Синтаксичний аналіз в Haskell

Керівник: Проценко В. С.

Виконала: Пивовар О. О.

Мета і постановка задачі

- Мета: проаналізувати та визначити особливості використання різних бібліотек Haskell (а саме Applicative Parsing, Attoparsec та Parsec), які надають можливість здійснювати синтаксичний аналіз.
- Постановка задачі: Розробити три програми, які надають можливість здійснювати синтаксичний аналіз певної мови програмування. Необхідно використати три різні бібліотеки (для кожної програми інша).

Синтаксичний аналіз

- Синтаксичний аналіз – процес під час якого встановлюється, чи певна послідовність символів відповідає правилам заданої граматики.
- Результат виконання програми у випадку коректних вхідних даних – дерево розбору.

Огляд існуючих засобів розробки

- Parsec
- Trifecta
- Megaparsec
- Attoparsec
- Happy
- Alex
- Regex-Applicative
- Aeson
- Cassava
- Frown
- Polyparser
- Frisby

Applicative Parsing

- Аналіз регулярної мови без використання монад
- Тип даних: RE s a
- Аплікативний стиль написання
- Функція match або =~ для тестування

```
-- base = number | "true" | "false" | "this" | "new" id
"(" ")" | id
base :: RE Char Expr
base = nmb <|> bool <|> this <|> newId <|> idExpr
 where
  nmb = lexem numb
  bool = BConst <$> (true <|> false)
  true = reserved "true" *> pure True
  false = reserved "false" *> pure False
  this = reserved "this" *> pure This
  idExpr = Var <$> (lexemIden)
  newId = NewId <$> (reservedSp "new" *> lexemIden <*
  symbol '(' <* symbol ')')
```

```
numb :: RE Char Expr
numb = (NConst . read) <$> (neg <|> num)
 where
  num = some (psym isNumber)
  neg = (:) <$> sym '-' <*> num
```

```
*ApplicativeParsing> match base "new A()"
Just (NewId "A")
*ApplicativeParsing> "10" =~ base
Just (NConst 10)
*ApplicativeParsing> "-d10" =~ base
Nothing
*ApplicativeParsing> "a)" =~ base
Nothing
```

```
*ApplicativeParsing> match base "10"
Just (NConst 10)
*ApplicativeParsing> match base "thue"
Just (Var "thue")
*ApplicativeParsing> match base "true"
Just (BConst True)
*ApplicativeParsing> match base "this"
Just This
*ApplicativeParsing> match base "new A"
Nothing
```

Attoparsec

- Ефективна робота із мережевими протоколами, складними текстовими, двійковими форматами файлів
- Поступовий аналіз рядків
- Робота з Text та ByteString
- Тип даних: Parser a
- do-нотація
- Аплікативний стиль

```
numb :: Parser Expr
numb = do
 n <- signed decimal;
 return $ NConst n
```

```
numb' :: Parser Expr
numb' = NConst <$> (signed decimal)
```

```
bool :: Parser Expr
bool = choice [true, false]
 where
  true = reserved "true" *> return (BConst True)
  false = reserved "false" *> return (BConst False)
```

```
parseStr :: String -> Either String Program
parseStr str = parseOnly program (pack str)
```

Attoparsec. Тестування

```
class Factorial {
    public static void main(String[] a) {
         System.out.println(new Fac().ComputeFac(10));
    }
}
class Fac {
    public int ComputeFac(int num) {
        int num_aux;
        if (num < 1)
            num_aux = 1;
        else
            num_aux = num * (this.ComputeFac(num-1));
        return num_aux;
    }
}
```

```
*AttoparsecParsing> parseStr prog1
Right (Program {maincls = MainClass {className = "F
actorial", argName = "a", mainVars = [], mainStmt =
 [Print (CallM (NewId "Fac") "ComputeFac" [NConst 1
0])]}, othercls = [ClassDecl {classId = "Fac", exte
ndsClass = Nothing, varsDecl = [], methodsDecl = [M
ethodDecl {methodType = In, methodId = "ComputeFac"
, methodParam = [(In,"num")], methodVars = [(In,"nu
m_aux")], methodStmt = [If (BinOp Less (Var "num")
(NConst 1)) (Assign "num_aux" (NConst 1)) (Assign "
num_aux" (BinOp Mul (Var "num") (CallM This "Comput
eFac" [BinOp Sub (Var "num") (NConst 1)])))], metho
dRes = Var "num_aux"}]}]})
```

```
*AttoparsecParsing> parseStr prog1
Left "endOfInput"
```

Parsec

- Аналоги в інших мовах програмування
- Встановлюється зі стандартними бібліотеками
- Комбінатор try
- Обробка помилок
- Додаткові модулі

```
*ParsecParsing> parseStr prog1
*** Exception: (line 9, column 13):
unexpected ";"
expecting space, "//", "/*", "{", "if", "while", "System" or letter
CallStack (from HasCallStack):
  error, called at ParsecParsing.hs:289:16 in main:ParsecParsing
```

```
identifier :: Parser Id
identifier = try (do
    idn <- iden
    if elem idn reservedL
    then unexpected ("reserved word " ++ show idn)
    else return idn)
```

```
*ParsecParsing> parse identifier "" "if"
Left (line 1, column 3):
unexpected reserved word "if"
expecting letter or digit or "_"
```

```
reserved :: String -> Parser ()
reserved str =
 try (lexem $ string str *>
 notFollowedBy alphaNum *> return ())
```

```
*ParsecParsing> parseStr prog1
Program {maincls = MainClass {className = "Factoria
l", argName = "a", mainVars = [], mainStmt = [Print
 (CallM (NewId "Fac") "ComputeFac" [NConst 10])]},
thercls = [ClassDecl {classId = "Fac", extendsClass
 = Nothing, varsDecl = [], methodsDecl = [MethodDec
l {methodType = In, methodId = "ComputeFac", method
Param = [(In,"num")], methodVars = [(In,"num_aux")]
, methodStmt = [If (BinOp Less (Var "num") (NConst
)) (Assign "num_aux" (NConst 1)) (Assign "num_aux"
(BinOp Mul (Var "num") (CallM This "ComputeFac" [Bi
nOp Sub (Var "num") (NConst 1)]))))], methodRes = Va
r "num_aux"}]}]}
```

Порівняння використаних бібліотек

- Аналіз нерегулярної мови
- Швидкість здійснення аналізу
- Робота із Text або ByteString
- Комбінатор try
- Робота з помилками

Висновки

- В процесі роботи було проаналізовано та вивчено особливості використання різних бібліотек Haskell (а саме Applicative Parsing, Attoparsec та Parsec), які надають можливість здійснювати синтаксичний аналіз. Як результат було створено три програми, які використовували функціонал вище згаданих бібліотек.

Дякую за увагу!