

Tabelle SLR

Costruzione di tabelle di Parsing SLR

Metodo simple LR

- Metodo di costruzione di una tabella di parsing LR
- Abbreviazione: SLR
- Parser SLR: parser LR che usa una tabella di parsing SLR
- Grammatica SLR: grammatica per cui esiste un parser SLR

Potenza del metodo

- È il metodo di costruzione di tabelle LR più semplice
- È il metodo meno potente: ha successo per meno grammatiche rispetto agli altri due che vedremo
- È il metodo più semplice da implementare

Item LR(0)

- Un item LR(0) per una grammatica G è una produzione di G con un **punto** in qualche posizione della parte destra della produzione
- Es: la produzione $A \rightarrow XYZ$ può dar luogo a quattro item LR(0):
 1. $A \rightarrow \cdot XYZ$
 2. $A \rightarrow X \cdot YZ$
 3. $A \rightarrow XY \cdot Z$
 4. $A \rightarrow XYZ \cdot$

Item LR(0)

- La produzione $A \rightarrow \varepsilon$ dà luogo ad un solo item LR(0), cioè $A \rightarrow \cdot$
- Un item può essere rappresentato efficacemente con due numeri interi: il numero della produzione e la posizione del punto
- Un item indica quanto di una produzione è stato visto ad un certo punto del parsing
- Es: $A \rightarrow X \cdot YZ$ indica che X è stato visto e ci si aspetta di vedere YZ

Idea Centrale del metodo SLR

- Gli item sono gli stati di un NFA che riconosce i viable prefixes
- Insiemi di item sono gli stati del DFA che si ottiene da questo NFA con la costruzione dei sottoinsiemi
- Una collezione di insiemi di item LR(0) costituisce la base per costruire un parser SLR

Collezione canonica LR(0)

- G grammatica con simbolo iniziale S
- G' grammatica G aumentata con un nuovo simbolo iniziale S' e la produzione $S' \rightarrow S$
- Questo accorgimento serve ad indicare al parser la fine del parsing: il parsing ha successo se e solo se l'input è terminato e c'è una riduzione con la produzione $S' \rightarrow S$

Operazione di *closure*

- I insieme di item
- $closure(I)$ è un insieme di item costruito da I come segue:
 1. Inizialmente poni $closure(I)$ uguale a I
 2. Se $A \rightarrow \alpha \cdot B\beta$ è in $closure(I)$ e $B \rightarrow \gamma$ è una produzione, allora aggiungi $B \rightarrow \cdot \gamma$ a $closure(I)$, se non è già presente. Applica questa regola fino a quando nessun altro item può essere aggiunto a $closure(I)$

Operazione di *closure*

- Intuizione: la presenza di $A \rightarrow \alpha \cdot B\beta$ in $closure(I)$ indica che, ad un certo punto del processo di parsing, ci aspettiamo di vedere nell'input una stringa derivabile da $B\beta$
- Se $B \rightarrow \beta$ è una produzione, allora è possibile che ci sia una stringa derivabile da β , a questo punto dell'input
- Per questa ragione aggiungiamo anche l'item $B \rightarrow \cdot \beta$ a $closure(I)$

Esempio *closure*

- $E' \rightarrow E$
- $E \rightarrow E + T \mid T$
- $T \rightarrow T * F \mid F$
- $F \rightarrow (E) \mid id$
- $I = \{E' \rightarrow \cdot E\}$
- $closure(I) = \{E' \rightarrow \cdot E\} \cup \{E \rightarrow \cdot E + T, E \rightarrow \cdot T\} \cup \{T \rightarrow \cdot T * F, T \rightarrow \cdot F\} \cup \{F \rightarrow \cdot (E), F \rightarrow \cdot id\}$

Kernel item

- Si noti che se un item $B \rightarrow \cdot \gamma$ viene inserito in $closure(I)$, allora anche tutte le produzioni di B vengono inserite con il punto nella posizione più a sinistra
- In effetti basterebbe indicare solo B convenendo che rappresenta tutte le sue produzioni con il punto nella posizione più a sinistra

Kernel item

- Kernel item: $S' \rightarrow \cdot S$ oppure un item con il punto non nella posizione più a sinistra
- Non-kernel item: tutti gli altri (quelli che hanno il punto nella posizione più a sinistra)
- Ogni insieme di item può essere generato con una *closure* a partire da un certo insieme di kernel item
- Questa proprietà è utile per minimizzare lo spazio necessario per memorizzare gli insiemi di item

Operazione goto

- I insieme di item
- X simbolo della grammatica
- $goto(I, X) = closure(\{ A \rightarrow \alpha X \beta \mid A \rightarrow \alpha \bullet X \beta \in I \})$
- Se I è un insieme di item validi per un qualche viable prefix γ , allora $goto(I, X)$ è il set di item validi per il viable prefix γX

Esempio goto

- $I = \{ E' \rightarrow \bullet E, E \rightarrow E \bullet + T \}$
- $goto(I, +) = closure(\{ E \rightarrow E \bullet + T \}) = \{ E \rightarrow E \bullet + T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id \}$

Collezione canonica LR(0)

```
procedure items(G');
begin
  C := { closure({S' → •S}) }
  repeat
    for each insieme di item I in C e simbolo
      X tale che goto(I, X) non è vuoto do
      aggiungi goto(I, X) a C
  until non possono essere aggiunti nuovi
  insiemi di item a C
end;
```

Esempio

- Costruiamo la collezione canonica LR(0) per la solita grammatica
- $E' \rightarrow E$
- $E \rightarrow E + T \mid T$
- $T \rightarrow T * F \mid F$
- $F \rightarrow (E) \mid id$

Esempio

- Il primo insieme di item che inseriamo in C è $I_0 = closure(\{ E' \rightarrow \bullet E \}) = \{ E' \rightarrow \bullet E, E \rightarrow \bullet E + T, E \rightarrow \bullet T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id \}$
- Iniziamo il ciclo. Vediamo subito che $goto(I_0, E)$ non è vuoto perché in I_0 ci sono i due item $E' \rightarrow \bullet E$ e $E \rightarrow \bullet E + T$

Esempio

- $goto(I_0, E) = closure(\{ E' \rightarrow E \bullet, E \rightarrow E \bullet + T \}) = \{ E' \rightarrow E \bullet, E \rightarrow E \bullet + T \}$
- Chiamiamo I_1 questo nuovo insieme ed aggiungiamolo a C

Esempio

- Continuiamo con $goto(I_0, T) = closure(\{E \rightarrow T \cdot, T \rightarrow T \cdot * F\}) = \{E \rightarrow T \cdot, T \rightarrow T \cdot * F\}$
- Anche questo insieme non è presente in C e quindi lo inseriamo con il nome I_2
- $goto(I_0, F) = closure(\{T \rightarrow F \cdot\}) = \{T \rightarrow F \cdot\} = I_3$

Esempio (continuando)

- $goto(I_0, '(') = closure(\{F \rightarrow (\cdot E)\}) = \{F \rightarrow (\cdot E), E \rightarrow \cdot E + T, E \rightarrow \cdot T, T \rightarrow \cdot T * F, T \rightarrow \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot id\} = I_4$
- Questa volta l'operazione di chiusura ha rigenerato tutti gli item di I_0 dall'item $F \rightarrow (\cdot E)$

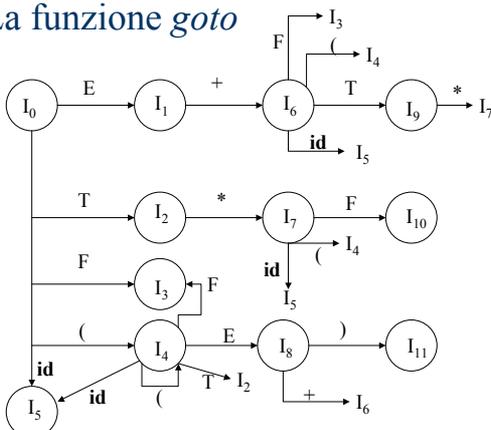
Esempio (continuando)

- $I_5 = \{F \rightarrow id \cdot\}$
- $I_6 = \{E \rightarrow E + \cdot T, T \rightarrow \cdot T * F, T \rightarrow \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot id\}$
- $I_7 = \{T \rightarrow T * \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot id\}$
- $I_8 = \{E \rightarrow E \cdot + T, F \rightarrow (E \cdot)\}$

Esempio

- $I_9 = \{E \rightarrow E + T \cdot, T \rightarrow T \cdot * F\}$
- $I_{10} = \{T \rightarrow T * F \cdot\}$
- $I_{11} = \{F \rightarrow (E) \cdot\}$

La funzione goto



La funzione goto

- Si noti che il precedente è un DFA: non è un caso!
- Se ogni stato di questo automa è uno stato finale e I_0 è lo stato iniziale, allora l'automata riconosce tutti e soli i viable prefixes della grammatica aumentata
- L'algoritmo aveva intenzione di costruire proprio un automa di questo genere

Item validi

- Un item $A \rightarrow \beta_1 \bullet \beta_2$ si dice **valido** per un viable prefix $\alpha\beta_1$ se e solo se esiste una derivazione:

$$S \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha \beta_1 \beta_2 w$$

- In generale un item sarà valido per più viable prefixes
- L'informazione sul fatto che un certo $A \rightarrow \beta_1 \bullet \beta_2$ è **valido** per $\alpha\beta_1$ ci aiuta molto nella decisione fra lo shift e il reduce

Item validi

- Se durante il parsing troviamo sullo stack $\alpha\beta_1$ allora:
 - Se β_2 è diversa da ϵ allora la è *possibile* che la handle non è ancora stata messa interamente sullo stack. *Dovremmo* procedere quindi a fare uno shift
 - Se β_2 è ϵ allora è *possibile* che $A \rightarrow \beta_1$ sia la handle e quindi la mossa *dovrebbe* essere una riduzione con questa produzione

Item validi

- Tuttavia può succedere che due diversi item validi per lo stesso viable prefix indichino mosse diverse
- Alcuni di questi **conflitti** possono essere risolti guardando il simbolo di lookahead oppure applicando metodi più potenti
- Come sappiamo, in generale, non tutti i conflitti possono essere risolti se il metodo LR scelto viene usato su una grammatica arbitraria

Item validi

- Come trovare gli item validi per un certo viable prefix?
- **Teorema:** L'insieme degli item validi per un viable prefix γ è esattamente l'insieme di item raggiunto dallo stato iniziale lungo un cammino etichettato γ del DFA rappresentato dalla funzione *goto* fra gli stati della collezione canonica LR(0)

Item validi: esempio

- È facile convincersi che $E + T^*$ è un viable prefix per la nostra grammatica aumentata
- “Eseguendo” questa stringa nell'automa che rappresenta la funzione goto si arriva nello stato I_7
- $I_7 = \{ T \rightarrow T^* \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id \}$
- Questi tre item sono esattamente gli item validi per $E + T^*$

Item validi: esempio

- Per convincerci che sono validi, consideriamo le seguenti derivazioni rightmost:
 1. $E' \Rightarrow E \Rightarrow E+T \Rightarrow E+T^*F$ ($T \rightarrow T^* \bullet F$ valido)
 2. $E' \Rightarrow E \Rightarrow E+T \Rightarrow E+T^*F \Rightarrow E+T^*(E)$ ($F \rightarrow \bullet (E)$ valido)
 3. $E' \Rightarrow E \Rightarrow E+T \Rightarrow E+T^*F \Rightarrow E+T^*id$ ($F \rightarrow \bullet id$ valido)
- Non ci sono altri item validi per $E + T^*$

Tabelle di parsing SLR

- Ingredienti:
 - Grammatica aumentata G'
 - DFA che riconosce i viable prefixes di G' (funzione *goto*)
 - FOLLOW(A) per ogni non terminale A di G'
- Output: tabella di parsing LR (se è multidefinita in almeno un'entrata la grammatica non è analizzabile SLR(1))

Algoritmo

1. Costruisci $C = \{I_0, I_1, \dots, I_n\}$, la collezione canonica LR(0)
2. Lo stato i del parser LR è costruito a partire da I_i . Le azioni di parsing per lo stato i sono determinate come segue:
 - a) Se $A \rightarrow \alpha \bullet a \beta \in I_i$ e $goto(I_i, a) = I_j$, allora poni $action[i, a] := \text{"shift } j\text{"}$ (a è terminale!)
 - b) Se $A \rightarrow \alpha \bullet \in I_i$, allora poni $action[i, a] := \text{"reduce } A \rightarrow \alpha\text{"}$ per tutte le $a \in FOLLOW(A)$ ($A \neq S'$)
 - c) Se $S' \rightarrow S \bullet \in I_i$, allora poni $action[i, \$] := \text{"accept"}$

Algoritmo

- La tabella *action+goto* del parser LR così ottenuto è data dalla tabella *action* così costruita e dalla funzione *goto* calcolata durante la costruzione della collezione canonica LR(0)
- Lo stato iniziale del parser LR così ottenuto è quello costruito dall'insieme di item che contiene $S' \rightarrow \bullet S$
- Tutte le entrate non definite sono entrate "error"

Esempio

- Calcoliamo la tabella SLR per la nostra solita grammatica aumentata
- Abbiamo già calcolato la collezione canonica LR(0)
- Esaminiamo tutti gli stati e seguiamo le istruzioni
- Cominciamo con lo stato I_0 che è quello iniziale

Esempio

- $I_0 = \{E' \rightarrow \bullet E,$
 $E \rightarrow \bullet E + T,$
 $E \rightarrow \bullet T,$
 $T \rightarrow \bullet T * F,$
 $T \rightarrow \bullet F,$
 $F \rightarrow \bullet (E),$
 $F \rightarrow \bullet id \}$
- L'item $F \rightarrow \bullet (E)$ indica di porre $action[0, (] := \text{"shift 4"}$ (poiché $goto(I_0, () = I_4$)
- L'item $F \rightarrow \bullet id$ indica di porre $action[0, id] := \text{"shift 5"}$ (poiché $goto(I_0, id) = I_5$)
- Gli altri item di I_0 non suggeriscono azioni

Esempio

- $I_1 = \{E' \rightarrow E \bullet,$
 $E \rightarrow E \bullet + T \}$
- L'item $E' \rightarrow E \bullet$ suggerisce di porre $action[1, \$] = \text{"accept"}$
- L'item $E \rightarrow E \bullet + T$ suggerisce di porre $action[1, +] = \text{"shift 6"}$ poiché $goto(I_1, +) = I_6$

Esempio

- $I_2 = \{E \rightarrow T \bullet, T \rightarrow T \bullet * F\}$
- L'item $E \rightarrow T \bullet$ suggerisce di effettuare una reduce. Si ha che $FOLLOW(E) = \{\$, +, \}$ e quindi $action[2, \$] := action[2, +] := action[2, \}] := \text{"reduce } E \rightarrow T\text{"}$
- L'item $T \rightarrow T \bullet * F$ suggerisce $action[2, *] = \text{"shift 7"}$

Esempio

- Continuando in questo modo otteniamo la tabella di parsing LR che avevamo già visto per questa grammatica quando illustravamo il funzionamento di un parser LR
- Siccome non ci sono entrate multidefinite concludiamo che la grammatica è SLR(1)

La tabella SLR(1)

Stato	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Grammatiche non SLR(1)

- Sappiamo già che nessuna grammatica ambigua può essere LR, quindi tantomeno SLR
- Ci sono grammatiche non ambigue che però non sono SLR. Ad esempio:
 $S \rightarrow L = R \mid R$
 $L \rightarrow * R \mid id$
 $R \rightarrow L$

Esempio

- Questa grammatica non è ambigua
- Genera gli assegnamenti fra identificatori e puntatori oppure espressioni di locazioni o valori
- Es: $id = * id, * id = id, **id$
- L sta per locazione, R è un valore che può essere memorizzato in una locazione, * sta per "il contenuto di"

Collezione canonica LR(0)

$$\begin{aligned}
 I_0 &= \{S' \rightarrow \bullet S, S \rightarrow \bullet L = R, S \rightarrow \bullet R, L \rightarrow \bullet * R, L \rightarrow \bullet id, R \rightarrow \bullet L\} \\
 I_1 &= \{S' \rightarrow S \bullet\} \\
 I_2 &= \{S \rightarrow L \bullet = R, R \rightarrow L \bullet\} \\
 I_3 &= \{S \rightarrow * \bullet R\} \\
 I_4 &= \{L \rightarrow * \bullet R, R \rightarrow * \bullet L, L \rightarrow * * \bullet R, L \rightarrow * id\} \\
 I_5 &= \{L \rightarrow id \bullet\}
 \end{aligned}$$

Esempio

$I_6 = \{S \rightarrow L \bullet R,$

$R \rightarrow \bullet L,$

$L \rightarrow \bullet * R,$

$L \rightarrow \bullet id\}$

$I_7 = \{L \rightarrow * R \bullet\}$

$I_8 = \{R \rightarrow L \bullet\}$

$I_9 = \{S \rightarrow L = R \bullet\}$

Esempio

- Consideriamo lo stato 2.
- L'item $S \rightarrow L \bullet = R$ fa porre $action[2, =] := \text{"shift 6"}$
- L'item $R \rightarrow L \bullet$ suggerisce una riduzione. Ma il simbolo $=$ appartiene a $FOLLOW(R)$ e quindi si ha anche che $action[2, =] := \text{"reduce } R \rightarrow L \text{"}$
- Un conflitto shift/reduce sul simbolo di input $=$ nello stato 2.
- Il metodo SLR non è abbastanza potente per decidere quale azione intraprendere avendo visto una stringa riducibile ad L e il segno $=$