**Langages et Compilation**

***Mini projet***

TORTEL Paul – PILAUDEAU Matthieu – COURSAC Vincent

Groupe F – F#

Table des matières

[Bilan 1](#_Toc477959779)

[Fichier d’entrée 1](#_Toc477959780)

[Structures de données 2](#_Toc477959781)

[Algorithmes 2](#_Toc477959782)

[Traces d’exécutions 3](#_Toc477959783)

# **Bilan**

# **Fichier d’entrée**

Pour représenter une grammaire dans notre programme, il faut la décrire dans un fichier texte correspondant en respectant le format que nous avons choisis. Ainsi, nous avons un fichier par grammaire.

Chaque ligne du fichier texte décrit un symbole non terminal. Exemple :

Expreb = Expreb ou Termeb | Termeb

Cette ligne peut se décomposer de la manière suivante :

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Expreb** |  | **=** |  | **Expreb** |  | **ou** |  | **Termeb** |  | **|** |  | **Termeb** |

*Chaque expression ou operateur est séparé par un espace. Cela facilite la lecture du fichier et clarifie la représentation.*

Il y a deux *opérateurs* gérés dans notre format : **=** et **|**

**=** permet de représenter la flèche 🡪, qui a pour but d’indiquer quel est le non terminal décrit par les règles situées à droite de cet opérateur.

**|** permet de représenter l’opérateur logique « OU », qui permet de créer plusieurs règles pour un symbole non terminal. Dans notre exemple, le symbole Expreb possède donc deux règles :

**Expreb 🡪 Expreb ou Termeb OU Expreb 🡪 Termeb**

Deuxième règle

Première règle

**Exemple concret**

Représentation de la grammaire suivante :

**Représentation théorique**

S 🡪 (L) | a

L 🡪 L,S | S

**Représentation dans notre format texte**

S = ( L ) | a

L = L , S | S

***Représentation en tableaux***

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S** |  | **=** |  | **(** |  | **L** |  | **)** |  | **|** |  | **a** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **L** |  | **=** |  | **L** |  | **,** |  | **S** |  | **|** |  | **S** |

# **Structures de données**

# **Algorithmes**

## **Lecture du fichier**

La lecture et l’interprétation du fichier texte repose sur un algorithme simple. Il suffit de lire ligne par ligne le fichier et d’analyser chaque ligne grâce à des méthodes simples.

L’analyse repose sur le principe de « split », c’est-à-dire transformer une chaine de caractères en un tableau de sous chaines de caractères en utilisant un symbole pour effectuer le découpage. Par exemple, si on split la chaine « A-B-C-D » par le symbole ‘ **-** ‘, on obtient un tableau de taille 4.



|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | D |

Split par ‘ – ‘

Grâce à ce procédé, on peut dégager des étapes pour effectuer l’analyse de la ligne.

Prenons comme exemple pour la suite de l’explication la ligne suivante

S = ( L ) | a

|  |  |
| --- | --- |
| **S** | **( L ) | a** |

**1. Split par ‘ = ‘**

*On a dans un tableau de taille 2 d’un côté le non terminal qui est en train d’être défini, et de l’autre ses règles qui attendent d’être analysées*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **(** | **L** | **)** | **|** | **a** |

**2. Split par ‘ ‘**

*On a split la partie de droite du tableau obtenu à l’étape1. On obtient un nouveau tableau contenant toutes les sous chaines de la ligne. Il faut maintenant traiter l’opérateur* ***|***

|  |
| --- |
| a |

|  |  |  |
| --- | --- | --- |
| ( | L | ) |

**3. Création des règles**

## **Elimination de la récursivité à gauche**

# **Traces d’exécutions**