Homework #7: CMPT-413

Distributed on Mon, Mar 15; Due on Mon, Mar 22

Anoop Sarkar – anoop@cs.sfu.ca

(1) **Earley Recognition for CFGs** (Submit files: `earley.pl`, `trace-1.txt` and `trace-2.txt`)
Consider the CFG with S as the start symbol:

$$\begin{aligned} S &\rightarrow SBJ\ VP \\ VP &\rightarrow V\ NP \\ VP &\rightarrow VP\ PP \\ NP &\rightarrow NP\ PP \\ NP &\rightarrow Det\ N \\ PP &\rightarrow P\ NP \end{aligned} \qquad \begin{aligned} SBJ &\rightarrow Calvin \\ V &\rightarrow saw \\ Det &\rightarrow the \\ N &\rightarrow man \mid telescope \mid hill \\ P &\rightarrow with \mid on \end{aligned}$$

Check that the input: *Calvin saw the man with the telescope on the hill* for the above CFG results in 5 parse trees, each representing a valid meaning for the input sentence. The ambiguity in this grammar is called *PP-attachment ambiguity*. The above CFG can be represented as Perl data in the following manner:

```
@start = ('S');
$cfgrule{'S'}   = [ ['SBJ', 'VP', '#' ] ];
$cfgrule{'VP'}  = [ ['V', 'NP', '#' ], ['VP', 'PP', '#' ] ];
$cfgrule{'NP'}  = [ ['NP', 'PP', '#' ], ['Det', 'N', '#' ] ];
$cfgrule{'PP'}  = [ ['P', 'NP', '#' ] ];
$cfgpos{'SBJ'}  = [ ['Calvin', '#' ] ];
$cfgpos{'V'}    = [ ['saw', '#' ] ];
$cfgpos{'Det'}  = [ ['the', '#' ] ];
$cfgpos{'N'}    = [ ['man', '#' ], ['telescope', '#'], ['hill', '#'] ];
$cfgpos{'P'}    = [ ['with', '#'], ['on', '#'] ];
```

The `%cfgrule` hash table contains the non-lexical rules (i.e. rules that do not contain any terminal symbols) while the `%cfgpos` hash table contains rules of the form $A \rightarrow a$ where A is a non-terminal and a is a terminal symbol signifying the part of speech tag rules in the grammar.

Using the above Perl representation of a CFG, implement the Earley recognition algorithm. The Earley algorithm is described in Figure 10.16 on page 381 in the Jurafsky and Martin textbook. Note that you *do not* have to implement the retrieval of parse trees from the chart. **Provide the Perl code for your implementation of the Earley recognition algorithm in the file `earley.pl`. Also provide a trace of your recognition algorithm for 3 sentences accepted by the grammar `trace-1.txt`, and 3 sentences that are not accepted `trace-2.txt`.** Your trace must have the same format as those supplied in the files `ex-trace-1.txt` and `ex-trace-2.txt`. The example grammar from the textbook is provided as the file `atisgram.pl` and the trace for the input *book that flight* is given in `ex-trace-3.txt`.

Hint: You can represent the *state* data structure described in the algorithm as follows: A state: $(A \rightarrow B \bullet C, [5, 7])$ can be represented in Perl as

```
$state = [ 'A', [ 'B', 'C', '#'], 1, 5, 7 ];
```

where `$state->[0]` is the left hand side of the CFG rule, `$state->[1]` is a reference to a list of right hand side symbols for the CFG rule, `$state->[2]` is the location of the dot in the right hand side, in this example, since the dot position is 1 the dot is immediately to the left of `'C'` in the right hand side of the rule, and finally `$state->[3]` and `$state->[4]` are the i, j values for the span of the dotted rule in the input string: $5, 7$ in this case. If you represent the state in this manner the following functions might be useful to you:

```
sub incomplete {
    my ($state) = @_;
    return (($state->[1]->[$state->[2]] eq '#') ? 0 : 1);
}

sub nextCat {
    my ($state) = @_;
    return ($state->[1]->[$state->[2]]);
}
```

To implement recognition in a straightforward manner, the implementation of `earlyRecognize` should return the chart after parsing of the input has finished. We can use the chart to decide whether the input string was accepted by the CFG or not. The following code is an example of how we can check the chart to see if a start symbol successfully spans the entire input string. Note that this code assumes that states are represented as described above. It also assumes that the chart is implemented as a reflist of reflists containing the states.

```
my @input = qw(Calvin saw the man with the telescope);
my $chart = earleyRecognize(@input);
my $length = $#input+1;
my $yes = 0;
for my $start (@start) {
    for my $finalState (@{$chart->[$length]}) {
        $yes = 1 if (recognizeSuccess($finalState, $start, 0, $length));
    }
}
print (($yes) ? "yes" : "no", "\n");

sub recognizeSuccess {
    my ($finalState, $start, $begin, $end) = @_;
    return (($finalState->[0] eq $start) and
            (! incomplete($finalState)) and
            ($finalState->[3] == $begin) and
            ($finalState->[4] == $end));
}
```

A sample final state for the input *Calvin saw the man with the telescope* which would succeed in this test (given the CFG above) is the state ($S \rightarrow SBJ\ VP\bullet, [0, 7]$) or in the Perl equivalent:
`['S', ['SBJ', 'VP', '#'], 2, 0, 7]`
History: The Earley algorithm was proposed by J. Earley in his 1968 CMU CS PhD thesis as a more elegant form of the Cocke-Younger-Kasami parsing algorithm which was the first polynomial time parsing algorithm for CFGs. The Cocke-Younger-Kasami algorithm was independently discovered by the three people mentioned in its name and is often referred to as the CKY or the CYK algorithm. Both the Earley algorithm and the CKY algorithm take worst case time: $O(n^3)$ where n is the length of the input string.