Université Laval
Faculté des sciences et de génie
Département d'informatique et de génie logiciel
IFT-3101

Danny Dubé
Version : Automne 2018

Exercices reliés au chapitre 4

Exercices

Voici les exercices que je recommande de faire :

- **Exercices 4.2.1** et **4.2.2**. (Dans la 1ère édition, l'exercice 4.1 correspond à l'exercice 4.2.2 (e), l'exercice 4.2, à l'exercice 4.2.2 (f) et l'exercice 4.3, à l'exercice 4.2.2 (g).)
  Note : lorsqu'on demande de justifier, on ne demande pas une démonstration mathématique complète.
  Note : au numéro 4.2.2 (g), le livre omet de mentionner la chaîne à utiliser. Utilisez :

  true and (**not false or true and true**)

- **Exercices 4.2.3 (a)–(f)**. (Dans la 1ère édition, les exercices 4.6 (a)–(c) sont similaires.)
- **Exercices 4.2.4** et **4.2.5**. (Dans la 1ère édition, l'exercice 4.9 est similaire.)
  Note : au numéro 4.2.5, on devrait lire `if expr` *et non pas* `if stmt` *.*
- **Exercices 4.3.1** et **4.3.2**. (Dans la 1ère édition, l'exercice 4.4 est similaire à l'exercice 4.3.1 et l'exercice 4.11, à l'exercice 4.3.2 (d).)
  Note : Lorsqu'on demande si la grammaire est appropriée pour effectuer de l'analyse syntaxique descendante, vous pouvez vous contenter d'observer si la grammaire a des défauts flagrants (récursions, facteurs communs à gauche, ambiguïté, etc...). À ce stade ci, vous pourrez uniquement démontrer qu'une grammaire n'est pas *appropriée, on ne demande pas de démontrer qu'elle est appropriée.*
- **Exercice 4.3.3**. (Exercice 4.5 dans la 1ère édition.)
- **Exercices 4.4.1** et **4.4.4**. (Dans la 1ère édition, l'exercice 4.14 correspond à l'exercice 4.4.1 (f) et l'exercice 4.15 est du même genre.)
- **Exercices 4.4.3**

- **Exercice supplémentaire 1**. Soit une grammaire G sans ϵ-production et telle que, pour tout non-terminal A, chaque A-production débute avec un terminal distinct. Montrer que G est nécessairement LL(1). (Exercice 4.18 dans la 1ère édition.)

- **Exercice supplémentaire 2**. Calculer les ensembles FIRST et FOLLOW des non-terminaux et construire la table d'analyse pour chacune des grammaires suivantes.
  - La grammaire des permutations de a, b et c :

$$\begin{array}{lcl} S & \rightarrow & aA \mid bB \mid cC \\ A & \rightarrow & bc \mid cb \\ B & \rightarrow & ac \mid ca \\ C & \rightarrow & ba \mid ab \end{array}$$

- La grammaire des types comportant les types simples entiers et caractères ainsi que les types composés représentant les tableaux et les pointeurs :

$$
\begin{array}{rcl}
\textit{type} & \rightarrow & *\textit{type} \mid \textit{array} \\
\textit{array} & \rightarrow & \textit{simple indices} \\
\textit{indices} & \rightarrow & \textit{index indices} \mid \epsilon \\
\textit{index} & \rightarrow & [\ \textit{opt_dim}\ ] \\
\textit{opt_dim} & \rightarrow & \textbf{digit} \mid \epsilon \\
\textit{simple} & \rightarrow & \textbf{int} \mid \textbf{char} \mid (\ \textit{type}\ )
\end{array}
$$

- **Exercice supplémentaire 3**. Effectuez une trace de l'algorithme d'analyse prédictive...
  - En utilisant la table d'analyse trouvée en 4.4.1 a) avec l'entrée `000111`.
  - En utilisant la table d'analyse trouvée en 4.4.1 b) avec l'entrée `+a*a+aa`.
  - En utilisant la table d'analyse trouvée en 4.4.1 e) avec l'entrée `(a,(a,a),a)`.
  - En utilisant la table d'analyse trouvée en 4.4.1 f) avec l'entrée :

    `true and not (false or false)`.

- **Exercice supplémentaire 4**. Soit la grammaire suivante :

$$
\begin{array}{rcl}
A & \rightarrow & a\ B \mid \epsilon \\
B & \rightarrow & b
\end{array}
$$

  - (a) Donnez la plus petite solution pour FIRST et FOLLOW de A et de B.
  - (b) Les ensembles suivants représentent-ils une solution pour FIRST et FOLLOW de A et B ?

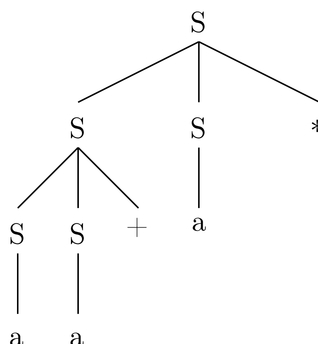
$$
\begin{array}{rcl}
FIRST(A) & = & \{a, \epsilon\} \\
FIRST(B) & = & \{b\} \\
FOLLOW(A) & = & \{a, b, \$\} \\
FOLLOW(B) & = & \{a, b, \$\}
\end{array}
$$

  - (c) Cette solution est-elle plus ou moins avantageuse que la solution la plus petite ?

Réponses

4.2.1

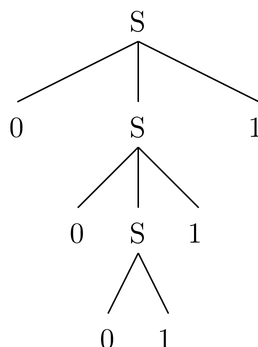
- (a) $S \to SS* \to SS+S* \to aS+S* \to aa+S* \to aa+a*$
- (b) $S \to SS* \to Sa* \to SS+a* \to Sa+a* \to aa+a*$
- (c)



- (d) La grammaire est non-ambiguë ; en effet, chaque opérateur a un nombre défini et fixe d'opérandes. On pourrait écrire un reconnaisseur récursif $S()$ qui lit les mots de droite à gauche. Un tel $S()$ aurait trois opérations possibles :
  - Lire `a`
  - Lire `+`, puis appeler $S()$ deux fois récursivement
  - Lire `*`, puis appeler $S()$ deux fois récursivement
- (e) Il s'agit des expressions mathématiques sur des `a`, avec des additions et/ou des multiplications, en notation postfixe

4.2.2 a)

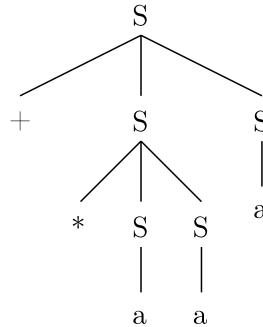
- (a) $S \to 0S1 \to 00S11 \to 000111$
- (b) $S \to 0S1 \to 00S11 \to 000111$
- (c)



- (d) La grammaire est non-ambiguë, en effet pour un nombre `n` de `0` et de `1`, il suffit d'appliquer `n - 1` fois la production $S \to 0S1$ puis une fois la production $S \to 01$; il n'existe pas d'autres productions donc c'est la seule possibilité.
- (e) Il s'agit du langage $0^n 1^n$ avec $n > 0$

4.2.2 b)

- (a) $S \to +SS \to +*SSS \to +*aSS \to +*aaS \to +*aaa$
- (b) $S \to +SS \to +Sa \to +*SSa \to +*Saa \to +*aaa$
- (c)



- (d) Voir 4.2.1 (d), la justification sera la même (mais en lisant de gauche à droite).
- (e) Il s'agit des expressions mathématiques sur des `a`, avec des additions et/ou des multiplications, en notation préfixe (notation Polonaise)

4.2.2 c)

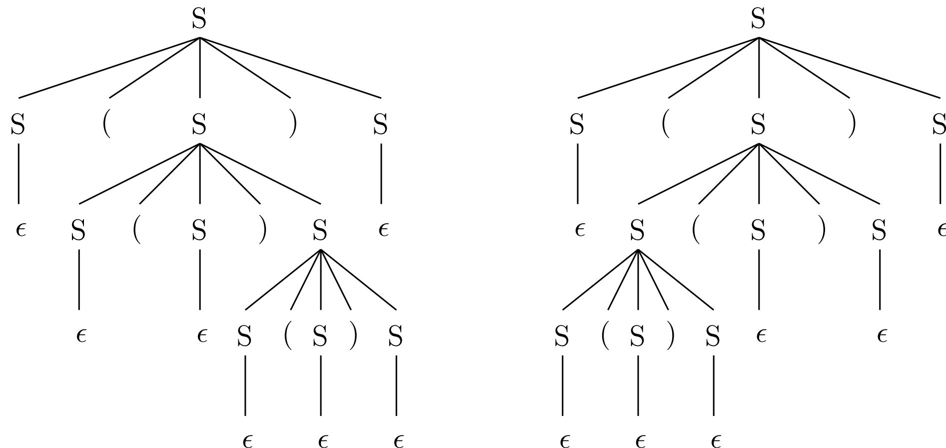
- (a)

$$\begin{aligned} S &\to S(S)S \to (S)S \to (S(S)S)S \to ((S)S)S \to (()S)S \\ &\to (()S(S)S)S \to (()(S)S)S \to (()()S)S \to (()())S \to (()()) \end{aligned}$$

- (b)

$$\begin{aligned} S &\to S(S)S \to S(S) \to S(S(S)S) \to S(S(S)) \to S(S()) \\ &\to S(S(S)S()) \to S(S(S)()) \to S(S()()) \to S(()()) \to (()()) \end{aligned}$$

- (c) Voir (d)
- (d) La grammaire est ambiguë, en effet les deux arbres de dérivation suivants sont possibles pour la chaîne `(()())` :



— (e) La langage des parenthèses bien balancées.

4.2.2 d)

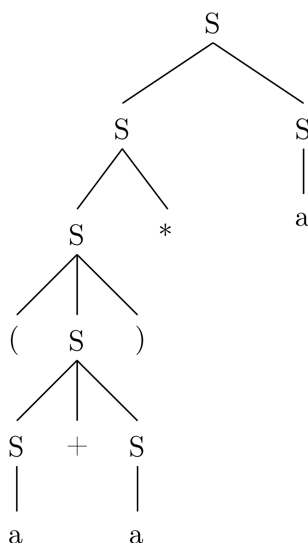
— (a)

$$\begin{aligned} S &\rightarrow SS \rightarrow S*S \rightarrow (S)*S \rightarrow (S+S)*S \rightarrow (a+S)*S \\ &\rightarrow (a+a)*S \rightarrow (a+a)*a \end{aligned}$$

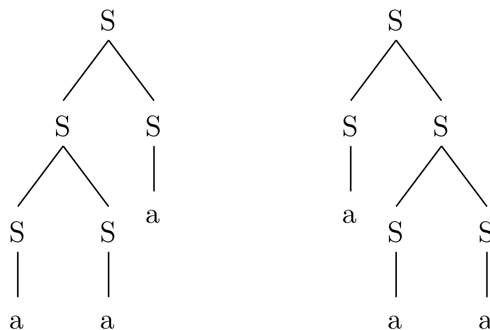
— (b)

$$\begin{aligned} S &\rightarrow SS \rightarrow Sa \rightarrow S*a \rightarrow (S)*a \rightarrow (S+S)*a \\ &\rightarrow (S+a)*a \rightarrow (a+a)*a \end{aligned}$$

— (c)



— (d) La grammaire est ambiguë, en effet les deux arbres de dérivation suivants sont possibles pour la chaîne `aaa` :



— (e) Le langage des expressions bien parenthésées avec des mots de la forme a^+, où les opérateurs $+$, $*$ et *juxtaposition* sont permis. L'opérateur $+$ est binaire et infixe ; l'opérateur $*$ est *unaire* et *postfixe* (ce n'est *pas* un opérateur binaire de multiplication, car on peut générer la chaîne `a*` avec la grammaire).

4.2.2 e)

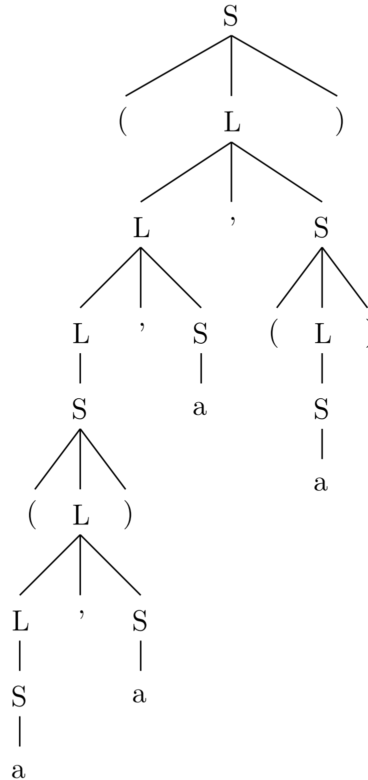
— (a)

$$
\begin{aligned}
S \;&\rightarrow\; (L) \;\rightarrow\; (L,S) \;\rightarrow\; (L,S,S) \;\rightarrow\; (S,S,S) \;\rightarrow\; ((L),S,S) \\
&\rightarrow\; ((L,S),S,S) \;\rightarrow\; ((S,S),S,S) \;\rightarrow\; ((a,S),S,S) \;\rightarrow\; ((a,a),S,S) \\
&\rightarrow\; ((a,a),a,S) \;\rightarrow\; ((a,a),a,(L)) \;\rightarrow\; ((a,a),a,(S)) \;\rightarrow\; ((a,a),a,(a))
\end{aligned}
$$

— (b)

$$
\begin{aligned}
S \;&\rightarrow\; (L) \;\rightarrow\; (L,S) \;\rightarrow\; (L,(L)) \;\rightarrow\; (L,(S)) \;\rightarrow\; (L,(a)) \;\rightarrow\; (L,S,(a)) \\
&\rightarrow\; (L,a,(a)) \;\rightarrow\; (S,a,(a)) \;\rightarrow\; ((L),a,(a)) \;\rightarrow\; ((L,S),a,(a)) \\
&\rightarrow\; ((L,a),a,(a)) \;\rightarrow\; ((S,a),a,(a)) \;\rightarrow\; ((a,a),a,(a))
\end{aligned}
$$

— (c)



— (d) La grammaire n'est pas ambiguë. Justification semi-formelle : on pourrait écrire un reconnaisseur qui lit les jetons de droite à gauche et reconnait ce langage. Un tel reconnaisseur aurait une fonction $S()$ pour lire une production de S, et une fonction $L()$ pour lire une production de L. $S()$ peut aisément déterminer quelle production utiliser en observant si le caractère à lire est `a` ou `)`. $L()$, quant à lui, peut commencer par appeler $S()$, puis observer si le caractère suivant est une virgule.

— (e) Si on appelle T le langage généré, on peut définir définir T récursivement en disant que les éléments de T sont soit la chaîne a, soit des tuples contenant n éléments de T $(n \geq 1)$.

4.2.2 f)

— (a)

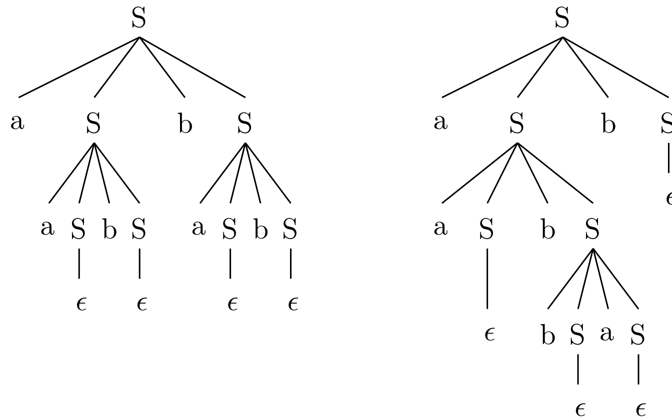
$$\begin{aligned} S &\rightarrow aSbS \rightarrow aaSbSbS \rightarrow aabSbS \rightarrow aabbS \rightarrow aabbaSbS \\ &\rightarrow aabbabS \rightarrow aabbab \end{aligned}$$

— (b)

$$\begin{aligned} S &\rightarrow aSbS \rightarrow aSbaSbS \rightarrow aSbaSb \rightarrow aSbab \rightarrow aaSbSbab \\ &\rightarrow aaSbbab \rightarrow aabbab \end{aligned}$$

— (c) Voir (d)

— (d) La grammaire est ambiguë, en effet les deux arbres de dérivation suivants sont possibles pour la chaîne `aabbab` :



— (e) Le langage est celui des chaînes dans $\{a, b\}^*$ qui contiennent autant de a que de b. En effet, la grammaire ajoute toujours un `a` et un `b` à la fois, donc la grammaire ne génère que des langages qui contiennent autant de `a` que de `b`.

4.2.2 g)

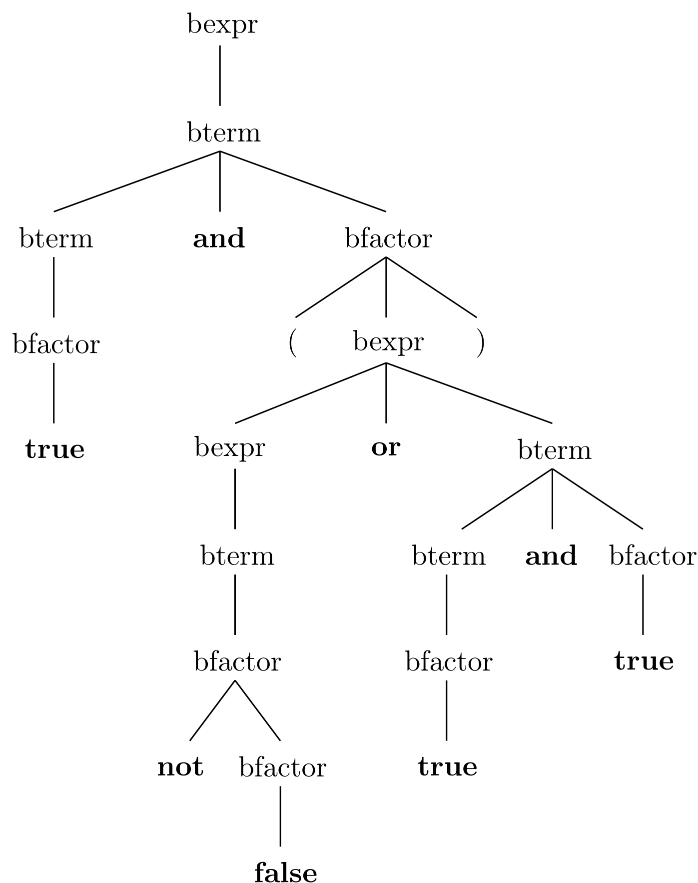
— (a)

$$
\begin{aligned}
bexpr &\rightarrow bterm \\
&\rightarrow bterm\ \textbf{and}\ bfactor \\
&\rightarrow bfactor\ \textbf{and}\ bfactor \\
&\rightarrow \textbf{true and}\ bfactor \\
&\rightarrow \textbf{true and}\ (\ bexpr\ ) \\
&\rightarrow \textbf{true and}\ (\ bexpr\ \textbf{or}\ bterm\ ) \\
&\rightarrow \textbf{true and}\ (\ bterm\ \textbf{or}\ bterm\ ) \\
&\rightarrow \textbf{true and}\ (\ bfactor\ \textbf{or}\ bterm\ ) \\
&\rightarrow \textbf{true and}\ (\ \textbf{not}\ bfactor\ \textbf{or}\ bterm\ ) \\
&\rightarrow \textbf{true and}\ (\ \textbf{not false or}\ bterm\ ) \\
&\rightarrow \textbf{true and}\ (\ \textbf{not false or}\ bterm\ \textbf{and}\ bfactor\ ) \\
&\rightarrow \textbf{true and}\ (\ \textbf{not false or}\ bfactor\ \textbf{and}\ bfactor\ ) \\
&\rightarrow \textbf{true and}\ (\ \textbf{not false or true and}\ bfactor\ ) \\
&\rightarrow \textbf{true and}\ (\ \textbf{not false or true and true}\ )
\end{aligned}
$$

— (b)

$$
\begin{aligned}
bexpr &\rightarrow bterm \\
&\rightarrow bterm\ \textbf{and}\ bfactor \\
&\rightarrow bterm\ \textbf{and}\ (\ bexpr\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ bexpr\ \textbf{or}\ bterm\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ bexpr\ \textbf{or}\ bterm\ \textbf{and}\ bfactor\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ bexpr\ \textbf{or}\ bterm\ \textbf{and true}\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ bexpr\ \textbf{or}\ bfactor\ \textbf{and true}\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ bexpr\ \textbf{or true and true}\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ bterm\ \textbf{or true and true}\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ bfactor\ \textbf{or true and true}\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ \textbf{not}\ bfactor\ \textbf{or true and true}\ ) \\
&\rightarrow bterm\ \textbf{and}\ (\ \textbf{not false or true and true}\ ) \\
&\rightarrow bfactor\ \textbf{and}\ (\ \textbf{not false or true and true}\ ) \\
&\rightarrow \textbf{true and}\ (\textbf{not false or true and true}\ )
\end{aligned}
$$

— (c)



— (d) La grammaire est non-ambiguë ; en effet, on peut voir qu'elle implémente la priorité et l'associativité des opérateurs

— (e) Le langage des expressions booléennes (avec les opérateurs et, ou, non et les valeurs vrai et faux). **and** a priorité sur **or** .

4.2.3

— (a) Cette grammaire permet d'ajouter des `1` à volonté, mais ne permet d'ajouter un `0` que si celui-ci est immédiatement suivi par un `1`.

$$S \quad \rightarrow 01S \ | \ 1S \ | \ \epsilon$$

— (b) En ajoutant toujours le même symbole terminal à gauche et à droite, on s'assure que le mot généré soit symétrique.

$$S \quad \rightarrow 0S0 \ | \ 1S1 \ | \ 0 \ | \ 1 \ | \ \epsilon$$

— (c) Voir le numéro 4.2.2 (f), il s'agit essentiellement du même problème et donc de la même grammaire. L'idée est de toujours ajouter à la fois un `0` et un `1`, ainsi il y a nécessairement un nombre égal des deux symboles.

$$S \quad \rightarrow 0S1S \ | \ 1S0S \ | \ \epsilon$$

— (d) Soit Z le cas où il y a davantage de zéros et U le cas où il y a davantage de uns. Utilisons B pour désigner une (sous) chaîne avec un nombre balancé de zéros et de uns. Notons par ailleurs qu'il ne faut pas permettre la chaîne vide, car elle contient un nombre égal de 0 et de 1.

$$\begin{array}{lll} S & \rightarrow & B1BU \mid B0BZ \\ U & \rightarrow & B1BU \mid \epsilon \\ Z & \rightarrow & B0BZ \mid \epsilon \\ B & \rightarrow & 0B1B \mid 1B0B \mid \epsilon \end{array}$$

— (e) Cette question est identique au numéro 3.3.5 (h), à l'exception qu'on demande maintenant une grammaire hors-contexte et non une expression régulière.

$$\begin{array}{lll} S & \rightarrow & 1S \mid T \\ T & \rightarrow & 0T \mid 01T \mid \epsilon \end{array}$$

— (f) Soit w une chaîne appartenant à ce langage. Alors on sait que

$$w = a_1 \dots a_k \, a_{k+1} \dots a_{2k}$$

où a_n représente le n^e caractère de w, et où :

$$a_1 \dots a_k \neq a_{k+1} \dots a_{2k}$$

Soit i l'indice du caractère de la première moitié qui ne correspond pas au caractère en même position dans la seconde moitié (il peut y avoir plus d'un caractère différent entre les deux sous-chaînes ; nous ne nous soucions ici que d'un seul caractère différent car c'est le minimum à respecter pour rencontrer les critères du langage demandé). On a que $1 \leq i \leq k$.
Soit aussi Σ, dont chaque occurrence est soit 0, soit 1. Alors, on peut générer w ainsi :

$$\begin{array}{lll} \Sigma & \rightarrow & \mathtt{0} \mid \mathtt{1} \\ S & \rightarrow & \Sigma^{i-1}\, \mathtt{0}\, \Sigma^{k-i}\, \Sigma^{i-1}\, \mathtt{1}\, \Sigma^{k-i} \\ & \mid & \Sigma^{i-1}\, \mathtt{1}\, \Sigma^{k-i}\, \Sigma^{i-1}\, \mathtt{0}\, \Sigma^{k-i} \end{array}$$

Toutefois, cette expression n'est pas une grammaire hors-contexte (il est connu que les langages de la forme $a^n b^m c^n d^m$ ne peuvent être reconnus par une grammaire hors contexte, et il en va de même pour le cas qui nous intéresse, qui est de la forme $a^n \; 0 \; b^m \; c^n \; 1 \; d^m$).
Or, on remarque que le centre de l'expression, $\Sigma^{k-i}\, \Sigma^{i-1}$, ne sert qu'à générer $k-1$ caractères aléatoires ; on pour donc remanier l'expression ainsi :

$$\begin{array}{lll} \Sigma & \rightarrow & \mathtt{0} \mid \mathtt{1} \\ S & \rightarrow & \Sigma^{i-1}\, \mathtt{0}\, \Sigma^{i-1}\, \Sigma^{k-i}\, \mathtt{1}\, \Sigma^{k-i} \\ & \mid & \Sigma^{i-1}\, \mathtt{1}\, \Sigma^{i-1}\, \Sigma^{k-i}\, \mathtt{0}\, \Sigma^{k-i} \end{array}$$

Il ne reste plus qu'à retirer l'utilisation de l'exponentiation pour obtenir une grammaire hors-contexte :

$$\begin{array}{rcl} \Sigma & \rightarrow & \mathtt{0} \mid \mathtt{1} \\ S & \rightarrow & L_0R_1 \mid L_1R_0 \\ L_0 & \rightarrow & \Sigma\, L_0\, \Sigma \mid \mathtt{0} \\ R_0 & \rightarrow & \Sigma\, R_0\, \Sigma \mid \mathtt{0} \\ L_1 & \rightarrow & \Sigma\, L_1\, \Sigma \mid \mathtt{1} \\ R_1 & \rightarrow & \Sigma\, R_1\, \Sigma \mid \mathtt{1} \end{array}$$

Finalement, on peut observer que L_0 est identique à R_0, et que L_1 est identique à R_1, et simplifier :

$$\begin{array}{rcl} \Sigma & \rightarrow & \mathtt{0} \mid \mathtt{1} \\ S & \rightarrow & L_0L_1 \mid L_1L_0 \\ L_0 & \rightarrow & \Sigma\, L_0\, \Sigma \mid \mathtt{0} \\ L_1 & \rightarrow & \Sigma\, L_1\, \Sigma \mid \mathtt{1} \end{array}$$

4.2.4

— *(i)* Toute production de la forme $S \rightarrow \alpha\ [\beta]\ \gamma$ peut être réécrite sous la forme $S \rightarrow \alpha\beta\gamma \mid \alpha\gamma$. Pour toute grammaire utilisant cette extension, il y a donc une grammaire équivalente qui n'utilise pas cette extension.

— *(ii)* Toute production de forme $S \rightarrow \alpha\ \{\beta\}\ \gamma$ peut être réécrite ainsi :

$$\begin{array}{ll} S & \rightarrow \alpha R \gamma \\ R & \rightarrow \beta R \mid \epsilon \end{array}$$

où R est un symbole qui n'était pas auparavant utilisé dans la grammaire.
Pour toute grammaire utilisant cette extension, il y a donc une grammaire équivalente qui n'utilise pas cette extension.

4.2.5

$$\begin{array}{rcl} \mathit{stmt} & \rightarrow & \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}\ [\ \textbf{else}\ \mathit{stmt}\ ] \\ & \mid & \textbf{begin}\ \mathit{stmt}\ \{;\mathit{stmt}\ \}\ \textbf{end} \end{array}$$

4.3.1

— (a) Il n'y a aucune factorisation à gauche à effectuer

$$\begin{array}{lcl} \mathit{rexpr} & \rightarrow & \mathit{rexpr}\ +\ \mathit{rterm} \mid \mathit{rterm} \\ \mathit{rterm} & \rightarrow & \mathit{rterm}\ \mathit{rfactor} \mid \mathit{rfactor} \\ \mathit{rfactor} & \rightarrow & \mathit{rfactor}\ * \mid \mathit{rprimary} \\ \mathit{rprimary} & \rightarrow & \textbf{a} \mid \textbf{b} \end{array}$$

— (b) Non, par exemple cela n'élimine pas les récursions à gauche.

— (c) On identifie les récursions à gauche suivantes :

$$\begin{array}{rcl} rexpr & \rightarrow & rexpr\ +\ rterm \\ rterm & \rightarrow & rterm\ \ rfactor \\ rfactor & \rightarrow & rfactor\, * \end{array}$$

La récursion 1 est éliminée ainsi :

$$\begin{array}{rcl} rexpr & \rightarrow & rterm\ \ rexpr' \\ rexpr' & \rightarrow & +rterm\ \ rexpr'\ \ |\ \epsilon \end{array}$$

La récursion 2 est éliminée ainsi :

$$\begin{array}{rcl} rterm & \rightarrow & rfactor\ \ rterm' \\ rterm' & \rightarrow & rfactor\ \ rterm'\ \ |\ \epsilon \end{array}$$

La récursion 3 est éliminée ainsi :

$$\begin{array}{rcl} rfactor & \rightarrow & rprimary\ \ rfactor' \\ rfactor' & \rightarrow & *\ rfactor'\ \ |\ \epsilon \end{array}$$

On obtient donc la grammaire finale suivante :

$$\begin{array}{rcl} rexpr & \rightarrow & rterm\ \ rexpr' \\ rexpr' & \rightarrow & +rterm\ \ rexpr'\ \ |\ \epsilon \\ rterm & \rightarrow & rfactor\ \ rterm' \\ rterm' & \rightarrow & rfactor\ \ rterm'\ \ |\ \epsilon \\ rfactor & \rightarrow & rprimary\ \ rfactor' \\ rfactor' & \rightarrow & *\ rfactor'\ \ |\ \epsilon \\ rprimary & \rightarrow & \mathbf{a}\ \ |\ \mathbf{b} \end{array}$$

— (d) La factorisation à gauche et la suppression des récursions à gauche ne rend pas nécessairement une grammaire utilisable pour de l'analyse syntaxique descendante. Il faudrait déterminer la table d'analyse (ou l'ensemble PREDICT de chaque symbole) pour s'en assurer.

4.3.2a

— (a) Le préfixe commun $S \rightarrow SS...$ peut être factorisé.

$$\begin{array}{rcl} S & \rightarrow & S\ S\ T\ \ |\ a \\ T & \rightarrow & +\ |\ * \end{array}$$

— (b) Non, la grammaire contient toujours des récursions à gauche

— (c) Une élimination des récurrences directes donne :

$$\begin{array}{lcl} S & \rightarrow & aS' \\ S' & \rightarrow & S\ T\ S' \ \mid\ \epsilon \\ T & \rightarrow & + \mid * \end{array}$$

— (d) La réponse à cette question n'est pas évidente, il faudrait calculer la table d'analyse.

4.3.2b

— (a) Le préfixe commun $S \rightarrow 0...$ peut être factorisé.

$$\begin{array}{lcl} S & \rightarrow & 0T \\ T & \rightarrow & S1 \mid 1 \end{array}$$

— (b) La réponse à cette question n'est pas évidente, il faudrait calculer la table d'analyse.

— (c) Il n'y a aucune récursion directe ou indirecte à gauche. Toutefois on peut appliquer l'algorithme d'élimination des récursions, ici en utilisant l'ordre { S, T }, et obtenir cette version modifiée :

$$\begin{array}{lcl} S & \rightarrow & 0T \\ T & \rightarrow & 0T\,1 \mid 1 \end{array}$$

— (d) La grammaire est appropriée. En effet, on pourrait écrire un reconnaisseur récursif de cette façon :
  - — La fonction $S()$ lit un `0` puis appelle $T()$
  - — La fonction $T()$ regarde le premier caractère ; s'il s'agit d'un `0`, elle lit le caractère, puis s'appelle récursivement, et au retour de l'appel récursif lit un `1` ; s'il s'agit d'un `1`, $T()$ lit le caractère et retourne.

4.3.2c

— (a) Il n'y a aucun préfixe commun à factoriser

— (b) Non car elle est ambiguë.

— (c) En appliquant directement la technique générale d'élimination des récursions, on trouve :

$$\begin{array}{lcl} S & \rightarrow & S' \\ S' & \rightarrow & (\ S\ )\ S\ S' \ \mid\ \epsilon \end{array}$$

— (d) Non car elle est ambiguë. En effet, puisque S est en tout point équivalent à S', la forme de phrase (S) $S\ S'$ est équivalente à (S') $S'\ S'$.
Numérotons les symboles ainsi : (S') $S'_1\ S'_2$
On voit que si on veut produire la chaîne `()()`, on peut générer la seconde paire de parenthèses autant avec S'_1 qu'avec S'_2.

4.3.2d

- (a) Il n'y a aucun préfixe commun à factoriser
- (b) Non, car elle comporte des récursions à gauche.
- (c)

$$\begin{array}{lcl} S & \rightarrow & (\ L\ )\ \mid\ a \\ L & \rightarrow & S\ L' \\ L' & \rightarrow & ,S\ L'\ \mid\ \epsilon \end{array}$$

- (d) La grammaire applique correctement la priorité des opérateurs et semble correcte. Pour une preuve formelle, il faudrait calculer la table d'analyse.

4.3.2e

- (a) Il n'y a aucun préfixe commun à factoriser
- (b) Non, car elle comporte des récursions à gauche.
- (c)

  $bexpr \rightarrow bexpr$ **or** $bterm \ \mid\ bterm$ devient :

$$\begin{array}{lcl} bexpr & \rightarrow & bterm\ \ bexpr' \\ bexpr' & \rightarrow & \textbf{or}\ bterm\ \ bexpr'\ \mid\ \epsilon \end{array}$$

  $bterm \rightarrow bterm$ **and** $bfactor \ \mid\ bfactor$ devient :

$$\begin{array}{lcl} bterm & \rightarrow & bfactor\ \ bterm' \\ bterm' & \rightarrow & \textbf{and}\ bfactor\ \ bterm'\ \mid\ \epsilon \end{array}$$

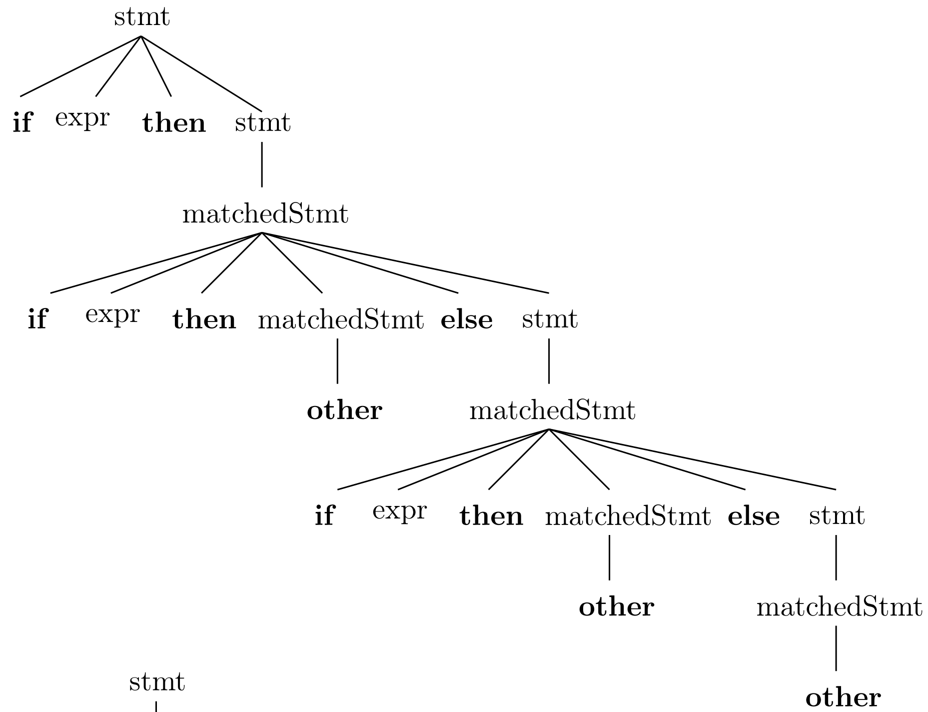
- (d) La grammaire applique correctement la priorité des opérateurs et semble correcte. Pour une preuve formelle, il faudrait calculer la table d'analyse.

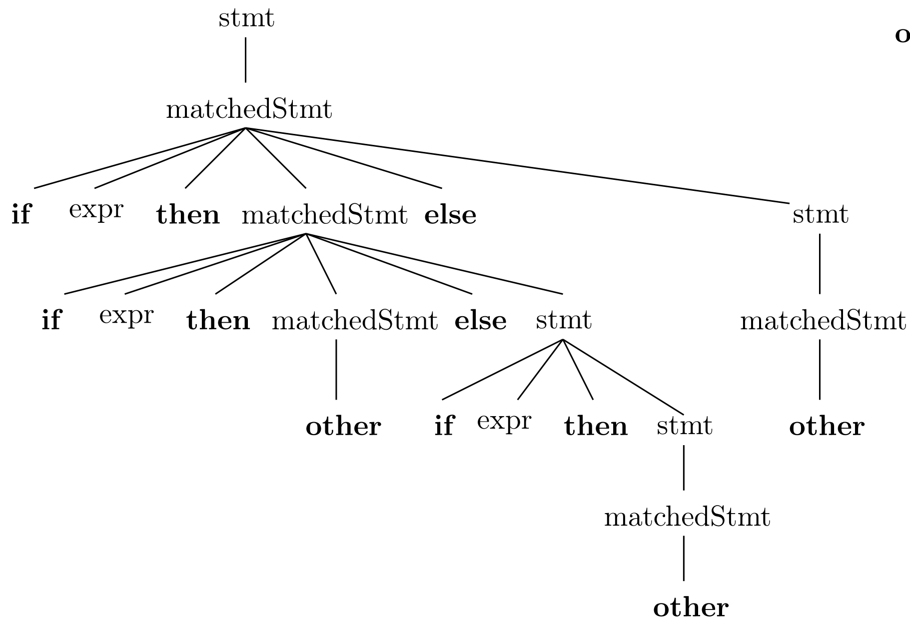
4.3.3

La chaîne

if *expr* **then if** *expr* **then other else if** *expr* **then other else other**

a deux arbres de dérivation avec cette grammaire :





4.4.1a

Réutilisons la grammaire sans récursion à gauche trouvée au 4.3.2b)

$$\begin{array}{rcl} S & \rightarrow & 0\,T \\ T & \rightarrow & 0\,T\,1 \mid 1 \end{array}$$

On calcule First :

$$\begin{array}{rcl} \text{FIRST}(\ S\ ) & = & \text{FIRST}(\ 0\,T\ ) = \{0\} \\ \text{FIRST}(\ T\ ) & = & \text{FIRST}(\ 0\,T\,1) \cup \text{FIRST}(\ 1) = \{0, 1\} \end{array}$$

On fixe les contraintes déterminant Follow :

$$\begin{array}{rcl} \text{FOLLOW}(\ S\ ) & \supseteq & \{\$\} \\ \text{FOLLOW}(\ T\ ) & \supseteq & \text{FOLLOW}(\ S\ ) \\ \text{FOLLOW}(\ T\ ) & \supseteq & \text{FIRST}(\ 1) - \{\epsilon\} = \{1\} \end{array}$$

On trouve la solution suivante pour Follow :

$$\begin{array}{rcl} \text{FOLLOW}(\ S\ ) & \supseteq & \{\$\} \\ \text{FOLLOW}(\ T\ ) & \supseteq & \{\$, 1\} \end{array}$$

On trouve ensuite les conditions pour construire la table d'analyse M :

Règle	Condition
$S \rightarrow 0\,T$	FIRST($0\,T$) = {0} donc on ajoute cette règle à M[S, 0]
$T \rightarrow 0\,T\,1$	FIRST($0\,T\,1$) = {0} donc on ajoute cette règle à M[T, 0]
$T \rightarrow 1$	FIRST(1) = {1} donc on ajoute cette règle à M[T, 1]

On obtient enfin la table d'analyse :

	0	1	$
S	$S \rightarrow 0\,T$	–	–
T	$T \rightarrow 0\,T\,1$	$T \rightarrow 1$	–

4.4.1b

On calcule First :

$$\text{FIRST}(\ S\ ) = \{a, +, *\}$$

On trouve la solution suivante pour Follow :

$$\text{FOLLOW}(\ S\ ) = \{a, +, *, \$\}$$

On obtient enfin la table d'analyse :

	+	*	a	$
S	$S \rightarrow +S\,S$	$S \rightarrow *S\,S$	$S \rightarrow a$	–

4.4.1c

Utilisons la grammaire sans récursion trouvée au numéro 4.3.2(c) :

$$
\begin{aligned}
S &\rightarrow S' \\
S' &\rightarrow (\ S\ )\ S\ S' \ |\ \epsilon
\end{aligned}
$$

On calcule First :

$$
\begin{aligned}
\text{FIRST}(\ S\ ) &= \{(, \epsilon\} \\
\text{FIRST}(\ S'\ ) &= \{(, \epsilon\}
\end{aligned}
$$

On trouve la solution suivante pour Follow :

$$
\begin{aligned}
\text{FOLLOW}(\ S\ ) &= \{(,), \$\} \\
\text{FOLLOW}(\ S'\ ) &= \{(,), \$\}
\end{aligned}
$$

On obtient enfin la table d'analyse :

	(	)	$
S	$S \rightarrow S'$	$S \rightarrow S'$	$S \rightarrow S'$
S'	$S' \rightarrow \epsilon$ $S' \rightarrow (S)S\ S'$	$S' \rightarrow \epsilon$	$S' \rightarrow \epsilon$

On conclut que cette grammaire n'est pas LL(1). (Notons qu'on pouvait s'en douter sans même construire la table car cette grammaire est ambiguë.)

4.4.1d

Commençons par factoriser à gauche la grammaire :

$$
\begin{aligned}
S &\rightarrow S\ T \ |\ (\ S\ ) \ |\ a \\
T &\rightarrow +S \ |\ S \ |\ *
\end{aligned}
$$

Puis éliminons les récursions à gauche (l'ordre {T, S} a été utilisé lors de l'utilisation de l'algorithme d'élimination des récurrences) :

$$
\begin{aligned}
S &\rightarrow (\ S\ )\ S' \ |\ a\, S' \\
S' &\rightarrow T\ S' \ |\ \epsilon \\
T &\rightarrow +S \ |\ S \ |\ *
\end{aligned}
$$

Nous pouvons maintenant calculer FIRST :

$$
\begin{aligned}
\text{FIRST}(\ T\ ) &= \{(, a, *, +\} \\
\text{FIRST}(\ S\ ) &= \{(, a\} \\
\text{FIRST}(\ S'\ ) &= \{(, a, *, +, \epsilon\}
\end{aligned}
$$

On trouve la solution suivante pour Follow :

$$\begin{aligned} \text{FOLLOW}(\ T\ ) &= \{(,\$,a,+,*,)\} \\ \text{FOLLOW}(\ S\ ) &= \{(,\$,a,+,*,)\} \\ \text{FOLLOW}(\ S'\ ) &= \{(,\$,a,+,*,)\} \end{aligned}$$

On obtient enfin la table d'analyse :

	(	)	a	+	*	$
S	$S \rightarrow (\ S\ )S'$	$S \rightarrow aS'$	–	–	–	–
S'	$S' \rightarrow T\ S'$ $S' \rightarrow \epsilon$	$S' \rightarrow \epsilon$	$S' \rightarrow T\ S'$ $S' \rightarrow \epsilon$	$S' \rightarrow T\ S'$ $S' \rightarrow \epsilon$	$S' \rightarrow T\ S'$ $S' \rightarrow \epsilon$	$S' \rightarrow \epsilon$
T	$T \rightarrow S$	–	$T \rightarrow S$	$T \rightarrow +S$	$T \rightarrow *$	–

On conclut que cette grammaire n'est pas LL(1). (Notons qu'on pouvait s'en douter sans même construire la table car cette grammaire est hautement ambiguë.)

4.4.1e

Utilisons la grammaire sous la forme sans récursion à gauche ou préfixe commun trouvée précédemment.

$$\begin{aligned} S &\rightarrow (\ L\ )\ |\ a \\ L &\rightarrow S\ L' \\ L' &\rightarrow ,S\ L'\ |\ \epsilon \end{aligned}$$

Nous pouvons maintenant calculer FIRST :

$$\begin{aligned} \text{FIRST}(\ S\ ) &= \{(,a\} \\ \text{FIRST}(\ L\ ) &= \{(,a\} \\ \text{FIRST}(\ L'\ ) &= \{,,\epsilon\} \end{aligned}$$

On trouve la solution suivante pour Follow :

$$\begin{aligned} \text{FOLLOW}(\ S\ ) &= \{),\$,,\} \\ \text{FOLLOW}(\ L\ ) &= \{)\} \\ \text{FOLLOW}(\ L'\ ) &= \{)\} \end{aligned}$$

On obtient enfin la table d'analyse :

	(	)	a	,	$
S	S $\rightarrow$ (L)	–	S $\rightarrow$ a	–	–
L	$L \rightarrow S\ L'$	–	$L \rightarrow S\ L'$	–	–
L'	–	$L' \rightarrow \epsilon$	–	$L' \rightarrow ,SL'$	–

4.4.1f

Utilisons la grammaire sous la forme sans récursion à gauche ou préfixe commun trouvée précédemment.

$$\begin{array}{rcl} \textit{bexpr} & \rightarrow & \textit{bterm} \ \ \textit{bexpr'} \\ \textit{bexpr'} & \rightarrow & \textbf{or} \ \textit{bterm} \ \ \textit{bexpr'} \ \mid \epsilon \\ \textit{bterm} & \rightarrow & \textit{bfactor} \ \ \textit{bterm'} \\ \textit{bterm'} & \rightarrow & \textbf{and} \ \textit{bfactor} \ \ \textit{bterm'} \ \mid \epsilon \\ \textit{bfactor} & \rightarrow & \textbf{not} \ \textit{bfactor} \ \mid (\ \textit{bexpr} \ ) \mid \ \textbf{true} \ \mid \ \textbf{false} \end{array}$$

Calculons FIRST :

$$\begin{array}{rcl} \text{FIRST}(\ \textit{bexpr} \ ) & = & \{\textbf{false} \ , \textbf{true} \ , \textbf{not} \ , (\} \\ \text{FIRST}(\ \textit{bexpr'} \ ) & = & \{\textbf{or} \ , \epsilon\} \\ \text{FIRST}(\ \textit{bterm} \ ) & = & \{\textbf{false} \ , \textbf{true} \ , \textbf{not} \ , (\} \\ \text{FIRST}(\ \textit{bterm'} \ ) & = & \{\textbf{and} \ , \epsilon\} \\ \text{FIRST}(\ \textit{bfactor} \ ) & = & \{\textbf{false} \ , \textbf{true} \ , \textbf{not} \ , (\} \end{array}$$

On trouve la solution suivante pour Follow :

$$\begin{array}{rcl} \text{FOLLOW}(\ \textit{bexpr} \ ) & = & \{), \$\} \\ \text{FOLLOW}(\ \textit{bexpr'} \ ) & = & \{), \$\} \\ \text{FOLLOW}(\ \textit{bterm} \ ) & = & \{\textbf{or} \ ,), \$\} \\ \text{FOLLOW}(\ \textit{bterm'} \ ) & = & \{\textbf{or} \ ,), \$\} \\ \text{FOLLOW}(\ \textit{bfactor} \ ) & = & \{\textbf{and} \ , \textbf{or} \ ,), \$\} \end{array}$$

On obtient enfin la table d'analyse :

	$	(	)	**not**
bexpr	–	*bexpr* → *bterm bexpr'*	–	*bexpr* → *bterm bexpr'*
bexpr'	*bexpr'* → ϵ	–	*bexpr'* → ϵ	–
bterm	–	*bterm* → *bfactor bterm'*	–	*bterm* → *bfactor bterm'*
bterm'	*bterm'* → ϵ	–	*bterm'* → ϵ	–
bfactor	–	*bfactor* → (*bexpr*)	–	*bfactor* → **not** *bfactor*

	or	**and**	**true**	**false**
bexpr	–	–	*bexpr* → *bterm bexpr'*	
bexpr'	*bexpr'* → **or** *bterm bexpr'*	–	–	–
bterm	–	–	*bterm* → *bfactor bterm'*	
bterm'	*bterm'* → ϵ	*bterm'* → **and** *bfactor bterm'*	–	–
bfactor	–	–	*bfactor* → **true**	*bfactor* → **false**

4.4.3

Posons les contraintes pour le calcul des ensembles FIRST :

$$\begin{aligned} \text{FIRST}(\ S\ ) &\supseteq \text{FIRST}(\ S\ S +) \cup \text{FIRST}(\ S\ S *) \cup \text{FIRST}(\ a) \\ &= \text{FIRST}(\ S\ ) \cup \text{FIRST}(\ a) \end{aligned}$$

car $\epsilon \notin \text{FIRST}(\ S\ )$.

La plus petite solution aux contraintes est la suivante :

$$\text{FIRST}(\ S\ ) = \{a\}.$$

Posons les contraintes pour le calcul des ensembles FOLLOW :

$$\begin{aligned} \text{FOLLOW}(\ S\ ) &\supseteq \{\$\} \\ \text{FOLLOW}(\ S\ ) &\supseteq \text{FIRST}(\ S +) \\ \text{FOLLOW}(\ S\ ) &\supseteq \text{FIRST}(\ +) \\ \text{FOLLOW}(\ S\ ) &\supseteq \text{FIRST}(\ S *) \\ \text{FOLLOW}(\ S\ ) &\supseteq \text{FIRST}(\ *) \end{aligned}$$

La plus petite solution aux contraintes est la suivante :

$$\text{FOLLOW}(\ S\ ) = \{a, +, *, \$\}.$$

4.4.4

Bien que la plupart des grammaires de l'exercice 4.2.2 ont été considérées dans l'exercice 4.4.1 (à l'exception de la grammaire (f)) et ont vu leurs ensembles FIRST et FOLLOW être calculés, ce sont presque toutes des versions modifiées des grammaires qui ont été considérées. Ainsi, nous calculons ici les ensembles FIRST et FOLLOW sur les grammaires telles que données à l'exercice 4.2.2.

Nous ne donnons les détails des calculs que pour les deux premières grammaires. Ensuite, nous ne donnons que les résultats.

• Grammaire 4.2.2(a).

Posons les contraintes pour le calcul des ensembles FIRST :

$$\text{FIRST}(\ S\ ) \supseteq \text{FIRST}(\ \mathtt{0}S\ \mathtt{1}) \cup \text{FIRST}(\ \mathtt{01})$$

La plus petite solution aux contraintes est la suivante :

$$\text{FIRST}(\ S\ ) = \{\mathtt{0}\}.$$

Posons les contraintes pour le calcul des ensembles FOLLOW :

$$\begin{aligned} \text{FOLLOW}(\ S\ ) &\supseteq \{\$\} \\ \text{FOLLOW}(\ S\ ) &\supseteq \text{FIRST}(\ \mathtt{1}) \end{aligned}$$

La plus petite solution aux contraintes est la suivante :

$$\text{FOLLOW}(\ S\ ) \quad = \quad \{\texttt{1}, \$\}.$$

- Grammaire 4.2.2(b).

  Posons les contraintes pour le calcul des ensembles FIRST :

$$\text{FIRST}(\ S\ ) \quad \supseteq \quad \text{FIRST}(\ +S\ S\ ) \cup \text{FIRST}(\ *S\ S\ ) \cup \text{FIRST}(\ a)$$

La plus petite solution aux contraintes est la suivante :

$$\text{FIRST}(\ S\ ) \quad = \quad \{a, +, *\}.$$

Posons les contraintes pour le calcul des ensembles FOLLOW :

$$\begin{array}{rcl} \text{FOLLOW}(\ S\ ) & \supseteq & \{\$\} \\ \text{FOLLOW}(\ S\ ) & \supseteq & \text{FIRST}(\ S\ ) \\ \text{FOLLOW}(\ S\ ) & \supseteq & \text{FIRST}(\ \epsilon) - \{\epsilon\} \\ \text{FOLLOW}(\ S\ ) & \supseteq & \text{FOLLOW}(\ S\ ) \end{array}$$

La plus petite solution aux contraintes est la suivante :

$$\text{FOLLOW}(\ S\ ) \quad = \quad \{a, +, *, \$\}.$$

- Grammaire 4.2.2(c).

$$\text{FIRST}(\ S\ ) \quad = \quad \{\epsilon, (\}.$$

$$\text{FOLLOW}(\ S\ ) \quad = \quad \{(,), \$\}.$$

- Grammaire 4.2.2(d).

$$\text{FIRST}(\ S\ ) \quad = \quad \{a, (\}.$$

$$\text{FOLLOW}(\ S\ ) \quad = \quad \{a, +, (,), *, \$\}.$$

- Grammaire 4.2.2(e).

$$\begin{array}{rcl} \text{FIRST}(\ S\ ) & = & \{a, (\} \\ \text{FIRST}(\ L\ ) & = & \{a, (\}. \end{array}$$

$$\begin{aligned} \text{FOLLOW}(\ S\ ) &= \{), \mathbf{,}, \$\} \\ \text{FOLLOW}(\ L\ ) &= \{), \mathbf{,}\}. \end{aligned}$$

- Grammaire 4.2.2(f).

$$\text{FIRST}(\ S\ ) = \{\epsilon, a, b\}.$$

$$\text{FOLLOW}(\ S\ ) = \{a, b, \$\}.$$

- Grammaire 4.2.2(g).

$$\begin{aligned} \text{FIRST}(\ \textit{bexpr}\ ) &= \{\textbf{not}, (, \textbf{true}, \textbf{false}\} \\ \text{FIRST}(\ \textit{bterm}\ ) &= \{\textbf{not}, (, \textbf{true}, \textbf{false}\} \\ \text{FIRST}(\ \textit{bfactor}\ ) &= \{\textbf{not}, (, \textbf{true}, \textbf{false}\}. \end{aligned}$$

$$\begin{aligned} \text{FOLLOW}(\ \textit{bexpr}\ ) &= \{\textbf{or},), \$\} \\ \text{FOLLOW}(\ \textit{bterm}\ ) &= \{\textbf{and}, \textbf{or},), \$\} \\ \text{FOLLOW}(\ \textit{bfactor}\ ) &= \{\textbf{and}, \textbf{or},), \$\}. \end{aligned}$$

Exercice supplémentaire 1

Soit G une grammaire de la forme :

$$\begin{array}{lllllllll} A & \rightarrow & a\alpha_1 & | & b\alpha_2 & | & c\alpha_3 & | & \dots \\ B & \rightarrow & a\beta_1 & | & b\beta_2 & | & c\beta_2 & | & \dots \\ C & \rightarrow & a\gamma_1 & | & b\gamma_2 & | & c\gamma_2 & | & \dots \\ \vdots & & \vdots & & \vdots & & \vdots & & \ddots \end{array}$$

où chaque production est optionnelle

Calculons la table d'analyse de cette grammaire. Puisque toutes les productions commencent par un terminal, nous aurons que ϵ ne fera partie d'aucun ensemble FIRST dans cette grammaire. Étant donné que l'algorithme de construction de table d'analyse n'a recours à FOLLOW que dans le cas où ϵ appartient à un ensemble FIRST, nous savons que nous n'aurons pas besoin de calculer FOLLOW.

L'algorithme de construction de table place une règle dans les colonnes correspondant au FIRST de ses productions. Puisque G respecte la condition que toutes les productions commencent par un terminal différent, nous avons la certitude que chaque case de la table d'analyse contiendra au plus une règle.

Autrement dit, la table d'analyse de G sera :

	a	b	c	...
A	$A \rightarrow a\alpha_1$	$A \rightarrow b\alpha_2$	$A \rightarrow c\alpha_3$	...
B	$B \rightarrow a\beta_1$	$B \rightarrow b\beta_2$	$B \rightarrow c\beta_3$	...
C	$C \rightarrow a\gamma_1$	$C \rightarrow b\gamma_2$	$C \rightarrow c\gamma_3$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Chaque règle étant optionnelle, une règle absente dans G résultera simplement en un trou dans la table d'analyse.

G est donc LL(1).

Exercice supplémentaire 2

— (permutations) Calculons FIRST :

$$\begin{aligned} \text{FIRST}(\ S\ ) &= \{a, b, c\} \\ \text{FIRST}(\ A\ ) &= \{b, c\} \\ \text{FIRST}(\ B\ ) &= \{a, c\} \\ \text{FIRST}(\ C\ ) &= \{a, b\} \end{aligned}$$

On trouve la solution suivante pour Follow :

$$\begin{aligned} \text{FOLLOW}(\ S\ ) &= \{\$\} \\ \text{FOLLOW}(\ A\ ) &= \{\$\} \\ \text{FOLLOW}(\ B\ ) &= \{\$\} \\ \text{FOLLOW}(\ C\ ) &= \{\$\} \end{aligned}$$

On trouve enfin la table suivante :

	a	b	c	\$
S	$S \rightarrow aA$	$S \rightarrow bB$	$S \rightarrow cC$	
A		$A \rightarrow bc$	$A \rightarrow cb$	
B	$B \rightarrow ac$		$B \rightarrow ca$	
C	$C \rightarrow ab$	$C \rightarrow ba$		

— (types simples et composés) la table d'analyse obtenue devrait être la suivante (en renommant les 6 non-terminaux par A, ..., F, respectivement) :

	$*$	[	]	**digit**	**int**	**char**	(	)	\$
A	$A \rightarrow *A$				$A \rightarrow B$	$A \rightarrow B$	$A \rightarrow B$		
B					$B \rightarrow FC$	$B \rightarrow FC$	$B \rightarrow FC$		
C		$C \rightarrow DC$						$C \rightarrow \epsilon$	$C \rightarrow \epsilon$
D		$D \rightarrow [E]$							
E			$E \rightarrow \epsilon$	$E \rightarrow \textbf{digit}$					
F					$F \rightarrow \textbf{int}$	$F \rightarrow \textbf{char}$	$F \rightarrow (A)$		

Exercice supplémentaire 3

— (a)

Pile	Entrée	Sortie
S \$	000111\$	$S \to 0T$
0 T \$	000111\$	(Consommer le terminal 0)
T \$	00111\$	$T \to 0T1$
0 T 1 \$	00111\$	(Consommer le terminal 0)
T 1 \$	0111\$	$T \to 0T1$
0 T 1 1 \$	0111\$	(Consommer le terminal 0)
T 1 1 \$	111\$	$T \to 1$
1 1 1 \$	111\$	(Consommer le terminal 1)
1 1 \$	11\$	(Consommer le terminal 1)
1 \$	1\$	(Consommer le terminal 1)
\$	\$	(Succès)

— (b)

Pile	Entrée	Sortie
S \$	+a*a+aa\$	$S \to +SS$
+ S S \$	+a*a+aa\$	(Consommer le terminal '+')
S S \$	a*a+aa\$	$S \to a$
a S \$	a*a+aa\$	(Consommer le terminal 'a')
S \$	*a+aa\$	$S \to *SS$
* S S \$	*a+aa\$	(Consommer le terminal '*')
S S \$	a+aa\$	$S \to a$
a S \$	a+aa\$	(Consommer le terminal 'a')
S \$	+aa\$	$S \to +SS$
+ S S \$	+aa\$	(Consommer le terminal '+')
S S \$	aa\$	$S \to a$
a S \$	aa\$	(Consommer le terminal 'a')
S \$	a\$	$S \to a$
a \$	a\$	(Consommer le terminal 'a')
\$	\$	(Succès)

— (c)

Pile	Entrée	Sortie
S \$	(a,(a,a),a)\$	$S \to (\ L \ )$
(L) \$	(a,(a,a),a)\$	(Consommer le terminal '(')
L) \$	a,(a,a),a)\$	$L \to S \ L'$
$S \ L'$) \$	a,(a,a),a)\$	$S \to a$
$a \ L'$) \$	a,(a,a),a)\$	(Consommer le terminal 'a')
L') \$	,(a,a),a)\$	$L' \to , \ S \ L'$
, $S \ L'$) \$	,(a,a),a)\$	Consommer le terminal ',')
$S \ L'$) \$	(a,a),a)\$	$S \to (\ L \ )$
(L) L') \$	(a,a),a)\$	(Consommer le terminal '(')
L) L') \$	a,a),a)\$	$L \to S \ L'$
$S \ L'$) L') \$	a,a),a)\$	$S \to a$
$a \ L'$) L') \$	a,a),a)\$	(Consommer le terminal 'a')
L') L') \$	,a),a)\$	$L' \to , \ S \ L'$
, $S \ L'$) L') \$	,a),a)\$	Consommer le terminal ',')
$S \ L'$) L') \$	a),a)\$	$S \to a$
$a \ L'$) L') \$	a),a)\$	(Consommer le terminal 'a')
L') L') \$	),a)\$	$L' \to \epsilon$
) L') \$	),a)\$	(Consommer le terminal ')')
L') \$	,a)\$	$L' \to , \ S \ L'$
, $S \ L'$) \$	,a)\$	Consommer le terminal ',')
$S \ L'$) \$	a)\$	$S \to a$
$a \ L'$) \$	a)\$	(Consommer le terminal 'a')
L') \$	)\$	$L' \to \epsilon$
) \$	)\$	(Consommer le terminal ')')
\$	\$	(Succès)

— (d)

Pile	Entrée	Sortie
$bexpr$ \$	**t and not f or f** \$	$bexpr \to bterm\ bexpr'$
$bterm\ bexpr'$ \$	**t and not f or f** \$	$bterm \to bfact\ bterm'$
$bfact\ bterm'\ bexpr'$ \$	**t and not f or f** \$	$bfact \to$ **t**
t $bterm'\ bexpr'$ \$	**t and not f or f** \$	(Consommer le terminal **t**)
$bterm'\ bexpr'$ \$	**and not f or f** \$	$bterm' \to$ **and** $bfact\ bterm'$
and $bfact\ bterm'\ bexpr'$ \$	**and not f or f** \$	(Consommer le terminal **and**)
$bfact\ bterm'\ bexpr'$ \$	**not f or f** \$	$bfact \to$ **not** $bfact$
not $bfact\ bterm'\ bexpr'$ \$	**not f or f** \$	(Consommer le terminal **not**)
$bfact\ bterm'\ bexpr'$ \$	**f or f** \$	$bfact \to$ **f**
f $bterm'\ bexpr'$ \$	**f or f** \$	(Consommer le terminal **f**)
$bterm'\ bexpr'$ \$	**or f** \$	$bterm' \to \epsilon$
$bexpr'$ \$	**or f** \$	$bexpr' \to$ **or** $bterm\ bexpr'$
or $bterm\ bexpr'$ \$	**or f** \$	(Consommer le terminal **or**)
$bterm\ bexpr'$ \$	**f** \$	$bterm \to bfact\ bterm'$
$bfact\ bterm'\ bexpr'$ \$	**f** \$	$bfact \to$ **f**
f $bterm'\ bexpr'$ \$	**f** \$	(Consommer le terminal **f**)
$bterm'\ bexpr'$ \$	\$	$bterm' \to \epsilon$
$bexpr'$ \$	\$	$bexpr' \to \epsilon$
\$	\$	(Succès)

Exercice supplémentaire 4

— (a)

$$\begin{aligned} FIRST(A) &= \{a, \epsilon\} \\ FIRST(B) &= \{b\} \\ FOLLOW(A) &= \{\$\} \\ FOLLOW(B) &= \{\$\} \end{aligned}$$

— (b) Oui, car toutes les contraintes sont respectées. Dans le case des ensembles FIRST, les contraintes sont respectées strictement. Tandis que, dans les cas des ensembles FOLLOW, les contraintes ne nous forcent pas à inclure tous ces éléments.

— (c) La solution minimale est la solution la plus avantageuse, car en ne rajoutant pas de symboles inutiles, on évite de créer des conflits artificiels dans la table d'analyse.