Department of Information and Computing Sciences
Utrecht University

INFOB3TC – Solutions for final exam 2021

Jurriaan Hage

Thursday 4th of February 2021, 15.15-18.15

Please keep in mind that there are often many possible solutions and that these example solutions may contain mistakes.

1. Given is the following left-recursive grammar.

$S \to aSb$
$S \to Sba$
$S \to A$
$A \to Aa$
$A \to bS$
$A \to c$

Construct the grammar in which left-recursion is removed. •

. . . /**10**

Solution 1. There are two directe cases of left recursion:
For A we get

$A \to bS \mid bSZ \mid c \mid cZ$
$Z \to a \mid aZ$

For S we get

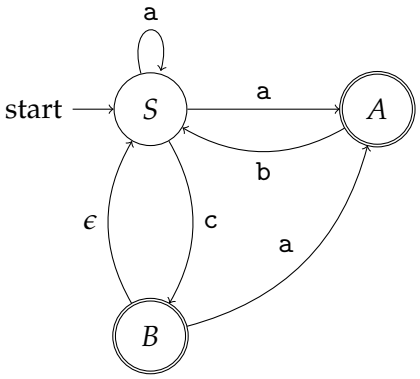
$S \to A \mid AY \mid aSb \mid aSbY$
$Y \to ba \mid baY$

Something like this is also allowed for S (and for A)

$S \to AY \mid aSbY$
$Y \to baY \mid \varepsilon$

Turning direct left recursion into to indirect is not allowed (0 points)
Some students used regular right hand sides. I permitted this, since it does show the right understanding.
In most cases (by far) where students did not use the algorithm to remove it, mistakes were made. Particularly, the order in accepted strings was changed. ◦

2. Consider the following NFA, with start state S, terminals $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, and final state A and B.



Construct a DFA (Deterministic Finite Automaton) that accepts the same language. You should use the subset construction, but should exclude states that are unreachable from the start state or that can never lead to a final state. It is *essential* that in your answer it is clear what each of your DFA states corresponds to in the original NFA; do not forget to indicate the start and end states of the DFA. •

. . . /**10**

Solution 2. You get about three points for the states, 1 for the start, 2 for the final states and about 1 pt. per correct edge.
If the result is not deterministic: you lose at least 5 pts regardless of what you did right.
An often made mistake is the epsilon edge is kept. This is not allowed in an nfa. Having too many states (ones that are not needed) leads to 2 pts less. If the state names do not reflect the original states from the machine: 2 pts less.

○

An AVL tree is a classical data structure, designed in 1962 by Georgy Adelson-Velsky and Evgenii Landis. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. The datatype *AVL* can defined as follows (note the invariants given for the constructors in comments).

```
data AVL e = E                          — Empty Tree
           | N (AVL e) e (AVL e)        — left height = right height + 1
           | Z (AVL e) e (AVL e)        — right height = left height
           | P (AVL e) e (AVL e)        — right height = left height + 1
```

3. Give the algebra type for the datatype *AVL e*. •

. . . /**4**

Solution 3.

$$\textbf{type}\ AVLAlg\ e\ r = (r, r \to e \to r \to r, r \to e \to r \to r, r \to e \to r \to r)$$

○Aantal args 1: twee type vars: 1, eerste component: 1, de andere goed: 1.

4. Now, give the type and the definition of the fold function associated with the datatype *AVL e*. •

. . . /**4**

Solution 4.

```
foldAVL :: AVLAlg e r → AVL e → r
foldAVL (e,n,z,p) = fold where
  fold E         = e
  fold (N l m r) = n (fold l) (fold m) (fold r)
  fold (Z l m r) = z (fold l) (fold m) (fold r)
  fold (P l m r) = p (fold l) (fold m) (fold r)
```

1 pt for the type, 1 for the pattern match, 1 for the recursions and 1 for the correct use of e,n,z,p. ○

5. The height of an *AVL* tree is an essential concept in *AVL* trees; we note the empty AVL tree has height zero. Implement the function *heightAVL* **efficiently** as a *foldAVL* **making use of the invariants given for the constructors**? •

. . . /**4**

Solution 5.

```
> heightAVL = foldAVL (e,n,z,p) where
  e       = 0
  n l m r = 1 + l
  z l m r = 1 + r
  p l m r = 1 + r
```

Some people used max, but then you are not taking advantage of the invariant. You get about half points for the idea of the computation, half points for the calling fold in the right way (eg. if you write something recursive here, you lose those points). ○

As a reminder, the following formulation of the pumping lemma proof strategy is provided:
For *every* natural number n,
find a word xyz in L with $|y| \geqslant n$ (*you choose* the word),
such that for *every* splitting $y = uvw$ with $|v| > 0$,
there exists a number i (*you figure out* the number),
such that $xuv^iwz \notin L$ (you have to *prove* it).

6. Given is the following language $R = \{a^n b^{m+n} \mid n, m \geqslant 1\}$. Prove with the Pumping Lemma for Regular Languages that R is not *regular*. •

. . . /**12**

Solution 6. Let n be the number of the pumping lemma. There are various ways to do this: one is to start from $xyz = a^n b^{n+1}$, where y contains all a's. In that case, you have to pump up (i=2). Another possibility is that y contain n b's. Then you have to pump down (i=0). Another completely different way is to say $xyz = a^n b^{n+m}$ for *some* value m and then that y consists of all a's. In that case, i cannot be any fixed constant, since you cannot prove that you leave the language. However, you can choose $i = m + 1$ for example.
I work out only the second one below: Take the word $xyz = a^n b^{n+1}$ where $x = a^n$, $y = b^n$ and $z = b$. Note that $y \geqslant n$.
Then we break up y in all possible ways. This means $y = uvw$ such that $u = b^p, v = b^q, w = b^r$ where $p + q + r = n$ and we may assume that $q >$.
Now, take $i = 0$. Then we get the word $xuv^0wz = a^n b^p b^r b = a^n b^{n+1-q}$, and since $q > 0$, $n + 1 - q$ is not larger than n. In other words, this word is not in R. ◦

7. Given is the following regular expression:

$$ba(((ba)^* + aa)^* + a)^*b$$

Give a finite state automaton that generates exactly the language of sentences described by this regular expression.

•

. . . /**8**

Solution 7. There are huge numbers of answers possible here. Something many students did was realize that this formula would be rewritten into $ba(ba + aa + a)^*b$ and then even the aa is superfluous. Whether you simplify or not, it does not affect your grade as long as you do it right.
I subtracted 1 pt if the resulting machine was correct, but overly complicated. You are not allowed to use edges labelled with anything except epsilon or a single terminal. Start state and end state(s) should not be forgotten. ○

8. Suppose we have an nfa with a finite set of states Q and alphabet X. And suppose that while drawing its diagram we observe the following following properties:

- it has one start state,
- it has one end state, and
- for all states $q \in Q$ and labels $v \in X \cup \{\epsilon\}$, q has at most one outgoing edge labelled with v.

Is this nfa necessarily a dfa? Explain your answer. •

. . . /**6**

Solution 8. No, you can still have non-determinism due to ϵ-edges. ○

9. Construct the $LR(0)$ automaton for the following grammar:

$S' \to S\,\$$
$S \to aA$
$S \to Sba$
$A \to bS$
$A \to c$

•

. . . /**10**

Solution 9. ○

10. List all the conflicts in this $LR(0)$ automaton, and indicate which kind of conflict each is. •

. . . /**4**

Solution 10. The state with items

$A \to bS\,.$
$S \to S.ba$

is in shift-reduce conflict. This is the only one. ○

11. Is the use of 1-lookahead in the SLR(1) parser enough to resolve the conflicts in the $LR(0)$ automaton? Argue using the follow sets of the non-terminals why this is the case. •

. . . /**4**

Solution 11. The shift is for b, and b is in the follow set of A, so the lookahead does not help. ○

12. Construct a pushdown automaton for the language over the alphabet $\{a, b\}$

$$L = \{w \mid \text{nr of } a\text{'s in } w \text{ is one more or one less than the nr of } b\text{'s in } w\}$$

It may, but need not be deterministic. •

. . . /8

Solution 12. If you forgot to allow that a's and b's can be mixed: -4. If you did not count right: -4. 1 pt for doing the +1 or -1 correct. Quite a few students assumed the extra was the first character; that is too restrictive. Some students first wrote a CFG and then turned it into a PDA. That is fine, too.

○

13. Suppose we have a context-free grammar $G_1 = (T_1, N_1, R_1, S_1)$. We now define

$$G = (T_1, N_1 \cup \{S\}, R_1 \cup \{S \to S_1 S_1\}, S)$$

where $S \notin N_1$. Describe the language of G in terms of the language generated by G_1? •

. . . / **6**

Solution 13. If $L(G_1)$ is the language accepted by G_1, then $L(G) = \{w_1 w_2 \mid w_1, w_2 \in L(G_1)\}$, so all words that you get by concatenating any two words of $L(G_1)$. Note that w_1 and w_2 can be different words. ◦