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

Danny Dubé
Hiver 2019
Version: 24 avril

Corrigé du travail pratique #2

Réponses

Note: les réponses aux différentes questions ne sont pas toujours uniques.

1. (a) On utilise la première des deux grammaires, puisqu'il n'y a pas d'ordre imposé sur les lettres dans les mots. On ajoute au non-terminal A les deux attributs synthétisés $A.b$ et $A.c$, qui comptent les symboles du même nom.

Productions	Règles sémantiques
$S \to A$	$S.ok := (A.b \geq 1) \wedge (A.c \geq 2)$
$A \to \texttt{a}\, A_1$	$A.b := A_1.b$ $\quad A.c := A_1.c$
$A \to \texttt{b}\, A_1$	$A.b := A_1.b + 1$ $\quad A.c := A_1.c$
$A \to \texttt{c}\, A_1$	$A.b := A_1.b$ $\quad A.c := A_1.c + 1$
$A \to \epsilon$	$A.b := 0$ $\quad A.c := 0$

Un cas positif possible provient de la chaîne `cbc` et un cas négatif possible provient de la chaîne ϵ. Ici, je ne dessine pas les arbres annotés; c'est trivial de faire cette partie de l'exercice.

(b) On utilise à nouveau la première des deux grammaires. On ajoute au non-terminal A les deux attributs synthétisés $A.c$ et $A.ok$. L'attribut $A.c$ indique le symbole qui débute le suffixe de la chaîne d'entrée généré par A. L'attribut $A.ok$ indique si aucune répétition interdite n'a été vue jusqu'à date, en traitant la chaîne d'entrée de droite à gauche. Notez que l'initialisation de $A.c$ se fait avec un symbole étranger, histoire de ne pas déclencher par accident la détection d'une répétition chez le parent.

Productions	Règles sémantiques
$S \to A$	$S.ok := A.ok$
$A \to \mathtt{a}\, A_1$	$A.c := \mathtt{a}$ $A.ok := A_1.ok \wedge (A_1.c \neq \mathtt{a})$
$A \to \mathtt{b}\, A_1$	$A.c := \mathtt{b}$ $A.ok := A_1.ok \wedge (A_1.c \neq \mathtt{b})$
$A \to \mathtt{c}\, A_1$	$A.c := \mathtt{c}$ $A.ok := A_1.ok \wedge (A_1.c \neq \mathtt{c})$
$A \to \epsilon$	$A.c := \mathtt{d}$ $A.ok :=$ vrai

Un cas positif possible provient de la chaîne ϵ et un cas négatif possible provient de la chaîne $\mathtt{aa}$.

(c) Cette fois, il est préférable d'utiliser la seconde grammaire. À nouveau, il suffit de faire du comptage. À cette fin, on définit les attributs $A.a$, $A.b$, $A.c$, $B.b$, $B.c$ et $C.c$.

Productions	Règles sémantiques		
$S \to A$	$S.ok := (247 + 12 \cdot A.a = 23 \cdot A.b + 7 \cdot A.c)$		
$A \to \mathtt{a}\, A_1$	$A.a := A_1.a + 1$	$A.b := A_1.b$	$A.c := A_1.c$
$A \to B$	$A.a := 0$	$A.b := B.b$	$A.c := B.c$
$B \to \mathtt{b}\, B_1$		$B.b := B_1.b + 1$	$B.c := B_1.c$
$B \to C$		$B.b := 0$	$B.c := C.c$
$C \to \mathtt{c}\, C_1$			$C.c := C_1.c + 1$
$C \to \epsilon$			$C.c := 0$

Dans cet exercice, ce qui est difficile, c'est de trouver une chaîne dans L_3. Comme le but du cours n'est pas de faire de la théorie des nombres, je ne m'attarderai pas aux moyens à utiliser pour trouver une solution ou, en général, l'ensemble des solutions. En cherchant un peu, on peut trouver une solution comme celle-ci: $i = 0$, $j = 8$ et $k = 9$. Ceci donne la chaîne $\mathtt{b}^8\mathtt{c}^9$ comme cas positif possible. Un cas négatif possible provient de la chaîne ϵ.

(d) On revient à la première grammaire. La solution qu'on présente ici n'utilise pas de fonction auxiliaire; histoire de montrer que c'était faisable ainsi. On utilise une combinaison d'attributs hérités et synthétisés. En descendant dans l'arbre, on compte le nombre d'apparitions de la chaîne `bac`. En remontant dans l'arbre, on copie le nombre total d'apparitions de `bac`, on énumère les nombres de Fibonacci successifs et on vérifie s'il y a égalité. Durant la descente, l'attribut $A.k$ compte combien d'apparitions de `bac` on a vu jusqu'à ce point. L'attribut $A.l$ indique la longueur du plus long préfixe de `bac` qui est épelé par les symboles immédiatement à gauche (i.e. chez les frères des ancêtres les plus près à partir du point courant dans l'arbre). Il n'y a que $A.k$ et $A.l$ qui sont hérités. Maintenant, l'attribut synthétisé $A.n$ indique le nombre total d'apparitions de `bac` dans la chaîne d'entrée. Les attributs $A.f$ et $A.g$ contiennent respectivement un nombre de la suite de Fibonacci et le précédent. Finalement, l'attribut $A.ok$ indique si on a observé quelque part l'égalité entre $A.n$ et $A.f$. La suite de Fibonacci est visitée en remontant dans l'arbre et suffisamment d'éléments de la suite sont visités pour s'assurer de tester avec exactitude si $A.n$ est un nombre de Fibonacci.

Productions	Règles sémantiques
$S \to A$	$A.k := 0$
	$A.l := 0$
	$S.ok := A.ok$
$A \to \mathtt{a}\, A_1$	$A_1.k := \textbf{if } A.l = 3 \textbf{ then } A.k + 1 \textbf{ else } A.k$
	$A_1.l := \textbf{if } A.l = 1 \textbf{ then } 2 \textbf{ else } 0$ // $\mathtt{b} \mapsto \mathtt{ba}$?
	$A.n := A_1.n$
	$A.f := A_1.f + A_1.g$
	$A.g := A_1.f$
	$A.ok := A_1.ok \vee (A.n = A.g)$
$A \to \mathtt{b}\, A_1$	$A_1.k := \textbf{if } A.l = 3 \textbf{ then } A.k + 1 \textbf{ else } A.k$
	$A_1.l := 1$ // $w \mapsto \mathtt{b}$
	$A.n := A_1.n$
	$A.f := A_1.f + A_1.g$
	$A.g := A_1.f$
	$A.ok := A_1.ok \vee (A.n = A.g)$
$A \to \mathtt{c}\, A_1$	$A_1.k := \textbf{if } A.l = 3 \textbf{ then } A.k + 1 \textbf{ else } A.k$
	$A_1.l := \textbf{if } A.l = 2 \textbf{ then } 3 \textbf{ else } 0$ // $\mathtt{ba} \mapsto \mathtt{bac}$?
	$A.n := A_1.n$
	$A.f := A_1.f + A_1.g$
	$A.g := A_1.f$
	$A.ok := A_1.ok \vee (A.n = A.g)$
$A \to \epsilon$	$A.n := \textbf{if } A.l = 3 \textbf{ then } A.k + 1 \textbf{ else } A.k$
	$A.f := 1$
	$A.g := 0$
	$A.ok := (A.n = A.g)$

Un cas positif possible provient de la chaîne ϵ. Un cas négatif possible provient de la chaîne `bacbacbacbac`.

(e) La stratégie employée ici consiste à accumuler deux séquences de bits: celle de gauche, qui est l'encodage, selon C, des symboles de Σ_5 et celle de droite, qui est directement la suite de bits présente dans la chaîne d'entrée. Les attributs synthétisés $S.g$ et $S.d$ contiennent les séquences accumulées de gauche et de droite, respectivement. L'attribut $T.d$ est similaire à l'attribut $S.d$.

Productions	Règles sémantiques		
$S \to \mathtt{a}\, S_1$	$S.ok := (S.g = S.d)$	$S.g := C(\mathtt{a}) \cdot S_1.g$	$S.d := S_1.d$
$S \to \mathtt{b}\, S_1$	$S.ok := (S.g = S.d)$	$S.g := C(\mathtt{b}) \cdot S_1.g$	$S.d := S_1.d$
$S \to \mathtt{c}\, S_1$	$S.ok := (S.g = S.d)$	$S.g := C(\mathtt{c}) \cdot S_1.g$	$S.d := S_1.d$
$S \to \mathtt{d}\, S_1$	$S.ok := (S.g = S.d)$	$S.g := C(\mathtt{d}) \cdot S_1.g$	$S.d := S_1.d$
$S \to :T$	$S.ok := (S.g = S.d)$	$S.g := \epsilon$	$S.d := T.d$
$T \to \mathtt{0}\, T_1$			$T.d := \mathtt{0} \cdot T_1.d$
$T \to \mathtt{1}\, T_1$			$T.d := \mathtt{1} \cdot T_1.d$
$T \to \epsilon$			$T.d := \epsilon$

Un cas positif possible provient de la chaîne ':'. Un cas négatif possible provient de la chaîne ': 0'.

2. Tout arbre généré par T doit respecter l'invariant d'ordonnancement et l'invariant de profondeur égale. On ajoute à T les attributs hérités *min* et *max* afin d'indiquer aux sous-arbres les bornes entre lesquelles leurs clés doivent se trouver afin de respecter l'ordonnancement. On ajoute à T l'attribut synthétisé $T.h$, lequel indique la hauteur de l'arbre généré par T. Enfin, on ajoute à T l'attribut synthétisé $T.ok$, lequel indique si l'arbre généré respecte les deux invariants.

Productions	Règles sémantiques	
$S \to T$	$T.min := -\infty$;	$T.max := +\infty$
	$S.ok := T.ok$	
$T \to [\, T_1$, **num** , $T_2\,]$	$T_1.min := T.min$;	$T_1.max := \mathbf{num}.lexval$
	$T_2.min := \mathbf{num}.lexval$;	$T_2.max := T.max$
	$T.h := T_1.h + 1$	
	$T.ok := T_1.ok \ \wedge \ T_2.ok \ \wedge \ (T_1.h = T_2.h)$	
$T \to [\, T_1$, $\mathbf{num}_1$, T_2 , $\mathbf{num}_2$, $T_3\,]$	$T_1.min := T.min$;	$T_1.max := \mathbf{num}_1.lexval$
	$T_2.min := \mathbf{num}_1.lexval$;	$T_2.max := \mathbf{num}_2.lexval$
	$T_3.min := \mathbf{num}_2.lexval$;	$T_3.max := T.max$
	$T.h := T_1.h + 1$	
	$T.ok := T_1.ok \ \wedge \ T_2.ok \ \wedge \ T_3.ok \ \wedge$	
	$(T_1.h = T_2.h) \ \wedge \ (T_2.h = T_3.h)$	
$T \to []$	$T.h := 0$;	$T.ok := T.min \leq T.max$

Il faut noter que la vérification du bon ordonnancement des clés est déléguée aux arbres vides. Toute inversion dans l'ordonnancement est dûment détectée grâce à la présence d'un arbre vide où la borne inférieure est plus grande que la borne supérieure.

3. Seules les E-productions et les T-productions doivent être modifiées. En effet, les non-terminaux F et A ne souffrent pas de récursion à gauche. De plus, le fait d'éliminer la récursion à gauche dans les E-productions et les T-productions ne change pas les "interfaces" fournies par E et T; c'est-à-dire que E et T continuent de synthétiser $E.b$ et $T.b$, respectivement.

   Il faut établir quels sont les éléments dans le système de traduction qui correspondent aux éléments mentionnés dans les notes de cours.

   - Chez les E-productions, le non-terminal récursif à gauche est E, l'équivalent du non-terminal Y est la forme de phrase '**ou** T', l'équivalent du non-terminal X est le non-terminal T, la fonction g est le "ou" logique et la fonction f est l'identité.
   - Chez les T-productions, le non-terminal récursif à gauche est T, l'équivalent du non-terminal Y est la forme de phrase '**et** F', l'équivalent du non-terminal X est le non-terminal F, la fonction g est le "et" logique et la fonction f est l'identité.

$$
\begin{array}{rcll}
E & \to & T & \{E'.i := T.b\} \\
 & & E' & \{E.b := E'.s\} \\
E' & \to & \textbf{ou } T & \{E'_1.i := E'.i \vee T.b\} \\
 & & E'_1 & \{E'.s := E'_1.s\} \\
 & | & \epsilon & \{E'.s := E'.i\} \\
T & \to & F & \{T'.i := F.b\} \\
 & & T' & \{T.b := T'.s\} \\
T' & \to & \textbf{et } F & \{T'_1.i := T'.i \wedge F.b\} \\
 & & T'_1 & \{T'.s := T'_1.s\} \\
 & | & \epsilon & \{T'.s := T'.i\} \\
F & \to & \textbf{non } F_1 & \{F.b := \neg\, F_1.b\} \\
 & | & A & \{F.b := A.b\} \\
A & \to & (\ E\ ) & \{A.b := E.b\} \\
 & | & \textbf{faux} & \{A.b := \mathsf{faux}\} \\
 & | & \textbf{vrai} & \{A.b := \mathsf{vrai}\} \\
 & | & \textbf{id} & \{A.b := \text{get\_value}(\textbf{id}.\mathit{entry})\}
\end{array}
$$

4. Dans cette solution, nous nous efforçons de générer du code de qualité en ne faisant au plus qu'un saut par clé. Dans le cas de la dernière clé d'une clause, on fait un saut *s'il n'y a pas* égalité (et ce saut est vers la prochaine clause). Dans le cas des clés qui précèdent la dernière, on fait un saut *s'il y a* égalité (et ce saut est vers l'énoncé de la clause). Voici le patron de code que l'on générerait pour l'exemple (b).

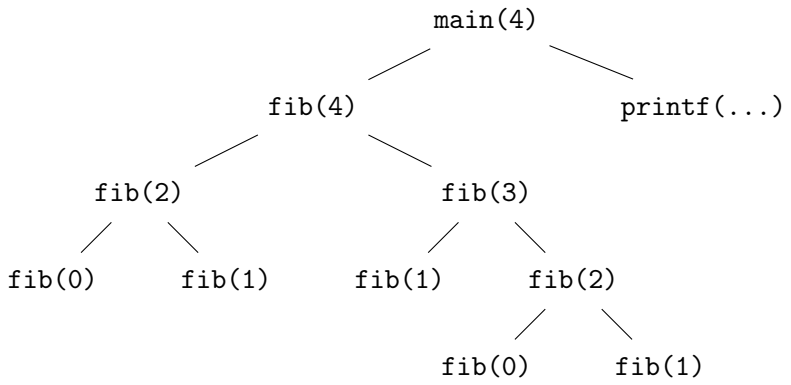
`E.code`	`if E.place = 3 goto Lok2`	`S3.code`
	`if E.place = 5 goto Lok2`	`Lexit:`
`if E.place ≠ 12 goto Lskip1`	`if E.place = 7 goto Lok2`	
`Lok1:`	`if E.place ≠ 11 goto Lskip2`	
`S1.code`	`Lok2:`	
`goto Lexit`	`S2.code`	
`Lskip1:`	`goto Lexit`	
	`Lskip2:`	

Dans la définition orientée-syntaxe qui suit:

- S synthétise l'attribut *code*;
- C hérite de la *place* contenant la valeur de E, synthétise l'étiquette de sortie *exit* et synthétise son *code* et
- K hérite de la *place* contenant la valeur de E, hérite de l'étiquette *skip* permettant de sauter par-dessus la clause courante, synthétise l'étiquette *ok* permettant de sauter à l'énoncé de la clause courante et synthétise son *code*.

Productions	**Règles sémantiques**
$S \rightarrow$ **switch** E **with** C	$C.place := E.place$ $S.code := E.code \parallel C.code$
$C \rightarrow$ **case** K S ; C_1	$K.place := C.place$ $K.skip :=$ newlabel() $C_1.place := C.place$ $C.exit := C_1.exit$ $C.code := K.code \parallel S.code \parallel$ gen(**goto** $C.exit$) $\parallel$ gen($K.skip$ **:**) $\parallel C_1.code$
$C \rightarrow$ **else** S	$C.exit :=$ newlabel() $C.code := S.code \parallel$ gen($C.exit$ **:**)
$C \rightarrow$ **end**	$C.exit :=$ newlabel() $C.code :=$ gen($C.exit$ **:**)
$K \rightarrow$ **num** , K_1	$K_1.place := K.place$ $K_1.skip := K.skip$ $K.ok := K_1.ok$ $K.code :=$ gen(**if** $K.place =$ **num**.*val* **goto** $K.ok$) $\parallel K_1.code$
$K \rightarrow$ **num** :	$K.ok :=$ newlabel() $K.code :=$ gen(**if** $K.place \neq$ **num**.*val* **goto** $K.skip$) $\parallel$ gen($K.ok$ **:**)

5.



6.

Variable	**Durée vie**	**Alloc.**	**Affect.**
t_1	1–10	reg.	R1
t_2	~~1–5~~ 2–5	reg.	R2
t_3	3–4	reg.	R3
t_4	4–5	reg.	R3
t_5	5–6	reg.	R3
t_6	6–10	reg.	R3
t_7	7–11	reg.	R2
t_8	8–9	reg.	R4
t_9	10–11	reg.	R1
a	$\forall$	mem.	—
b	$\forall$	mem.	—
c	$\forall$	mem.	—
z1	$\forall$	mem.	—
z2	$\forall$	mem.	—