CS558 Programming Languages – Fall 2023 – Study Questions Lecture 1b

These questions are intended for self-study, to help review and deepen your understanding of the lecture. Sample answers are available. There is nothing to hand in.

1. Consider the grammar that has the set of terminals {`k`, `i`, `t`, `e`, `y`, `n`}, the two nonterminals `S` (the start symbol) and `B`, and the following production rules:

```
S  →  k
S  →  i B t S
S  →  i B t S e S
B  →  y
B  →  n
```

(a) Draw a parse tree for the sentence

```
i n t i y t k
```

(Hint: There is only one such parse tree.)

(b) Write two *different* linear derivation sequences corresponding to the parse tree you gave in (a).

(c) Draw *two* different parse trees for the sentence

```
i y t i n t k e k
```

(d) Explain precisely why this grammar is ambiguous.

2. Consider the expression grammar on slide 19.

(a) Draw the parse tree for this grammar applied to the sentence `a-b*c-d`. Observe how the grammar forces you to give `*` higher precedence than − and to treat − as left-associative. Try to get some intuition for how this works.

(b) How would we change the grammar if we wanted to make + and − *right*-associative instead?

3. (a) Following the pattern suggested on slides 23 and 24, give a tree grammar for abstract syntax trees that can express all the features of the expression grammar on slide 19.

(b) Using your grammar, draw an AST for the expression `a*(b+c)-d`. This AST should have many fewer nodes than a parse tree for the expression. Which nodes have disappeared, and why?

4. To read and write the abstract syntax of different toy languages in the labs, we will use s-expressions, as described in slides 29-31. Like any other language, we can specify both a concrete and abstract syntax for s-expressions. Here is the concrete syntax for s-expressions used in the lab assignments:

```
sexpr ::= '('  slist ')' | sym | num | str
slist ::= sexpr | sexpr slist
```

where the terminals (each carrying the obvious attribute) are as follows

- `num` is an (possibly signed) decimal integer literal

- `str` is a string literal delimited by double quotes (`"`)
- `sym` is a string of one or more printing characters excluding `(`,`)` , and `"` that is not a `number`

In addition, comments enclosed in matching curly braces ({}) are permitted anywhere; this is not shown in the grammar.

Convince yourself that this grammar is unambiguous. It is also very simple—yet it still has "syntactic noise" that we can get rid of in an abstract syntax, such the following:

$SList : SExpr \rightarrow SExpr_1 \ldots SExpr_n$
$SSym : SExpr \rightarrow (string)$
$SNum : SEepr \rightarrow (int)$
$SString : SExpr \rightarrow (string)$

Draw a parse tree and an AST based on these grammars for a simple example at the bottom of slide 30.