Note

# Verifica che una grammatica sia Lineare Destra nel GrammaReader

*Dispensa di Linguaggi di Programmazione*

Corrado Mencar

## Sommario

*In questa dispensa si descrive una soluzione per verificare che una grammatica memorizzata mediante il programma GrammarReader sia lineare destra.*

*Il materiale contenuto nella dispensa è aderente a quello richiesto per lo svolgimento di una traccia d'esame scritto*

## Traccia

Siano definiti i seguenti tipi di dati in linguaggio C:

```
typedef char Symbol;

typedef struct
{
        Symbol word [100];
        unsigned length;
} Word;

typedef struct
{
        Word left;
        Word right;
} Production;

typedef struct
{
        Production productions[100];
        unsigned numprod;
} Grammar;
```

Assumendo che i simboli terminali del linguaggio siano rappresentati da caratteri alfabetici minuscoli e i simboli non terminali da caratteri alfabetici maiuscoli, realizzare una funzione in linguaggio C per controllare se la grammatica caricata all'interno di `Grammar` è **lineare destra** (si assuma che la grammatica sia già stata caricata in precedenza).

Note

Dare una breve descrizione della strategia solutiva.

## Soluzione

Al fine di verificare che una grammatica sia lineare destra (LD), è necessario controllare che tutte le produzioni che la costituiscono sono del tipo:

$$v \rightarrow w$$

dove $v$ è una parola costituita da un unico simbolo non-terminale, mentre $w$ verifica uno dei seguenti casi:

- È vuota ($\lambda$-produzione)
- È costituita da un unico simbolo terminale
- È costituita da due simboli: il primo terminale e il secondo non-terminale

La strategia risolutiva deve dunque prevedere una scansione sequenziale delle produzioni costituenti una grammatica, al fine di verificare su ciascuna di esse le proprietà precedentemente descritte. Se una produzione non verifica tale proprietà, la scansione può arrestarsi subito, poiché sicuramente la grammatica analizzata non è LD.

Si può pertanto delineare la seguente funzione, scritta nel linguaggio C, che fa uso delle strutture dati ipotizzate:

```
int is_rightlinear(Grammar* g)
/*
  INPUT: Puntatore ad una grammatica caricata in
     memoria
  OUTPUT: 0  - la grammatica non è Lineare Destra
          !0 - la grammatica è Lineare destra.
*/
{
 int i=0;       /* indice della produzione */
 int ld = !0;   /* flag: se 0 la gramm. non è LD */
 Production* p; /* puntatore alla produzione da
                   esaminare */
 Word* l;
 Word* r;

 /* Verifica che g sia un puntatore valido */
 if (g == NULL)
 {
     printf ("Errore: grammatica non valida\n");
     return 0;
 }
```

Note

```
 while (ld && i < g->numprod)
 {

  p = &g->productions[i];
  l = &p->left;
  r = &p->right;


  if (l->length > 1 || !is_nonterminal(l->word[0]))
     ld = 0;
  else switch (r->length)
  {
   case 0: ld = !0; break;
   case 1: ld = is_terminal(r->word[0]); break;
   case 2: ld = is_terminal(r->word[0]) &&
               is_nonterminal(r->word[1]); break;
   default: ld = 0;
  }

  i++; /* Passa alla produzione successiva */
 }

 return ld;
}
```

La funzione `is_rightlinear` necessita delle funzioni `is_nonterminal` e `is_terminal` che verificano, rispettivamente, se un simbolo è non-terminale e terminale. Per ipotesi un simbolo è non-terminale se è un carattere ASCII maiuscolo, mentre è terminale se è un carattere ASCII minuscolo. Pertanto, le due funzioni possono essere così definite:

```
int is_nonterminal (Symbol s)
/*
     INPUT: un simbolo
     OUTPUT:   0 – il simbolo non è non-
                    terminale
               !0 – il simbolo è un non-
                    terminale
*/
{
    return (s >= 'A') && (s <= 'Z');
}
int is_terminal (Symbol s)
```

```
/*
     INPUT: un simbolo
     OUTPUT:   0 – il simbolo non è terminale
               !0 – il simbolo è un terminale
*/
{
    return (s >= 'a') && (s <= 'z');
}
```

Note