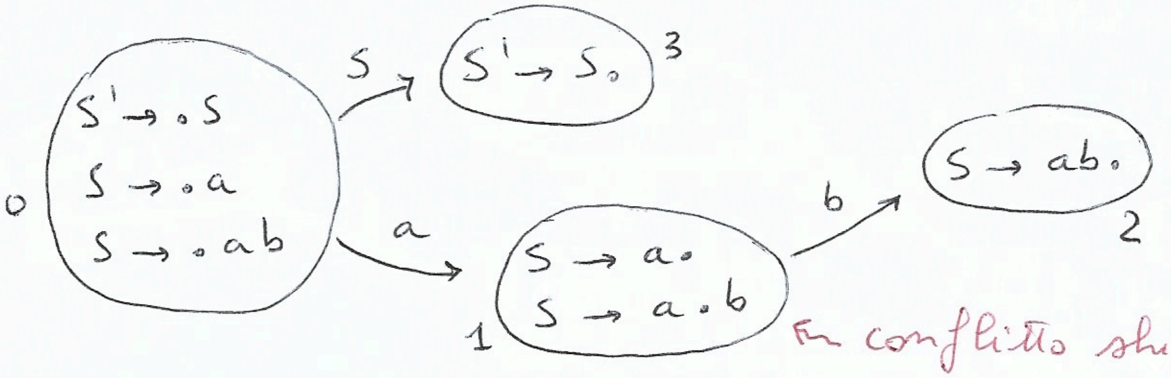
Ma non sempre siamo così fortunati...

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

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



⇒ conflitto shift-reduce!!

	a	b	\$	S
0	s1			g3
1	r2	r2/s2	r2	
2	r3	r3	r3	
3			acc	

Tabella di parsing LR(0)

Come risolvere il conflitto? Usiamo il look-ahead!
Cioè guardiamo il Follow(S): se b ∈ Follow(S), allora il conflitto è reale! Altrimenti, no!
(può essere)

Ma Follow(S) = {\$} ⇒ risolviamo il conflitto a favore dello shift

	a	b	\$	S
0	s1			g3
1		s2	r2	
2			r3	
3			acc	

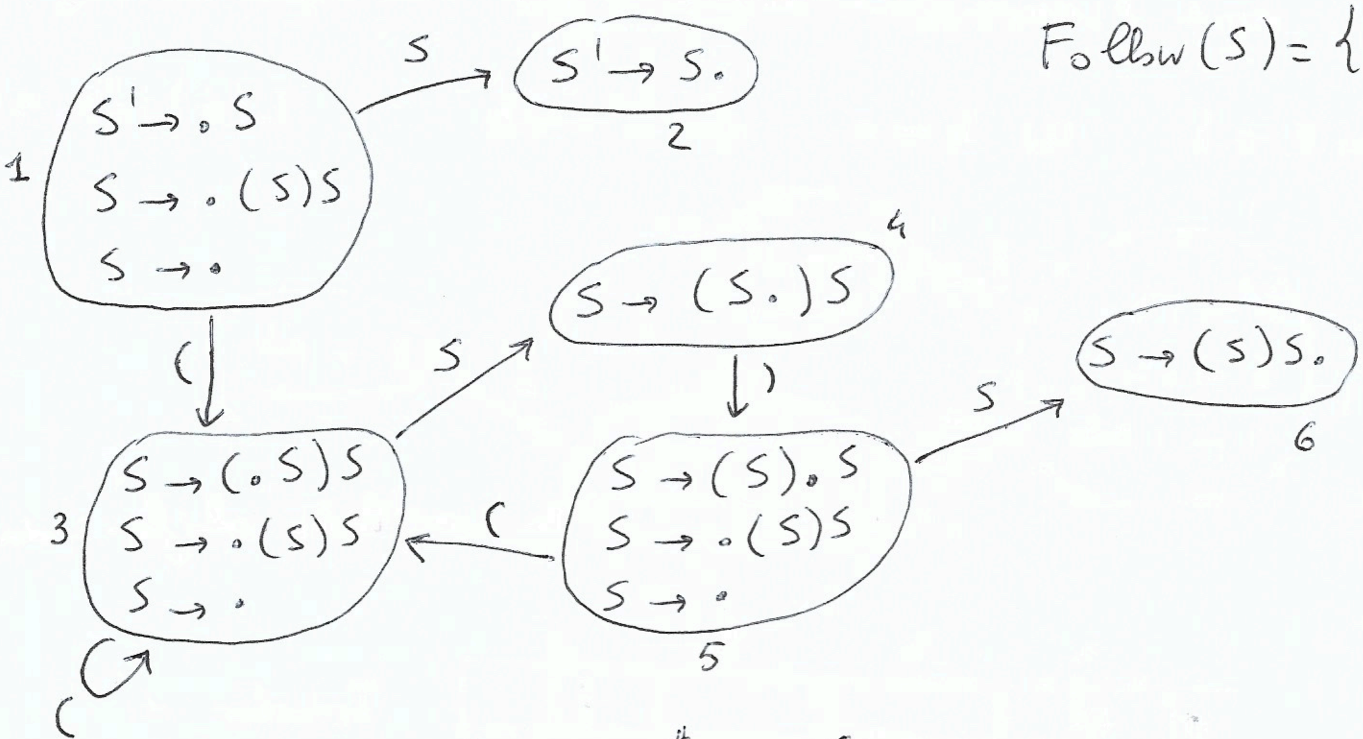
Tabella di parsing SLR(1)
S ↑ simple

– solo per i caratteri nel Follow(S) mettiamo r2

Esempio: Parentesi Bilanciate

(0) $S' \to S$ (1) $S \to (S)S$ (2) $S \to \varepsilon$

Follow(S) = {), $ }



	(	)	$	S
1	s3/r2	r2	r2	g2
2			acc	
3	s3/r2	r2	r2	g4
4		s5		
5	s3/r2	r2	r2	g6
6	r1	r1	r1	

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

	(	)	$	S
1	s3	r2	r2	g2
2			acc	
3	s3	r2	r2	g4
4		s5		
5	s3	r2	r2	g6
6		r1	r1	

Tabella di parsing SLR(1) (senza conflitti)

$\Downarrow$

G è SLR(1)

(ma non LR(0))

OSS:

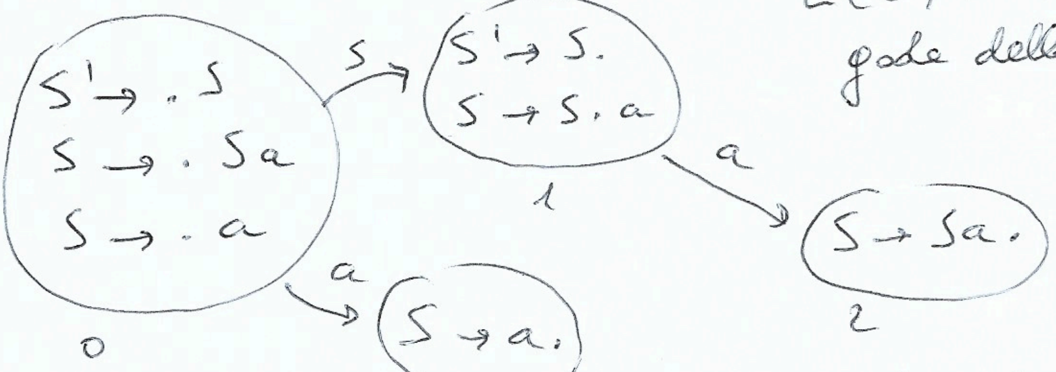
(1) Se G ha produzioni ε, allora G difficilmente è LR(0)
(caso banale in cui uno stato con item $A \to .$ non ha item del tipo $B \to \alpha . a \beta$)

(2) Se L è libero deterministico e gode della prefix-property ("L gode della prefix property se $\nexists x, y \in L$ 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. Ovvero se L è finito e non gode della prefix-property, allora L non è LR(0)
$L = \{a, ab\}$

(4) Se L è LR(0) ma è infinito, può non godere della prefix-property

$S \to Sa \mid a$] G è LR(0) ma $L(G) = a^+$ (che non gode della prefix-property)



Non ci sono conflitti

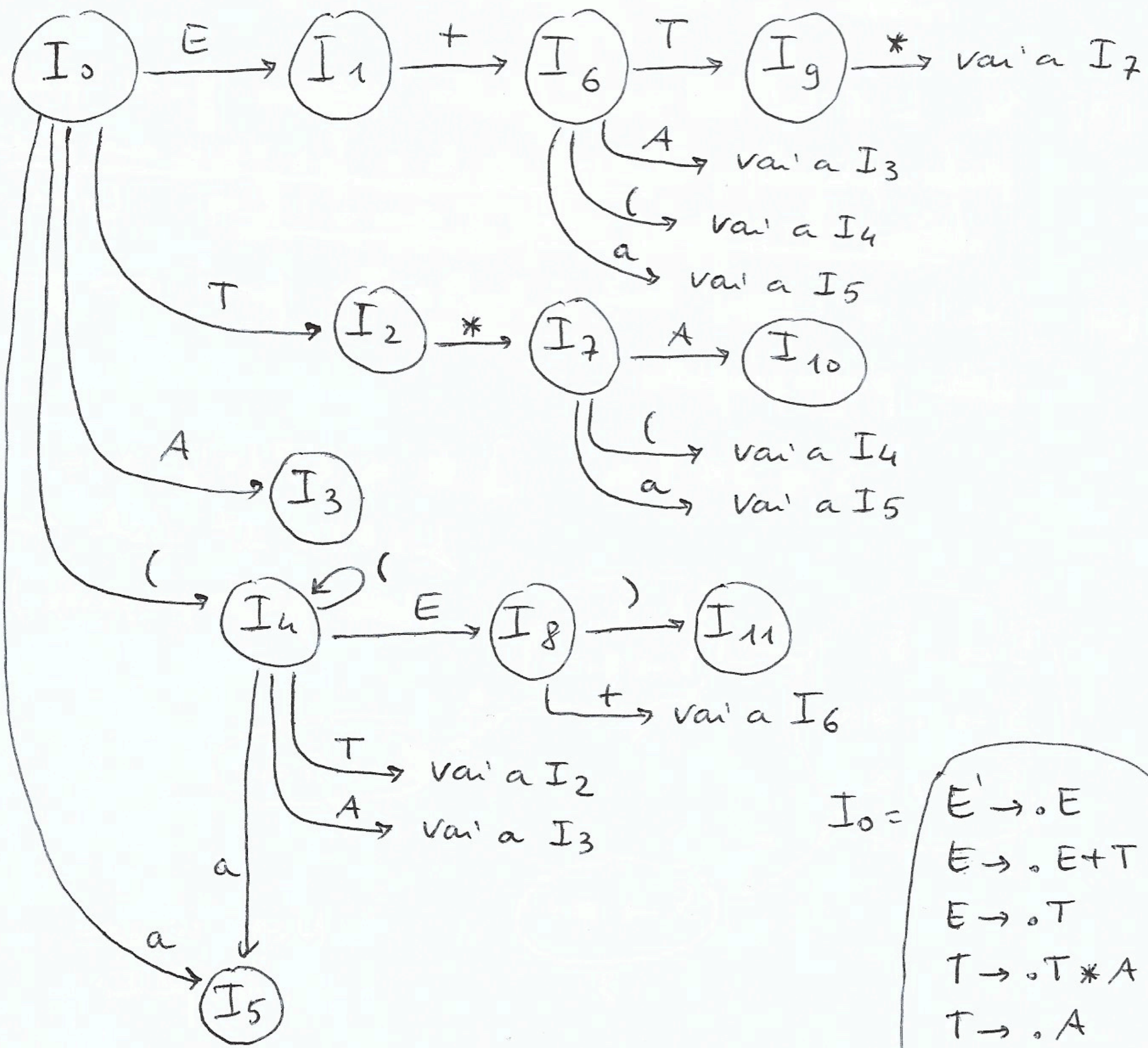
Esercizio $S \to Sa \mid \varepsilon$ è LR(0)!

Esercizio "corposo"

(0) $E' \to E$ (1) $E \to E+T$ (2) $E \to T$

(3) $T \to T*A$ (4) $T \to A$ (5) $A \to a$ (6) $A \to (E)$

- Costruire l'automa canonico LR(0) (a partire da $Clos(\{E' \to .E\}) = I_0$ (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 = $
$E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*A$
$T \to .A$
$A \to .a$
$A \to .(E)$

Tabella di Parsing SLR(1)

simple

- colonne: $T \cup \{\$\} \cup NT$
- 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 $A \to \alpha . \in s$ e $A \neq S'$, inserisci reduce $A \to \alpha$ in $M[s, x]$ per tutti gli $x \in Follow(A)$
3. se $S' \to S . \in s$, 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)

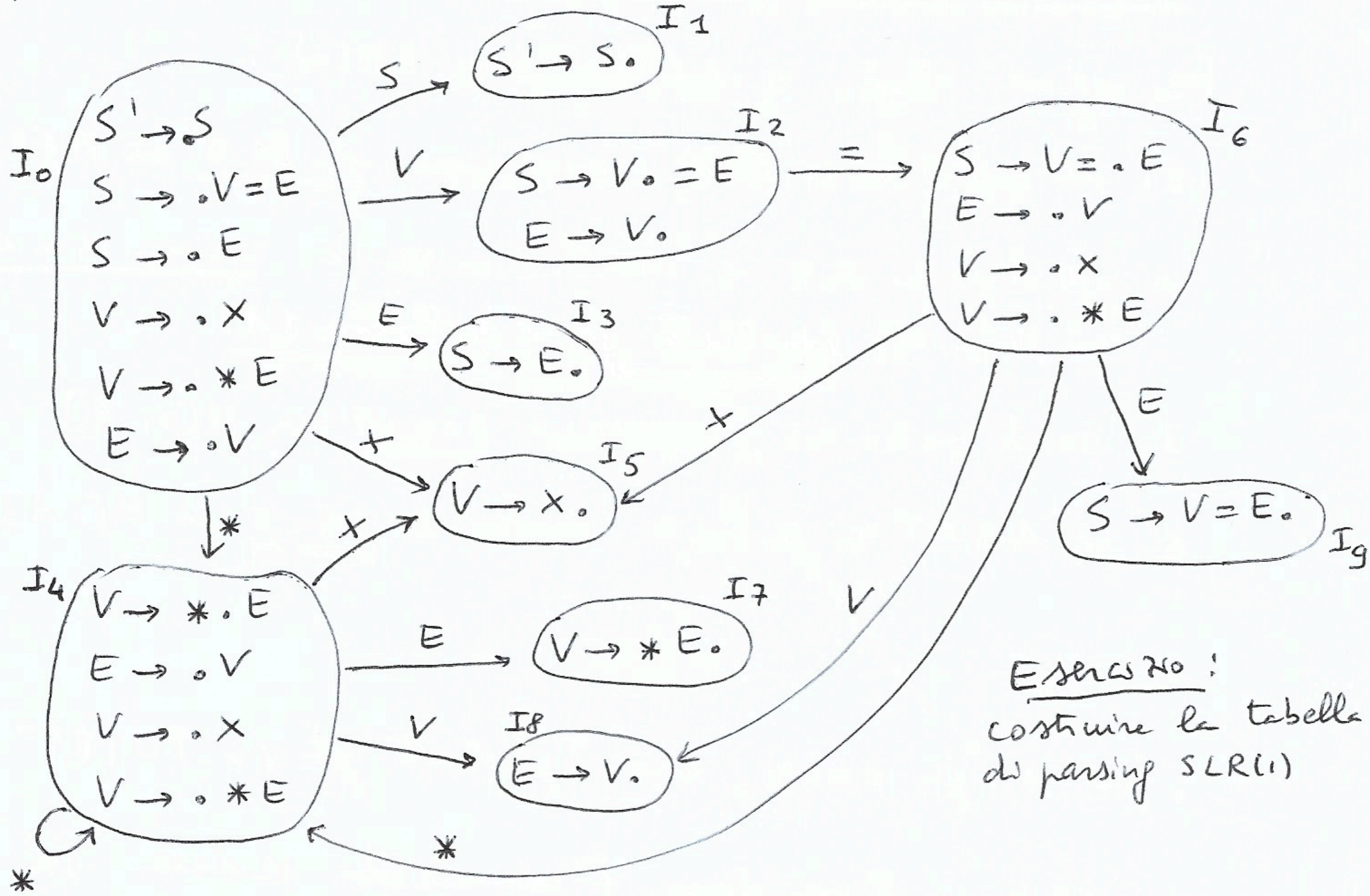
Vedremo che LR(1) usa esplicitamente il look-ahead già nella definizione di item LR(1)

Ma non sempre siamo così fortunati...

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

(0) $S' \to S$ (1) $S \to V = E$ (2) $S \to E$ (3) $E \to V$

(4) $V \to x$ (5) $V \to * E$



Esercizio! costruire la tabella di parsing SLR(1)

Consideriamo I_2 $\begin{array}{l} S \to V. = E \\ E \to V. \end{array}$

conflitto shift/reduce perché $= \in \mathrm{Follow}(E)$ (a causa di $V \to *E$, si ha $\mathrm{Follow}(V) \subseteq \mathrm{Follow}(E)$)

$M[I_2, =] = \{s6, r3\}$ nella tabella di parsing SLR(1)

Oss: Se noi sappiamo con certezza che il carattere successivo è davvero "=", allora la reduce $E \to V.$ non va considerata, perché non è mai possibile derivare

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

$\Rightarrow$ ha senso solo fare lo shift!

Oss: $= \in$ Follow (E) vuol dire che esiste una derivazione tale che $S \Rightarrow^* \alpha E = \beta$, 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 $= \in$ Follow (E), ma è impossibile derivare

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

per cui nello stato I_2, se in 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 Follow (E)!

$\Rightarrow$ item LR(1): una coppia formata da

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

Esempio

I_2 $\boxed{\begin{matrix}[S \to V_\circ = E, \$] \\ [E \to V_\circ \quad , \$]\end{matrix}}$

- stato dell'automa canonico LR(1)
- se leggo $=$, allora shift
- se leggo $\$$, allora reduce $E \to V$

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

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

cioè x può davvero essere fatta in questa derivazione

(quando uso il Follow(A) nella 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!)

- $[A \to \alpha \circ, x]$

$\Rightarrow$ fai la reduce $A \to \alpha$ se il prossimo input è proprio x.

Item LR(1)

usiamo esplicitamente 1 carattere in avanti dell'input, associato direttamente all'item LR(0) - core

NFA LR(1)

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

(N.B. $First(\beta a) \subseteq T \cup \{\$\}$)

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 $Clos(I)$ e $Goto(I, X)$

  partendo dallo stato iniziale

  $clos([S' \to .S, \$])$

```
Clos (I) {
   ripeti finché I è modificato {
      per ogni item [A → α . Xβ, a] ∈ I
         per ogni produzione X → γ
            per ogni b ∈ First(βa)
               aggiungi [X → .γ, b] a I;
            }
   return I;
   }
```

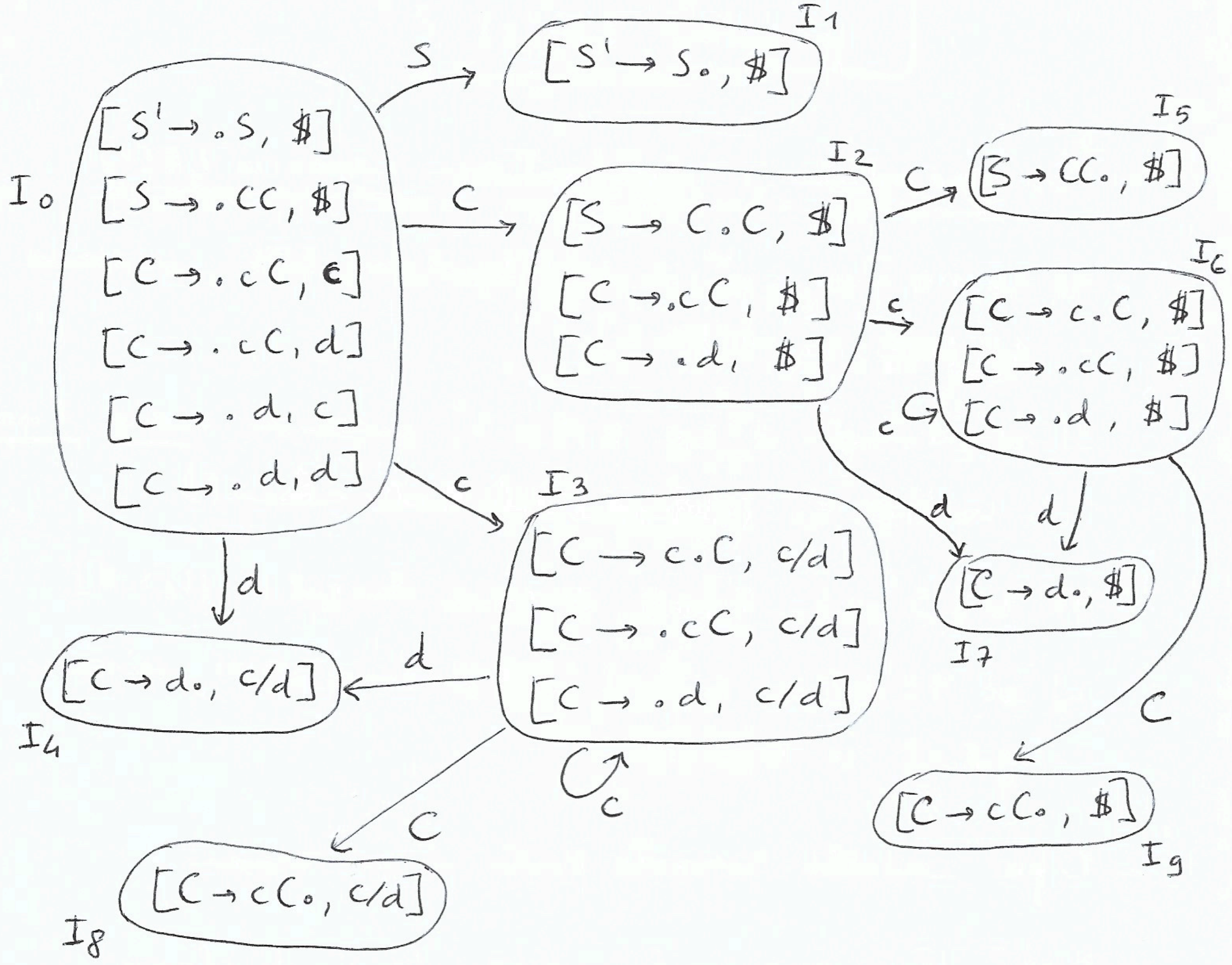
```
Goto (I, X) {
   inizializza J = ∅;
   per ogni item [A → α . Xβ, a] ∈ I
      aggiungi [A → αX . β, a] a J;
   return Clos(J);
```

Stato iniziale dell'automa canonico LR(1)

è $Clos([S' \to .S, \$])$

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

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



Esercizio: costruite l'automa canonico LR(0) per questa grammatica. Osserverete che ci sono meno stati! E anche che non ci sono conflitti ...

(per 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 $[A \to \alpha \cdot, x] \in s$ e $A \neq S'$, inserisci reduce $A \to \alpha$ in $M[s, x]$ (solo per x del look-ahead!)

3) se $[S' \to S \cdot, \$] \in s$, 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) $S' \to S$

(1) $S \to CC$

(2) $C \to cC$

(3) $C \to d$

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

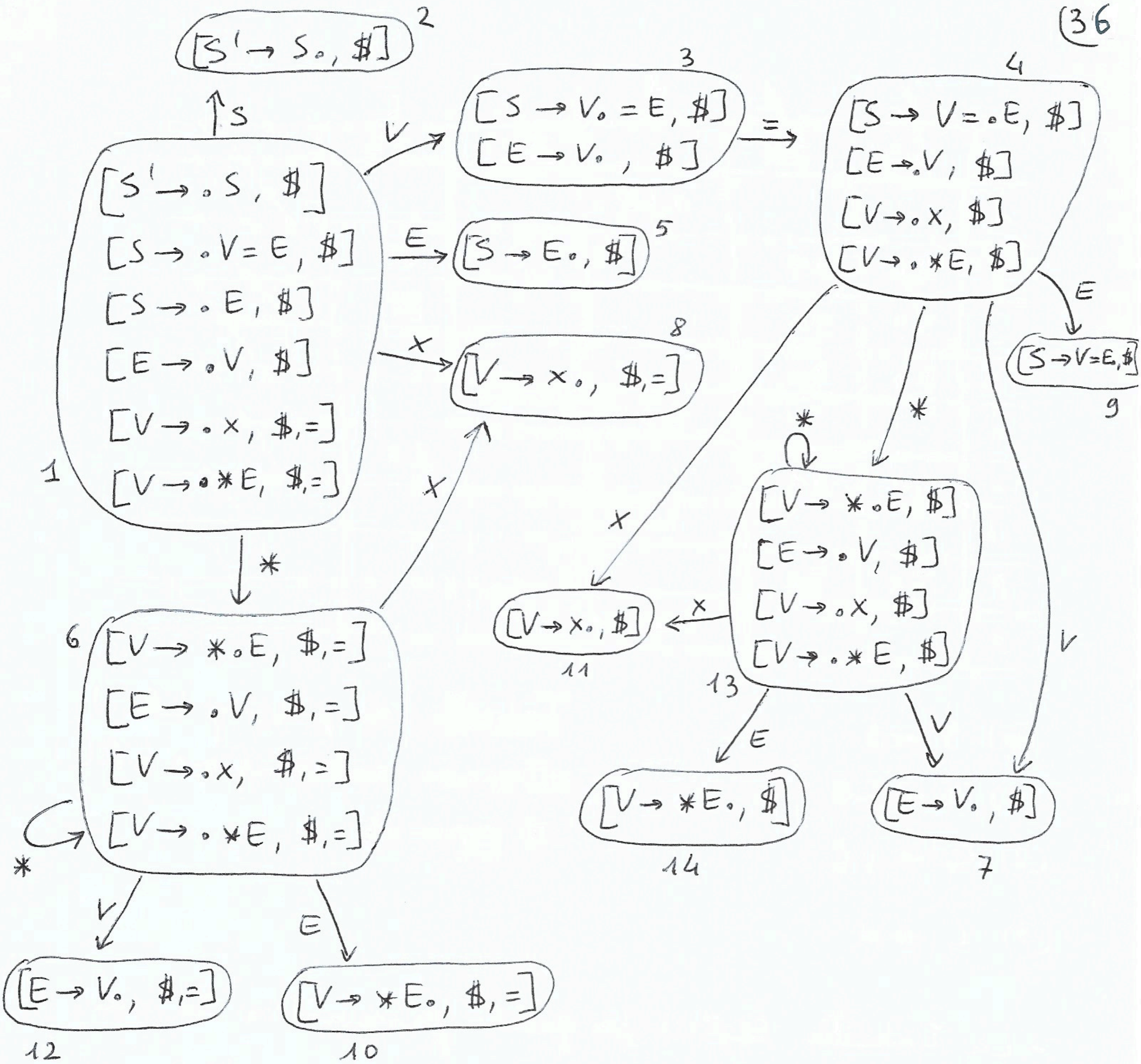
- La tabella SLR(1) per questa grammatica ha solo 7 stati, anziché 10, ed è pure senza conflitti
- Vedremo che questa grammatica è (ovviamente) anche LALR(1)

Riconsideriamo la grammatica NON SLR(1)

(0) $S' \to S$ (1) $S \to V = E$ (2) $S \to E$

(3) $E \to V$ (4) $V \to x$ (5) $V \to * E$

e vediamo se è LR(1)!



6 – 13
7 – 12
8 – 11
10 – 14

coppie di stati che in LR(0) sarebbero stati fusi insieme

Tabelle di parsing LR(1)

	x	*	=	\$	S	E	V
1	s8	s6			g2	g5	g3
2				acc			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(0) $S' \to S$ (1) $S \to V = E$ (2) $S \to E$

(3) $E \to V$ (4) $V \to x$ (5) $V \to * E$

Non ci sono conflitti

$\Rightarrow$ G è LR(1)!

Parser LALR(1)

look-ahead

- LR(1): tabelle di parsing molto grandi
  (centinaia di migliaia di stati per ling. di media grandezza)

- LALR(1): buon compromesso tra semplicità (e compattezza) 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 Goto(I, X) usa di I solo la parte "nucleo"/LR(0) dell'item LR(1)

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

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

Riprendiamo l'esempio di pagina 33+35

Tabella LR(1)

	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) $S' \to S$

G:
(1) $S \to CC$
(2) $C \to cC$
(3) $C \to d$

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}		g1	g2
1			acc		
2	s_{36}	s_{47}			g5
36	s_{36}	s_{47}			g89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

G è LALR(1) perché non ci sono conflitti

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

Se G è LR(1) e anche LALR(1),

allora ... ?

L'automa LR(1) e quello LALR(1) si mimano perfettamente su input corretti.
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)
(0, ε, ccd$)
(03, c, cd$)
(033, cc, d$)
(0334, ccd, $) e trova errore perché
M[4, $] = "bianca"

LALR(1)
(0, ε, ccd$)
(0 36, c, cd$)
(0 36 36, cc, d$)
(0 36 36 47, ccd, $) che non è bloccato, ma

$\underbrace{47}$ → goto 89, $\underbrace{d}_{C}$

(0 36 36 89, ccC, $)

$\underbrace{36\ 89}$ → goto 89, $\underbrace{cC}_{C}$

(0 36 89, cC, $)

$\underbrace{36\ 89}$ → goto 2, $\underbrace{cC}_{C}$

(0 2, C, $) e finalmente trova errore
perché M'[2, $] = "bianca"

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

⇒ i due parser consumano la stessa porzione di input prima di rilevare l'errore

⇒ LALR(1) è corretto come sostituto di LR(1)

Riprendiamo l'esempio di pag. 36+37

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				acc			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

Tabella LR(1)

Guardando l'automa LR(1) a pag. 36, si vede che

6 – 13
7 – 12
8 – 11
10 – 14

possono essere fusi!

Tabella LALR(1)

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				acc			
3			s4	r3			
4	s8	s6				g9	g7
5				r2			
6 – 13	s8	s6				g10	g7
7 – 12			r3	r3			
8 – 11			r4	r4			
9				r1			
10 – 14			r5	r5			

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

Non ci sono conflitti $\Rightarrow$ G è LALR(1)!

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

Passando da LR(1) a LALR(1)

- 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 conflitto shift-reduce. Allora, esiste in S un item $[A \to \alpha\cdot, a]$ e un item $[B \to \beta\cdot a\gamma, b]$. Supponiamo, w.l.o.g., che $[A \to \alpha\cdot, a] \in S_1$. Allora $[A \to \alpha\cdot, a]$ e $[B \to \beta\cdot a\gamma, c]$ (per qualche c) appartengono ad S_1!

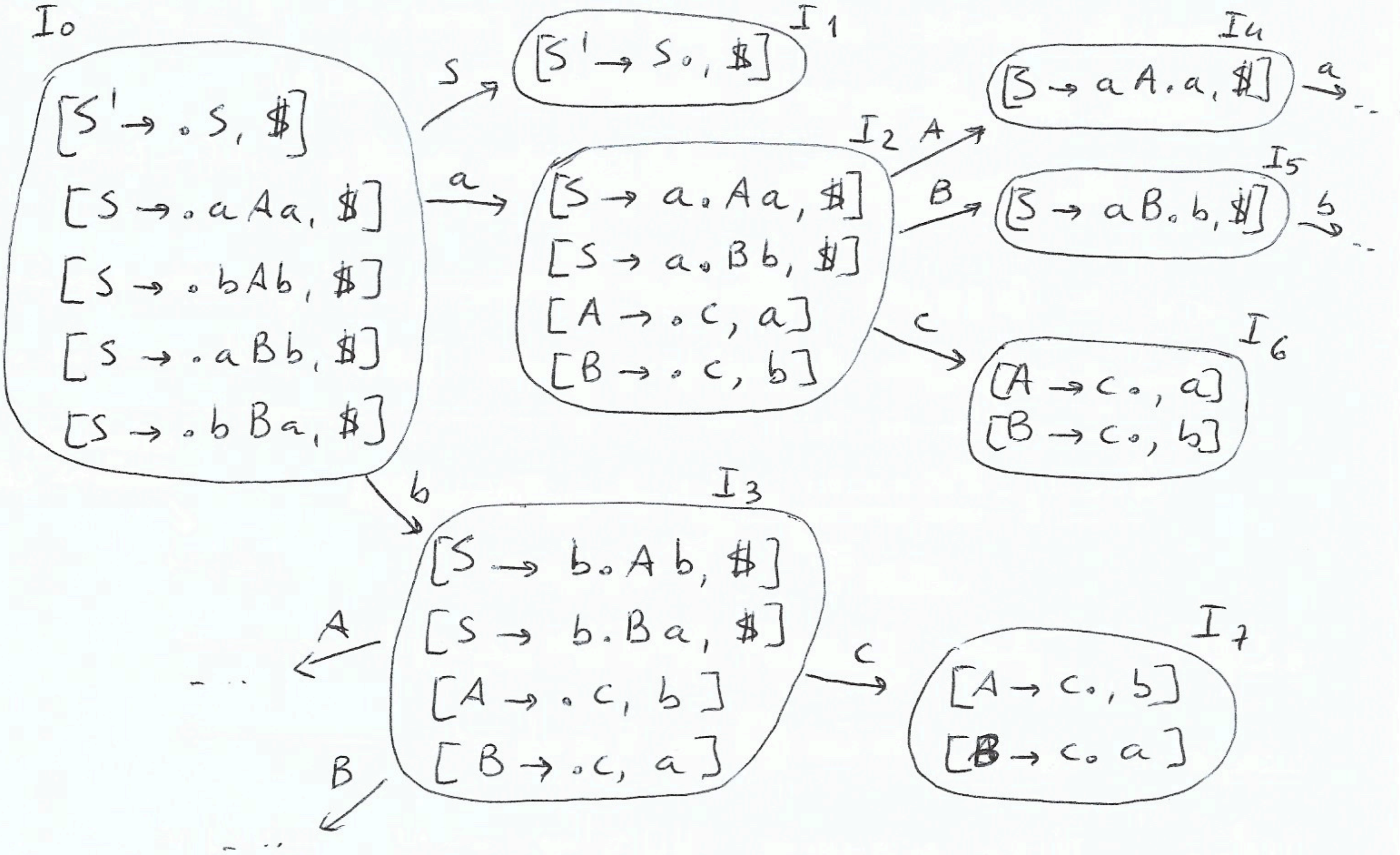
  $\Rightarrow$ pure S_1 in LR(1) avrebbe un conflitto shift-reduce, contro l'ipotesi che la tabella LR(1) non presenti conflitti!

  $\Rightarrow$ 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 è LR(1) ma non LALR(1)

$$\left.\begin{array}{ll} S' \to S & S \to aAa \mid bAb \mid aBb \mid bBa \\ & A \to c \qquad B \to c \end{array}\right] G$$



I_6 e I_7 hanno lo stesso core – ma se li fondo,

I_{67}

$$\begin{array}{l} [A \to c\circ,\ a/b] \\ [B \to c\circ,\ a/b] \end{array}$$

ora è presente un conflitto reduce – reduce

$$M'[67, a] = \{ \underset{A \to c}{\text{reduce}},\ \underset{B \to c}{\text{reduce}} \}$$

$$M'[67, b] = \{ \underset{A \to c}{\text{reduce}},\ \underset{B \to c}{\text{reduce}} \}$$

<u>N.B.</u> Nello stato I_6 non c'è conflitto, e neanche nello stato I_7! ⟹ 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).

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

$\Rightarrow$ maggiore efficienza nella costruzione

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

Ricapitolando

stati di automa LR(0) | stato di automa LR(1)

	LR(0)	SLR(1)	LR(1)
shift t	$\longleftarrow$	$M[s,x]$ se $s \xrightarrow{x} t$, per $x \in T$	$\longrightarrow$
accept	$\longleftarrow$	$M[s,\#]$ se $S' \to S. \in s$	$\longrightarrow$
reduce $A \to \alpha$	$M[s,x]$ se $A \to \alpha. \in s$ per ogni $x \in T \cup \{\#\}$	$M[s,x]$ se $A \to \alpha. \in s$ $\forall x \in Follow(A)$	$M[s,x]$ se $[A \to \alpha., x] \in s$
goto t	$\longleftarrow$	$M[s,A]$ se $s \xrightarrow{A} t$ per $A \in NT$	$\longrightarrow$

---

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