Homework 2

September 18, 2018

For this homework, you will modify the following BNF grammar to resolve ambiguity that includes operator precedence and associativity.

```
<e> ::= <e> + <e>
   |  <e> - <e>
   |  <e> * <e>
   |  <e> / <e>
   |  <e> ^ <e>
   |  <e> !
   |  <e> %
   |  ( <e> )
   |  - <e>
   |  E
   |  PI
   |  <NUMBER>
```

Note that your grammars do not consider white spaces and `<NUMBER>` is a non-terminal that refers to numbers.

1 Requirement

1. Your new grammar should have the following precedence.

$$+, - \ \leq \ *, / \ \leq \ \hat{} \ \leq \ !, \% \ \leq \ -$$

2. Your new grammar should enforce left associativity for `+, -, *, /` and right associativity for `^`.

3. You should verify your solution using `jison`. To use `jison`, go to the website `https://github.com/zaach/jison` and follow the instruction to install the source code using `npm` and examples using `git`. Note that if you should also install `node.js`, `npm`, and `git`, which I assume you can easily figure out by yourself.

   I have provided a `jison` source file called `hwk2.jison` for you to modify. Before you do that, you can test it by running

```
jison hwk2.jison
node hwk2.js test
```

where `test` is a text file with the content:

```
1 + 2 + 3 - 4 * -5! - PI / E ^ 6 ^ 7 + 8%
```

The execution result is

```
(((((1 + 2) + 3) - (4 * ((-5)!))) - (PI / (E ^ (6 ^ 7)))) + (8%))
```

where the parenthesises correspond to the shape of the parse tree.

The relevant portion of the `jison` input is shown below, which you should modify:

```
e
    : e '+' e
        {$$ = '(' + $1 + ' + ' + $3 + ')';}
    | e '-' e
        {$$ = '(' + $1 + ' - ' + $3 + ')';}
    | e '*' e
        {$$ = '(' + $1 + ' * ' + $3 + ')';}
    | e '/' e
        {$$ = '(' + $1 + ' / ' + $3 + ')';}
    | e '^' e
        {$$ = '(' + $1 + ' ^ ' + $3 + ')';}
    | e '!'
        {$$ = '(' + $1 + '!' + ')';}
    | e '%'
        {$$ = '(' + $1 + '%' + ')';}
    | '-' e %prec UMINUS
        {$$ = '(-' + $2 + ')';}
    | '(' e ')'
        {$$ = '(' + $2 + ')';}
    | NUMBER
        {$$ = Number(yytext);}
    | E
        {$$ = 'E';}
    | PI
        {$$ = 'PI';}
    ;
```

You do not need to modify the lines enclosed by { and }, which are used to generate the output. Note that this `jison` file is not ambiguous because the following lines above the production rules, which establishes the precedence and associativity.

```
%left '+' '-'
%left '*' '/'
%right '^'
%right '!'
%right '%'
%left UMINUS
```

You should delete them from your `jison` file. After deleting these lines, if you run `jison hwk2.jison`, you will get errors that say the grammar is ambiguous for parsing. You should add non-terminals to resolve the ambiguity. In particular, you should add non-terminals called

(a) `RootExp` to represent any expression that does not include operators,

(b) `NegExp` to represent negation expression,

(c) `UnaryExp` to represent unary expressions ! and %,

(d) `PowExp` to represent exponentiation, and

(e) `MulExp` to represent multiplication and division.

For example, below is the grammar after adding `RootExp`.

```
e
    : e '+' e
        {$$ = '(' + $1 + ' + ' + $3 + ')';}
    | e '-' e
        {$$ = '(' + $1 + ' - ' + $3 + ')';}
    | e '*' e
        {$$ = '(' + $1 + ' * ' + $3 + ')';}
    | e '/' e
        {$$ = '(' + $1 + ' / ' + $3 + ')';}
    | e '^' e
        {$$ = '(' + $1 + ' ^ ' + $3 + ')';}
    | e '!'
        {$$ = '(' + $1 + '!' + ')';}
    | e '%'
        {$$ = '(' + $1 + '%' + ')';}
    | '-' e %prec UMINUS
        {$$ = '(-' + $2 + ')';}
    | '(' e ')'
        {$$ = '(' + $2 + ')';}
    | RootExp
        {$$ = $1;}
    ;

RootExp
    :  '(' e ')'
```

```
    {$$ = '(' + $2 + ')';}
| NUMBER
    {$$ = Number(yytext);}
| E
    {$$ = 'E';}
| PI
    {$$ = 'PI';}
;
```

Note that the line `{$$ = $1;}` in the lines

```
...

| RootExp
    {$$ = $1;}
...
```

is needed to collect the string representing `RootExp`. You should do the same for other new non-terminals. If your implementation is correct, you should get the same output as you do with the original `hwk2.jison`.

2 Submission

Please submit a file `hwk2.txt` that contains the BNF grammar in text and a file called `hwk2.jison` that reflects your solution in `jison` to the dropbox.