Projet de Compilation :

Traduction d'un tableau LaTeX en tableau HTML

Alain MARGUERITE
Romain RINCÉ
Master ALMA
2011-2012

1 Introduction

Le projet consiste à transformer un fichier LaTeX générant un tableau en un fichier HTML représentant ce même tableau. L'analyseur lexicale utilisé est Flex et l'analyseur syntaxique est Bison. Nous présenterons les différents choix effectués ainsi que les problèmes qui se sont présentés durant la réalisation de ce projet.

2 Choix dans l'analyse syntaxique

Il est nécessaire d'évoquer les choix effectués sur la transformation d'un tableau LaTeX en tableau HTML :

- Le programme écrit en Flex/Bison reconnaît toujours le même en-tête minimal pour la création d'un tableau. Il avait initialement été prévu de créer une « poubelle » qui devait capturer un ensemble d'en-têtes plus larges. Cependant la réalisation de cette « poubelle » fut assez poussive voire pénible. Il fallait en effet attraper tout mot pouvant s'y trouver et l'ignorer, mais au cours de nos essais il s'est avéré assez difficile d'empêcher la poubelle de récupérer tout le fichier LaTeX.
- Pour les bordures, le programme ne regarde que l'existence de bordures sur les colonnes dans le champ d'option du tableau. Si le nombre de séparateurs est au moins égal à la moitié du nombre de colonnes alors tout le tableau HTML aura des bordures sinon il n'en aura aucun.
- Lorsque l'on effectue la somme sur des lignes ou des colonnes, le résultat sera toujours un nombre flottant avec 2 chiffres après la virgule.
- Une somme sur une ligne ou sur une colonne sera effectué même si le tableau ne contient pas que des valeurs numériques. La somme n'est cependant faite que si toute la ligne ou colonne ne contient que des valeurs numériques.
- Les caractères spéciaux tel que Δ ne sont pas transformés mais le code du caractères est recopié. Ainsi pour « *Delta* », on recopiera « *Delta* » dans le fichier HTML.
- De même pour les fonctions telles que les fonctions pour mettre tout en majuscules ou tout en minuscules, on ne réécrira que la chaîne de caractères sur laquelle la fonction est appliquée mais sans effectuer de modification.
- Les options de positionnement sur le tableau ne sont pas appliquées en HTML.

3 Difficultés rencontrées

Lors de la réalisation de ce projet de deux difficultés se sont présentées.

La méconnaissance de HTML et LaTeX est celle qui nous a posé le plus de problèmes ; la découverte chronique d'options possibles dans un tableau LaTeX nécessitait d'importantes modifications du code et donc une perte de temps. Parfois les modifications n'étaient pas faites (bien qu'elles puissent se montrer utiles, tel que centrer le tableau par exemple) de peur de casser le code Flex/Bison. Ce fut un choix malheureusement récurrent en fin de projet.

Le second problème est en fait sous-jacent au premier puisqu'il s'agit de la relative instabilité du code Flex/Bison à la moindre modification. Il est en effet assez déstabilisant de voir à quel point il parfois difficile d'ajouter une petite possibilité de lecture et le temps perdu pour finalement réussir à l'implémenter.

4 Informations complémentaires

Lors de la compilation, le compilateur nous renvoie un total de 12 décalage/réduction. Celles-ci apparaissent à cause des différentes règles vides qui apparaissent dans le programme. L'origine de ces conflits est une grammaire légèrement trop expressive et donc, à cause de ces règles vides, non déterministe. Cependant ces cas de conflits ne sont pas de véritables problèmes puisque Bison effectue par défaut le décalage ; nous avons donc veillé à ce que cela ne génère pas de problème de parsing en vérifiant que les cas ignorés par Bison n'aient jamais de sens pour la lecture du fichier.

Les décalages/réductions n'ont pas été supprimé car cela nécessitait une refonte profonde du code et permettait d'obtenir une certaine liberté d'expressivité au niveau de la grammaire.

5 Annexes

5.1 Résultats

1.1	1.2	1.3
2.1	2.2	2.3

1.10 1.20 1.30 3.60
2.10 2.20 2.30 6.60
3.20 3.40 3.60
Il fait beau sur mon tableau

Table 1: Il fait beau sur mon tableau

1.1 1.4	1.2 1.5	1.3 Δ
1.1	*blah* chNNat	
4		

Il fait beau sur mon tableau

1.10	1.20	1.30	3.60
1.40	Delta	1	
1.10	blah chnnat		
4			4.00

Il fait beau sur mon tableau

5.2 Code du projet

5.2.1 Flex

```
/*exo1.l */
%{
#include "projet_compilation_syn.h"
%}

/*def*/
deb "\\"begin
tabular tabular
table table
document document
documentclass "\\"documentclass\{article\}
graphicx "\\"usepackage\{graphicx\}
caption "\\"caption
fusion "\\"multicolumn
trait_hor "\\"hline
separ_col "&"
fin_ligne "\\""\\"
separ \|
fin "\\"end
entier [0-9]+
reel {entier}"."[0-9]+
word [a-zA-Z][a-zA-Z0-9"%"]*
antislash "\\"
croch_G "["
croch_D "]"
garbage [a-zA-Z0-9"&""\\"""""."]*

accol_G \{
accol_D \}

space [ \t\n]*
```

```
/*rules analyseur lexico*/
%%
{deb}          {printf("DEB ");return DEB;}
{document}      {printf("");return DOC;}
{documentclass} {printf("");return DC;}   /* permet de recuperer \documentclass{article} en debut de fichier*/
{graphicx}      {printf("");return GFX;}  /* permet de recuperer \usepackage{graphix} en debut de fichier*/
{table}        {printf("");return TABLE;}
{tabular}       {printf("");return TABULAR;}
{caption}       {printf("");return CAPTION;}
{fusion}        {printf("FUSION ");return FUSION;}
{trait_hor}     {printf("TH ");return TRAIT_HOR;}
{separ_col}     {printf("SC ");return SEPAR_COL;}
{fin_ligne}     {printf("FL ");return FIN_LIGNE;}
{separ}         {printf("SEPAR ");return SEPAR;}
{fin}          {printf("FIN ");return FIN;}
{accol_G}       {printf("AG ");return ACCOL_G;}
{accol_D}       {printf("AD ");return ACCOL_D;}
{entier}        {printf("INT ");yylval.entier=atoi(yytext);return ENTIER; }
{reel}         {printf("REEL ");yylval.reel=atof(yytext);return REEL; }
{word}          {printf("MOT ");yylval.str=strdup(yytext);return MOT;}
{space}         {printf("SPACEZ ");yylval.str=strdup(yytext);return SPACE;}
{antislash}     {printf("SLASH ");return ANTISLASH;}
{croch_G}       {printf("CG ");return CROCH_G;}
{croch_D}       {printf("CD ");return CROCH_D;}

.            {printf("/n%serreur/n", yytext);}

%%
```

5.2.2 Bison

```
/*
lex projet_compilation.l
yacc -d projet_compilation.y
mv lex.yy.c projet_compilation_lex.c
mv y.tab.c projet_compilation_syn.c
mv y.tab.h projet_compilation_syn.h
cc -o proj projet_compilation_lex.c projet_compilation_syn.c -lfl
 */
```

```
/*
lettre[a-zA-Z]
/*aaa [a-zA-Z0-9_] */
/*
{a}{n}{d} {return AND}
{lettre}{aaa}* {yyval chaine = (char *)strdup(yylead);return IDENTIFIANT;}
 */
```

Bison

```
//projet_compilation.yacc
%{
#include <stdio.h> //FILE
#include <stdlib.h> //atoi
#include <string.h>
  extern int yylex(void);
  extern int yyleng;
  void yyerror(char *s);
  extern FILE* yyin;
  extern FILE* yyout;
  extern FILE* yyerr;
  int nbcol = 0;
  int nbseparateur = 0;
  int  numberInLine = 1;
  float  sumLine = 0;
  int  colonneCourant = 0;
  int* numberInColonne;
  float* sumColonne;
  char * positionnement = "center";

  void ajouteColonne(char* chaine,char* position, float num);
  void redigeOption( int border, char* pos);
  char* redigeColonne(char* texte, char* pos, int multicolumn, int colspan);
  char* redigeColonneReel(double val, char* pos, int multicolumn, int colspan);
  char* redigeColonneEntier(int val, char* pos, int multicolumn, int colspan);
  void initialiseTabBool(int* tab, int size);
  void initialiseTabReel(float* tab, int size);
  void changeTab(int* tab, int deb, int end);
  void sommeColonne(char* chaine, char* position, int* appliqueSomme, float* toutesSommes, int nbFois);
%}
```

```
//

%union{ char* str; double reel; int entier;}
%token DEB
%token DOC
%token DC
%token GFX
%token TABLE
%token TABULAR
%token CAPTION
%token OPTION_T
%token FUSION
%token TRAIT_HOR
%token SEPAR_COL
%token FIN_LIGNE
%token SEPAR
%token FIN
%token ACCOL_G
%token ACCOL_D
%token ANTISLASH
%token CROCH_G
%token CROCH_D
%token<entier> ENTIER
%token<reel> REEL
%token<str> MOT
%token<str> SPACE
%type<str> mots
%type<str> blancs
%type<str> colonne
%type<str> colonnes
%type<str> phrase

%start fichier


%%

fichier   : debLatex blancs init tableau end blancs endLatex blancs{fprintf(yyout,"</html>");}
;
```

```
debLatex  : DC blancs GFX blancs DEB ACCOL_G DOC ACCOL_D blancs
;

endLatex  : FIN ACCOL_G DOC ACCOL_D
;
  /*differents types de declarations de tableau*/
init      :  tabledeb  posdeb   DEB ACCOL_G TABULAR ACCOL_D { fprintf(yyout, "<table ");}
| tabledeb  DEB ACCOL_G TABULAR ACCOL_D { fprintf(yyout, "<table ");}
| DEB ACCOL_G TABULAR ACCOL_D { fprintf(yyout, "<table ");}
;

end       : FIN ACCOL_G TABULAR ACCOL_D blancs posend blancs caption tableend
;

tabledeb  : DEB ACCOL_G TABLE ACCOL_D CROCH_G MOT CROCH_D blancs
;

tableend  : FIN ACCOL_G TABLE ACCOL_D
|
;

posdeb       :  DEB ACCOL_G MOT ACCOL_D blancs{positionnement = $3;}
;

posend    : FIN ACCOL_G MOT ACCOL_D
;

caption : CAPTION ACCOL_G phrase ACCOL_D blancs {fprintf(yyout,"%s",$3);}
|
;

  /*lorsque toutes les lignes du tableau ont été lues,*/
  /*on alloue une chaine de caractères qui va contenir la ligne HTML des totaux des colonnes,*/
  /*et on lance la fonction sommeColonne qui va rediger cette ligne dans la chaine de caractères*/
  /*Une fois la chaine complétée, on la rédige dans le fichier*/
tableau   : options blancs lignes {char* ligneDesTotaux = malloc(sizeof(char)*1000);
sommeColonne(ligneDesTotaux,positionnement, numberInColonne, sumColonne, nbcol);
 fprintf(yyout,"%s</tbody>\n</table>\n",ligneDesTotaux);
 free(ligneDesTotaux);}
;
```

```
 /*Lorsque les options ont été lues,*/
 /*on alloue deux tableau de taille égale au nb de colonnes dans le tableau*/
 /*Le tableau numberInColonne contient des booléens qui indique pour chaque colonnes*:
 /*si elle ne contient que des valeurs numériques*/
 /*Le tableau sumColonne stocke la somme de chacune des colonnes*/
 /*Enfin on lance la fonction redigeOption qui prends une valeur booléenne qui indique si le tableau aura des bordures*/
options   :  ACCOL_G option ACCOL_D {numberInColonne = malloc(nbcol * sizeof(int));
  sumColonne = malloc(nbcol*sizeof(float));
  initialiseTabBool(numberInColonne, nbcol);
  initialiseTabReel(sumColonne, nbcol);
  redigeOption(nbseparateur>=nbcol/2, positionnement);}
;

/*option permet de connaitre le nombre de colonnes et de separateur*/
option    : SEPAR option {nbseparateur++;}
| MOT option {nbcol =+ strlen($1);}
| SEPAR {nbseparateur++;}
| MOT {nbcol =+ strlen($1);}
;

/*parcourt toutes les lignes*/
lignes    : ligne FIN_LIGNE  blancs lignes{}
| ligne FIN_LIGNE blancs {}
| TRAIT_HOR blancs lignes;
| TRAIT_HOR blancs
;

/*Lorsqu'une ligne est traité on recupère la valeur renvoyée par colonnes et on l'affiche entre deux balise */
/*Et l'on reinitialise les variables "numberInLine" qui dit si la ligne ne contient que des valeurs numeriques */
/*colonneCourante indique le numero de la colonne parcouru lors du parcours d'une ligne*/
/*sumLigne va conserver la sum de toutes les valeurs numériques de la ligne */
ligne     : colonnes {
 if (numberInLine)
   {
     ajouteColonne($1,positionnement, sumLine);
   }
 fprintf(yyout,"<tr>%s</tr>\n",$1);
 free($1);
 numberInLine = 1;
```

```
  colonneCourant = 0;
  sumLine = 0;}
;

/*Défini l'apparence des colonnes*/
colonnes  :blancs colonne blancs SEPAR_COL  colonnes {$$= strcat($2, $5);}
| blancs colonne blancs {$$ = $2;}
;

/*colonne va lister tout ce que peut contenir une colonne (entier, reel, mot... */
/*colonne renvoie une chaine de caractères fourni par la fonction redigeColonne*/
colonne   : phrase  {$$ = redigeColonne($1, positionnement, 0,0 );
   numberInColonne[colonneCourant] = 0;
   numberInLine = 0;
   colonneCourant++;}
| ENTIER {$$ = redigeColonneEntier($1, positionnement, 0, 0);
   sumLine += $1;
   sumColonne[colonneCourant] += $1;
   colonneCourant++;}
| REEL {$$ = redigeColonneReel($1, positionnement, 0, 0);
   sumLine += $1;
   sumColonne[colonneCourant] += $1;
   colonneCourant++;}
| FUSION ACCOL_G ENTIER ACCOL_D ACCOL_G option ACCOL_D ACCOL_G phrase ACCOL_D {$$ =
redigeColonne($9, positionnement, 1, $3);
   changeTab(numberInColonne, colonneCourant, $3);
   colonneCourant += $3;
   numberInLine = 0;}
| FUSION ACCOL_G ENTIER ACCOL_D ACCOL_G option ACCOL_D ACCOL_G ENTIER ACCOL_D {$$ =
redigeColonneEntier($9, positionnement, 1, $3);
   changeTab(numberInColonne, colonneCourant, $3);
   colonneCourant += $3;
   sumLine += $9;
 }
| FUSION ACCOL_G ENTIER ACCOL_D ACCOL_G option ACCOL_D ACCOL_G REEL ACCOL_D {$$ =
redigeColonneReel($9, positionnement, 1, $3);
   changeTab(numberInColonne, colonneCourant, $3);
   colonneCourant += $3;
   sumLine += $9;
 }
;
```

```
/*phrase permet de gérer des suite de mots séparer par des blancs*/
phrase : mots  phrase {$$= strcat($1, $2);}
| mots        {$$ = $1;}
;

/*mots corresponds aux chaines des caractères composées de lettres ou de chiffres*/
/*et aux formules latex de la forme : {\d phrase} et \d{phrase} pour géré les symboles particuliers ou les mises en formes*/
/*On peut constater que la mise en forme est ignoré. Seule la phrase est recopiée*/
mots      : MOT  blancs   {$$ =strcat( $1,$2);}
| ACCOL_G ANTISLASH MOT SPACE phrase  ACCOL_D blancs{$$= strcat($5,$7);}
|ANTISLASH MOT ACCOL_G phrase ACCOL_D blancs {$$ = strcat($4,$6);}
|ANTISLASH MOT blancs{$$=strcat($2,$3);}
;

blancs    : blancs SPACE {$$ = strcat($2, $1);}
| {$$ ="";}
;

%%
void yyerror(char *s)
{
  fprintf(stderr,"yyerror : %s\n",s);
  return;
}

char tab[]= "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
4.01//EN\"   \"http://www.w3.org/TR/html4/strict.dtd\">\n<html>\n<head><head/>\n<body>";

/*redige le champ d'option du tableau html*/
void redigeOption( int border, char * pos)
{
  fprintf(yyout, "style=text-align: %s; width: 90%%; border=\"", pos);
  if (border)
    fprintf(yyout,"1");
  else
    fprintf(yyout,"0");
  fprintf(yyout,"\"cellpadding=\"2\" cellspacing=\"2\">\n<tbody>\n<tr>");
}

/*renvoie une chaine de caractère contenant le code d'une colonne en html*/
```

```
/*texte est une string correspondant à la valeur à mettre dans la colonne*/
/*pos l'alignement dans la colonne*/
/*si multicolumn est à 1 alors colspan indique la taille de colspan*/
char* redigeColonne(char* texte, char* pos, int multicolumn, int colspan)
{
 char* retour = malloc(sizeof(char)*1000);
 if(multicolumn)
   {
    sprintf(retour,"<td style=\"text-align: %s;\" colspan=\"%d\" rowspan = \"1\">%s</td>", pos, colspan, texte);
   }
 else
   {
    sprintf(retour,"<td style=\"text-align: %s ;\">%s</td>", pos, texte);
   }
 return retour;
}

/*idem que redigecolumn mais pour les reels*/
char* redigeColonneReel(double val, char* pos, int multicolumn, int colspan)
{
 char* retour = malloc(sizeof(char)*1000);
 if(multicolumn)
   {
    sprintf(retour,"<td style=\"text-align: %s;\" colspan=\"%d\" rowspan = \"1\">%.2f</td>", pos, colspan, val);
   }
 else
   {
    sprintf(retour,"<td style=\"text-align: %s;\">%.2f</td>", pos, val);
   }
 return retour;
}
/*idem que redigecolumn mais pour les entiers*/
char* redigeColonneEntier(int val, char* pos, int multicolumn, int colspan)
{
 char* retour = malloc(sizeof(char)*1000);
 if(multicolumn)
   {
    sprintf(retour,"<td style=\"text-align: %s;\" colspan=\"%d\" rowspan = \"1\">%d</td>", pos, colspan, val);
   }
 else
```

```
    {
      sprintf(retour,"<td style=\"text-align: %s;\">%d</td>", pos, val);
    }
  return retour;
}

/*Permet d'initialiser numberInColonne*/
void initialiseTabBool(int* tab, int size)
{
  int i;
  for(i = 0; i<size; i++)
    {
      tab[i] = 1;
    }
}

/*Permet d'initialiser sumColonne*/
void initialiseTabReel(float* tab, int size)
{
  int i;
  for(i = 0; i<size; i++)
    {
      tab[i] = 0;
    }
}

/*permet de changer les valeurs de numberInColonne dans le cas d'une multicolumn*/
void changeTab(int* tab, int deb, int quantity)
{
  int i;
  for(i = deb; i<deb+quantity; i++)
    {
      tab[i] = 0;
    }
}

/* Fonction ajoutant la colonne de somme d'une ligne*/
void ajouteColonne(char* chaine, char* position, float num)
 {
   char* aCopier=malloc(100* sizeof(char));
```

```
  sprintf(aCopier,  "<td style=\"text-align: %s;\">%.2f</td>",position, num);
  strcat(chaine, aCopier);
  free(aCopier);
 }

/* Fonction d'ajouter la somme des colonnes en fin de tableau*/
void sommeColonne(char* chaine, char* position, int* appliqueSomme, float* toutesSommes, int nbFois)
 {
  int i, bool;
  bool=1;
  sprintf(chaine , "");
  char* aCopier;
  for(i=0;i<nbFois;i++)
   {
        bool *= appliqueSomme[i];
   }
  if (bool)
   {
        strcat(chaine, "<tr>");
        for (i=0;i<nbFois;i++)
         {
          if (appliqueSomme[i])
           {
                aCopier=malloc(50* sizeof(char));
                sprintf(aCopier,"");
                ajouteColonne(aCopier, position, toutesSommes[i]);
                strcat(chaine, aCopier);
                free(aCopier);
           }
          else
           strcat(chaine, "<td></td>");
         }
        strcat(chaine, "</tr>\n");
   }
 }

int main(int argc, char* argv[])
{
 if (argc >1)
  {
```

```
    yyin=fopen(argv[1],"r");
    if (!yyin)
            {
             printf("error when open input file\n");
             exit(1);
            }
    if (argc >2)
            {
             yyout=fopen(argv[2], "w");
            }
    if(!yyout)
            {
             printf("error when open output file\n");
             exit(1);
            }
    fprintf(yyout, "%s", tab);
    if (!yyparse() )
            {
             fclose(yyin);
             fclose(yyout);
            }
    else
            {
             printf("parse error\n" );
             exit(1);
            }
  }
 else
  {
    printf("call : ./exo3 texxte.txt...");
    exit(1);
  }
}
```

5.2.3 Fichier de compilation

Le fichier de compilation est un fichier bash :

```
#!/bin/sh


rm *.c *.h
```

```
rm proj
echo "compile lex"
lex projet_compilation.l
echo "compile yacc"
yacc --report=states -d projet_compilation.y
mv lex.yy.c projet_compilation_lex.c
mv y.tab.c projet_compilation_syn.c
mv y.tab.h projet_compilation_syn.h
echo "compile yacc"
cc -o proj projet_compilation_lex.c projet_compilation_syn.c -lfl
```