Grammatiche LR

Marco Moschettini

22 aprile 2015

Capitolo 1

Riconoscitori LR(0)

1.1 Introduzione

1.2 Calcolo dei contesti: procedura operativa

Procedura passo-passo per definire l'ASF caratteristico dell'oracolo di un riconoscitore LR(0).
Si adotteranno le seguenti conventzioni

1.2.1 Convenzioni

- Ogni frase lecita è espressamente terminata dal terminatore **$**
- Data una forma di frase, il cursore **.** denota il **confine** tra la parte già analizzata (sinistra) e la parte ancora da analizzare (destra)

1.2.2 Procedimento

Prendiamo, ad esempio, la grammatica:

$Z \to S\$$
$S \to aSAB \mid BA$
$A \to aA \mid B$
$B \to b$

All'inizio l'input è ancora tutto da leggere e ogni frase lecita deriva dallo scopo Z; quindi la situazione è schematizzabile con

$Z \to .S\$$

Poichè a destra del cursore c'è S, le frasi possono iniziare solo con produzioni di S. Ergo, per descrivere completamente questo stato occorre aggiungere **tutte le produzioni di S**:

$Z \to .S\$$
$S \to .aSAB \mid .BA$

Inoltre, poichè una di queste produzioni può iniziare per B, anche **tutte le produzioni di B**:

$B \to .b$

A questo punto il primo stato dell'automa caratteristico, che rappresenta tutte le situazioni iniziali è descritto da questo stato (lo chiameremo **stato 1**)

Stato 1

Cominciamo dallo stato iniziale:

$Z \to .S\$$
$S \to .aSAB \mid .BA$
$B \to .b$

Ora dobbiamo *investigare* le possibili evoluzioni di questo stato, ossia *spostare* il cursore a destra di una posizione **in tutti i modi possibili**:

- con input S: $Z \to S.\$$ (**stato F**)
- con input a: $S \to a.SAB$ (**stato 2**)
- con input B: $S \to B.A$ (**stato 4**)
- con input b: $B \to b.$ (**stato R1**)

dove con F intendiamo lo stato **finale** del nostro riconoscitore, con gli stati numerici degli stati in cui effettuare una **shift** con gli stati R^* intendiamo gli stati in cui effettuare una **reduce**.

Stato 2

A questo punto lo stato 2 ha una produzione che può iniziare per S. È, dunque, necessario considerare tutte le produzioni relative ad S; Poichè tra queste ultime una può iniziare per B, va aggiunta anche la produzione di B.

$S \to a.SAB$
$S \to .aSAB \mid .BA$
$B \to .b$

Da qui dobbiamo provare tutte le possibili combinazioni di input:

- con input S: $S \to aS.AB$ (**stato 3**)
- con input a: $S \to a.SAB$ (**stato 2**)
- con input B: $S \to B.A$ (**stato 4**)
- con input b: $B \to b.$ (**stato R1**)

Stato 4

Nello stato 4 compare una A e quindi dobbiamo considerare tutte le possibili produzioni di A:

$S \to B.A$
$A \to .aA \mid .B$
$B \to .b$

Stesso ragionamento per il calcolo degli spostamenti:

- con input A: $S \to BA.$ (**stato R3**)
- con input a: $A \to a.B$ (**stato 6**)
- con input B: $A \to B.$ (**stato R2**)
- con input b: $B \to b.$ (**stato R1**)

Stato 3

In questo caso compare una A e quindi, come sopra, dobbiamo considerare tutte le produzioni di A:

$S \to aS.AB$
$A \to .aA \mid .B$
$B \to .b$

Spostamenti:

- con input A: $S \to aSA.B$ (**stato 5**)
- con input a: $A \to a.A$ (**stato 6**)
- con input B: $A \to B.$ (**stato R2**)
- con input b: $B \to b.$ (**stato R1**)

Stato 6

In questo caso compare una A e quindi, ancora una volta, dobbiamo considerare tutte le produzioni di A.

$A \to a.A$
$A \to .aA \mid .B$
$B \to .b$

Spostamenti:

- con input A: $A \to aA.$ (**stato R4**)
- con input a: $A \to a.A$ (**stato 6**)
- con input B: $A \to B.$ (**stato R2**)
- con input b: $B \to b.$ (**stato R1**)

Stato 5

Eccoci giunti all'ultimo stato da rappresentare. In questo caso compare solo una B e quindi dobbiamo considerare tutte le produzioni di B:

$S \to aSA.B$
$B \to .b$

Spostamenti:

- con input B: $S \to aSAB.$ (**stato R5**)
- con input b: $B \to b.$ (**stato R1**)

1.2.3 Fine procedimento

Arrivati a questo punto possiamo notare che l'ASF generato è deterministico, non presenta ambiguità e quindi è possibile disegnarlo ed implementare la corrispondente parsing table per l'**oracolo**. In particolare, preso l'ASF risultante:

- ogni arco corrisponde ad un'operazione **shift**.
- ogni stato finale corrisponde ad un'operazione **reduce**.

1.2.4 Tabella di parsing

	a	b	$	S	A	B
1	s2	s11		g10		g4
2	s2	s11		g3		g4
3	s6	s11			g5	g12
4	s6	s11			g13	g12
5		s11				g15
6	s6				g14	g4
10			a			
11	r1	r1	r1			
12	r2	r2	r2			
13	r3	r3	r3			
14	r4	r4	r4			
15	r5	r5	r5			

Con:

- 10 = stato F
- 11 = stato R1
- 12 = stato R2
- 13 = stato R3
- 14 = stato R4
- 15 = stato R5

e con:

- $B \to b$ (riduzione 1)
- $A \to B$ (riduzione 2)
- $S \to BA$ (riduzione 3)
- $A \to aA$ (riduzione 4)
- $S \to aSAB$ (riduzione 5)

1.2.5 Osservazioni

Nel caso si osservino dei problemi di ambiguità durante la stesura dell'ASF e il grafico corrispondente non risultasse deterministico, **il linguaggio non sarà LR(0)**.

Capitolo 2

Riconoscitori LR1(1)

2.1 Procedura operativa

La procedura operativa per un linguaggio LR(1) è molto simile a quella mostrata per LR(0). L'unica aggiunta degna di nota è quella dei cosiddetti **Look Ahead symbols** che rappresentano previsioni per il prossimo carattere letto da input. Prendiamo, ad esempio, la grammatica:

$Z \to .S\$ \quad (?)$
$S \to .aSAB \mid .BA \quad (a, =)$
$B \to .b \quad (\$)$

I simboli tra parentesi mostrano i possibili prossimi input per la singola produzione. Come si calcola il **lookahead set**?

2.1.1 Calcolo del lookahead set

Si procede come segue:

- si guarda quali caratteri possono seguire il simbolo terminale VN nella produzione di livello superiore a quella trattata.
- si concatena tali caratteri con il lookahead set attuale
- si prende l'iniziale della stringa (perchè stiamo considerando LR(1)... si estende a k stringhe se considerassimo LR(k)).

Facciamo un esempio:

$Z \to .S\$ \quad (?)$
$S \to .CbBa \quad (\$)$
$C \to .a \quad (...)$

Nota: $(?)$ sta per *anything* e va inserito al livello dello scopo. Calcoliamo ora il lookahead set per la produzione $(C \to .a)$:

- si guardano i caratteri che possono **seguire C nella produzione di livello superiore a quella usata attualmente** (in questo caso $(S \to .CbBa)$
- **si concatena il risultato** con il lookahead attuale (**$**) ottenendo (**b$**)
- si tiene solo il **primo carattere** (**b**)

Si prosegue in questo modo fino a giungere all'ASF.

2.1.2 Osservazioni

Possiamo notare che questo metodo è molto più complesso del caso LR(0) perchè è necessario, ad ogni passaggio, ricomputare l'insieme dei lookahead symbols svolgendo in pratica passo per passo le stesse elaborazioni che portano alla costruzione dei contesti LR(1).

Tuttavia, seppur aumentando la complessità, notiamo che con questo metodo è possibile risolvere molte ambiguità legate alla scelta di produzioni in LR(0). In particolare nel caso di più strade possibili con uno stesso simbolo in ingresso, sarà sufficiente analizzare i lookahead symbols per scegliere una strada piuttosto che un'altra.

Capitolo 3

Riconoscitori SLR

L'esperienza conferma che è praticamente impossibile costruire un parser deterministico LR(1) per un "vero" linguaggio di programmazione, poiché le sue dimensioni sarebbero intrattabili. Di conseguenza si utilizzano versioni semplificate di tale riconoscitore. In particolare SLR allarga leggermente la dimensione dei contesti di LR(0) riducendo aumentandone la portata ma anche esponendo i contesti a maggiore sovrapposizione (ricordiamo che nel caso di sovrapposizione dei contesti l'ASF presenterò ambiguità).

3.1 Procedura operativa

Un modo molto semplice per generare l'ASF di un riconoscitore SLR è quello di partire dal relativo riconoscitore LR(0) togliendo alcune parti della parsing table a seconda di alcune regole. Più precisamente: una riduzione del tipo $(A \to \alpha)$ va inserita nella tabella di parsing **solo se** il prossimo simbolo appartiene a **FOLLOW(A)**. Se si ottiene un automa **senza conflitti**, allora la grammatica è SLR(1). Altrimenti, no.

Capitolo 4

Riconoscitori LALR(1)

E se l'approccio SLR non funziona? Un approccio alternativo per ridurre le dimensioni della parsing table consiste nel **collassare in un solo stato gli stati identici a meno dei simboli di Look-Ahead**. Si parla allora di parser LALR(1).

4.1 Procedura operativa

La procedura prevede di generare completamente l'ASF del parser LR(1) salvo poi collassare gli stati identici (in cui cambiano solo i lookahead sets). Per verificarne la correttezza bisogna garantire la totale assenza di conflitti.

4.2 Osservazioni

Esistono casi in cui un parser LALR(1) presenta gli stessi stati di un possibile parser SLR che presenta dei conflitti.

Capitolo 5

Riepilogo

Riepilogando abbiamo la seguente gerarchia di parser (ordinato per complessità crescente):

1. LR(0)
2. SLR
3. LALR(1)
4. LR(1)
5. LR(k)

5.1 Ulteriori considerazioni

- Ogni grammatica SLR(k) è anche LR(k)
- Ogni grammatica LL(k) aumentata della produzione ($Z \rightarrow S\$$) è anche LR(k)
- Esistono grammatiche LR(1) ma non SLR(k)
- Esistono grammatiche LL(1) ma non SLR(1)