CSE305, Spring 2023 Assignment 1 Due Thu. Feb. 23, 11:59pm

Reading:

This week finished the long section 3.3 of Chapter 3, expanding several of its examples to some of the syntax of OCaml. For next week, *skim* section 3.4 on attribute grammars and "super-skim" section 3.5 on formal semantics. In section 3.4, the goal of focusing on the assignment-and-expressions example (numbered Example 3.6 in Sebesta 12th ed.) is not to *write* this kind of attribute grammar but rather to understand all the places in a big expression where OCaml does type inference and also type-checking. And in section 3.5, the nuggets to pick out are:

- how the C for-loop in section 3.5.1 gets translated into lower-level code with `goto`;
- the "two simple examples" in section 3.5.2 including how binary strings are inductively quantified as numbers;
- the analogously inductive and tree-based way to define the evaluation of expressions that comes next; and
- just the fact that *axiomatic semantics* uses proofs in the deduction style with horizontal bars the way you may have seen in CSE191 or a similar course.

(Incidentally, the expressions example in section 3.5.2 is written in the `case-match` syntax of *Standard* ML—it's kind-of weird for Sebesta to reference it so early there. But it times OK with next week's coverage of the corresponding OCaml `match-with` syntax.)

Then also skim-read Sebesta chapter 4 with this attitude: neither we nor Sebesta intends to do a full-scale compilers course, so what's there is not for its own analysis but to inform other things we care about. The section 4.2 on lexing has a hideous long example of writing a tokenizer in C, but we'll just get an idea of what kind of scanning it does and focus that idea on trying to understand what liberties OCaml does and does-not allow with spaces around dots, for instance. The first parsing section 4.3 doesn't say much (IMPHO) but the recursive-descent section 4.4 brings the "ETF" grammar idea to new life—with the variation that the EBNF form does not specify left-or-right association. This is first of all like how toward the end of the Thursday 2/16 lecture I said that OCaml's use of EBNF for tuples communicated that associativity is switched off. This section can be viewed as saying what to do if you want to switch it back on again. Section 4.5 too can be appreciated as just having a lot of nice examples giving further riffs on the expression grammar examples. [I said to skim all this in prior years too; indeed, in 2007 I wrote on the analogous assignment that sections 3.4–3.6 are "too technical too soon and distract from the natural flow of the course." Chapter 5 will be where we resume following Sebesta in detail and building on it like was done with section 3.3.]

Last, read about OCaml pattern-matching. This is the first main section under "Language" in the sidebar of the front page https://ocaml.org/docs, and the direct link is https://ocaml.org/docs/data-types The page is terse but its pace is rather fast. So I can recommend another source (suggested by a student in this class!) to complement it: *Programming Well: Abstraction and Design in Computation* by Stuart Schieber, available freely online at https://book.cs51.io/. You wouldn't know it's an OCaml manual from the cover except that it has a camel. Except that it often divebombs into long examples it is not as deep or intense—for instance, its own syntax chapter 3 doesn't get as analytical or comparative as Sebesta's section 3.3 or my lectures, but it gets you more directly into OCaml expressions and how they are typed. Schieber's book is actually even more "literate/liberal-artsy" than Sebesta or my lectures, and brings up the philosophy of "programmer intention" quite often. A read through chapters 2–6 is a good way to do a second pass on some of the things recitations and lectures have already shown about OCaml. But for next week, *please read its chapter 7 on "Structured and composite types" and also section 11.1* (stopping short of the even-longer examples in the rest of that chapter).

—-**Assignment 1**, due Thu. 2/23 with separate parts on *TopHat* and *CSE Autograder*—-

(1) The *TopHat portion*: 13 multiple-choice questions worth 1 or 2 points each, totaling **20 pts.** They are organized into one assigned document, and answers are automatically recorded and revealed as you work through it. Scores from it will eventually be ported to *Autograder* and later everything to *UBLearns*, which is being used only as a gradebook.

The rest is to be submitted as **a single PDF file** via *CSE Autograder*. You are welcome to write answers in handwriting and take snapshots, using some utility to sequence them into one PDF file, or just pasting them into MS Word and exporting a PDF file.

(2) Using the "ETF" grammar from lecture, or the cut-down version of it in Sebesta's Example 3.4 (ignoring the assignment rule and making lowercase identifiers `a | b | c` in place of his capitals), give both parse trees and leftmost derivations for the following expressions:

(i) `a + b + c`

(ii) `a*(b + c)`

(iii) `a * (b * (c + a))`

For (iii) you may shortcut things by drawing the parse tree and then saying how many steps the corresponding leftmost derivation would have. Or you may abbreviate sequences of unit rules in that derivation to save a lot of steps with lengthy re-copying. Finally also:

(iv) Give a parse tree and leftmost derivation of `a * (b * (c + a))` in the original simple-but-ambiguous expression grammar with only the `EXP` nonterminal. Compare the size of the tree and number of steps in the derivation with your answer to (iii).

(6+6+9+9 = 30 pts. total)

(3) In C, C++, and Java, the modulus operator `%` has equal precedence with `*` and `/` and is left associative, while the shift operator `>>` has lower precedence, i.e. "binds looser than," addition and subtraction. Extend the "ETF" grammar to create an unambiguous grammar for expressions with these two additional binary operators (besides `+,-,*,/` that is). Give a leftmost derivation of `r%2*n>>d-1` and explain how it is parsed. (12 + 6 + 4 = 22 pts., for 72 points on the set)