PRLOG : Programmation Logique

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

2 février 2023

TD1 : ASP

Notions:

- Logique de Prédicats, Modèle Herbrand, ground
- Syntaxe en ASP
- Écriture de Programmes Logiques en ASP

1 Programme Logic : ground

1.1 Question 1

Étant donné le programme logic π_1, écrire : (1) l'Universe de Herbrand pour ce programme $\forall^h_{\pi_1}$, (2) la Base de Herbrand B_{π_1}, et (3) le programme *grounded*, i.e. ground(π_1).

Programme Logic π_1

```
fly(X) :- bird(X), not ab(X)
bird(X) :- penguin(X)
ab(X) :- penguin(X)
penguin(X).
```

1.2 Question 2

Étant donné le programme logic π_2, écrire : (1) l'Universe de Herbrand pour ce programme $\forall^h_{\pi_2}$, (2) la Base de Herbrand B_{π_2}, et (3) le programme *grounded*, i.e. ground(π_2).

Programme Logic π_2

```
a :- not b.
b :- not a.
```

1.3 Question 3

Étant donnée le programme logic π_3, ecrire : (1) l'Universe de Herbrand pour ce programme $\forall^h_{\pi_3}$, (2) un ébauche de la Base de Herbrand B_{π_3}, sa cardinalité et (3) trois interprétations Herbrand possibles.

Programme Logic π_3

```
p(X, Y, Z) :- p(X, Y, U), h(X, Y), t(Z, U, r).
h(X, U) :- p(X, Y, U), h(X, Y), t(Z, U, r).
p(0, 0, b).
h(0, 0).
t(a, b, r).
```

2 Syntaxe ASP : choix, conditions

Dans le PL qui suit, décrire quelle forme auront les ensembles de prédicats que nous obtiendrons après exécuter ce PL avec clasp 4.5.4 ?

```
q(1..5).
t(11..15).
2{r(U):q(U); s(V):t(V)}4.
```

3 Syntaxe ASP : Integrity constraint

Les *Integrity constraints* permettent d'interdire l'apparition de certains types d'atomes dans l'ensemble de modèles stables. Par exemple, la contrainte :

```
:- p(X),q(X)
```

va interdire l'apparition des atomes $p(X)$ ET $q(X)$ simultanément dans l'ensemble de réponses E. En d'autre mots, l'absence de ces atomes dans une interprétation (candidate) implique qu'elle sera vraie et alors elle fera partie des modèles en E. Par conséquence, l'expression :

```
:- not p(X)
```

signifie qu'on ne peut pas générer un modèle dans lequel l'atome $p(X)$ ne soit pas présente. Dans le PL qui suit, quels sont les ensembles de prédicats `r/1` que nous obtiendrons à l'exécution de ce PL avec clasp 4.5.4 ?

```
q(1..10).
{r(U+1):q(U)}.
:- not {r(U)}5.
```

4 Écriture de Programmes Logiques en ASP : variables

Un programme logique sans variables ne fera pas appel à l'instatiateur de variables (*gringo*). Comment le programme *Who killed Mr Boody?* (ci en bàs) peut être amélioré pour qu'il contienne des variables ?

```
% Specifier qu'il a un seul assasin, avec une seule arme et dans un seul endroit
1{murderer(ms_Scarlet),murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver),weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen),place_of_crime(hall),place_of_crime(dining-room)}1.
% Déclarer ce qu'on déduit de nos cartes
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard), not weapon_of_crime(revolver).
weapon_of_crime(candlestick).
```

5 Écriture de Programmes Logiques en ASP : optimisation

On dispose d'un prédicat `ville/1` pour stocker les noms de différents villes, d'un prédicat `villep/2` pour associer à chaque ville son nombre d'habitants, et d'un prédicat `metro/1` pour indiquer qu'une ville dispose d'un métro. Ecrire un programme en ASP qui permettra trouver la ville qui a le plus grand nombre d'habitants et qui n'a pas de métro. Vous pouviez choisir les noms de prédicats et constantes qui vous conviennent ou vous inspirer de ces données fictifs :

```
1 ville(brest;rennes;nantes;paris;lille).
2 villep(brest,140). villep(rennes,200). villep(nantes,300).
3 villep(lille,230). villep(paris,2300).
4 metro(rennes;paris;lille).
```

6 Écriture de Programmes Logiques en ASP : graphes

Étant donné une instance d'un problème (déclaration d'atomes à utiliser dans les règles), par exemple :

```
maison(maison1, bleu).  maison(maison2, rouge).
maison(maison3, vert).  maison(maison4, rouge).
```

nous pouvons *extraire* un sous-ensemble d'information en utilisant une règle de type:

```
couleur(X) :- maison(_,X).
```

Cette règle nous permettra d'obtenir tous les couleurs (sans répétitions) utilisés dans la déclaration du problème.

Exercice : Le temps du trajet qui prend un taxi pour récuperer un client est donnée par la déclaration du problème suivante :

```
cout(taxi1,client1,10).  cout(taxi2,client1,8).  cout(taxi3,client1,12).
cout(taxi1,client2,11).
cout(taxi2,client2,15).  cout(taxi3,client2,13).  cout(taxi1,client3,7).
cout(taxi2,client3,7).  cout(taxi3,client3,10).
```

6.1 Question 1

Comment idéntifier les taxis et les clients ?

6.2 Question 2

Comment générer le prédicat `sol(T, C)` pour qu'une solution donnée contienne un taxi par client et un client par taxi ? C'est à dire si nous construisons un graphe avec 6 noeuds (3 taxis et 3 clients), si $sol(T, C)$ crée des arêtes dans le graphe, comment générer ces arêtes de façon à avoir seulement une sortie et une entrée par noeud ?

6.3 Question 3

Comment minimiser le temps de trajet total des 3 taxis, i.e. quelle règle doit être rajoutée dans le programme pour générer seulement des prédicats `sol(T,C)` qui minimisent le temps du trajet total ?
Revenant sur la répresentation du probème par des graphes. Nous avions proposé dans la question précedente une représentation où les arêtes du graphe sont construits avec les prédicats $sol(T, C)$. Maintenant, le poids des arêtes du graphe représentera le temps du trajet donnée par le prédicat *cout*. Comment obtenir le graphe qui a un poids (de ses arrêtes) minimale ?