Answer Set Programming (ASP)

Exercice 1 Sémantique stable

Question 1. Programmes positifs

On considère les programmes positifs suivants :

$$\Pi_1 \begin{cases} a \leftarrow b. \\ b \leftarrow a. \\ a. \end{cases} \qquad \Pi_2 \begin{cases} a \leftarrow b. \\ b \leftarrow a. \end{cases}$$

1. Donner tous les modèles (ensemble d'atomes satisfaisant toutes les règles du programme) de Π_1 et de Π_2.

2. Parmi ces modèles, indiquer quel est le modèle stable (ou answer set) de Π_1, puis de Π_2. Pourquoi est-il unique ?

3. Exprimer l'unique modèle stable en terme de point fixe. Faire le lien avec le chaînage avant vu pour les systèmes de règles.

Question 2. Programmes normaux

On considère maintenant les programmes normaux suivants :

$$\Pi_3 \begin{cases} a \leftarrow not\ b. \\ c \leftarrow not\ a. \end{cases} \qquad \Pi_4 \begin{cases} a \leftarrow not\ b. \\ b \leftarrow not\ a. \end{cases}$$

1. Donner tous les modèles de Π_3 et Π_4.
2. Parmi ces modèles, lesquels sont des modèles stables (ou answer sets) de Π_3 ? de Π_4 ? On utilisera pour cela le théorème avec la réduction du programme.

Question 3. Contraintes d'intégrité

On considère les programmes suivants, contenant des contraintes d'intégrité :

$$\Pi_5 \begin{cases} a \leftarrow b. \\ b. \\ \leftarrow a. \end{cases} \qquad \Pi_6 \begin{cases} a \leftarrow not\ b. \\ b \leftarrow not\ a. \\ c \leftarrow a. \\ c \leftarrow b. \\ \leftarrow not\ a, c. \end{cases}$$

Déterminer les answer sets de Π_5 et Π_6.

Exercice 2 Détermination des answer sets - Automatisation

En utilisant l'algorithme de Smodels, déterminer les answer sets des programmes suivants :

$$P_1 \begin{cases} a \leftarrow b. \\ c \leftarrow b. \\ d \leftarrow a, c. \\ b. \end{cases} \qquad P_2 \begin{cases} a \leftarrow b. \\ c \leftarrow not\ b. \\ d \leftarrow not\ c. \end{cases} \qquad P_3 \begin{cases} p \leftarrow not\ q. \\ r \leftarrow p. \\ s \leftarrow not\ q, r. \end{cases}$$

$$P_4 \begin{cases} a \leftarrow not\ b. \\ c \leftarrow a. \\ b \leftarrow not\ c. \\ d \leftarrow b. \\ e \leftarrow not\ a. \end{cases} \qquad P_5 \begin{cases} p \leftarrow not\ p. \end{cases} \qquad P_6 \begin{cases} p \leftarrow not\ p, q. \\ r \leftarrow not\ q. \\ q \leftarrow not\ r. \end{cases}$$

Exercice 3 Négation explicite

Question 1. Interprétation

On considère les quatre programmes suivants :

$$\Pi_A \begin{cases} \text{innocent} \leftarrow \neg\text{coupable}. \\ \text{coupable} \leftarrow \textit{not}\ \text{innocent}. \end{cases} \qquad \Pi_B \begin{cases} \text{innocent} \leftarrow \textit{not}\ \text{coupable}. \\ \text{coupable} \leftarrow \textit{not}\ \text{innocent}. \end{cases}$$

$$\Pi_C \begin{cases} \text{innocent} \leftarrow \textit{not}\ \neg\text{innocent}. \\ \text{coupable} \leftarrow \neg\text{innocent}. \end{cases} \qquad \Pi_D \begin{cases} \text{innocent} \leftarrow \neg\text{coupable}. \\ \text{coupable} \leftarrow \neg\text{innocent}. \end{cases}$$

Donner les answer sets de ces programmes et discuter de leur sens et de leurs différences. Lesquels sont compatibles avec la présomption d'innocence ?

Question 2. On considère maintenant :

$$\Pi_E \begin{cases} p \leftarrow \textit{not}\ p, \textit{not}\ \neg q. \\ q \leftarrow \textit{not}\ r. \\ \neg q \leftarrow \textit{not}\ p. \\ r \leftarrow \textit{not}\ \neg q. \end{cases} \qquad \Pi_F \begin{cases} p \leftarrow \textit{not}\ q. \\ q \leftarrow \textit{not}\ p, \textit{not}\ \neg q. \\ \neg q \leftarrow \textit{not}\ r. \\ r \leftarrow \textit{not}\ \neg q. \\ \leftarrow r, \textit{not}\ p. \end{cases}$$

Déterminer les answer sets de ces deux programmes.

Exercice 4 Axiomes de choix et contraintes de cardinalité

Déterminer les answer sets des programmes suivants.

$$P_A \begin{cases} a \leftarrow \textit{not}\ b. \\ b \leftarrow \textit{not}\ a. \\ c \leftarrow \textit{not}\ \neg c. \\ \neg c \leftarrow \textit{not}\ c. \\ d \leftarrow \textit{not}\ a, \textit{not}\ c. \\ \leftarrow 2\{a, b, c, d\}. \end{cases} \qquad P_B \begin{cases} \{p, q\}. \\ r \leftarrow p, \textit{not}\ q. \\ q \leftarrow \textit{not}\ r. \end{cases} \qquad P_C \begin{cases} 1\{p, q\}2. \\ \{r, s\} \leftarrow \textit{not}\ p. \\ \leftarrow 3\{p, q, r, s\}. \\ \leftarrow \{p, q, r, s\}1. \end{cases}$$

Exercice 5 Instantiation

Instancier les programmes suivants et donner leurs answer sets :

1. ```
   c(1..2).
   a(X) :- not b(X),c(X).
   b(X) :- not q(X),c(X).
   ```
2. ```
   p(a;c).
   q(1..2).
   1{r(X,Y):q(Y)}1 :- p(X).
   ```
3. ```
   d(a;b,1..2).
   0{p(X,0..1)}1 :- d(X,2).
   ```

Exercice 6 Traduction de problèmes

Traduire les phrases suivantes par une ou plusieurs règles en ASP.

1. *Pour tout objet X verifiant $q(X)$, on choisit pour cet X un et un seul nombre entre 1 et 9.*

   Note : On considère ici qu'on a déjà défini des prédicats `q(`X`)` et que l'on veut retranscrire le choix demandé avec un prédicat `choix(`X`,`N`)` où N est le nombre choisi pour X.

2. *Chaque joueur a au plus un poste.*

   Note : On considère ici que la liste des joueurs est donnée par le prédicat `joueur(`X`)`, la liste des postes par `poste(`P`)` et que l'attribution d'un poste à un joueur se fait par `aPoste(`X`,`P`)`, non encore défini.
   ```

3. *Au cours d'une journée de championnat, une équipe peut jouer un match, mais pas plus, contre une des autres équipes du championnat.*

   Note : On utilise ici les prédicats `jour(`J`)` et `equipe(`E`)`, qui listent les jours et les équipes du championnat (déjà renseignés) et `match(`J`,`X`,`Y`)` qui indique qu'un match a lieu entre les équipes X et Y au jour J.

4. *Au cours du championnat, une équipe doit jouer au moins un match contre chacune des autres équipes du championnat.*

   Note : On utilise les même prédicats que pour la phrase précédente.

Annexe : rappel de définitions et algorithme Smodels

Programme logique

La programmation par ensembles-réponses (**ASP** - Answer Set Programming) est une évolution récente, et très active, de la programmation logique. Dans cette approche, un programme logique est vu comme un ensemble de propositions et les réponses à une requête sont extraites des modèles de cet ensemble de propositions. Un programme logique est constitué de règles qui se présentent sous la forme suivante :

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$$

A_0 est appelé la *tête* de la règle, et $A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$ est appelé son *corps*. Plus précisément, $A_1, \ldots, A_m$ est appelé le *corps positif* de la règle et $not\ A_{m+1}, \ldots, not\ A_n$ son *corps négatif*.

Si $n = 0$, c'est-à-dire si la règle a un corps vide (donc toujours vrai), $A_0 \leftarrow$ s'écrit 'A_0.'. On dit que c'est un *fait*. Si $m = n$, c'est-à-dire si la règle ne contient pas de corps négatif, on dit que la règle est *positive*. Sinon, on la dit *normale*. Si la tête est vide (toujours fausse), soit $A_0 = \bot$, la règle est une *contrainte d'intégrité*.

Un programme qui ne contient que des faits et/ou des règles positives est un *programme positif*. Sinon, on parle de *programme normal*.

Sémantique stable

Un modèle d'un programme P est un ensemble d'atomes S qui satisfait toutes ses règles. Un règle est satisfaite si sa tête est satisfaite par S ou si son corps n'est pas satisfait. Une tête A_0 est satisfaite par S ssi $A_0 \in S$. Un corps $A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$ est satisfait par S ssi son corps positif est dans S (i.e. $\{A_1, \ldots, A_m\} \subseteq S$) et que son corps négatif est disjoint de S (i.e. $S \cap \{A_{m+1}, \ldots, A_n\} = \emptyset$).

Le *modèle stable* d'un programme positif P est le modèle de P minimal pour l'inclusion. Il est unique.

La **réduction** Π^X d'un programme Π à un ensemble X d'atomes est obtenu en effectuant les étapes suivantes :

1. supprimer de Π toutes les règles contenant un "*not* A" dans leurs corps tel que $A \in X$.
2. supprimer tous les littéraux négatifs de la forme "*not* A" des règles restantes.

Un programme présentant des littéraux négatifs (négation par défaut *not*) comme Π_2, Π_3 ou Π_4 (voir exercice 3) peut posséder de 0 à un très grand nombre d'**ensembles-réponses** (*answer sets*), aussi appelés **modèles stables** (*stable models*).

L'un des principaux théorèmes de l'ASP établit que X est un modèle stable, ou *answer set*, de Π si X est également le modèle stable du programme positif correspondant Π^X.

Algorithme de Smodels

L'algorithme utilisé dans `smodels`, un des premiers solveurs ASP performants, est un algorithme de recherche arborescente d'une structure similaire à DPLL (Déduction, Analyse, Choix). L'algorithme

`clasp` qui sera utilisé en TME a un mécanisme de propagation similaire, mais se rapproche plus d'une structure CDCL (avec analyse des conflits et retour arrière non synchrone).

L'algorithme de `smodels` est toutefois plus facile pour rechercher "manuellement" les answer sets d'un programme. Il comporte trois phases.

On note Lit l'ensemble des littéraux positifs du programme P et $notLit$ l'ensemble de ses littéraux négatifs. Si P contient des contraintes d'intégrité, on les réécrit pour remplacer leur tête vide par $\bot$.

Initialisation

Modèle en cours $M := \emptyset$

Choix non confirmés $D := \emptyset$

Programme en cours $P' := P$

Ces variables évoluent au cours de l'algorithme.

1. **Déduction.** On répète la procédure suivante jusqu'à ce que M n'évolue plus, c'est-à-dire jusqu'à ce que M', qui représente les littéraux ajoutés à M à chaque étape, soit vide.
   (a) *Initialisation.* $M' := \emptyset$, $P' := P$.
   (b) *Propagation positive.* On ajoute au modèle en cours tous les atomes découlant de faits [1], soit $M' := \{a \in Lit \setminus M | 'a.' \in P'\}$.
   (c) *Propagation négative.* On ajoute au modèle en cours tous les littéraux négatifs dont les atomes ne sont plus présents dans aucune tête de règle de P',
   soit $M' := M' \cup \{not\ a \in notLit \setminus M | \forall r \in P', a \notin head(r)\}$.
   (d) *Mise à jour de P' par M'.* On ajoute M' au modèle courant ($M = M \cup M'$) et on modifie P' par suppression ou modifications de ses règles comme suit.
      i. Remplacement des littéraux. Pour chaque $l \in M' \setminus \{\bot\}$, on remplace chaque occurence de l par $\top$ et chaque occurence du complémentaire [2] de l par $\bot$.
      ii. Simplification des règles. Pour chaque règle r de P' :
         — Si la règle est satisfaite ($\top$ en tête ou $\bot$ dans le corps), on la retire de P'.
         — Sinon, on retire tous les (éventuels) $\top$ apparaissant dans le corps de r.
      iii. On met à jour le programme $P = P'$.
   (e) Si $M' \neq \emptyset$, retour en (a)
2. **Analyse.**
   — Si M est incohérent ($\bot \in M$ ou $\exists a \in Lit/\{a, not\ a\} \subseteq M \cup D$) ou contradictoire ($\exists a \in Lit/\{a, \neg a\} \subseteq M$), la branche ne produit aucun answer set. On backtracke (si possible) jusqu'au dernier point de décision (en remettant M, D et P' dans l'état où ils étaient au moment de la décision).
   — Si P' est devenu un programme positif,on a trouvé un answer set qui est égal à l'ensemble des littéraux positifs de M : on renvoie $M \cap Lit$. Si on cherche plus d'answer set, on backtracke au point de décision précédent.
   — Sinon, on doit procéder à une décision.
3. **Décision.** On choisit un littéral négatif $not\ a$ dans $notLit \setminus M$ et on décide de le supposer satisfait ou non en ajoutant respectivement dans D $not\ a$ ou a.
   — Si on rajoute $not\ a$ à D, on fait une mise à jour de P' par $\{not\ a\}$.
   — Si on rajoute a à D, on fait une mise à jour de P' par a mais **sans remplacer a par $\top$** (on se contente de remplacer $not\ a$ par $\bot$) [3].

   On reprend alors à l'étape de déduction avec ce nouveau P'.

1. Cette phase est similaire à une propagation unitaire, les faits étant de même nature que des clauses unitaires.
2. i.e. $not\ a$ si $L = a$ et a si $L = not\ a$.
3. Il s'agit en fait d'une mise à jour par $not\ not\ a$.

Annexe ASP 2 : Axiomes de choix et contraintes de cardinalité

Axiomes de choix

Un axiome de choix permet de choisir parmi un ensemble de littéraux positifs un certain nombre d'entre eux qui seront considérés comme des faits. Ainsi, toute règle contenant en tête $\{p, q, r\}$ peut se remplacer au choix par un ensemble X de règles où $X \subseteq \{p, q, r\}$. Cela crée autant de versions alternatives du programme contenant cette règle, et tout *answer set* de l'une des versions de ce programme est un *answer set* du programme avec l'axiome de choix.

Par exemple, le programme Π ci-dessous peut se traduire en 4 programmes alternatifs, selon que X vaut $\emptyset$, $\{p\}$, $\{q\}$ ou $\{p, q\}$. On a alors $AS(\Pi) = AS(\Pi_\emptyset) \cup AS(\Pi_{\{p\}}) \cup AS(\Pi_{\{q\}}) \cup AS(\Pi_{\{p,q\}}) = \{\emptyset\} \cup \{\{p\}\} \cup \{\{q, r\}\} \cup \emptyset = \{\emptyset, \{p\}, \{q, r\}\}$. On a donc 3 *answer sets*. Notez qu'il est ici possible, contrairement aux programmes normaux simples, d'avoir un *answer set* inclus dans un autre (e.g. $\emptyset \subset \{p\}$).

$$\Pi \left\{ \begin{array}{c} \{p, q\} \leftarrow not\ s. \\ r \leftarrow q. \\ s \leftarrow p, r. \end{array} \right. \qquad \Pi' \left\{ \begin{array}{c} p \leftarrow not\ \neg p, not\ s. \\ \neg p \leftarrow not\ p, not\ s. \\ q \leftarrow not\ \neg q, not\ s. \\ \neg q \leftarrow not\ q, not\ s. \\ r \leftarrow q. \\ s \leftarrow p, r. \end{array} \right.$$

$$\Pi_\emptyset \left\{ \begin{array}{c} r \leftarrow q. \\ s \leftarrow p, r. \end{array} \right. \qquad \Pi_{\{p\}} \left\{ \begin{array}{c} p \leftarrow not\ s. \\ r \leftarrow q. \\ s \leftarrow p, r. \end{array} \right. \qquad \Pi_{\{q\}} \left\{ \begin{array}{c} q \leftarrow not\ s. \\ r \leftarrow q. \\ s \leftarrow p, r. \end{array} \right. \qquad \Pi_{\{p,q\}} \left\{ \begin{array}{c} p \leftarrow not\ s. \\ q \leftarrow not\ s. \\ r \leftarrow q. \\ s \leftarrow p, r. \end{array} \right.$$

Alternativement, on peut remplacer l'axiome de choix par un couple de règles $a \leftarrow not\ \neg a, \ldots$ et $\neg a \leftarrow not\ a, \ldots$ pour chacun des littéraux de son ensemble. On projette alors les *answer sets* obtenus en éliminant les $\neg a$ (si d'autres $\neg a$ apparaissent dans le programme, on les renomme d'abord en na en ajoutant la contrainte d'intégrité $\leftarrow a, na.$. Ainsi pour Π on obtient le programme Π' (cf ci-dessus) qui a pour *answer set* $AS(\Pi') = \{\{p, \neg q\}, \{\neg p, q, r\}, \{\neg p, \neg q\}\}$, ce qui donne pour Π, par projection, $AS(\Pi) = \{\{p\}, \{q, r\}, \emptyset\}$ (qui correspond bien à la même réponse).

Contraintes de cardinalité

Il est aussi possible d'exprimer des contraintes de cardinalité. Ainsi dans le **corps** d'une règle $\{p, q, r\} \geq k$ est satisfait par X si et seulement si au moins k des atomes de l'ensemble $\{p, q, r\}$ sont présents dans X (soit si et seulement si $|X \cap \{p, q, r\}| \geq k$). On peut facilement adapter la définition du "réduc" selon X pour tester ces contraintes sur X et retirer la règle si la contrainte n'est pas satisfaite ou effacer la contrainte du corps sinon. Le forme générale d'une contrainte de cardinalité est $\{a_1, \ldots, a_n\}\ comp\ k$ où $comp \in \{=, \neq, <, \leq, >, \geq\}$ et $k \in \mathbb{N}$.

Considérons par exemple

$$P \left\{ \begin{array}{c} p \leftarrow not\ q. \\ q \leftarrow not\ p. \\ r \leftarrow q. \\ s \leftarrow \{p, q, r\} \geq 2. \end{array} \right.$$

Ce programme a pour *answer set* $\{\{p\}, \{q, r, s\}\}$. On donne par exemple les "réduc" pour $\{p\}, \{p, q\}, \{q, r, s\}$:

$$P^{\{p\}} \left\{ \begin{array}{c} p. \\ r \leftarrow q. \end{array} \right. \qquad P^{\{p,q\}} \left\{ \begin{array}{c} r \leftarrow q. \\ s. \end{array} \right. \qquad P^{\{q,r,s\}} \left\{ \begin{array}{c} q. \\ r \leftarrow q. \\ s. \end{array} \right.$$

On s'autorise aussi l'écriture inverse $k\ comp^{-1}\ \{a_1, \ldots, a_n\}$ (où $k\ comp^{-1}\ p$ si et seulement si $p\ comp\ k$) et la formule compacte $k_1\ comp_1^{-1}\ \{a_1, \ldots, a_n\}\ comp_2\ k_2$ (équivalente à la conjonction des deux contraintes). Dans l'écriture avec borne à gauche, on peut omettre $\geq^{-1}=\leq$ et dans l'écriture à droite on peut omettre $\leq$. Ainsi $k_1\{a_1, \ldots, a_n\}k_2$ signifie $k_1 \leq \{a_1, \ldots, a_n\} \leq k_2$.

Axiomes de choix contraints

Un littéral de la forme $k_1\{a_1,\ldots,a_n\}k_2$ en **tête** de règle, est en fait le cumul d'un axiome de choix et de deux contraintes de cardinalité prises comme contraintes d'intégrité. Ainsi, "k_1 $comp_1^{-1}$ $\{a_1,\ldots,a_n\}$ $comp_2$ k_2." est sémantiquement strictement équivalente aux deux règles suivantes : "$\{a_1,\ldots,a_n\}$." et "$\leftarrow$ *not* k_1 $comp_1^{-1}$ $\{a_1,\ldots,a_n\}$ $comp_2$ k_2.". Si la règle possède un corps, celui-ci est conservé. Ainsi "$1\{p,q\}1 \leftarrow r, not\ s$." est équivalent à "$\{p,q\} \leftarrow r, not\ s$." et "$\leftarrow not\ 1\{p,q\}1, r, not\ s$.".

Alternativement, on peut, comme pour les axiomes de choix, écrire plusieurs programmes alternatifs où "$k_1\{a_1,\ldots,a_n\}k_2$." est remplacé par un ensemble de règles pour chaque $X \subseteq \{a_1,\ldots,a_n\}$ tel que $k_1 \leq |X| \leq k_2$, mais, dans ce cas, on doit ajouter aussi systématiquement la contrainte "$\leftarrow not\ k_1\{a_1,\ldots,a_n\}k_2,\ldots$." (dispensable si aucune des a_i n'apparaît dans la tête d'une autre règle).

Adaptation de l'algorithme de Smodels

L'algorithme de `smodels` peut être employé pour des programmes contenant des axiomes de choix et/ou des contraintes de cardinalité. Si un atome apparaît en tête au sein d'un agrégat, on considère qu'il apparaît en tête d'une règle, et il n'est donc pas pris en compte par la propagation négative. En plus de cela, il suffit d'ajouter les traitements suivants aux mises à jour d'un programme entre les étapes (i) et (ii).

1. Mise à jour des contraintes de cardinalité. Après le remplacement des littéraux, on met à jour les contraintes dans lesquelles des $\bot$ ou des $\top$ ont été introduits comme suit.
   — On supprime tous les $\bot$ apparaissant à l'intérieur d'une contrainte de cardinalité sans en modifier les compteurs.
   — Si $\top$ apparaît dans une contrainte, on le retire et on soustrait 1 aux deux compteurs (à faire autant de fois qu'il y a de $\top$). Ainsi $k\{\top,p_1,\ldots,p_n\}k'$ est réécrit $(k-1)\{p_1,\ldots,p_n\}(k'-1)$.
2. Evaluation de la satisfaction des contraintes. Pour chaque contrainte de cardinalité $k_1\{p_1,\ldots,p_n\}k_2$ modifiée lors de la phase précédente, on fait les tests suivants :
   — Si $k_1 \leq 0$ et $k_2 \geq n$, la contrainte est nécessairement satisfaite : on la remplace par $\top$.
   — Si $k_1 > n$ ou $k_2 < 0$, la contrainte est insatisfiable : on la remplace par $\bot$
   Si ce processus fait apparaître un *not* $\top$ (respectivement *not* $\bot$), on le remplace par $\bot$ (resp. $\top$). On continue ensuite la mise à jour normalement par la simplification des règles.

Notation intensive d'un ensemble

Avant instanciation, plutôt que de représenter exhaustivement les agrégats, il est possible d'utiliser une notation plus compacte, permettant de définir par $\{p(X) : q(X)\}$ l'ensemble de tous les $p(X)$ tels que $q(X)$ appartient à l'*answer set*. Ainsi si on a dans le programme les faits $q(1)$., $q(3)$. et $q(5)$., $\{p(X) : q(X)\}$ est instancié en $\{p(1), p(3), p(5)\}$. Si on a deux alternatives, on utilise un séparateur entre chacune ; de même, s'il y a plusieurs conditions.

Ces séparateurs diffèrent entre les versions 3- et 4+ de `clingo`. On donne ici les deux versions avec un exemple pour représenter tous les $p(X,Y)$ tels que $q(X)$ et $r(X,Y)$ mis avec tous les $s(Z,Z)$ tels que $q(Z)$ et $Z > 2$.

— en `clingo 3`, cela s'écrit $\{p(X,Y) : q(X) : r(X,Y), s(Z,Z) : q(Z) : Z > 2\}$ (conditions supplémentaires avec ' :' et séparateurs entre deux expressions avec ',').
— en `clingo 4`, cela s'écrit $\{p(X,Y) : q(X), r(X,Y); s(Z,Z) : q(Z), Z > 2\}$ (un seul séparateur en ' :', les conditions suivantes sont séparées par des ',', et le séparateur entre deux expressions est alors ';'). Dans les deux cas, si on a $q(1)$., $q(3)$., $q(5)$., $r(5,2)$., $r(3,2)$., $r(2,4)$. et $r(3,3)$., cet ensemble est instancié en $\{p(3,2), p(3,3), p(5,2), s(3,3), s(5,5)\}$.