Zadanie K: SML - Parsowanie lambda termów

Rozważamy następujący typ dla reprezentacji lambda termów:

```
type label = string
datatype lterm = Number of (IntInf.int)
                         |Label of label
                         |App of lterm*lterm
                         |Abs of label*lterm
                         |Let of ((label*lterm) list) * lterm
```

Zadanie polega na implementacji parsera lambda termów. Wyjściowa gramatyka dla lambda termów:

```
<term> ::= "("<term>")"
        | <number>
        | <label>
        | <term> <term>  (aplikacja)
        | <term> "+" <term>
        | <term> "::" <term>
        | "fn" <label> "=>" <term>
        | "let" <definition>* "in" <term> "end"
<definition> ::= "val" <label> "=" <term>
        |"fun" <label> <label>+ "=" <term>
```

Wszystkie ciągi w cudzysłowach to symbole terminalne. Ponadto terminalnymi symbolami są następujące ciągi:

```
<label> ::= <alpha> <alphaNum>*
<number> ::= <digit>+
```

gdzie digit, alpha, alphaNum odpowiadają pojedynczym znakom spełniającym predykaty isDigit, isAlpha, isAlphaNum (wg. Standard ML Basis Library). Parsowanie terminali (tzw. analiza leksykalna) powinno być "chciwe", tzn. parsowany jest zawsze najdłuższy terminal jaki jest możliwy (np. "123" jest parsowane jako 123, a nie 12 i 3). Między symbolami terminalnymi oraz na początku i na końcu strumienia wejściowego może być dowolnie wiele białych znaków (spełniających isSpace). Te znaki należy zigorować. Terminale nie zawierają białych znaków w środku - ": :" jest niepoprawne, a " :: " poprawne. Zakładamy dodatkowo, że słowa kluczowe `fn,let,in,end,val,fun` nie mogą być parsowane jako `<label>`.

Niejednoznaczności powstałe przez operatory infiksowe "+", "::" oraz aplikację rozwiązujemy za pomocą następujących reguł:

- aplikacja wiąże do lewej,
- :: wiąże do prawej,
- + wiąże do lewej,
- najmocniej wiąże +, potem ::, najsłabiej aplikacja.

<term> pod regułą "fn" powinien być parsowany chciwie.

Zdefiniowane operacje w zapisie infiksowym powinny być interpretowane następująco:

- `t_1 t_2 --> App t1 t2`
- `t_1+t_2 --> App (App (Label "PLUS") t1) t2`
- `t_1::t_2 --> App (App (Label "CONS") t1) t2`

Wysłany program powinien:

- przeczytać ze standardowego wejścia najdłuższy ciąg, który jest poprawną reprezentacją termu,
- sparsować term,
- wypisać term na standardowe wyjście zamieniając go na `string` za pomocą dostarczonej funkcji `lterm2str: lterm -> string`,
- w przypadku gdy żaden prefiks ciągu znaków na standardowym wejściu nie jest poprawną reprezentacją lambda termu program powinien wypisać "NO PARSE".

Dopuszczalne i wskazane jest wykorzystanie kodu monad z ćwiczeń. Nie należy w pliku z programem definiować typu `lterm` ani funkcji `lterm2str`.

```
- parseT "S K K";
val it = App (App (Label "S",Label "K"),Label "K") : lterm
- parseT "fn a => fn b => b a";
val it = Abs ("a",Abs ("b",App (Label "b",Label "a"))) : lterm
- parseT "fn a => fn b=> b+a";
val it = Abs ("a",Abs ("b",App (App (Label "PLUS",Label "b"),Label "a"))) : lterm
- parseT "1::2+3::NIL";
val it =App (App (Label "CONS",Number 1), App
       (App (Label "CONS",App (App (Label "PLUS",Number 2),Number 3)),
        Label "NIL")) : lterm
- parseT "hd 1::2 3";
val it = App (App (Label "hd",App (App (Label "CONS",Number 1),Number 2)),Number 3) : lterm
- parseT "let val x=1 in y end + 3";
val it = App (App (Label "PLUS",Let ([("x",Number 1)],Label "y")),Number 3) : lterm
- parseT "let fun f x y= x+y val z=2 in f z z end";
val it = Let
    ([("f",Abs ("x",Abs ("y",App (App (Label "PLUS",Label "x"),Label "y")))),
  ("z",Number 2)],App (App (Label "f",Label "z"),Label "z")) : lterm
```