !
lie(c1) :- triple(tiger, in, cell1).
lie(c2) :- triple(tiger, in, cell2).
lie(c3) :- triple(tiger, in, cell3).
lie(c4) :- triple(tiger, in, cell4).
lie(c5) :- triple(tiger, in, cell5).
lie(c6) :- triple(tiger, in, cell6).
lie(c7) :- triple(tiger, in, cell7).
lie(c8) :- triple(tiger, in, cell8).
lie(c9) :- triple(tiger, in, cell9). % trop de ligne, mauvais codage !
```