Analyse syntaxique : Mettaton Synth-EX-ique

Lilian CHAMPROY, Axel DUBROCA, Manon-Julie PINTAULT, Carlos NEZOUT

30 avril 2016

Table des matières

1 Introduction **2**
1.1 Dépôt git 2
1.2 Attribution des rôles 2

2 Analyse syntaxique du fichier **3**
2.1 Grammaire utilisée 3

3 Difficultés rencontrées **6**
3.1 Gestion du temps 6
3.2 Incompréhensions au niveau du sujet 6
3.3 Web statique ??? 6
3.4 Section interne au projet : partie théorie 6

Chapitre 1

Introduction

1.1 Dépôt git

Le dépôt git est situé à cette adresse : https ://github.com/Pyraexyrin/Mettaton-syntEXique

1.2 Attribution des rôles

Axel DUBROCA : Théorisation
Manon-Julie PINTAULT : Programmation
Carlos NEZOUT : Programmation
Lilian CHAMPROY : Rédaction du rapport

Chapitre 2

Analyse syntaxique du fichier

2.1 Grammaire utilisée

La grammaire à pu être représentée par l'arbre suivant, sur un des exemples dans le sujet du projet :

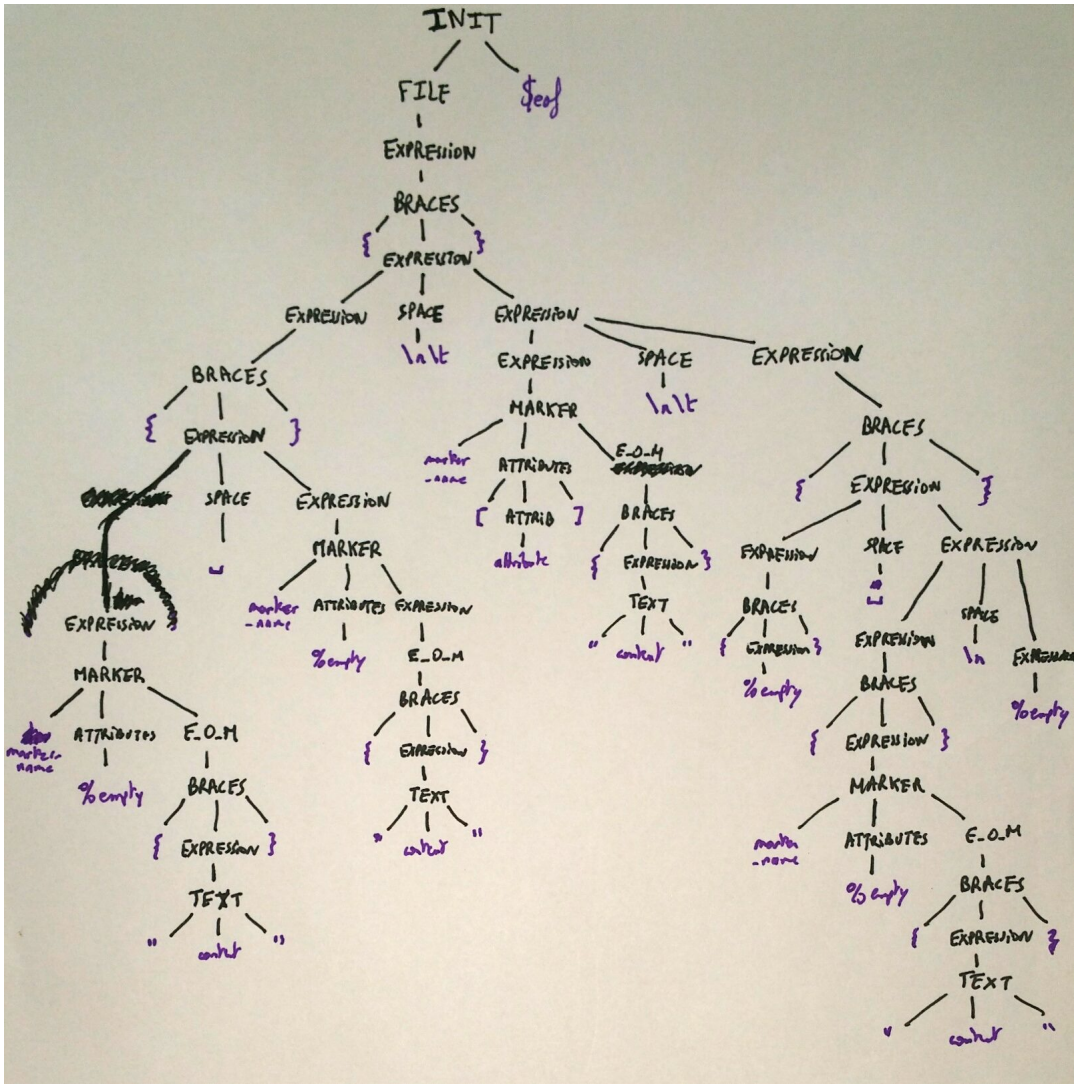

FIGURE 2.1

Le fichier lex main.l va servir d'analyseur syntaxique. Et va parser chaque mot du fichier donné pour ainsi print les balises en format adéquat HTML.
La grammaire prend normalement en compte les "\" Ainsi que les "\ \ "

Nous l'avons ensuite créée sous la forme suivante :

```
%token E_O_F TEXT_CONTENT ATTRIBUTE SPACE
%token XML_FREE_TEXT
%start init

%%

// Règle initiale

init  : expr_set E_O_F
;

// Cas du fichier vide
// Puisque la balise <body> est posée, on suppose pouvoir
// mettre directement du texte

expr_set  : expr_set SPACE expression
     | expression
     | %empty
;

// Expressions principales

expression  : marker
    | braces
    | text
;

// Balises

marker  : XML_FREE_TEXT attributes end_of_marker
;

// Attributs

attributes  : '[' attrib ']'
    | %empty
;

attrib  : attrib ',' attrib_alt
| attrib ',' SPACE attrib_alt
        | attrib_alt
;

attrib_alt : text_content '=' text
;

// Fin de balise
end_of_marker  : braces
       | '/'
;

// Accolades isolées (indépendantes d'une balise)
// Suffit par réductabilité d'expression

braces  : '{' expr_set '}'
;
```

```
// Texte simple

text  : '"' text_content '"'
;

text_content  : TEXT_CONTENT
| XML_FREE_TEXT
;
```

Chapitre 3

Difficultés rencontrées

Nous avons rencontré de nombreuses difficultés durant le projet.

3.1 Gestion du temps

Nous avons commencé trop tardivement le projet. La formation de l'équipe s'est faite assez tard, de plus nous n'avons réellement commencé à travailler sur le projet que récemment.
A la fois à cause des examens se rapprochant, ou d'autres projets à réaliser en parallèle, mais aussi à cause d'obligations personnelles.

3.2 Incompréhensions au niveau du sujet

Autre fait qui ait pu nous retarder, c'est la compréhension du sujet.
Nous avons eu du mal à pouvoir commencer efficacement à avancer dès le début du projet. Si nous pensions que seul un fichier bison aurait pu suffire, nous avons plutôt changer de cap par la suite.

3.3 Web statique ???

A l'annonce du projet, nous étions suspicieux sur une partie de ce dit projet.
Nous n'avons, pour la plupart d'entre nous, eu aucune formation pour ce qui est HTML/CSS/etc...
Ainsi nous avons pour certains d'entre nous été déstabilisé sur une partie du sujet. Notamment sur la partie "Fonctionnalités à ajouter".
Qui plus est, en dehors de l'aspect projet, nous estimons dommage que ce projet ne soit efficient uniquement que sur du web statique.

3.4 Section interne au projet : partie théorie

La première difficulté, c'est d'évidemment trouver une grammaire. Il n'y a pas d'algorithme, c'est uniquement du feeling. Déjà, certains n'ont toujours pas compris la distinction entre analyse lexicale et syntaxique, qui, honnêtement, est tellement bien explicitée en cours que certains n'ont toujours pas compris. Comme si dire DICTIONNAIRE et RÈGLES GRAMMATICALES était compliqué. Dans notre groupe, ce n'est pas arrivé, mais j'ai pu constater que chez certains, si. Abbération.

Mais à cela s'ajoute le *ressenti*, le *feeling*, permettant de dégager une grammaire. Et, en plus, il faut la décrire en bison. Et, en plus, il faut résoudre tous les conflits réduction/réduction et décalage/réduction. Et, en plus, il faut la tester. Oh ! Mais on ne peut pas.

Parce que, deuxième difficulté, bison et flex sont inextricables. On ne peut pas tester l'un sans l'autre. Et, de plus, la sortie de bison est ignorée. Inutile de faire des printf/fprintf ou autre dans le fichier yacc. Il y a toujours moyen de tricher, comme toujours, mais ce n'est pas une feature de base. C'est toujours la

sortie de flex qui sera affichée. Ce qui est relativement inutile. Ah, si ! On a les terminaux fies de bison. C'est au moins ça... On peut vérifier d'une certaine manière notre grammaire. Mais pas l'arbre syntaxique.

Parce que, troisième difficulté : l'arbre syntaxique. Afficher un simple arbre syntaxique relève de l'érudition. Sérieusement ? Je reviendrai sur les spécificités de flex/bison après, j'ai d'autres remarques.

Quatrième difficulté : survivre aux erreurs, changements et "oublis" (volontaires j'espère) du sujet. C'est loin des cahiers des charges dont on rêve tous. Il y a tellement d'ambigüités que le projet peut-être fait de 36 façons différentes. Est-ce mal ? Non, sauf quand on nous donne deux semaines avant le rendu la démarche attendue. Les structures de données, par exemple. C'est un choix à faire dès le début : on les donne, ou non ? Et outre ces structures, certaines difficultés sont annoncées par le sujet... Qui n'existent même pas ! Si notre grammaire est bien faite, le champ *space* de la structure ne sert absolument à rien, sinon induire en erreur ! Mais bref, je ne suis pas là pour mal parler de l'UE, j'aurai l'occasion de le faire dans l'évaluation des enseignements.

Autre difficulté, je reviens sur flex/bison. COMMENT CE FAIT-CE QU'ON NE PUISSE PAS CHANGER SIMPLEMENT LE TYPE DE YYLVAL. Sérieusement. Qui a créé ces langages ? Il est É-VI-DENT que nous avons besoin de deux choses en retour du flex : le type de donnée (donnée par le return - qui met fin à yyparse, bien pensé !), et la donnée. Donc, je m'attends à avoir mon int, et un char*. Et bien non. Trop facile. Plutôt que laisser les convertions à l'utilisateur, non, autant tout faire salement en bison/flex, je ne sais même plus quel langage gère quoi. C'est excessivement mal interfacé. Je le redis, ils sont tellement inextricables que le bon sens ne suffit plus. Et me parlons pas des #include magiques qui font un peu leur vie. D'où je dois avoir besoin de déclarer yyerror et yylex ?

En fait, le plus énervant, ce n'est pas la théorie, qui me semble être l'essence de cette UE. C'est le code. Comme toujours. Cette UE, c'est pas une introduction à la compilation, mais un cours de flex et de bison. On passe un semestre entier sur deux langages, quand le cours privé de flex/bison tiendrait en moins d'une heure. Hello, l'équilibre ? Et où est la préparation à la compilation ? Car, de mes souvenirs, l'UE de compilation se fiche de flex et bison. Donc, introduction vers quoi ? Vers une prochaine UE, vers la compilation de tous les jours, vers la pertinence ?

Bien. Je pense avoir été clair. La difficulté aurait du venir des grammaires : la partie fonctionnelle est un véritable défi. Mais ce n'est pas le cas. Ce n'est qu'une question d'implémentation. Impertinent. Alors que je voulais travailler sur la couche théorique, au rythme des développeurs, c'était finalement inutile, car ce n'était pas la partie complexe du projet. Et c'est pour moi une infinie déception. Je n'aime pas coder, et j'ai du essayer quand même, m'énerver contre des illogismes parce que le tout est déséquilibré.

Mais ainsi soit-il. Je n'ai pas envie de fournir d'effort supplémentaire. Bien d'autres choses méritent plus mon attention. Je conclurai donc, comme d'habitude, par quelques mots complètement impertinents :

Lights ! Camera ! Action !
Drama ! Romance ! Bloodshed !
I'm the idol everyone craves !
Smile for the camera !