Projet IN406 - Évaluation d'expressions booléennes

DANKOU Mathis et SOURSOU Adrien

Mai 2019

1 Questions théoriques

Question 1:

On définit G la grammaire reconnaissant les expressions booléennes :

$G = (\Sigma,\ V,\ S,\ P)$
$\Sigma = \{0,\ 1,\ +\ ,\ .\ ,\ \Rightarrow\ ,\ \Leftrightarrow\}$
$V = \{E,\ C,\ OP\}$
$S = E$
$P = ($
$\quad E \rightarrow C \mid (E) \mid NON\ E \mid E\ OP\ E$
$\quad C \rightarrow\ 0 \mid 1$
$\quad OP \rightarrow\ + \mid . \mid\ \Rightarrow\ \mid\ \Leftrightarrow$
)

Question 3:

c = [[NON | (*]* [0 | 1])*
e = c [[+ | . | $\Rightarrow$ | $\Leftrightarrow$] c]*
On définit le langage L, tel que $L = \{\ w \in e,\ n\ \geq 0,\ |w|_(= |w|_) = n\ \}$

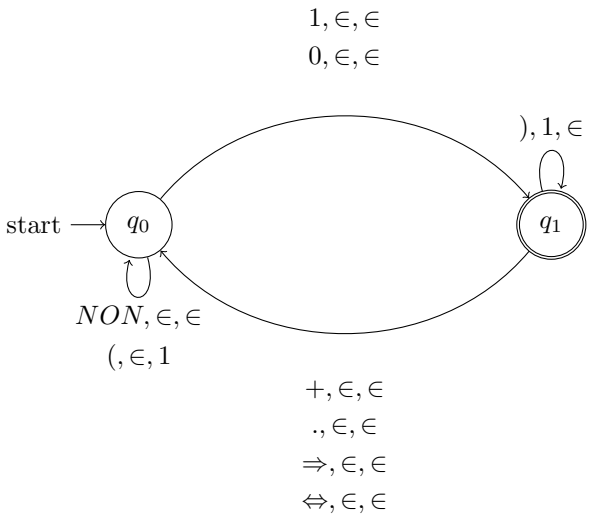

Figure 1: Automate à pile reconnaissant le langage par état final et pile vide.

Question 7:

La priorité des opérateurs est décrite par le tableau de priorité ci-dessous:

Opérateur	Priorité
NON	1
.	2
+	3
$\Rightarrow$	4
$\Leftrightarrow$	5

2 Commentaires

```
typedef struct token* liste_token;
struct token
{
        e_type type;
        e_valeur valeur;
        liste_token suivant;
};

typedef struct arbre* arbre_token;
struct arbre
{
        e_type type;
        e_valeur valeur;
```

```
        arbre_token gauche;
        arbre_token droite;
};
```

Le programme par défaut n'affiche que le résultat de l'expression, une variable DEBUG peut être modifiée afin de visualiser l'arbre construit.
En plus des fonctions *string_to_token* et *arbre_to_int*, nous utilisons la fonction *liste_token_to_postfixe*, permettant de transformer notre liste de tokens en liste postfixe. Au sein de la fonction *arbre_to_int*, les éléments de type constante de la liste sont ajoutés à un tableau d'arbre_token servant de pile. Les arbres sont combinés ensemble lorsque un token avec le type opérateur est rencontré.
La conversion de la chaîne de caractère en liste de tokens se fait en un seul parcours. La liste de tokens est parcourue à trois reprise: une première fois afin de la transformer en liste de tokens postfixe, une seconde afin de calculer la taille maximum de notre pile d'arbres et une dernière fois pour construire notre arbre de tokens.
On peut ainsi estimer que la complexité de notre programme est en $O(n)$ linéaire.