CS536: Homework 5

Keith Funkhouser

October 13th, 2015

The operands in our language regular expressions are single letters or ϵ (epsilon). The operators are:

- | means "or" (alternation)
- writing two or more things next to each other means "followed by" (catenation)
- * means "zero or more" (closure or iteration)
- + means "one or more" (positive closure)
- ( ) are used for grouping

In a regular expression, * and + have the same, highest precedence, "followed by" has middle precedence, and | has the lowest precedence. All of the operators are left associative.

Below are 6 incorrect CFGs for the language of regular expressions. For each CFG, do ***one*** of the following:

1. Give one string that is a legal regular expression (given our definition above), but is not in the language of the CFG.
2. Give one string that is not a legal regular expression (given our definition above), but is in the language of the CFG.
3. Show that the CFG is ambiguous by drawing two different parse trees for some string in the language of the CFG.

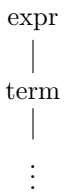
For cases (a) and (b), be sure to say which of the two cases you are illustrating.

Note that the terminals are `LTR`, `EPS`, `OR`, `STAR`, `PLUS`, `LPAR`, and `RPAR`. Note also that there is a difference between the terminal `EPS` (which represents the token epsilon in our language of regular expressions) and the symbol ϵ (which is used on the right-hand-side of a grammar production indicating the non-terminal on the left-hand-side derives an empty sequence of symbols).

1 CFG 1:

```
expr -> expr OR term | term
term -> term item | item
factor -> item STAR | item PLUS | item
item -> LTR | EPS | LPAR expr RPAR
```

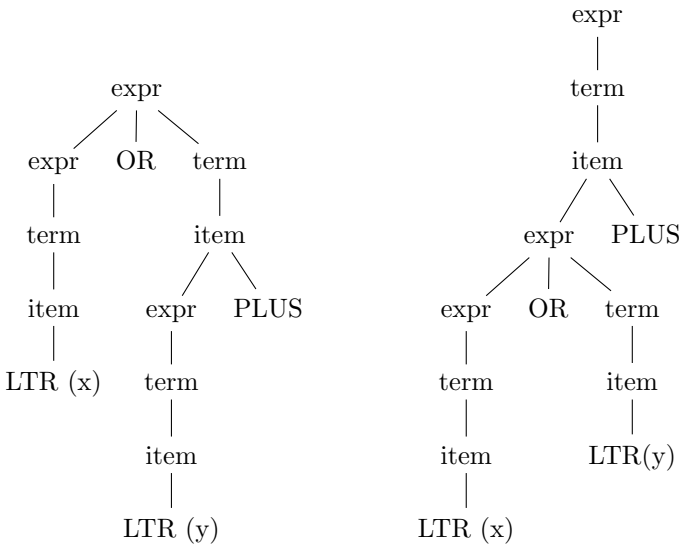
The expression `x+` is a legal regular expression, but is not in the language of the CFG. It cannot be derived from the starting nonterminal `expr`. The reason for this is that, once we get to this point, we cannot derive `LPAR expr RPAR` from `term` because `factor` does not show up on the right hand side of any of the productions:



2 CFG 2:

```
expr -> expr OR term | term
term -> term item | item
item -> expr STAR | expr PLUS | LTR | EPS | LPAR expr RPAR
```

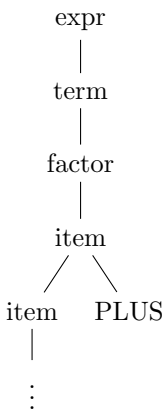
Consider the two parse trees below for the regular expression `x|y+`:



3 CFG 3:

```
expr -> LPAR expr RPAR | term
term -> term OR factor | factor
factor -> factor item | item
item -> item STAR | item PLUS | LTR | EPS
```

The expression `(x)+` is a legal regular expression, but is not in the language of the CFG. It cannot be derived from the starting nonterminal `expr`. The reason for this is that, once we get to this point, we cannot derive `LPAR expr RPAR` from `item`:



4 CFG 4:

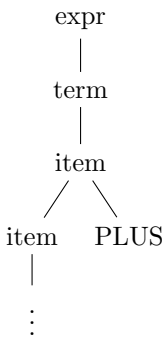
```
expr -> expr OR term | term
term -> term item | ϵ
item -> item STAR | item PLUS | LTR | EPS | LPAR expr RPAR
```

The empty sequence of symbols ϵ is not a legal regular expression, but is in the language of the CFG:

5 CFG 5:

```
expr -> expr OR term | term
term -> term item | LPAR expr RPAR | item
item -> item STAR | item PLUS | LTR | EPS
```

The expression `(x)+` is a legal regular expression, but is not in the language of the CFG. It cannot be derived from the starting nonterminal `expr`. The reason for this is that, once we get to this point, we cannot derive `LPAR expr RPAR` from `item`:



6 CFG 6:

```
expr -> LTR | EPS | term
```

```
term -> term OR factor | factor
factor -> factor item | item
item -> item STAR | item PLUS | LPAR item RPAR | expr
```

Consider the two parse trees below for the regular expression `x`: