Implementazione di DFA in C

Dispensa di Laboratorio di Linguaggi di Programmazione

Corrado Mencar, Pasquale Lops

Sommario

Questa dispensa fornisce le linee guida per l'implementazione, nel linguaggio C, di Automi a Stati Finiti Deterministici (DFA) per il riconoscimento di linguaggi regolari.

Introduzione

Gli automi a stati finiti deterministici (DFA) sono macchine computazionali in grado di riconoscere tutti e soli i linguaggi regolari. Tali linguaggi sono spesso utilizzati all'interno di linguaggi più complessi (come i linguaggi di programmazione) per definire i costrutti di base, detti anche *token*. Ad esempio, gli identificatori, le costanti numeriche (intere con o senza segno, a singola o doppia precisione, nella rappresentazione decimale, ottale, etc.) sono token che appartengono a linguaggi regolari. Il riconoscimento di parole di tali linguaggi è dunque fondamentale per lo sviluppo di compilatori o interpreti.

I DFA sono macchine *astratte*, nel senso che sono definite da costrutti formali indipendenti dal tipo di calcolatore che le esegue. Di conseguenza, è necessario codificare un DFA in un opportuno programma per poter essere eseguito su una macchina di riferimento. Di seguito si descrive la metodologia per codificare un DFA in una procedura nel linguaggio C.

Esempio guida

Per meglio comprendere la metodologia di codifica, si riporta un automa di esempio che verrà successivamente tradotto in una procedura opportuna.

L'automa di esempio consente di riconoscere costanti numeriche rappresentate in notazione scientifica normalizzata alla prima cifra, come ad esempio 0, 1.33E+2, -2.21E-21, etc*.

Rappresentazione dell'automa

Il diagramma delle transizioni dell'automa (che illustra anche l'insieme degli stati) è il seguente:

* In questo contesto, un numero in notazione scientifica è lo zero, oppure è un numero costituito da una cifra intera (eventualmente preceduto da un segno), da un separatore decimale, da una sequenza di cifre per la parte decimale, dal simbolo di esponente ('e' o 'E'), e dall'esponente, inteso come una qualunque sequenza di cifre, eventualmente preceduta dal segno.

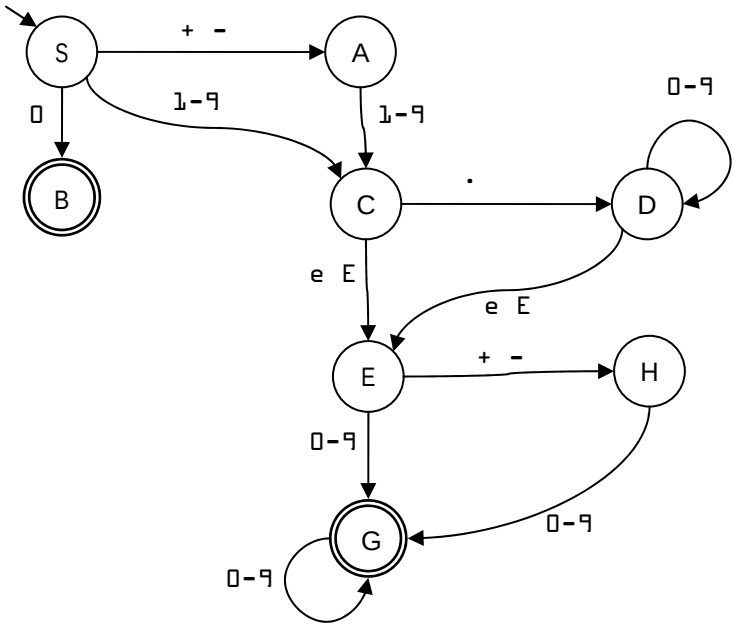

Figura 1: Diagramma di transizione del DFA per il riconoscimento di numeri in notazione scientifica

Gli stati finali dell'automa sono stati denotati con un doppio cerchio ◎. La definizione completa del DFA d'esempio su un alfabeto X è la seguente:

$$M = (Q, d, S, F)$$

dove:

- L'alfabeto di ingresso è $X = \{0,1,2,3,4,5,6,7,8,9,e,E,.,+,-\}$
- L'insieme degli stati è $Q = \{A,B,C,D,E,G,H,S\}$
- Lo stato iniziale è S
- L'insieme degli stati finali è $F = \{B,G\}$

– La funzione di transizione è definita dal seguente schema tabulare:

d	0	1-9	e, E	.	+, -
A		C			
B					
C			E	D	
D	D	D	E		
E	G	G			H
G	G	G			
H	G	G			
S	B	C			A

Implementazione dell'automa

Il modello astratto di DFA prevede che la funzione *d* possa essere *definita parzialmente*, nel senso che per alcune configurazioni ingresso/stato, essa non sia definita (per esempio, la configurazione $\langle 0, B \rangle$). Per convenzione si assume che, nel caso si presentino tali configurazioni, l'esecuzione dell'automa cessa e la stringa in input non viene riconosciuta. In effetti, la stringa è riconosciuta solo quando l'esecuzione cessa su uno stato finale (laddove la terminazione dell'esecuzione è dovuta alla fine della stringa da esaminare).

Quando un DFA deve essere implementato su un calcolatore, è necessario gestire esplicitamente le configurazioni ingresso/stato che non danno luogo ad alcuna transizione di stato. La soluzione più semplice e generale consiste nella definizione di uno stato speciale, detto *stato pozza*, a cui l'automa transita quando si presenta una configurazione non valida. Il programma di simulazione dell'automa verifica continuamente lo stato dell'automa e, se questo è lo stato pozza, la scansione della stringa termina con un messaggio di errore.

Pseudocodice

La simulazione dell'automa può seguire il seguente flusso, descritto in pseudo-codice:

1. Stato Corrente = S
2. Fino a quando non termina la scansione della stringa e lo Stato Corrente non è pozza,
   2.1. Determina il nuovo stato sulla base della funzione di transizione (aumentata con lo stato pozza)
3. Al termine del ciclo, se lo Stato Corrente è uno stato finale, allora restituisci RICONOSCIUTO, altrimenti restituisci NON RICONOSCIUTO

Implementazione della funzione di transizione

Per determinare lo stato risultante dall'applicazione della funzione di transizione, è necessario rappresentare quest'ultima all'interno della procedura di riconoscimento. Esistono diverse possibilità in proposito, tra cui:

- **La costruzione di una matrice di transizione**. Ad ogni riga della matrice si fa corrispondere uno stato, mentre ad ogni colonna un simbolo di ingresso. Ogni cella contiene uno stato, compreso lo stato pozza. Ad ogni coppia ingresso/stato, questa viene tradotta negli indici di riga e colonna e viene letto il contenuto della cella corrispondente. Tale contenuto è assegnato allo Stato Corrente. Questa soluzione è generale e semplice da implementare, ma può determinare un notevole dispendio di risorse di memoria.
- **La definizione di un flusso di controllo basato su istruzioni IF-THEN annidate**. Questa soluzione, lievemente più complessa, consente di ridurre lo spazio di memoria necessario alla codifica della funzione di transizione. Al primo livello esistono tanti IF quanti sono gli stati dell'automa (escluso lo stato pozza). All'interno di ciascuna istruzione IF esisteranno tante IF di secondo livello quanti sono i simboli in ingresso. Nel corpo di una IF di secondo livello si entrerà solo per una specifica configurazione stato/ingresso. All'interno di tale corpo sarà contenuta l'istruzione per il cambio di stato. Sempre all'interno del corpo, è possibile aggiungere anche ulteriori istruzioni che verranno eseguite solo quando una determinata coppia stato/ingresso viene presentata. In questo modo, questa soluzione risulta più generale rispetto a quella definita dalla matrice di transizione;

La seconda soluzione può essere resa più leggibile se si sostituiscono gli IF di primo livello da un'istruzione di selezione multipla (**`switch`**). Ciò è possibile perché gli stati possono essere definiti da costanti numeriche o, meglio, enumerative.

Implementazione di DFA in C

A questo punto è possibile definire lo schema di una procedura per il riconoscimento di stringhe di simboli mediante implementazione del corrispondente DFA riconoscitore.

Sia dato un automa:

$$M = (Q, d, S, F)$$

definito su un alfabeto X sottoinsieme dei caratteri ASCII. Si definisce una funzione con la seguente intestazione:

```
01 int scan(char* s)
```

All'interno della funzione, si dichiara una variabile di tipo enumerativo che rappresenta lo stato corrente. Se $Q = \{q_0, q_1, ...q_n\}$ e $S = q_0$, allora la seguente variabile può essere definita:

```
02 enum {q0,q1,…,qn,pozza} current_state = q0;
```

È opportuno dichiarare anche un indice per la scansione della stringa in input:

```
03 int i=0;
```

A questo punto può partire il ciclo di scansione:

```
04 while (s[i]!='\0' && s[i]!='\n' && current_state!=pozza)
```

All'interno del ciclo sarà contenuta un'istruzione di *selezione multipla* che consente di eseguire il giusto brano di codice in corrispondenza dello stato corrente:

```
05 switch(current_state)
```

All'interno della **switch** vi saranno tante istruzione **case** quanti sono gli stati, ad esclusione dello stato pozza perché questo serve solo nel controllo della condizione del ciclo **while**. Un esempio di blocco **case** è il seguente:

```
06 case qk:
07      if (s[i] == primo_simbolo)
08           current_state = nuovo stato;
09      else if (s[i] == secondo simbolo)
10           current_state = nuovo stato;
11      else if …
12      else current_state = pozza;
13      break;
```

Al termine del ciclo **while** è vera una delle seguenti due condizioni:

- È terminata la stringa da esaminare;
- Lo stato è quello pozza.

È possibile che la stringa da esaminare sia terminata quando lo stato corrente è diverso da uno stato finale. Conseguentemente, per restituire il risultato corretto della scansione è necessario verificare che lo stato corrente sia uno stato finale valido. Se $q_{i1}, q_{i2}, ..., q_{ik}$ sono stati finali, la seguente istruzione di selezione consente di restituire il risultato della scansione:

```
14 if (current_state == qi1 || current_state == qi2 || … ||
   current_state == qik)
15      return !0;
16 else
17      return 0;
```

oppure, più semplicemente:

```
18 return (current_state == qi1 || current_state == qi2 || …
   || current_state == qik);
```

Uso della funzione scan

La funzione `scan` viene utilizzata all'interno di un'altra procedura che ha il compito di fornire le stringhe da riconoscere. Questa procedura potrebbe essere la funzione `main` del programma. Un'utile funzionalità è quella di leggere da un file di testo delle righe, una per volta, e di fornire in input le righe lette alla procedura di scansione.

Per realizzare questo processo, è opportuno definire nella funzione `main` una variabile di tipo stringa, che costituirà il *buffer* delle stringhe da scandire. L'accesso al file avviene mediante ciclo **`while`**, che termina non appena si raggiunge lo stato di EOF. All'interno del ciclo, si legge dal file una riga alla volta, utilizzando la funzione `fgets`. La funzione `fgets` restituisce il puntatore `buffer` se la lettura è andata a buon fine, altrimenti restituisce NULL. Questa possibilità può essere sfruttata per garantire che la funzione `scan` possa essere richiamata sempre con una stringa valida da riconoscere.

Quando il buffer viene riempito con una riga, viene passato alla funzione `scan` per il riconoscimento. Lo stato del riconoscimento può essere visualizzato con opportuni messaggi a video. Segue il corpo del ciclo **`while`**:

```
19 while (!feof(file))
20  {
21   if (fgets(buffer,MAX_LENGTH,file))
22      if (scan(buffer))
23          printf("%s RICONOSCIUTO\n", buffer);
24      else
25       printf("%s NON RICONOSCIUTO\n", buffer);
26  }
```

Esercizi proposti

1) Sviluppare il programma C interamente funzionante per la simulazione dell'automa descritto nell'esempio guida;
2) Creare un file di testo contenente diverse stringhe (alcune corrette, altre no) per effettuare il test del programma;
3) Simulare altri automi di riconoscimento, sviluppando programmi con lo stesso impianto descritto in questa dispensa.
4) Modificare la procedura `scan` affinché restituisca 0 se la stringa è stata riconosciuta, 1 se non è stata riconosciuta perché le transizioni sono terminate su uno stato non finale, 2 se non è stata riconosciuta perché è stato letto un simbolo esterno all'alfabeto, 3 se non è stata riconosciuta perché è stata tentata una transizione non definita (transizione a stato pozza) anche se il simbolo letto appartiene all'alfabeto.