Parsing SLR

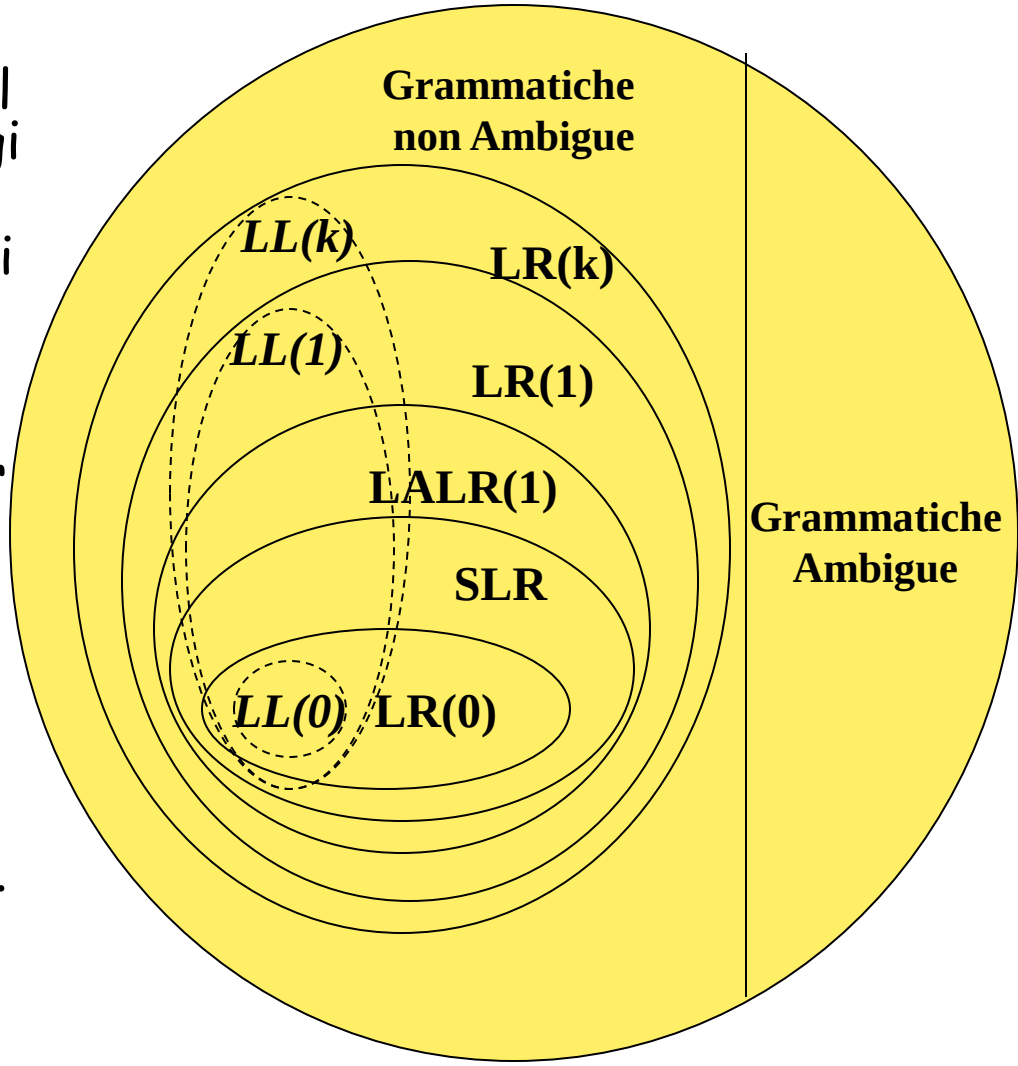
Lunedì 23 Aprile

Parser LR(0), SLR, LR(1), LALR(1): cosa hanno in comune?

- Usano azioni di "shift" e "reduce";
- Sono macchine guidate da una tabella:
  - Sono raffinamenti di LR(0)
    - Calcolano un FSA usando la costruzione basata sugli item
    - SLR: usa gli stessi "item" di LR(0).
      - Usa anche le informazioni dell'insieme FOLLOW
    - LR(1)/LALR(1): un item contiene anche informazioni date dai simboli lookahead.
      - LALR(1) è una semplificazione di LR(1) per ridurre il numero degli stati
- Consentono di definire classi di grammatiche
  - se il parser LR(0) (o SLR, LR(1), LALR(1)) calcolato dalla grammatica non ha conflitti shift/reduce o reduce/reduce, allora G è per definizione una grammatica LR(0) (o SLR, LR(1), LALR(1)).

LR Parsing

- Le grammatiche LR sono più potenti delle LL.
- LR(0) ha esclusivo interesse didattico.
- Simple LR (SLR) consentono il parsing di famiglie di linguaggi interessanti.
- La maggior parte dei linguaggi di programmazione ammettono una grammatica LALR(1).
  - Molti generatori di parser usano questa classe.
- LR(1) fornisce un parsing molto potente.
  - L'implementazione è poco controllabile.
  - Si cercano grammatiche LALR(1) equivalenti.



Simple LR parser (SLR o SLR(1))

E' un modo semplice di costruire parser più potenti di **LR(0)** *utilizzando il prossimo token di input per decidere su alcune azioni e costruire la tabella.*

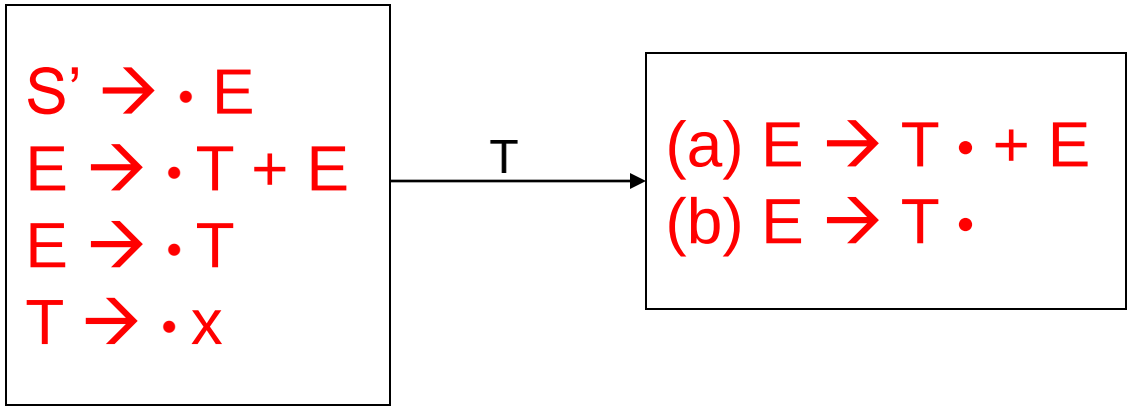
Sia s lo stato corrente:

1. se lo stato s contiene un item della forma A-> α.xβ e x è l'etichetta di una transizione uscente, allora TAB_{SRS}[s,x]=shift u, dove u è lo stato che contiene A-> αx.β.
2. se lo stato s contiene A-> γ. allora TAB_{SRS}[s,t]=reduce A-> γ per tutti i token t contenuti in FOLLOW(A). Ciò vale nel caso in cui A sia diverso da S'.
3. se lo stato s contiene S'-> S. allora TAB_{SRS}[s,$]=accept.

Le grammatiche per cui le tabella di analisi prodotte dai parser **SLR(1)** non contengono ambiguità sono dette ***grammatiche* SLR(1)**
Molte grammatiche dei linguaggi di programmazione sono **SLR(1)**

Follow(E) = { $ }

S' → E
E → T + E
E → T
T → x



Viene eliminata l'ambiguità dalla tabella poichè:

- reduce (b) sul token "$"
- shift (a) sul token "+"

Tabella SLR

(0) S' → E
(1) E → T + E
(2) E → T
(3) T → x

La riduzione avviene solo se il token successivo è un simbolo valido nella riduzione.

	x	+	$	E	T
1	s5			g2	g3
2			a		
3		s4	r2		
4	s5			g6	g3
5		r3	r3		
6			r1		

Una grammatica è SLR se e solo se per ogni stato s:

1. Per ogni item A->α.xβ con x terminale, non c'è l'item B->γ. in s con x in Follow(B).
2. Per ogni coppia di item A->α. e B->β. Follow(A) e Follow(B) sono disgiunti

Esercizi

1. Costruire la tabella SLR per la grammatica
E'-> E
E-> E+n | n

2. Costruire la tabella SLR per la grammatica
S'-> S
S-> (S)S | ε

3. Costruire la tabella SLR per la grammatica
E -> T E -> E+T T -> (E)
T -> k

Esempio di grammatica SLR

(0) E' → E
(1) E → E + T | T
(2) T→ T * F | F
(3) F → (E) | id

	id	+	*	(	)	$	E	T	F
0	s5			s4			g1	g2	g3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			g8	g2	g3
5		r6	r6		r6	r6			
6	s5			s4				g9	g3
7	s5			s4					g10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Algoritmo di parsing SLR(1)

Sia $\mathbf{TAB_{SLR}}$ la tabella **SLR(1)** e sia **u** lo stato corrente (*quello in testa alla pila*), le azioni sono:

sia "**t**" il prossimo token di input, sia "**u**" lo stato sulla pila;
azione di reduce: se $\mathbf{TAB_{SLR}}$[**u**, **t**] = **reduce k**, dove **k** rappresenta la produzione *A* -> α . Allora si rimuove la stringa α dalla pila, assieme a tutti gli stati corrispondenti (fino allo stato immediatamente prima di α) e si inserisce il non-terminale *A*.

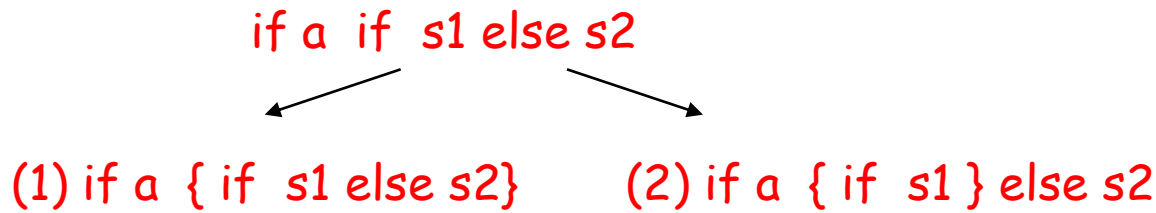
siano **u'***A* gli elementi in testa alla pila, e sia $\mathbf{TAB_{SLR}}$[**u'**,*A*]= **goto u"** , si inserisce sulla pila lo stato **u"**
altrimenti si sposta sulla pila il token "**t**" in testa al'input

azione di shift: se $\mathbf{TAB_{SLR}}$[**u**, **t**] = **shift u'** allora inserire sulla pila lo stato **u'**
altrimenti si rileva un **errore**

Esistono grammatiche non SLR

- Ogni grammatica SLR è non ambigua, ma esistono molte grammatiche non ambigue che non sono SLR.
  Per esempio:

  S->L=R
  S-> R
  L-> *R
  L-> id
  R-> L
- Puo' ammettere conflitti shift/reduce o reduce/reduce;
- Molti SR parser risolvono automaticamente i conflitti shift/reduce privilegiando lo shift al reduce (ciò incorpora la regola dell'annidamento più vicino nel problema dell'else pendente). Per esempio una versione semplificata della grammatica è:
  S-> I| other
  I-> if S | if S else S (ambiguità in corrispondenza del token else)



Privilegiare lo shift, dà l'interpretazione 1, privilegiare il reduce dà la 2

Esempio di conflitto reduce/reduce

- La seguente grammatica modella statement che possono rappresentare chiamate a procedure senza parametri o assegnazione di espressioni a variabili

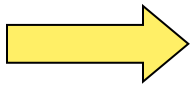
stmt->call-stmt | assign-stmt

call-stmt -> **identifier**

assign-stmt -> var := esp

var -> var **[esp]** | identifier

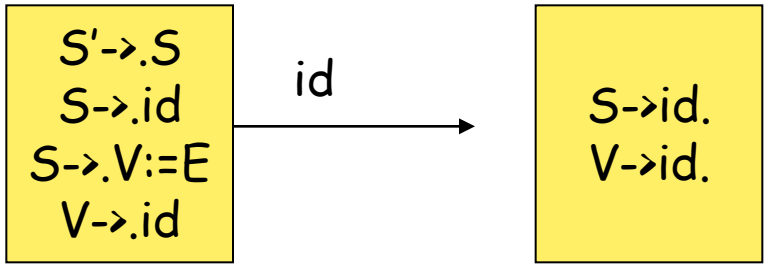
esp -> var | **number**



S -> id | V := E

V -> id

E -> V | n



Follow(S)={$} Follow(V)={:=, $}

In realtà una variabile si può trovare alla fine di un input, ma solo dopo aver trovato il token :=