LFC Lab (Linguaggi Formali e Compilatori) - Note del Corso

Edoardo Lenzi

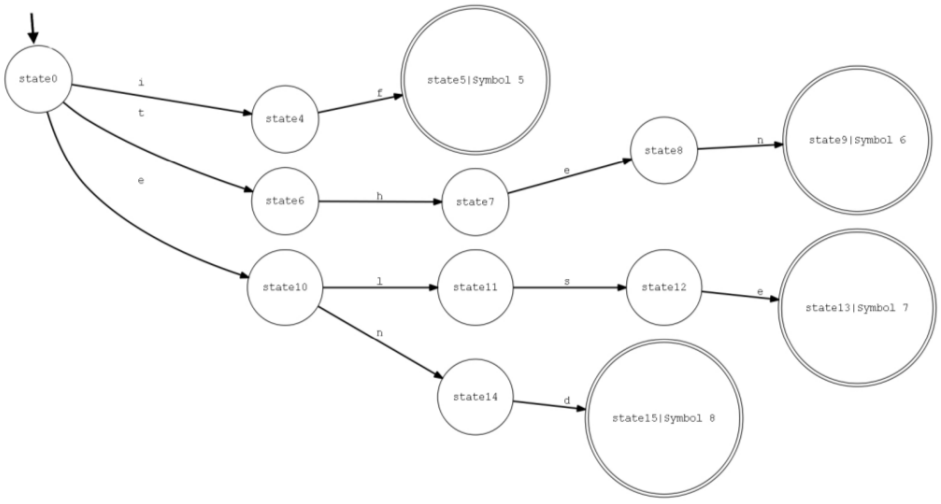
November 9, 2017

Contents

Chapter 1

Introduzione

Un Lexer λ é un DFA, ogni stato finale ϕ_i é associato ad una regular expression α_i. Una stringa $s \in L(\lambda)$, il pattern matching termina in uno stato finale ϕ_j ed s é associata ad una regular expression α_j; token: (s, α_j).



- (if, 5)
- (then, 6)
- (else, 7)
- (end, 8)

Data la descrizione del linguaggio che vuoi matchare lui ti genera il codice C per farlo.

1.1 Get started

Windows	https://sourceforge.net/projects/winflexbison/
Linux	sudo apt-get install bison flex gcc
Mac	XCode

Chapter 2

FLEX (Fast Lex)

2.1 Lex matching character

2.1.1 Operatori

$\alpha \cdot \beta$	concatenazione
$\alpha \vert \beta$	alternazione
α^+	una o piú ripetizioni
α^*	zero o piú ripetizioni
$\alpha?$	0 o 1 ripetizione
$\alpha\{n,m\} \vert n \leq m$	matches α from n to m times

```
[a-z]    tutti i caratteri da 'a' a 'z'
[0-9a-z_]* tutti i caratteri n volte
.        ogni carattere tranne \n
a?       a 0 o 1 volta
a{n,m}     matcha a da n m volte
a$       a se e' alla fine di una linea
^a       a se e' all'inizio di una linea
a/b      a iif b segue

[^C]     complementare
[^CB]    = [^C^B] matcha bat ma non Bat o Cat
<<EOF>>
"stringa"
"float" | "int"
."at"      matches words: "cat", "rat, etc.
[0-9a-z_] * identificatori
```

2.2 Template Lex

```
%{ /* This code is copied verbatim (tale e quale) into the lexer s source code . */ %}

/* " Named " regular expressions here . */

%%
/* " Anonymous " Regular expressions here . */
%%

/* This section is copied verbatim into the lexer 's source . */
```

```
%{ /* Copied verbatim in lexer 's source . */ %}

Type ( " int " | " float " )
%%
{ Type }   { /* A Semantic action . */ }
[a - z ]+  { /* Another one . */ }
%%
```

```
int yywrap () {
   return 1;  //if I find eof should stop lexing?
}

int main ( int iArgC , char ** lpszArgV ) {
   yylex ();  // Starts lexing .
}
```

2.3 Compilare

```
> flex Input.l
> CC lex.yy.c -o Lexer.out -std=c99
> Lexer.out < In.txt
```

2.4 Variabili generate da Lex

```
FILE * yyin ;    /* Default value is stdin . */
FILE * yyout ;   /* Default value is stdout . */
int yyleng ;     /* Number of characters read . */
char * yytext ;    /* The buffer on which characters are copied during pattern matching*/
```

2.5 Context Sensitivity (CS)

In base al token letto devo comportarmi in modo diverso. Ho delle **start conditions** (o start state), solo una é attiva. Il primo start state é l'**initial**. Uno start state puó essere **inclusivo** o **esclusivo**.

Da uno start state esclusivo solo re correlate ad esso sono raggiungibili.

Da uno start state inclusivo tutte le re correlate ad esso ed anche quelle non correlate agli altri start state sono raggiungibili.

```
%{ /* */ %}

% x String  // Exclusive start states
% s Cond   // Inclusive start states

%%
...
%%
```

2.6 Associare regole a stati

Per assegnare uno start state ad una re uso <**NOME SS**>. Per entrare in uno stato uso il comando **BEGIN**. All inizio il lexer é nello stato **INITIAL**.

```
%{ /* */ %}

% x Cond

%%
" \" "          { /* Read " char . */ BEGIN Cond ;}
< Cond >(^[ " ])+ { /* Consume string content . */ }
< Cond > " \" "    { /* Read " char . */ BEGIN INITIAL ;}
%%
```

2.6.1 Stati inclusivi ed esclusivi

```
%{ /* EXCLUSIVE START STATE*/ %}

%x String Unreacheable

%%
" \" "              { /* Read " char . */ BEGIN String ;}
< String > (^[ " ])+ { /* Consume string content . */ }
< String > " \" "  { /*Read " char . */ BEGIN INITIAL ;}
< Unreacheable > "end" { /* Will never be matched . */ }
%%

%{ /* INCLUSIVE START STATE */ %}

%s String

%%
" \" "           { /* Read " char . */ BEGIN String ;}
< String >(^[ " ])+ { /* Consume string content . */ }
< String > " \" "  { /* Read " char . */ BEGIN INITIAL ;}
" end "          { /* Will be matched . */ }
%%
```

Gli start state sono realizzati tramite uno stack, ho tre funzioni per manipolarli:

- void yy push state(int NewState) //pusho in cima allo stack, equivalente a BEGIN NewState;
- void yy pop state() //fa una pop, equivalente a BEGIN;
- int yy top state() //fa la top, non esistono metacomandi per farla;

2.7 Ambiguous Specifications

Dati α e β regular expressions taliché $L(\alpha) \subset L(\beta)$, devo stare attento all'ordine in cui le scrivo, quelle piú in alto hanno precedenza su quelle piú in basso.

> Se con molte regular expression posso arrivare in un nodo, vince quella a prioritá maggiore. Regular expressions generating constant finite languages must be placed first, or they will be obscured.

Una volta che una stringa é matchata viene **consumata**, le altre re non la potranno piú matchare. Posso anche usare **REJECT** per darla in pasto alla seconda re in ordine di prioritá.

```
%{ int iCounter = 0; %}

%%
" abcde "  { iCounter +=1; REJECT ;}
" abcd " { iCounter +=1; REJECT ;}
" abc "    { iCounter +=1; REJECT ;}
" ab "     { iCounter +=1; REJECT ;}
"a"        { iCounter +=1; }
.       { /* Consumes the rest . */ }
%%

int main (){
   yylex ();
   /* iCounter = 5 when input is " abcde ". */
}
```

Chapter 3

Exercises

3.1 Lex

Your time: Devise a lexer accepting a Windows file path. Devise a lexer accepting a Linux file path. Devise a lexer accepting a non regular language.