

# Dispensa 1

## 1.1 Costruzione Parsing Table: 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

(EFG)

Data una grammatica context free esistono tre metodi diversi per costruirne la parsing table per un parser LR. Queste tre tecniche, che studieremo, sono le seguenti:

- *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.

Pertanto queste grammatiche sono quelle a cui il parser può essere applicato con successo deterministico. 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 che studieremo 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.

## 1.2 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 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.

*Prima ESSERE INIZIALE poi MIGLIAIA.*

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  $I_0 I_1 \dots I_n$  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 in 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)* finché 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

$$\text{closure}(I) = \{$$

$$E' \rightarrow . E$$

$$E \rightarrow . E + T$$

$$E \rightarrow . T$$

$$T \rightarrow . T * F$$

$$T \rightarrow . F$$

$$F \rightarrow . ( E )$$

$$F \rightarrow . id \quad \}$$