ASP : Answer Set Programming

Carito Guziolowski
Département MATH-INFO, LS2N, École Centrale de Nantes

24 février ou 3 mars 2022

TP1 (Partie I) ASP

Il est conseillé de travailler avec clingo 4.5.4 dans tous les TPs, disponible dans :

https://sourceforge.net/progects/potassco/files/clingo/4.5.4/

clingo est disponible pour Windows, Linux et Mac OS.
Vous pouvez également télécharger le solveur clasp et le grounder gringo. clingo combine les deux outils.

Commandes `gringo`, `clasp` et `clingo`

L'outil `gringo` est un *grounder* (instantiateur), qui traduit des programmes logiques (PL) avec variables, écrits par l'utilisateur, en programmes logiques avec constantes (grounded) équivalents. Les ensembles de réponses (answer sets) des programmes instantiés avec `ground` peuvent être générés avec l'outil `clasp`, qui est le *solver*. `gringo` gère certaines fonctions arithmétiques qui sont évaluées dans l'instantiation. Un troisième outil `clingo`, rassemble les fonctionalités de `gringo` et `clasp`. Voici quelques instructions utiles pour les TPs :

- `gringo -t prog.lp`, génère le PL *grounded* (sans variables) du PL `prog.lp`.
- `gringo -t prog1.lp prog2.lp`, génère le PL *grounded* des programmes concaténés `prog1.lp` et `prog2.lp`.
- `gringo -t prog.lp --const n=5`, génère le PL *grounded* du PL `prog.lp` en remplaçant la constante n prédéfinie dans le PL `prog.lp` avec la valeur 5.
- `gringo prog.lp | clasp`, va résoudre le PL `prog.lp` et afficher le premier *answer set*.
- `gringo prog.lp | clasp -n 0`, permet d'afficher tous les *answer sets*. Vous pouvez restreindre le numéro de solutions en choisisant différents arguments pour n.
- Le symbole `%` permet de commenter les lignes d'un PL.
- Pour filtrer la sortie des modèles stables (answer sets), par exemple, montrer seulement les prédicats de type `couleur(C,M)` dans la liste des *answer sets*, rajouter à la fin du programme logique la ligne suivante :

```
#show couleur/2.
```

1 Introduction

Supposons qu'on nous donne des informations sur les températures d'aujourd'hui dans plusieurs villes, par exemple :

Ville	Rennes	Nantes	Marseille	Paris
Température	4°	5°	8°	2°

Table 1: Villes et leur températures.

Nous allons utiliser ASP pour trouver toutes les villes où la température est plus élevée qu'à Rennes. Appelons ces villes "chaudes". Les programmes ASP sont constitués de règles. Pour trouver les villes chaudes, nous avons besoin de la règle :

Listing 1: Règle pour trouver les villes plus chaudes que Rennes

```
1 chaude(V) :- t(V,T1), t(rennes,T2), T1>T2.
```

Cette règle est composée de deux parties. Une tête :

`chaude(V)`

et un corps :

`t(C,T1), t(rennes,T2), T1>T2`

Ces parties sont séparées par le symbole `:-` qui ressemble à la flèche ← et se lit "si". La fin d'une règle est indiquée par un point. Les identifiants en majuscules dans une règle (dans ce cas, V, T1 et T2) sont des variables. `rennes` est une constante du programme. Les variables et constantes peuvent être collectivement appelés *termes*. Le symbole `chaude` au début de la tête indique la propriété d'être une ville chaude; il est un prédicat. Le symbole `t` dans le corps est aussi un prédicat; il exprime une relation entre une ville et une température. Les expressions consistant par un prédicat suivi d'une liste de termes entre parenthèses, tels que `chaude(V)`, `t(V, T1)` et `t(austin, T2)`, sont appelés *atomes*. L'expression `T1>T2` à la fin du corps est une comparaison. Les symboles pouvant être utilisés dans les comparaisons sont :

`=  !=  <  >  <=  >=`

La tête d'une règle est généralement un atome, et le corps d'une règle est souvent une liste d'atomes et comparaisons, comme dans l'exemple ci-dessus. Les quatre derniers symboles parmi les relations présentées ci-dessus sont généralement appliqués aux nombres, mais `clingo` permet aussi de les appliquer aux constantes symboliques. Cela arrive que selon l'ordre utilisé par `clingo` pour de telles comparaisons, le symbole `rennes` est plus grand que le symbole `nantes` et `nantes` est supérieur à 3. La règle du Listing 1 peut être lue comme suit :

V est une ville chaude si
la température de V est T1,
la température de rennes est T2,
et $T1 > T2$.

Notez que cette phrase est déclarative : elle ne décrit pas un algorithme. Elle simplement explique comment nous comprenons "chaude". Le tableau 1 peut être représenté par les règles

Listing 2: Villes et leur température

```
1 t(rennes,4). t(nantes,5). t(marseille,8). t(paris,2).
```

Ces règles n'ont pas de corps, et par conséquent, il n'y a pas de symbole "si" dans celles-ci. Ils sont également *grounded*, c'est-à-dire ne contiennent pas de variables. Les règles de base sans le corps sont appelés des faits. Pour ordonner à `clingo` de trouver toutes les villes chaudes, nous créons un fichier texte, disons `villechaude.lp`, contenant les lignes dans les Listing (1) et (2). Si nous exécutons la commande

```
% clingo villechaude.lp
```

Alors `clingo` produira une collection d'atomes *grounded* :

```
1 t(rennes,4) t(nantes,5) t(marseille,8) t(paris,2) chaude(marseille)
2 chaude(nantes)
```

Cet ensemble d'atomes *grounded* est le modèle stable du programme `villechaude.lp`. De manière informelle, nous pouvons dire que le modèle stable se compose des atomes *grounded* qui peuvent être "dérivés" en utilisant les règles du programme. Pour voir, par exemple, pourquoi l'atome `chaude(nantes)` est inclus dans le modèle stable de `villechaude.lp`, considérez l'instance :

```
1 chaude(nantes) :- t(nantes,5), t(rennes,4), 5>4.
```

de la règle dans le Listing (1) qui en est obtenue en substituant les valeurs nantes, 5, et 4 pour les variables V, T1 et T2. Les deux atomes dans le corps de cette instance sont parmi les faits donnés (Listing 2), et la comparaison $5 > 4$ est vraie. Par conséquent, cette instance justifie d'inclure sa tête `chaude(nantes)` dans le modèle stable de `villechaude.lp`.

Exercice 1.1. Quelle instance de la règle dans le Listing 1 justifie d'inclure `chaude(marseille)` dans le modèle stable ?

♢

Les deux derniers atomes du modèle stable généré par `clingo` répondent à notre question : les villes chaudes sont Marseille et Nantes. Nous pouvons souhaiter supprimer le reste de la sortie, qui reproduit simplement les faits donnés. Cela peut être accompli en incluant les instruction suivantes dans notre programme `villechaude.lp` :

```
#show chaude/1.
```

Lorsque nous nous référons au symbole d'un prédicat utilisé dans un programme ASP, nous ajoutons son arité - le nombre d'arguments - après une barre oblique; dans ce cas, l'arité est de 1. Cette directive ordonne à `clingo` de montrer les atomes formés à partir du symbole de prédicat chaud et un argument, au lieu de l'ensemble du modèle *grounded*.

Exercice 1.2. Au lieu de comparer avec la température de Rennes, définissons "chaude" par la condition "la température est supérieure à 4°". Modifier le fichier `villechaude.lp` en conséquence.

♦

En cas d'erreur de syntaxe, `clingo` produit généralement un message spécifiant la place dans le fichier où l'analyse a cessé. Certains messages d'erreur disent que le programme a des "unsafe variables". Un tel message indique généralement que la tête de l'une des règles inclue une variable qui n'apparaît pas dans son corps ou que le corps négatif de la règle contienne une variable que n'apparaisse pas dans le corps positif; les modèles stables de tels programmes peuvent être infinis.

Exercice 1.3. Considérez la règle :

Listing 3: Parent - Enfant

```
1 enfant(Y,X) :- parent(X,Y).
```

(a) Comment vous lissiez cette règle ? (b) Si nous exécutons `clingo` dans le programme en Listing 3 et les faits :

Listing 4: Faits Parents-Enfants

```
1 parent(lucie,robert). parent(robert,mattis). parent(robert,anna).
```

Quel est le modèle stable produit ?

♦

Un groupe de faits contenant le même symbole de prédicat peut être "regroupé" à l'aide de points-virgules. Par exemple, le Listing 2 peut être raccourci en :

```
1 t(rennes,4; nantes,5; marseille,8; paris,2).
```

Exercice 1.4. Raccourcir de cette façon la règle dans le Listing 4

♦

Exercice 1.5. Si vous exécutez `clingo` dans le programme :

```
1 p(1,2; 2,4; 4,8; 8,16).
```

Quel est le modèle stable résultant ?

♦

2 Arithmétique

La règle

Listing 5: Polynôme

```
1 p(N,N*N+N+41) :- N=0..10.
```

se lit :

N et $N^2 + N + 41$ sont dans une relation `p/2` si
N est un entier entre $0, \ldots, 10$.

Le modèle stable de cette programme montre les valeurs pour le polynôme $N^2 + N + 41$ pour tout N de 0 à 10 :

```
1 p(0,41) p(1,43) p(2,47) p(3,53) p(4,61) p(5,71) p(6,83) p(7,97)
2 p(8,113) p(9,131) p(10,151)
```

Exercice 2.1. Pour chaque programme suivant, prédire quel est le modèle stable produit par `clingo`

```
1 p(N,N*N+N+41) :- N+1=1..11.
```

```
1 p(N,N*N+N+41) :- N=-10..10, N>=0.
```

♢

`clingo` seulement connait des entiers. Le modèle stable du programme suivant :

```
1 p(M,N) :- N=1..4, N=2*M.
```

aura seulement les atomes :

```
1 p(1,2) p(2,4)
```

parce que les entiers 1 et 3 ne peuvent pas être représentes dans la forme $2 * M$ où M est un entier. En plus des symboles $+$ et $*$, les termes d'un programme en ASP peuvent inclure les symboles :

```
**   /     \     | |
```

pour exponentiation, division entière, reste, et valeur absolue.

Exercice 2.2. Écrire un programme qui ne contient pas des regroupements et qui produise le même modèle stable du programme de l'exercice 1.5.

♢

Exercice 2.3. Pour chaque ensemble *grounded* d'atomes, écrire une programme en ASP avec une seule règle (sans regroupement) et qui produise le même ensemble.

```
p(0,1) p(1,-1) p(2,1) p(3,-1) p(4,1)
```

```
p(1,1)
p(2,1) p(2,2)
p(3,1) p(3,2) p(3,3)
p(4,1) p(4,2) p(4,3) p(4,4)
```

♢

Les intervalles peuvent être utilisés non seulement dans les règles, comme dans les exemples ci-dessus, mais dans les têtes aussi. Le modèle stable de cette programme :

```
square(1..8,1..8).
```

consiste de 64 atomes qui correspondent aux carrés d'un échiquier de taille 8×8

```
square(1,1) ... square(1,8)
. . . . . . . . . . . . . .
square(8,1) ··· square(8,8)
```

Exercice 2.4. Considérez le programme qui consiste par les 2 faits suivants :

```
p(1..2,1..4).  p(1..4,1..2).
```

Combien d'atomes aura le modèle stable ?

♢

La règle décrite dans le Listing 5 qui décrit les valeurs du polynôme $N^2 + N + 41$ pour tout N allant de 0 à 10, peut être écrite de façon plus générale :

Listing 6: quadratic.lp

```
p(N,a*N*N+b*N+c) :- N=0..n.
```

La règle précédente utilise les constantes a, b, c, n pour décrire la table de valeurs d'un polynôme arbitraire $aN^2 + bN + c$ avec des coefficients entiers pour les valeurs de N allant de 0 à une borne supérieure arbitraire n. Les valeurs des constantes peuvent être spécifiés avec la ligne de commande:

```
$ clingo -c a=1 -c b=1 -c c=41 -c n=10 quadratic.lp
```

Alternativement, les valeurs des constantes pourront être ajoutés au programme `quadratic.lp` de cette façon :

```
#const a=1. #const b=1. #const c=41. #const n=10.
```

Nous pourrons écrire la règle du Listing 1 de façon plus générale :

```
chaude(C) :- t(C,T1), t(c,T2), T1>T2.
```

Exercice 2.5. Écrire une instruction pour spécifier que c est `rennes`.

♢

3 Définitions

De nombreuses règles en ASP peuvent être considérées comme des définitions. Nous pouvons dire, par exemple, que la règle dans le Listing 1 définit le prédicat `chaude/1` en termes du prédicat `t/2`.

Exercice 3.1. (a) Comment pourriez vous définir le prédicat `grandparent/2` en termes du prédicat `parent/2` ? (b) Si vous exécutez `clingo` dans cette définition combiné avec les faits du Listing 4, quel modèle stable obtiendrez vous ?

♢

Exercice 3.2. (a) Comment pourriez vous définir le prédicat `sibling/2` (enfants ayant les mêmes parents) en terme du prédicat `parent/2` ? (b) Si vous exécutez `clingo` dans votre définition combinée avec les faits du Listing 4, quel modèle stable obtiendrez vous ?

♢

Parfois, la définition d'un prédicat comprend plusieurs règles. Par exemple, le pair de règles suivant :

```
1 parent(X,Y) :- pere(X,Y).
2 parent(X,Y) :- mere(X,Y).
```

défini le prédicat `parent/2` à partir des prédicats `pere/2` et `mere/2`. Un prédicat peut être défini de manière récursive. Dans une définition récursive, le prédicat défini apparaisse non seulement dans les têtes des règles mais aussi dans le corps. La définition de `ancetre/2` en termes de `parent/2` est un exemple :

Listing 7: Définition d'ancêtre

```
1 ancetre(X,Y) :- parent(X,Y).
2 ancetre(X,Z) :- ancetre(X,Y), ancetre(Y,Z).
```

Exercice 3.3. Si vous exécutez `clingo` dans le programme avec les règles dans le Listing 7 et 4, quel modèle stable obtiendrez vous ?

♢

Parfois, un prédicat ne peut pas être défini en une seule étape et des prédicats auxiliaires doivent être introduits en premier. Considérons, par exemple, la propriété d'être un nombre premier de l'ensemble $\{1, ..., n\}$. Il est plus facile de définir la propriété opposée : d'être un nombre composé de cet ensemble:

Listing 8: Nombre composé

```
1 compose(N) :- N=1..n, I=2..N-1, N\I = 0.
```

(N est composé s'il s'agit d'un multiple d'un nombre I de l'ensemble $\{2, ..., N-1\}$.) `prime/1` peut être défini en termes de `compose/1` par la règle :

Listing 9: Nombre prime

```
1 prime(N) :- N=2..n, not compose(N).
```

(N est un nombre prime s'il est diffèrent de 1 et n'est pas un nombre composé.) La règle précédente est un exemple d'utilisation de la négation dans un programme ASP. Dans quel sens cette règle peut-elle utilisée pour "dériver" l'atome `prime(7)` (en supposant, par exemple, que n = 10) ? A ce propos, la règle :

Listing 10: 7 est un nombre prime

```
1 prime(7) :- 7=2..10, not compose(7).
```

exprime que l'expression `not compose(7)` dans son corps est justifiée dans le sens que toute tentative d'utiliser la règle dans le Listing 8 pour "dériver" `compose(7)` va échouer. L'idée de la négation comme échec joue un rôle important dans le ASP. Pour souligner cette compréhension, nous pouvons lire la règle dans le Listing 9 comme suit :

N est prime si
est un nombre entre 2 et n
et il n'y a pas une évidence que soit un nombre composé.

Listing 10: PL Nombre prime

```
1  % Prime numbers from 1 to n
2  % input : positive integer n
3
4  composite(N) :- N=1..n, I=2..N-1, N\I = 0.
5  % achieved: composite(N) iff N is a composite number from {1, ..., n}
6
7  prime(N) :- N=2..n, not composite(N).
8
9  % achieved : prime(N) iff N is a prime number from {1, ..., n}
10 # show prime/1
```

Le Listing 10 montre un programme constitué des règles dans les Listings 8 et 9, une directive #show, et quelques commentaires. n sera une constante du programme, un espace réservé pour un nombre entier positif; la valeur de n doit être spécifiée lors de l'éxécution du programme. La directive #show sert à afficher à l'écran seulement les prédicats de type prime/1 qui appartient aux Answer Sets du programme.