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

Danny Dubé
Automne 2020
Version : 20 octobre

# Corrigé du travail pratique #1

## Réponses

1. La réponse est (f). En effet, on ne se limite pas qu'à des préfixes car `ood` se trouve dans $S$. On ne se limite pas qu'à des suffixes non plus car `foo` se trouve dans $S$. Toutefois, $S$ ne contient pas toutes les sous-séquences (ni propres, ni quelconques), sinon `fd` s'y trouverait. $S$ contient donc des sous-chaînes. Comme $S$ ne contient pas $\epsilon$ ni `food`, alors $S$ se limite aux sous-chaînes propres.

2. La réponse est (b).

   La réponse (a) n'est pas bonne car l'expression régulière donnée s'assure seulement de faire respecter au moins l'une des deux conditions du langage. Par exemple, l'expression peut générer $\epsilon$, qui ne contient pas `cab` comme sous-chaîne.

   La réponse (c) n'est pas bonne pour plusieurs raisons. Entre autres, en concaténant deux sous-chaînes produites par $f$, on peut quand même arriver à créer la sous-chaîne `bb` :

   $$L(f^*) \supseteq L(f^2) \ni \texttt{ab} \cdot \texttt{bc} \ ,$$

   donc $g$ peut générer des chaînes qui ne sont pas dans le langage.

   La réponse (d) n'est pas bonne car elle est trop restrictive en requérant une lettre autre que `b` à la fois à la gauche et à la droite de chaque apparition de `b`. Par exemple, $g$ n'arrive pas à générer `bcab`.

3. La réponse est (d). Cette expression régulière est trop restrictive car elle exige qu'on retrouve exactement quatre chiffres non-nuls dans la chaîne. Donc, les chaînes acceptables `31` et `12345` ne peuvent être générées par l'expression régulière.

4. La réponse est (a). On remarque que l'ouverture `/*` et la fermeture `*/` des commentaires sont exigées. À l'intérieur des commentaires, on permet que des astérisques apparaissent mais on s'assure que tout astérisque ou suite d'astérisques soit suivi d'autre chose que la barre oblique. Il n'y a pas de restriction supplémentaire, tel que suggéré dans la réponse (c).

5. La réponse est (b).

   La réponse (a) mentionne un non-déterminisme qui existe bel et bien ; il ne faudrait jamais qu'il y ait du non-déterminisme dans les diagrammes.

   La réponse (c) mentionne une transition où plus d'un caractère pourrait être consommé d'un coup ; ceci est interdit.

   La réponse (d) mentionne une transition qu'on pourrait effectuer en ne consommant aucun (zéro) caractère ; ceci est interdit.

   La réponse (e) propose un état non-acceptant qui comporte une étoile ; ceci est interdit.

6. La réponse est (a). On voit bien que c'est vers l'état 4 qu'il faut aller quand le caractère est '`*`'. Seules les réponses (a) et (d) proposent d'aller vers l'état 4. Toutefois, la réponse (d) propose ensuite de faire le test `(c != '/')`, ce qui n'est pas le bon test pour faire la transition vers l'état 5. La réponse (c) a, au premier abord, une certaine crédibilité car, en ne faisant rien lorsque le caractère est '`*`', on *laisserait* l'état continuer à être 4. Cependant, la proposition pour remplir la deuxième boîte n'a pas de sens : il n'existe pas de valeur `OTHER` qui serait capable d'être *égale* à n'importe quel caractère autre que '`*`' et '`/`'.

7. La réponse est (c). Voici une grammaire qui génère ce langage. On veut générer $u$ et $v^R$ en deux étapes : la première où $u$ et $v$ ont le même préfixe et la deuxième où on a obtenu une évidence claire que $u$ est bel et bien lexicographiquement plus petit que $v$.

$$
\begin{array}{rcll}
S & \to & \mathtt{a}\,S\,\mathtt{a} & \text{// Mêmes préfixes} \\
 & | & \mathtt{b}\,S\,\mathtt{b} & \\
 & | & \mathtt{a}\,A\,\mathtt{b} & \text{// Lexicographiquement plus petit} \\
 & | & D\,\mathtt{a} & \\
 & | & D\,\mathtt{b} & \\
A & \to & \mathtt{a}\,A\,\mathtt{a} & \text{// N'importe quelle suite} \\
 & | & \mathtt{a}\,A\,\mathtt{b} & \\
 & | & \mathtt{b}\,A\,\mathtt{a} & \\
 & | & \mathtt{b}\,A\,\mathtt{b} & \\
 & | & \mathtt{a}\,G & \\
 & | & \mathtt{b}\,G & \\
 & | & D\,\mathtt{a} & \\
 & | & D\,\mathtt{b} & \\
 & | & \mathtt{c} & \\
D & \to & D\,\mathtt{a} & \text{// N'importe quoi à droite} \\
 & | & D\,\mathtt{b} & \\
 & | & \mathtt{c} & \\
G & \to & \mathtt{a}\,G & \text{// N'importe quoi à gauche} \\
 & | & \mathtt{b}\,G & \\
 & | & \mathtt{c} &
\end{array}
$$

8. La réponse est (a). Si on utilise la première $S$-production, alors on est condamné à générer une chaîne qui a plus de '`a`' que de '`b`'. C'est l'inverse si on utilise la deuxième $S$-production.

   La réponse (b) propose un langage dont l'invariant est trop complexe pour être vérifié par une grammaire hors-contexte.

   La réponse (c) est clairement fausse : la chaîne '`a`' peut être générée par la grammaire.

   La réponse (d) est clairement fausse : ni la chaîne $\epsilon$, ni la chaîne `ab` peuvent être générées par la grammaire.

   La réponse (e) peut être déterminée comme étant fausse si on considère l'utilisation de la première $S$-production et qu'on inspecte les $A$-productions.

9. La réponse est (b). C'est vrai car le cas récursif de la suite de Fibonacci détermine le prochain nombre en se basant seulement sur les deux nombres précédents. Donc, une base d'induction pour $\texttt{a}^{f_0}$ et $\texttt{a}^{f_1}$ aurait suffi.

   Le choix de réponse (c) ne contredit pas le théorème. En effet, l'énoncé du théorème affirme ce que *peuvent* générer les non-terminaux $A$, $B$ et $C$. Il n'affirme rien à propos de ce que ces non-terminaux ne *peuvent pas* générer. C'est la même idée à propos du choix de réponse (e).

10. La réponse est (c). Les deux non-terminaux spécialisés *matched_stmt* et *open_stmt* continuent de générer des énoncés fermés et des énoncés ouverts, respectivement. Toutefois, à cause du changement à la grammaire, le non-terminal *open_stmt* permet de générer des énoncés ouverts de manière ambiguë. Voici deux dérivations à gauche d'abord différentes qui génèrent la même chaîne. (Notez que nous ignorons le non-terminal *expr* puisqu'il n'a aucun rôle à jouer dans la démonstration de l'ambiguïté.)

    *stmt* ⇒ *open_stmt*
    ⇒ **if** *expr* **then** *stmt*
    ⇒ **if** *expr* **then** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then** *stmt* **else** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then** *matched_stmt* **else** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else if** *expr* **then** *stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else if** *expr* **then** *matched_stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else if** *expr* **then other**

    *stmt* ⇒ *open_stmt*
    ⇒ **if** *expr* **then** *stmt* **else** *open_stmt*
    ⇒ **if** *expr* **then** *open_stmt* **else** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then** *stmt* **else** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then** *matched_stmt* **else** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else** *open_stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else if** *expr* **then** *stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else if** *expr* **then** *matched_stmt*
    ⇒ **if** *expr* **then if** *expr* **then other else if** *expr* **then other**

11. La réponse est (d). La logique de cette grammaire est que les nombres $i$ et $j$ fixent le nombre de fois où on doit utiliser la deuxième $S$-production, puis le nombre de fois où on doit générer deux fois plus de '`a`' que de '`b`' (ou deux fois plus de '`b`' que de '`a`').

    Chez les choix de réponse (a) et (c), on a deux arbres de dérivation différents pour la chaîne `aaabb`.

    Chez le choix de réponse (b), on a deux arbres de dérivation différents pour la chaîne `aaaaabbb`.

12. Remarquons le fait que les $F$-productions et les $A$-productions sont restées inchangées. Dans le cas des $A$-productions, ça ne nous donne pas d'indice car aucun membre droit commence par un non-terminal. Au contraire, le fait que les $F$-productions soient restées inchangées nous informe que $A$ doit être placé après $F$ dans l'ordre des non-terminaux. Ceci élimine les choix de réponses (c) et (d).

    Maintenant, remarquons que les $E$-productions ont été modifiées mais uniquement à cause de la récursion immédiate, pas à cause qu'il aurait fallu éliminer les récursions à gauche de $E$ vers $T$. Donc, $E$ est placé avant $T$ dans l'ordre des non-terminaux. Ceci élimine les choix de réponses (b) et (e) des choix restants.

    Finalement, à cause des modifications faites aux $T$-productions, on constate que le non-terminal $A$ devait être placé avant dans la liste, puisqu'on retrouve tous les membres droits des $A$-productions chez les nouvelles $T$-productions. En effet, la production $T \to F$ doit d'abord être transformée, entre autres, en la production $T \to A$. C'est cette dernière qui doit être à nouveau transformée en utilisant toutes les $A$-productions.

    Le bon ordre sur les non-terminaux est donc $F$, $A$, $E$, $T$, qui correspond au choix de réponse (a).

13. La bonne réponse est (a). Dans tous les autres cas, il y aurait eu une préfixe quelconque à factoriser, chez les $S$-productions.

14. La réponse est (c). Il suffit de regarder les productions de la grammaire et d'appliquer la recette d'écriture du pseudo-code.

15. La réponse est (a). Si on regarde la trace, les modifications apportées à la pile entre la première et la deuxième lignes forcent la production à la case $M[T, \mathbf{float}]$ à être $T \to B\,C$. Aussi, les modifications apportées à la pile entre la deuxième et la troisième lignes forcent la production à la case $M[B, \mathbf{float}]$ à être $B \to \mathbf{float}$. Il n'y a que le choix de réponse (a) qui corresponde à ces deux constats.

16. Posons tout d'abord la contrainte qui détermine FIRST($A$).

$$\begin{aligned}\text{FIRST}(A) &= \text{FIRST}(\mathtt{c}\,E) \cup \text{FIRST}(B\,\mathtt{e}\,A)\\ &= \{\mathtt{c}\} \cup \text{FIRST}(B\,\mathtt{e}\,A)\end{aligned}$$

Maintenant, pour calculer FIRST($B\,\mathtt{e}\,A$), nous devons calculer FIRST($B$). Posons donc la contrainte pour calculer FIRST($B$).

$$\begin{aligned}\text{FIRST}(B) &= \text{FIRST}(A\,\mathtt{e}\,B) \cup \text{FIRST}(\mathtt{d}\,E) \cup \text{FIRST}(\epsilon)\\ &= \text{FIRST}(A\,\mathtt{e}\,B) \cup \{\mathtt{d}\} \cup \{\epsilon\}\\ &= \text{FIRST}(A\,\mathtt{e}\,B) \cup \{\mathtt{d}, \epsilon\}\end{aligned}$$

Nous constatons que FIRST($A$) et FIRST($B$) sont mutuellement dépendants. Une observation que l'on peut faire, c'est que $\epsilon \in \text{FIRST}(B)$. Ceci nous permet de faire du progrès dans le calcul de FIRST($A$).

$$\begin{aligned}\text{FIRST}(A) &= \{\mathtt{c}\} \cup \text{FIRST}(B\,\mathtt{e}\,A)\\ &= \{\mathtt{c}\} \cup \Big(\text{FIRST}(B) - \{\epsilon\}\Big) \cup \text{FIRST}(\mathtt{e}\,A)\\ &= \{\mathtt{c}\} \cup \Big(\text{FIRST}(B) - \{\epsilon\}\Big) \cup \{\mathtt{e}\}\\ &= \{\mathtt{c}, \mathtt{e}\} \cup (\text{FIRST}(B) - \{\epsilon\})\end{aligned}$$

Cette nouvelle version de la contrainte nous permet de conclure que $\epsilon \notin \text{FIRST}(A)$. Ceci nous permet maintenant de faire du progrès dans le calcul de FIRST($B$).

$$\begin{aligned}\text{FIRST}(B) &= \text{FIRST}(A\,\mathtt{e}\,B) \cup \{\mathtt{d}, \epsilon\}\\ &= \text{FIRST}(A) \cup \{\mathtt{d}, \epsilon\}\end{aligned}$$

Ayant posé ces deux contraintes, il est simple de calculer la plus petite solution. On obtient FIRST($A$) = $\{\mathtt{c}, \mathtt{d}, \mathtt{e}\}$ et FIRST($B$) = $\{\mathtt{c}, \mathtt{d}, \mathtt{e}, \epsilon\}$.

17. Pour déterminer un ensemble FOLLOW quelconque, il faut malheureusement poser les contraintes concernant tous les ensembles FOLLOW et calculer la plus petite solution.

La première règle pour poser une contrainte concerne le symbole de départ.

$$\text{FOLLOW}(E) \supseteq \{\$\}$$

La deuxième règle pose une contrainte par non-terminal qu'on retrouve dans les membres droits.

$$\begin{aligned}
\text{FOLLOW}(E) &\supseteq \text{FIRST}(\epsilon) - \{\epsilon\} = \{\epsilon\} - \{\epsilon\} = \emptyset \\
\text{FOLLOW}(A) &\supseteq \text{FIRST}(\texttt{c}) - \{\epsilon\} = \{\texttt{c}\} - \{\epsilon\} = \{\texttt{c}\} \\
\text{FOLLOW}(B) &\supseteq \text{FIRST}(B\,\texttt{a}) - \{\epsilon\} \\
&= \Big(\big(\text{FIRST}(B) - \{\epsilon\}\big) \cup \text{FIRST}(\texttt{a})\Big) - \{\epsilon\} = \{\texttt{a}, \texttt{c}, \texttt{d}, \texttt{e}\} \\
\text{FOLLOW}(B) &\supseteq \text{FIRST}(\texttt{a}) - \{\epsilon\} = \{\texttt{a}\} - \{\epsilon\} = \{\texttt{a}\} \\
\text{FOLLOW}(E) &\supseteq \text{FIRST}(\epsilon) - \{\epsilon\} = \{\epsilon\} - \{\epsilon\} = \emptyset \\
\text{FOLLOW}(B) &\supseteq \text{FIRST}(\texttt{e}\,A) - \{\epsilon\} = \{\texttt{e}\} - \{\epsilon\} = \{\texttt{e}\} \\
\text{FOLLOW}(A) &\supseteq \text{FIRST}(\epsilon) - \{\epsilon\} = \{\epsilon\} - \{\epsilon\} = \emptyset \\
\text{FOLLOW}(A) &\supseteq \text{FIRST}(\texttt{e}\,B) - \{\epsilon\} = \{\texttt{e}\} - \{\epsilon\} = \{\texttt{e}\} \\
\text{FOLLOW}(B) &\supseteq \text{FIRST}(\epsilon) - \{\epsilon\} = \{\epsilon\} - \{\epsilon\} = \emptyset \\
\text{FOLLOW}(E) &\supseteq \text{FIRST}(\epsilon) - \{\epsilon\} = \{\epsilon\} - \{\epsilon\} = \emptyset
\end{aligned}$$

La troisième règle pose une contrainte par non-terminal qu'on retrouve à la fin d'un membre droit.

$$\begin{aligned}
\text{FOLLOW}(E) &\supseteq \text{FOLLOW}(E) \\
\text{FOLLOW}(E) &\supseteq \text{FOLLOW}(A) \\
\text{FOLLOW}(A) &\supseteq \text{FOLLOW}(A) \\
\text{FOLLOW}(B) &\supseteq \text{FOLLOW}(B) \\
\text{FOLLOW}(E) &\supseteq \text{FOLLOW}(B)
\end{aligned}$$

Grâce à ces contraintes, on peut calculer la plus petite solution aux ensembles FOLLOW.

| **Symbole** | FOLLOW |
|---|---|
| $E$ | $\{\texttt{a}, \texttt{c}, \texttt{d}, \texttt{e}, \$\}$ |
| $A$ | $\{\texttt{c}, \texttt{e}\}$ |
| $B$ | $\{\texttt{a}, \texttt{c}, \texttt{d}, \texttt{e}\}$ |

18. La réponse est (c).

    Il n'y a qu'une seule $T$-production, son membre droit n'est pas annulable et on a déjà introduit la production dans toutes les cases $M[T, a]$, où $a \in \text{PREDICT}(T \to F\,T') = \text{FIRST}(F\,T') - \{\epsilon\} = \text{FIRST}(F)$. Il n'y a donc rien dans la case $M[T, \$]$.

    La production $T' \to \epsilon$ doit être insérée dans la case $M[T', \$]$ car $\$ \in \text{PREDICT}(T' \to \epsilon) = \Big(\text{FIRST}(\epsilon) - \{\epsilon\}\Big) \cup \text{FOLLOW}(T') = \text{FOLLOW}(T') = \{\$\}$.

    Ces constats sont suffisants pour identifier le bon choix de réponse.

19. La réponse est (c).

20. La réponse est (d). C'est flagrant quand on exécute le programme.