Progetto di Linguaggi di Programmazione
Prima parte:
Analizzatore Lessicale

May 17, 2007

Realizzare un analizzatore lessicale in ML che sia in grado di riconoscere:

- numeri interi (es. 10, ~ 234);
- stringhe (es. "via verdi 12");
- identificatori (es. var1);
- le seguenti parole chiave: let, in, end, letrec, and, NIL, T, F;
- i seguenti operatori: ADD, SUB, MUL, DIV, REM, EQ, LEQ, CAR, CDR, CONS, ATOM, IF e LAMBDA;
- i seguenti segni di punteggiatura: ( ) [ ] = :: $ (dove $ e' il marcatore della fine della stringa da analizzare).

L'analizzatore lessicale riceve in input una stringa e produce in output un lista di coppie (token,lexema).
I token vengono modellati in ML con il seguente datatype:

```
datatype token = LET | IN | END | LETREC | AND |
                 LAMBDA | OP | ID | SYM | NM| STR | BOOL | Nil| Notoken
```

Il token ausiliario "Notoken" puo essere utile nell'implementazione ma non è obbigatorio utilizzarlo.
I lexemi vengono modellati con i seguenti tipi:

```
datatype s_espressione = NUM of int |
                         STRINGA of string |
                         T | F | NIL

datatype meta_S = M of s_espressione | S of string
```

Il tipo delle coppie token-lexema puo' essere abbreviato con:

```
type token_lexema = token * meta_S
```

Le coppie token-lexema corrispondenti agli oggetti che devono essere riconosciuti sono le seguenti:

- numeri interi: $(NM, M(NUM(i)))$, dove i è un intero;
- stringhe: $(STR, M(STRINGA(s)))$, dove s è una stringa;
- identificatori: $(ID, S(is))$, dove is è la stringa corrispondente all'identificatore;
- let: $(LET, S(\text{“}let''))$;
- in: $(IN, S(\text{“}in''))$;

- end: $(END, S(\text{“}end''))$;
- letrec: $(LETREC, S(\text{“}letrec''))$;
- and: $(AND, S(\text{“}and''))$;
- NIL: $(Nil, M(NIL))$;
- T, F :$(BOOL, M(T))$, $(Bool, M(F))$;
- operatori: $(OP, S(os))$, dove os è la stringa corrispondente al nome dell'operatore (es. $(OP, S(\text{“}ADD''))$);
- LAMBDA: $(LAMBDA, S(''LAMBDA''))$;
- segni di punteggiatura: $(SYM, S(sy))$, dove sy è la stringa contenente il simbolo di punteggiatura.

La signature della funzione che realizza l'analizzatore è dunque la seguente:

```
lexi(s:string, tkl: token_lexema list) : token_lexema list
```

Per esempio:

```
val x = "IF ( LEQ ( I 0 ) CONS ( SUB ( 0  I ) NIL )  CONS ( I  NIL  ) )  $";
lexi(x,[]);
```

da' in output:

```
[(OP, S "IF"), (SYM, S "("), (OP, S "LEQ"), (SYM, S "("), (ID, S "I"),
     (NM, M(NUM 0)), (SYM, S ")"), (OP, S "CONS"), (SYM, S "("), (OP, S "SUB"),
     (SYM, S "("), (NM, M(NUM 0)), (ID, S "I"), (SYM, S ")"), (Nil, M NIL),
     (SYM, S ")"), (OP, S "CONS"), (SYM, S "("), (ID, S "I"), (Nil, M NIL),
     (SYM, S ")"), (SYM, S ")"), (SYM, S "$")] : (token * meta_S) list
```

Suggerimenti

1. Per passare dalla stringa in input alla lista di caratteri in essa contenuti utilizzare la funzione built-in $explode$. In particolare questo comporta che nella stringa in input non possono esserci tab oppure new-lines.
2. Gli identificatori sono sequenze che iniziano con una lettera maiuscola o minuscola seguita da lettere o cifre compreso il carattere "_". Potrebbe essere utile la seguente funzione:

```
fun isletter(c: char): bool =
if c = #"a" orelse c = #"b" orelse c = #"c"  orelse c = #"d" orelse c = #"e"
orelse c = #"f" orelse c = #"g"  orelse c = #"h" orelse c = #"i"
orelse c = #"j" orelse c = #"k"  orelse c = #"l" orelse c = #"m"
orelse c = #"n" orelse c = #"o"  orelse c = #"p" orelse c = #"q"
orelse c = #"r" orelse c = #"s"  orelse c = #"t" orelse c = #"u"
orelse c = #"v" orelse c = #"w"  orelse c = #"x" orelse c = #"y"
orelse c = #"z" orelse
c = #"A" orelse c = #"B" orelse c = #"C"  orelse c = #"D" orelse c = #"E"
orelse c = #"F" orelse c = #"G"  orelse c = #"H" orelse c = #"I"
orelse c = #"J" orelse c = #"K"  orelse c = #"L" orelse c = #"M"
orelse c = #"N" orelse c = #"O"  orelse c = #"P" orelse c = #"Q"
orelse c = #"R" orelse c = #"S"  orelse c = #"T" orelse c = #"U"
orelse c = #"V" orelse c = #"W"  orelse c = #"X" orelse c = #"Y"
orelse c = #"Z" orelse c= #"_"
then true
else false;
```