

# Assignment 1

CSE 655

Due Friday, Jan 16, by 9:30 am

1. (4 pts) Consider the following grammar for hexadecimal unsigned integers (i.e., with base 16), with starting non-terminal  $\langle hex\_literal \rangle$ :

$$\begin{aligned}\langle hex\_literal \rangle &::= 0x\langle number \rangle \\ \langle number \rangle &::= \langle number \rangle \langle number \rangle \mid \langle digit \rangle \\ \langle digit \rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid A \mid B \mid C \mid D \mid E \mid F\end{aligned}$$

This grammar describes the format of hexadecimal literals in the Java languages. String `0xF3` is an element of the language generated by this grammar. How many *distinct derivation sequences* exist for this string? How many *distinct parse trees* exist for this string? Show all parse trees.

2. (4 pts). Define a context-free grammar (and show the corresponding BNF) for generating all and only unsigned hexadecimal integers (e.g., `0xFD4B0DC`) in which each 7 appears immediately after `AB` and is followed immediately by `CD`. For example `0x5`, `0x5AB7CD`, `0x5D5AB7CDF6`, and `0xAB7CD` are all legal; but `0x57`, `0x7`, `0xF7D`, and `0x572` are all illegal. A valid number  $n$  is a non-empty sequence of hexadecimal digits, preceded by `0x`, such that every 7 in  $n$  is immediately followed by `CD` and immediately follows `AB`.
3. (4 pts). Consider the following grammar of identifiers:

$$\begin{aligned}\langle id \rangle &::= \langle letter \rangle \langle id \rangle \mid \langle letter \rangle\_ \langle id \rangle \mid \langle letter \rangle \\ \langle letter \rangle &::= A \mid B \mid \dots \mid Z\end{aligned}$$

This grammar allows `A`, `A_B`, `A_B_C` etc. as legal identifiers, but not `A_`, `A_B_`, `_B` etc. Change the grammar so that a series of an odd number of `_` is allowed in the middle of an identifier, although the identifier must not start or end with a `_`. Note that the total number of `_` can be odd *or* even, but every non-empty maximal-length subsequence of `_` must contain an odd number of `_`. (An identifier with a single letter is also legal.) Examples of *invalid* identifiers are `_A`, `_A_B`, `A__B`, `A____B`, `A___B__C`, and `A___B_`; on the other hand, `A_B_C` is valid.

4. (4 pts). Consider the following simple grammar of expressions:

$$\begin{aligned}\langle exp \rangle &::= \langle id \rangle \mid \langle hex\_literal \rangle \mid \langle exp \rangle + \langle exp \rangle \mid \langle exp \rangle * \langle exp \rangle \\ \langle id \rangle &::= A \mid B \mid \dots \mid Z\end{aligned}$$

Rewrite this grammar such that the operations will be performed strictly right to left; i.e., `0x3FA + Y * Z` will be parsed as “multiply Y and Z, and then add the result to `0x3FA`”, and `X * 0x0A1B2C3D4E + Z` will be parsed as “add `0x0A1B2C3D4E` and Z, and then multiply the result with X”. In your grammar, addition and multiplication should have the same precedence. Do not change the generated language: the language corresponding to your new grammar should be exactly the same as the language corresponding to the above grammar. Your new grammar should be non-ambiguous.

5. (4 pts). Consider the grammar you came up with for the previous question. Suppose you wanted to add a *pre-increment* operator `++` to the grammar to represent expressions of the form `++A`, similarly to languages such as C, C++, Java, and C#. The pre-increment operator

applies *only to identifiers*, and has higher precedence than + and \*. Show the corresponding generalized version of the grammar. Note that ++ is represented by its own terminal symbol, not by two terminal symbols for the + operator. Show a parse tree for the expression  $0x3FA + ++ A + ++ B * ++ C$ . Your new grammar should be non-ambiguous.

- Assignments are to be done independently. General high-level discussion of assignments with others in the class is allowed, but the actual work should be your own. Assignments that show excessive similarities will be taken as evidence of cheating and dealt with accordingly.
- Assignments should be turned in **by the beginning of class** on the due day. Late assignments turned in by the beginning of the next class will be graded with 30% reduction. Assignments turned in later than that will not be accepted.
- Make the assignments readable and understandable. They are to be handed in on regular paper, legibly written or typed. If you have more than one sheet, **staple the sheets together**. If the grader has trouble reading or understanding what you have done, points will be deducted even if it can finally be determined that you have the correct answer.
- Your solutions have to be **precise and detailed**: you have to work out **all** details that are necessary to solve the problem using the approaches discussed in class. You also have to write your solutions in a way that convinces the grader that you understand all these details. Be careful, precise, and thorough.