Laboratorio di Software di Base
Analizzatore semantico di pagine web

Sandro Tosi

25 Luglio 2001

1 Introduzione

Il progetto si articola in tre parti: la prima riguarda la compilazione di un analizzatore lessicale usando Lex, la seconda invece si tratta di una analisi sintattica e semantica di pagine web tramite un programma in discesa ricorsiva e l'ultima parte esegue questo stesso compito ma utilizzando YACC. Le ultime due parti utilizzano l'analizzatore lessicale precedentemente ottenuto, ma sono tra di loro indipendenti.

2 La grammatica

La grammatica utilizzata per l'analizzatore lessicale di questo progetto rappresenta solo un piccolo sottoinsieme utilizzabile per la realizzazione di pagine web, e in questo ambito ci siamo limitati alle parti essenziali del codice HTML utilizzando la seguente grammatica:

```
Html     ::=  <html> Head <body> Body </body> </html>
Char     ::=  a | b | ...  | z | A | B | ...  | Z
Stringa  ::=  ϵ | Char Stringa
Head     ::=  <head> <title> Stringa </title> </head>
Body     ::=  ϵ | Stringa Body |
              <a href=''Stringa.htm''>Stringa</a> Body
```

in cui sono stati sottolineati i simboli terminali mentre sono indicati con il primo carattere maiuscolo i simboli non terminali.

2.1 Ricorsione sui link

La grammatica appena vista consente anche l'introduzione di link all'interno della pagina da riconoscere e quindi è stata aggiunta la caratteristica di

analisi di tutte le pagine a cui viene fatto riferimento, come unica esclusione per la pagina stessa per cui non è consentito il self-refering, anche per implementare una sorta di controllo semantico.

Seguire i collegamenti presenti nella pagina corrente per eseguire l'analisi anche di queste pagine ha rivelato inattesi problemi, il più importante dei quali riguarda la possibilità di entrare in un loop infinito: si consideri la pagina 1 al cui interno è presente un link verso la pagina 2, la quale a sua volta contiene un collegamento nuovamente verso la pagina 1; a questo punto, se non venissero prese delle precauzioni ci sarebbe la possibilità di continuare ad analizzare sempre queste due pagine senza mai fermarsi.

Per ovviare a questo problema abbiamo utilizzato il seguente metodo: durante la scansione della pagina vengono aggiunti in uno stack i collegamenti ad altre pagine che via via vengono trovati solo se queste non sono già state visitate oppure aggiunte allo stack. In questo modo l'analisi di una pagina viene portata a termine prima di analizzare le pagine collegate a questa, e facendo così in modo che le pagine ancora da analizzate siano contenute nello stack e possano essere prelevate e scandite successivamente, una volta che la scansione della pagina corrente sia andata a buon fine, altrimenti la scansione si interromperebbe senza proseguire con le pagine successive.

Inoltre, per evitare che un riferimento venga fatto sulla stessa pagina attualmente esaminata, viene utilizzata una variabile che contiene il nome del file corrente: se questo nome corrisponde al file verso il quale punta il collegamento allora viene generato un errore e l'analisi si interrompe, restituendo un errore.

Per dare un limite al numero di link verso pagine diverse, viene utilizzata anche una costante, chiamata `maxS`, che è stata posta al valore 100 e che limita anche la dimensione dell'array dove vengono mantenute le informazioni delle pagine già visitate.

3 L'utilizzo di Lex e YACC

Il programma che analizza le pagine web in discesa ricorsiva fa uso dell'analizzatore lessicale generato da Lex secondo la specifica allegata alla relazione, così che la pagina sorgente venga suddivisa in token e la sua analisi risulti notevolmente più semplice lasciandoci il compito di scrivere le procedure Pascal per gestire l'analizzatore.

Per l'analisi tramite YACC viene utilizzata nuovamente la specifica Lex ma questa volta tutto il lavoro di analisi sintattica viene svolta dallo stesso YACC sollevandoci dall'implementazione prima necessaria nell'analisi in discesa ricorsiva. La prevenzione di link diretti sulla pagina corrente e sull'aggiunta delle pagine collegate a questa nello stack, sono realizzate con le stesse procedure già utilizzate nel programma di discesa ricorsiva e aggiun-

gendo una riga di codice Pascal nella sezione delle regole che richiama le procedure menzionate.

4 Sorgenti

Oltre ai sorgenti e ai programmi compilati sono stati inseriti anche dei gruppi di pagine di prova per testare il programma, due corretti (nominati `ok1.htm` e `ok2.htm`), e quattro sbagliati (chiamati `ko1.htm`, `ko2.htm`, `ko3.htm` e `ko4.htm`) i quali mostrano la gamma di errori ottenibili in questo tipo di esercizio.

Listing 1: Programma Lex (Progetto.l)

```
link            "<a href=\""
html             "<html>"
htmlfine         "</html>"
body             "<body>"
bodyfine         "</body>"
head             "<head>"
headfine         "</head>"
title            "<title>"
titlefine        "</title>"


file            ({lettera}|{cifra})+".htm"
linkfine        "\">"{stringa}"</a>"
spazi           [ \t\n]
seq_spazi       {spazi}+
lettera         [a-zA-Z]
cifra           [0-9]
stringa         ({lettera}|{cifra}|" ")*


%%

{html}          return(htmlCode);
{htmlfine}      return(htmlfineCode);
{body}          return(bodyCode);
{bodyfine}      return(bodyfineCode);
{head}          return(headCode);
{headfine}      return(headfineCode);
{title}         return(titleCode);
{titlefine}     return(titlefineCode);
{stringa}       return(stringaCode);
{link}          return(linkCode);
```

```
{file}          return(fileCode);
{linkfine}      return(linkfineCode);
{spazi}           ;
{seq_spazi}       ;
.               return(errorCode);

%%
```

Listing 2: Programma in discesa ricorsiva (ProgDisc.pas)

```
program Discesa;

uses LexLib;

const
  htmlCode=10;
  htmlfineCode=20;
  bodyCode=30;
  bodyfineCode=40;
  headCode=50;
  headfineCode=60;
  titleCode=70;
  titlefineCode=80;
  stringaCode=90;
  linkcode=100;
  fileCode=110;
  linkfineCode=120;
  errorCode=500;

  maxS=100;

type
  list = ^node;
  node = record
           key: string;
           next: list;
         end;

var
  token: integer;
  current: string;
  stringinput: string;
  stack: list;
  tabsym: array[1..maxS] of string;
```

```
  N: integer;

function yylex:integer; forward;
procedure ErrorProc; forward;
Procedure Header; forward;
Procedure Main; Forward;
Procedure Link; Forward;
Procedure Add2Stack(link: string); Forward;


Procedure AvanzaToken;
begin
  token:=yylex;
end;

Procedure ErrorProc;
begin
  if token=errorCode then
    writeln(current,' Errore lessicale: ',yytext)
  else
    Writeln(current,' Errore sintattico: ',yytext);
  halt(0);
end;

Procedure Main;
begin
  if token=htmlcode then
  begin
    avanzatoken;
    Header;
  end
  else
    errorproc;
  if token=bodyCode then
  begin
    avanzatoken;
    while (token=stringacode) or (token=linkcode) do
    begin
      if token=linkcode then
        Link
      else
        avanzatoken;
    end;
    if token=bodyfinecode then
```

```
    begin
      avanzatoken;
      if token<>htmlfinecode then
        errorproc
    end
    else
      errorproc
  end
  else
    errorproc;
end;

Procedure Link;
begin
  if token=linkcode then
  begin
    avanzatoken;
    if token=filecode then
    begin
      add2stack(yytext);
      avanzatoken;
      if token=linkfinecode then
        avanzatoken
      else
        errorproc
    end
    else
      errorproc
  end
end;

Procedure Header;
begin
  if token=headcode then
  begin
    avanzatoken;
    if token=titlecode then
    begin
      avanzatoken;
      while token=stringacode do
        avanzatoken;
      if token=titlefinecode then
      begin
        avanzatoken;
```

```
        if token=headfinecode then
          avanzatoken
        else
          errorproc
      end
      else
        errorproc
    end
    else
      errorproc
  end
  else
    errorproc
end;

{$I progetto.pas}

procedure Push(link: string);
var
  temp: list;
begin
  new(temp);
  temp^.key:=link;
  temp^.next:=stack;
  stack:=temp;
end;

function Pop: string;
begin
  pop:=stack^.key;
  stack:=stack^.next;
end;

procedure add2stack(link: string);
var
  i: integer;
  visited: boolean;
begin
  if link=current then
  begin
    writeln(link,
     ' Non sono ammessi link verso la stessa pagina!');
    halt(0);
  end;
```

```
  i:=0;
  visited:=false;
  repeat
    i:=i+1;
    if tabsym[i]=link then
      visited:=true;
  until (visited) or (i=N);
  if (N=maxS) and (not visited) then
  begin
    writeln('Troppi link!');
    halt(0);
  end;
  if not visited then
  begin
    push(link);
    N:=N+1;
    tabsym[N]:=link;
  end;
end;

procedure Analizza(link: string);
begin
  assign(yyinput, link);
  assign(yyoutput, '');
  reset(yyinput);
  rewrite(yyoutput);
  avanzatoken;
  Main;
  writeln(current,' riconosiuto');
  close(yyoutput);
  close(yyinput);
end;

begin
  if paramcount<>1 then
  begin
    writeln;
    writeln('Usage:  Prog nome.htm');
    halt(0);
  end;
  new(stack);
  stack:=nil;
  N:=1;
  tabsym[1]:=ParamStr(1);
```

```
  Push(ParamStr(1));
  while stack<>nil do
  begin
    current:=pop;
    analizza(current);
  end;
end.
```

Listing 3: Programma YACC (ProgYacc.y)

```
%{
uses Lexlib, YaccLib;

const
  maxS=100;

type
  list = ^node;
  node = record
           key: string;
           next: list;
         end;

var
  token: integer;
  current: string;
  stringinput: string;
  stack: list;
  tabsym: array[1..maxS] of string;
  N: integer;

procedure Push(link: string);
var
  temp: list;
begin
  new(temp);
  temp^.key:=link;
  temp^.next:=stack;
  stack:=temp;
end;

function Pop: string;
begin
  pop:=stack^.key;
```

```
  stack:=stack^.next;
end;

procedure add2stack(link: string);
var
  i: integer;
  visited: boolean;
begin
  if link=current then
  begin
    writeln(link,
     ' Non sono ammessi link verso la stessa pagina!');
    halt(0);
  end;
  i:=0;
  visited:=false;
  repeat
    i:=i+1;
    if tabsym[i]=link then
      visited:=true;
  until (visited) or (i=N);
  if (N=maxS) and (not visited) then
  begin
    writeln('Troppi link!');
    halt(0);
  end;
  if not visited then
  begin
    push(link);
    N:=N+1;
    tabsym[N]:=link;
  end;
end;


%}

%token htmlCode htmlfineCode bodyCode bodyfineCode
%token headCode headfineCode titleCode titlefineCode
%token stringaCode linkCode fileCode linkfineCode
%token errorCode

%%
```

```
START       : htmlCode HEAD BODY htmlfineCode
            ;

HEAD        : headCode titleCode STRINGA titlefineCode
              headfineCode
            ;

LINK        : linkCode fileCode {add2stack(yytext);}
              linkfineCode
            ;

BODY        : bodyCode BODYINT bodyfineCode
            ;

BODYINT     :
            | STRINGA BODYINT
            | LINK BODYINT
            ;


STRINGA     :
            | stringaCode STRINGA
            ;

%%

{$I PROGETTO.PAS}

begin
  if paramcount<>1 then
  begin
    writeln;
    writeln('Usage:  Prog nome.htm');
    halt(0);
  end;
  new(stack);
  stack:=nil;
  N:=1;
  tabsym[1]:=ParamStr(1);
  Push(ParamStr(1));
  while stack<>nil do
  begin
    current:=pop;
```

```
    assign(yyoutput, '');
    rewrite(yyoutput);
    assign(yyinput, current);
    reset(yyinput);
    if yyparse=0 then
    begin
      writeln(current,' riconosciuto!!');
    end
    else
    begin
      writeln(current,' non riconosciuto!!');
      halt(0);
    end;
  end;
end.
```