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

Danny Dubé
Version : Hiver 2014

Exercices reliés au chapitre 2

Exercices

Voici les exercices que je recommande de faire :

— **Exercice 2.2.1**. Il s'agit d'une description *informelle* du langage qui est demandée dans l'exercice 2.2.1 (c). (Exercice 2.1 dans la 1ère édition.)
— **Exercices 2.2.2** et **2.2.3**. (Exercices 2.2 et 2.3 dans la 1ère édition.)
— **Exercice 2.2.4**. Votre justification n'a pas nécessairement à être formelle et mathématique. Elle doit au moins être une argumentation assez serrée pour que le lecteur soit forcé d'agréer avec la justification. (Exercice 2.4 dans la 1ère édition.)
— **Exercice 2.2.5**.
— **Exercice 2.2.6**. Incluez les nombres romains suivants :

Lettre	Valeur
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

On peut générer un nombre romain en retirant successivement la plus grande valeur possible ; par exemple, une boucle pourrait procéder ainsi :

> "si le nombre est supérieur ou égal à 1000, soustraire 1000 et concaténer `M` ; sinon, si le nombre est supérieur ou égal à 500, soustraire 500 et concaténer `D` ; sinon ..."

Il y a toutefois une règle supplémentaire qui est communément utilisée afin de réduire la taille des nombres romains générés ; lorsque le rapport entre une lettre et la lettre suivante est de 5 (par exemple, entre `I` et `V`, ou encore entre `X` et `L`) ou supérieur, on utilise une écriture soustractive pour éviter de devoir répéter trop de fois de suite le même symbole : `IV` à la place de `IIII`, `IX` à la place de `VIIII`, etc. Notez que la notation soustractive ne s'applique *pas* lorsque le rapport entre deux lettres est de 2 (par exemple, V et X : impossible donc d'écrire `VX`) [1].
Notez qu'après avoir écrit (par exemple) `XL` (40), il n'est plus possible d'ajouter de lettres supérieures ou égales à `X`, la suite ne peut contenir que des `V` ou des `I` – par

1. Adapté de `http://en.wikipedia.org/wiki/Roman_numerals`

exemple, on ne pourrait pas écrire `XLX` (le même principe s'applique pour toutes les lettres).
Commencez par écrire une grammaire qui ne tient pas compte de la notation soustractive ; par la suite, étendez votre grammaire pour permettre de générer des nombres utilisant cette notation.
Nous ne supporterons pas ici la notation soustractive à plus d'un nombre, comme `IIX` pour 8.
N'essayez pas de gérer le caractère optimal ou non d'un nombre romain au niveau de la grammaire (par exemple `IIIII` pourrait être écrit bien plus simplement pas `V` ; toutefois, tenter de forcer les nombres à être optimaux au niveau de la grammaire résulterait en une explosion du nombre d'états).

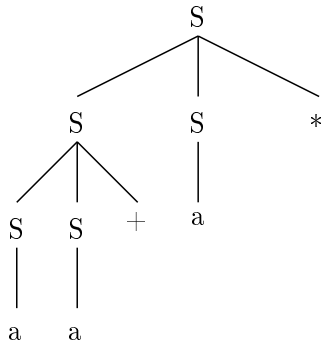
Réponses

2.2.1

— (a) La chaîne $aa + a*$ peut être générée par la dérivation suivante :

$$\begin{array}{lllll} S & \rightarrow SS* & \rightarrow SS+S* & \rightarrow aS+S* \\ & \rightarrow aa+S* & \rightarrow aa+a* \end{array}$$

— (b)



— (c) Il s'agit du langage des a additionnés et/ou multipliés, en notation mathématique postfixe (notation Polonaise inversée). En effet, la notation postfixe utilise la forme $SS+$ pour les additions et $SS*$ pour les multiplications, où S est une sous-expression. Dans le cas de cette grammaire, les sous-expressions ne peuvent être que des additions, des multiplications ou a.

2.2.2

— (a) La grammaire génère le langage 0^n1^n avec $n \geq 1$.

Justification :
- toutes les productions insèrent des `0` à gauche et des `1` à droite, donc on obtient nécessairement une chaîne commençant par des `0` et finissant par des `1` ;
- toutes les productions insèrent un nombre égal de `0` et de `1`, donc le nombre de `0` est nécessairement égal au nombre de `1`
- la seule production ne contenant pas de non-terminaux est $S \rightarrow$ 01, donc tous les mots générés contiennent nécessairement au moins un `0` et un `1`.

— (b)
Le langage des `a` additionnés et/ou multipliés, en notation mathématique préfixe.

Justification : La notation préfixe utilise la forme $+SS$ pour les additions et $*SS$ pour les multiplications, où S est une sous-expression. Dans le cas de cette grammaire, les sous-expressions ne peuvent être que des additions, des multiplications ou `a`.

— (c) Le langage des chaînes qui sont formées de parenthèses ouvrantes et fermantes et qui sont bien balancées.

Justification : la grammaire insère toujours à la fois une parenthèse ouvrante et une parenthèse fermante dans cet ordre : `(`, `)`. De plus, elle permet la même construction avant, après et dans les parenthèses.

— (d) Le langage des chaînes qui sont formées de lettres `a` et de `b` en n'importe quel ordre mais en nombre égal.

Justification : la grammaire insère toujours à la fois un `a` et un `b`

— (e) Le langage des expressions mathématiques qui peuvent additionner, juxtaposer (aa) et/ou appliquer l'opérateur postfixe $*$ ($a*$) à une ou plusieurs variables `a`, possiblement avec des parenthèses. Il pourrait s'agir d'un langage représentant des expressions régulières, où l'opérateur `+` représente l'union.

2.2.3

(c), (d) et (e)

2.2.4

— (a)

$$S \rightarrow SS+ \mid SS- \mid SS* \mid SS/ \mid \textbf{number}$$

— (b)

$$S \rightarrow S, \textbf{id} \mid \textbf{id}$$

— (c)

$$S \rightarrow \textbf{id}, S \mid \textbf{id}$$

— (d) *Explication :* afin d'éliminer toute ambiguité, la priorité des opérateurs est tenue en compte dans cette grammaire. La grammaire traite en premier les opérateurs $+$ et $-$, qui sont les moins prioritaires ; puis elle "descend" traiter les termes (`T`), qui sont constitués des opérateurs $*$ et $/$, qui sont plus prioritaires. Finalement, on identifie les opérandes du terme (ou le terme entier) comme étant une valeur (`V`) soit, pour cet exercice, un entier ou un identificateur.

$$\begin{array}{lcl} S & \rightarrow & S+T \mid S-T \mid T \\ T & \rightarrow & T*V \mid T/V \mid V \\ V & \rightarrow & \textbf{number} \mid \textbf{id} \end{array}$$

— (e)

$$\begin{array}{lcl} S & \rightarrow & S+T \mid S-T \mid T \\ T & \rightarrow & T*V \mid T/V \mid V \\ V & \rightarrow & \textbf{number} \mid \textbf{id} \mid +V \mid -V \end{array}$$

2.2.5 (a)

Cas de base

Les nombres binaires `11` et `1001`, qui correspondent respectivement à 3 et 9, sont tous deux divisibles par 3.

Induction sur num → num `0`

Si "**num**" est divisible par 3, alors "**num** `0`" l'est aussi. Preuve : soit **num** $= 3x$ où x est entier (un tel x existe car, par hypothèse d'induction, **num** est un multiple de 3) ; or, en binaire, ajouter un `0` à la droite d'un nombre correspond à le multiplier par deux ; donc on sait que **num** `0` $= 6x$. On constate que "**num** `0`" est divisible par 6, et donc est divisible par 3.

Induction sur num → $\mathbf{num_1}$ $\mathbf{num_2}$

Si "$\mathbf{num_1}$" et "$\mathbf{num_2}$" sont tous deux divisibles par 3, alors "$\mathbf{num_1}$ $\mathbf{num_2}$" l'est aussi. Preuve :

- Soit `len(a)`, une fonction retournant le nombre de caractères dans 'a', un mot composé de `0` et de `1`. (Par exemple, `len(011)` = 3, et `len(01110)` = 5).
- La concaténation de caractères "$\mathbf{num_1}$ $\mathbf{num_2}$" peut se réécrire sous la forme mathématique suivante, si on considère $\mathbf{num_1}$ et $\mathbf{num_2}$ comme des nombres et non plus comme des suites de caractères : "$\mathbf{num_1} * 2^{\mathbf{len(num_2)}} + \mathbf{num_2}$"
- Puisque $\mathbf{num_1}$ est divisible par 3 (par hypothèse d'induction), alors $\mathbf{num_1} * 2^n$ est aussi divisible par 3 pour tout $n \geq 0$. Pour s'en convaincre, il suffit de poser $\mathbf{num_1} = 3x$; on voir alors bien que $\mathbf{num_1} * 2^n = 3x * 2^n$ est divisible par 3.
- Effectuer l'addition $p = m + n$ où n et m divisibles par 3 donne un résultat p divisible par 3 ; pour s'en convaincre, il suffit de poser $m = 3x$ et $n = 3y$; alors on a $p = (x + y) * 3$ et on voit que p est bel et bien divisible par 3.
- Par conséquent, "$\mathbf{num_1} * 2^{\mathbf{len(num_2)}} + \mathbf{num_2}$", qui est une addition de deux nombres divisibles par 3, donne un résultat divisible par 3.

2.2.5 (b)

Non, car le nombre `10101` (21) est divisible par 3 mais n'est pas généré par la grammaire. En effet, `10101` ne contient ni `11`, ni `1001` ; or, on voit bien que toute chaîne générée par cette grammaire doit forcément contenir l'un ou l'autre.

2.2.6

Voici la première version de la grammaire, qui ne supporte pas la notation soustractive :

$$
\begin{aligned}
M &\rightarrow \text{`M'}\ M \mid D \\
D &\rightarrow \text{`D'}\ D \mid C \\
C &\rightarrow \text{`C'}\ C \mid L \\
L &\rightarrow \text{`L'}\ L \mid X \\
X &\rightarrow \text{`X'}\ X \mid V \\
V &\rightarrow \text{`V'}\ V \mid I \\
I &\rightarrow \text{`I'}\ I \mid \epsilon
\end{aligned}
$$

Voici la version améliorée qui gère la notation soustractive :

$$
\begin{aligned}
M &\rightarrow \text{`M'}\ M \mid \text{`IM'} \mid \text{`VM'}\ I \mid \text{`XM'}\ V \mid \text{`LM'}\ X \mid \text{`CM'}\ L \mid D \\
D &\rightarrow \text{`D'}\ D \mid \text{`ID'} \mid \text{`VD'}\ I \mid \text{`XD'}\ V \mid \text{`LD'}\ X \mid \text{`CD'}\ L \mid C \\
C &\rightarrow \text{`C'}\ C \mid \text{`IC'} \mid \text{`VC'}\ I \mid \text{`XC'}\ V \mid L \\
L &\rightarrow \text{`L'}\ L \mid \text{`IL'} \mid \text{`VL'}\ I \mid \text{`XL'}\ V \mid X \\
X &\rightarrow \text{`X'}\ X \mid \text{`IX'} \mid V \\
V &\rightarrow \text{`V'}\ V \mid \text{`IV'} \mid I \\
I &\rightarrow \text{`I'}\ I \mid \epsilon
\end{aligned}
$$