COMP3012 - Compilers
Lab Session 1 - Exercises

Venanzio Capretta

Wednesday 2 October 2024

We start developing a Haskell interpreter for a simple language of Arithmetic Expressions. The formal grammar of the language is the following:

expr ::= **int** | **expr** `+` **expr** | **expr** `-` **expr** | **expr** `*` **expr** | **expr** `/` **expr** | `-` **expr** | `(` **expr** `)`

1 Scanner

Write a scanner function that takes as input a string and, in the case that the string is a correct expression according to the grammar, produces a list of tokens:

```
scan :: String -> [Token]
```

where tokens are defined by the following data type:

```
data Token = IntLiteral Integer    -- Numbers
           | Oper Operator         -- Operators (no arity distinction)
           | OpenPar | ClosedPar   -- Parentheses
data Operator = Plus | Minus | Times | Divide
```

2 Parser

Write a parsing function that maps the list of tokens to an Abstract Syntax Tree:

```
parse :: [Token] -> AST
```

where the type of ASTs is defined as follows:

```
data AST = LitInteger Integer
         | BinOp BinOperator AST AST
         | UnOp  UnOperator AST
data BinOperator = Addition | Subtraction | Multiplication | Division
data UnOperator = Negation
```

3 Evaluator

Write an evaluation function that calculates the value of an AST:

```
evaluate :: AST -> Integer
```

You can then put the three components together to produce an interpreter:

```
eval :: String -> Int
```

4 Extension

Extend your interpreter by adding a new binary operator `%` for the reminder of the division, and unary operator `abs` for the absolute value of an expression.