Note

Analizzatore Lessicale
Parte I – Scanner

Dispensa di Linguaggi di Programmazione

Corrado Mencar, Pasquale Lops, Stefano Ferilli

Sommario

In questa dispensa si descrive un approccio alla costruzione di un analizzatore lessicale per il riconoscimento di token appartenenti ad un linguaggio di programmazione Pascal-like.

Introduzione

L'analisi lessicale è un passaggio fondamentale nel processo di compilazione di un programma. L'analisi lessicale ha il ruolo di scandire un programma sorgente, carattere per carattere, e di produrre una sequenza di simboli detti "token", che possono essere utilizzati dal *parser* per produrre opportune strutture dati necessarie al processo di traduzione.

Esempi di token in un linguaggio di programmazione sono:

- Gli identificatori (variabili, nomi di funzioni, etc.);
- Le parole chiave (p.e. **`begin`**, **`while`**, **`if`**, etc.);
- Le costanti numeriche e stringhe (p.e. `-2`, `0`, `3.2`, `'testo'` etc.);
- Gli operatori, le parentesi, i terminatori (p.e. `+`, `-` , `(` , `)` , `;` , etc.);

In un tipico linguaggio di programmazione, inoltre, è possibile inserire commenti al codice utili a migliorare la leggibilità del programma sorgente. In fase di analisi lessicale, tali commenti devono essere "saltati" perché non servono alle successive fasi di parsing e traduzione.

Specifica di un token

Ogni token appartiene ad una classe e possiede un valore. Per esempio, il token associato alla stringa di testo `-2.24` appartiene alla classe delle costanti reali, e possiede il valore -2.24. Se ad ogni classe si associa un codice univoco, allora un token può essere rappresentato da una coppia:

token = (*classe*, *valore*)

Per esempio, il token associato alla stringa di testo `-2.24` può essere definito dalla coppia:

token = (2, -2.24)

laddove si ipotizza che il numero 2 si riferisce alla classe delle costanti reali.

Note

Ruolo dell'analisi lessicale

Il ruolo dell'analizzatore lessicale è di trasformare un programma sorgente (definito da una sequenza di caratteri) in una sequenza di token. Per esempio, il programma Pascal:

```
01 Program mioprogramma;
02 Begin
03      Writeln ('ciao mondo');
04 End.
```

viene convertito in una sequenza di token, come ad esempio:

(0, "`Program`") → (1, "`mioprogramma`") → (5, "`;`") → (0, "`Begin`") →

(0, "`Writeln`") → (2, "`(`") → (3, "`ciao mondo`") → (4, "`)`") →

(5, "`;`") → (0, "`End`") → (5, "`.`")

laddove si è ipotizzato la seguente codifica delle classi:

0 – Parola chiave	1 – identificatore	2 – parentesi aperta
3 – costante stringa	4 – parentesi chiusa	5 – terminatore

I token estratti di solito non contengono il valore vero e proprio ma un indice ad una tabella dei simboli ("symbol table"). Per semplicità di esposizione, tuttavia, non si farà per ora riferimento a tale tabella.

La sequenza di token prodotta dall'analizzatore lessicale consente al parser di generare gli alberi sintattici relativi al programma, verificarne la correttezza sintattica e infine procedere al processo di traduzione nel linguaggio oggetto.

Analizzatore lessicale ed automi riconoscitori

La produzione di token a partire da un programma sorgente prevede il riconoscimento della classe di appartenenza. Questo compito è relativamente semplice poiché i token di ciascuna classe hanno valori che appartengono a linguaggi regolari. Per riconoscere un token di una *specifica* classe, pertanto, è sufficiente implementare un automa a stati finiti. Per riconoscere un token appartenente una *qualunque* classe, è necessario comporre più automi a stati finiti insieme. Il modello risultante, comunque, rimane ancora un automa a stati finiti.

Rispetto alla classica implementazione di un automa a stati finiti, è però necessario considerare alcune differenze sostanziali:

1. Un classico automa a stati finiti implementato in un linguaggio di programmazione prende la forma di una funzione che restituisce un valore logico (riconosciuto / non riconosciuto). L'automa a stati finiti per l'analisi lessicale deve invece restituire il codice della classe di appartenenza del token (oltre al valore). Mentre per il classico automa di riconoscimento la terminazione delle transizioni di stato in un qualunque stato finale comporta semplicemente il riconoscimento della stringa, per l'automa dell'analizzatore lessicale ogni stato finale determina una diversa classe di appartenenza del token.
2. Un classico automa a stati finiti riceve in input una stringa di caratteri che corrisponde ad una sola parola del linguaggio da riconoscere. Per terminare il riconoscimento è dunque sufficiente verificare se l'intera stringa è stata scandita. L'automa a stati finiti per l'analizzatore lessicale riceve invece un intero programma sorgente in input che corrisponde ad una *sequenza* di token da riconoscere. È necessario pertanto implementare un meccanismo in grado di verificare quando un token termina.

Note

Lookahead

Il precedente punto 2 solleva un problema particolarmente delicato che può essere meglio illustrato con un esempio. Si consideri la seguente linea di codice Pascal:

```
05 If x>=2 Then y:=3.1;
```

All'inizio lo scanner legge il carattere 'I'. Esso potrebbe essere un identificatore (per esempio nell'istruzione `I:=0`) oppure l'inizio di una parola chiave o di un identificatore più lungo. Esso pertanto legge il carattere successivo 'f'. La stringa finora letta è "If", che può corrispondere alla parola chiave **`If`** oppure all'inizio di un identificatore più lungo (per esempio, `Ifi`). Viene pertanto letto il carattere successivo, che corrisponde ad uno spazio. Lo spazio determina la fine del token, che pertanto viene riconosciuto come parola chiave.

Lo spazio viene saltato e si passa al carattere successivo 'x'. Ancora una volta, esso potrebbe essere un identificatore o l'inizio di un identificatore più lungo. Si guarda il carattere successivo '>', da cui si può dedurre che 'x' è un identificatore e viene quindi riconosciuto come token.

Il carattere '>' viene ripreso in considerazione. Esso potrebbe corrispondere all'operatore "maggiore" oppure all'inizio dell'operatore "maggiore o uguale". Viene pertanto letto il carattere successivo '='. Tale carattere consente di riconoscere il token ">=" classificato come

operatore logico. Si passa al carattere '2', che potrebbe corrispondere alla costante intera 2 oppure all'inizio di una costante più lunga. Viene pertanto letto il carattere successivo, che corrisponde ad uno spazio. Pertanto, la stringa "2" viene riconosciuta come token, di classe costante intera.

Lo spazio viene saltato e si passa al carattere successivo 'T'. Allo stesso modo della stringa "If" si procede a scandire i caratteri successivi fino a trovare lo spazio. La stringa letta è "Then" che viene riconosciuta come parola chiave. Allo stesso modo si riconosce la stringa "y" come identificatore. Si legge successivamente il carattere ":", che può corrispondere al separatore di etichetta (le *label* del Pascal) oppure all'inizio dell'operatore di assegnazione. La lettura del carattere successivo '=' consente di eliminare questa ambiguità, potendo riconoscere la stringa ':=' come token di classe operatore di assegnamento.

Lo scanner legge il carattere '3'. La lettura del carattere successivo '.' informa lo scanner che è in corso la lettura di una costante reale. Esso pertanto legge il carattere successivo '1'. Essendo una cifra, la include nella costante reale e legge il carattere successivo ';'. Poiché esso non è una cifra, lo scanner riconosce la stringa "3.1" come costante reale. Il carattere ';' viene riconsiderato e viene immediatamente riconosciuto come token di classe terminatore.

Come si è potuto osservare, per verificare se la stringa di caratteri letta dallo scanner fino ad un certo istante di tempo corrisponde ad un token intero oppure è solo una parte di esso, si legge il carattere successivo e si decide se esso fa parte del token corrente oppure segna l'inizio di un token successivo. Questo meccanismo di lettura anticipata di un carattere è detto **lookahead**. Il carattere letto in anticipo può essere consumato, se esso fa parte del token corrente o è uno spazio, oppure può essere riconsiderato nella procedura di riconoscimento se esso è parte di un token successivo. Il lookahead è un meccanismo semplice ma fondamentale per la corretta implementazione di una procedura di riconoscimento.

Implementazione di un analizzatore lessicale

In questa sezione si descrive brevemente l'implementazione di un analizzatore lessicale per un linguaggio Pascal-Like*.

Note

* Un linguaggio Pascal-Like ha una struttura simile a quella del Pascal, ma con alcune caratteristiche secondarie che lo distinguono dal Pascal vero e proprio. Nel caso presentato, il linguaggio ha la medesima struttura del Pascal, ma presenta alcune semplificazioni.

Strutture dati

Note

La struttura dati principale è quella che rappresenta un Token. Essa è costituita da una struttura di due campi, uno per la classe e l'altro per il valore:

```
typedef struct
{
 TokenType type;
 String value;
} Token;
```

Il tipo `TokenType` definisce le classi di un token. Esso può essere convenientemente definito come enumerazione di simboli:

```
typedef enum
{
 IDENTIFIER,
 KEYWORD,
 COMMENT,
 LEFT_PAR,
 RIGHT_PAR,
 PLUS,
 MINUS,
 ASTERIX,
 SLASH,
 EQUAL,
 SEMICOLON,
 DOT,
 COMMA,
 GREATER,
 GREATER_OR_EQUAL,
 LESS,
 LESS_OR_EQUAL,
 DIFFERENT,
 COLON,
 ASSIGNMENT,
 INTEGER,
 REAL,
 STRING,
 ERROR
} TokenType;
```

Si osservi l'inclusione anche di una classe speciale, ERROR, che individua eventuali errori lessicali. Il tipo `String` è stato definito per rappresentare una stringa di lunghezza prefissata:

```
38 typedef char String[MAX_STRING_LENGTH+1];
```

Note

Funzione getToken

La funzione `getToken` riceve in input un puntatore a file e valorizza una struttura `Token` con la classe e il valore del token correntemente letto dal file. La funzione restituisce anche un valore logico per indicare la fine del file.

La dichiarazione della funzione è la seguente:

```
39 int getToken (FILE* src, Token* tkn)
```

Chiamate successive alla funzione `getToken` consentono di scandire sequenzialmente il file e di restituire i vari token che costituiscono il programma sorgente. Per implementare il meccanismo del lookahead, è necessario che il carattere letto dal file sia conservato tra una chiamata e l'altra della funzione. Per evitare che la variabile locale che conserva il carattere letto dal file sia distrutta al termine della funzione, essa viene dichiarata nella funzione con lo specificatore **`static`**.

```
40 static char c;
```

Allo stesso modo, è necessario dichiarare una variabile logica che indichi se il carattere contenuto in `c` è stato consumato oppure deve essere ripreso in considerazione per il riconoscimento:

```
41 static int lookahead = 0;
```

L'assegnazione del valore 0 alla variabile avviene solo alla prima chiamata della funzione. Nelle successive chiamate la variabile conserverà il valore assegnatole nelle chiamate precedenti.

La funzione `getToken` segue lo stesso schema di un automa a stati finiti per il riconoscimento. Alcune varianti consentono l'implementazione del meccanismo di lookahead. Ad esempio, per rendere disponibile il carattere da analizzare, si usa il seguente brano di istruzioni:

```
42    if (!lookahead)
43       c = fgetc(src);
44    else
45       lookahead = 0;
```

Se infatti la variabile `lookahead` assume valore falso, è necessario leggere il carattere dal file `src`. Altrimenti, esso è già stato letto nella scansione del token precedente, quindi esso è già presente nella variabile `c`. In questo caso, è sufficiente azzerare il valore della variabile `lookahead`.

Note

Un brano di codice che illustra l'uso del lookahead è il seguente, che permette il riconoscimento di costanti numeriche:

```
46  …
47      case S:
48           … if (isdigit(c))
49           {
50                /* letta una cifra */
51                current_state = NUM;
52                /* passa allo stato NUM */
53                tkn->value[i++] = c;
54                tkn->type = INTEGER;
55           }
56  …
57      case NUM:
58           if (isdigit(c))
59                  /* letta un'altra cifra */
60              tkn->value[i++] = c;
61           else if (c == '.')
62           {
63                  /* letto il punto */
64                current_state = NUM_R;
65                  /* è una costante reale */
66                tkn->value[i++] = c;
67           }
68           else
69           {
70             /* il carattere letto non è né un
71                   Punto né una cifra. Ciò
72                   Significa che il carattere
73                   Letto fa parte di un altro
74                   Token. Quindi lookahead deve
75                   Assumere valore vero */
76                tkn->type = INTEGER;
77                eot = lookahead = !0;
78             /* eot è un flag che assume valore
79                   Vero se il token corrente è
80                   terminato*/
81           }
82           break;
83      case NUM_R:
84           if (isdigit(c))
85              tkn->value[i++] = c;
```

```
86              else
87              {
88                  tkn->type = REAL;
89                  eot = lookahead = !0;
90              }
91              break;
92  ...
```

Note

Esercizi proposti

1. Analizzare e discutere lo schema di riconoscimento delle varie classi di token del linguaggio di riferimento (fare riferimento al codice sorgente)
2. Estendere l'analizzatore lessicale con token del Pascal non inclusi nel caso di studio. Esempi di estensione sono:
    a. I commenti delimitati da {}
    b. Le costanti reali in notazione esponenziale
    c. Le costanti carattere
    d. Le costanti numeriche esadecimali
3. Costruire un analizzatore lessicale per il riconoscimento di token di un linguaggio C-like.