EFREI 2016/2017 - L3 - Langages et Compilation

Mini projet

L'objectif principal du mini projet est de réaliser un analyseur descendant prédictif sur la base d'une grammaire donnée en entrée.

Conditions de travail

Travail par équipes de 3 ou 4 selon le découpage suivant :

Groupe E (39 étudiants) : 13 équipes de 3
Groupe F (40 étudiants) : 12 équipes de 3 / 1 équipes de 4
Groupe G (38 étudiants) : 10 équipes de 3 / 2 équipes de 4
Groupe H (28 étudiants) : 8 équipes de 3 / 1 équipe de 4

Les délégués de groupes doivent me fournir la constitution des équipes au plus tard au début de la 1ère séance de TP, en donnant un numéro à chaque équipe (E1 à E13, F1 à F13, G1 à G12, H1 à H9). Merci d'identifier les étudiants par leurs noms de famille, et non pas par leurs prénoms.

Programme à développer en C ou C++.

La date limite de rendu du travail vous sera communiquée ultérieurement.
Vous devrez envoyer un email à l'adresse :

herve.barbot@efrei.fr

en respectant les règles suivantes :

- le titre de votre email doit être « L3-LC-<numéro équipe> » ;
- en pièce jointe, vous devez mettre :
  - votre code source,
  - vos fichiers de données (test),
  - votre rapport au format PDF.

Aucun fichier archive (zip, rar, …) ne sera accepté. Tous vos fichiers doivent être directement en pièces jointes. Lors de la compilation / exécution de votre programme, tous vos fichiers doivent pouvoir être installés dans le même répertoire (n'utilisez aucun sous-répertoire !).
Tous les fichiers joints doivent être préfixés par votre numéro d'équipe (par exemple : F3main.cpp, F3rapport.pdf, …).

Le rapport que vous devez fournir doit contenir les informations suivantes :

- bilan de ce qui est réalisé,
  Tableau synthétique, en indiquant ce qui fonctionne, ce qui a été codé mais ne fonctionne pas correctement, ce qui n'a pas été fait.
- description de la structure du fichier en entrée,
- description des structures de données utilisées pour représenter les différentes informations nécessaires,
  Code C/C++ et schémas. Vous pouvez utiliser un exemple pour illustrer votre description.
- les algorithmes mis en œuvre,
  Pseudo-code ou schémas « logigramme ». Pas de code C/C++ à ce niveau.
- les traces d'exécution pour les jeux de test.
  Les jeux de test à utiliser vous seront fournis ultérieurement.

Evaluation / Notation du travail effectué

L'évaluation portera sur les points suivants :

- ce qui fonctionne ou pas (vérifié au travers des traces que vous fournirez, ainsi que par l'analyse de votre code et son exécution),
- vos choix de mise en œuvre (structures de données choisies, algorithmes utilisés, …),
- la lisibilité de votre code source,
- la qualité de votre rapport (structure, lisibilité, clarté/précision des informations fournies).

Réalisation

Grammaires à prendre en compte

Votre programme doit pouvoir traiter des grammaires respectant les caractéristiques suivantes :

- les terminaux sont des caractères : lettres, chiffres, symboles tels que '+', '-', … ;
- les non terminaux sont des lettres majuscule de l'alphabet.

Les symboles spéciaux utilisés pour la description de la grammaire dans un fichier texte ne peuvent pas être utilisés en tant que terminaux.

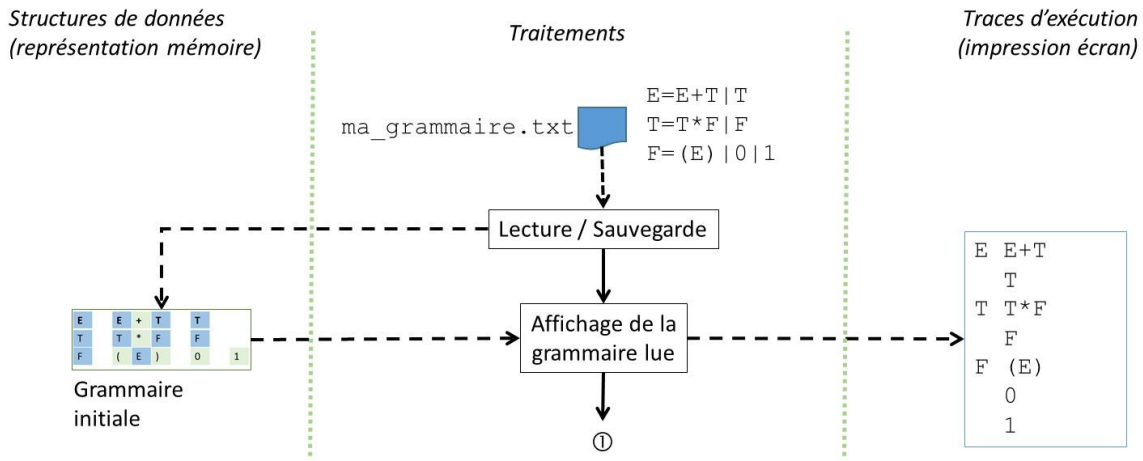
Lecture de la grammaire initiale

Votre programme débute par la lecture d'une grammaire décrite dans un fichier au format « texte ».
Vous êtes libres de la structure de ce fichier. Il vous est conseillé de faire « simple ».
Par exemple, une grammaire peut être décrite sous la forme :

```
E=E+T|T
T=T*F|F|#
F=(E)|0|1
```

On peut déterminer aisément que E,T et F sont les non terminaux, alors que les symboles '(', ')', '*' et '+' ainsi que les chiffres '0' et '1' sont les terminaux (ils n'apparaissent pas en partie gauche d'une règle de production).
Le premier non-terminal défini (première règle de production) est considéré comme étant l'axiome de la grammaire.
Les symboles '=', '|' et '#' sont ici considérés comme des symboles spéciaux et ne font pas partie du langage : '=' identifie une règle de production, '|' délimite deux règles de production définies pour un même symbole non terminal, '#' remplace le 'ε' (qui est difficile à saisir dans un fichier texte avec un outil tel que le « bloc-note » de MS Windows). Vous pouvez bien entendu utiliser n'importe quels symboles spéciaux.

Le résultat doit être stocké en mémoire, dans les structures de données de votre choix. Une fonction reprend ensuite ce qui est stocké en mémoire pour l'afficher à l'écran (au format de votre choix).



Dans le schéma précédent, et dans les suivants, la colonne de droite représente les traces d'exécution de votre programme, et la colonne de gauche représente les structures de données utilisées.
Ce ne sont bien évidemment que des exemples. Vous pouvez utiliser le format que vous souhaitez.

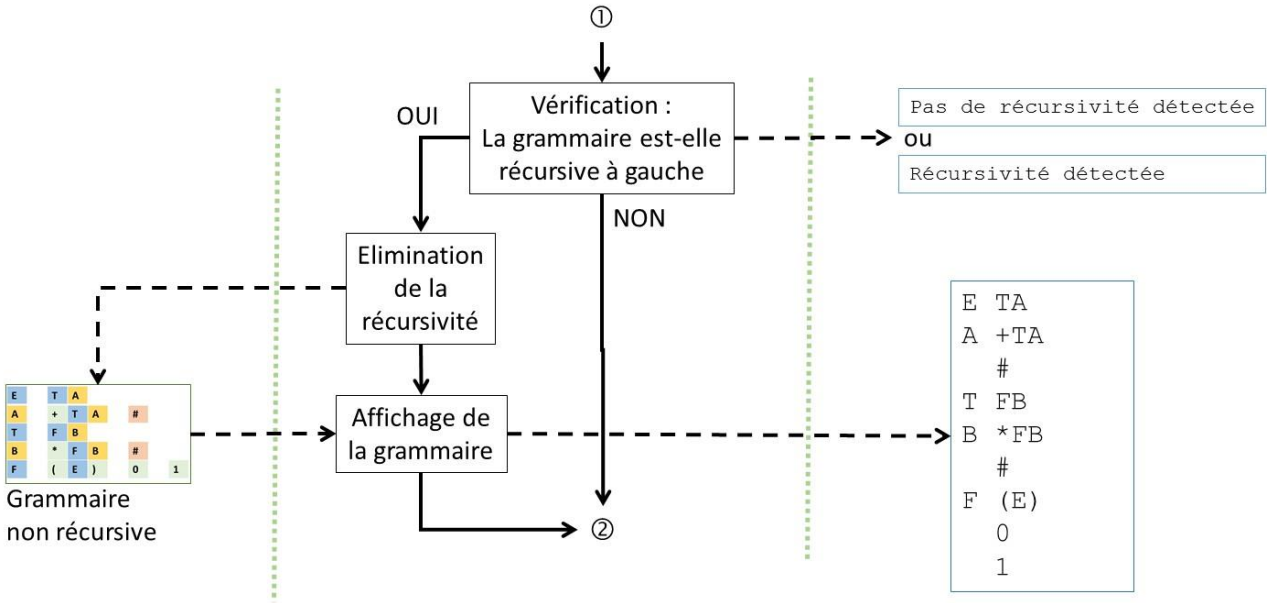
Remarques :
Votre structure de données doit vous permettre :

- d'identifier le symbole non terminal « axiome » de la grammaire,
- d'identifier tous les symboles, terminaux et non terminaux,
- d'associer à chaque symbole non terminal la liste des règles de production associées (chaque règle de production est définie comme une suite de symboles terminaux et/ou non terminaux, ou bien par la chaîne vide ε).

Rien ne vous empêche, si cela vous semble plus simple, d'ajouter d'autres informations dans votre fichier. Par exemple, on peut facilement ajouter en début de fichier : une ligne pour la définition des symboles terminaux, une ligne pour la définition des symboles non terminaux, une ligne pour l'identification de l'axiome.

Elimination de la récursivité à gauche

Votre programme doit ensuite identifier si la grammaire contient une définition récursive à gauche ou non. Si oui, la récursivité est éliminée.

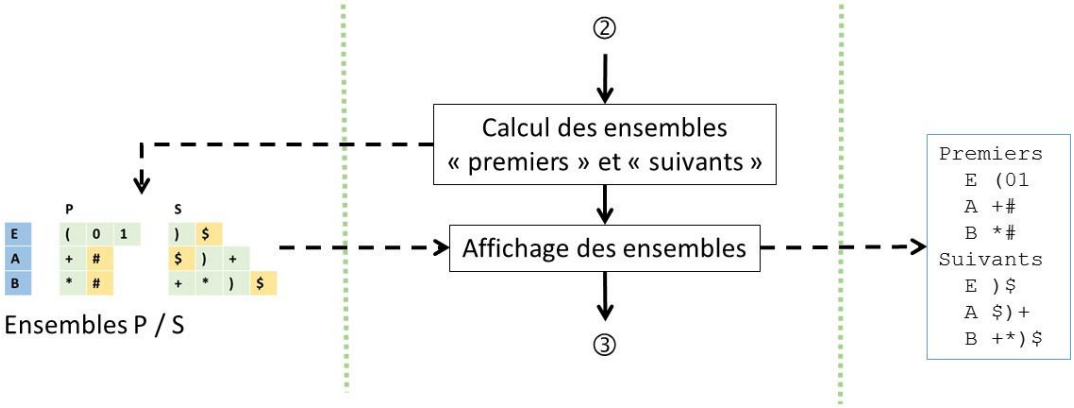


Point laissé à votre appréciation :

Bien que cela ne semble pas nécessaire, il n'est pas interdit de créer une nouvelle structure de données pour représenter une grammaire dans laquelle la récursivité a été enlevée.
Dans ce cas, si la grammaire ne contient pas de définition récursive, votre programme doit bien entendu en faire une copie dans cette seconde structure de données.

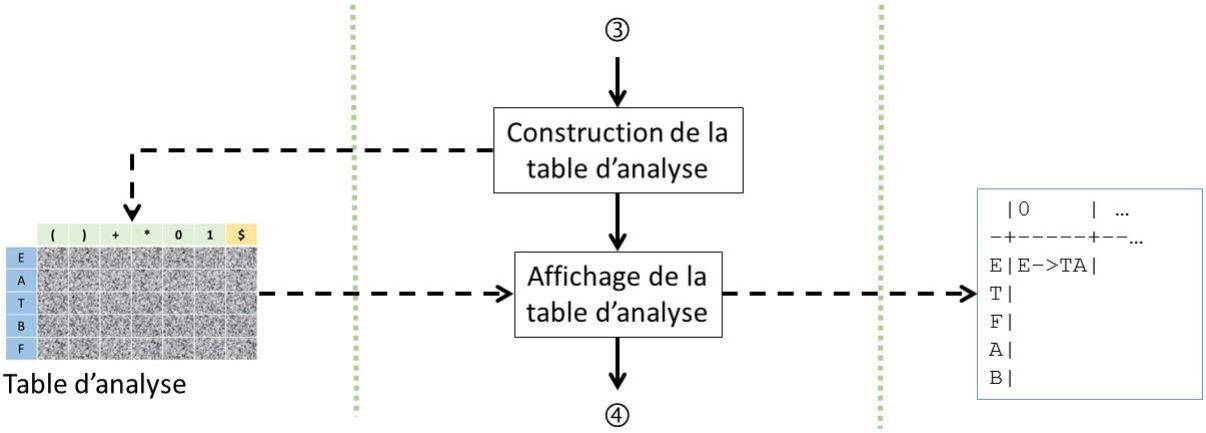
Calcul des ensembles « premiers » et « suivants »

Afin de produire un analyseur, il vous faut maintenant calculer, pour chaque dérivation, les ensembles « premiers » et « suivants ».
Le résultat est stocké dans des structures de données, là aussi « de votre choix ».
Ces ensembles sont ensuite affichés à l'écran.



Construction de la table d'analyse

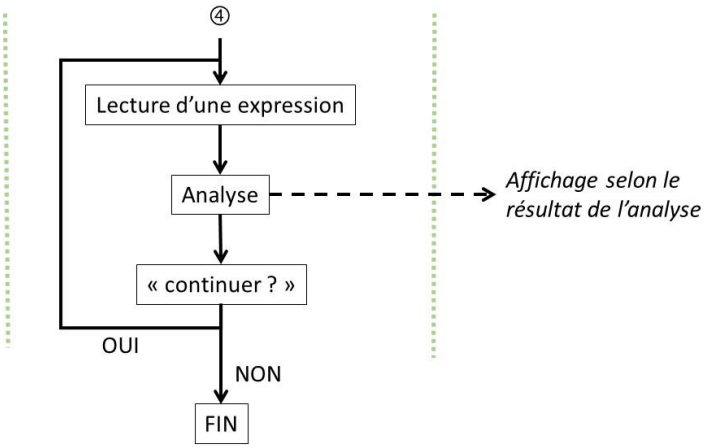
Ayant la grammaire non récursive ainsi que les ensembles « premiers » et « suivants », votre programme peut désormais construire la table d'analyse prédictive.
Le résultat est affiché à l'écran sous forme d'un tableau.



Analyse d'une chaîne en entrée

Dernière étape : votre programme lit en entrée (clavier) une expression et l'analyse. Il indique en résultat si elle correspond ou non à la grammaire.
A vous d'être aussi précis que possible dans l'identification du résultat : localisation de l'erreur en cas de non reconnaissance de l'expression.

Votre programme doit être capable de boucler sur la saisie de plusieurs chaînes en entrée, sans être obligé de redémarrer votre programme.



-- fin --