

Progetto di Linguaggi e Compilatori 1 – Parte 2  
A.A. 2015/16

## Gruppo 14

Marco Bucchiarone  
Emanuele Tonetti  
Francesco Zilli

### Esercizio 1

(a)

Dato un testo formattato come

cognome nome/nomi data(gg/mm/aa) matricola altro-testo

con i campi separati da un numero arbitrario di spazi, le espressioni regolari, nella sintassi di flex, componenti l'espressione regolare  $e_{in}$  per eseguire la riformattazione del testo sono:

cognome ( $[a-zA-Z\-\backslash']^+$ )<sub>1</sub>

nome/nomi ( $([a-zA-Z\-\backslash])^+([ \ ]+[a-zA-Z\-\backslash])^+)^*$ )<sub>2</sub>

gg ( $([1-9])|([12][0-9])|3[01]$ )<sub>3</sub>

mm ( $([1-9])|(1[0-2])$ )<sub>4</sub>

aa ( $([0-9])|([1-9][0-9])$ )<sub>5</sub>

matricola ( $[0-9]+\{6\}$ )<sub>6</sub>

separatore (" ")

spazi ( $[ \ ]^+$ )

altro-testo (.)

dove, per semplicità di notazione, sono state numerate solo le parentesi contenenti le *regex* facenti il match dei campi che si vuole siano presenti nell'espressione  $e_{out}$ .

Quindi la *regex*  $e_{in}$  assumerà forma

$$e_{in} = \{\text{cognome}\}\{\text{spazi}\}\{\text{nome/nomi}\}\{\text{spazi}\}\{\text{gg}\}\{\text{separatore}\}\{\text{mm}\}\{\text{separatore}\}\{\text{aa}\}\{\text{spazi}\}\{\text{matricola}\}\{\text{spazi}\}\{\text{altro-testo}\}.$$

Volendo  $e_{out}$  della forma

$$\text{matricola nome/nomi cognome data(aaaa-mm-gg)}$$

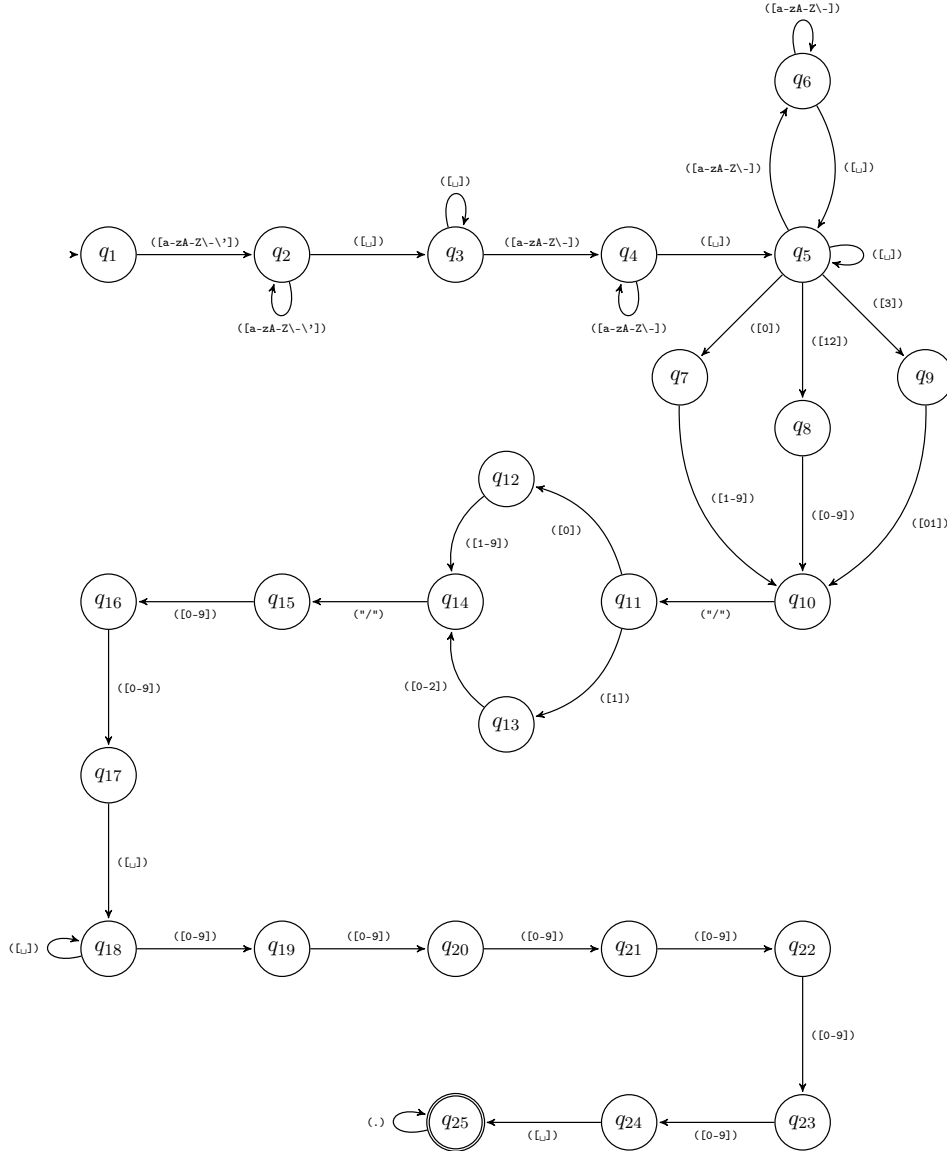
con i campi separati con un tabulatore ed assumendo che, tutte le date successive al 2000, non abbiano singole cifre non precedute da 0, si avrà

$$e_{out} = \backslash 6 \backslash \text{t} \backslash 2 \backslash \text{t} \backslash 1 \backslash \text{t} 20 \backslash 5 - \backslash 4 - \backslash 3$$

dove  $\backslash \text{t}$  indica il carattere di tabulazione.

(b)

Preso l'alfabeto  $\Sigma$  contenente i caratteri ASCII, il DFA minimo per  $e_{in}$  è:



dove, per chiarezza illustrativa, si sono indicati sugli archi il range di caratteri che, ricevuti dallo stato  $q_i$  causano la transizione allo stato  $q_{i+1}$ . Per la stessa ragione è stata omessa la rappresentazione dello stato pozzo a cui puntano tutti gli stati qualora ricevano un carattere non accettato dalle transizioni esplicitate.

## Esercizio 2

```

program esercizio ;
  const IMAX = 9 ; UNKNOWN = ...;    {IMAX 4, UNKNOWN 5 }
  type harray = array [ 0.. IMAX , 0.. IMAX ] of integer ;
  history = ^harray;
  ... ..
  var target , aim : integer ; {target 6, aim 7}
  mainharray : harray ;        { mainharray 8}

  procedure init ( i , j : integer ; h : history ; var z : integer );
    var u : integer ;
  begin
    for u := 0 to min ( IMAX , j ) do
      begin
        p(u*i , z);
        h^[i,u] := mainarray[u,j];
      end
    end;
  {
  init:
      ENT 2          — 1 + temp
      LDA 0 9        — variabile temporanea
      MST 1
      LDA 1 4        — IMAX
      IND
      LDA 0 5        — j di init
      IND
      CUP 2 min      — chiamo min(imax,j)
      STO            — temp := min(IMAX, j)
      LDA 0 8        —carico u
      LDC int 0
      STO            — u := 0
      UJP guard-init — while u <= temp do S; u += 1;

  body-init:
      MST 1
      LDA 0 8
      IND            — u
      LDA 0 4
      IND            — i
      MUL int        — u * i
      LDA 0 7
      IND            — l-val di z
      CUP 2 p        —siamo a p(u*i,z)
      LDA 0 6        — h : history
      IND            — history := ^harray

```

```

LDA 0 4      — index i
IND
IXA 10       — h[i][] pointer
LDA 0 8      — index u
IND
IXA 1        — h[i][u] pointer
LDA 1 8      —prendo esternamente mainharray(offset 8
IND
LDA 0 8      — index u
IND
IXA 10       — [u] pointer
LDA 0 5      — index j
IND
IXA 1        — [u][j] pointer
IND          —carico r-value di  mainharray[u,j]
STO          —memorizzo
LDA 0 8      — da qui faccio u += 1
LDA 0 8      — posso usare dup
IND
LDC int 1
SUM int
STO
guard-init:
LDA 0 8      — u
IND
LDA 0 9      — temp
IND
GTR
FJP body-init
RETP
}
function f ( n : real ) : integer ;
begin ... .. end ;
{
f:
ENT 0
...
RETF
}
procedure p ( x : integer ; var y : integer );
begin
y := f ( f ( x ));
while ( y = 0) and ( x > target ) do
p ( y , y )

```

```

    end;
{
p:
    ENT 0      — nessuna local var
    LDA 0 5
    IND        — viene fatto perch y  passata per reference
    MST 1      — pre-chiamata al primo f
    MST 1      — pre-chiamata al secondo f
    LDA 0 4
    IND
    FLT        — f() si aspetta real
    CUP 1 f
    CUP 1 f
    STO        — y:= f(f(x))
    UJP guard-p

body-p:
    MST 1      — preparo ricors p(y,y)
    LDA 0 5
    IND
    IND        — passo il valore del primo param
    LDA 0 5
    IND        — passo la "var" del secondo param
    CUP 2 p

guard-p:
    LDA 0 5
    IND
    IND
    LDC int 0
    NEQ        — devo invertire controllo
    LDA 0 4
    IND
    LDA 1 6    — carico target
    IND        —
    LEQ        — <=
    OR
    FJP body-p
    RETP
}
function min (x , y : integer ): integer ;
begin ... .. end ;
{
min:
    ENT 0
    ...

```

```

                                RETF
}
function alt ( i : integer ; x : real ) : real ;

    function f ( x : real ) : real ;
    begin f := 1 + 1/ x end ;
{
f:                                —interna ad alt
                                ENT 0                                —niente variabili locali
                                LDA 0 0    —variabile f stesso nome funzione
                                LDC real 1
                                LDC real 1    — oppure ricarico con LDC
                                LDA 0 4                                — x interna alla funz
                                IND
                                DIV real
                                SUM real
                                STO                                — assegno al valore di ritorno
                                RETF
}
begin
    if odd ( i ) then
        alt := alt ( i -1 , f ( x ) )
    else
        alt := alt ( i -1 , x )
    end ;
{
alt:
                                ENT 0
                                LDA 0 4
                                IND                                — carico i
                                ODD
                                FJP else
then:
                                LDA 0 0    — dove sar memorizzato val ritorno
                                MST 1    — preparo chiamata ricorsiva alt
                                LDA 0 4
                                IND
                                LDC int 1
                                SUB int
                                MST 0    — preparo chiamata alla funzione interna f;
                                LDA 0 5
                                IND
                                CUP 1 f
                                CUP 2 alt

```

```

                                STO
                                RETF
else :
                                LDA 0 0
                                MST 1
                                LDA 0 4
                                IND
                                LDC int 1
                                SUB int    — param i-1
                                LDA 0 5
                                IND          — param x
                                CUP 2 alt
                                STO
                                RETF
}
begin
    ...;
    p ( target , aim );
    init ( 20 , 30 , @mainharray , target );
    ...
end;
{
main:
    ...
    MST 0
    LDA 0 6 —target
    IND
    LDA 0 7 —aim
    CUP 2 p
    MST 0
    LDC int 20
    LDC int 30
    LDA 0 8    — mainharray
    LDA 0 6
    CUP 4 init
    ...
}

```

### Esercizio 3

compilato con bison 3.0.4 flex 2.6.0