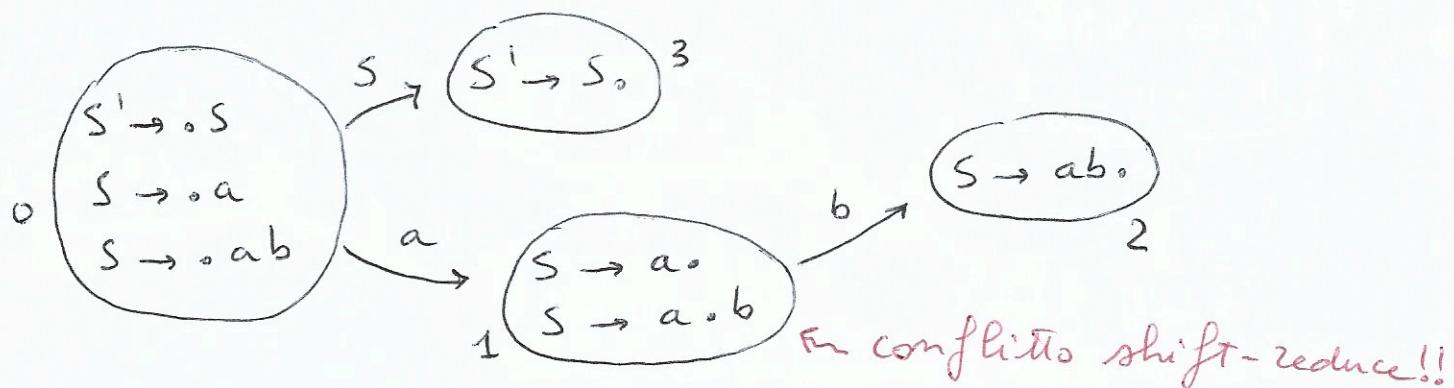


Ma non sempre siamo così fortunati...

Una grammatica libera G può non essere LR(0)!

$$(1) S' \rightarrow S \quad (2) S \rightarrow a \quad (3) S \rightarrow ab \quad L(G) = \{a, ab\}$$



	a	b	\$	S
0	$s_1$			$g_3$
1	$r_2$	$r_2/s_2$	$r_2$	
2	$r_3$	$r_3$	$r_3$	
3			acc	

Tabelle di parsing LR(0)

Come risolvere il conflitto? Usiamo il look-ahead!

Cioè guardiamo il  $\text{Follow}(S)$ : se  $b \in \text{Follow}(S)$ , allora il conflitto è reale! Altrimenti, no!

(può essere)

Ma  $\text{Follow}(S) = \{\$\}$   $\Rightarrow$  risolviamo il conflitto a favore dels shift

	a	b	\$	S
0	$s_1$			$g_3$
1		$s_2$	$r_2$	
2			$r_3$	
3			acc	

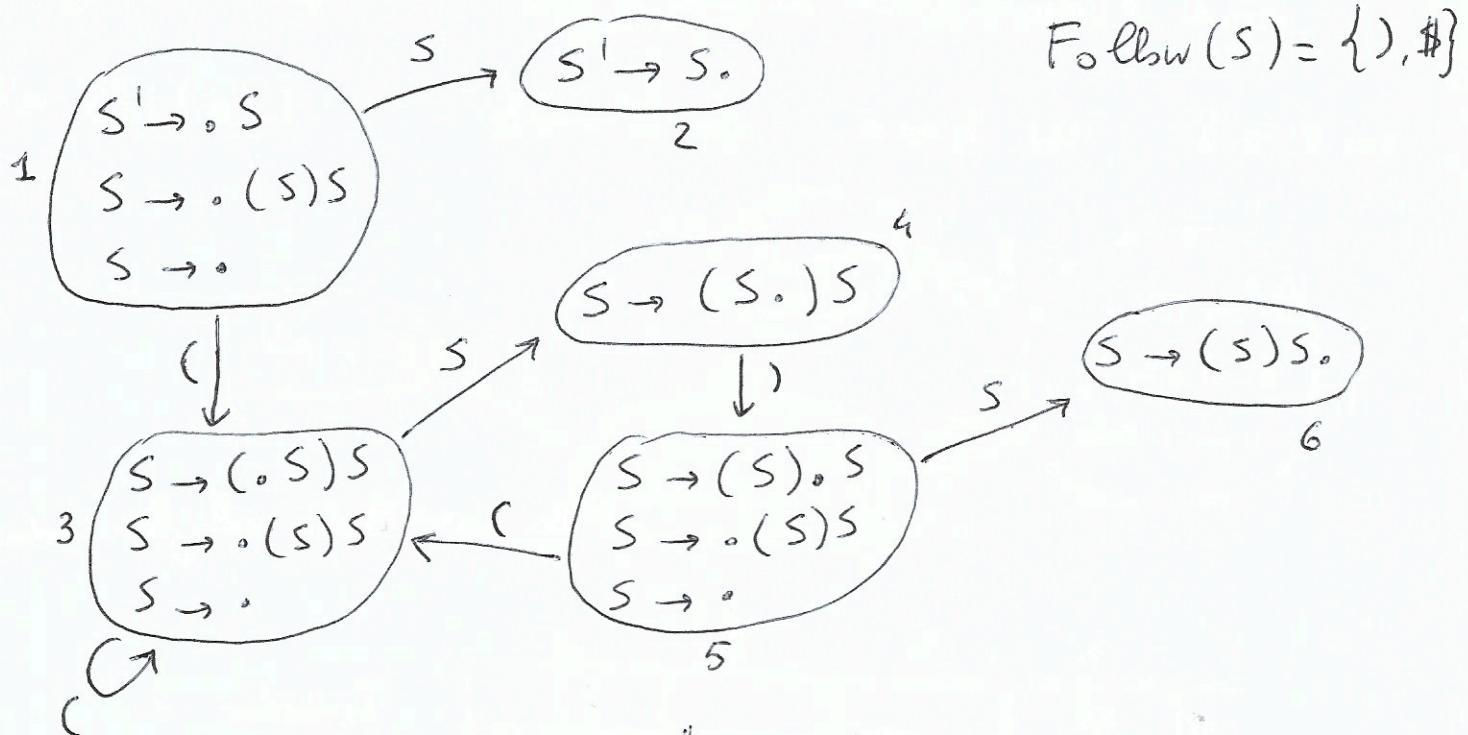
Tabelle di parsing SLR(1)  
↑  
simple

- solo per i caratteri nel  $\text{Follow}(S)$  mettiamo  $r_2$

# Esempio : Parser Bilanciato

(24)

- (0)  $S' \rightarrow S$
- (1)  $S \rightarrow (S)S$
- (2)  $S \rightarrow \epsilon$



	(	)	\$	S
1	$S_3/22$	22	22	g2
2			acc	
3	$S_3/22$	22	22	g4
4		s5		
5	$S_3/22$	22	22	g6
6	r1	r1	r1	

Tabella di parsing LR(0)  
(con 3 conflitti)

	(	)	\$	S
1	$S_3$	22	22	g2
2			acc	
3	$S_3$	22	22	g4
4		s5		
5	$S_3$	22	22	g6
6		r1	r1	

Tabella di parsing SLR(1)  
(senza conflitti)

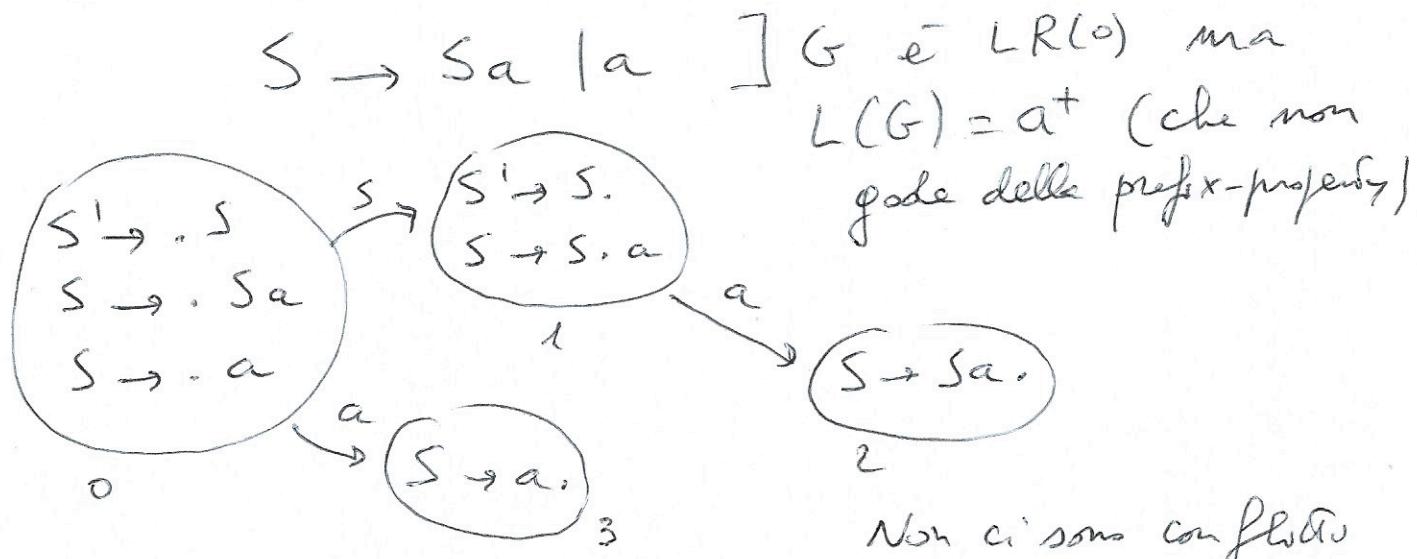


Ge SLR(1)  
(ma non LR(0))

OSS:

(25)

- (1) Se G ha produzioni ε, allora G difficilmente è LR(0)  
(caso banale in cui uno stato con item  $A \rightarrow \cdot$  non ha item del tipo  $B \rightarrow \cdot a \cdot \beta$ )
- (2) Se L è libero deterministico e gode della prefix-property ("L gode della prefix-property se  $\exists x, y \in L$  tale che  $x$  è prefisso di  $y$ "),  
allora L è LR(0)  
Quindi, se L è libero det. ma non è LR(0),  
allora L non gode della prefix-property
- (3) Se L è LR(0) ed è finito, allora gode della prefix-property. Ovvvero se L è finito e non gode delle prefix-properties, allora L non è LR(0)  
 $L = \{a, ab\}$
- (4) Se L è LR(0) ma è infinito, può non godere della prefix-property



Esercizio

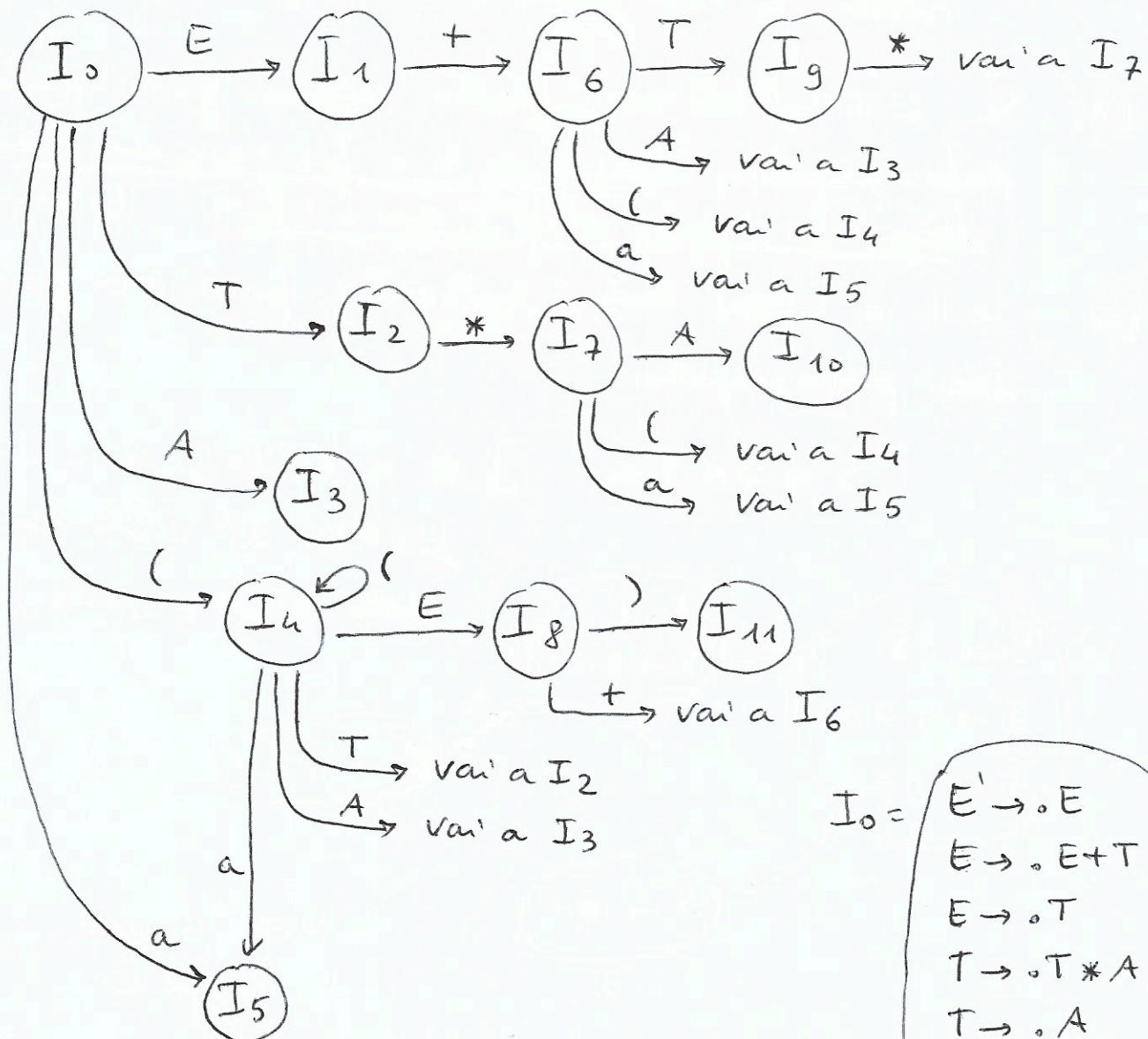
$S \rightarrow Sa | \epsilon$   
è LR(0)!

## Esercizio "corposo"

(26)

- (0)  $E' \rightarrow E$
- (1)  $E \rightarrow E + T$
- (2)  $E \rightarrow T$
- (3)  $T \rightarrow T * A$
- (4)  $T \rightarrow A$
- (5)  $A \rightarrow a$
- (6)  $A \rightarrow (E)$

- Costruire l'automa canonico LR(0) (a partire da  $C_{\text{BS}}(\{E' \rightarrow .E\}) = I_0$  (ci sono 12 stati))
- costruire la tabella di parsing LR(0)
- verificare se ci sono conflitti
- eventualmente, modificare la tabella di parsing in SLR(1)



$I_0 =$

$E' \rightarrow .E$
$E \rightarrow .E + T$
$E \rightarrow .T$
$T \rightarrow .T * A$
$T \rightarrow .A$
$A \rightarrow .a$
$A \rightarrow .(E)$

## Tabella di Parsing SLR(1)

(27)

- colonne:  $T \cup \{\$\}$  <sup>simple</sup>  $UNT$
- righe: stati dell'automa canonico  $LR(0)$

Come si riempie la tabella?

Per ogni stato  $s$  dell'automa  $LR(0)$

1. se  $x \in T$  e  $s \xrightarrow{x} t$ , inserisci shift t in  $M[s, x]$
2. se  $A \rightarrow d_0 \in s$  e  $A \neq S'$ , inserisci reduce  $A \rightarrow d$  in  $M[s, x]$  per tutti gli  $x \in \text{Follow}(A)$
3. se  $S' \xrightarrow{*} S_0 \in s$ , inserisci Accept in  $M[s, \$]$
4. se  $A \in NT$  e  $s \xrightarrow{A} t$ , inserisci goto t in  $M[s, A]$

- prima, per  $LR(0)$ , era "per tutti gli  $x \in T \cup \{\$\}$ "
- Effetto: limitare l'uso della reduce solo a casi plausibili!

SLR(1): S - simple

L - left-to-right

R - rightmost derivation

1 - un simbolo di look-ahead  
(in modo non esplicito, ma attraverso i follow dei nonterminali)

Vedremo che  $LR(1)$  usa esplicitamente il look-ahead già nella definizione di item  $LR(1)$

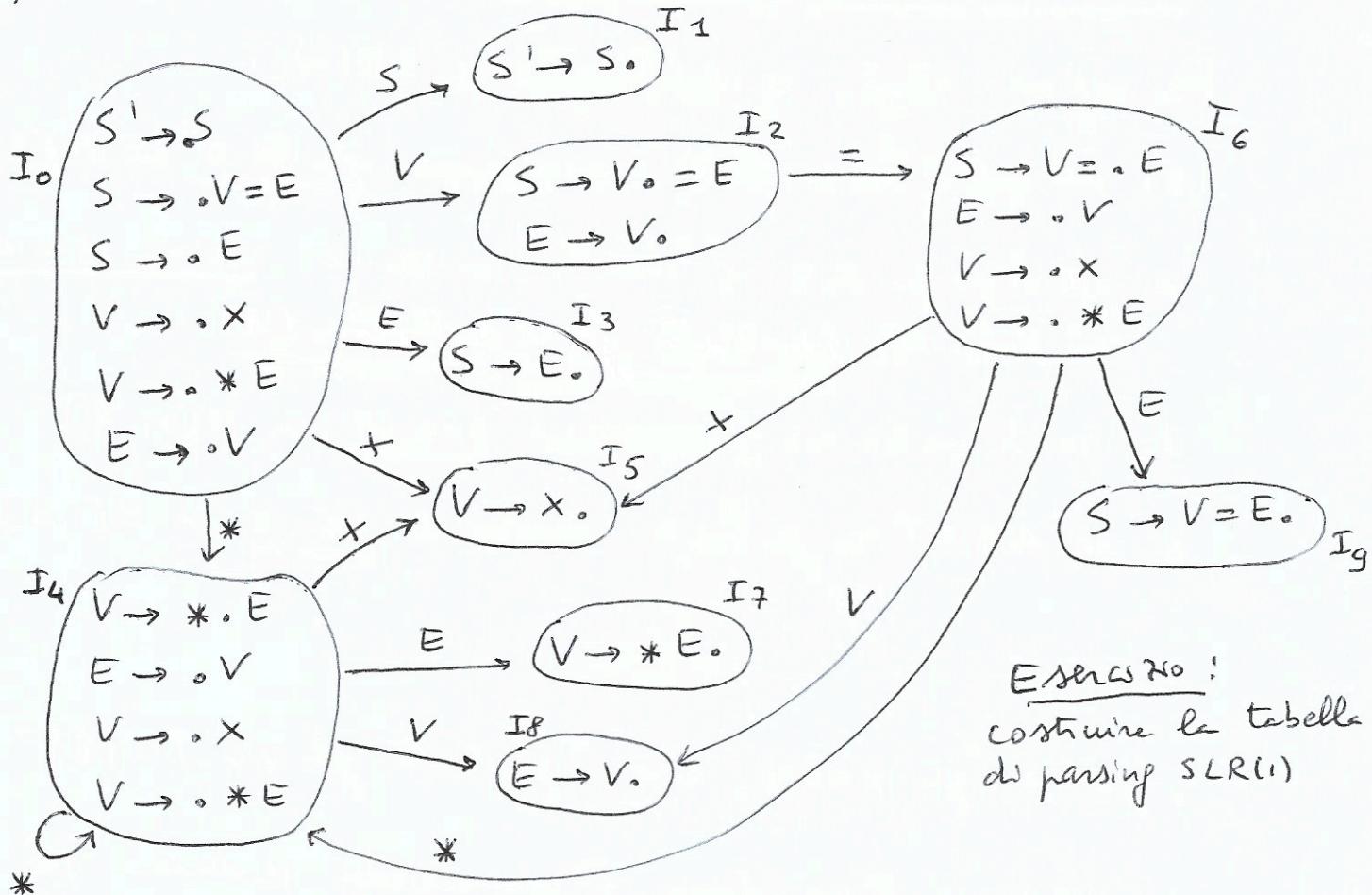
Ma non sempre siamo così fortunati...

(28)

Una grammatica libera G potrebbe non essere nemmeno  $SLR(1)$ !

(0)  $S \xrightarrow{!} S$     (1)  $S \xrightarrow{} V = E$     (2)  $S \xrightarrow{} E$     (3)  $E \xrightarrow{} V$

(4)  $V \xrightarrow{} X$     (5)  $V \xrightarrow{} *E$



Consideriamo  $I_2$

$$\begin{cases} S \xrightarrow{\cdot} V = E \\ E \xrightarrow{\cdot} V. \end{cases}$$

confitto shift/reduce  
perché  $= \in \text{Follow}(E)$

(a causa di  $V \xrightarrow{\cdot} *E$ ,  
 $\text{Follow}(V) \subseteq \text{Follow}(E)$ )

Oss: Se noi sappiamo con certezza che il carattere successivo è davvero " $=$ ", allora la reduce  $E \xrightarrow{\cdot} V.$  non va considerata, perché non è mai possibile derivare  $S \Rightarrow^* E = \dots$

$\Rightarrow$  ha senso solo fare lo shift!

Oss:  $\gamma \in \text{Follow}(E)$  vuol dire che esiste una derivazione tale che  $S \Rightarrow^* \gamma E = \beta$ , ma a noi serve sapere se

$$S \Rightarrow^* E = \beta$$

perché questa è la derivazione "corrente"

N.B.  $S \Rightarrow V = E \Rightarrow * E = E$

per cui effettivamente  $\gamma \in \text{Follow}(E)$ , ma è impossibile derivare

$$S \Rightarrow^* E = \dots$$

per cui nello stato  $I_2$ , se un input c'è  $=$ , devo sicuramente fare lo shift a  $I_6$ .

Da questo esempio, si capisce che bisogna ridurre i casi in cui si possa applicare una "reduce" a casi ancora più plausibili di quanto dica il  $\text{Follow}(E)$ !

$\Rightarrow$  item  $\underline{\text{LR}(1)}$ : una coppia formata da

- un item  $\text{LR}(0)$  (detto nucleo o core)
- un simbolo di look-ahead in  $T \cup \{\#\}$

Esempio

$I_2$        $[S \rightarrow V_0 = E, \#]$   
 $[E \rightarrow V_0, \#]$

- stato dell'automa canonico  $\text{LR}(1)$
- se leggo  $=$ , allora shift
- se leggo  $\#$ , allora reduce  $E \rightarrow V$

Intuizione: se l'automa canonico  $LR(1)$  (30)  
(che definiremo tra poco) è in uno stato che contiene  
l'item  $LR(1) [A \rightarrow \alpha \cdot \beta, X]$

- sta cercando di riconoscere la manifera  $\alpha \beta$ ;
- di essa,  $\alpha$  è già sulla pila;
- sull'input si aspetta una stringa derivabile  
da  $\beta X$

• cioè  $X$  può davvero essere fatto  
in questa derivazione

(quando uso il  $Follow(A)$  nelle  $SLR(1)$ ,  
so solo che esiste una derivazione  
che produce  $X \in Follow(A)$ , ma non è  
detto che sia proprio quella che  
sto esaminando!)

•  $[A \rightarrow \alpha \cdot, X]$

⇒ farà la reduce  $A \rightarrow \alpha$  se il prossimo input  
è proprio  $X$ .

Item  $LR(1)$

usiamo esplicitamente i caratteri in  
avanti dell'input, associato direttamente  
all'item  $LR(0)$  - core

## NFA LR(1)

- stati: item LR(1) della grammatica aumentata
- $[S' \rightarrow . S, \#]$  è lo stato iniziale
- dallo stato  $[A \rightarrow \alpha . X \beta, a]$  c'è una transizione allo stato  $[A \rightarrow \alpha X . \beta, a]$  etichettata  $X$ , per  $X \in T \cup NT$
- dallo stato  $[A \rightarrow \alpha . X \beta, a]$ , per  $X \in NT$  e per ogni produzione  $X \rightarrow \gamma$ , c'è una  $\epsilon$ -transizione verso lo stato  $[X \rightarrow . \gamma, b]$  per ogni  $b \in \text{First}(\beta)$   
 $(N.B. \text{First}(\beta) \subseteq T \cup \{\$\})$

## Automa Canonico LR(1)

Si può ottenere in 2 modi:

- DFA ottenuto da NFA LR(1) con la costruzione dei sottoinsiemi
- In modo diretto, usando le funzioni  $\text{clos}(I)$  e  $\text{Goto}(I, X)$   
 partendo dallo stato iniziale  
 $\text{clos}([S' \rightarrow . S, \$])$

$\text{Clos}(I) \{$

rifletti finché  $I$  è modificato {  
 per ogni item  $[A \rightarrow \alpha \cdot X\beta, a] \in I$   
 per ogni produzione  $X \rightarrow \gamma$   
 per ogni  $b \in \text{First}(\beta a)$   
 aggiungi  $[X \rightarrow \cdot \gamma, b]$  a  $I$ ;  
 }

return  $I$ ;

}

$\text{Goto}(I, X) \{$

inizializza  $J = \emptyset$ ;

per ogni item  $[A \rightarrow \alpha \cdot X\beta, a] \in I$   
 aggiungi  $[A \rightarrow \alpha X \cdot \beta, a]$  a  $J$ ;

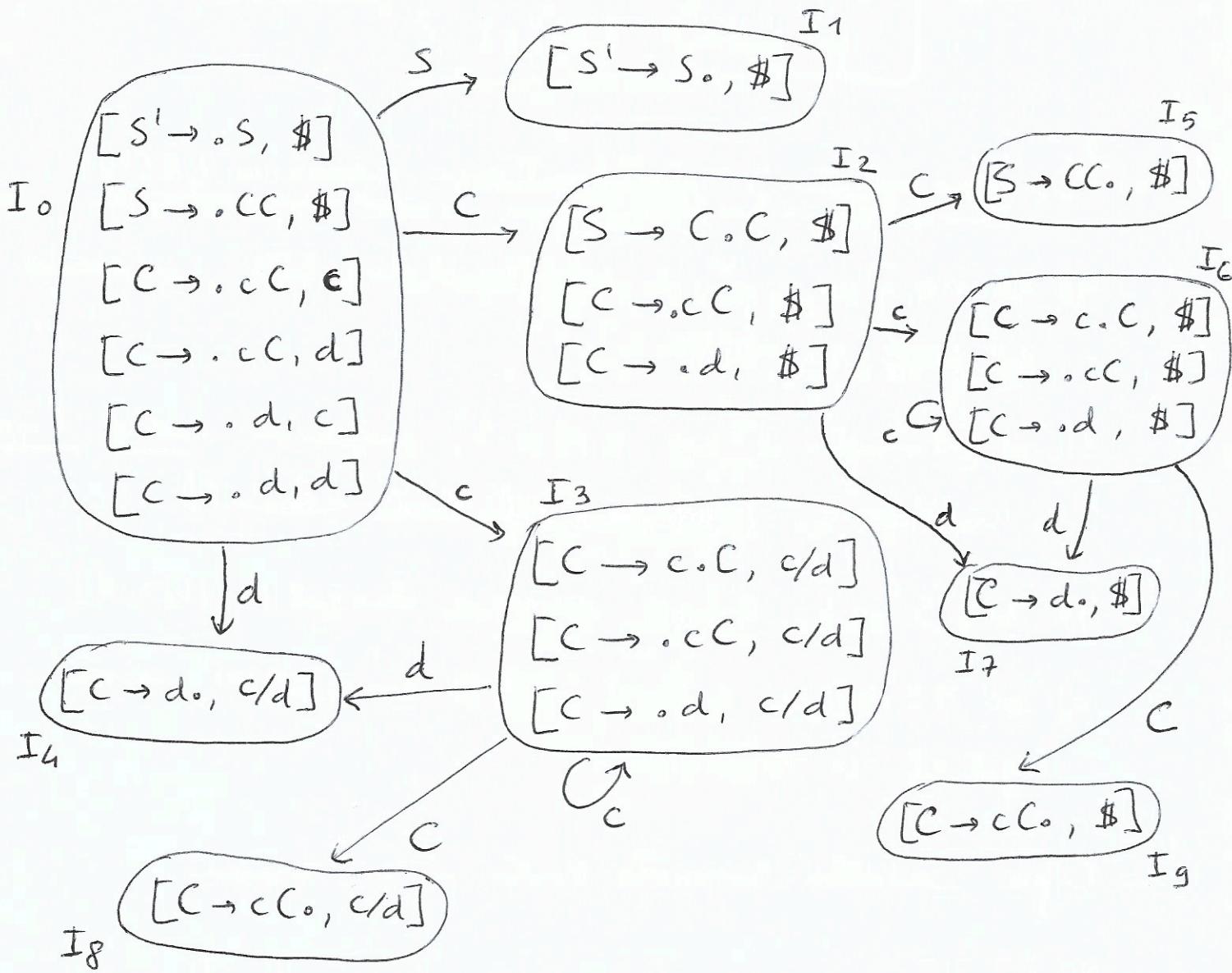
return  $\text{Clos}(J)$ ;

Stato iniziale dell'automa canonico LR(1)

è  $\text{Clos}([S^* \rightarrow \cdot S, \$])$

OSS:  $\text{Goto}(I, X)$  non considera il look-ahead, cioè  
 agisce solo sulla parte "core/LR(0)" dell'item LR(1),  
 (salvo poi effettuare  $\text{Clos}(J)$  in chiusura)

(0)  $S' \rightarrow S$  (1)  $S \rightarrow CC$  (2)  $C \rightarrow cC$  (3)  $C \rightarrow d$  (33)



Esercizio: costruire l'automa canonico LR(0) per questa grammatica. Osserverete che ci sono meno stati!

E anche che non ci sono conflitti...

(In questa semplice grammatica, non serve usare la tecnica LR(1))

Come si riempie la tabella di parsing LR(1) ?

Per ogni stato  $s$  dell'automa canonico  $LR(1)$

- 1) se  $x \in T$  e  $s \xrightarrow{x} t$  nell'automa  $LR(1)$ , inserisci shift t in  $M[s, x]$
  - 2) se  $[A \rightarrow \alpha_0, x] \in s$  e  $A \neq S'$ , inserisci reduce A  $\rightarrow$   $\alpha$  in  $M[s, x]$  (solo per  $x$  del look-ahead!)
  - 3) se  $[S' \rightarrow s_0, \$] \in s$ , inserisci Accept in  $M[s, \$]$
  - 4) se  $A \in NT$  e  $s \xrightarrow{A} t$  nell'automa  $LR(1)$ , inserisci goto  $t$  in  $M[s, A]$
- Ogni casella rimasta vuota è un errore
  - Def. Una grammatica libera  $G$  è di classe  $LR(1)$  se ogni casella della sua tabella di parsing  $LR(1)$  ha al più un elemento (no conflitti)

	c	d	\$	s	C	
0	s3	s4		g1	g2	
1			acc			
2	s6	s7			g5	
3	s3	s4			g8	
4	r3	r3				
5			r1			
6	s6	s7			g9	
7			r3			
8	r2	r2				
9			r2			

- (0)  $S' \rightarrow S$
- (1)  $S \rightarrow CC$
- (2)  $C \rightarrow cC$
- (3)  $C \rightarrow d$

Tabella di parsing LR(1)  
ottenuta dall'automa LR(1)  
di pagina 33

- La Tabella SLR(1) di questa grammatica ha solo 7 stati, anches 10, ed è pure senza conflitti
- Vedremo che questa grammatica è (ovviamente) anche LALR(1)

Ricordiammo la grammatica NON SLR(1)

- |                        |                           |                        |
|------------------------|---------------------------|------------------------|
| (0) $S' \rightarrow S$ | (1) $S \rightarrow V = E$ | (2) $S \rightarrow E$  |
| (3) $E \rightarrow V$  | (4) $V \rightarrow x$     | (5) $V \rightarrow *E$ |

e vediamo se è LR(1)!

(36)

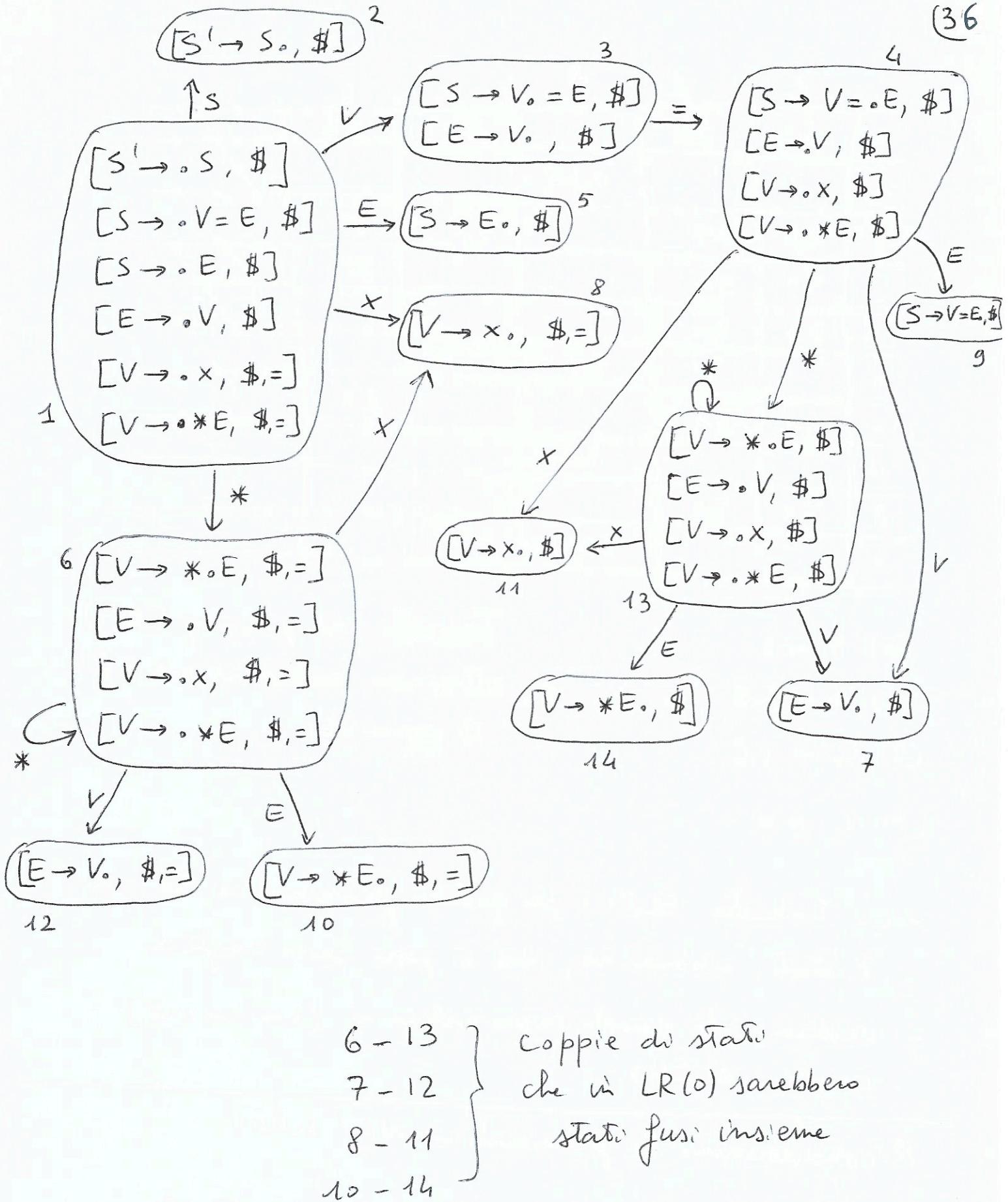


Tabelle du parsing LR(1)

	$X$	*	=	\$	S	E	V
1	$S_8$	$S_6$			$g_2$	$g_5$	$g_3$
2				acc			
3				$\tau_4$	$\tau_3$		
4	$S_{11}$	$S_{13}$					$g_9$ $g_7$
5					$\tau_2$		
6	$S_8$	$S_6$					$g_{10}$ $g_{12}$
7					$\tau_3$		
8				$\tau_4$	$\tau_4$		
9					$\tau_1$		
10				$\tau_5$	$\tau_5$		
11					$\tau_4$		
12				$\tau_3$	$\tau_3$		
13	$S_{11}$	$S_{13}$					$g_{14}$ $g_7$
14					$\tau_5$		

- (0)  $S^1 \rightarrow S$       (1)  $S \rightarrow V = E$       (2)  $S \rightarrow E$   
 (3)  $E \rightarrow V$       (4)  $V \rightarrow X$       (5)  $V \rightarrow * E$

Non avions conflit  
 $\Rightarrow G \text{ est LR}(1) !$

Parser      LALR(1)  
look-ahead

- LR(1): tabelle di parsing molto grandi  
(centinaia di migliaia di stati per lunghezza media grande)
- LALR(1): buon compromesso tra semplicità (e compattatezza) di SLR(1) e selettività di LR(1)

Come si ottiene il parser LALR(1)?

Osserva che:

- (1) Nucleo di uno stato LR(1):  
insieme di item LR(0) ottenuto dimenticando i look-ahead dagli item LR(1)
- (2) Nucleo di stato LR(1) = stato dell'automa LR(0)
- (3) Le transizioni dell'automa LR(1) dipendono solo dal nucleo: la funzione  $\text{Goto}(I, X)$  usa da  $I$  solo la parte "nucleo"/LR(0) dell'item LR(1)

⇒ La tabella di parsing LALR(1) si ottiene da quella LR(1) fondendo insieme gli stati con lo stesso nucleo

- tante righe quante gli stati dell'automa LR(0)
- meno "reduce" della tabella SLR(1)

Riprendiamo l'esempio di pagina 33+35 (39)

Tabella LR(1)

	c	d	\$	S	C
0	$S_3$	$S_4$		$g_1$	$g_2$
1			acc		
2	$S_6$	$S_7$			$g_5$
3	$S_3$	$S_4$			$g_8$
4	$z_3$	$z_3$			
5			$z_1$		
6	$S_6$	$S_7$			$g_9$
7			$z_3$		
8	$z_2$	$z_2$			
9			$z_2$		

- G [ (0)  $S^* \rightarrow S$   
 (1)  $S \rightarrow CC$   
 (2)  $C \rightarrow cC$   
 (3)  $C \rightarrow d$

Guardando

l'automa LR(1) a  
 pg. 33, si vede  
 che  $3-6$   
 $4-7$   
 $8-9$

possono essere fusi !!

Tabella LALR(1)

	c	d	\$	S	C
0	$S_{36}$	$S_{47}$		$g_1$	$g_2$
1			acc		
2	$S_{36}$	$S_{47}$			$g_5$
36	$S_{36}$	$S_{47}$			$g_{89}$
47	$z_3$	$z_3$	$z_3$		
5			$z_1$		
89	$z_2$	$z_2$	$z_2$		

Ge LALR(1)  
 perché non ci  
 sono conflitti

$$I_{36} = I_3 \cup I_6 = \{ [C \rightarrow c.C, c/d/\$], \\ [C \rightarrow .cC, c/d/\$], \\ [C \rightarrow .d, c/d/\$] \}$$

Se G è LR(1) e anche LALR(1),

allora ... ↴

L'automa  $LR(1)$  e quello  $LALR(1)$  si mimano perfettamente su input corretti. (40)

Su input erronei,  $LALR(1)$  può fare qualche riduzione in più prima di accorgersi dell'errore.

Ad esempio, per la G della pagina 39, su input  $ccd\$$

$LR(1)$      $(\emptyset, \epsilon, ccd\$)$

$(03, c, cd\$)$

$(033, c\cancel{a}, d\$)$

$(0334, ccd, \$)$  e trova errore perché

$$M[4, \$] = \text{"bianca"}$$

$LALR(1)$      $(\emptyset, \epsilon, ccd\$)$

$(036, c, cd\$)$

$(03636, cc, d\$)$

$(03636\overset{\substack{\downarrow \\ \text{goto 89}}}{47}, \overset{\substack{\downarrow \\ \text{c}}}{cc}, \$)$  che non è bloccato, ma

$(03636\overset{\substack{\downarrow \\ \text{goto 89}}}{89}, \overset{\substack{\downarrow \\ \text{C}}}{cc}, \$)$

$(036\overset{\substack{\downarrow \\ \text{goto 2}}}{89}, \overset{\substack{\downarrow \\ \text{C}}}{cc}, \$)$

$(02, c, \$)$  e finalmente trova errore perché  $M[2, \$] = \text{"bianca"}$

L'automa  $LALR(1)$  non fa mai degli shift in più dell'automa  $LR(1)$  per un certo input.

$\Rightarrow$  i due parser consumano la stessa porzione di input prima di rilevare l'errore

$\Rightarrow LALR(1)$  è corretto come sostituto di  $LR(1)$

Riprendiamo l'esempio di pag. 36 + 37

(4)

	$x$	*	=	\$	S	E	V
1	$s_8$	$s_6$			$g_2$	$g_5$	$g_3$
2				acc			
3			$s_4$	$r_3$			
4	$s_{11}$	$s_{13}$				$g_9$	$g_7$
5				$r_2$			
6	$s_8$	$s_6$				$g_{10}$	$g_{12}$
7				$r_3$			
8			$r_4$	$r_4$			
9				$r_1$			
10			$r_5$	$r_5$			
11				$r_4$			
12			$r_3$	$r_3$			
13	$s_{11}$	$s_{13}$				$g_{14}$	$g_7$
14				$r_5$			

Tabella

LR(1)

Guardando  
l'automa LR(1)  
a pag. 36, si  
vede che  
6 - 13  
7 - 12  
8 - 11  
10 - 14  
non sono fusi!

Tabella LALR(1)

	$x$	*	=	\$	S	E	V
1	$s_8$	$s_6$			$g_2$	$g_5$	$g_3$
2				acc			
3			$s_4$	$r_3$			
4	$s_8$	$s_6$				$g_9$	$g_7$
5				$r_2$			
6 - 13	$s_8$	$s_6$				$g_{10}$	$g_7$
7 - 12			$r_3$	$r_3$			
8 - 11			$r_4$	$r_4$			
9				$r_1$			
10 - 14			$r_5$	$r_5$			

- (0)  $S' \rightarrow S$
- (1)  $S \rightarrow V = E$
- (2)  $S \rightarrow E$
- (3)  $E \rightarrow V$
- (4)  $V \rightarrow x$
- (5)  $V \rightarrow *E$

Non ci sono conflitti  $\Rightarrow G \in \text{LALR}(1)$ !

Tabella grande come quelle SLR(1) per  $G$ , ma  
questa non ha conflitti!!

## Passando da LR(1) a LALR(1)

(42)

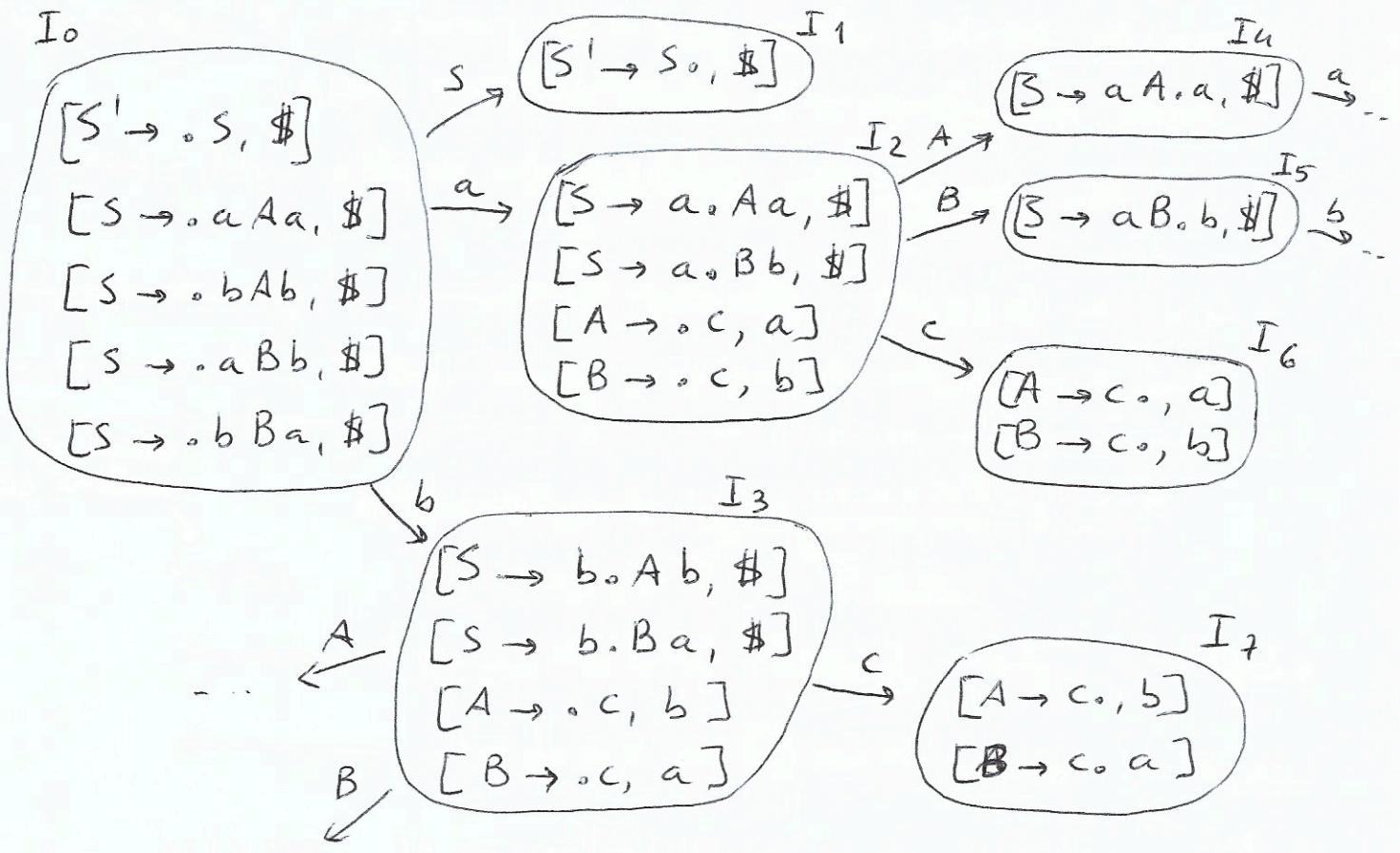
- La fusione di due stati LR(1) con lo stesso core può causare conflitti.  
Sono possibili solo nuovi conflitti reduce/reduce. Infatti, supponiamo che in  $s$ , stato ottenuto per fusione di 2 stati LR(1)  $s_1$  e  $s_2$ , presenti un confitto shift-reduce. Allora, esiste in  $s$  un item  $[A \rightarrow \alpha_0, a]$  e un item  $[B \rightarrow \beta_0, a\gamma, b]$ . Supponiamo, w. l. o. g., che  $[A \rightarrow \alpha_0, a] \in s_1$ . Allora  $[A \rightarrow \alpha_0, a]$  e  $[B \rightarrow \beta_0, a\gamma, b]$  (per qualche) appartengono ad  $s_1$ !  
 $\Rightarrow$  pure  $s_1$  in LR(1) avrebbe un conflitto shift-reduce, contro l'ipotesi che la tabella LR(1) non presenta conflitti!
- $\Rightarrow$  Se LR(1) è senza conflitti, LALR(1) potrebbe solo presentare conflitti reduce-reduce.

Se si generano conflitti, allora  $G$  non è LALR(1), pur essendo LR(1).

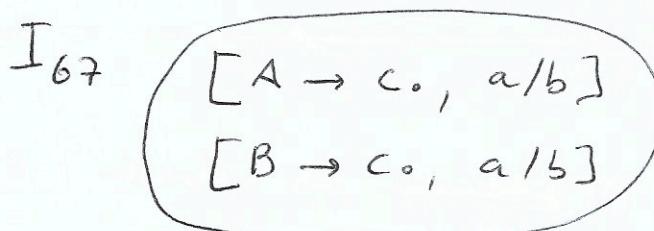
Esempio:  $G \in LR(1)$  ma non LALR(1)

$$S' \rightarrow S \quad S \rightarrow aAa \mid bAb \mid aBb \mid bBa \quad G$$

$$A \rightarrow c \quad B \rightarrow c$$



$I_6$  e  $I_7$  hanno lo stesso core - ma se li fondi



ora c'è presente un conflitto  
reduce-reduce

$$M'[67, a] = \{ \begin{array}{l} \text{reduce } A \rightarrow c, \\ \text{reduce } B \rightarrow c \end{array} \}$$

$$M'[67, b] = \{ \begin{array}{l} \text{reduce } A \rightarrow c, \\ \text{reduce } B \rightarrow c \end{array} \}$$

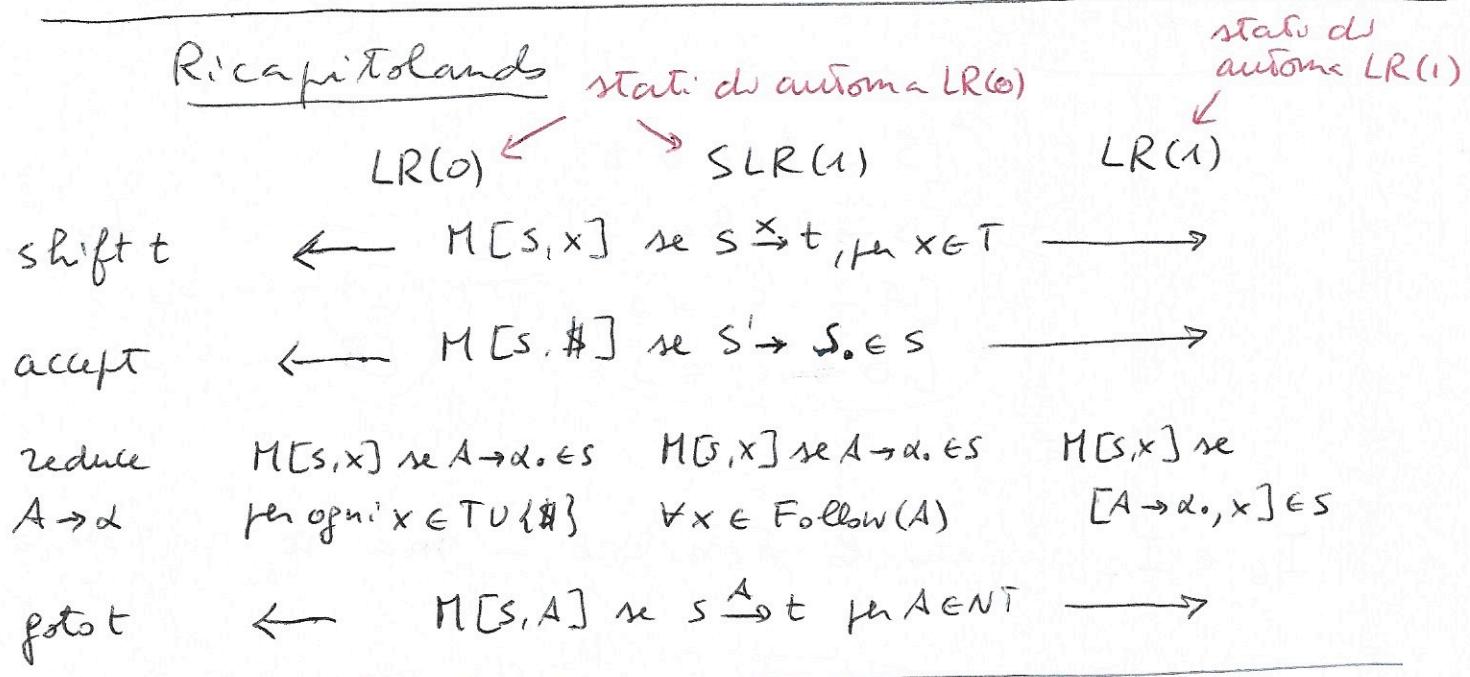
N.B. Nello stato  $I_6$  non c'è conflitto, e neanche nello stato  $I_7$ !  $\Rightarrow G$  è davvero LR(1), ma  $G$  non è LALR(1)

Abbiamo descritto come costruire un parser LALR(1), a partire da un parser LR(1). (44)

Tuttavia è possibile costruire il parser LALR(1) anche senza dover prima generare l'automa LR(1), ma direttamente dall'automa LR(0).

⇒ maggiore efficienza nella costruzione

Questa tecnica, che non vedremo, è quella usata dai generatori di parser, quali YACC (che vedremo).



per LALR(1), si prende la tabella LR(1) e  
si fondono gli stati con lo stesso "core LR(0)".