ESERCIZI SU PARSER SLR

Lunedì 26 Novembre

Grammatica SLR

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

Follow(E)={+,$}

SLR	n	+	$	E
1	s2			g3
2		E->n	E->n	
3		s4	acc	
4	s5			
5		E->E+n	E->E+n	

Grammatica SLR

Costruire la tabella SLR per la grammatica
D->tL;   L->i   L->L,i   *aggiungo D'->D*
che schematizza la dichiarazione di una serie di identificatori preceduti dal tipo.
Genera il linguaggio regolare   t i (,i)*;

SLR	t	i	,	;	$	D	L
0	s1					g2	
1		s3					g4
2					acc		
3			r2	r2			
4			s5	s6			
5		s7					
6					r1		
7			r3	r3			

E' LR(0)

Grammatica SLR

Costruire la tabella SLR per la grammatica

D->tL;   L->i   L->i,L   *aggiungo D'->D*

che schematizza la dichiarazione di una serie di identificatori preceduti dal tipo.

Genera lo stesso linguaggio regolare   t i (,i)*;

SLR	t	i	,	;	$	D	L
0	s1					g2	
1		s3					g4
2					acc		
3			s5	r2			
4				s6			
5		s3					
6					r1		
7				r3			

E' LR(0)?

Parser LR(1)

Lunedì 26 Novembre

Parser LR(1)

Consente di ovviare a molte ambiguità dei parser **SLR(1)** al prezzo di una crescita sostanziale della complessità dell'algoritmo ma poco usato in pratica poiché poco efficiente:

- si preferisce il più semplice **LALR(1)** poichè è efficiente come **SLR(1)**
- è simile agli automi **LR(0)**: cambiano gli **item** e le operazioni **closure**, **goto** e **reduce**
- **Fu il primo ad essere introdotto [Knuth 1965]**

NOTA:

1. il **problema** del parser **SLR(1)** è che utilizzava il lookahead **dopo** la costruzione del DFA
2. nei parser **LR(1)** il lookahead è utilizzato **durante** la costruzione dell'automa (quindi si prendono in considerazione i simboli che veramente possano seguire un certo handle)

Costruzione della tabella LR(1)

Gli item LR(1) hanno la forma A-> $\alpha.\beta$, **t**

dove A -> $\alpha.\beta$ è un item **LR(0)** e "**t**" è un token (il lookahead) oppure **t**=$

(Ciò indica il fatto che la sequenza α si trova in cima alla pila e che alla testa dell'input c'è la stringa derivabile da βt).

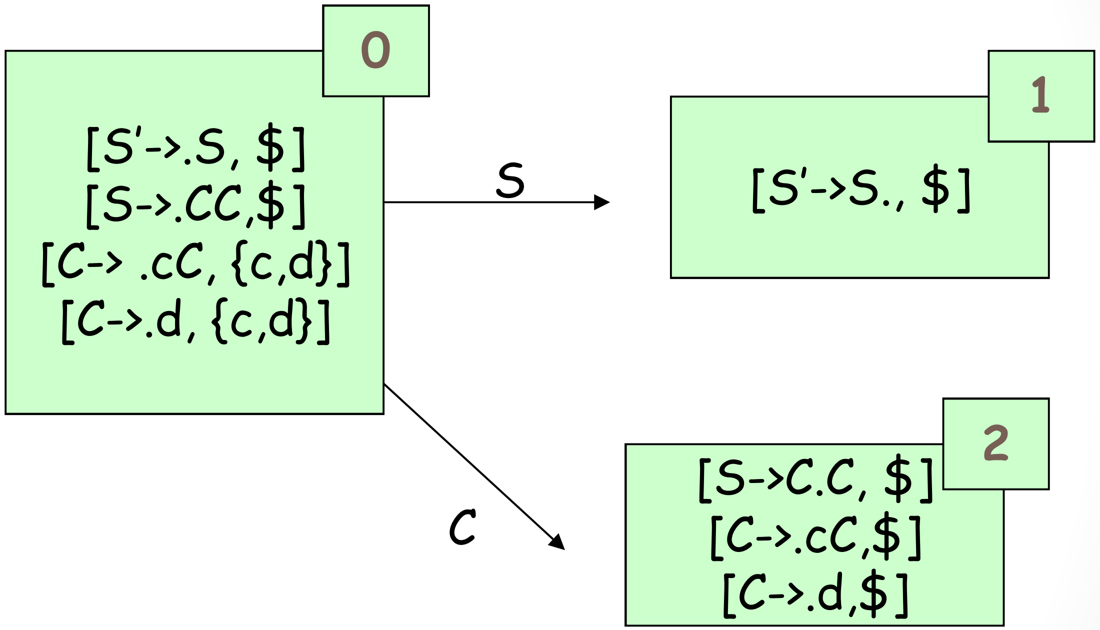
```
Function Closure(I);
begin
   J:=I;
    repeat
       for each item [A-> α.Xβ,z ] in J
         for each  X->γ
           for each w∈FIRST(βz)
              add [X->.γ,w ] to J;
    until no more items can be added to J;
    return J;
end
```

Esempio:
S'-> S
S-> CC
C-> cC|d

Closure ([S'->.S, $])=
{[S'->.S, $], [S->.CC,$]
[C-> .cC, {c,d}],
[C->.d, {c,d}] }

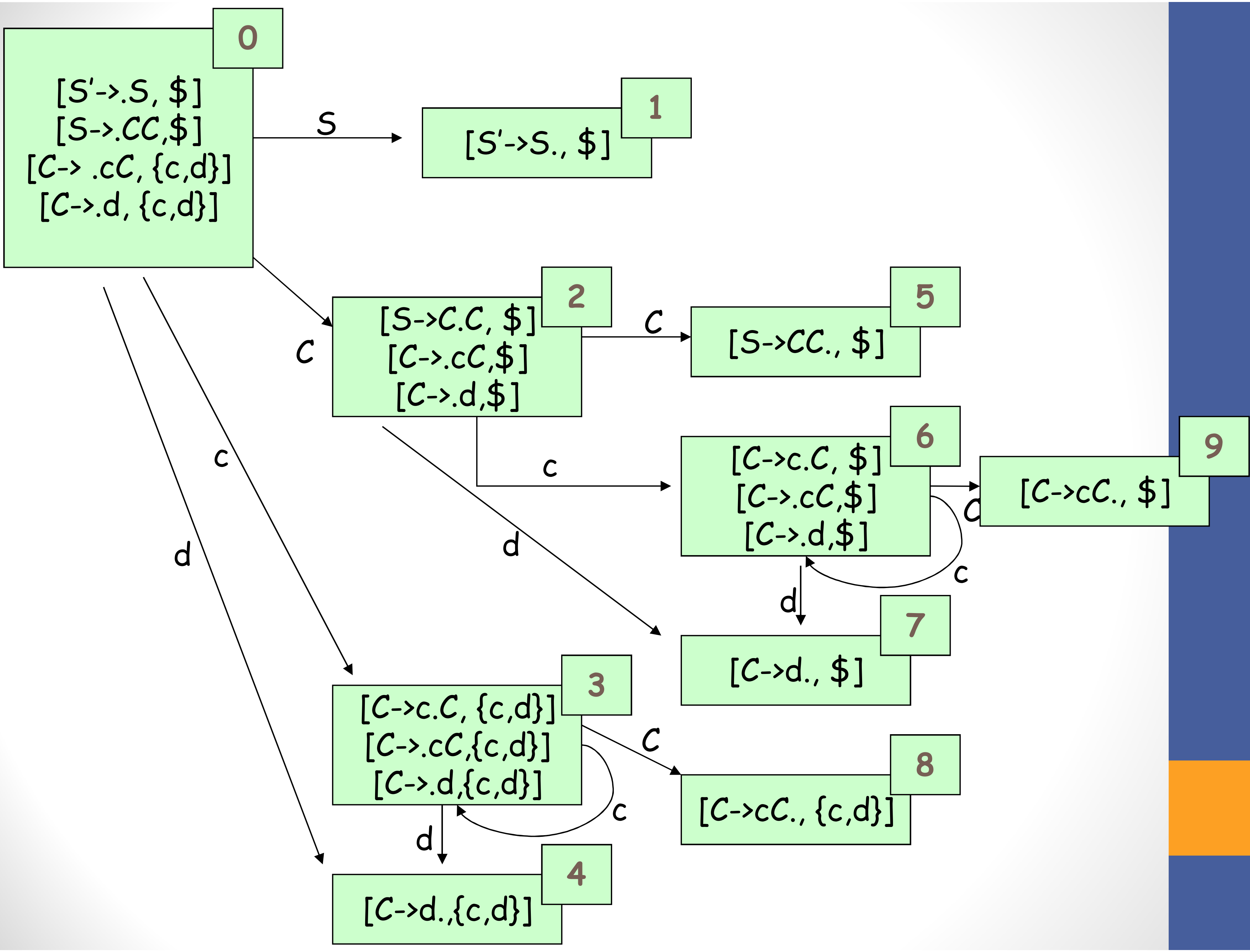
Costruzione della tabella LR(1)

Esempio:
S'-> S
S-> CC
C-> cC|d



Function Goto(I, X);
begin
  J := insieme degli item [A-> $\alpha X.\beta$,a] tali che
  [A-> $\alpha.X\beta$,a] è in I;
  return CLOSURE(J);
end

E così via…


0
[S'->.S, $]
[S->.CC,$]
[C-> .cC, {c,d}]
[C->.d, {c,d}]
S
1
[S'->S., $]
C
2
[S->C.C, $]
[C->.cC,$]
[C->.d,$]
C
5
[S->CC., $]
c
c
6
[C->c.C, $]
[C->.cC,$]
[C->.d,$]
C
9
[C->cC., $]
c
d
d
d
7
[C->d., $]
3
[C->c.C, {c,d}]
[C->.cC,{c,d}]
[C->.d,{c,d}]
C
8
[C->cC., {c,d}]
c
d
4
[C->d.,{c,d}]

Costruzione della tabella LR(1)

La tabella è strutturalmente simile a quella **LR(0)**

- **azioni shift**: se dallo stato **s** esiste una transizione "**t**" ("**t**" **simbolo terminale**) nello stato **s'**, inserire "**shift s'**" in corrispondenza di (**s** , "**t**")
- **azioni goto**: se dallo stato **s** esiste una transizione "S" ("S" **simbolo non-terminale**) in **s'**, inserire "**goto s'**" in corrispondenza di (**s** , "S")
- **azioni reduce**: se lo stato **s** contiene un item **LR(1)** del tipo [X -> γ**.** , **t], con t simbolo terminale e X diverso da S'** e "**k**" identifica la produzione "X -> γ", allora inserire "**reduce k**" in corrispondenza di (**s** , "**t**")
- **azione accept:** se lo stato **s** contiene l'item [S'-> S., $], allora inserire "**accept"** in corrispondenza di (**s, "$"**)

osservazione: il parser **LR(1)** ridurrà **soltanto** quando il simbolo in testa all'input sarà "**t**"

Lo stato iniziale è quello costruito da [S'->.S, $]

Esercizi

- Creare la tabella LR(1) per la grammatica
  S->L=R
  S-> R
  L-> *R
  L-> id
  R-> L

Parsing LALR(1)

Le tabelle di analisi **LR(1)** sono di solito di **ordini di grandezza maggiori** di quelle **SLR(1)**:

se una tabella **SLR(1)** di un linguaggio di programmazione è intorno ai 10 KB, una tabella **LR(1)** dello stesso linguaggio è intorno ai MB, con il doppio della memoria per costruirla.

osservazione: se consideriamo l'automa **LR(1)** della grammatica S'->S, S->CC, C->cC|d e ignoriamo i lookahead, alcune coppie di stati sono identici:

gli stati 8 e 9,

gli stati 4 e 7,

gli stati 3 e 6,

Il parser **LALR(1)** consiste nell'identificare questi stati, combinando i loro lookahead, con l'obiettivo di ottenere un DFA **LR(1)** **simile** al DFA **LR(0).**

Infatti:

1. Le prime componenti degli item LR(1) sono item LR(0)
2. Se due stati s e t LR(1) hanno la stessa prima componente e se da s esce una transizione con X verso lo stato s', allora anche da t uscirà una transizione con X verso t' e t e t' avranno le stesse prime componenti

Condizioni LALR(1)

- Nota che una grammatica soddisfa la condizione LALR(1) se valgono entrambe le condizioni:
  - Ogni candidata di riduzione ha un insieme di prospezione disgiunto dalle etichette terminali uscenti;
  - Se vi sono due candidate di riduzione i loro insiemi di prospezione sono disgiunti;

Grammatica LR(1)

- Sia data la grammatica

S->aXb

S->bXc

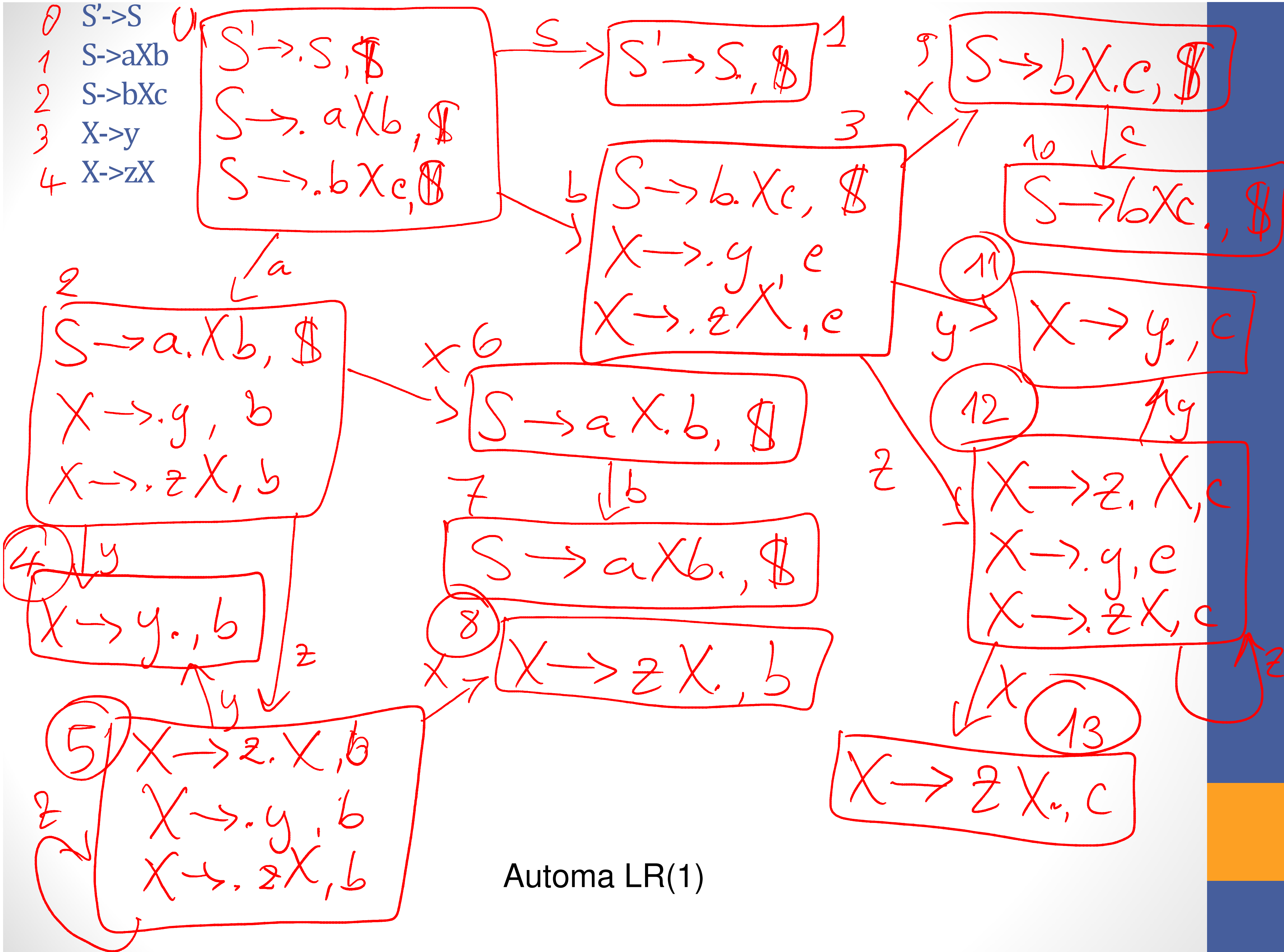
X->y

X->zX

Costruire l'automa LR(1).

Sugg. Ha 14 stati e 3 coppie di stati possono essere fusi, dando luogo all'automa LALR(1) con 11 stati.

0 S'->S
1 S->aXb
2 S->bXc
3 X->y
4 X->zX



Automa LR(1)

Tabella

LR(1)	a	b	c	y	z	$	S	X
0	s2	s3					g1	
1						acc		
2				s4	s5			g6
3				s11	s12			g9
4		r(X->y)						
5				s4	s5			g8
6		s7						
7						r(S->aXb)		
8		r(X->zX)						
9			s10					
10						r(S->bXc)		
11			r(X->y)					
12				s11	s12			g13
13			r(X->zX)					

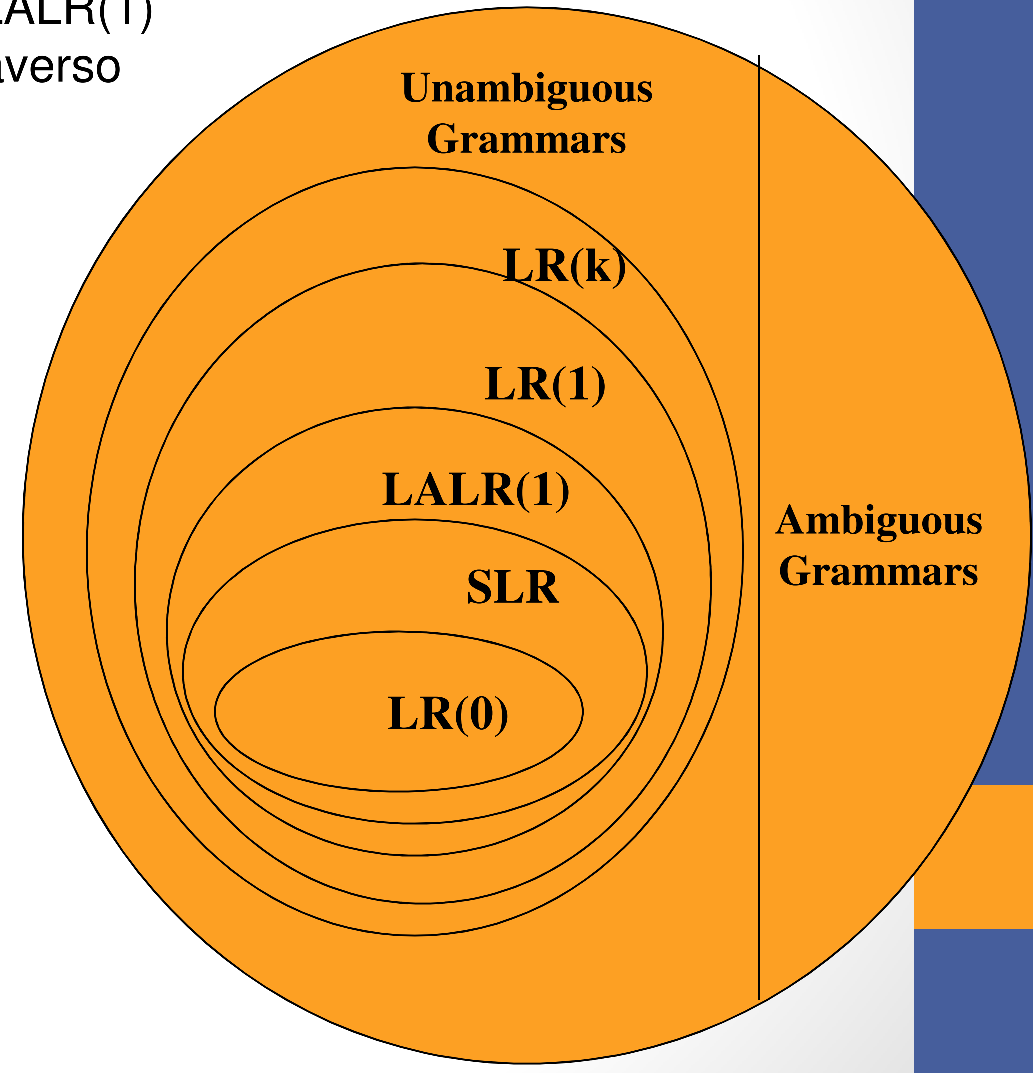
Esempio LR(1) ma non LALR(1)

S->A | Ba |bAa |bB

A->a

B->a

• Un parsing LALR(1) potrebbe generare conflitti che il LR(1) corrispondente non genererebbe (ciò non accade in pratica).
• Si dimostra che se una grammatica è LR(1), la tabella LALR(1) non può avere conflitti shift/reduce ma solo reduce/reduce.
• E' possibile computare il DFA del LALR(1) direttamente dal DFA del LR(0) attraverso un processo chiamato **lookahead propaganti**



Regole per risolvere l’ambiguità

- Spesso può essere comodo usare grammatiche ambigue ed usare delle regole per risolvere l’ambiguità.
- Precedenza ed associatività
  E->E+E | E*E | (E) | id   (è ambigua poiché non specifica la precedenza e l’associatività tra gli operatori)

La grammatica non ambigua equivalente è:

E->E+T | T

T-> T * F | F

F -> (E) | id

- La prima è preferibile perché:
  - si possono cambiare precedenza e associatività senza cambiare le produzioni
  - il parser con meno produzioni è più veloce.
- Stabilire delle regole di precedenza e di associatività fa risolvere al parser i conflitti.
- Un altro esempio riguarda l’ambiguità del “dangling-else”.

Trattamento delle grammatiche ambigue per risolvere l'ambiguità

- Le grammatiche ambigue non sono certamente LR(1)
- In alcuni casi si sa come trattare l'ambiguità
- Costruiamo la tabella SLR di

S'->S
S->iSeS
S->iS
S->a

- Si predilige lo shift

SLR	i	e	a	$	S
0	s2		s3		g1
1				acc	
2	s2		s3		
3		r3		r3	
4		s5 r2			g6
5	r3	r3		r3	
6	r1	r1		r1	

Proprietà dei linguaggi e delle grammatiche LR(k)

- La famiglia dei linguaggi verificabili da parser deterministici coincide con quella dei linguaggi generati dalle grammatiche LR(1).
  - Ciò non significa che ogni grammatica il cui linguaggio è deterministico, sia necessariamente LR(1): potrebbe essere ambigua o richiedere una prospezione di lunghezza k>1; esisterà una grammatica equivalente LR(1);

- La famiglia dei linguaggi generati dalle grammatiche LR(k) coincide con quella dei linguaggi generati da LR(1). Quindi un linguaggio context free ma non-deterministico non può avere una grammatica LR(k).

- Per ogni k>=1, esistono grammatiche LR(k) ma non LR(k-1).

- Data una grammatica, è indecidibile se esista un k>0 per cui tale grammatica risulti LR(k); di conseguenza non è decidibile se il linguaggio generato da una grammatica CF è deterministico. E' decidibile soltanto se k è fissato.

Considerazioni su linguaggi e grammatiche LL(k) e LR(k):

- Ogni linguaggio regolare è LL(1);
-- Ogni linguaggio LL(k) è deterministico, ma vi sono linguaggi deterministici per cui non esiste alcuna grammatica LL(k);

-Per ogni k>=0, una grammatica LL(k) è anche LR(k);
-Le grammatiche LL(1) e LR(0) non sono incluse una nell'altra;
-Quasi tutte le grammatiche LL(1) sono LALR(1)