KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 5

Due: Sunday March 28 at 23:59 CET.
Submission: Please submit your solutions via *Absalon* as PDF file. State your name and e-mail address close to the top of the first page. Solutions should be written in LaTeX or some other math-aware typesetting system with reasonable margins on all sides (at least 2.5 cm). Please try to be precise and to the point in your solutions and refrain from vague statements. *Write so that a fellow student of yours can read, understand, and verify your solutions.* In addition to what is stated below, the general rules in the course information always apply.
Collaboration: Discussions of ideas in groups of two to three people are allowed—and indeed, encouraged—but you should always write up your solutions completely on your own, from start to finish, and you should understand all aspects of them fully. It is not allowed to compose draft solutions together and then continue editing individually, or to share any text, formulas, or pseudo-code. Also, no such material may be downloaded from the internet and/or used verbatim. Submitted solutions will be checked for plagiarism.
Grading: A score of 150 points is guaranteed to be enough to pass this problem set.
Please note: *If you have failed to pass some previous problem set(s), then running up enough extra points on this problem set to compensate for the total number of missing points from previous problem sets is guaranteed to make you eligible for the exam, so please make sure to work extra hard on the problems below in this applies to you.*
Questions: Please do not hesitate to ask the instructor or TAs if any problem statement is unclear, but please make sure to send private messages sometimes specific enough questions could give away the solution to your fellow students, and we want all of you to benefit from working on and learning on the problems. *Please note that you will need to watch both lectures from 2020 uploaded on Absalon, or to digest the material covered in those lectures in some other way, in order to be able to solve all problems on this problem set, so start by doing this before asking too many questions.* Good luck!

1 (120 p) In this problem, we want to understand the languages generated by specified regular expressions and context-free grammars.

1a (30 p) Which of the words below belong to the language generated by the regular expression $(b(ab)^* a) \mid (b^* ab^* ab^*)$? Motivate briefly your answers.

1. *babababa*
2. *babbabbb*
3. *ba*
4. *babaabaaab*
5. *babbaabbbba*
6. *babba*

1b (90 p) Consider the following context-free grammars, where a, b, c, d are terminals, A, B, S are non-terminals, and S is the starting symbol.

Grammar 1:

$$S \to abS \tag{1a}$$
$$S \to B \tag{1b}$$
$$B \to aB \tag{1c}$$
$$B \to cB \tag{1d}$$
$$B \to dB \tag{1e}$$
$$B \to \tag{1f}$$

Grammar 2:

$$S \to AbS \tag{2a}$$
$$S \to \tag{2b}$$
$$A \to aAa \tag{2c}$$
$$A \to \tag{2d}$$

Grammar 3:

$$S \to aSa \tag{3a}$$
$$S \to bS \tag{3b}$$
$$S \to \tag{3c}$$

Which of these grammars generate regular languages? For each grammar, write a regular expression that generates the same language, or argue why the language generated by the grammar is not regular.

2 (120 p) In this problem, we want to write regular expressions and context-free grammars generating specified languages.

2a (30 p) Write a regular expression for the language consisting of all finite (possibly empty) bit strings (i.e., over the alphabet $\{0, 1\}$) that contain an even number of 0s. Examples of such strings are ε (the empty string), 00, 1010001, and 111, whereas 10 and 01010 do not qualify for membership.

2b (50 p) Write a regular expression for the language consisting of all finite, non-empty bit strings that contain no two consecutive 1s. Examples of strings in this language are 0000, 01010, and 1001, while 111, 00110, and ε do not qualify. For partial credit (if your regular expression is wrong or missing), argue why this language is clearly regular.

2c (40 p) Give a context-free grammar for the language $\left\{a^m b^n c^{m+n} \,\middle|\, m, n \in \mathbb{N}\right\}$. Examples of strings in this language are *aacc* and *abbccc*, whereas *abc* and *abccc* do not make the cut.

3 (120+ p) Consider the regular expression $b^*(ab^*(aab^*|\varepsilon)cb^*)^*$

3a (60 p) Translate this regular expression to a nondeterministic finite automaton using the method from Mogensen's notes that we learned in class. Make sure to explain which part of the regular expression corresponds to which part of the NFA.

3b (60 p) Translate the nondeterministic finite automaton to a deterministic finite automaton using the method from Mogensen's notes that we learned in class. Make sure to explain how you perform the subset construction, so that it is possible to follow your line of reasoning.

3c *Bonus problem (worth 20 p extra):* How small a DFA can you produce that accepts precisely the language generated by the regular expression above? (Note that that you can solve this problem even if you did not solve the other subproblems above.)

4 (170 p) In this problem we want to construct parsers for context-free languages. We assume that (,), [,], +, *, and **num** are tokens return by a lexer (i.e., from the point of view of the parser these are terminals in the alphabet).

4a (70 p) Consider the following simplified version of the grammar for arithmetic expressions with precedence that we saw in class, where E is the starting symbol:

$$E \to E + E_2 \tag{4a}$$
$$E \to E_2 \tag{4b}$$
$$E_2 \to E_2 * E_3 \tag{4c}$$
$$E_2 \to E_3 \tag{4d}$$
$$E_3 \to \textbf{num} \tag{4e}$$
$$E_3 \to (\ E\ ) \tag{4f}$$

Is this an LL(1) grammar?

If your answer is yes, then construct *FIRST*, *FOLLOW*, and *Nullable*, and give pseudo-code for a recursive descent parser.

If your answer is no, then explain why the grammar fails to be LL(1). Is it possible to build a predictive parser for the language in some other way by using more characters of look-ahead?

4b (100 p) Consider a grammar for parenthesized lists of **num** tokens as follows, with S as the starting symbol:

$$S \to P\ S \tag{5a}$$
$$S \to B\ S \tag{5b}$$
$$S \to N \tag{5c}$$
$$P \to (\ S\ ) \tag{5d}$$
$$B \to [\ S\ ] \tag{5e}$$
$$N \to \textbf{num}\ N \tag{5f}$$
$$N \to \tag{5g}$$

Is this an LL(1) grammar?

If your answer is yes, then construct *FIRST*, *FOLLOW*, and *Nullable*, and give pseudo-code for a recursive descent parser.

If your answer is no, then explain why the grammar fails to be LL(1). Is it possible to build a predictive parser for the language in some other way by using more characters of look-ahead?