Generatori automatici di parser

Bison

Giovedì 29 Novembre

Generatori di parser

- Un generatore automatico di parser prende in input un file che specifica la sintassi di un certo linguaggio:
  - solitamente nella forma delle regole di una grammatica CF
  - …e includendo altre funzioni ausiliarie, definizioni di token, …
- …e produce in output un codice (scritto in un certo linguaggio) che implementa il ruolo del parser
- Erano chiamati compilatori di compilatori poichè tradizionalmente tutti gli step della compilazione venivano realizzate dal parser. Adesso il parser è considerato solo uno step del processo.
  - Yacc (Yet another compiler-compiler) è un generatore largamente usato che incorpora la tecnica del parsing LALR(1)
  - Una delle implementazioni più usate di Yacc, nonché di pubblico dominio è BISON

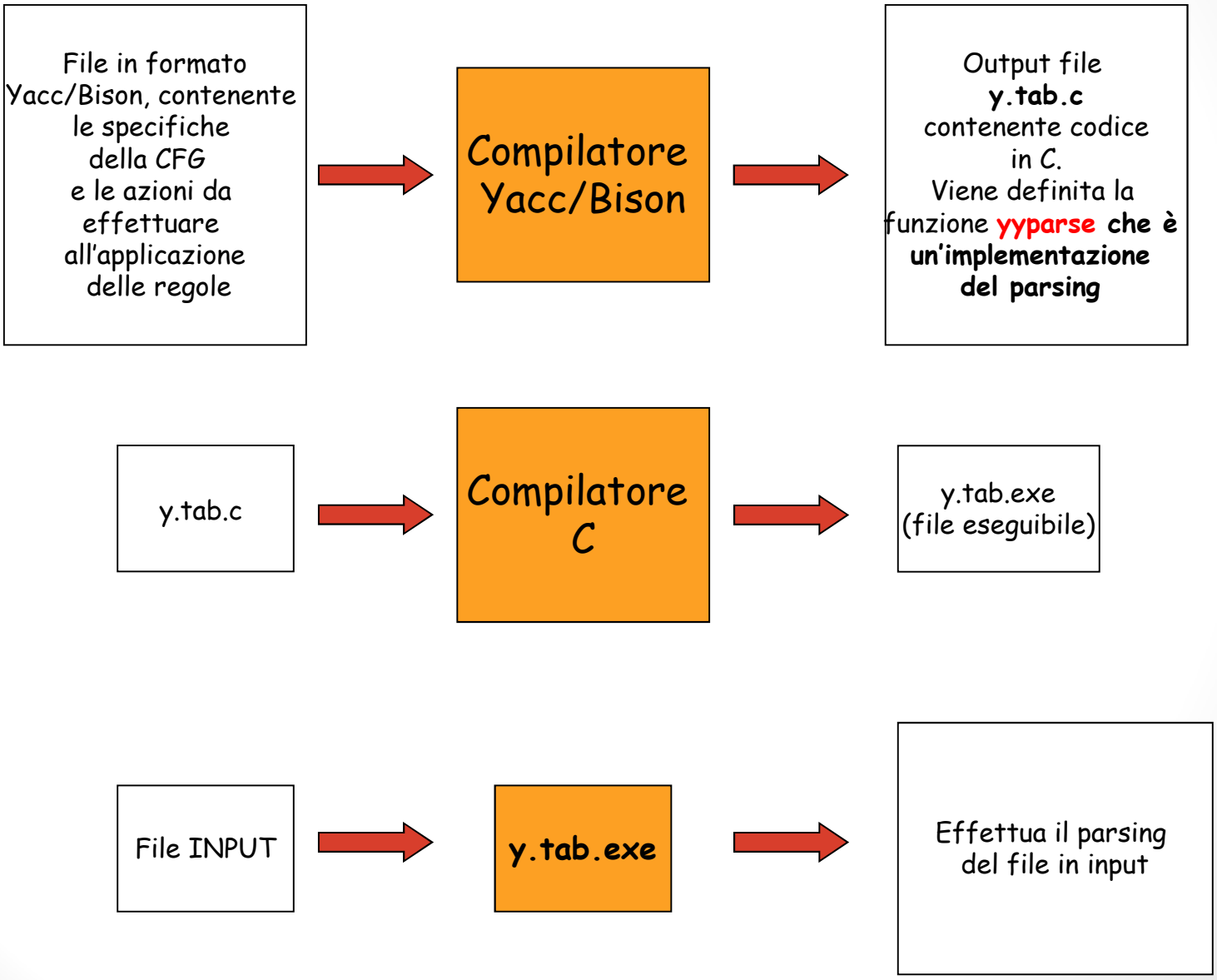
A quali linguaggi si rivolge Bison?

- Il linguaggio deve essere descritto mediante una grammatica context-free
- Bison è ottimizzato per grammatiche LALR(1)
- Con dichiarazioni opportune, Bison è anche capace di fare il parsing per una classe più estesa di grammatiche.

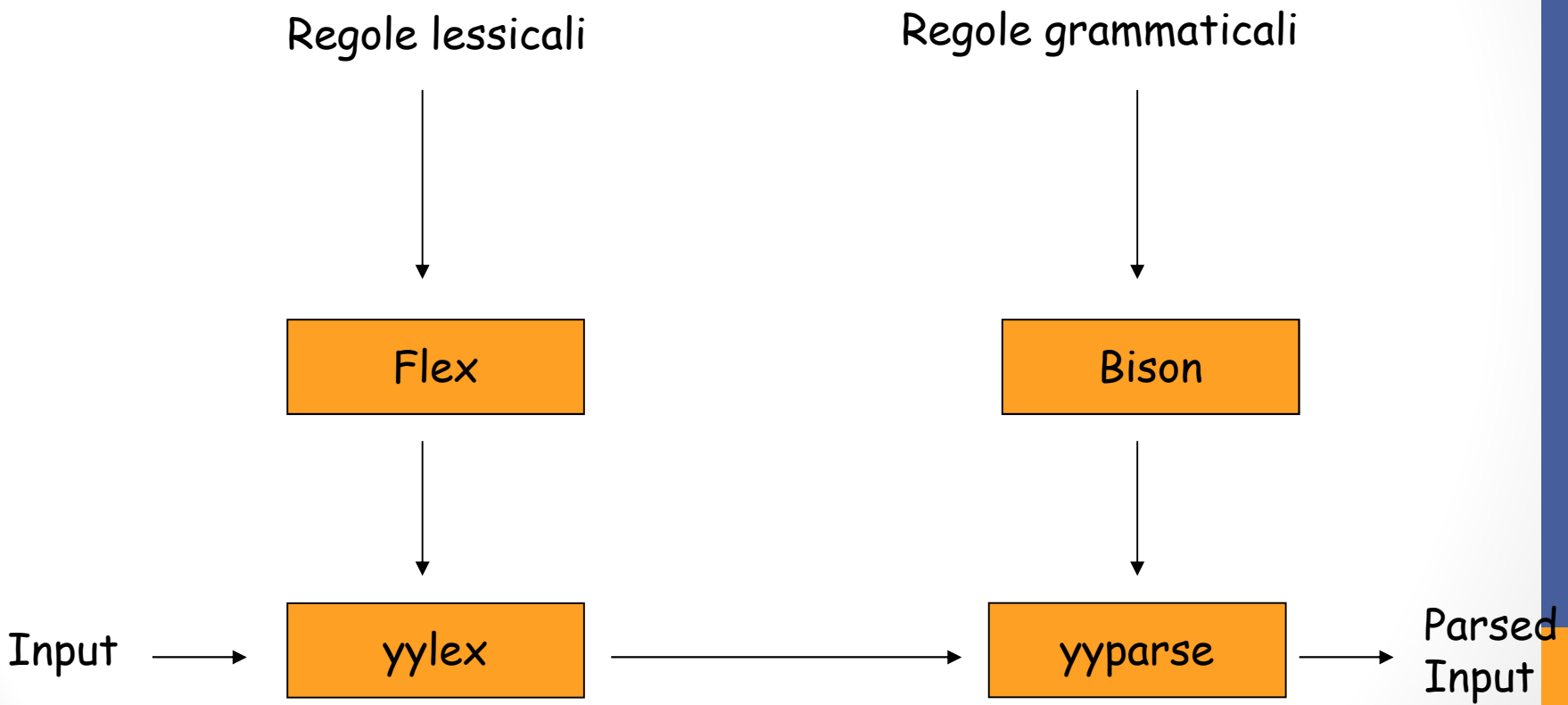
Grammatiche GLR

Se si usa l'opzione %glr-parser Bison produrrà un parser LR generalizzato che alla presenza di un conflitto shift-reduce o reduce-reduce, clonerà se stesso per seguire entrambe le possibilità

Uso di Yacc o Bison



Uso di Flex e Bison



Scrivere un programma in Bison

Un file in formato Bison consiste di 3 sezioni, separate da %%:

DEFINIZIONI
%%
REGOLE della GRAMMATICA
%%
FUNZIONI AUSILIARIE

La routine yyparse ha bisogno di altre funzioni:

1. l'analizzatore lessicale (yylex) che può essere scritto a mano o prodotto da Flex
2. La funzione che riporta gli errori (yyerror)
3. La funzione main che deve contenere la chiamata yyparse

Specifiche Bison

```
%{
#include <stdio.h>
int regs[26];
int base;
%}
%start espr
%token DIGIT LETTER
%left '|'
%left '&'
%left '+' '-'
%left '*' '/' '%'
%left UMINUS
%%
espr:    /*empty */
          |
         espr istr DIGIT
          |
         espr error LETTER
          {
            funzione();
          }
          ;
%%
main() { … }
```

Tra %{ e %} si trovano definizioni di tipi, variabili usate nelle azioni, direttive, prototipi di funzioni definite in seguito.
Nomi dei simboli terminali e non, regole di precedenza degli operatori, tipi di dato dei valori semantici

Regole della grammatica ed eventuali azioni

Routine ausiliarie

Azioni e valore semantico

- Ogni token potrebbe portare con se il corrispondente lessema (eventualmente memorizzato nella variabile yylval di tipo YYSTYPE)
- Il proposito delle azioni è spesso quello di calcolare il valore semantico di un certo costrutto

```
expr : expr '+' expr   {$$ = $1 + $3;}
```

Esercizio

Espressioni aritmetiche: scrivere un parser bottom-up (ed un analizzatore lessicale) per le seguenti grammatiche per espressioni aritmetiche:

E -> E + T
E -> E - T
E -> T
T -> T * F
T -> T / F
T -> F
F -> numero
F -> (E)

E-> E + E
E-> E-E
E-> E * E
E-> E/E
E-> numero
E-> (E)

dove numero è un intero senza segno;

Analizzatore lessicale – espr.fl

```
%{
#include <stdio.h>
#include "espr.tab.h"
%}
/* regular definitions
*/
delim [ \t\n]
ws    {delim}+
digit [0-9]
number      {digit}+
%option noyywrap
%%
{ws}        ;
{number} {return(NUM);}
\+    {return(PIU);}
\-    {return(MENO);}
\*    {return(PER); }
[/]   {return(DIVISO); }
\(    {return(PAR_AP); }
\)    {return(PAR_CH); }
%%
```

Analizzatore sintattico – espr.y

```
%{
#include <stdio.h>
%}

%token NUM PIU MENO PER DIVISO PAR_AP
PAR_CH

%start Expr
%error-verbose

%%
Expr: Expr PIU Term | Expr MENO Term | Term

Term: Term PER Factor | Term DIVISO Factor |
Factor

Factor: NUM | PAR_AP Expr PAR_CH

%%

Int main(void)
{
  yyparse();
  return 0;
}

yyerror (char *s)

{
  printf ("Errore Sintattico\n");
}
```

Esercizi

1. Scrivere un programma in Bison che valuti un'espressione in forma postfissa.

2. Scrivere un programma in Bison che converta in notazione postfissa, le espressioni in forma infissa corretta.

3. Scrivere un programma in Bison che converta in notazione infissa, le espressioni in forma postfissa corretta.

4. Scrivere un programma che testi se una parola è palindroma