Note

Verifica che una grammatica sia Context Free nel GrammaReader

Dispensa di Linguaggi di Programmazione

Corrado Mencar – Pasquale Lops – Stefano Ferilli

Sommario

In questa dispensa si descrivono alcune soluzioni per verificare che una grammatica memorizzata mediante il programma GrammarReader sia non contestuale.

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:

```
01 typedef char Symbol;
02
03 typedef struct
04 {
05         Symbol word [100];
06         unsigned length;
07 } Word;
08
09 typedef struct
10 {
11         Word left;
12         Word right;
13 } Production;
14
15 typedef struct
16 {
17         Production productions[100];
18         unsigned numprod;
19 } 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 libera da contesto (si assuma che la grammatica sia già stata caricata in precedenza).

Note

Dare una breve descrizione della strategia solutiva.

Prima Soluzione

Al fine di verificare che una grammatica sia libera da contesto (CF), 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, e questo sia non-terminale.
La strategia risolutiva deve dunque prevedere una scansione sequenziale delle produzioni costituenti una grammatica, al fine di verificare su ciascuna di esse la proprietà precedentemente descritta. Se una produzione non verifica tale proprietà, la scansione può arrestarsi subito, poiché sicuramente la grammatica analizzata non è CF. Si può pertanto delineare la seguente funzione, scritta nel linguaggio C, che fa uso delle strutture dati ipotizzate:

```
int is_contextfree(Grammar* g)
/*
  INPUT:   Puntatore ad una grammatica caricata in memoria
  OUTPUT:  0  - la grammatica non è C.F.
          !0 - la grammatica è C.F.
*/
{
 int i=0;       /* indice della produzione */
 int cf = !0;   /* flag: se 0 la grammatica non è CF */
 Production* p; /* puntatore alla produzione da esaminare */
 Word* l;       /* puntatore alla testa della produzione da
                   esaminare */

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


 /*   Scandisci   sequenzialmente   le   produzioni   della
grammatica, e verifica se ciascuna di essa è context-free.
Appena si scandisce una produzione non C.F. la scansione
termina.  Ciò  evita  di  scandire  tutta  la  grammatica,
ottenendo una maggiore efficienza del programma */

 while (cf && i < g->numprod)
 {
     /* Considera la testa della i-esima produzione */

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

     /*  Verifica  che  la  testa  della  produzione  sia
costituita da un unico simbolo non-terminale e non abbia
simboli terminali */

     /* Se la lunghezza della testa è >1, sicuramente la
produzione non è C.F. Se il primo simbolo della testa non è
non-terminale, la produzione non è C.F. */
      if (l->length > 1 || !is_nonterminal(l->word[0]))
          cf = 0;

     i++; /* Passa alla produzione successiva */
 }
  return cf;
}
```

La funzione is_contextfree necessita della funzione is_nonterminal, che verifica se un simbolo è non-terminale. Per ipotesi un simbolo è non-terminale se è un carattere ASCII maiuscolo, pertanto la funzione is_nonterminal può essere così definita:

Note

```
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');
}
```

Note

Seconda Soluzione

```
int is_contextfreeGrammar(Grammar* g)
/*
  INPUT:   Puntatore ad una grammatica caricata in memoria
  OUTPUT: 0  - la grammatica non è C.F.
          1  - la grammatica è C.F.
*/
{
 int i=0;       /* indice della produzione */

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


/* Scandisci sequenzialmente le produzioni della grammatica,
e verifica se ciascuna di essa è context-free. */

 for (i=0; i < g->numprod, i++)
     if (!is_contextfreeProduction(&g->productions[i]))
        return 0;
 return 1;
}

int is_contextfreeProduction(Production* p)
/*
  INPUT:   Puntatore ad una produzione
  OUTPUT: 0  - la produzione non è C.F.
          1 - la produzione è C.F.
*/
{
  if (p->left.length==1 && isupper(p->left.word[0]))
      return 1;
  else return 0;
}
```

Terza Soluzione

Note

Dalla teoria si apprende che una grammatica è libera da contesto (CF) se e solo se ogni produzione che la costituisce contiene un unico simbolo, che deve essere non-terminale, nella sua parte sinistra.

La strategia risolutiva può quindi basarsi su una scansione sequenziale delle produzioni costituenti la grammatica, che verifichi su ciascuna di esse tale proprietà. Se una produzione non verifica tale proprietà, la scansione può arrestarsi subito, poiché sicuramente la grammatica analizzata non è CF. Al termine del ciclo, basterà controllare se il contatore (che sarà pari al numero di produzioni CF) corrisponde al numero totale di produzioni, restituendo "vero" in tal caso o "falso" altrimenti.

Relativamente alla realizzazione in linguaggio C, che faccia uso delle strutture dati ipotizzate, la traccia precisa che è richiesta una funzione che svolga il compito descritto, per cui va esclusa la codifica della strategia sopra delineata direttamente nel main.

```
int checkCF(Grammar *g) {
int i=0;

while ( i < g->numprod && g->productions[i].left.length==1
        && is_nonterminal(g->productions[i].left[0]))
      i++;
return ( i == g->numprod );
}

int is_nonterminal (Symbol s) {
    return isupper(s);
}
```

Note di progettazione/realizzazione.

La grammatica è passata per riferimento, sebbene la funzione debba solo effettuare un controllo su di essa, contravvenendo dunque alle normali regole di buona programmazione al fine di renderne più efficiente l'esecuzione in termini di memoria e di tempo. Il passaggio per valore richiederebbe infatti di copiare interamente la grammatica in una variabile locale alla funzione, e nella peggiore delle ipotesi la grammatica potrebbe anche assumere le considerevoli dimensioni di 100 (numero massimo produzioni) x 200 (numero massimo di simboli nelle parti destra e sinistra di ciascuna produzione) = 20000 Byte, escludendo i byte necessari a memorizzare i 201 interi che rappresentano le lunghezze di ciascuna parte sinistra e destra e della grammatica nel suo complesso.

L'indice di scansione `i` parte da 0, che corrisponde alla prima posizione nell'array delle produzioni, e viene incrementato per ogni produzione riconosciuta come libera da contesto. Di conseguenza, al termine di

ogni iterata contiene il numero di produzioni libere da contesto effettivamente riscontrate fino a quel momento, il che garantisce la terminazione (prima o poi si uscirà dal ciclo per aver raggiunto il numero di produzioni memorizzate) e la correttezza del confronto finale nell'istruzione `return`. Da notare che, proprio per l'indicizzazione degli array in C, la condizione di uscita prevede il confronto di disuguaglianza stretto fra l'indice (che sarà sempre sfasato di uno in meno rispetto al numero di produzione che si sta controllando) ed il numero delle produzioni.

Note

L'introduzione di una funzione `is_nonterminal` che controlla se un simbolo è non terminale rende più generale la funzione e meno soggetta a cambiamenti nelle specifiche delle strutture destinate a contenere la grammatica e/o le sue componenti.

Si sarebbe potuta definire una funzione `int is_CF_prod(Production p)` incaricata di controllare se una data produzione è libera da contesto, il che si sarebbe ridotto a sua volta a controllare se la sua parte sinistra contiene un solo elemento, a sua volta non terminale:

```
p[i].left.length==1 && is_nonterminal(p[i].left[0])
```

ma non lo si è ritenuto necessario, in quanto la funzione di controllo che l'intera grammatica sia libera da contesto incapsula (e dunque conosce) già la strategia da seguire. Ovviamente, se ci fossero state (o si prevedesse che ci saranno in futuro) altre funzionalità del programma nel suo complesso per le quali fosse utile sapere se una produzione è libera da contesto, l'introduzione di tale funzione aggiuntiva sarebbe stata la scelta più opportuna.

L'uso della funzione di libreria C `int isupper(char)` richiede l'inclusione della corrispondente libreria tramite una direttiva `#include<ctype.h>`. Alternativamente, sfruttando le proprietà della codifica ASCII, si poteva controllare direttamente che il carattere fosse maiuscolo controllando che fosse compreso fra `'A'` e `'Z'`.