Funktionale Programmierung

`http://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2013/`

Übungsblatt 7 (Parsen, Review)

Mi, 2013-12-04

Hinweise

- Lösungen sollen als Haskell Quellcode in das persönliche Subversion (svn) Repository hochgeladen werden. Die Adresse des Repositories wird per Email mitgeteilt.
- **Alle** Aufgaben müssen bearbeitet und pünktlich abgegeben werden. Falls das sinnvolle Bearbeiten einer Aufgaben nicht möglich ist, kann eine stattdessen eine Begründung abgegeben werden.
- Wenn die Abgabe korrigiert ist, wird das Feedback in das Repository hochgeladen. Die Feedback-Dateinamen haben die Form `Feedback-<user>-ex<XX>.txt`.
- Allgemeinen Fragen zum Übungsblatt können im Forum (`http://proglang.informatik.uni-freiburg.de/forum/viewforum.php?f=38`) geklärt werden.

Abgabe: Mi, 2013-12-11

Aufgabe 1 (Review)

Wie Sie vielleicht bemerkt haben, sind die Korrekturen der vergangenen Blätter erst kürzlich fertig geworden.

Aus diesem Grund gibt es auf diesem Blatt nur eine richtige Aufgabe (nämlich Aufgabe 2). Das soll Ihnen Gelegenheit geben, das Feedback zu den vorherigen Abgaben anzuschauen. Wenn die Korrekturen unverständlich sein sollten oder Sie weitere Fragen zu den vorherigen Blättern haben, melden Sie sich bitte im Forum, in der Übung oder beim Assistenten.

Aufgabe 2 (While)

Implementieren Sie einen Parser für die folgende Grammatik einer einfachen Programmiersprache:

```
stmts ::= stmt ';' stmts
        | stmt
stmt  ::= 'while' exp 'do' stmts 'done'
        | id ':=' exp
exp   ::= 'if' exp 'then' exp 'else' exp 'fi'
        | aexp cmp aexp
        | 'not' exp
        | aexp
aexp  ::= num
        | id
        | '(' aexp op aexp ')'
cmp   ::= '<=' | '>' | '==' | '!='
op    ::= '+' | '-' | '*' | '/'
num   ::= "[0-9]+"
id    ::= "[a-zA-Z][a-zA-Z0-9]*"
```

Zur Grammatik-Syntax: Terminalsymbole stehen entweder wörtlich in einfachen Anführungszeichen (z.B. `'if'`) oder werden als regulärer Ausdruck in doppelten Anführungszeichen dargestellt (z.B. `"[0-9]+"`).

Ein Beispielprogramm der Sprache:

```
x:=0; y :=5;
while x <= 10 do
y := (y * 5); x := (x + 1)
done;
y := if y > 10000 then 10000 else y fi
```

Auf der Homepage finden Sie das Modul `Parser2.hs` mit dem Parser aus der Vorlesung mit entsprechenden **Functor**, Applicative, Alternative, **Monad** und **MonadPlus** Instanzen. Sie finden dort auch das Modul `While.hs`, das einen Typ `Program` definiert und einen Lexer

```
1 lexer :: Parser Char [Token]
```

der zum Vorverarbeiten des Strings verwendet werden kann.

Implementieren Sie also eine Funktion

```
1 parseWhile :: String -> Maybe Program
```

die einen String in einen Program Wert parst, falls möglich.