Linguaggi per il Web

prof. Riccardo Rosati
corso di Laurea in Ingegneria Informatica
Sapienza Università di Roma, a.a. 2014/2015

Esercizi su analisi sintattica e JavaCC

Esercizio 1 Si consideri il frammento del linguaggio Java costituito dalle stringhe che corrispondono alla definizione di un singolo metodo `void` che accetta solo parametri di tipo `int` e, nelle istruzioni, solo dichiarazioni di variabili di tipo `int` e assegnazioni di variabili con espressioni uguali a singoli valori interi o a singole variabili.

Un esempio di stringa appartenente a questo linguaggio è il seguente:

```
void metodo1 (int x) {
  int y;
  y=1;
  x=y;
}
```

Altro esempio:

```
void metodo2 (int x, int p) {
  int y;
  int z;
  z = p;
  y = x;
  z = 12;
}
```

(1) Scrivere una grammatica non contestuale per tale linguaggio, dividendo la specifica del lessico del linguaggio (che va definita mediante espressioni regolari) dalla specifica della sintassi vera e propria; (2) scrivere una specifica JavaCC corrispondente alla specifica scritta al punto 1.

Soluzione Specifica del lessico:

IDENT : (“a” | . . . | “z” | “A” | . . . | “Z”) (“a” | . . . | “z” | “A” | . . . | “Z” | “0” | . . . | “9”)*
VALINT : (“+” | “-”)? (“0” | . . . | “9”)$^+$
TIPOVOID : “void”
TIPOINT : “int”
VIRG : “,”
PVIRG : “;”
TONDAAP : “(”
TONDACH : “)”
GRAFAP : “{”
GRAFCH : “}”

UGUALE : “=”
SEPARATORE : (“ ” | “\n” | “\r” | “\t”)$^+$

N.B. la classe SEPARATORE è una classe di token da non restituire (non va considerata nella specifica della grammatica).

Specifica della grammatica non contestuale:

start → TIPOVOID IDENT TONDAAP listapar TONDACH GRAFAP seqistr GRAFCH
listapar → ϵ | dichpar restolistapar
dichpar → TIPOINT IDENT
restolistapar → ϵ | VIRG dichpar restolistapar
seqistr → ϵ | istruzione seqistr
istruzione → dichiarazione | assegnazione
dichiarazione → TIPOINT IDENT PVIRG
assegnazione → IDENT UGUALE latodestroassegnazione PVIRG
latodestroassegnazione → IDENT | VALINT

Specifica JavaCC:

```
SKIP:
{ (" "|"\n"|"\r"|"\t") }

TOKEN:
{
   <TIPOVOID: "void">
 | <TIPOINT: "int">
 | <IDENT: ("a"|...|"z"|"A"|...|"Z")("a"|...|"z"|"A"|...|"Z"|"0"|...|"9")*>
 | <VALINT: ("+"|"-")?("0"|...|"9")+>
 | <VIRG: ",">
 | <PVIRG: ";">
 | <TONDAAP: "(">
 | <TONDACH: ")">
 | <GRAFAP: "{">
 | <GRAFCH: "}">
 | <UGUALE: "=">
}

void start() : {}
{ <TIPOVOID> <IDENT> <TONDAAP> listapar() <TONDACH> <GRAFAP> seqistr() <GRAFCH> }

void listapar() : {}
{ dichpar() restolistapar() | {} }

void dichpar() : {}
{ <TIPOINT> <IDENT> }

void restolistapar() : {}
{ <VIRG> dichpar() restolistapar() | {} }
```

```
void seqistr() : {}
{ istruzione() seqistr() | {}}

void istruzione() : {}
{ dichiarazione() | assegnazione() }

void dichiarazione() : {}
{ <TIPOINT> <IDENT> <PVIRG> }

void assegnazione() : {}
{ <IDENT> <UGUALE> latodestroassegnazione() <PVIRG> }

void latodestroassegnazione() : {}
{ <IDENT> | <VALINT> }
```

Esercizio 2 Estendere il linguaggio dell'esercizio precedente come segue:

1. estendere il lessico ammettendo la parola chiave `String` e le costanti di tipo `String`, cioè sequenze (anche vuote) di caratteri tra doppi apici;
2. estendere la sintassi ammettendo anche il tipo `String` nelle dichiarazioni di parametri e di variabile, nonché le costanti di tipo `String` nel lato destro delle istruzioni di assegnazione.

Un esempio di stringa appartenente a questo linguaggio è il seguente:

```
void metodo1 (int x, String s) {
  int y;
  String p;
  y=1;
  p="ciao";
}
```

Altro esempio:

```
void metodo2 (int x, String s, String p) {
  String y;
  String z;
  z = p;
  y = "ciao ciao";
  z = 324;
}
```

Soluzione Specifica del lessico:

IDENT : (“a” | . . . | “z” | “A” | . . . | “Z”) (“a” | . . . | “z” | “A” | . . . | “Z” | “0” | . . . | “9”)
VALINT : (“+” | “-”)? (“0” | . . . | “9”)$^+$
VALSTRING : \“ (“a” | . . . | “z” | “A” | . . . | “Z” | “0” | . . . | “9” | “-” | “ ” | “\n” | “\r” | “\t” | . . .)*\“
TIPOVOID : “void”
TIPOINT : “int”

TIPOSTRING : “String”
VIRG : “,”
PVIRG : “;”
TONDAAP : “(”
TONDACH : “)”
GRAFAP : “{”
GRAFCH : “}”
UGUALE : “=”
SEPARATORE : (“ ” | “\n” | “\r” | “\t”)$^+$

N.B. la classe SEPARATORE è una classe di token da non restituire (non va considerata nella specifica della grammatica).

Specifica della grammatica non contestuale:

start → TIPOVOID IDENT TONDAAP listapar TONDACH GRAFAP seqistr GRAFCH
listapar → ϵ | dichpar restolistapar
dichpar → TIPOINT IDENT | TIPOSTRING IDENT
restolistapar → ϵ | VIRG dichpar restolistapar
seqistr → ϵ | istruzione seqistr
istruzione → dichiarazione | assegnazione
dichiarazione → TIPOINT IDENT PVIRG | TIPOSTRING IDENT PVIRG
assegnazione → IDENT UGUALE latodestroassegnazione PVIRG
latodestroassegnazione → IDENT | VALINT | VALSTRING

Specifica JavaCC:

```
SKIP:
{ (" "|"\n"|"\r"|"\t") }

TOKEN:
{
   <TIPOVOID: "void">
 | <TIPOINT: "int">
 | <TIPOSTRING: "String">
 | <IDENT: ("a"|...|"z"|"A"|...|"Z")("a"|...|"z"|"A"|...|"Z"|"0"|...|"9")*>
 | <VALINT: ("+"|"-")?("0"|...|"9")+>
 | <VALSTRING: \"("a"|...|"z"|"A"|...|"Z"|"0"|...|"9"|" "|"\n"|"\r"|"\t"|"-"|...)*\">
 | <VIRG: ",">
 | <PVIRG: ";">
 | <TONDAAP: "(">
 | <TONDACH: ")">
 | <GRAFAP: "{">
 | <GRAFCH: "}">
 | <UGUALE: "=">
}

void start() : {}
{ <TIPOVOID> <IDENT> <TONDAAP> listapar() <TONDACH> <GRAFAP> seqistr() <GRAFCH> }
```

```
void listapar() : {}
{ dichpar() restolistapar() | {} }

void dichpar() : {}
{ (<TIPOINT> <IDENT>) | (<TIPOSTRING> <IDENT>) }

void restolistapar() : {}
{ <VIRG> dichpar() restolistapar() | {} }

void seqistr() : {}
{ istruzione() seqistr() | {}}

void istruzione() : {}
{ dichiarazione() | assegnazione() }

void dichiarazione() : {}
{ (<TIPOINT> <IDENT> <PVIRG>) | (<TIPOSTRING> <IDENT> <PVIRG>) }

void assegnazione() : {}
{ <IDENT> <UGUALE> latodestroassegnazione() <PVIRG> }

void latodestroassegnazione() : {}
{ <IDENT> | <VALINT> | <VALSTRING> }
```

Esercizio 3 Estendere il linguaggio dell'esercizio precedente come segue:

1. estendere il lessico ammettendo le parole chiave `if` e `else` e il simbolo `==`;
2. estendere la sintassi ammettendo anche l'istruzione `if-else` (cioè la presenza del ramo else è obbligatoria) con condizione dell'istruzione del tipo *variabile* `==` *variabile*.

Un esempio di stringa appartenente a questo linguaggio è il seguente:

```
void metodo1 (int x, String s) {
  int y;
  String p;
  y = 0;
  if (x==y)
    y=1;
  else p="ciao";
}
```

Altro esempio:

```
void metodo2 (int x, int z, String s) {
  int y;
  String p;
  y = 0;
  if (x==y) {
```

```
    y=1;
    if (z==x)
      p="ciao";
    else p = "ciao ciao";
  }
  else {
    p="ciao ciao ciao";
    y=123;
  }
}
```

Esercizio 4 Estendere il linguaggio dell'esercizio precedente ammettendo espressioni con l'operatore `+` nei lati destri delle assegnazioni, e la presenza dell'operatore booleano `&&` nelle condizioni dell'istruzione `if`.

Un esempio di stringa appartenente a questo linguaggio è il seguente:

```
void metodo1 (int x, String s) {
  int y;
  int z;
  String p;
  z = x;
  y = x + 10;
  if (x==y && z==1)
    y=1;
  else p="ciao";
}
```

Altro esempio:

```
void metodo2 (int x, int z, String s) {
  int y;
  String p;
  y = x + 10;
  if (x==y && z==1) {
    y = z + 1 + x;
    if (z==x)
      p="ciao";
    else p = "ciao ciao";
  }
  else {
    p="ciao ciao ciao";
    y=1+x+123;
  }
}
```

Esercizio 5 Estendere il linguaggio dell'esercizio precedente ammettendo la definizione di più metodi nella stessa stringa.

Un esempio di stringa appartenente a questo linguaggio è il seguente:

```
void metodo1 (int x, String s) {
  int y;
  int z;
  String p;
  z = x;
  y = x + 10;
  if (x==y && z==1)
    y=1;
  else p="ciao";
}
void metodo2 (int x, int z, String s) {
  int y;
  String p;
  y = x + 10;
  if (x==y && z==1) {
    y = z + 1 + x;
    if (z==x)
      p="ciao";
    else p = "ciao ciao";
  }
  else {
    p="ciao ciao ciao";
    y=1+x+123;
  }
}
```