Rapport de projet

Wollenburger Antoine, Chénais Sébastien

Sommaire

Introduction

Il nous a été demandé d'écrire un compilateur, en OCAML, d'un langage de notre cru vers du svg (format pour décrire des images vectorielles). Pour mener à bien cette tâche, nous nous reposerons sur les notions que nous avons vu en cours.

Chapitre 1

Évolution du projet

1.1 Définition d'une grammaire

La première étape de ce projet a été la création d'un langage et sa grammaire associée. Même si le projet global se fera de manière incrémentale, il faut penser au fonctionnement global dès le début.

Le fichier SVG est déclaré par l'instruction "drawing" suivi du nom du dessin et la taille du canevas sous la forme "[largeur , hauteur]". Les instructions sont incluses entre crochets.

Une instruction peut être de différents types :

Une déclaration de variable : on créé une variable en indiquant son type, son nom et enfin les valeurs qui servent à la construire.

Le dessin d'une variable : pour les variables dont le type est adéquat, il s'agit de l'instruction qui déclenche l'affichage de ladite variable. La forme est "draw nomDeLaVariable".

Une conditionnelle : il s'agit d'un test sur une ou plusieurs conditions qui influe sur la suite de l'exécution. Cette instruction est de la forme "if(-conditions-){}else{}". Si la condition est vérifiée, alors le bloc suivant le if est exécuté, celui de else sinon.

Une boucle : il s'agit d'une boucle sur condition. L'instruction est de la forme "while(-conditions-){}". Le bloc entre crochets est répété jusqu'à ce que la condition soit invalidée.

Les fonctions et procédures : il s'agit d'un appel a une fonction ou une procédure. L'instruction est de la forme "nomDeLaFonction (-paramètres-)". Pour les fonctions, il est possible de récupérer une valeur de retour.

Listing 1.1– Exemple de fichier source

```
/* blabla bla commentaire
sur plusieurs lignes... */

drawing monDessin [256 , 256]
```

```
{
    //com sur une ligne
    /*function dessineSablier(Point centre, Number hauteur, Number largeur)
        :void
    {
        Point HG(centre.x-hauteur/2,centre.y-largeur/2);
        Point HD(centre.x-hauteur/2,centre.y+largeur/2);
        Point BG(centre.x+hauteur/2,centre.y-largeur/2);
        Point BD(centre.x+hauteur/2,centre.y+largeur/2);

        Line diag2(BD,HG);
        Line haute(HD,HG);
        Line diag1(BG,HD);
        Line basse(BG,BD);

        draw diag2;
        draw diag1;
        draw haute;
        draw basse;
    }*/

    //Number test := 2;
    Point origine(50,50); //commentaire
    Point destination(100,100);

    Line ligne(origine,destination); //ligne
    /*if(test = 2){
        draw origine;
    }
    else{
        draw destination;
    }
    while(test > 0){
        draw destination;
        test := test -1;
    }*/
    //origineSablier.x := 40;
    //dessineSablier(c,20,20);
    draw ligne;
}
```

1.2 Lexer

Nous avons alors écrit l'analyseur syntaxique, comprenant les mots de notre grammaire. Cette étape est de loin la plus facile.

1.3 Parser

Ensuite, nous avons transformé le fichier d'entrée en un arbre binaire, modèle de représentation intermédiaire, qui nous permettra l'écriture du fichier svg. Nous avons

décidé d'implémenter nos arbres comme suit :

Listing 1.2– Type arbre

```
type operation =
    Drawing
  | DrawingSize
  | BlocEmbrace
  | Declaration
  | BlocBrace
  | Parameters
  | Point
  | Line
  | Instruction
  | Draw
  | Var of (string)
  | Number of (int);;

type t_arbreB = Empty | Node of node
        and node = { value: operation; left: t_arbreB; right: t_arbreB };;
```

1.4 Et le bonheur arrive.

Viens alors la partie de loin la plus fastidieuse. En effet, pour transformer cet arbre binaire en svg, nous avons besoin d'une implémentation d'un automate à pile. Pour ceci point de choix. Nous passerons par une table shift-reduce. A la vue de la taille du vocabulaire, cette table est tout sauf petite.