Introduction to Language Theory and Compilation: Exercises

Session 12: `yacc/bison` parser generator

ULB

Introduction

- `yacc` is a tool that automagically generates *parsers*.
- Its input is a *specification* that describes a grammar and also associates *actions* to be done at different points during parsing.
- Its output is source code that implements the given parser.
  - `lex` generated the scanner function
  - `yacc` generates the parser function

ULB

Properties of `yacc`-generated parsers

The parser generated by `yacc` has the following properties:

- It parses the grammar in a *bottom-up* fashion (i.e. shift/reduce).
- It comminicate with a lexical analyser. This function is usually the one generated by `lex`, but it's not mandatory.
- It calls the `yyerror()` function when something goes wrong and has to be written. A default implementation is available by linking with `liby`.
- Whenever a production rule is *reduced*, the parser executes the corresponding *action*.

ULB

CUP Java implementation of a LR-parser

- Communication between JFlex and CUP explained in the previous session
- Write the JFlex specification *tokens.flex*
- Write the CUP specification *grammar.cup* and additional Java files.
- Produce the Scanner Java file: *jflex tokens.flex*
- Produce the Parser Java file: *java -cp java-cup.jar:. java_cup.Main -parser Parser -symbols Symbols grammar.cup* (-parser and -symbols are optional)
- Compile all Java files: *javac -cp java-cup.jar:. *.java*
- Run your main method. If the main is added into your parser: *java -cp java-cup.jar:. Parser*

ULB

CUP specification

As with `JFlex`, `CUP` specifications are broken up into three parts mot separated delimiters.

1. Arbitrary source code (class, import, ...)
   - Have the following format:

     ```
     import java_cup.runtime.*;
     ```

CUP specification

As with `JFlex`, `CUP` specifications are broken up into three parts separated by no delimiters.

2. Declarations of symbols, precedence and additional code
   - Symbols:

     `non? terminal ObjectType? list of symbols separated by comma;`
   - Precedence:

     `precedence left list of symbols separated by comma;`
   - Additional code:

     `parser code {: code source :}`

     (can have different value than `parser`)

CUP specification

As with `JFlex`, CUP specifications are broken up into three parts separated by no delimiters.

3. Production rules
   - Have the following format:

```
variable ::= definition1:optional_var_name
                        {RESULT = optional_var_name;}
           | definition2 {action2}
           | ...
           ;
```

   - Actions are activated whenever the corresponding rule is reduced by the parser.
   - *Definitions* are made of *symbols* defined in the Declaration part

Exercises 1-3

1. Informally describe the accepted language of the compiler we'd generate from the specifications.
2. Adjust the specification so it only accepts polynomials of a single variable. We input a polynomial per line, but there can only be one variable used on each line.
3. Add the necessary code to show the first derivative of a polynomial. For example, if `2x^3+2x^2+5` was given on input, we would output :
   `First derivative:  6x^2+4xsetc`

Exercises 4-5

4. Add a way to recognize polynomial products and adjust the derivative calculation. For example, if `(3x^2+6x)*(9x+4)` is given on input, we would output:
   `First derivative:  (3x^2+6x)*(9)+(6x+6)*(9x+4)`
5. Add a way to evaluate a polynomial and its first derivative for a given value. The user should be able to input the variable value, followed by a semicolon, followed by the polynomial (all this on the same line). For example :

```
2 ; (3x^2+6x)*(9x+4)
First derivative : (3x^2+6x)*(9)+(6x+6)*(9x+4)
p(2) = 528, p'(2) = 612
```