Dispensa 2

2.1 Costruzione Parsing Table LR: generalità

Come tutti i parser tabellari predittivi, anche i parser LR possono essere applicati solo a parsing table senza conflitti (ossia entrate multiple) per assicurarne il determinismo. Nel parsing LR un *conflitto* è definito come un' entrata multipla del tipo shift/reduce oppure reduce/reduce nella parte action della parsing table. Più precisamente:

- *conflitto shift/reduce*: significa che nella stessa entrata sono presenti contemporaneamente una azione di shift e una o più azioni di reduce
- *conflitto reduce/reduce*: significa che nella stessa entrata sono presenti contemporaneamente due o più azioni reduce

Data una grammatica context free esistono tre metodi diversi per costruirne la parsing table per un parser LR:

- *tecnica SLR*
- *tecnica LALR*
- *tecnica LR canonica*

Ciascuna tecnica dà luogo a una famiglia di grammatiche così definite:

Definizione (grammatica SLR):
Una grammatica context free si dice SLR se la parsing table costruita con la tecnica SLR è priva di conflitti.

Definizione (grammatica LALR):
Una grammatica context free si dice LALR se la parsing table costruita con la tecnica LALR è priva di conflitti.

Definizione (grammatica LR canonica):
Una grammatica context free si dice LR canonica se la parsing table costruita con la tecnica LR canonica è priva di conflitti.

In sostanza queste grammatiche sono quelle a cui il parser può essere applicato con successo deterministicamente. Un importante risultato formale che caratterizza queste famiglie di grammatiche è la cosiddetta *gerarchia di grammatiche LR*, che stabilisce che esse sono propriamente incluse l'una nell'altra: la famiglia più grande è quella LR canonica che include quella LALR che a sua volta include quella SLR che è la più piccola.
Pertanto, a livello di potere riconoscitivo, la tecnica LR canonica è quella più potente, poi via via le altre due. Ciò, ad esempio, significa che:

- una grammatica che è LR canonica può non essere LALR (e né SLR)
- una grammatica che è LALR è sicuramente LR canonica ma può non essere SLR
- una grammatica che è SLR è sicuramente LR canonica e LALR

Al di fuori della famiglia più grande LR canonica ci sono le grammatiche context free a cui il parsing LR non può essere applicato, per esempio le grammatiche ambigue. Ricordiamo infatti che la tecnica di parsing LR è comunque una tecnica di parsing non universale.

Oltre che in potere riconoscitivo, le tre tecniche di costruzione parsing table variano anche in termini di dimensione della tabella prodotta. In questo caso i vantaggi sono contrari, nel senso che le tecniche SLR e LALR producono di norma parsing table più piccole rispetto alla tecnica LR canonica applicata alla stessa grammatica. Ad esempio, si pensi che per la grammatica del Pascal le tabelle SLR e LALR hanno un numero di stati dell'ordine delle centinaia, mentre quella LR canonica dell'ordine delle migliaia.
In definitiva, il miglior compromesso fra potere riconoscitivo e taglia della tabella prodotta è offerto dalla tecnica LALR che risulta infatti quella più usata nella pratica (fra l'altro, il tool YACC è proprio un generatore automatico di parser LALR).

Iniziamo ora lo studio della costruzione di parsing table, cominciando dalla tecnica SLR che pur essendo la più debole e poco usata in pratica, è la tecnica più semplice ai fini didattici e su cui ci soffermeremo di più. Essa permette infatti di introdurre in modo semplice e chiaro una serie di concetti che saranno poi facilmente estendibili alle altre due tecniche più importanti LALR e LR canonica.

2.2 La tecnica SLR

Introduciamo anzitutto le nozioni di *grammatica aumentata* e *LR(0) item*:

Definizione (grammatica aumentata):
Data una grammatica context free G con simbolo iniziale S, G' è la grammatica aumentata di G con un nuovo simbolo iniziale S' e una nuova produzione $S' \rightarrow S$

In altre parole, se G = (N, T, S, P) allora G' = (N' = N $\cup${S'}, T, S', P $\cup${$S' \rightarrow S$ })

E' evidente che G e G' generano lo stesso linguaggio, infatti sono sostanzialmente la stessa grammatica con la differenza che G' aggiunge sempre il passo di derivazione iniziale $S' \Rightarrow S$.
Lo scopo è quello di indicare in modo univoco al parser quando fermarsi con successo nel suo processo di riconoscimento, cosa che avverrà quando il parser riduce la produzione iniziale $S' \rightarrow S$.
Pertanto, da ora in poi, il primo passo nella costruzione di parsing table è sempre quello di passare dalla grammatica sorgente alla sua grammatica aumentata.

Definizione (LR(0) item):
Un LR(0) item è una produzione con un punto da qualche parte nel lato destro

Ad esempio, per la produzione $A \rightarrow X\ Y\ Z$ si hanno quattro possibili LR(0) items:

$A \rightarrow .\ X\ Y\ Z$
$A \rightarrow X\ .\ Y\ Z$
$A \rightarrow X\ Y\ .\ Z$
$A \rightarrow X\ Y\ Z\ .$

Sostanzialmente, il punto indica quanto di una produzione (ciò che sta prima del punto) è stato visto a un dato momento del parsing.
Ad esempio, nel caso $A \rightarrow X\ .\ Y\ Z$ significa che il parser nel suo processo di riconoscimento ha già esaminato X (e cioè proprio il token X se X è un terminale, oppure tutto il contesto X se X è un non-terminale) e si appresta a vedere Y (cioè proprio il token Y se Y è un terminale, oppure ciò che

è derivabile da Y se Y è un non-terminale); oppure nel caso $A \rightarrow X\ Y\ Z\ .$ significa che il parser ha esaminato tutto il contesto A e si appresta a ridurre la produzione.

Raggruppando opportunamente LR(0) items è possibile formare gli stati dell'automa che riconosce il linguaggio generato dalla grammatica aumentata, pertanto l'obiettivo ora è quello di costruire una collezione di insiemi di LR(0) items $I_0\ I_1 \ldots I_n$ che formano proprio gli stati del parser.
Per costruire questi insiemi di items la tecnica SLR fa uso di due funzioni *closure* e *goto* così definite.

Funzione *closure*

La funzione *closure* si applica a un insieme di LR(0) items e produce ancora un insieme di LR(0) items. In termini operativi essa è composta dalle seguenti due regole:

closure(I)
1) ogni LR(0) item di I è inserito in closure(I)
2) per ogni LR(0) item $A \rightarrow \alpha\ .\ B\ \beta$ in closure(I)
   e per ogni produzione $B \rightarrow \gamma$ in G
   aggiungi $B \rightarrow .\ \gamma$ in closure(I)

Sostanzialmente, la regola 1) copia I in closure(I), poi la regola 2) aggiunge altri LR(0) items in closure(I) finchè non è possibile aggiungere più nulla. Se in un LR(0) item il punto sta alla fine o prima di un terminale allora la regola 2) non fa nulla, altrimenti se il punto sta prima di un non-terminale la regola 2) aggiunge tutte le produzioni con a sinistra quel non-terminale e col punto all'inizio. In altre parole, la closure "scava" dentro i non-terminali con l'obiettivo di arrivare a ciò che è derivabile da essi.

Esempio
Sia la grammatica aumentata:

$E' \rightarrow E$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (\ E\ )$
$F \rightarrow$ **id**

Se si considera $I = \{\ E' \rightarrow .\ E\ \}$ allora
closure(I) = {
$E' \rightarrow .\ E$
$E \rightarrow .\ E + T$
$E \rightarrow .\ T$
$T \rightarrow .\ T * F$
$T \rightarrow .\ F$

$F \rightarrow .\ (\ E\ )$
$F \rightarrow .\ \mathbf{id}$ }

Pertanto in generale dopo aver scaricato I in closure(I) la domanda da porci nell'applicazione iterativa della closure è

"c'è un non-terminale dopo il punto?"

Se la risposta è NO allora ci si ferma, se è SI allora si continua "scavando" dentro quel non-terminale. Infatti il parser si aspetta come prossimi simboli tutto ciò che è derivabile da quel non-terminale.

Funzione *goto*

La funzione *goto* si applica a un insieme di LR(0) items e ad un simbolo grammaticale, e produce un insieme di LR(0) items. In termini dichiarativi essa è così definita:

goto(I, x) è la closure dell'insieme di tutti gli LR(0) items $A \rightarrow \alpha\ x\ .\ \beta$ tali che
$A \rightarrow \alpha\ .\ x\ \beta$ è in I

Sostanzialmente, quando si applica la goto(I, x) si vanno a vedere in I gli LR(0) items che hanno un punto prima del simbolo x, si sposta il punto a destra di x e si applica la closure.

Esempio
Per la grammatica aumentata dell'esempio precedente, se si considera
I = { $E' \rightarrow\ E\ .$
$E \rightarrow E\ .\ +\ T$ }
allora goto(I, +) = {
$E \rightarrow E + .\ T$
$T \rightarrow .\ T * F$
$T \rightarrow .\ F$
$F \rightarrow .\ (\ E\ )$
$F \rightarrow .\ \mathbf{id}$ }

Nota: L'applicazione di una goto(I, x) = I' si può rappresentare graficamente da una transizione su un grafo dal nodo I al nodo I' attraverso l'arco etichettato x.

A questo punto, il passo fondamentale della tecnica di costruzione parsing table SLR consiste nel costruire la collezione di insiemi di LR(0) items (corrispondente agli stati del parser) che si ottiene applicando iterativamente la funzione goto, e le successive closure, all'insieme di partenza:

I_0 = { closure(S’ → . S) }

La funzione goto va applicata per ogni simbolo grammaticale possibile, ossia per ogni simbolo che nel corrente insieme di items ha un punto subito prima di esso. In tal modo, a partire dall’insieme iniziale I_0 in cui si fa la closure dell’item iniziale S’ → . S si costruiscono proprio gli stati I_0 I_1 … I_n della parsing table SLR.

Esercizio 1 (estratto da traccia d’esame):

Costruire la collezione di insiemi di LR(0) items per la grammatica context free con produzioni:

$S \rightarrow a\ B\ c$
$B \rightarrow b\ d$
$B \rightarrow b$

Soluzione:
Come prima cosa si considera la grammatica aumentata:

S’ → S
S → a B c
B → b d
B → b

A partire da questa grammatica aumentata, si applica iterativamente la funzione goto (e le successive closure) innescandola sull’insieme di partenza I_0 = { closure(S’ → . S) }. In tal modo si ottiene:

I_0 = { closure(S’ → . S) } = {S’ → . S
S → . a B c }
I_1 = goto(I_0, S) = { S’ → S . }
I_2 = goto(I_0, a) = { S → a . B c
B → . b d
B → . b }
I_3 = goto(I_2, B) = { S → a B . c }
I_4 = goto(I_3, c) = { S → a B c . }
I_5 = goto(I_2, b) = { B → b . d
B → b . }
I_6 = goto(I_5, d) = { B → b d . }

Nota: se ogni insieme di LR(0) items I_i si rappresenta come un nodo di un grafo e ogni applicazione della goto I_k = goto(I_i, x) si rappresenta come una transizione sul grafo dal nodo I_i al nodo I_k con

arco labellato x, il grafo ottenuto si chiama *goto grafo SLR* e rappresenta il diagramma stati transizione dell'automa che sta dietro il parser.

Esercizio 2 (estratto da traccia d'esame):

Costruire la collezione di insiemi di LR(0) items per la grammatica context free con produzioni:

$S \rightarrow A\ b$
$A \rightarrow A\ a$
$A \rightarrow a$

Soluzione:
Anzitutto si costruisce la grammatica aumentata:

S' → S
S → A b
A → A a
A → a

A partire da questa grammatica aumentata, si applica iterativamente la funzione goto (e le successive closure) innescandola sull'insieme di partenza I_0 = { closure(S' → . S) }. In tal modo si ottiene:

I_0 = { closure(S' → . S) } = {S' → . S
S → . A b
A → . A a
A → . a }
I_1 = goto(I_0, S) = { S' → S . }
I_2 = goto(I_0, A) = { S → A . b
A → A . a }
I_3 = goto(I_0, a) = { A → a .}
I_4 = goto(I_2, b) = { S → A b . }
I_5 = goto(I_2, a) = { A → A a . }

2.3 Algoritmo di costruzione parsing table SLR

Data una grammatica context free, il passo fondamentale per la costruzione della sua parsing table SLR sta nella costruzione della sua collezione di insiemi di LR(0) items (o equivalentemente nella costruzione del goto grafo SLR se ne si considera la rappresentazione grafica). Questi items contengono infatti tutta l'informazione necessaria al parser per fare le sue mosse e procedere nel riconoscimento della stringa input.

Tuttavia per codificare questa informazione nel particolare formato richiesto dalle entrate della parsing table vanno applicati dei semplici passaggi di trasformazione. Diamo di seguito le linee generali dell'algoritmo di codifica:

G context free $\rightarrow$ Parsing Table SLR

Algoritmo

L'algoritmo costruisce la grammatica aumentata di G e crea la sua parsing table SLR applicando i seguenti passi:

1) Anzitutto si costruisce la collezione di insiemi di LR(0) items. Gli insiemi di items $I_0\ I_1 \ldots I_n$ (cioè i nodi del goto grafo) corrispondono agli stati 0, 1, … n della parsing table. Lo stato iniziale 0 corrisponde all'insieme I_0 = { closure(S' $\rightarrow$. S) }

2) Per ogni transizione etichettata con un simbolo terminale *a*:

I_k = goto(I_i, *a*)

si inserisce l'azione $shift_k$ nell'entrata ACTION[i, *a*] della parsing table

3) Per ogni transizione etichettata con un simbolo non-terminale A:

I_k = goto(I_i, A)

si inserisce l'azione $shift_k$ nell'entrata GOTO[i, A] della parsing table

4) Per ogni insieme I_i che ha al suo interno un LR(0) item col punto alla fine A $\rightarrow \alpha$. si inserisce l'azione $reduce_{A\rightarrow\alpha}$ nell' entrata ACTION[i, *a*] della parsing table sotto tutti i terminali *a* appartenenti a Follow(A)

5) Si considera l'unico insieme I_i che contiene l'item S' $\rightarrow$ S. e si inserisce "ACCETTA" nell'entrata ACTION[i, $] della parsing table

6) Tutte le entrate della parsing table rimaste vuote si settano a "ERRORE"

Nota: Se dopo il passo 4) la parsing table in costruzione contiene almeno un conflitto, ossia una entrata multipla del tipo shift/reduce oppure reduce/reduce nella parte action, allora il parsing fallisce e la grammatica input G è detta essere non SLR.