Projet de Compilation

Mathieu Brochard
Antoine Chauvin

25 décembre 2016

1 Introduction

Ce projet consiste à réaliser un compilateur capable de traduire une expression mathématique écrite en LaTeX vers un flux MathML. Cette application reprend les notions de compilation vues en cours et utilise les outils Flex et Bison vus en travaux pratiques. Ce traducteur, appelé TeX2ML, se présente sous la forme d'un exécutable prenant une expression LaTeX en argument et écrit le flux MathML sur la sortie standard. Cela rend l'exécutable facilement utilisable par un CGI web, par exemple.

Exemple

`tex2ml "\frac{-b*4ac}{\sqrt{b+4ac}}"` produira le flux MathML :

```
<math display="block">
    <mfrac>
        <mrow>
            <mo>-</mo>
            <mi>b</mi>
            <mn>4</mn>
            <mi>a</mi>
            <mi>c</mi>
        </mrow>
        <mrow>
            <msqrt>
                <mrow>
                    <mi>b</mi>
                    <mo>+</mo>
                    <mn>4</mn>
                    <mi>a</mi>
                    <mi>c</mi>
                </mrow>
            </msqrt>
        </mrow>
    </mfrac>
</math>
```

Ce qui, dans un navigateur adapté, produira ce résultat :

$$\frac{-b * 4ac}{\sqrt{b + 4ac}}$$

A noter que l'expression LaTeX fournie à l'exécutable doit obligatoirement être entourée de guillemets droits pour éviter à l'interpréteur de commandes d'échapper certains caractères à cause de la barre oblique.

2 Usage

Le projet est architecturé ainsi :

```
/
├── src
│   ├── tex2ml.l
│   └── tex2ml.y
├── bin
│   └── tex2ml
├── README.md
├── LICENSE
└── Makefile
```

Le répertoire `src/` contient les sources de l'analyseur lexical (`tex2ml.l`) et les sources de l'analyseur syntaxique (`tex2ml.y`). Lors de la compilation, l'executable `tex2ml` sera placé dans le répertoire `bin/`.

Pour compiler le projet, exécutez le `Makefile` avec la commande `make`. Ensuite, lancez le programme `tex2ml` situé dans le répertoire `bin`. Le programme attend un argument qui est l'expression LaTeX à traduire qui doit obligatoirement être entre guillemets droits. Le programme retournera le flux MathML sur la sortie standard. Vous pourrez rediriger ce flux à votre gise (dans un fichier, par exemple).

3 Analyseur lexical

Tout d'abord, ce compilateur se compose d'un analyseur lexical écrit avec Flex. Son rôle est de lire l'expression LaTeX et de la découper en *tokens*. Une expression comporte plusieurs types d'éléments comme les *identifiers* (a, b, c, ...), les *numbers* (1, 2, 3, ...) mais aussi des symboles spéciaux comme les fractions (`\frac`) ou encore les racines carrées (`\sqrt`).

On utilise donc des expressions rationnelles pour reconnaître ces motifs. Par exemple, pour reconnaître un *identifiers*, nous utilisons l'expression rationnelle `[a-zA-Z]`. Quand l'analyseur lexical reconnaît ce motif, il informe l'analyseur syntaxique que le prochain *token* est un *identifier* et sa valeur et a, par exemple.

Exemple

Pour reconnaître un *identifier* :

```
[a-zA-Z] { yylval = strdup(yytext); return CHAR; }
```

Lorsque l'analyseur lexical reconnaît un *identifier*, il stocke sa valeur (a, b, ...) dans la table d'analyse et renvoie la constante `CHAR`. Cette constance sera utilisée par l'analyseur syntaxique pour qu'il sache que le *token* courant est un *identifier*.
A noter que l'on utilise la fonction `strdup` qui permet de dupliquer une chaîne de caractères. En effet, on ne peut se contenter d'assigner à `yylval` un pointeur, car celui-ci change à chaque *token* lu, nous perdrion sa valeur.

4 Analyseur syntaxique

L'analyseur syntaxique va permettre de vérifier la validité syntaxique de l'expression fournie. Par exemple, le *token* `\sqrt` doit toujours être suivi d'une accolade ouvrante, d'une expression et d'une accolade fermante. De plus, il va effectuer certaines actions qui permettront de traiter l'aspect sémantique de l'expression. Par exemple, quand l'analyseur syntaxique lira un nombre, il effectuera l'action qui consiste a entourer le nombre par des balises `<mn></mn>`.

Exemple

Prenons l'expression `\sqrt{2a}`. L'analyseur syntaxique lit le premier *token* qui correspond à la constante `SQRT`. Il utilise donc cette grammaire :

```
expr:
    expr identifier
    | expr numeric
    |
;

sqrt:
    SQRT        { strcat(mathml, MSQRT_OPEN);  }
    OPEN_BRACE  { strcat(mathml, MROW_OPEN);   }
    expr        { strcat(mathml, MROW_CLOSE);  }
    CLOSE_BRACE { strcat(mathml, MSQRT_CLOSE); }
;
```

L'analyseur syntaxique stocke dans une chaîne de caractères la balise `<msqrt>` suivi de la balise `<mrow>`. Dans ces balises, il va analyser l'expression `2a`, et l'entourer des balises `<mn></mn>` et `<mi></mi>`. Il stocke cette expression à l'intérieur de la balise `<mrow>`. Ensuite, il ferme la balise `<mrow>` et `<msqrt>`. L'expression MathML sera donc :

```
<msqrt>
    <mrow>
        <mn>2</mn>
        <mi>a</mi>
    </mrow>
</msqrt>
```

A noter que la grammaire a été très simplifiée pour l'exemple.