First take-home assignment

Edda Steinunn Skúli Arnarsson

Due: January 6th 2018

Problem 1. *Define an unambiguous context-free grammar for the syntax the language of Roman numerals (from 1 to 3999). The terminals should be $T = \{i, v, x, l, c, d, m\}$. You are free to choose the set of non-terminals as you see fit. The start non-terminal should be R*

Solution. Consider context-free grammar G_1 that recognizes the language of Roman numerals from 1-3999. G_1 is formally defined as the 4-tuple:

$$G_1 = \{\{R, A, B, E, F, G, H, J\}, T, R_s, R\}$$

Where T is the set of terminals $T = \{i, v, x, l, c, d, m\}$ and R_s is set of rules described as follows:

$R \rightarrow ABEF \mid ABE \mid ABF \mid AEF \mid BEF \mid AB \mid AE \mid AF \mid BE \mid BF \mid EF \mid A \mid B \mid E \mid F$
$F \rightarrow G \mid vG \mid v \mid iv \mid ix$
$G \rightarrow i \mid ii \mid iii$
$E \rightarrow H \mid lH \mid l \mid xl \mid xc$
$H \rightarrow x \mid xx \mid xxx$
$B \rightarrow J \mid dJ \mid d \mid cd \mid cm$
$J \rightarrow c \mid cc \mid ccc$
$A \rightarrow m \mid mm \mid mmm$

The problem of determining whether any context-free language is ambiguous is proven to be undecidable so the un-ambiguity of G_1 is not verifiable. However, various valid strings in the language of roman numerals were derived from G_1 as an attempt to detect ambiguity and none were found to possibly be derived ambiguously.

Problem 2. *Give a denotational semantics for the language of Roman numerals by defining a function sem_R : $tree_R \to N$ where by $tree_R$ I mean the set of all derivation trees whose root is labelled with R (which of course represent strings derivable from R). Of course you will need similar functions for the other non-terminals. When defining these semantic functions, feel free to use standard means such as case distinction on the structure of the given tree.*

Solution. In order to give meaning to derivation trees of grammar G_1 in terms of natural numbers (which we will here recognize as decimals) we define the following eight functions:

$$\begin{gathered} sem_R := tree_R \to N \\ sem_F := tree_F \to N \\ sem_G := tree_G \to N \\ sem_E := tree_E \to N \\ sem_H := tree_H \to N \\ sem_B := tree_B \to N \\ sem_J := tree_J \to N \\ sem_A := tree_A \to N \end{gathered}$$

Where N stands for the set of natural numbers. These semantic functions are defined as follows:

$$\begin{aligned} &\mathrm{sem}_R(ABEF) = sem_A(A) + sem_B(B) + sem_E(E) + sem_F(F) \\ &sem_R(ABE) = sem_A(A) + sem_B(B) + sem_E(E) \\ &sem_R(ABF) = sem_A(A) + sem_B(B) + sem_F(F) \\ &sem_R(AEF) = sem_A(A) + sem_E(E) + sem_F(F) \\ &sem_R(BEF) = sem_B(B) + sem_E(E) + sem_F(F) \\ &sem_R(AB) = sem_A(A) + sem_B(B) \\ &sem_R(AE) = sem_A(A) + sem_E(E) \\ &sem_R(AF) = sem_A(A) + sem_F(F) \\ &sem_R(BE) = sem_B(B) + sem_E(E) \\ &sem_R(BF) = sem_B(B) + sem_F(F) \\ &sem_R(EF) = sem_E(E) + sem_F(F) \\ &sem_R(A) = sem_A(A) \\ &sem_R(B) = sem_B(B) \\ &sem_R(E) = sem_E(E) \\ &sem_R(F) = sem_F(F) \end{aligned}$$

$sem_F(v) = 5$
$sem_F(iv) = 4$
$sem_F(ix) = 9$
$sem_F(vG) = 5 + sem_G(G)$
$sem_F(G) = sem_G(G)$

$sem_G(i) = 1$
$sem_G(ii) = 2$
$sem_G(iii) = 3$

$sem_E(l) = 50$
$sem_E(xl) = 40$
$sem_E(xc) = 90$
$sem_E(H) = sem_H(H)$
$sem_E(lH) = 50 + sem_H(H)$

$sem_H(x) = 10$
$sem_H(xx) = 20$
$sem_H(xxx) = 30$

$sem_B(d) = 500$
$sem_B(cd) = 400$
$sem_B(cm) = 900$
$sem_B(J) = sem_J(J)$
$sem_B(dJ) = 500 + sem_J(J)$

$sem_J(c) = 100$
$sem_J(cc) = 200$
$sem_J(ccc) = 300$

$sem_A(m) = 1000$
$sem_A(mm) = 2000$
$sem_A(mmm) = 3000$

For testing purposes we map some strings in the range of $1 - 3999$ to decimal numbers via the above denotational semantics and check their correctness:

Derivation of mmcdxii *(correct decimal notation: 2412)*:

$$\begin{aligned}
&sem_R(mmcdxii) = sem_A(mm) + sem_B(cd) + sem_E(x) + sem_F(ii)\\
&\Rightarrow sem_R(mmcdxii) = 2000 + sem_B(cd) + sem_E(x) + sem_F(ii)\\
&\Rightarrow sem_R(mmcdxii) = 2000 + 400 + sem_E(x) + sem_F(ii)\\
&\Rightarrow sem_R(mmcdxii) = 2000 + 400 + sem_H(x) + sem_F(ii)\\
&\Rightarrow sem_R(mmcdxii) = 2000 + 400 + 10 + sem_F(ii)\\
&\Rightarrow sem_R(mmcdxii) = 2000 + 400 + 10 + sem_G(ii)\\
&\Rightarrow sem_R(mmcdxii) = 2000 + 400 + 10 + 2 \;=\; \boldsymbol{2412} \quad \square
\end{aligned}$$

Derivation of cdxxi *(correct decimal notation: 421)*:

$$\begin{aligned}
&sem_R(cdxxi) = sem_B(cd) + sem_E(xx) + sem_F(i)\\
&\Rightarrow sem_R(cdxxi) = 400 + sem_E(xx) + sem_F(i)\\
&\Rightarrow sem_R(cdxxi) = 400 + sem_H(xx) + sem_F(i)\\
&\Rightarrow sem_R(cdxxi) = 400 + 20 + sem_F(i)\\
&\Rightarrow sem_R(cdxxi) = 400 + 20 + sem_G(i)\\
&\Rightarrow sem_R(cdxxi) = 400 + 20 + 1 \;=\; \boldsymbol{421} \quad \square
\end{aligned}$$

Derivation of mmmcmx *(correct decimal notation: 3910)*

$$\begin{aligned}
&sem_R(mmmcmx) = sem_A(mmm) + sem_B(cm) + sem_E(x)\\
&\Rightarrow sem_R(mmmcmx) = 3000 + sem_B(cm) + sem_E(x)\\
&\Rightarrow sem_R(mmmcmx) = 3000 + 900 + sem_E(x)\\
&\Rightarrow sem_R(mmmcmx) = 3000 + 900 + sem_H(x)\\
&\Rightarrow sem_R(mmmcmx) = 3000 + 900 + 10 \;=\; \boldsymbol{3910} \quad \square
\end{aligned}$$

The denotational semantics appear to successfully map roman numerals to natural numbers (that is in this case, decimals).

Problem 3. $T = \{a, b, c\}$. *Define context-free grammars for the following sets of strings:*

(a) all strings with whose first letter is a, whose last letter is c and which contain exactly three occurrences of b (but can have any positive number of occurrences of a and c)

(b) all strings of the form $a^i b^j c^k$ *where* $i = j + k$

(c) all strings that contain abba as a substring

Solution.

a. Consider context-free grammar G_2 that recognizes all strings on the form:

$$\{w \mid w \text{ starts with } a\text{, ends with } c \text{ and contains exactly three b's}\}$$

G_2 is formally defined as the 4-tuple $G_2 = \{\{S_2, U\}, T, R_2, S_2\}$ Where T is the set of terminals $T = \{a, b, c\}$ and R_2 is set of rules described as follows:

$$S_2 \rightarrow aUbUbUbUc$$
$$U \rightarrow Ua \mid Uc \mid \varepsilon$$

b. Consider context-free grammar G_3 that recognizes all strings on the form:

$$\{w \mid w := a^*b^*c^*\text{, where the number of a's equal the sum of the number of b's and c's}\}$$

G_3 is formally defined as the 4-tuple $G_3 = \{\{S_3, V, X\}, T, R_3, S_3\}$ Where T is the set of terminals $T = \{a, b, c\}$ and R_3 is set of rules described as follows:

$$S_3 \rightarrow V \mid X \mid \varepsilon$$
$$V \rightarrow aVc \mid X$$
$$X \rightarrow aXb \mid \varepsilon$$

c. Consider context-free grammar G_4 that recognizes all strings on the form:

$$\{w \mid w \text{ contains the substring } abba\}$$

G_4 is formally defined as the 4-tuple $G_4 = \{\{S_4, U\}, T, R_4, S_4\}$ Where T is the set of terminals $T = \{a, b, c\}$ and R_4 is set of rules described as follows:

$$S_4 \rightarrow UabbaU$$
$$U \rightarrow aU \mid bU \mid cU \mid \varepsilon$$

Problem 4. *Consider the context-free grammar* $(\{S\}, \{a, b\}, R, S)$ *where* R *is this set of rules:*

$$S \rightarrow$$
$$S \rightarrow SaSb$$

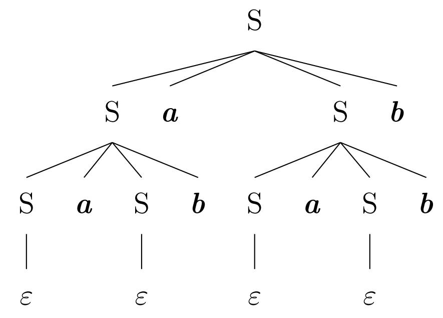
(a) Write down a derivation tree for the string abaabb. Is this the only derivation tree possible for this string or can you find several?

(b) Argue that there cannot be a derivation tree for the string abbaab

(c) Find a string that has multiple derivation trees or argue that there can be no such string

Solution.

a. The following derivation tree shows how the string **abaabb** is derived from the context-free grammar:



No other from of a derivation tree is possible as only one derivation sequence is possible:

1. **S**: the start symbol S **must** initially apply the rule $S \rightarrow SaSb$ and not $S \rightarrow \varepsilon$ in order for the string to be non-empty

2. **SaSb**: We observe now that our desired string has the prefix "ab". We further observe that in order to have a prefix of single a precede a b, the leading S variable in our string must generate a sub-string with the prefix ab. Contrarily, if the first S were to become ε, the string could never have this ab prefix as the second S would always create the irreversible prefix "aa". Therefore the the rule $S \rightarrow SaSb$ **must** be applied to the first S variable of our current derived string.

3. **SaSbaSb**: Now to have our newly derived have the prefix "ab" we have two options; applying the rule $S \rightarrow \varepsilon$ to the first S, or to continue the derivation of the first S by applying the rule $S \rightarrow SaSb$ to it. However, if we were to apply the rule $S \rightarrow SaSb$ to the first S, we could never get our desired suffix "$aabb$" from the remaining symbols $aSbaSb$ as the suffix must then be on the form $a^*b^*a^*b$ which can never equal to $aabb$. Therefore the rule $S \rightarrow \varepsilon$ **must** be applied to the first S.

4. ε**aSbaSb**: Further, the rule $S \rightarrow \varepsilon$ **must** also apply to the second S to generate our desired prefix "ab" instead of the irreversible prefix "aa" which is inevitable if we apply another rule to that S.

5. ε**a**ε**baSb**: After that string has been derived as $abaSb$, the only way to expand the string at this point is to apply the $S \rightarrow SaSb$ rule instead of the $S \rightarrow \varepsilon$ rule to the only S variable in the string. So we **must** apply that rule.

6. ε**a**ε**baSaSbb:** Now apart from the S variables our string is as it should be. To then fully form the specified string, both the S-variables in our derivation $abaSaSb$ **must** apply the rule $S \rightarrow \varepsilon$ so that the string does not expand. So we end up with $\varepsilon a \varepsilon b a \varepsilon a \varepsilon b b := abaabb$

Above constraints on the derivation sequence of the string prove that the string **abaabb** has an unique derivation sequence and that no other sequence can be used to generate the same string. That also means that the derivation tree is unique for $abaabb$. which is why no other derivation tree can be generated $\square$

b. Towards contradiction we assume that the string **abbaab** can be derived from the context-free grammar. We therefore attempt to derive the string using the context-free grammar's rules:

1. **S:** Since our string is non-empty, we **must** begin by applying the rule $S \rightarrow SaSb$.
2. **SaSb**: We observe now that our desired string has the prefix "ab". Just as in part (a.) of this problem, we further observe that in order to have a prefix of single a precede a b, the leading S variable in our string must generate a sub-string with the prefix ab. Contrarily, if the first S were to become ε, the string could never have this "ab" prefix as the second S would always create the irreversible prefix "aa". Therefore the the rule $S \rightarrow SaSb$ **must** be applied to the first S variable of our current derived string.
3. **SaSbaSb**: Now to have our newly derived have the prefix "ab" we have two options; applying the rule $S \rightarrow \varepsilon$ to the first S, or to continue the derivation of the first S by applying the rule $S \rightarrow SaSb$ to it. However, if we were to apply the rule $S \rightarrow SaSb$ to the first S, we could never get our desired suffix "$baab$" from the remaining symbols $aSbaSb$ as the suffix must then be on the form $a^*b^*a^*b$ which can never equal to $baab$. Therefore the rule $S \rightarrow \varepsilon$ **must** be applied to the first S.
4. ε**aSbaSb**: Further, the rule $S \rightarrow \varepsilon$ **must** also apply to the second S to generate our desired prefix "ab" instead of the irreversible prefix "aa" which is inevitable if we don't apply the rule $S \rightarrow \varepsilon$ to that S.
5. ε**a**ε**baSb**: We observe now that what we have derived can never be our desired string $abbaab$ since our derivation has the irreversible prefix of "aba". **We arrive at a contradiction**.

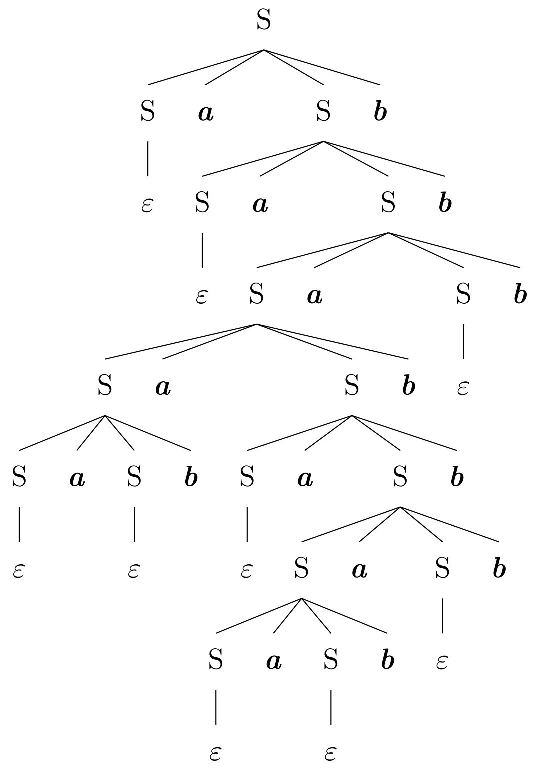
Alternatively, we can argue that a derivation tree cannot exist for **abbaab** via intuition: for every b that occurs in a string, there must be one preceding a to meet it since b is always generated by the rule $S \rightarrow SaSb$. We see already given the prefix "abb" that we have two b's and only one preceding a. This breaks that rule. Therefore the string cannot be in the language of the context free grammar.

This proves that the string $abbaab$ is not derivable from the context-free grammar. $\square$

c. Upon inspecting the grammar, we see that it generates the language $a^i(a^j(a^kb^k)^*b^j)^*b^i$ where $i, j, k \geq 0$. That is, the grammar generates outer encapsulation of symmetric number of a's and b's. Within that encapsulation, that is between the a's and the b's there are zero, one or more inner encapsulations of symmetric number of a's and b's, which contains zero, one or more inner encapsulation of symmetric number of a's and b's and so on and so forth.

We derive $SaSb$ from the start symbol, as this is the only way we'd derive a non-empty string. We see then that the only way to generate a brand new encapsulation of symmetric a's and b's is to use the former S symbol, as the latter will only result in expansion of the existing encapsulation, namely, some existing $a^c(a,b)^*b^c$ encapsulation becomes $a^{c+1}(a,b)^*b^{c+1}$.

So, this gives us that first S always generates a new encapsulation within recursively from that point in our string, while the second S determines expansion of existing encapsulation. Let's examine this behaviour by deriving the string *aaabaaababbbabbb* from the grammar, which we can be certain that is in the language of the grammar as it matches the string $a^i(a^{j_1}b^{j_1})(a^{j_2}(a^{k_1}b^{k_1})(a^{k_2}b^{k_2})b^{j_2})(a^{j_3}b^{j_3})b^i$ where $i = 2$, $j_1 = j_3 = 1$, $j_2 = 2$ and $k_1 = k_2 = 1$:



1. We see that on the second level of the derivation tree that when using the second S, the outermost ab encapsulation inevitably expands, so now our string has gone from the form $a^i(*)b^i$ to $a^{i+1}(*)b^{i+1}$ where $(*)$ denotes the possibility from where the string can expand.

2. We see that on the third level of the derivation tree that when using the first S, the we inevitably create a new string within our outermost encapsulation, so now our string has gone from the form $a^i(*)b^i$ to $a^i(*)a^j(*)b^jb^i$. This is consistent with our theory of expansion vs. generation, as new a^jb^j string can be generated from the first $(*)$ to fulfill the Kleene star property of the a^jb^j clause in the grammar's language $L = a^i(a^j(a^kb^k)^*b^j)^*b^i$, yet the second $(*)$ fulfills the property of increasing the value of j, so that a^jb^j can expand to $a^{j+1}b^{j+1}$.

3. We see that on the fourth level of the derivation tree that when using the first S, the we inevitably create a new string within our innermost encapsulation, so now our string has gone from the form $a^i(*)a^j(*)b^jb^i$ to $a^i(*)a^{j_1}(*)b^{j_1}a^{j_2}(*)b^{j_2}b^i$. This further supports our theory of expansion vs. generation, as a new inner encapsulation, the j-encapsulation, can only be generated from the left of all other j-encapsulations, while whithin they can only expand.

4. Again on the fourth level of the derivation tree we see that that when using the second S, the we inevitably expand our old j-encapsulation, so now our string has gone from the from $a^i(*)a^{j_1}(*)b^{j_1}a^{j_2}(*)b^{j_2}b^i$ to $a^i(*)a^{j_1}(*)b^{j_1}a^{j_2+1}(*)b^{j_2+1}b^i$.

This derivation goes on, where each derivation is consistent with our theory. This tells us that generation of any new encapsulation in the grammar is derived **uniquely** and always to the left of another existing encapsulation and the expansion of any existing encapsulation in the grammar is also derived **uniquely** for that encapsulation. Further, this tells us that however we derive any string in the language generated by the grammar, namely the language $L = a^i(a^j(a^kb^k)^*b^j)^*b^i$ no matter how nested it is of ab encapsulations and how expanded those encapsulations are, we will **always** uniquely derive it as we only ever have one choice of generation within the encapsulations and only one choice of expansion for each encapsulation, however we form a string.

Therefore, **no string in the language of the context-free grammar can exist with multiple deriviaton trees** as we have proven that each string in the language is uniquely derived from the grammar. This means that the grammar is **not ambiguous**. $\square$

(P.S. We are **so** sorry for this, whoever grades our assignment)

Problem 5. *Consider the program* $y := 1;$ ***while*** $\neg(x == 1)$ ***do*** $(y := y * x;\ x := x - 1)$ *in the example language of the book (Sec. 2.5). Write down the computation (maximal sequence of steps) of this program from the initial state σ = (x, 3) according to the structural operational semantics given in the book.*

Solution.

$\rightarrow \{c', \sigma\ [\mathrm{y} \leftarrow 1]\ \}$
$\rightarrow \{if\neg(\mathrm{x} == 1)$, then y:=(y × x); x:=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 1, \mathrm{x} \leftarrow 3\ ]\ \}$
$\rightarrow \{if\ \neg(3 == 1)$, then y:=(y × x); x :=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 1, \mathrm{x} \leftarrow 3\ ]\ \}$
$\rightarrow \{if\ \neg$ff then y =(y × x); x :=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 1, \mathrm{x} \leftarrow 3\ ]\ \}$
$\rightarrow \{if\ tt\ then\ y := (y \times \mathrm{x})$; x:=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 1, \mathrm{x} \leftarrow 3\ ]\ \}$
$\rightarrow \{y := (y \times \mathrm{x})$; x :=(x - 1); c'; $\sigma[\mathrm{y} \leftarrow 1, \mathrm{x} \leftarrow 3\ ]\ \}$
$\rightarrow \{y := (1 \times 3)$; x :=(3 - 1); c'; $\sigma[\mathrm{y} \leftarrow 1, \mathrm{x} \leftarrow 3\ ]\ \}$
$\rightarrow \{y := 3; x := 2;\ c'; \sigma[\mathrm{y} \leftarrow 1, \mathrm{x} \leftarrow 3\ ]\ \}$
$\rightarrow \{c'; \sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{if\neg(\mathrm{x} == 1)$, then y:=(y × x); x:=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{if\ \neg(2 == 1)$, then y:=(y × x); x :=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{if\ \neg$ff then y =(y × x); x :=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{if\ tt\ then\ y := (y \times \mathrm{x})$; x:=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{y := (y \times \mathrm{x})$; x :=(x - 1); c'; $\sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{y := (3 \times 2)$; x :=(2 - 1); c'; $\sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{y := 6; x := 1;\ c'; \sigma[\mathrm{y} \leftarrow 3, \mathrm{x} \leftarrow 2\ ]\ \}$
$\rightarrow \{c'; \sigma[\mathrm{y} \leftarrow 6, \mathrm{x} \leftarrow 1\ ]\ \}$
$\rightarrow \{if\neg(\mathrm{x} == 1)$, then y:=(y × x); x:=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 6, \mathrm{x} \leftarrow 1\ ]\ \}$
$\rightarrow \{if\ \neg(1 == 1)$, then y:=(y × x); x :=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 6, \mathrm{x} \leftarrow 1\ ]\ \}$
$\rightarrow \{if\ \neg$ tt then y =(y × x); x :=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 6, \mathrm{x} \leftarrow 1\ ]\ \}$
$\rightarrow \{if\ ff\ then\ y := (y \times \mathrm{x})$; x:=(x - 1); c' else skip $\sigma[\mathrm{y} \leftarrow 6, \mathrm{x} \leftarrow 1\ ]\ \}$
$\rightarrow \{skip, \sigma[\mathrm{y} \leftarrow 6, \mathrm{x} \leftarrow 1\ ]\ \}$
$\rightarrow \{\sigma[\mathrm{y} \leftarrow 6, \mathrm{x} \leftarrow 1\ ]\ \}$