Théorie des langages et compilation

Projet : Traduction Prolog –> GraphViz

Sujet

Les parseurs (analayseurs syntaxiques) peuvent être utilisés pour transformer un langage en une structure de données ou encore un autre langage. L'avantage de Lex et Yacc combinés est la simplicité à traiter une entrée afin de générer la sortie souhaitée. Le but de ce projet est de transformer un programme logique en une modélisation sous forme de graphe afin de pouvoir visualiser les relations entre les règles du programme. Pour cela nous utiliserons deux langages connus Prolog (du moins une version simplifiée de celui-ci) pour l'entrée ainsi que GraphViz (sans prendre en compte l'aspect visuel de celui-ci) pour la sortie. Tout cela sera géré par une structure de données intermédiaire nous permettant de choisir l'entrée et la sortie de notre programme afin de permettre une extension des langages supportés par notre parseur.

Prolog

Prolog est l'un des principaux langages de programmation logique. Le nom Prolog est un acronyme de PROgrammation en LOGique. Il est reconnu en intelligence artificielle pour sa simplicité syntaxique. Ce langage tient ses racines de la logique du premier ordre ainsi que la logique formelle, c'est donc un langage de programmation déclaratif dans lequel l'information est représentée à l'aide de faits et de règles. Nous cherchons ici à ne représenter qu'une petite partie de ce langage dans laquelle nous gèrerons les aspects lexicaux suivants :

- Les constantes : chaines de caractères commençant par une lettre minuscule et pouvant contenir des chiffres et des lettres
- Les variables : chaines de caractères commençant par une lettre majuscule suivis de chiffres et de lettres
- Les constantes numériques : nombres entiers
- Les chaines de caractères : chaines de caractères contenues entre guillemets (considérées comme des constantes)
- Les séparateurs : l'ensemble des séparateurs utilisés tels que ',' pour la conjonction, ' :-' pour la séparation entre tête et corps d'une règle, et '.' pour terminer une règle
- Les opérateurs : l'ensemble des opérateurs binaires classiques tels que (+, -, *, /, =) ainsi que les comparateurs (<, <=, ==, >=, >, !=)
- Les commentaires : précédés par le symbole % ou compris entre /* et */

Ainsi que les aspects syntaxiques suivants :

- Les arguments : ils peuvent être soit des constantes, des constantes numériques, des variables ou des chaînes de caractères
- Les prédicats : sont aussi des constantes (commençant par une lettre minuscule et pouvant contenir des chiffres et des lettres)
- Les atomes : constitués d'une constante seule pour un prédicat d'arité 0 ou suivi d'arguments entre parenthèses (exemple : p(X,50,a) pour un prédicat p d'arité 3)
- Les opérations : peuvent-être une variable, une constante numérique ou une opération entre deux éléments
- Les comparaisons : s'effectuent entre deux opérations
- Les faits : ils sont formés d'un simple atome
- Les règles : elles sont constituées d'une tête suivi du symbole ' :-' et d'un corps, la tête étant un simple atome et le corps une conjonction d'atomes, d'opérations, et de coomparaisons
- Les contraintes : Ce sont des règles spéciales constituées uniquement d'un corps on les représentera en les préfixant du symbole ' :-'

Petit exemple de programme devant être identifié par le parseur :
$famille(1).$
$femme(titi).$
$homme("toto").$
$parent(X,Y) :- enfant(Y,X).$
$pere(X) :- parent(X,Y),\ homme(X).$
$famille(Y) :- nbfamille(Z),\ famille(X),\ Y = X+1,\ Y <= Z.$
$\%Il\ est\ posible\ de\ melanger\ des\ variables\ avec\ des\ constantes\ pour\ un\ meme\ predicat\ comme\ par\ exemple\ :$
$:- marie(X,titi),\ femme(X).$

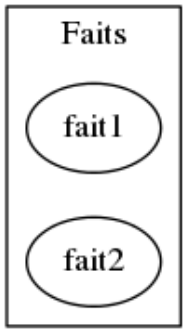
GraphViz

GraphViz est un langage de représentation visuelle pour les graphes basé sur la notation DOT. Celui-ci permet de manière simple et formelle de représenter des graphes avec des effets visuels à l'aide d'un simple fichier textuel. Ce langage est utilisé notamment dans la représentation de graphes dans les documentations Doxygen. On veut ici pouvoir représenter un programme logique à partir de graphes en représentant l'ensemble des dépendances entre prédicats ainsi que la représentation des faits, des contraintes et les informations sur les constantes et les variables contenues dans le programme. On devra donc représenter l'ensemble des informations sur le programme de la manière suivante. On utilisera tout d'abord un graphe orienté de gauche à droite en utilisant l'option :

$digraph\{$
$rankdir = LR;$
$//definitions\ du\ graphe$
$\}$

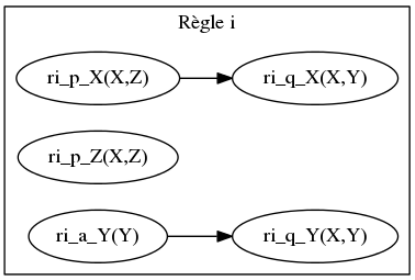
Puis on séparera le graphe en deux parties en commençant par les faits puis les règles et contraintes. Pour cela, on utilisera une syntaxe de sous-graphe par exemple pour les faits :

$subgraph\ cluster_0\{$
$label = "Faits";$
$fait1;$
$fait2;$
$\}$



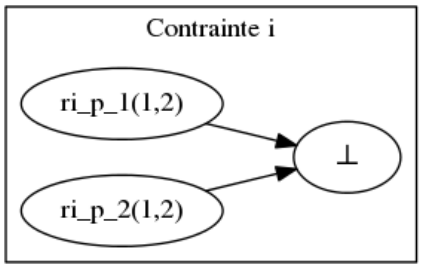
Une règle est elle composée d'un corps et d'une tête, on créera un sous graphe par règle. Chaque prédicat d'une règle sera préfixé par ri_ avec i le numéro de la règle et suffixé par l'argument considéré _X puis la liste de ses arguments (X,Y). On ajoutera aussi pour chaque prédicat du corps une flèche vers un prédicat contenant la même variable. Les prédicats n'ayant pas de liaison avec d'autres seront représentés par de simples noeuds.

$subgraph\ cluster_i\{$
$label = "Regle\ i";$
$"ri\_p\_X(X,Z)" -> "ri\_q\_X(X,Y)"$
$"ri\_p\_Z(X,Z)"$
$"ri\_a\_Y(Y)" -> "ri\_q\_Y(X,Y)"$
$\}$



et pour terminer les contraintes seront des règles dont tous les atomes mènent à $\perp$

$subgraph\ cluster_i\{$
$label = "Contrainte\ i";$
$ri\_p\_1(1,2) -> \perp;$
$ri\_p\_2(1,2) -> \perp;$
$\}$



Compilation d'un programme GraphViz à l'aide de DOT, permettant de créer une image :

dot -Tpng filename.dot -o outfile.png

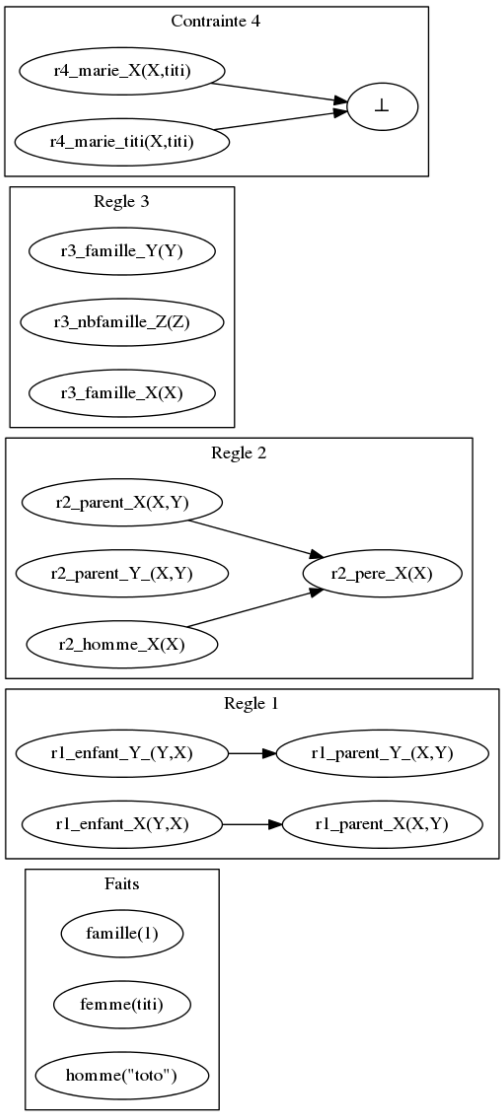
Le programme

Pour terminer, le programme permettera le passage d'un programme ProLog en entrée à la réécriture de celui-ci en GraphViz en sortie, pour cela on stockera l'ensemble des informations qui nous intéresse dans une structure de données adaptée. Le travail sera noté sur 3 aspects l'analyse syntaxique du programme en entrée, le stockage dans la structure de données et la réutilisation de celle-ci pour restranscrire le programme dans le nouveau format. Attention : On ne demande pas dans ce projet de calculer la réponse à un programme mais simplement d'avoir une structure adaptée à la fois à la réécriture mais qui pourrait aussi servir pour rediriger la lecture vers un solveur ou encore un algorithme sur les graphes.

Exemple complet

$famille(1).$
$femme(titi).$
$homme("toto").$
$parent(X,Y) :- enfant(Y,X).$
$pere(X) :- parent(X,Y),\ homme(X).$
$famille(Y) :- nbfamille(Z),\ famille(X),\ Y = X+1,\ Y <= Z.$
$\%Il\ est\ posible\ de\ melanger\ des\ variables\ avec\ des\ constantes\ pour\ un\ meme\ predicat\ comme\ par\ exemple\ :$
$:- marie(X,titi),\ femme(X).$

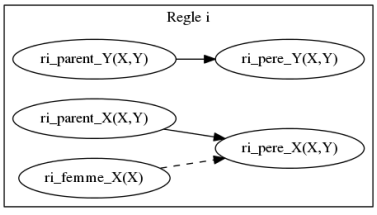
```
digraph{
rankdir = LR;
subgraph cluster_0{
label = "Faits";
"famille(1)";
"femme(titi)";
"homme(\"toto\")";
}
subgraph cluster_1{
label = "Regle 1";
"r1_enfant_Y(Y,X)" -> "r1_parent_Y(X,Y)";
"r1_enfant_X(Y,X)" -> "r1_parent_X(X,Y)";
}
subgraph cluster_2{
label = "Regle 2";
"r2_parent_X(X,Y)" -> "r2_pere_X(X)";
"r2_parent_Y(X,Y)";
"r2_homme_X(X)" -> "r2_pere_X(X)";
}
subgraph cluster_3{
label = "Regle 3";
"r3_famille_Y(Y)";
"r3_nbfamille_Z(Z)";
"r3_famille_X(X)";
}
subgraph cluster_4{
label = "Contrainte 4";
"r4_marie_X(X,titi)" -> ⊥;
"r4_marie_titi(X,titi)" -> ⊥;
}
}
```



Bonus

Ajouter la négation par défaut notée "not" qui peut s'appliquer aux atomes du corps d'une règle. On distinguera alors le corps positif du corps négatif ce qui ajoutera un nouveau type d'arc sur notre graphe représenté par une flèche pointillée.

$pere(X,Y) \ :- \ parent(X,Y), \ not \ femme(X).$



Ajouter un lien entre la tête d'une règle et le corps d'une autre lorsqu'il existe une dépendance entre prédicats. Pour cela il sera nécessaire de calculer les dépendances entre chaque règles.

$q(X,Y) \ :- \ p(X,Y).$
$r(X,Y) \ :- \ q(Y,X).$