Lisp parser

Sebastian Cielemęcki

7 lutego 2015

1 Opis projektu

Lisp parser to prosty parser wyrażeń lispowych. Napisany został w języku Haskell. Dla podanego wyrażenia generowane jest AST (Abstract Syntax Tree) reprezentowane za pomocą zdefiniowanego typu danych Expr.

2 Struktura projektu

2.1 Reprezentacja wyrażeń lispowych

Wyrażenie ma jedną z następujących postaci:

- atom (np. +, f, cdr)
- string (np. “napis”)
- liczba (ciąg cyfr)
- lista, będąca ciągiem (0 lub więcej) wyrażeń oddzielonych białymi znakami, ograniczonym nawiasami (np. (+ 1 2))

W Haskellu wyrażenie reprezentowane jest w następującej formie:

data **Expr** = **Atom** String |**Number** Int |**Str** String |**List** [Expr]

2.2 Parsery monadyczne

Definiujemy typ

newtype **Parser** a = **Parser** (**String** ->[(a, **String**)])

Parser operuje na ciągu znaków i zwrasta listę par zawierających przeczytaną wartość typu a oraz pozostałą część tekstu.
Ponadto definiujemy **Parser** jako instancję klasy **Monad** oraz **MonadPlus** (operacje **return**, **bind**, **zero**, **mplus**). Pozwoli to na łączenie prostych parserów w bardziej złożone. Operacja **mplus** jest kombinatorem wyboru.

2.3 Podstawowe parsery i kombinatory

- item :: **Parser Char** - "zjada" pojedynczy znak
- sat :: (**Char** ->**Bool**) ->**Parser Char** - zjada znak, jeśli spełnia predykat
- isChr :: **Char** ->**Parser Char** - czyta znak, o ile jest równy podanemu argumentowi

Aby czytać sekwencję określonych wartości, potrzebujemy odpowiednich kombinatorów do wielokrotnej aplikacji:

- many :: **Parser** a ->**Parser** [a] - 0 lub więcej wystąpień określonej wartości **Parser** a ->**Parser** [a] - 1 lub więcej wystąpień
- sepby :: **Parser** a ->**Parser** a ->**Parser** [a] - czyta ciąg wartości przedzielonych separatorem

2.4 Leksykalne kombinatory do wyrażeń lispowych

- space :: **Parser String** - czyta 1 lub więcej spacji
- qt :: **Parser Char** - sprawdza, czy podany znak jest cudzysłowem (tzn. jego początkiem lub końcem)
- lpar :: **Parser Char** oraz rpar :: **Parser Char** - odpowiednio czytanie otwierającego i zamykającego nawiasu (okrągłego)
- digit :: **Parser Char** - czyta znak, jeśli jest cyfrą
- number :: **Parser Int** - czyta ciąg cyfr
- sym :: **Parser Char** - czyta litery i symbole (składniki atomów i stringów)
- sq :: **Parser String** - czyta ciągi symboli

Zdefiniowanych jest 5 parserów generujących elementy AST:

- numTerm
- atomTerm
- strTerm
- listTerm
- expr

Gdzie **expr** jest połączeniem powyższych za pomocą kombinatorów wyboru:

expr = numTerm ‘mplus‘ atomTerm ‘mplus‘ strTerm ‘mplus‘ listTerm

Funkcja parseExpr :: **String** ->**Expr**
parseExpr = fst . head . parse expr

zwraca AST sparsowanego wyrażenia, przy czym
parse (**Parser** a) = a

3 Przykładowe wykonania

>5
    Number 5
>(+ 1 (* 2 3))
    List [Atom "+",Number 1,List [Atom "*",Number 2,Number 3]]
>(sth "one" "two")
    List [Atom "sth",Str "one",Str "two"]

[0]Monadic Parsing in Haskell - `http://www.cs.nott.ac.uk/~gmh/pearl.pdf`
[0]Monadic parser - `https://www.fpcomplete.com/user/fniksic/monadic-parser`