CMSI 488 Homework 1

Don Rowe

Due February 21, 2012

1. (a) `/[A–CEGHJ–NPR–TVXY]\d[A–CEGHJ–NPR–TV–Z] \d[A–CEGHJ–NPR–TV–Z]\d/`

   Canadian postal are in the format A0A 0A0, where "A" is any English capital except for D, F, I, O, Q, or U and "0" is any digit. Additionally, the first letter cannot be W or Z. Also, notice the space between the groups of three.

   (b) `/4\d{12}\d{3}?/`

   Visa card numbers begin with a 4 and are followed by either 12 or 15 more digits. The above regular expression assumes that there are always at least the first 13 digits with an optional additional 3 digits.

   (c) `/5[1−5]\d{14}/`

   MasterCard numbers begin with a 5, and the second digit is between 1 and 5. The remaining 14 digits can be any digit.

   (d) `/\d+(_\d+)*#[\dA–F](_[\dA–F]+)*(\.[\dA–F]+(_[\dA–F]+)*)?#(E[\+\−]?\d+(_\d)*)?|\d+(_\d+)*(\.\d(_\d+)*)?(E[\+\−]?\d+(_\d)*)?`

   Numeric literals in Ada95 come in two flavors: based literals and decimal literals.

   A based literal is separated into three sections delimited by # signs. The first section is any number of digits, possibly separated at arbitrary intervals after the first digit by an underscore character; this sequence, represented by `\d+(_\d+)*` in the regular expression, is simply called a numeral. The second section is a based numeral possibly having a fractional part after a dot; based numerals are like numerals, only they are hexadecimal and are represented by `[\dA−F](_[\dA−F]+)*` in the regular expression. The third section is an optional exponent notated with a capital E, an optional plus or minus sign, and a numeral; an exponent is represented in the regular expression by `(E[\+\−]?\d+(_\d)*)`.

   A decimal literal is comprised of a numeral followed by an optional fractional part and an optional exponent.

   (e) `/^(c|b(?!ab)|a(?!a))*$/`

   This language includes strings made up of any combination of the characters "a", "b", and "c", provided "a" is never followed immediately by another "a" (represented in the regular expression by `a(?!a)`) and provided "b" is never followed immediately by "ab" (represented by `b(?!ab)`). To ensure that the pattern is applied to the whole string, the beginning ând ending $ are included in the regex; otherwise, legal substrings of an illegal string can be matched. For example, the string accccbabaacccbcbba does not match as a whole, but the substrings acccc, ab, and accccbcbba (split by the first b and third a) are matched.

2.
```
var program =
{
    'declare': ['x', 'y'],
    'while':
    {
        'condition':
```

```
{
    'minus':
    {
        'left': 'y',
        'right': '5'
    }
},
'body':
{
    'declare': 'y',
    'read': ['x', 'y'],
    'assign':
    {
        'left': 'x',
        'right':
        {
            'times':
            {
                'left': '2',
                'right':
                {
                    'plus':
                    {
                        'left': '3',
                        'right': 'y'
                    }
                }
            }
        }
    }
},
'write': '5';
}
```

3. (a) This grammar produces strings made of the characters "a" and "b". The shortest string in this language is "a": if a string begins with a "b", it must be followed by at least "aa". If a string in this language ends with a "b", it must be immediately preceded by an "a".

S

A     M

'a'   E     S

A     M

'b'   A     A

'a'   E   'a'   E

3
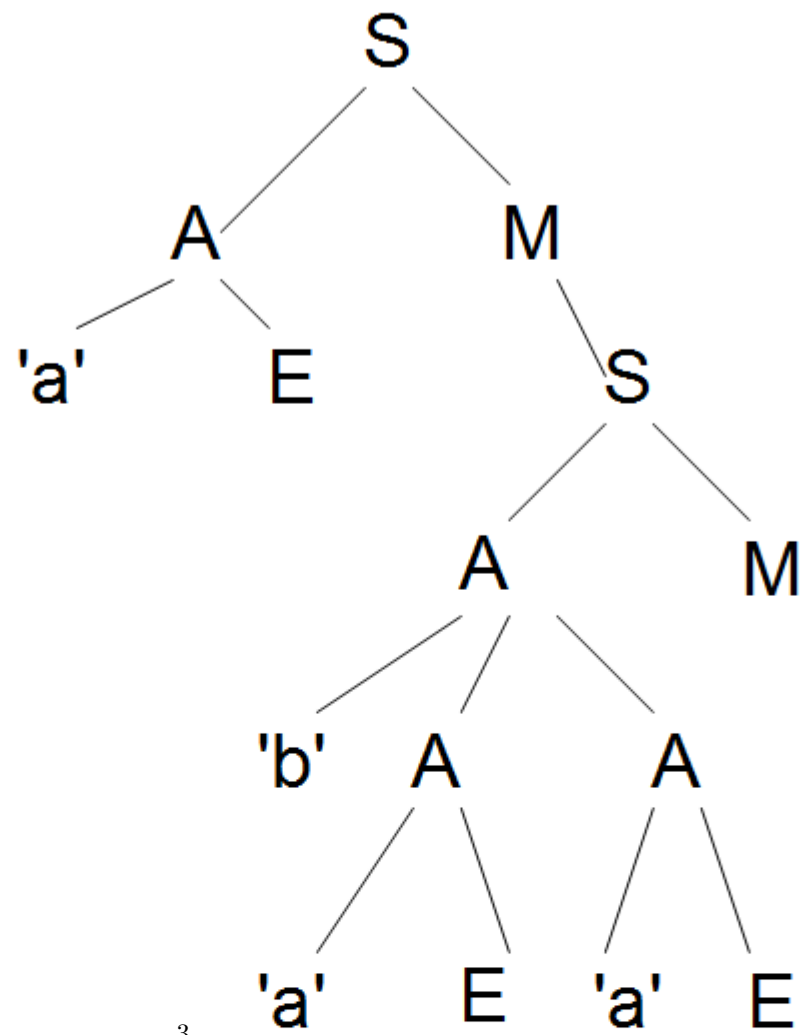
(c) This grammar is not LL(1). If one looks ahead one token and sees an 'a', that could have been from rule A, E, or B?

(d) Grammars are ambiguous if there exists a string for which there is more than one way to parse it. Below is another parse tree for "abaa". Therefore, this grammar is ambiguous.
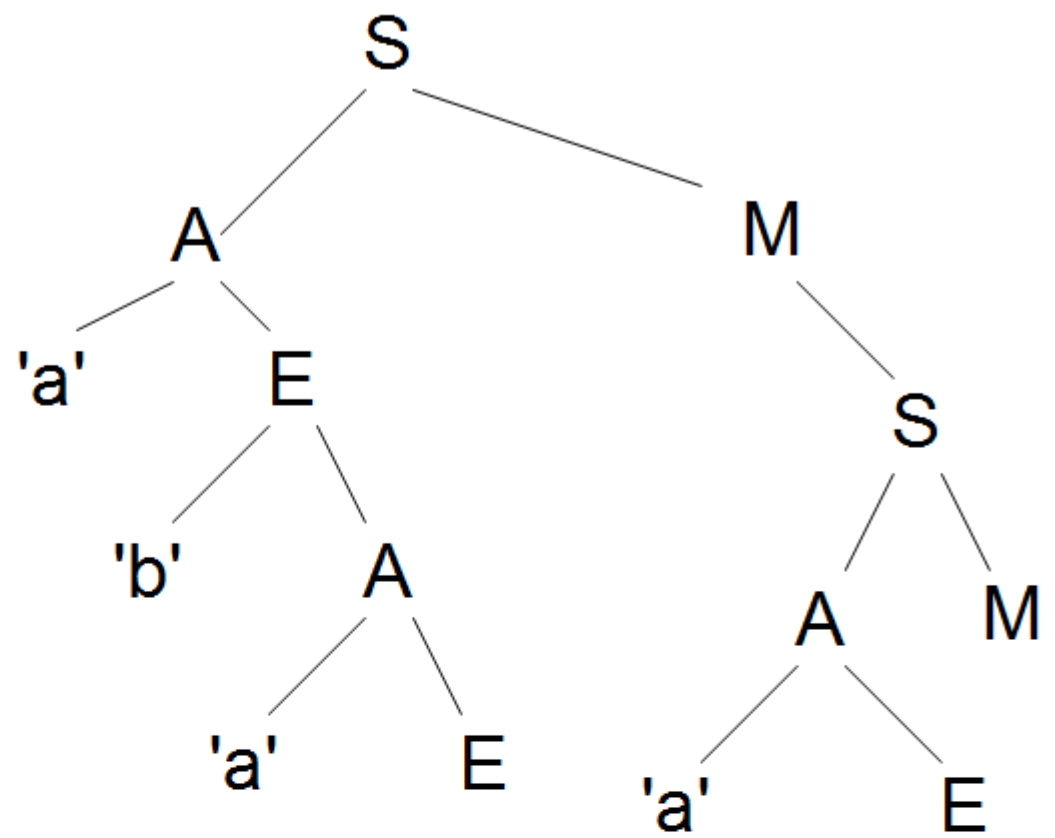
4. (a) At the rule EXP lies a choice point with two choices: *ID ":=" EXP* and *TERM TERM_TAIL* .

5. The unary negation operator was placed with the ADDOPs for symmetry– indeed, unary negation in the form of $-x$ is an implication of $0 - x$. Additionally, since $y - -x$ (that is, subtracting $-x$ from $y$) cannot occur in Ada, it is consistent to throw out any case of an operator immediately preceding a unary negation operator.

The expression $-8 * 5$ can be thus represented with this abstract syntax tree:

```
        -
        |
        *
       / \
      8   5
```
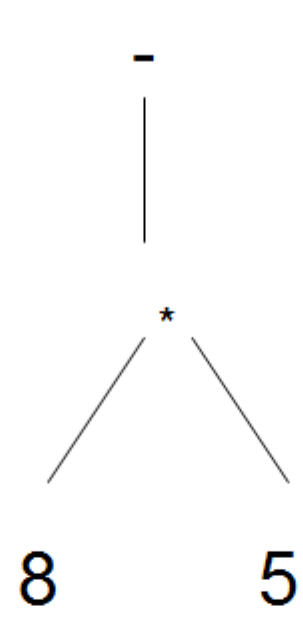
This parsing is similar to parsing with the negation operator in EXP4 in that the answer is still -40 since the negation operator is commutative. The parsing is different, though, in that the negation is applied to the whole product of $8 * 5$ as opposed to applying the negation first to 8.

6.