CMPU 331 · Compilers · Fall 2019

Assignment 2

Due Thursday, October 3, 2019

This assignment covers top-down parsing and bottom-up parsing. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work, and you should indicate in your submission who you worked with, if applicable. Assignments should be submitted through Moodle as a PDF by 11:59pm. (You can export a PDF from Google Docs, Word, LibreOffice, LaTeX, or any other text editor you prefer.)

Problem 1

Consider the following BNF grammar for a simple Scheme-like language:

```
<expression> ::= <constant> | <identifier>
    | (<operator> <expressions>)
    | (let <identifier> <expression>)
<expressions> ::= <expression> | <expression> <expressions>
<constant> ::= <boolean> | <number>
<boolean> ::= #t | #f
<operator> ::= + | - | * | /
<number> ::= <number> <digit> | <digit>
<identifier> ::= <letter> | <letter> <identifier>
<letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

(a) Left-factoring modifies a grammar, to factor out common prefixes of productions. What are the common prefixes in the `<expression>` production? How could you rewrite `<expression>` to factor out the common prefixes?

(b) A left-recursive grammar has a non-terminal as the first element of its own production. How could you rewrite the `<number>` production to eliminate left recursion?

(c) Draw a parse tree for the input string:

```
(+ 9 (* 5 6))
```

(d) If you were writing an LL(1) parsing table, what would the parsing table entry be for:

```
[<expression>, 1]
```

That is, which alternate of the `<expression>` production should be chosen when the next input token is ”1”?

(e) Given the following *state* of an LR(0) shift-reduce parser (where `|` represents the top of the stack):

```
(let | foo #f)
```

If the DFA to recognize viable prefixes for the parser yields the *item*:

```
expression -> (let . identifier expression)
```

Should the next action of the parser be to *shift* or *reduce*, and why?