

Linguaggi formali e compilazione

Corso di Laurea in Informatica

A.A. 2014/2015

Parsing shift-reduce

Generalità sul parsing bottom-up

Parsing $SLR(1)$

Elementi generali

- ▶ Un parser generico di tipo bottom-up procede operando una sequenza di riduzioni a partire dalla stringa di input $\alpha_0 = \alpha$ e cercando di risalire così all'assioma iniziale.
- ▶ Al generico passo di riduzione il parser individua, nella stringa corrente α_i , un'opportuna sottostringa β che corrisponde alla parte destra di una produzione $A \rightarrow \beta$ e sostituisce β con A , così *riducendo* α_i ad α_{i+1} :

$$\alpha_i = \gamma\beta\delta, \quad \alpha_{i+1} = \gamma A \delta$$

- ▶ Il processo termina con successo se, per un opportuno valore di i , risulta $\alpha_i = \mathcal{S}$.
- ▶ Nell'ambito del processo di riduzione il parser può costruire (dal basso verso l'alto) un albero di derivazione e/o produrre direttamente codice.

Parsing “Shift-reduce”

- ▶ Un parser *shift-reduce* è un parser di tipo bottom-up che fa uso di uno stack nel quale vengono memorizzati simboli (terminali o non terminali) della grammatica.
- ▶ Il nome deriva dal fatto che le due operazioni fondamentali eseguite del parser sono dette, appunto, *shift* (spostamento) e *reduce* (riduzione).
 - ▶ L'operazione *shift* legge un simbolo dallo stream di input e lo inserisce sullo stack.
 - ▶ L'operazione *reduce* sostituisce sullo stack gli ultimi k simboli inseriti (poniamo X_1, \dots, X_k , con X_k sulla cima) con il simbolo A , naturalmente se esiste la produzione $A \rightarrow X_1 \dots X_k$.
- ▶ Le sole altre operazioni che il parser esegue sono: accettare l'input o segnalare una condizione di errore.

Handle (maniglie)

- ▶ Un parser di tipo shift-reduce deve individuare, come sottostringhe da ridurre a non terminale, non già qualunque sequenza che formi la parte destra di una produzione, bensì esattamente quelle sequenze (e quelle produzioni) usate nella derivazione canonica destra.
- ▶ Tali sequenze devono inoltre essere individuate “nell’ordine giusto”, e cioè l’ordine rovesciato rispetto alla corrispondente derivazione canonica destra.
- ▶ Queste sequenze (ma meglio sarebbe dire “produzioni”) vengono chiamate *handle* (maniglie), di modo che il problema centrale della realizzazione di un tale parser può essere espresso sinteticamente come il problema di individuare le handle.

Esempio

- ▶ Si consideri la grammatica:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

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

- ▶ La corretta riduzione di $\mathbf{id} + \mathbf{id} \times \mathbf{id}$ ad E individua le handle indicate dalla sottolineatura:

$$\underline{\mathbf{id}} + \mathbf{id} \times \mathbf{id} \leftarrow \underline{F} + \mathbf{id} \times \mathbf{id}$$

$$\leftarrow \underline{T} + \mathbf{id} \times \mathbf{id}$$

$$\leftarrow E + \underline{\mathbf{id}} \times \mathbf{id}$$

$$\leftarrow E + \underline{F} \times \mathbf{id}$$

$$\leftarrow E + T \times \underline{\mathbf{id}}$$

$$\leftarrow E + \underline{T \times F}$$

$$\leftarrow \underline{E + T}$$

$$\leftarrow E$$

Handle (continua)

Parsing
shift-reduce

Generalità sul parsing
bottom-up

Parsing $SLR(1)$

- ▶ Riguardiamo il precedente parsing della stringa **id + id × id** evidenziando il contenuto dell'input e dello stack di un parser che correttamente individua le handle.

Esempio

- Azioni eseguite (su input $\mathbf{id + id \times id}$) da un parser shift-reduce che riconosce correttamente le handle.

| Stack | Input | Azione | Stringa α_j |
|---------------------|-----------------------|--------|--------------------|
| \$ | <u>id</u> + id × id\$ | shift | id + id × id\$ |
| \$ <u>id</u> | +id × id\$ | reduce | id + id × id\$ |
| \$ <u>F</u> | +id × id\$ | reduce | F + id × id\$ |
| \$ <u>T</u> | +id × id\$ | reduce | T + id × id\$ |
| \$ <u>E</u> | + <u>id</u> × id\$ | shift | E + id × id\$ |
| \$E+ | <u>id</u> × id\$ | shift | E + id × id\$ |
| \$E + <u>id</u> | × id\$ | reduce | E + id × id\$ |
| \$E + <u>F</u> | × id\$ | reduce | E + F × id\$ |
| \$E + <u>T</u> | × <u>id</u> \$ | shift | E + T × id\$ |
| \$E + T × | <u>id</u> \$ | shift | E + T × id\$ |
| \$E + T × <u>id</u> | \$ | reduce | E + T × id\$ |
| \$E + <u>T × F</u> | \$ | reduce | E + T × F\$ |
| \$ <u>E + T</u> | \$ | reduce | E + T\$ |
| \$ <u>E</u> | \$ | accept | E\$ |

- ▶ Ad ogni dato istante, l'attuale forma di frase (la stringa α_i) si trova “parte sullo stack e parte ancora sullo stream di input”.
- ▶ Più precisamente, se lo stack contiene una stringa $\alpha\beta_1$ (dal basso verso l'alto) e lo stream di input contiene la stringa $\beta_2\gamma$, allora la forma di frase “corrente” nella derivazione destra è $\alpha\beta_1\beta_2\gamma$.
- ▶ Se la prossima handle è la produzione $A \rightarrow \beta_1\beta_2$ allora:
 - ▶ se $\beta_2 = \epsilon$ allora la prossima mossa è la riduzione;
 - ▶ se $\beta_2 \neq \epsilon$ allora la prossima mossa è uno shift;
- ▶ Se la prossima handle non è $A \rightarrow \beta_1\beta_2$ allora il parser esegue uno shift o dichiara errore (come vedremo).

- ▶ L'osservazione più importante è che la prossima handle da utilizzare “prima o poi” si trova esattamente sulla cima dello stack.
- ▶ Questa proprietà vale perché consideriamo derivazioni canoniche destre; non varrebbe nel caso volessimo riprodurre una derivazione canonica sinistra.

Esempio

- ▶ Azioni eseguite (su input **id + id**) da un parser shift-reduce che “ricostruisce” una derivazione canonica sinistra, riconoscendo le handle.
- ▶ Come si può vedere, non è possibile garantire che le handle siano sempre sulla cima dello stack.

| Stack | Input | Azione | Stringa α_j |
|-------------------------|---------------------|--------|--------------------|
| \$ | id + id \$ | shift | id + id \$ |
| \$ id | + id \$ | shift | id + id \$ |
| \$ id + | <u>id</u> \$ | shift | id + id \$ |
| \$ id + id | \$ | reduce | id + id \$ |
| \$ id + F | \$ | reduce | id + F \$ |
| \$ <u>id</u> + T | \$ | reduce | id + T \$ |
| \$ <u>F</u> + T | \$ | reduce | F + T\$ |
| \$ <u>T</u> + T | \$ | reduce | T + T\$ |
| \$ <u>E</u> + T | \$ | reduce | E + T\$ |
| \$ <u>E</u> | \$ | accept | E\$ |

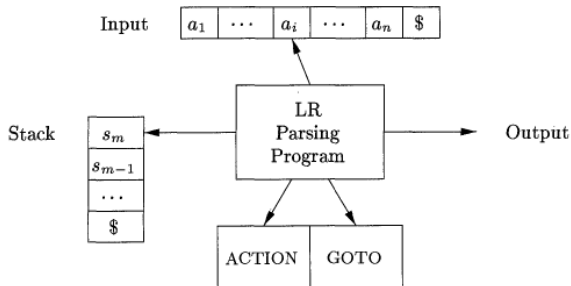
Il cuore computazionale del problema

- ▶ La difficoltà di progettazione del parser sta tutta nella capacità di riconoscere quando è corretto operare uno shift e quando invece è corretto operare una riduzione.
- ▶ Il problema coincide con quello di determinare esattamente le handle. Infatti, se fossimo in grado di risolvere quest'ultimo sapremmo sempre quando operare uno shift e quando eseguire una riduzione.
- ▶ Dovremmo infatti “ridurre” quando e solo quando una maniglia appare sulla cima dello stack.
- ▶ Sfortunatamente ci sono grammatiche per le quali il paradigma shift-reduce non è applicabile, ad esempio grammatiche ambigue.

- ▶ Si tratta di una classe di parser di tipo shift-reduce (con analisi dell'input da sinistra a destra, "Left to Right"), caratterizzati da una struttura di programma comune ma con capacità di analisi diverse.
- ▶ La diversa capacità di effettuare il parsing dipende dall'informazione contenuta in apposite tabelle di parsing che guidano il comportamento del programma.
- ▶ In questi appunti analizzeremo un solo tipo di parser LR , il più semplice, che prende (non a caso) il nome di (parser) $SLR(1)$.
- ▶ Per prima cosa vedremo però la "program structure" comune.

Struttura di un parser LR

- Un parser LR è caratterizzato da un programma di controllo (essenzialmente un automa a stati finiti) che ha accesso ad uno stack e ad una tabella di parsing, oltre che a opportuni supporti di input e output.



Struttura di un parser *LR* (continua)

- ▶ Le tabelle prescrivono il comportamento del programma di controllo in funzione del contenuto dello stack e dei primi k caratteri presenti in input (per noi $k = 1$).
- ▶ Lo stack, a differenza dei parser shift-reduce visti precedentemente, contiene stati anziché simboli.
- ▶ Tuttavia, come vedremo, ad ogni stato è associato univocamente un simbolo della grammatica (l'inverso non è necessariamente vero).
- ▶ Come nel caso generico di parser shift-reduce, possiamo quindi ricostruire la forma di frase corrente (di una derivazione canonica destra) utilizzando i simboli memorizzati sullo stack concatenati con i simboli ancora sullo stream di input.

Tabelle di parsing

- ▶ Le tabelle di parsing di un parser *LR* hanno un numero di righe pari al numero di stati dell'automa che costituisce il controllo.
- ▶ Le colonne sono indicizzate dai simboli terminali e non terminali. Le colonne relative ai terminali formano quella che viene detta “parte *ACTION*” della tabella, mentre le altre formano la “parte *GOTO*”.
- ▶ Nella parte action sono previste 4 tipi di azioni:
 - ▶ avanzamento di un carattere sullo stream di input e inserimento di uno stato in cima allo stack;
 - ▶ esecuzione di una riduzione;
 - ▶ accettazione dell'input;
 - ▶ rilevamento di un errore.
- ▶ La parte *GOTO* prescrive stati da inserire nello stack.

Funzionamento del parser

- ▶ Il funzionamento del parser è definito come segue.
- ▶ Inizialmente, lo stack contiene un solo stato (lo stato iniziale, naturalmente).
- ▶ Al generico passo, sia q lo stato in cima allo stack e x il prossimo carattere in input.
- ▶ Se $ACTION[q, x] = \text{shift } r$, il parser avanza il puntatore di input e inserisce lo stato r sullo stack.
- ▶ Se $ACTION[q, x] = \text{reduce } i$, il parser utilizza la i -esima produzione (secondo una numerazione arbitraria ma prefissata). Più precisamente, se $A \rightarrow \alpha$ è tale produzione, il parser rimuove $k_i = |\alpha|$ stati dallo stack e vi inserisce lo stato $GOTO[q', A]$ dove q' è lo stato sulla cima dello stack dopo le k_i rimozioni.
- ▶ Il parser si arresta in seguito ad accettazione o errore.

Esempio

- Consideriamo la grammatica che genera sequenze di parentesi bilanciate:

$S \rightarrow (S)S$ Produzione 1

$S \rightarrow \epsilon$ Produzione 2

e consideriamo la seguente tabella di parsing (di cui vedremo più avanti la costruzione):

| Stato | ACTION | | | GOTO |
|-------|---------|----------|----------|------|
| | (|) | \$ | S |
| 0 | shift 2 | reduce 2 | reduce 2 | 1 |
| 1 | | | accept | |
| 2 | shift 2 | reduce 2 | reduce 2 | 3 |
| 3 | | shift 4 | | |
| 4 | shift 2 | reduce 2 | reduce 2 | 5 |
| 5 | | reduce 1 | reduce 1 | |

- Consideriamo il comportamento del parser su input $()()$.

Esempio (continua)

| Stack | Input | Azione |
|------------|---------|---------------------------------|
| \$0 | ()())\$ | shift 2 |
| \$02 |)())\$ | reduce $S \rightarrow \epsilon$ |
| \$023 |)())\$ | shift 4 |
| \$0234 | ()\$ | shift 2 |
| \$02342 |)\$ | reduce $S \rightarrow \epsilon$ |
| \$023423 |)\$ | shift 4 |
| \$0234234 | \$ | reduce $S \rightarrow \epsilon$ |
| \$02342345 | \$ | reduce $S \rightarrow (S)S$ |
| \$02345 | \$ | reduce $S \rightarrow (S)S$ |
| \$01 | \$ | accept |

- ▶ Si ricordi che la riduzione con $S \rightarrow (S)S$ prima rimuove 4 stati dallo stack, quindi inserisce lo stato $GOTO[q', S]$, dove q' è lo stato che rimane in cima allo stack dopo le rimozioni.
- ▶ Analogamente, la riduzione con $S \rightarrow \epsilon$ rimuove 0 stati.

Parsing SLR(1)

Parsing shift-reduce

Generalità sul parsing
bottom-up

Parsing *SLR(1)*

- ▶ Come detto, l'unico tipo di parser *LR* che analizziamo è detto *Simple LR parser* (o semplicemente *SLR*).
- ▶ È caratterizzato da tabelle di parsing di relativamente semplice costruzione (da cui il nome) ma che danno minori garanzie sulla possibilità di analisi di grammatiche libere.
- ▶ In altri termini, ci sono diverse grammatiche libere di interesse che non possono essere analizzate con parser *SLR* (e, segnatamente, *SLR(1)*).
- ▶ Si tratta comunque di un primo caso di interesse per capire la “logica” di un parser *LR*.

Automa LR(0)

- ▶ Il passo fondamentale consiste nella definizione di un automa (che di fatto sarà poi “trasferito” nella tabella di parsing), detto *automa LR(0)*.
- ▶ Data la grammatica G , la si “aumenta” con una produzione aggiuntiva, $S' \rightarrow S$ (il cui significato sarà chiaro più avanti).
- ▶ A partire dalle produzioni della grammatica aumentata, si definiscono poi speciali “oggetti”, che chiameremo *item*.
- ▶ Un item è una produzione con inserito un marcatore nella parte destra, tipicamente un punto.
- ▶ Ad esempio, gli item associati alla produzione $S \rightarrow (S)S$ sono: $S \rightarrow \cdot(S)S$, $S \rightarrow (\cdot S)S$, $S \rightarrow (S\cdot)S$, $S \rightarrow (S)\cdot S$ e $S \rightarrow (S)S\cdot$.
- ▶ Ad una produzione tipo $S \rightarrow \epsilon$ è associato il solo item $S \rightarrow \cdot$.

Automa $LR(0)$ (continua)

- ▶ Qual è il significato di un item associato ad una data produzione?
- ▶ Intuitivamente, esso indica il “punto” al quale siamo arrivati nel processo di riconoscimento della parte destra della produzione stessa.
- ▶ Ad esempio, l'item $S \rightarrow (S) \cdot S$ indica che abbiamo riconosciuto una stringa descritta da (S) e che ci “attendiamo” di riconoscere una stringa descrivibile da S .
- ▶ Un item con il puntino in fondo indica quindi che il processo di riconoscimento della parte destra è completato e dunque che si può operare la riduzione (vedremo sotto quale altra condizione).

Automa $LR(0)$ (continua)

- ▶ Gli item vengono poi raggruppati in collezioni, ognuna delle quali definisce uno “stato” nel processo di riconoscimento.
- ▶ Ad esempio, una collezione per il parser della grammatica per le parentesi appena vista sarà costituito dagli item:

$$S \rightarrow (S) \cdot S$$

$$S \rightarrow \cdot (S)S$$

$$S \rightarrow \cdot$$

- ▶ Essa descrive la situazione in cui abbiamo riconosciuto (S) e ci attendiamo di riconoscere S (primo item), cioè ci attendiamo di riconoscere ancora un'istanza completa di $(S)S$ (secondo item) o, in alternativa, la stringa vuota (terzo item).
- ▶ Le collezioni di item costituiranno proprio gli stati dell'automa al cuore del parser.

- ▶ Si noti che l'intersezione di due collezioni può non essere vuota.
- ▶ Ad esempio, l'item $S \rightarrow (\cdot S)S$ forma gruppo ancora con $S \rightarrow \cdot (S)S$ e $S \rightarrow \cdot$ (per la stessa ragione di prima).
- ▶ Sono le collezioni nel loro insieme che devono essere distinte.
- ▶ Naturalmente, in casi particolari un item può formare uno stato/collezione da solo.
- ▶ Questo è il caso, ad esempio, dell'item $S \rightarrow (S \cdot)S$.

Esempio

- Per la grammatica “aumentata”

$$S' \rightarrow S$$

$$S \rightarrow (S)S \mid \epsilon$$

sono definiti i seguenti insiemi di item:

$$\begin{array}{ll}
 I_0 : S' \rightarrow \cdot S & I_3 : S \rightarrow (S \cdot) S \\
 S \rightarrow \cdot (S) S & \\
 S \rightarrow \cdot & \\
 \\
 I_1 : S' \rightarrow S \cdot & I_4 : S \rightarrow (S) \cdot S \\
 & S \rightarrow \cdot (S) S \\
 & S \rightarrow \cdot \\
 \\
 I_2 : S \rightarrow (\cdot S) S & I_5 : S \rightarrow (S) S \cdot \\
 S \rightarrow \cdot (S) S & \\
 S \rightarrow \cdot &
 \end{array}$$

Come costruire gli insiemi $LR(0)$

- ▶ Diamo ora una descrizione dettagliata del procedimento di costruzione degli insiemi di item.
- ▶ L'insieme iniziale (che indicheremo sempre con I_0) contiene l'item $S' \rightarrow \cdot S$ e tutti gli item ottenuti dalle produzioni di S inserendo il punto all'inizio.
- ▶ Nell'esempio appena considerato, si aggiungono a $S' \rightarrow \cdot S$ due soli item (perché ci sono due produzioni relative ad S).
- ▶ Si procede poi ricorsivamente, lavorando ad ogni passo su un insieme I_j già formato.
- ▶ Si considerano tutti i simboli della grammatica immediatamente alla destra del punto in item di I_j .
- ▶ Per ogni simbolo così individuato, si forma un gruppo I_k che contiene, inizialmente, gli item ottenuti spostando il punto alla destra del simbolo considerato.

Come costruire gli insiemi $LR(0)$ (continua)

- ▶ Ad esempio, fra gli item di I_0 (per la grammatica appena considerata) ci sono due soli simboli alla destra del punto, S e $($:

$$\begin{aligned}
 I_0 : \quad & S' \rightarrow \cdot S \\
 & S \rightarrow \cdot (S)S \\
 & S \rightarrow \cdot
 \end{aligned}$$

- ▶ Per ognuno di essi si creano due nuovi insiemi, I_1 e I_2 , che contengono inizialmente un solo item ciascuno:

$$I_1 : S' \rightarrow S \cdot$$

$$I_2 : S \rightarrow (\cdot S)S$$

Come costruire gli insiemi $LR(0)$ (continua)

- ▶ Se il nuovo insieme I_k appena inizializzato contiene item in cui il punto precede un simbolo non terminale A , si aggiungono ad I_k tutti gli item ottenuti dalle produzioni di A inserendo il punto all'inizio.
- ▶ Quest'ultima operazione è detta *chiusura* dell'insieme I_k .
- ▶ Continuando l'esempio precedente, poiché l'insieme I_2 contiene l'item $S \rightarrow (\cdot S)S$, ad esso si aggiungono gli item $S \rightarrow \cdot(S)S$ e $S \rightarrow \cdot$:

$$I_2 : \begin{array}{l} S \rightarrow (\cdot S)S \\ S \rightarrow \cdot(S)S \\ S \rightarrow \cdot \end{array}$$

- ▶ Il procedimento termina quando non ci sono più insiemi di item da considerare.

Funzioni *CLOSURE* e *GOTO*

Parsing shift-reduce

Generalità sul parsing
bottom-up

Parsing *SLR(1)*

- ▶ Il procedimento appena descritto (in maniera alquanto discorsiva) può essere sinteticamente ricapitolato facendo uso delle due funzioni *CLOSURE* e *GOTO*, che lavorano su insiemi di item.
- ▶ Dato un insieme di item I , $CLOSURE(I)$ si ottiene aggiungendo (ricorsivamente) ad I item del tipo $B \rightarrow \cdot \gamma$ sotto le seguenti condizioni:
 - ▶ in I esista inizialmente un item del tipo $A \rightarrow \alpha \cdot B\beta$, oppure,
 - ▶ ad I sia già stato aggiunto un item del tipo $A \rightarrow \cdot B\beta$.
- ▶ Il procedimento termina quando non si possono più aggiungere item sulla base delle precedenti regole.

Funzione *CLOSURE(I)*

Parsing
shift-reduce

Generalità sul parsing
bottom-up

Parsing *SLR(1)*

```
SetOfItems CLOSURE(I) {  
    J = I;  
    repeat  
        for ( each item  $A \rightarrow \alpha \cdot B \beta$  in J )  
            for ( each production  $B \rightarrow \gamma$  of G )  
                if (  $B \rightarrow \cdot \gamma$  is not in J )  
                    add  $B \rightarrow \cdot \gamma$  to J;  
    until no more items are added to J on one round;  
    return J;  
}
```

Funzioni *CLOSURE* e *GOTO* (continua)

Parsing shift-reduce

Generalità sul parsing
bottom-up

Parsing *SLR*(1)

- ▶ Se I è un insieme di item e X un simbolo della grammatica $GOTO(I, X)$ è un insieme di item, che chiameremo J , calcolato nel seguente modo:
 - ▶ inizialmente si pone $J = \{\}$;
 - ▶ per ogni item $A \rightarrow \alpha \cdot X\beta$ in I , si aggiunge a J l'item $A \rightarrow \alpha X \cdot \beta$;
 - ▶ infine si pone $J \leftarrow CLOSURE(J)$.

- ▶ Utilizzando le funzione $CLOSURE$ e $GOTO$ possiamo definire con precisione il calcolo degli insiemi di item per una grammatica aumentata.

```
1:  $C \leftarrow \{CLOSURE(\{S' \rightarrow \cdot S\})\}$ 
2: repeat
3:   for each  $I \in C$  do
4:     for each  $X \in \mathcal{T} \cup \mathcal{N}$  do
5:       if  $GOTO(I, X) \neq \{\}$  &  $GOTO(I, X) \notin C$  then
6:          $C \leftarrow C \cup \{GOTO(I, X)\}$ 
7: until No new state is added to  $C$ 
```


Esempio

Parsing shift-reduce

Generalità sul parsing
bottom-up

Parsing SLR(1)

- ▶ Consideriamo la seguente grammatica (aumentata) che genera il linguaggio $\{a^n b^n | n \geq 1\}$:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow aSb \mid ab \end{aligned}$$

- ▶ L'insieme iniziale di item è $I_0 = CLOSURE\{S' \rightarrow \cdot S\} = \{S' \rightarrow \cdot S, S \rightarrow \cdot aSb, S \rightarrow \cdot ab\}$.
- ▶ I simboli immediatamente a destra del punto in I_0 sono S e a , per cui calcoliamo i due insiemi:
 - ▶ $I_1 = GOTO(I_0, S) = \{S' \rightarrow S \cdot\}$;
 - ▶ $I_2 = GOTO(I_0, a) = \{S \rightarrow a \cdot Sb, S \rightarrow a \cdot b, S \rightarrow \cdot aSb, S \rightarrow \cdot ab\}$

Esempio (continua)

- ▶ L'insieme I_1 non dà origine ad altri insiemi di item (perché non ci sono simboli a destra del punto).
- ▶ Nel'insieme I_2 ci sono tre simboli distinti a destra del punto, per cui formiamo tre insiemi:
 - ▶ $I_3 = GOTO(I_2, S) = \{S \rightarrow aS \cdot b\};$
 - ▶ $I_4 = GOTO(I_2, b) = \{S \rightarrow ab \cdot\};$
 - ▶ $I_5 = GOTO(I_2, a) = \{S \rightarrow a \cdot Sb, S \rightarrow a \cdot b, S \rightarrow \cdot aSb, S \rightarrow \cdot ab\}.$
- ▶ Tuttavia, I_5 viene “scartato”, in quanto coincide con I_2 .
- ▶ Infine lavorando su I_3 si ottiene (“riusando” il simbolo I_5):
 - ▶ $I_5 = GOTO(I_3, b) = \{S \rightarrow aSb \cdot\}.$

Esempio (continua)

- Ricapitolando, gli insiemi $LR(0)$ di item associati alla grammatica sono:

$$\begin{array}{ll}
 I_0 : & S' \rightarrow \cdot S \\
 & S \rightarrow \cdot a S b \\
 & S \rightarrow \cdot ab \\
 I_3 : & S \rightarrow a S \cdot b \\
 I_4 : & S \rightarrow ab \cdot
 \end{array}$$

$$\begin{array}{ll}
 I_1 : & S' \rightarrow S \cdot \\
 I_5 : & S \rightarrow a S b \cdot
 \end{array}$$

$$\begin{array}{l}
 I_2 : S \rightarrow a \cdot S b \\
 S \rightarrow a \cdot b \\
 S \rightarrow \cdot a S b \\
 S \rightarrow \cdot ab
 \end{array}$$

Esempio

- ▶ Da ultimo, consideriamo la costruzione degli insiemi di item $LR(0)$ per la grammatica aumentata

$$E' \rightarrow E$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

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

che, ricordiamo, non è adatta al parsing top-down.

- ▶ Nella slide seguente presentiamo direttamente la collezione degli insiemi di item ottenuta applicando l'algoritmo di costruzione degli insiemi di item.

Esempio (continua)

$$\begin{aligned}l_0 : & E' \rightarrow \cdot E \\ & E \rightarrow \cdot E + T \\ & E \rightarrow \cdot T \\ & T \rightarrow \cdot T \times F \\ & T \rightarrow \cdot F \\ & F \rightarrow \cdot (E) \\ & F \rightarrow \cdot \mathbf{id}\end{aligned}$$
$$\begin{aligned}l_1 : & E' \rightarrow E \cdot \\ & E \rightarrow E \cdot + T\end{aligned}$$
$$\begin{aligned}l_2 : & E \rightarrow T \cdot \\ & T \rightarrow T \cdot \times F\end{aligned}$$
$$l_3 : T \rightarrow F \cdot$$
$$\begin{aligned}l_4 : & F \rightarrow (\cdot E) \\ & E \rightarrow \cdot E + T \\ & E \rightarrow \cdot T \\ & T \rightarrow \cdot T \times F \\ & T \rightarrow \cdot F \\ & F \rightarrow \cdot (E) \\ & F \rightarrow \cdot \mathbf{id}\end{aligned}$$
$$l_5 : F \rightarrow \mathbf{id} \cdot$$
$$\begin{aligned}l_6 : & E \rightarrow E + \cdot T \\ & T \rightarrow \cdot T \times F \\ & T \rightarrow \cdot F \\ & F \rightarrow \cdot (E) \\ & F \rightarrow \cdot \mathbf{id}\end{aligned}$$
$$\begin{aligned}l_7 : & T \rightarrow T \times \cdot F \\ & F \rightarrow \cdot (E) \\ & F \rightarrow \cdot \mathbf{id}\end{aligned}$$
$$\begin{aligned}l_8 : & E \rightarrow E \cdot + T \\ & F \rightarrow (E \cdot)\end{aligned}$$
$$\begin{aligned}l_9 : & E \rightarrow E + T \cdot \\ & T \rightarrow T \cdot \times F\end{aligned}$$
$$l_{10} : T \rightarrow T \times F \cdot$$
$$l_{11} : F \rightarrow (E) \cdot$$

Parsing
shift-reduce

Generalità sul parsing
bottom-up

Parsing SLR(1)

Automa $LR(0)$

- ▶ Come già anticipato, le collezioni di item $LR(0)$ determinate con la procedura appena descritta costituiscono gli stati dell'automa $LR(0)$ (che, a sua volta, è alla base del parsing $SLR(1)$ che stiamo costruendo).
- ▶ Per completare la descrizione dell'automa è necessario definire la funzione δ di transizione.
- ▶ In realtà abbiamo già descritto tale funzione, che coincide “essenzialmente” con la funzione $GOTO$.
- ▶ Si noti che, tuttavia, che $GOTO(I, X)$ “costruisce” nuovi stati e dunque $J = GOTO(I, X)$ non viene aggiunto se risulta già definito,
- ▶ In tale caso vale comunque $\delta(I, X) = J$.

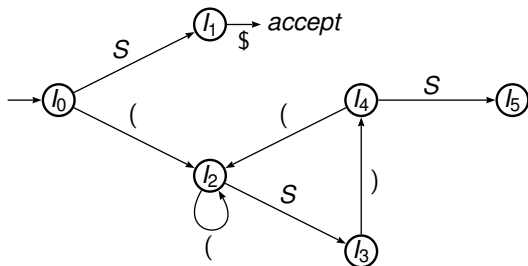
Esempio

- L'automa $LR(0)$ per la grammatica

$$S' \rightarrow S$$

$$S \rightarrow (S)S \mid \epsilon$$

è:



Insiemi di item e automa

$$\begin{aligned}
 I_0 : & S' \rightarrow \cdot S \\
 & S \rightarrow \cdot (S)S \\
 & S \rightarrow \cdot
 \end{aligned}$$

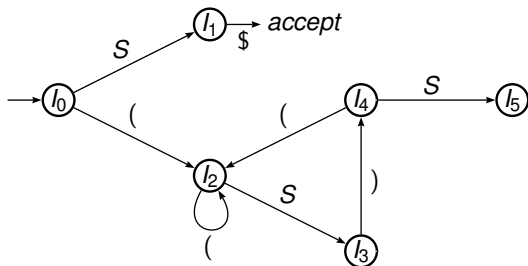
$$I_3 : S \rightarrow (S \cdot)S$$

$$\begin{aligned}
 I_4 : & S \rightarrow (S) \cdot S \\
 & S \rightarrow \cdot (S)S \\
 & S \rightarrow \cdot
 \end{aligned}$$

$$I_1 : S' \rightarrow S \cdot$$

$$\begin{aligned}
 I_2 : & S \rightarrow (\cdot S)S \\
 & S \rightarrow \cdot (S)S \\
 & S \rightarrow \cdot
 \end{aligned}$$

$$I_5 : S \rightarrow (S)S \cdot$$



Parsing
shift-reduce

Generalità sul parsing
bottom-up

Parsing *SLR(1)*

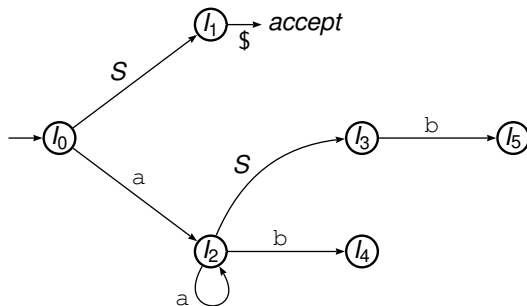
Esempio

- L'automa $LR(0)$ per la grammatica

$$S' \rightarrow S$$

$$S \rightarrow aSb \mid ab$$

è:



Esempio

- Ricordiamo anche gli insiemi di item:

$$l_0 : S' \rightarrow \cdot S \\ S \rightarrow \cdot a S b \\ S \rightarrow \cdot a b$$

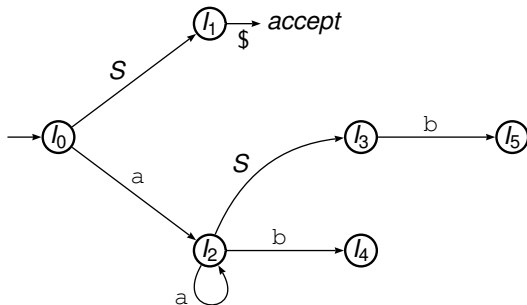
$$l_1 : S' \rightarrow S \cdot$$

$$l_2 : S \rightarrow a \cdot S b \\ S \rightarrow a \cdot b \\ S \rightarrow \cdot a S b \\ S \rightarrow \cdot a b$$

$$l_3 : S \rightarrow a S \cdot b$$

$$l_4 : S \rightarrow a b \cdot$$

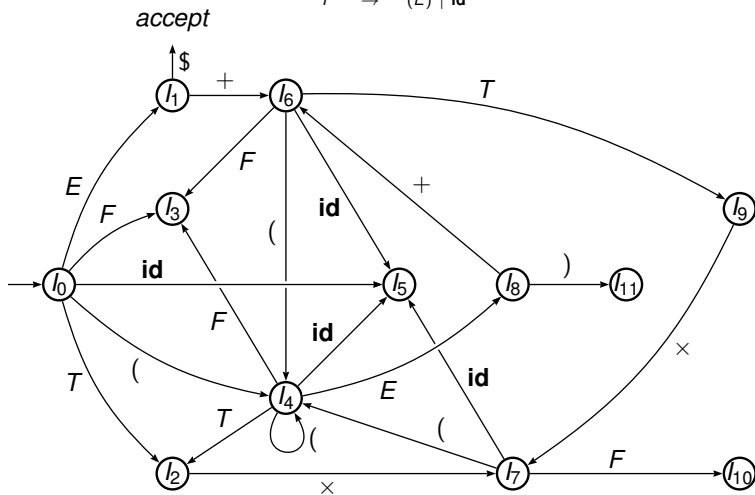
$$l_5 : S \rightarrow a S b \cdot$$



Esempio

- L'ultimo esempio è per la grammatica

$$\begin{aligned}
 E' &\rightarrow E \\
 E &\rightarrow E + T \mid T \\
 T &\rightarrow T \times F \mid F \\
 F &\rightarrow (E) \mid \text{id}
 \end{aligned}$$



Parsing
shift-reduce

Generalità sul parsing
bottom-up

Parsing SLR(1)

Gli insiemi di item

$$l_0 : E' \rightarrow \cdot E$$
$$E \rightarrow \cdot E + T$$
$$E \rightarrow \cdot T$$
$$T \rightarrow \cdot T \times F$$
$$T \rightarrow \cdot F$$
$$F \rightarrow \cdot (E)$$
$$F \rightarrow \cdot \mathbf{id}$$
$$l_1 : E' \rightarrow E \cdot$$
$$E \rightarrow E \cdot + T$$
$$l_2 : E \rightarrow T \cdot$$
$$T \rightarrow T \cdot \times F$$
$$l_3 : T \rightarrow F \cdot$$
$$l_4 : F \rightarrow (\cdot E)$$
$$E \rightarrow \cdot E + T$$
$$E \rightarrow \cdot T$$
$$T \rightarrow \cdot T \times F$$
$$T \rightarrow \cdot F$$
$$F \rightarrow \cdot (E)$$
$$F \rightarrow \cdot \mathbf{id}$$
$$l_5 : F \rightarrow \mathbf{id} \cdot$$
$$l_6 : E \rightarrow E + \cdot T$$
$$T \rightarrow \cdot T \times F$$
$$T \rightarrow \cdot F$$
$$F \rightarrow \cdot (E)$$
$$F \rightarrow \cdot \mathbf{id}$$
$$l_7 : T \rightarrow T \times \cdot F$$
$$F \rightarrow \cdot (E)$$
$$F \rightarrow \cdot \mathbf{id}$$
$$l_8 : E \rightarrow E \cdot + T$$
$$F \rightarrow (E \cdot)$$
$$l_9 : E \rightarrow E + T \cdot$$
$$T \rightarrow T \cdot \times F$$
$$l_{10} : T \rightarrow T \times F \cdot$$
$$l_{11} : F \rightarrow (E) \cdot$$

Parsing
shift-reduce

Generalità sul parsing
bottom-up

Parsing SLR(1)

Tabelle di parsing *SLR(1)*

Parsing shift-reduce

Generalità sul parsing
bottom-up

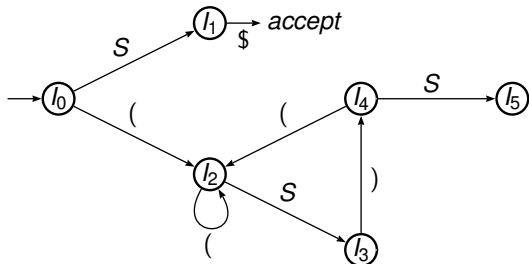
Parsing *SLR(1)*

- ▶ Completiamo ora la descrizione del parser con l'algoritmo di definizione della tabella di parsing.
- ▶ Le tabelle incorporano le informazioni contenute nell'automa, che da solo non è sufficiente per eseguire l'analisi (si ricordi che un automa a stati finiti non è in grado di riconoscere linguaggi liberi (che non siano anche regolari).
- ▶ L'algoritmo esamina gli stati dell'automa e le transizioni uscenti da ciascuno stato.
- ▶ Esso necessita anche di conoscere, per ogni simbolo non terminale A , l'insieme di simboli $FOLLOW(A)$.

Tabelle di parsing $SLR(1)$ (continua)

- ▶ Per ogni stato I_j , consideriamo le transizioni uscenti.
- ▶ Se esiste una transizione da I_j a I_k etichettata $X \in \mathcal{T}$ poniamo $ACTION[j, X] = \text{shift } k$.
- ▶ Se esiste una transizione da I_j a I_k etichettata $X \in \mathcal{N}$ poniamo $GOTO[j, X] = k$.
- ▶ Se nell'insieme di item corrispondenti a I_j esiste un item $A \rightarrow \alpha \cdot$, allora poniamo $ACTION[j, X] = \text{reduce } A \rightarrow \alpha$ per tutti i simboli X in $FOLLOW(A)$.
- ▶ Se I_j contiene l'item $S' \rightarrow S \cdot$ si pone $ACTION[j, \$] = \text{accept}$.
- ▶ Se, ad un qualunque passo dell'algoritmo, si manifesta un cosiddetto *conflitto shift-reduce* (cioè si tenta di inserire in una entry della parte ACTION sia un'azione di shift che una di riduzione) allora la grammatica non è $SLR(1)$.

Esempio (grammatica per le parentesi)



| Stato | ACTION | | | GOTO |
|-------|---------|----------|----------|------|
| | (|) | \$ | S |
| 0 | shift 2 | reduce 2 | reduce 2 | 1 |
| 1 | | | accept | |
| 2 | shift 2 | reduce 2 | reduce 2 | 3 |
| 3 | | shift 4 | | |
| 4 | shift 2 | reduce 2 | reduce 2 | 5 |
| 5 | | reduce 1 | reduce 1 | |

Esempio

Parsing shift-reduce

Generalità sul parsing
bottom-up

Parsing *SLR(1)*

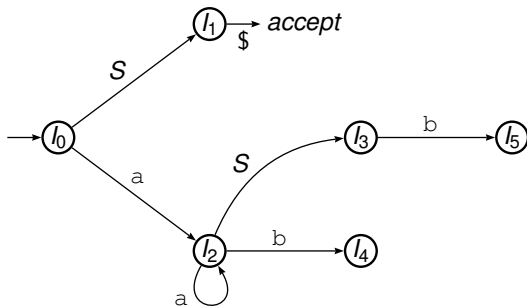
- ▶ Riconsideriamo la grammatica
$$S \rightarrow aSb \quad \text{Produzione 1}$$
$$S \rightarrow ab \quad \text{Produzione 2}$$
in cui abbiamo numerato (arbitrariamente) le produzioni.
- ▶ Per tale grammatica l'algoritmo appena delineato produce la tabella di parsing evidenziata nella seguente diapositiva (in cui riportiamo, per comodità, anche l'automa $LR(0)$).

Esempio (continua)

Parsing shift-reduce

Generalità sul parsing
bottom-up

Parsing SLR(1)



| Stato | ACTION | | | GOTO |
|-------|---------|----------|----------|------|
| | a | b | \$ | S |
| 0 | shift 2 | | | 1 |
| 1 | | | accept | |
| 2 | shift 2 | shift 4 | | 3 |
| 3 | | shift 5 | | |
| 4 | | reduce 2 | reduce 2 | |
| 5 | | reduce 1 | reduce 1 | |

Esempio (continua)

- Consideriamo il comportamento del parser su input
aabb

| Stack | Input | Azione |
|--------|--------|----------------------------|
| \$0 | aabb\$ | shift 2 |
| \$02 | abb\$ | shift 2 |
| \$022 | bb\$ | shift 4 |
| \$0224 | b\$ | reduce $S \rightarrow ab$ |
| \$023 | b\$ | shift 5 |
| \$0235 | \$ | reduce $S \rightarrow aSb$ |
| \$01 | \$ | accept |

Esempio

- Diamo infine la tabella di parsing per la grammatica

$E \rightarrow E + T$ Prod. 1 $T \rightarrow F$ Prod. 4

$E \rightarrow T$ Prod. 2 $F \rightarrow (E)$ Prod. 5

$T \rightarrow T \times F$ Prod. 3 $F \rightarrow \mathbf{id}$ Prod. 6

Parsing
shift-reduce

Generalità sul parsing
bottom-up

Parsing SLR(1)

| Stato | ACTION | | | | | | GOTO | | |
|-------|--------|-----|-----|-----|------|--------|------|---|----|
| | id | + | × | (|) | \$ | E | T | F |
| 0 | s 5 | | | s 4 | | | 1 | 2 | 3 |
| 1 | | s 6 | | | | accept | | | |
| 2 | | r 2 | s 7 | | r 2 | r 2 | | | |
| 3 | | r 4 | r 4 | | r 4 | r 4 | | | |
| 4 | s 5 | | | s 4 | | | 8 | 2 | 3 |
| 5 | | r 6 | r 6 | | r 6 | r 6 | | | |
| 6 | s 5 | | | s 4 | | | | 9 | 3 |
| 7 | s 5 | | | s 4 | | | | | 10 |
| 8 | | s 6 | | | s 11 | | | | |
| 9 | | r 1 | s 7 | | r 1 | r 1 | | | |
| 10 | | r 3 | r 3 | | r 3 | r 3 | | | |
| 11 | | r 5 | r 5 | | r 5 | r 5 | | | |

Esempio (continua)

- Consideriamo il comportamento del parser su input $\mathbf{id} \times (\mathbf{id} + \mathbf{id})$

| Stack | Input | Azione |
|-----------|--|------------------------------------|
| \$0 | id × (id + id) \$ | shift 5 |
| \$05 | × (id + id) \$ | reduce $F \rightarrow \mathbf{id}$ |
| \$03 | × (id + id) \$ | reduce $T \rightarrow F$ |
| \$02 | × (id + id) \$ | shift 7 |
| \$027 | (id + id) \$ | shift 4 |
| \$0274 | id + id) \$ | shift 5 |
| \$02745 | + id) \$ | reduce $F \rightarrow \mathbf{id}$ |
| \$02743 | + id) \$ | reduce $T \rightarrow F$ |
| \$02742 | + id) \$ | reduce $E \rightarrow T$ |
| \$02748 | + id) \$ | shift 6 |
| \$027486 | id) \$ | shift 5 |
| \$0274865 |) \$ | reduce $F \rightarrow \mathbf{id}$ |
| \$0274863 |) \$ | reduce $T \rightarrow F$ |
| \$0274869 |) \$ | reduce $E \rightarrow E + T$ |
| \$02748 |) \$ | shift 11 |
| \$0274811 | \$ | reduce $F \rightarrow (E)$ |
| \$02710 | \$ | reduce $T \rightarrow T \times F$ |
| \$02 | \$ | reduce $E \rightarrow T$ |
| \$01 | \$ | accept |