POLYTECH LILLE

Année 2019

Rapport de Projet Langages et Traducteurs

Conception d'un Analyseur/Traducteur de document LATEX – HTML

DEVISME Claire

LOUIS-FESTOC Pauline

Table des matières

Introduction

Le langage LATEX est un langage qui permet une mise en page simple pour obtenir des documents structurés. Le but du projet est de concevoir un traducteur qui reconnaît la syntaxe de déclaration de documents Latex et de générer un document HTML équivalent. Il s'agit donc d'établir un outil qui nous permettra, en quelque sorte, de convertir un document LATEX en un document HTML.

1. Grammaires et schémas de traduction dirigée par la syntaxe

Grammaire

On définit une grammaire permettant de décrire la structure syntaxique du langage LATEX.

racine → debut

début → options **BEGIN_DOC** maketitle contenu **END_DOC**
| **BEGIN_DOC** maketitle contenu **END_DOC**

maketitle → **MAKETITLE** | **ε**

options → author options | date options | title options | **ε**

author → **AUTHOR ACCOUV** mot **ACCFER**

date → **DATE ACCOUV** mot **ACCFER**

title → **TITLE ACCOUV** mot **ACCFER**

commande → **LIGNE** | **GRAS ACCOUV** mot **ACCFER** | **ITALIQUE ACCOUV** mot **ACCFER**
| **DOTS** | **ε**

mot → commande mot | commande | **MOT** commande mot | **ε**

contenu → commande contenu | mot contenu | liste contenu | num contenu
|section contenu | **ε**

liste → **BEGIN_ITEMIZE** mot item **END_ITEMIZE**
| **BEGIN_ITEMIZE** liste **END_ITEMIZE**
| **BEGIN_ITEMIZE** num **END_ITEMIZE**

num → **BEGIN_ENUM** mot item **END_ENUM**
| **BEGIN_ENUM** liste **END_ENUM**
| **BEGIN_ENUM** num **END_ENUM**

Item	→	ITEM mot fin | ITEM mot liste fin |ITEM mot num fin
fin	→	item | ε
section	→	SEC ACCOUV mot ACCFER contenu subsection subsubsection
subsection	→	SUBSEC ACCOUV mot ACCFER contenu subsection | ε
subsubsection	→	SUBSUBSEC ACCOUV mot ACCFER contenu subsubsection | ε

Ci-dessus, les terminaux sont définis en majuscule et en gras et les autres termes constituent les non-terminaux.

Schémas de traduction dirigée par la syntaxe

racine	→	debut
debut	→	options BEGIN_DOC maketitle contenu END_DOC
debut	→	BEGIN_DOC maketitle contenu END_DOC
maketitle	→	MAKETITLE
maketitle	→	ε
options	→	author options {options.nba=options1.nba + 1; options.nbd=options1.nbd; options.nbt=options1.nbt;}
options	→	date options {options.nba=options1.nba; options.nbd=options1.nbd+1; options.nbt=options1.nbt;}
options	→	title options {options.nba=options1.nba; options.nbd=options1.nbd; options.nbt=options1.nbt+1;}
options	→	ε {options.nba=0; options.nbd=0; options.nbt=0;}
author	→	AUTHOR ACCOUV mot ACCFER
author	→	DATE ACCOUV mot ACCFER
author	→	TITLE ACCOUV mot ACCFER
commande	→	LIGNE
commande	→	GRAS ACCOUV mot ACCFER
commande	→	ITALIQUE ACCOUV mot ACCFER

commande	→	DOTS
commande	→	ε
mot	→	commande mot
mot	→	commande
mot	→	MOT commande mot
mot	→	ε
contenu	→	commande contenu
contenu	→	mot contenu
contenu	→	liste contenu
contenu	→	num contenu
contenu	→	section contenu
contenu	→	ε
liste	→	BEGIN_ITEMIZE mot item END_ITEMIZE
liste	→	BEGIN_ITEMIZE liste END_ITEMIZE
liste	→	BEGIN_ITEMIZE num END_ITEMIZE
num	→	BEGIN_ENUM mot item END_ENUM
num	→	BEGIN_ENUM liste END_ENUM
num	→	BEGIN_ENUM num END_ENUM
item	→	ITEM mot fin
item	→	ITEM mot liste fin
item	→	ITEM mot num fin
fin	→	item
fin	→	ε
section	→	SEC ACCOUV mot ACCFER contenu subsection subsubsection
subsection	→	SUBSEC ACCOUV mot ACCFER contenu subsection
subsection	→	ε
subsubsection	→	SUBSUBSEC ACCOUV mot ACCFER contenu subsubsection
subsubsection	→	ε

2. Structures et types définis

Fichier Lex

Dans notre fichier Lex, nous avons défini plusieurs objets, dans les différentes sections du fichier.

Dans un premier temps, on s'intéresse à la section des déclarations, qui est la suivante :

```
/* Définition de macros */
chiffre         [0-9]
lettre          [a-zA-Zéàùïäüöçëêôûâîè]
caractere       [\-;,:\.'\(\)]
espace          [ ]
sep                     [\t\n]
```

Nous avons défini ici 5 macros (nom que l'on donne à un ensemble de caractères) différentes. Voici ce que les macros désignent :

- Macro chiffre : n'importe quel chiffre de 0 à 9
- Macro lettre : n'importe quelle lettre, minuscule ou majuscule, ainsi que les lettres accentuées
- Macro caractere : les caractères suivants : - , ; : . ' ( )
- Macro espace : un espace
- Macro sep : une tabulation ou bien un retour à la ligne.

On passe désormais à la section des règles de traduction qui est la suivante :

```
"\\begin\{document\}"   return BEGIN_DOC;
"\\end\{document\}"     return END_DOC;
"\\title"               return TITLE;
"\\author"              return AUTHOR;
"\\date"                return DATE;
"\\maketitle"           return MAKETITLE;
"\\section"             return SEC;
"\\subsection"          return SUBSEC;
"\\subsubsection"       return SUBSUBSEC;
"\\textbf"              return GRAS;
"\\textit"              return ITALIQUE;
"\\begin\{itemize\}"    return BEGIN_ITEMIZE;
"\\end\{itemize\}"      return END_ITEMIZE;
"\\begin\{enumerate\}"  return BEGIN_ENUM;
"\\end\{enumerate\}"    return END_ENUM;
"\\item"                return ITEM;
"\\\\"                  return LIGNE;
"\\ldots"               return DOTS;
({lettre}|{caractere}|{chiffre}|{espace})+       return MOT;
{sep}                           ;
"\{"                    return ACCOUV;
"\}"                    return ACCFER;
```

Cette section est séparée en deux parties : l'une contenant un motif, et l'autre une action. Quand l'analyseur aura reconnu un motif, il pourra effectuer l'action associée. Les motifs déclarés entre guillemets sont des mots qui peuvent se trouver dans un fichier .tex, comme par exemple \begin{document}, qui est la commande nécessaire pour commencer un document. De plus lors de la déclaration de ces motifs, on utilise l'opérateur \ qui permet « d'annuler » la fonction de certains caractères. C'est pourquoi pour déclarer \begin{document} on écrit \\begin\{document\}, de même pour les autres motifs. On définit également un mot grâce aux macros déclarées précédemment : ainsi, un mot ne peut être vide (commande +), et peut contenir un ou plusieurs caractères, chiffres, lettres ou espaces.

Ensuite, on associe des actions à tous ces motifs. Les actions sont toutes les mêmes : elles retournent toutes un token (nom donné à un non terminal de la grammaire) : MOT, ACCOUV, ITALIQUE… Seul le motif sep ne retourne pas de token. Dans ce cas, lorsque l'analyseur va rencontrer une tabulation ou un retour à la ligne, il ne va rien se passer.

Les tokens sont liés à des commandes que l'on peut rencontrer dans un fichier Latex. Voici le tableau des correspondances entre les commandes et les tokens :

\begin\{document\}	BEGIN_DOC
\end\{document\}	END_DOC
\title	TITLE
\author	AUTHOR

\date	DATE
\maketitle	TITLE
\section	SEC
\subsection	SUBSEC
\subsubsection	SUBSUBSEC
\textbf	GRAS
\textit	ITALIQUE
\begin\{itemize\}	BEGIN_ITEMIZE
\end\{itemize\}	END_ITEMIZE
\begin\{enumerate}	BEGIN_ENUM
\end\{enumerate}	END_ENUM
\item	ITEM
\\	LIGNE
\ldots	DOTS
({lettre}|{caractere}|{chiffre}|{espace}) +	MOT
{sep}	
{	ACCOUV
}	ACCFER

Enfin, nous n'avons rien déclaré dans la section des fonctions auxiliaires (section optionnelle).

Fichier Accent

On passe désormais au fichier Accent. De même qu'un fichier Lex, un fichier Accent comprend trois sections : une section de déclarations C, une section de déclaration des tokens, et une partie de règles de traduction.

La section des déclarations C est la suivante :

```
%prelude{ /* Code C */
    /* Inclusion de bibliotheques C */
    #include <stdio.h>
    #include <malloc.h>
    /* Action de fin d analyse */
    void fin_analyse(){
        printf("Syntaxe correcte\n");
    }
    /*Action qui vérifie le nombre d'options*/
    void nb_options(int a, int d, int t){
        int error=0;
        if(a>1) {printf("Erreur ! trop de commandes 'author' declarees\n");
error+=1;}
```

```
        if(d>1) {printf("Erreur ! trop de commandes 'date' declarees\n");
error+=1;}
        if(t>1) {printf("Erreur ! trop de commandes 'title' declarees\n");
error+=1;}
        if(error !=0) yyerror("\nRevoir le nombre d'options du debut de
document\n");
    }
}
```

On y a inclu les bibliothèques nécessaires pour l'exécution du fichier (stdio et malloc), et on a également défini les actions fin_analyse et nb_options qui seront appelées dans la partie des règles de traduction. Ces actions seront expliquées par la suite.

On passe à la section de déclaration des tokens :

```
/* Declaration des tokens */
%token BEGIN_DOC, END_DOC, AUTHOR, TITLE, DATE, MAKETITLE, SEC, SUBSEC,
SUBSUBSEC, GRAS, ITALIQUE, BEGIN_ITEMIZE, END_ITEMIZE, BEGIN_ENUM,
END_ENUM, ITEM, LIGNE, DOTS, MOT, ACCOUV, ACCFER;
```

Précédemment, nous avons associé, dans le fichier Lex, un token à chaque terminal de la grammaire. Ici, on déclare une nouvelle fois tous les tokens après le mot clé %token.

Enfin, on passe à la partie des règles de traduction. Cette section sert à décrire les différentes productions de la grammaire. Les parties à gauche sont des non-terminaux, et les parties de droites contiennent à la fois des terminaux (les tokens déclarés dans la section précédente) et des non-terminaux. Un même non-terminal peut produire plusieurs productions : on l'indique avec le symbole |, tandis que le symbole ; sépare les déclarations de productions.

C'est également dans cette section qu'on ajoute des règles sémantiques. Ces actions peuvent être insérées à n'importe quel endroit dans la partie droite des productions, entre les symboles { et }.

3. Attributs et variables utilisés, avec leurs objectifs

Cette partie a pour but de présenter les fonctions ou actions utilisées, ainsi que les attributs associés aux non-terminaux de la grammaire lors des règles sémantiques, et leurs objectifs.

Action fin_analyse

Cette action ne prend aucun paramètre en entrée et n'en retourne aucun. Cependant, si la grammaire reconnait un fichier, alors elle affiche « syntaxe correcte ».

Action nb_options

Cette action permet de vérifier qu'il y ait bien au maximum une commande title, author, et date de définie. Pour cela, elle prend en trois paramètres en entrée, tous des entiers. Pour chaque entier, elle vérifie s'il n'est pas strictement supérieur à 1. Si c'est le cas, alors un message d'erreur est affiché, et une variable locale error s'incrémente (la variable est initialisée à 0).

4. Tests réalisés

Afin de vérifier la cohérence du projet, nous avons effectué plusieurs tests, à plusieurs stades du projet. Ces tests étaient nécessaires pour avancer dans le projet.

Tests sur la grammaire

Pour tester si notre grammaire fonctionnait, et pour détecter les erreurs commises dans le cas contraire, nous avons repris le fichier *contenu.tex* mis à notre disposition. Nous avons ensuite repris les éléments petit à petit, du plus simple au plus complexe, pour tester leur fonctionnement. Pour illustrer nos propos, au tout début, notre fichier contenu1.tex ne contenait que les balises de début et de fin de document. Nous avons ensuite enrichi le document en intégrant et testant les autres commandes. C'est ainsi que nous avons pu corriger nos erreurs pour obtenir une grammaire correcte, qui reconnait le fichier *contenu.tex* sans problème.

Tests sur les actions sémantiques

Les contraintes suivantes devaient être respectées :

- Les commandes title, author et date ne devaient apparaître (au plus) une seule fois,
- Une commande subsection ne pouvait être déclarée dans avoir déclaré précédemment section,
- Même chose avec les commandes subsubsection et subsection.

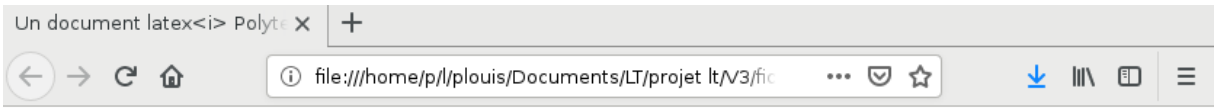
Pour réaliser ces contraintes, nous avons créé les règles sémantiques et actions présentées plus haut. Pour vérifier que ces contraintes fonctionnaient, nous avons créé un fichier erreur.txt dans lequel nous testions des choses non autorisées afin de voir si nous avions bien un message d'erreur. Par exemple, dans le fichier *erreur.txt*, nous avons déclaré plusieurs fois les commandes title et author, et nous avons eu un message d'avertissement. De même en déclarant une commande subsection avant de déclarer la commande section.

Tests sur la génération du HTML

Pour vérifier que notre grammaire générait un fichier HTML correct, nous avons exécuté notre programme et redirigé la sortie vers un fichier appelé essai.html, de la façon suivante (html étant le nom de notre exécutable) :

```
./html < contenu.tex > essai.html
```

Ensuite, nous avons simplement ouvert ce fichier dans une page web pour vérifier qu'il correspondait bien à ce qui était attendu.



Document *Latex*

Un document **Latex** est un document texte qui contient certaines commandes qui seront traitées par un compilateur.
L'une des forces de Latex est de pouvoir générer des documents très propres, *très structurés* avec une numérotation automatique.

Fonctionnement

En résumé, Latex :

- permet d'écrire des documents à l'aide d'un simple éditeur de texte,
- s'occupe de la mise en page,
- offre un résultat *incomparable* .
- La réalisation d'un document se fait en 3 phases :
    1. Edition du texte du document (vi, kwrite, xemacs, ...)
    2. Compilation du texte (latex, pdflatex)
    3. Visualisation du document généré (evince, acroread)

Voici une capture d'écran du résultat obtenu.

Conclusion

Le projet de Langages et Traducteur s'est plutôt bien déroulé, il y a eu une bonne entente au sein du binôme. Nous avons cependant eu du mal à procéder à l'écriture des règles de traduction dirigées par la syntaxe. Notamment, la numérotation des sections n'a pas pu être effectuée malgré les nombreux essais. Mais notre grammaire permet quand même de gérer l'organisation des sections puisqu'il est impossible de déclarer une sous-section avant une section, sans générer d'erreur. Nous pourrions encore améliorer notre travail en parvenant à terminer les règles de traductions.