Rapport Compilation

Sarra BOUTAHLIL
Alexis DONNART

Avril 2017

Résumé

Dans le cadre du cours de compilation dispensé à Nantes par M. Oussalah en 2017, il nous a été demandé de réaliser une brève présentation des outils d'analyse lex et yacc.

Table des matières

1 Introduction

Les outils *lex* et *yacc* (et leur équivalent *flex* et *bison*) sont des outils d'analyse lexical et syntaxique. Ces deux outils fonctionnent en collaboration afin d'effectuer l'analyse d'une grammaire, permettant la base de l'écriture d'un compilateur. Rapide présentation de ces deux outils.

2 Lex

Ce premier outil, écrit en C, permet la spécification d'analyseurs lexicaux. Plus spécifiquement, un programme écrit en lex pourra être converti en programme C proposant un analyseur lexical suivant le contenu du fichier.
Ce programme défini un ensemble de règles qui seront appliqués à un texte d'entrée, chaque règle correspond à une expression régulière qui sera testé contre le texte d'entrée. Si la règle est valide, une action associée est exécutée. Pour résumé, un programme C résultant d'un fichier lex correspond à notre analyseur G0 ou GPL.
Un fichier lex est décomposé en 2 grandes parties : les définitions régulières, et les règles. Les définitions régulières, décrites sous la forme *identificateur exprRegulière* forment les expressions que l'analyseur devra reconnaitre. L'expression sont de la forme classique des expressions régulières UNIX. Exemple :

```
lettre  [A-Za-z]
chiffre  [0-9]
```

Les règles, elles, correspondent aux actions à réaliser en cas de détection d'une expression. Un programme lex étant avant tout un programme C, les actions sont des instructions qui, au fil de du déroulement de l'analyse, seront exécutées. Ces règles sont de la forme : *identificateur {action}*. Exemple :

```
if                      {return SI;}
then                    {return ALORS;}
{lettre}{alphanum}*     {return IDENTIF;}
```

Certains choix vont être appliqués aux différentes règles : parmis toutes les règles, celle reconnaissant la plus grande chaine est appliquée, si elles ont la même taille, la première règle par ordre d'apparition est appliqué. Si aucune règle n'est appliqué pour un caractère, le programme l'affichera.
Une fois l'analyse terminée, le programme produit un fichier lex.yy.c qui contiendra l'ensemble des règles de définition de l'analyseur.

3 Yacc

Également écrit en C, cet outil complète lex en proposant de produire un analyseur syntaxique pour un schéma de traduction proposé en entrée qui sera également produit en C. Ce schéma de traduction, pouvant être le fichier généré par lex, se compose d'une gramme, et d'actions sémantique associés. Un programme yacc représente un analyseur LALR.
Le programme est divisé en deux grandes parties : les déclarations, et les règles de traduction. Les déclarations sont composés de variables globales au code, ainsi que de différentes spécificités de yacc tels que les tokens définissant des caractères terminaux. Exemple :

```
%token ENTIER
```

Les règles correspondent à une grammaire écrite sous la forme :

```
<partie gauche> : <alt1> {action semantique 1}
| <alt2> {action semantique 2}
...
| <altn> {action semantique n}
;
```

La partie gauche, correspondant au non terminal d'une règle et la partie droite à la règle de production. Chaque règle sera associée à une action, écrite sous la forme d'instructions C, qui seront exécutés lorsque la règle correspondante aura été reconnue dans le texte et un *reduce* aura permis de réduire la règle. On peut notamment accéder aux différents symboles d'une règle avec *$$* qui indiquera le non terminal en partie gauche, ou bien *$i* qui indiquera le i-ème terminal ou non terminal de la partie droite.
Finalement, et c'est là l'avantage de combiner yacc et lex, le programme yacc doit avoir des règles de productions des *tokens* définis précédemment, nous avons pour ce faire la possibilité d'écrire un programme C reconnaissant les différents symboles, ou bien d'utiliser l'analyseur produiot dans le fichier lex.yy.c .