

Session S4 info

APP 5

Semaines 11 à 13

**Développement d'un analyseur syntaxique
d'expressions arithmétiques**

Département de génie électrique et de génie informatique

Faculté de génie

Université de Sherbrooke

Hiver 2024

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner toutes les personnes.

Document GuideEtudiant.doc

Version précédente : 9 février 2023, faite par Ahmed Khoumsi

Version actuelle : 8 mars 2024, mise à jour par Ahmed Khoumsi

Copyright © 2024 Département de génie électrique et de génie informatique,
Université de Sherbrooke.

Table des matières

Table des matières.....	3
1. Éléments de compétences visés	4
2. Énoncé de la problématique : Développement d'un analyseur syntaxique d'expressions arithmétiques.....	5
3. Connaissances nouvelles à acquérir	7
4. Références essentielles à consulter.....	9
5. Activités liées à la problématique	10
6. Livrables	11
7. Semaine 1 : Procédural 1.....	12
8. Semaine 1 : Laboratoire	14
9. Semaine 1 : Procédural 2.....	15
10. Semaine 2 : Validation de la solution	16
11. Rapport d'APP et code : évaluation.....	17
12. Évaluation sommative.....	18
13. Évaluation de l'unité.....	19

1. Éléments de compétences visés

L'évaluation se fait selon les 4 éléments de compétences suivants :

GIF-340 Éléments de compilation

1. Décrire formellement des unités lexicales à l'aide d'expressions régulières et d'automates à états finis.
2. Décrire formellement une syntaxe à l'aide d'une grammaire. Analyser et manipuler une grammaire.
3. Concevoir et réaliser un analyseur lexical.
4. Concevoir et réaliser un analyseur syntaxique.

L'évaluation peut aussi se faire selon les trois qualités suivantes :

- **Qualité 1** : Utilisation des connaissances mathématiques
- **Qualité 2** : Analyses de problèmes
- **Qualité 4** : Conception

2. Énoncé de la problématique : Développement d'un analyseur syntaxique d'expressions arithmétiques

Dans l'entreprise où vous travaillez, un langage de programmation de haut niveau est utilisé pour le développement d'applications devant être téléchargées dans des systèmes embarqués. Un compilateur a été développé dans le passé pour l'environnement cible qui a été jusqu'alors utilisé. Pour des raisons techniques et économiques, la direction de l'entreprise a décidé de changer d'environnement cible. Le compilateur jusqu'alors utilisé doit donc être abandonné et un nouveau compilateur doit être conçu.

Afin d'éviter d'avoir à l'avenir à concevoir un compilateur en entier chaque fois que l'entreprise change d'environnement cible, votre patron vous confie le mandat de concevoir et réaliser la partie dite frontale d'un compilateur. Votre patron vous propose de suivre les conseils d'un collègue expérimenté en compilation mais qui n'a pas le temps de réaliser lui même la tâche qui vous est demandée. Après une première réunion avec votre collègue et suite à ses conseils, vous vous êtes fixé pour premier objectif de produire un prototype qui effectue l'analyse syntaxique d'expressions arithmétiques. C'est l'objectif qui sera considéré durant cette APP.

Les expressions arithmétiques sont constituées d'opérandes, d'opérateurs et de parenthèses ouvrantes et fermantes. Les opérandes sont de type entier. Les opérateurs sont l'addition (+), la soustraction (-), la multiplication (*) et la division (/). Les priorités des opérateurs sont comme suit : + et - ont même priorité ; * et / ont même priorité ; + et - sont moins prioritaires que * et /. Les quatre opérateurs sont associatifs à droite. Noter que les priorités peuvent être forcées en utilisant des parenthèses.

Un opérande peut avoir une des deux formes suivantes :

- Séquence de longueur non nulle constituée des chiffres 0 à 9.
- Lettre majuscule suivie d'une séquence possiblement vide constituée de caractères tels que :
 - chaque caractère est une lettre majuscule ou minuscule ou un tiret bas « _ » (underscore),
 - un tiret bas ne peut, ni être suivi par un autre tiret bas, ni être le dernier élément de la séquence.

Exemples corrects : AbCdE ZyXw_v_Ut

Exemples incorrects : aBcDe ZyXw__v_Ut ZyXw_v_Ut_

Voici un exemple d'expression arithmétique : $(X_a + Y_b) * Z_c / 59$. Les opérations y sont effectuées dans l'ordre suivant : l'addition, la division, et enfin la multiplication.

Après une seconde réunion avec votre collègue et d'un commun accord avec lui, vous avez décidé de commencer à concevoir un module particulier appelé analyseur lexical et qui sera utilisé par l'analyseur syntaxique. L'approche choisie par votre collègue et votre patron, consiste à spécifier les unités lexicales à l'aide d'expressions régulières (ER) et d'automates à états finis (plus brièvement : automates) ; un analyseur lexical peut ensuite être dérivé d'une manière systématique à partir des automates qui auront été produits. Il est décidé de ne pas utiliser Lex ou tout autre outil logiciel similaire.

Une fois l'analyseur lexical conçu, vient le moment de concevoir l'analyseur syntaxique. Après une recherche, vous avez constaté qu'il existe deux catégories de méthodes d'analyses syntaxiques : les méthodes ascendantes et les méthodes descendantes. Toutes ces méthodes nécessitent de modéliser la syntaxe des expressions à analyser par une grammaire. Afin d'obtenir un compilateur simple dont la conception ne nécessite pas l'utilisation d'outils logiciels spécialisés, la méthode qui vous a été suggérée, et qui a été choisie, est appelée méthode de la descente récursive (MDR).

Votre collègue vous avise que pour que la MDR soit applicable, il faut que la grammaire qui spécifie la syntaxe des expressions arithmétiques soit d'un certain type et respecte certaines contraintes. Pour que vous soyez en mesure de construire une grammaire adéquate, il vous conseille d'étudier les méthodes d'analyse syntaxique descendante, en particulier la méthode LL(1). La grammaire doit aussi avoir une forme qui garantit le respect des priorités des opérateurs.

En plus de déterminer si une expression arithmétique est syntaxiquement correcte, une tâche importante d'un analyseur syntaxique est la construction d'un arbre syntaxique abstrait (*Abstract Syntax Tree*, AST) qui modélise la structure syntaxique de l'expression arithmétique. Après construction de l'AST et afin de vérifier que cette construction est correcte, vous devez lire l'AST, c-à-d. traduire la structure de données de l'AST sous une forme lisible (sur écran ou sur fichier) et aussi déterminer son expression postfix. Pour souligner une utilité particulière de l'AST, vous devez aussi évaluer l'AST, c-à-d. évaluer le résultat du calcul de l'expression arithmétique (si tous les opérandes sont des valeurs).

Pour bénéficier des avantages de la programmation orientée-objet, vous avez pensé utiliser un des deux langages orientés-objet que vous connaissez : C++ et Java. Comme l'aspect temps-réel n'est pas primordial et pour avoir une bonne portabilité, la direction de l'entreprise a choisi Java. De plus, votre collègue a préparé des squelettes de classes java à utiliser (voir page Web de l'APP5 de S4info).

3. Connaissances nouvelles à acquérir

Connaissances déclaratives : QUOI

Mathématiques

- Expressions régulières
- Automates à états finis
- Grammaires

Sciences de l'ingénierie

- Analyse lexicale : méthode utilisant les automates à états finis
- Analyse syntaxique : méthodes descendantes, concepts PREMIER et SUIVANT
- Expressions postfix, infix, prefix

Remarque : d'autres connaissances sont utiles, mais elles ont déjà été acquises dans d'autres APPs.

Connaissances procédurales : COMMENT

- Décrire les unités lexicales d'un langage simple à l'aide d'expressions régulières et d'automates à états finis
- Décrire la syntaxe d'un langage simple par une grammaire non contextuelle
- Modifier une grammaire pour respecter des contraintes données
- Construire un arbre syntaxique abstrait
- Évaluer et lire un arbre syntaxique abstrait
- Lire dans un fichier les informations nécessaires
- Imprimer dans un fichier texte les informations produites dans chaque étape du programme
- Dériver un analyseur lexical à partir d'un automate à états finis
- Dériver un analyseur syntaxique à partir d'une grammaire non contextuelle en utilisant la méthode de la descente récursive
- Calcul de PREMIER et SUIVANT
- Évaluer un AST d'expression arithmétique
- Lire un AST d'expression arithmétique
- Dériver les expressions postfix, prefix et infix correspondant à un AST

APP : Développement d'un analyseur syntaxique

- Réaliser des tests de validation d'un analyseur lexical
- Réaliser des tests de validation d'un analyseur syntaxique

Connaissances conditionnelles : QUAND

- Choisir une grammaire qui respecte des contraintes données
- Choisir des structures de données pour décrire un AST
- Choisir des structures de données d'un analyseur lexical
- Choisir des structures de données d'un analyseur syntaxique
- Choisir des tests de validation d'un analyseur lexical
- Choisir des tests de validation d'un analyseur syntaxique

4. Références essentielles à consulter

4.1. Référence [1]

GIF 340 : *Éléments de compilation*, Ahmed Khoumsi

se trouve sur la page Web de l'APP 6

Chapitres 1 à 7

- **Introduction aux langages et à la compilation** : *Chapitre 1*, pages 1-11, 19-24

- **Analyse lexicale, expressions régulières (ER), automates (à états finis)**

Chapitre 2, pages 1-39

- **Analyse syntaxique** : *chapitres 3 à 6*

Grammaires et langages : *chapitre 3*, pages 1-23

Arbres syntaxiques abstraits (AST) : *chapitre 4*, pages 1-16

En particulier **évaluation et lecture d'AST** : pages 8-12

Introduction à l'analyse syntaxique : *chapitre 5*, pages 1-17

Analyse syntaxique descendante : *chapitre 6*, pages 1-42

- **Évaluation des représentations infix, postfix, AST** : *chapitre 7*

Représentation infix : page 3

Opérandes, précedence et associativité d'opérateurs : pages 4-6

Représentation postfix et AST : pages 7-10

Évaluation d'AST : pages 11-12

Évaluation d'expression postfix : 13-15

Conversion de infix à postfix : pages 16-22

4.2. Référence [2]

Mathématiques discrètes, Édition révisée, Kenneth H. Rosen, Chenelière/McGraw-Hill.

ISBN 2-89461-642-2. Chapitre 8, Chapitre 10.

Grammaires et langages : Sect. 10.1

Automates à états finis : Sect. 10.3

Reconnaissance de langage : Sect. 10.4

5. Activités liées à la problématique

Activités de la semaine 1

- Tutorat d'ouverture
- Procédural 1 : voir la section 7
- Laboratoire : voir la section 8
- Étude personnelle

Activités de la semaine 2

- Étude personnelle
- Procédural 2 : voir la section 9
- Rencontre d'élaboration collaborative à la problématique

Activités de la semaine 3

- Étude personnelle
- Validation de la solution : voir la section 10
- Rédaction finale du document d'APP : voir la section 11
- Remise du document d'APP avant le tutorat de fermeture
- Tutorat de fermeture : validation des connaissances acquises
- Évaluation formative
- Rencontre de consultation
- Évaluation sommative


6. Livrables

Rapport d'APP et code

Le livrable est constitué du rapport et du code java et doit être remis par équipe de **deux ou trois** le **jour du tutorat de fermeture avant 8h30**. Tous les éléments de réponses dans le rapport doivent être **justifiés**. La remise doit se faire électroniquement en déposant un **fichier ZIP** selon la procédure de dépôt habituelle. Le fichier ZIP doit contenir le rapport en format PDF et un répertoire app5 contenant tous les fichiers du code java développé.

Les consignes sur le contenu du rapport sont données dans la section 11 de ce document.

Schéma de concepts portant sur LA CONCEPTION ET LA RÉALISATION DE L'ANALYSEUR LEXICAL ET DE L'ANALYSEUR SYNTAXIQUE, à remettre à la fin de la deuxième rencontre de tutorat.

 *Inscrire votre nom, votre numéro de matricule et votre groupe de tutorat sur chaque document remis.*

Pour valider tous vos schémas, apportez-les et appliquez-les lors des rencontres de formation à la pratique procédurale et de formation à la pratique en laboratoire.

Les schémas sont à faire lors de l'étude personnelle.

7. Semaine 1 : Procédural 1

But de l'activité = développer la pratique de :

- Description d'unités lexicales par des expressions régulières et des automates (à états finis)
- Construction d'automates représentant la reconnaissance d'unités lexicales
- Dérivation d'un analyseur lexical à partir d'un automate
- Construction d'un arbre syntaxique abstrait (AST) correspondant à une expression arithmétique
- Exécution de grammaires en appliquant des règles de production correspondantes
- Dérivation d'un analyseur syntaxique à partir d'une grammaire
- Construction d'expressions régulières et de grammaires à partir d'automates

Exercices

Exercice 1 (25 minutes : 13h00 – 13h25)

On considère les deux unités lexicales suivantes :

- + qui désigne l'opérateur d'addition
- une chaîne de 0 et de 1 particulière définie par l'expression régulière $(1^+ 0 | 0) (0 | 10)^* 1?$
où *, +, ? et les parenthèses sont des méta-symboles.

Construire un automate qui représente la reconnaissance des deux unités lexicales.

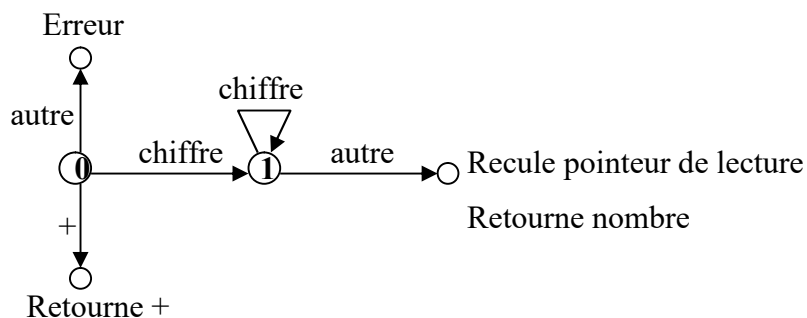
Exercice 2 (40 minutes : 13h25 – 14h05)

On considère les deux unités lexicales suivantes :

- + qui désigne l'opérateur d'addition
- une chaîne de 0 et de 1 quelconque définie par l'expression régulière $(0 | 1)^+$

L'automate qui reconnaît les deux unités lexicales est représenté ci-dessous.

Dériver un pseudo-code de l'analyseur lexical qui correspond à cet automate.



Exercice 3 (30 minutes + 10 minutes de pause : 14h05 - 14h45)

Il s'agit de construire, d'évaluer et de lire d'une manière récursive des arbres syntaxiques abstraits (AST) correspondant à des expressions arithmétiques. Les définitions de l'évaluation et de la lecture d'AST sont données dans le chapitre 4 de la référence [1].

1. Construire l'AST correspondant à l'expression $a+7*c+5$ lorsque $*$ est plus prioritaire que $+$, et $+$ et $*$ sont associatifs à gauche.
2. Reprendre la question 2 lorsque $+$ et $*$ sont associatifs à droite.
3. Expliquer comment on peut lire récursivement un AST.
4. Expliquer comment on peut obtenir récursivement l'expression postfix d'un AST
5. Expliquer comment on peut évaluer récursivement un AST.

Exercice 4 (45 minutes : 14h45 – 15h30)

On considère la grammaire suivante :

$$E \rightarrow T \mid T + E$$

$$T \rightarrow a$$

1. Déterminer les symboles terminaux, les symboles non-terminaux et le symbole de départ.
2. Déterminer une séquence de dérivations à gauche qui permet d'obtenir l'expression $a + a + a$
3. Déterminer une séquence de dérivations à droite qui permet d'obtenir l'expression $a + a + a$
4. En utilisant la méthode de la descente récursive, construire le pseudo-code de l'analyseur syntaxique qui correspond à la grammaire ci-dessus.

Exercice 5 (30 minutes : 15h30 – 16h00)

Construction de grammaire et d'expression régulière

On considère l'automate non déterministe défini par :

- l'ensemble d'états $Q = \{0, 1\}$, Ensemble d'états finaux $F = \{1\}$,
- l'état initial $= 0$,
- l'alphabet $\Sigma = \{a, b\}$,
- la fonction de transition $\delta : Q \times \Sigma \rightarrow Q$

$$\delta(0, a) = \{1\} \quad \delta(0, b) = \{0\} \quad \delta(1, a) = \{1\} \quad \delta(1, b) = \{0, 1\}$$

1. Construire le diagramme d'états de cet automate.
2. Construire une grammaire définissant le langage de cet automate.
3. Construire une expression régulière définissant le langage de cet automate.

8. Semaine 1 : Laboratoire

But de l'activité = réaliser des classes java qui permettent de réaliser un analyseur lexical et un analyseur syntaxique. Ces classes pourront par la suite être complétées et adaptées afin de résoudre la problématique. Le langage de programme est Java.

Exercices (il faut avoir accès aux squelettes de classes java fournies)

Exercice 1 (Analyse lexicale) (75 minutes : 9h00 – 10h15)

Il s'agit de réaliser l'analyseur lexical correspondant à l'automate de l'exercice 2 du 1^{er} procédural. Plus précisément, il faut réaliser les deux classes suivantes :

- **Terminal** représente un symbole terminal
- **AnalLex** représente l'analyseur lexical qui retourne les symboles terminaux reconnus et vérifie s'il n'y a pas d'erreur lexicale. **AnalLex** utilise **Terminal**.

Exercice 2 (Analyse syntaxique) (95 minutes + 10 minutes de pause : 10h15 – 12h00)

Il s'agit de réaliser l'analyseur syntaxique correspondant à la grammaire de l'exercice 4 du 1^{er} procédural. Plus précisément, il faut réaliser les quatre classes suivantes :

- **ElemAST** : classe abstraite qui représente une feuille ou un nœud d'AST
- **FeuilleAST** représente une feuille d'AST; elle hérite de **ElemAST**.
- **NoeudAST** représente un nœud d'AST; elle hérite de **ElemAST**.
- **DescenteRecursive** représente l'analyseur syntaxique qui construit l'AST décrivant la structure hiérarchique de l'expression arithmétique, et vérifie s'il n'y a pas d'erreur syntaxique.

Chaque classe **FeuilleAST** et **NoeudAST** contient, entre autres, deux méthodes **EvalAST** et **LectAST** permettant de faire l'évaluation et la lecture d'AST (voir exercice 3 du procédural 1).

DescenteRecursive utilise toutes les autres classes.

Des squelettes des classes **Terminal**, **AnalLex**, **ElemAST**, **FeuilleAST**, **NoeudAST**, et **DescenteRecursive** vous sont fournies. Deux classes **Reader** et **Writer** vous sont aussi fournies pour la lecture et l'écriture de fichier texte.

9. Semaine 2 : Procédural 2

Buts de l'activité = développer la pratique de :

- description d'unités lexicales par des expressions régulières et des automates (à états finis)
- détermination d'unités lexicales
- utilisation, construction et catégorisation de grammaires
- conception d'analyseur syntaxique
- dérivation d'arbre syntaxique abstrait (AST) et d'expressions postfix, infix et prefix

Exercices

Exercice 1 (30 minutes : 9h00 – 9h30)

Spécifier chacune des deux définitions suivantes par une expression régulière et par un automate :

1. Toute chaîne de lettres minuscules contenant une seule fois, et dans l'ordre, les 5 voyelles a, e, i, o, u.
2. Comme dans l'item 1, mais la séquence contenant les a, e, i, o, u peut se répéter un nombre indéterminé de fois.

Exercice 2 (70 minutes + 10 minutes de pause : 9h30 – 10h50)

On considère les expressions arithmétiques constituées :

- de parenthèses ouvrantes et fermantes,
- d'opérateurs + et ^, (x^y signifie x à puissance y)
- d'opérandes qui sont des séquences de chiffres.

La syntaxe est formellement définie par la grammaire suivante :

$X \rightarrow Y, \quad X \rightarrow Y + X, \quad Y \rightarrow Z, \quad Y \rightarrow Z \wedge Y, \quad Z \rightarrow (X), \quad Z \rightarrow \text{Operande}$

1. Quels sont les symboles terminaux, les symboles non terminaux, et le symbole de départ?
2. Déterminer les unités lexicales et exprimer les par des expressions régulières.
3. Calculer les ensembles PREMIER et SUIVANT des symboles non-terminaux de la grammaire.
4. Déterminer si la méthode de la descente récursive (MDR) est applicable à la grammaire ci-dessus.
5. Si la MDR n'est pas applicable, modifier la grammaire pour que l'on puisse lui appliquer la MDR.
6. En utilisant la MDR, construire le pseudo-code de l'analyseur syntaxique qui correspond à la grammaire ci-dessus (éventuellement modifiée).

Exercice 3 (30 minutes : 10h50 – 11h20)

On continue sur l'exercice 3 de la 1^{ère} formation à pratique procédurale (section 7).

1. Construire les expressions postfix, infix et prefix correspondant à chacun des deux AST.
2. Montrer comment on peut évaluer chacun des deux AST en faisant un parcours postfix.
3. Montrer comment on peut générer le pseudo-code assembleur correspondant à chacun des deux AST ou à leur expressions postfix.

Exercice 4 (10 minutes : 11h20 – 11h30) Utilisation de grammaire

On considère la grammaire suivante où S est le symbole de départ, les lettres minuscules sont des symboles terminaux, et les lettres majuscules sont des symboles non-terminaux :

$S \rightarrow ABa \quad A \rightarrow BB \quad B \rightarrow ab \quad AB \rightarrow b$

Montrer que la chaîne abababa appartient au langage créé par cette grammaire.

Exercice 5 (15 minutes : 11h30 – 11h45) Construction de grammaire

Trouver une grammaire qui définit le langage constitué des chaînes binaires contenant un nombre pair de 0 et un nombre pair de 1.

Exercice 6 (15 minutes : 11h45 – 12h00) Types de grammaires

Déterminer le type de chacune des grammaires suivantes, où S est le symbole de départ, les lettres minuscules sont des symboles terminaux, et les lettres majuscules sont des symboles non-terminaux :

$S \rightarrow ABA \quad A \rightarrow aB \quad B \rightarrow ab$

$S \rightarrow bA \quad A \rightarrow b \quad S \rightarrow \varepsilon$

$S \rightarrow AB \quad B \rightarrow aAb \quad aAb \rightarrow b$

$S \rightarrow ABS \quad BA \rightarrow ab$

10. Semaine 3 : Validation de la solution

Le but de ce laboratoire est de valider l'implantation de classes que vous avez développées. Il est nécessaire qu'avant le début de l'activité, vous ayez complété des fichiers de tests pour valider les différentes fonctionnalités de vos développements. Les résultats des tests devront être présentés d'une manière sommaire dans votre rapport d'APP.

11. Rapport d'APP et code : évaluation

Les consignes de remise du rapport sont données dans la section 6 de ce document.

Le rapport ne devrait pas dépasser 10 pages (code et dessins d'automates non inclus).

Comme indiqué à la section 1, l'activité GIF-340 comprend 4 éléments de compétences qui seront désignées par **#1**, **#2**, **#3**, **#4**.

Le rapport sera évalué comme précisé ci-dessous en se basant sur trois des qualités du BCAPG :

- Qualité 1 (**Q1**) : Utilisations de connaissances mathématiques
- Qualité 2 (**Q2**) : Analyse de problèmes
- Qualité 4 (**Q4**) : Conception

Les liens entre les qualités et les éléments de compétences sont indiqués ci-dessous

Évaluation de l'analyse lexicale

Q1 : critère « Utiliser les mathématiques discrètes pour spécifier » (#1) :	5 pts
Q2 : critère « Spécifier un système discret » (#1) :	12 pts
- Décrit toutes les unités lexicales (UL) par des expressions régulières (ER) et des automates	100%
- Décrit toutes les UL par des automates et la majorité des UL par des ER	85%
- Décrit toutes les UL par des automates	60%
- Décrit la majorité des UL par des automates	25%
Q1 : critère « Utiliser les mathématiques discrètes pour concevoir » (#3) :	3 pts
Q4 : critère « Concevoir un système discret » (#3) :	7 pts
- Décrit par un automate l'analyseur lexical reconnaissant toutes les UL	100%
- Décrit par un automate l'analyseur lexical reconnaissant plus des $\frac{3}{4}$ des UL	85%
- Décrit par un automate l'analyseur lexical reconnaissant plus de $\frac{1}{2}$ des UL	60%
- Décrit par un automate l'analyseur lexical reconnaissant plus de $\frac{1}{4}$ des UL	25%
Q4 : critère « Réaliser un système discret » (#3) :	10 pts
- Construit un code complet et clair d'analyseur lexical à partir d'un automate	100%
- Construit un code complet et assez clair	85%
- Construit un code clair mais incomplet (mais plus de la moitié présente)	60%
- Construit un code peu clair et incomplet	25%
Q4 : critère « Valider un système discret » (#3) :	9 pts
- Teste la reconnaissance de toutes les UL	100%
- Teste la reconnaissance de plus des $\frac{3}{4}$ des UL	85%
- Teste la reconnaissance de plus des $\frac{1}{2}$ des UL	60%
- Teste la reconnaissance de plus des $\frac{1}{4}$ des UL	25%

Évaluation de l'analyse syntaxique

Q1 : critère « Utiliser les mathématiques discrètes pour spécifier » (#2) :	4 pts
Q2 : critère « Spécifier un système discret » (#2) :	8 pts
- Décrit correctement et clairement toute la syntaxe à l'aide d'une grammaire	100%
- Décrit correctement toute la syntaxe à l'aide d'une grammaire	85%
- Décrit correctement la majorité de la syntaxe à l'aide d'une grammaire	60%
- Décrit correctement une partie de la syntaxe à l'aide d'une grammaire	25%
Q2 : critère « Analyser un modèle d'un système discret » (#2) :	11 pts
- Vérifie rigoureusement (en utilisant PR et SUIV) et clairement si la grammaire est LL(1)	100%
- Vérifie rigoureusement si la grammaire est LL(1)	85%
- Vérifie intuitivement et clairement si la grammaire est LL(1)	60%
- Vérifie vaguement si la grammaire est LL(1)	25%
Q1 : critère « Utiliser les mathématiques discrètes pour concevoir » (#4) :	4 pts
Q4 : critère « Concevoir un système discret » (#4) :	10 pts
- Transforme la grammaire et vérifie rigoureusement et clairement qu'elle est LL(1)	100%
- Transforme la grammaire et vérifie rigoureusement qu'elle est LL(1)	85%
- Transforme la grammaire et vérifie intuitivement et clairement qu'elle est LL(1)	60%
- Transforme la grammaire et vérifie vaguement qu'elle est LL(1)	25%
Q4 : critère « Réaliser un système discret » (#4) :	15 pts
- Construit un code complet et clair d'un analyseur syntaxique à partir d'une grammaire	100%
- Construit un code complet et assez clair	85%
- Construit un code clair mais incomplet	60%
- Construit un code peu clair et incomplet	25%
Q4 : critère « Valider un système discret » (#4) :	18 pts
- Teste la construction d'AST et la vérification de toute la syntaxe	100%
- Teste la construction d'AST et la vérification de la majorité de la syntaxe	85%
- Teste la vérification de toute la syntaxe	60%
- Teste la vérification d'une partie de la syntaxe	25%

12. Évaluation sommative

- Elle portera sur les compétences de l'activité GIF 340 et sur l'atteinte des objectifs d'apprentissage.
- Elle sera à livre fermé.

Les étudiants pourront avoir à produire du code Java ou du pseudo-code (choix libre). Dans un tel cas, on ne sanctionnera pas les erreurs de syntaxe dans le code Java, du moment que la signification est claire.

13. Évaluation de l'unité

L'évaluation portera sur les compétences figurant dans la description des activités pédagogiques. Ces compétences, ainsi que la pondération de chacune d'entre elles dans l'évaluation de cette APP, sont :

<i>Activité et élément de compétence</i>		<i>Rapport d'APP</i>	<i>Évaluation formative</i>	<i>Examen Sommatif</i>
GIF-340 Éléments de compilation				
1	Décrire formellement des unités lexicales à l'aide d'expressions régulières et d'automates à états finis.	17	--	32
2	Décrire formellement une syntaxe à l'aide d'une grammaire. Analyser et manipuler une grammaire.	23	--	42
3	Concevoir et réaliser un analyseur lexical.	29	--	54
4	Concevoir et réaliser un analyseur syntaxique.	47	--	86
<i>Total GIF 340</i>		116	--	214

Remarque : Dans le cadre de cette APP, la construction des arbres syntaxiques abstraits et l'évaluation d'expressions arithmétique sont associées à la compétence 4.