Projet IN406 - Évaluation d'expressions booléennes

BOUSSOUAR Feriel et MOUNGAD Massilia

25 mai 2021

Question 1 :

Lire une chaine de caractère contenant une expression arithmétique et la transformer en une liste de tokens.
Nous avons créé et stocké un token en utilisant des enums, des struct et des fonctions suivantes :

```
typedef enum { CONSTANTE, OPERATEUR, PARENTHESE } typeT;

typedef enum
{
    FAUX = 0, VRAI = 1,
    NON, ET, OU, IMPLICATION, EQUIVALENCE,
    GAUCHE , DROITE
}   valeurT;

typedef struct token* liste_token;
struct token
{
    typeT type;
    valeurT val;
    liste_token suiv;
};
```

- liste_token allouer_liste_token(typeT type, valeurT val) : permet d'allouer la mémoire à un token dans la liste.
- void desallouer_liste_token(liste_token liste) : permet de libèrer la liste de tokens en mémoire.
- liste_token string_to_token(const char *string) : permet de transformer une chaîne de caractères en liste de token, elle teste à chaque itération le caractère string de la chaine pour lui attribuer un type et une valeur par exemple s'il est égale à 0 le type sera CONSTANTE de valeur v = FAUX afin de le stocker dans un token.

Question 2 :

Donner un automate à pile reconnaissant le langage dont les mots sont les expressions booléennes.

Soit L le langage :
$L = \{\ w\ générée\ par\ G,\ |w|_(= |w|_)\ \}$
le nombre de parenthèses ouvrantes = le nombre de parenthèses fermantes

Définition formelle :

$A = (\Sigma,\ Q,\ \Gamma,\ q_0, \delta,\ F,\ T)$
$\Sigma = \{\ 0,\ 1,\ NON,\ +,\ .,\ \Rightarrow,\ \Leftrightarrow\ \}$
$Q = \{\ q_0,\ q_1\ \}$
$\Gamma = \{\ X\ \}$
δ
$q0 = \{\ q0\ \}$
$F = \{\ q_1\ \}$
$T = \{$
$(q_0,\ \delta,\ NON,\ \delta,\ q_0),\ (q_0,\ \delta,\ (, X,\ q_0\ ),$
$(q_0,\ \delta,\ 0,\ \delta,\ q_1\ ),\ (q_0,\ \delta,\ 1,\ \delta,\ q_1\ ),$
$(q_1,\ \delta,\ +,\ \delta,\ q_0),\ (q_1,\ \delta,\ .,\ \delta,\ q_0),$
$(q_1,\ \delta,\ \Rightarrow,\ \delta,\ q_0),\ (q_1,\ \delta,\ \Leftrightarrow,\ \delta,\ q_0),$
$(q_1,\ X,\ ),\ \epsilon,\ q_1),$
$\}$

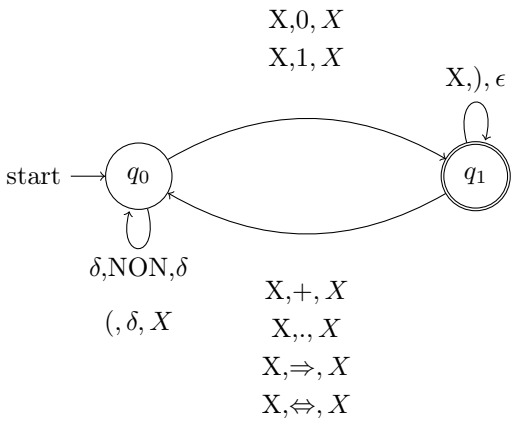

FIGURE 1 – Automate à pile reconnaissant le langage par état final et pile vide.

— Si on lit une parenthese ouvrante (: on empile dans la pile et on reste dans q0
— Si on lit NON : on fait rien dans la pile et on reste dans q0

— Si on lit 0 ou 1 : on fait rien dans la pile et on va de q0 à q1
— Si on lit une parenthese fermante) : on depile et on reste dans q1
— Si on lit + ou . ou ⇒ ou ⇔ : on fait rien dans la pile et on va de q1 à q0

Question 3 :

La fonction qui teste si une liste de token appartient au langage ou non
int expression_appartient_langage(liste_token liste) : elle prend en paramètre une liste et elle vérifie si l'expression est valide

Question 4 :

Créer l'arbre représentant l'expression booléenne
La strucutre pour stocker un arbre des tokens :

```
typedef struct arbre* arbre_token;
struct arbre
{
    typeT type;
    valeurT val;
    arbre_token gauche;
    arbre_token droite;
};
```

Struct arbre a deux pointeurs de type struct arbre* pour le fils gauche et le fils droit.
Les fonction pour créer l'arbre qui représente l'expréssion booléene :

— arbre_token alloue_arbre_token(liste_token liste) : permet d'allouer la memoire à un token pour le stocker dans l'arbre
— arbre_token liste_token_to_arbre_token(liste_token l) : permet de transformer une liste de tokens en arbre

Question 5 :

Calculer la valeur de l expression arithmétique et afficher le résultat

— int calculer(valeurT x, valeurT y, valeurT op), nous avons fait dans cette fonction un switch pour calculer et retourner une valeur de a et b en fonction de type de l opérateur.

```
 int calculer(valeurT x, valeurT y, valeurT op)
{
        switch (op)
        {
            case (NON) :
                return ! x;
            case (OU) :
                return x|y;
            case (ET) :
                return x&y;
            case (IMPLICATION) :
                return (! x) | y;
            case (EQUIVALENCE) :
                return ! x ^ y;
        }
}
```

— int arbre_to_int(arbre_token arb) : permet de calculer la valeur de l'expression booléenne stockée dans la liste des tokens dans l arbre en faisant un appel récursif de la fonction calculer pour calculer la valeur entre fils gauche et fils droit.

```
int arbre_to_int(arbre_token arb)
{
    if (! arb)   return 0;
    else if (arb->type == CONSTANTE)
        return at->valeur;
    return calcule(arbre_to_int(at->gauche), arbre_to_int(arb->droite), arb->valeur);
}
```

Commentaires Supplémentaires :

En plus de la fonction liste_token liste_token_to_arbre(liste_token liste), nous avons développé la fonction liste_token liste_token_postfixe(liste_token liste) permettant de transformer la liste token en liste postfixe à l'aide d'une pile en utilisant les macros PUSH et POP pour empiler et dépiler dans la pile afin de pouvoir évaluer notre expression booléene.