Travaux pratiques de mathématiques appliquées

Langages formels et grammaires

Durée prévue : 2 H

1. Introduction

1.1. Objectifs

Les objectifs principaux de ce laboratoire sont de :

- manipuler les notions de langages formels et de grammaires au moyen de l'outil JFLAP ;
- mettre en œuvre la notion de grammaire dans un programme qui implémente un générateur de phrases.

Ce laboratoire utilise et met en pratique les notions expliquées dans le chapitre 8 du cours théorique. L'étudiant est supposé avoir révisé la matière de ce chapitre avant le début du laboratoire !

1.2. Evaluation

Les séances de travaux pratiques ont un caractère essentiellement formatif. Les exercices proposés dans cet énoncé ne seront pas notés. Néanmoins, il est fondamental que vous tentiez d'en résoudre un maximum par vous-même, ceci afin d'acquérir une bonne compréhension et une bonne maîtrise de la matière du cours.

Si vous éprouvez des difficultés, n'hésitez pas à solliciter l'aide de votre responsable de laboratoire. Vous pouvez également collaborer avec vos condisciples pour concevoir les solutions. Assurez-vous cependant que vous avez compris les notions mises en pratique. Ne vous contentez pas de recopier une solution sans la comprendre : vous devez être capable de la recréer par vous-même.

1.3. JFLAP et JFLAP Book

Durant ce laboratoire, nous utiliserons l'outil JFLAP « Java Formal Languages and Automata Package » présenté lors du cours théorique pour explorer de manière interactive les notions de langages formels, grammaires et automates finis.

Les exercices proposés dans la section 2 sont tirés de l'édition 2006 du livre « JFLAP – An Interactive Formal Languages and Automata Package » de Susan H. Rodger et Thomas W. Finley. Nous nous référerons à ce livre en parlant du « JFLAP Book ».

Toutes les informations et ressources nécessaires concernant JFLAP sont disponibles sur l'espace HELMo Learn du cours.

2. Exercices avec JFLAP - Grammaires

2.1. Introduction

Cette partie du laboratoire est basée sur le chapitre 3 du livre « JFLAP Book ». Vous trouverez dans ce chapitre une description des outils proposés par JFLAP pour manipuler des grammaires.

- Section 3.1 – Création/édition d'une grammaire.
- Section 3.2 – Analyse par force brute ([Multiple] Brute Force Parser) pour déterminer une séquence de dérivation ou un arbre de dérivation.

Vous pouvez également regarder les vidéos introductives sur HELMo Learn.

Remarque
Par défaut, JFLAP utilise le symbole λ pour représenter le mot vide. Il est possible de modifier les préférences pour utiliser le symbole ε à la place.

2.2. Visualisation de la séquence de dérivation et de l'arbre de dérivation d'un mot appartenant au langage engendré par une grammaire

L'outil JFLAP vous permet d'explorer le comportement des productions d'une grammaire et de déterminer si un mot peut être généré ou pas par celle-ci.

En particulier, ***lorsqu'un mot fait partie du langage engendré par la grammaire***, JFLAP vous permet de visualiser la séquence de dérivation ou l'arbre de dérivation correspondant.

Séquence de dérivation / arbre de dérivation
• Chargez la grammaire régulière définie dans le fichier `ex3.1a`. Définissez-la. • Dans le menu « Input », sélectionnez « Brute Force Parse ». • Dans le champ « Input », écrivez « aqvx ». • Sélectionnez le mode « Derivation Table » dans la liste déroulante « Noninverted Tree ». Démarrez l'analyse (« Start »). *NB : un message indiquant si la chaîne saisie est acceptée ou non, apparaît en dessous du champ « Input ».* • Cliquez sur « Step » pour faire apparaître, une à une, les différentes étapes de la dérivation. *NB : un message commentant l'étape réalisée et indiquant que le processus est terminé apparaît dans le bas de la fenêtre.* • Sélectionnez à présent le mode « Noninverted Tree » ; l'arbre de dérivation correspond est présenté. • Entrez « bab » dans le champ « Input » ; qu'observez-vous ? • Testez d'autres chaînes de votre choix. • Répétez l'exercice avec la grammaire définie dans le fichier `ex6.1a`.

2.3. Déterminer le langage engendré par une grammaire

Étant donné une grammaire, êtes-vous capable de déterminer le langage qu'elle engendre ?
Quels sont les mots qui font partie du langage engendré et ceux qui n'en font pas partie ?
Plus globalement, quelle sera la forme générale des mots engendrés par la grammaire ?

Exercice 3.1 du livre « JFLAP Book »

- Chargez les grammaires régulières définies dans les fichiers listés ci-dessous.
  a) `ex3.reggrm-a`
  b) `ex3.reggrm-b`
  c) `ex3.reggrm-c`
- Pour chaque grammaire, déterminez 5 mots qui font partie du langage engendré par la grammaire et 5 mots qui n'en font pas partie.
  [Suggestion : utilisez l'outil « Multiple Brute Force Parse » de JFLAP pour vous guider.]
- Donnez une description en français du langage engendré par la grammaire, c'est-à-dire de la forme générale des mots générés.
- Donnez une description mathématique de $L(G)$ et une expression régulière ER_G.

Exercices supplémentaires :

d) `ex3.reggrm-d`
e) `ex3.reggrm-e`
f) `ex3.reggrm-f`

2.4. Concevoir une grammaire qui engendre un langage donné

Cet exercice est l'inverse du précédent. Étant donné la description des mots d'un langage, pouvez-vous concevoir une grammaire régulière qui permet d'engendrer ce langage ?

Ici encore, JFLAP vous permet de valider l'exactitude de votre grammaire sur base d'une liste de mots dont vous savez à l'avance s'ils doivent être générés ou pas par la grammaire.

Exercice 3.2 du livre « JFLAP Book »

- Créez une grammaire régulière qui engendre les langages sur l'alphabet $\Sigma = \{a, b\}$ décrits ci-dessous.
  [Suggestion : utilisez l'outil « Multiple Brute Force Parse » de JFLAP pour valider le bon comportement de votre grammaire.]
  a) Le langage des mots contenant un nombre pair de 'a' et exactement un 'b'.
  Exemples : les mots aba, b, aaaabaa et aaaba sont acceptés ; les mots abab et aaab sont rejetés.
  b) Le langage des mots contenant un nombre pair de 'a' suivis par trois 'b' au maximum.
  Exemples : les mots aab, bbb, aaaabb et aaaa sont acceptés ; les mots aabbbb et aabaab sont rejetés.

c) Le langage des mots dans lesquels un ‘b’ ne peut pas être adjacent à un autre ‘b’.
Exemples : les mots aba, b, aabaaabaa et baaaba sont acceptés ; les mots aaabba et babb sont rejetés.

Exercices supplémentaires :

d) Le langage des mots dans lesquels le nombre de ‘a’ est un multiple de 3.
Exemples : les mots ababa, bb, aaa et aababb sont acceptés ; les mots aababa et aaaaba sont rejetés.

e) Le langage des mots dans lesquels chaque ‘a’ doit être entouré par (au moins) un ‘b’ de chaque côté.
Exemples : les mots babab, bab, bbab et babbbabb sont acceptés ; les mots abab et baab sont rejetés.

f) Le langage des mots dans lesquels un ‘a’ sur deux (en démarrant avec le premier ‘a’) doit être immédiatement suivi par au moins un ‘b’.
Exemples : les mots babaab, bab, abaabbabab et abbaab sont acceptés ; les mots ababa et aab sont rejetés.

2.5. Transformer une grammaire algébrique en une grammaire régulière

Nous avons vu au cours théorique que les grammaires algébriques (type 2), aussi appelées « hors contexte » (« context-free » en anglais), proposent des productions plus complexes que les grammaires régulières (type 3). À ce titre, elles permettent d’engendrer des langages possédant une structure plus complexe que les grammaires régulières. Néanmoins, si le langage est régulier, il est possible de transformer la grammaire algébrique en une grammaire régulière équivalente. C’est le but de cet exercice.

Exercice 3.4 du livre « JFLAP Book »

- Chargez les grammaires définies dans les fichiers listés suivants :
  a) `ex3.isregular-a`
  b) `ex3.isregular-b`
- Pour chaque grammaire, déterminez si la grammaire est régulière ou pas.
  *[Note : la fonctionnalité « Test for Grammar Type » de JFLAP vous permet de **vérifier** si votre réponse est correcte.]*
- Si la grammaire n’est pas régulière, transformez-la en une grammaire régulière qui engendre le même langage. Assurez-vous de l’équivalence des deux grammaires en vérifiant que les mots acceptés ou rejetés sont les mêmes !

Exercices supplémentaires :

c) `ex3.isregular-c`
d) `ex3.isregular-d` <!>
e) `ex3.isregular-e`

Attention, il y a une erreur dans le fichier original provenant de JFLAP.org pour l'exercice d.
En effet, telle que fournie, la grammaire ne permet pas de se débarrasser du non-terminal S.
Il faut soit ajouter une production $S \rightarrow \varepsilon$, soit modifier la 1ère production en $S \rightarrow AB$.

3. Générateur de phrases

Lors du cours théorique, l'exemple choisi pour illustrer la définition d'une grammaire, était la formation de phrases (simples).

Dans cet exercice, nous allons implémenter un générateur de phrases, sur base d'une grammaire formelle, pour laquelle les productions sont plus complexes.

Voici ces productions :

1. <phrase> -> <sujet> <verbe> | <sujet> <verbe> <complément>
2. <sujet> -> <sujet> <qualificatif> | <sujet> qui <verbe> <complément> |
   Le chat | Le chien | Le lion | Une gazelle | Un singe
3. <prénom> -> <prénom> - <prénom> |Alberte | Isidore | Suzette | Joséphine | Marcel
4. <qualificatif> -> rose | à pois | grimaçant | hirsute
5. <verbe> -> <verbe> <adverbe> | court | mange | bondit | sautille | dort
6. <adverbe> -> vite | lentement | joyeusement | bruyamment
7. <complément> -> <complément> de <prénom> |
   dans le jardin | dans le bois | sur le divan | dans la savane | sous le lit

NB : les symboles **non-terminaux** sont indiqués entre chevrons (<mot>).

Exercice 3 – Générateur de phrases

- Téléchargez le projet de départ « 2425_B1MATH_LF_GRM_FSM.zip » mis à votre disposition sur HELMo Learn et importez-le dans Eclipse.
- Dans la classe « GenerateurDePhrases » du package « `grammaire` », complétez la méthode suivante :

  `public static String deriver(String phrase, String symbole, int compteur)`

  → Dérive le symbole `symbole` en choisissant aléatoirement une production convenable, incrémente le `compteur` de dérivations et renvoie la `phrase` modifiée (substitution selon la production sélectionnée).

- La classe « GenerateurDePhrases » définit les règles de la grammaire, ainsi que la constante NB_DERIVATIONS_MAX → 5.
- La méthode `public static String genererPhrase()` (déjà complétée) initialise le compteur de dérivations et démarre le processus récursif de dérivation.
- Testez que votre méthode fonctionne correctement en exécutant la fonction principale fournie.

Exemples d'exécution :

```
Dérivation initiale : <phrase>
  Dérivation après remplacement de <phrase> : <sujet> <verbe> <complément>
    Dérivation après remplacement de <verbe> : <sujet> bondit <complément>
    Dérivation après remplacement de <sujet> : <sujet> <qualificatif> bondit <complément>
      Dérivation après remplacement de <qualificatif> : <sujet> grimaçant bondit <complément>
      Dérivation après remplacement de <sujet> : Le chien grimaçant bondit <complément>
    Dérivation après remplacement de <complément> : Le chien grimaçant bondit sous le lit
===> Phrase complète : Le chien grimaçant bondit sous le lit
```

```
Dérivation initiale : <phrase>
  Dérivation après remplacement de <phrase> : <sujet> <verbe> <complément>
    Dérivation après remplacement de <complément> : <sujet> <verbe> dans le jardin
    Dérivation après remplacement de <verbe> : <sujet> dort dans le jardin
    Dérivation après remplacement de <sujet> : Un singe dort dans le jardin
===> Phrase complète : Un singe dort dans le jardin
```

```
Dérivation initiale : <phrase>
  Dérivation après remplacement de <phrase> : <sujet> <verbe>
    Dérivation après remplacement de <verbe> : <sujet> <verbe> <adverbe>
      Dérivation après remplacement de <adverbe> : <sujet> <verbe> bruyamment
      Dérivation après remplacement de <verbe> : <sujet> mange bruyamment
    Dérivation après remplacement de <sujet> : <sujet> qui <verbe> <complément> mange bruyamment
      Dérivation après remplacement de <sujet> : Le lion qui <verbe> <complément> mange bruyamment
      Dérivation après remplacement de <complément> : Le lion qui <verbe> sous le lit mange bruyamment
      Dérivation après remplacement de <verbe> : Le lion qui <verbe> <adverbe> sous le lit mange bruyamment
        Dérivation après remplacement de <adverbe> : Le lion qui <verbe> joyeusement sous le lit mange bruyamment
        Dérivation après remplacement de <verbe> : Le lion qui sautille joyeusement sous le lit mange bruyamment
===> Phrase complète : Le lion qui sautille joyeusement sous le lit mange bruyamment
```

```
Dérivation initiale : <phrase>
  Dérivation après remplacement de <phrase> : <sujet> <verbe> <complément>
    Dérivation après remplacement de <complément> : <sujet> <verbe> sur le divan
    Dérivation après remplacement de <verbe> : <sujet> bondit sur le divan
    Dérivation après remplacement de <sujet> : <sujet> <qualificatif> bondit sur le divan
      Dérivation après remplacement de <qualificatif> : <sujet> grimaçant bondit sur le divan
      Dérivation après remplacement de <sujet> : <sujet> qui <verbe> <complément> grimaçant bondit sur le divan
        Dérivation après remplacement de <sujet> : <sujet> qui <verbe> <complément> qui <verbe> <complément> grim
          Dérivation après remplacement de <sujet> : <sujet> <qualificatif> qui <verbe> <complément> qui <verbe>
===> Le nombre maximum de dérivations est atteint.
```

Vous pouvez maintenant modifier et/ou ajouter des règles à la grammaire.

Bon travail !