1 Partie modèles stables (8 pts + 2 pts bonus)

1.1 Application des algorithmes vus en cours (3,5 pts)

On se donne le programme de règles existentielles avec négation suivant:

```
p(1). p(2). p(3).
c1(X) :- p(X), not o1(X).
o1(X) :- c1(Y), p(X), X != Y.
c2(X) :- p(X), not c1(X), not o2(X).
o2(X) :- c2(Y), p(X), X != Y.
```

Question 1: mettre les deux premières règles de ce programme sous forme propositionnelle. Attention, dans votre programme propositionnel, *le symbole de différence devra disparaître*![1]

Question 2: en utilisant le programme de la question 1 et la définition par point fixe, prouvez que l'ensemble d'atomes:

{`p(1), p(2), p(3), c1(1), o1(2), o2(3), c2(2), o2(1), o2(3)`}

est un modèle stable du programme. Vous justifierez soigneusement votre réponse, notamment en exhibant le programme réduit qui, comme indiqué à la question 1, ne devra pas contenir de symbole de différence.

Question 3: Dessinez (une partie de) l'arbre de recherche ASPERIX utilisant le programme non instancié initial. Vous pourrez arrêter l'arbre de recherche quand il vous permettra de répondre à la question suivante: *le programme initial est-il stratifiable?* Vous justifierez soigneusement pourquoi vous vous êtes arrêté, mais n'avez pas droit au graphe de dépendance des prédicats pour justifier que le programme n'est pas stratifiable.

1.2 Modélisation: le problème du pont (4,5 pts)

Nous nous proposons dans cette partie de faire résoudre à Clingo un petit problème combinatoire. Je rappelle que si (1) les règles n'ont qu'un seul atome en tête et (2) les règles ne contiennent pas de variable existentielle, alors les sémantiques de Clingo et des règles existentielles avec négation coïncident. Dans les questions suivantes, vous vous conformerez à ces restrictions.

[1] En effet, supposons une règle `q(X) :- p(X), X != b.` Si vous instanciez avec {`X: a`}, alors l'atome `X != b` sera évalué à vrai et inutile dans la règle qui deviendra `q(a) :- p(a).` Par contre, si vous instanciez avec {`X: b`}, alors l'atome `X != b` sera évalué à faux et c'est toute la règle qui deviendra inutile, car elle n'est jamais déclenchable.

Le problème du pont: *Quatre personnes doivent traverser un pont en 17 minutes. Chacune d'entre elles marche à une vitesse maximale donnée. Appelons 1, la personne qui peut traverser le pont en 1 minute, 2 celle qui le traverse en 2 minutes, 5 celle qui le fait en 5 minutes et 10 celle qui le traverse en 10 minutes. Ces quatre personnes n'ont en tout qu'une torche et il est impossible de traverser le pont sans torche. Le pont ne peut supporter que le poids de 2 personnes. Dans quel ordre doivent traverser ces quatre personnes ?*

Vous ne répondrez pas à la question "*Dans quel ordre doivent traverser ces quatre personnes ?*", mais écrirez un programme qui permet à Clingo de le faire.

La base de faits et le vocabulaire: la base de faits est la suivante:

```
position(1, side1, 0). position(2, side1, 0).
position(5, side1, 0). position(10, side1, 0).
torch(side1, 0).
opposite(side1, side2). opposite(side2, side1).
```

Ici, `position(1, side1, 0).` signifie que la personne `1` (celle qui met 1 minute à traverser le pont) se trouve du côté 1 (`side1`) du pont au temps `0`; `torch(side1, 0).` signifie que la torche est du côté 1 au temps `0`, et les prédicats `opposite` vous aideront par la suite à écrire moins de règles.

Question 4: Afin de lancer la partie du programme qui va choisir la ou les personnes qui vont traverser avec la torche, nous avons la règle suivante:

```
init(S, N) :- torch(S, N), not win(N).
```

qui dit que si la torche est sur le côté `S` au temps `N` et qu'on n'a pas encore gagné au temps `N`, alors on en déduit l'atome `init(S, N)` qui sera utilisé pour indiquer qu'il faut faire traverser 1 ou 2 personnes depuis le côté `S` au temps `N`. Mais pour celà, il nous faut écrire une règle qui encode les conditions de succès.

Ecrire la règle dont la tête est `win(N)` et qui est applicable lorsque les conditions de succès sont vérifiées au temps `N`.

Question 5: En adaptant le plus simplement possible le programme de la question 1, compléter le programme écrit jusqu'à présent avec des règles qui ont pour effet, lorqu'on a déduit `init(W, N)`, de générer, pour chaque paire {`P1, P2`} de personnes situées sur le côté `W` un candidat modèle stable contenant les atomes `chose1(P1, N)` et `chose2(P2, N)`. Vous discuterez de l'opportunité de spécifier, comme à la question 1, que `P1` doit être différent de `P2`. Vous pourrez utiliser les prédicats auxiliaires `otherchoice1` et `otherchoice2`. *Attention*, vous vous faciliterez grandement les questions suivantes si vous réussissez à imposer que `P1` soit plus lent que `P2`.

Question 6: Maintenant que vous avez choisi la ou les personnes qui traversent au temps `N`, écrivez une ou plusieurs règles qui mettent à jour les positions des personnes et de la torche au temps `N + V`, où `V` est la vitesse de la plus lente des personnes qui ont traversé au temps `N`.

Question 7: Si vous avez correctement suivi les spécifications du programme jusqu'à présent, vous pouvez remarqué que votre programme ne s'arrête pas. Justifiez cette affirmation. Proposez des modifications simples de votre programme pour que l'algorithme ASPERIX s'arrête. Etes-vous certain que l'algorithme de Clingo s'arrête également?

Question 8: Rajoutez maintenant une règle pour que les seuls modèles stables énumérés par Clingo encodent une solution du problème.

1.3 Enumération des modèles stables ($*$ 2 points bonus)

Comme lors du contrôle précédent, nous cherchons ici à réduire le nombre de sous-ensembles du vocabulaire d'un programme à explorer pour énumérer tous les modèles stables de ce programme.

Question 9: Prouvez la propriété suivante:

Propriété A: *Si* Π *est un programme propositionnel et* $E \subseteq F$ *sont deux ensembles d'atomes, alors* $(\Pi_{|F})^* \subseteq (\Pi_{|E})^*$.

Nous notons ici $\Pi_{|E}$ le programme positif obtenu en réduisant le programme Π par l'esnemble d'atomes Π et Π^* l'ensemble d'atomes obtenu par saturation d'un programme positif Π.

Attention, vous justifierez quelle version (faible ou forte) du programme réduit vous utilisez pour arriver à ce résultat, et montrerez que cette affirmation est fausse si on utilise l'autre version.

Question 10: Utilisez la propriété A de la question 10 pour prouver les propriétés suivantes:

Propriété B: *Si* $(\Pi_{|E})^* \subsetneq E$, *alors aucun super-ensemble de* E *ne peut être un modèle stable de* Π.

Propriété C: *Si* $E \subsetneq \Pi_{|E})^*$, *alors aucun sous-ensemble de* E *ne peut être un modèle stable de* Π.