NEW YORK UNIVERSITY

CSCI-GA.2130-001

Compiler Construction

Lecture 7: Syntax Analysis II

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu

LR Parsers

+ Can be constructed to recognize virtually all programming languages constructed from context-free grammars
+ Most general non-backtracking shift-reduce parsing method
+ Very fast at detecting syntactic errors
- Too much work to construct LR parsing by hand

Definitions: States & Items

- Item: production rule with information about current parsing state

$A \to XYZ$ yields the four items

$$A \to \cdot XYZ$$
$$A \to X \cdot YZ$$
$$A \to XY \cdot Z$$
$$A \to XYZ \cdot$$

- State = set of items

Definitions: Closure of Item

We have a set of items ***I***:

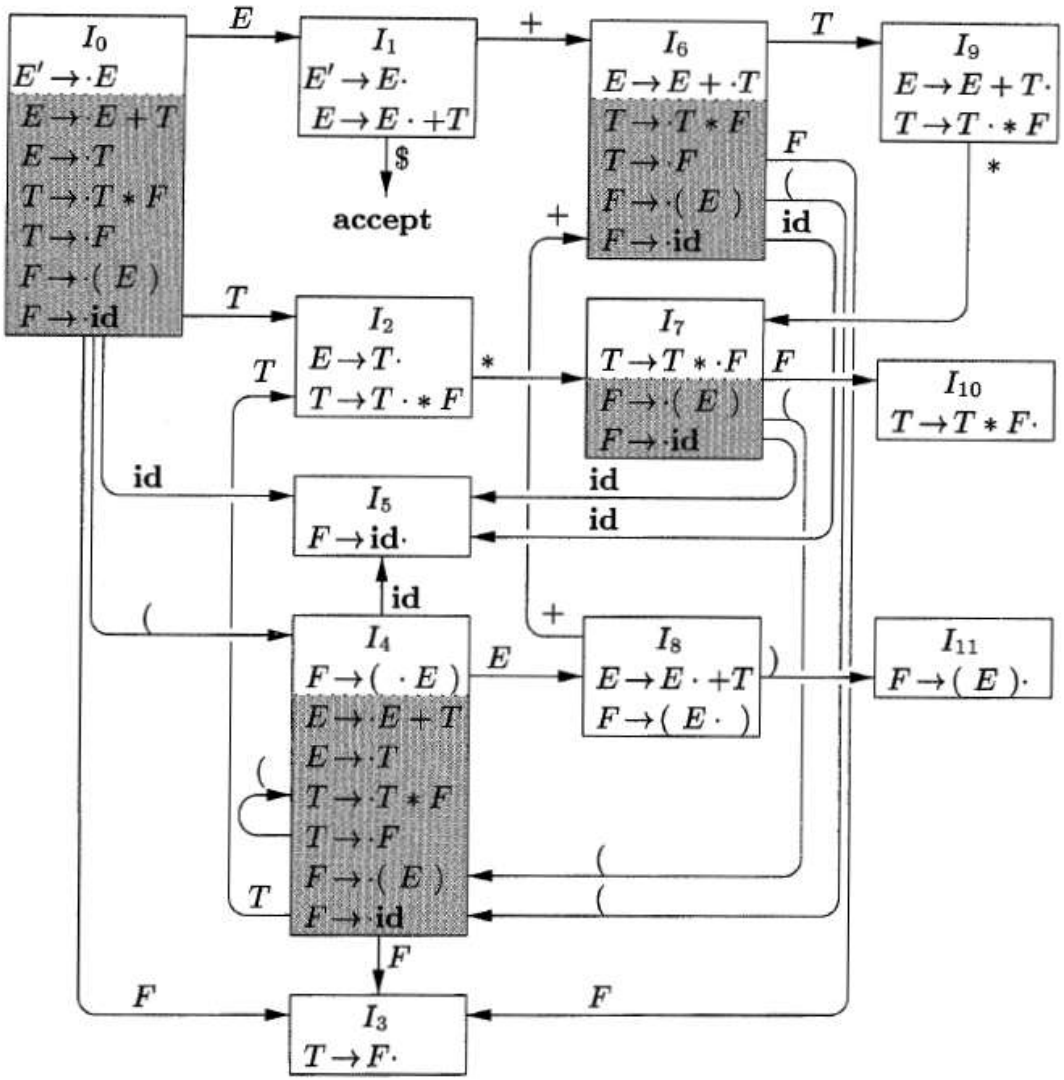
1. Initially, add every item in I to CLOSURE(I).
2. If $A \to \alpha \cdot B\beta$ is in CLOSURE(I) and $B \to \gamma$ is a production, then add the item $B \to \cdot\gamma$ to CLOSURE(I), if it is not already there. Apply this rule until no more new items can be added to CLOSURE(I).

DFA: LR(0) Automaton

- Built from Collection of sets of LR(0) items
- Presents the different states during parsing
- Given grammar G, we have to augment it by an extra production E'->E where E is start symbol

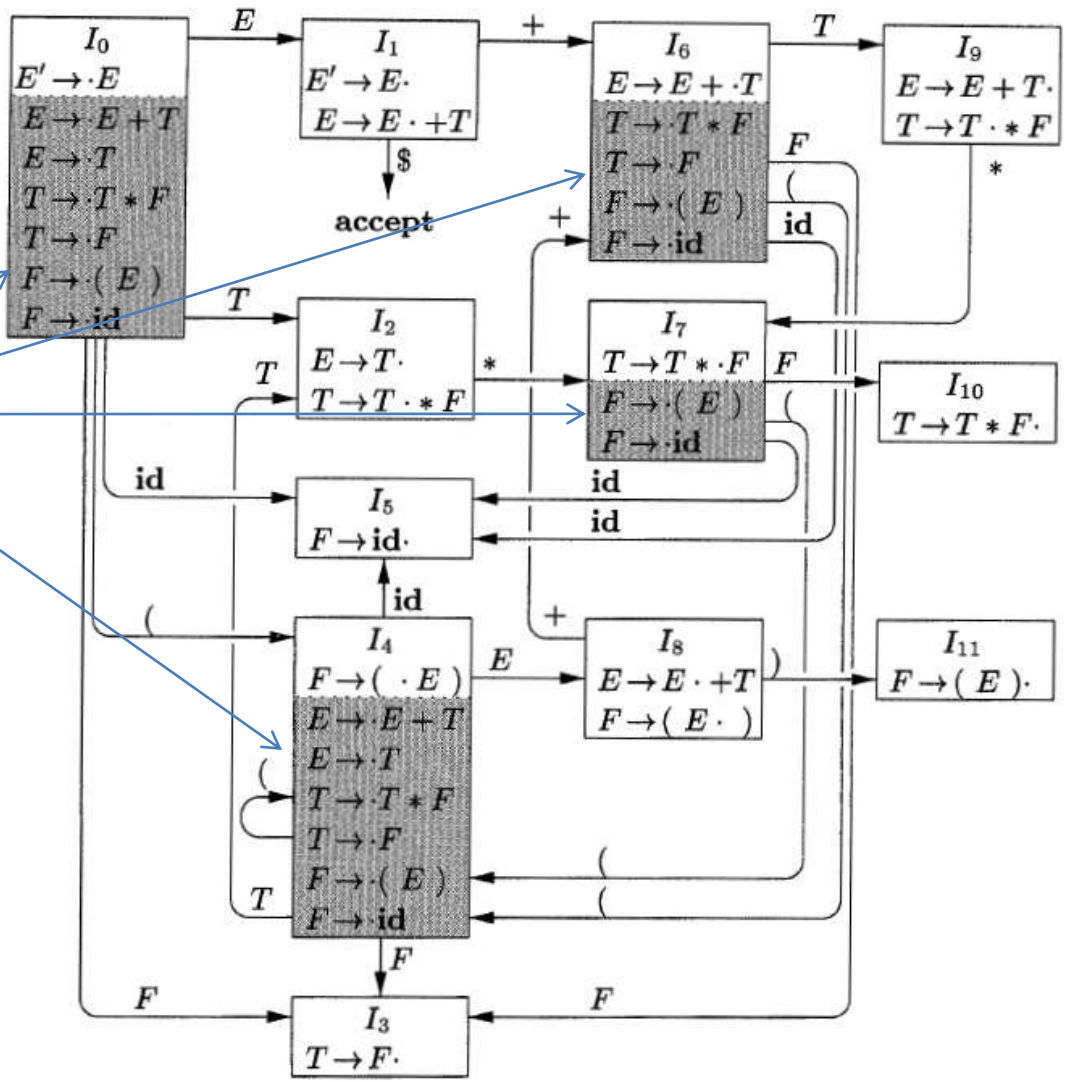
$$
\begin{array}{lcl}
E & \rightarrow & E+T \mid T \\
T & \rightarrow & T*F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E+T \mid T \\
T & \rightarrow & T*F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$



$$
\begin{array}{lcl}
E & \rightarrow & E+T \mid T \\
T & \rightarrow & T*F \mid F \\
\mathrm{F} & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

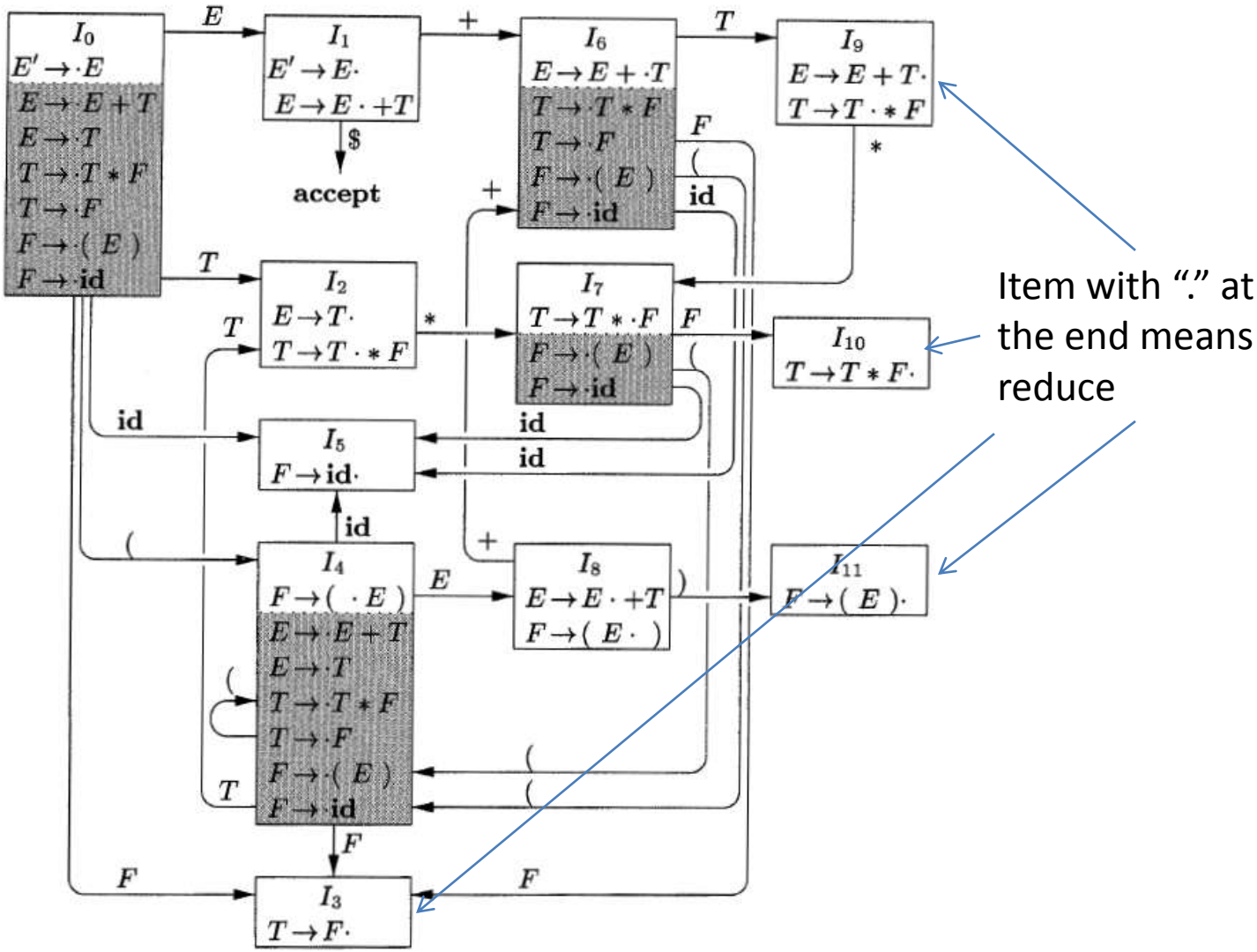
$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E+T \mid T \\
T & \rightarrow & T*F \mid F \\
\mathrm{F} & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$



- nonkernel items
- no need to be stored
- dot at far left

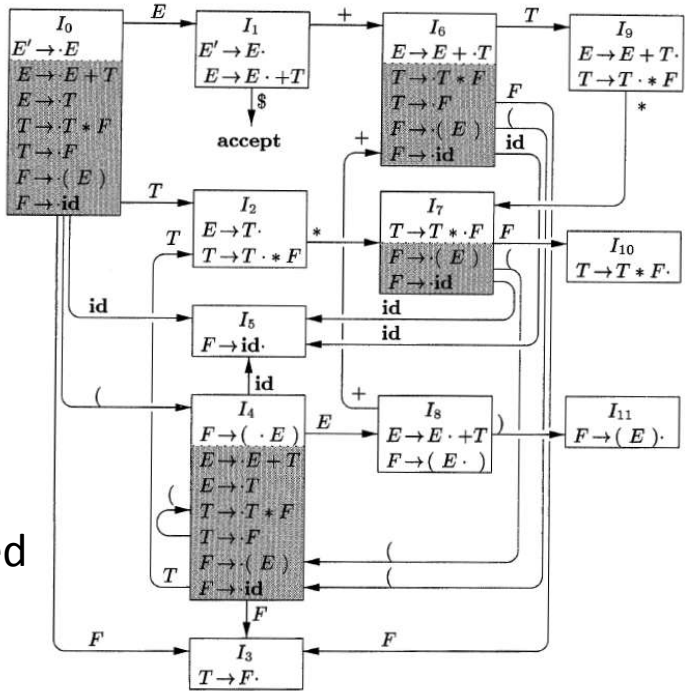
$$
\begin{array}{lcl}
E & \rightarrow & E+T \mid T \\
T & \rightarrow & T*F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E+T \mid T \\
T & \rightarrow & T*F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$



Shift-Reduce Revisited

$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
E & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

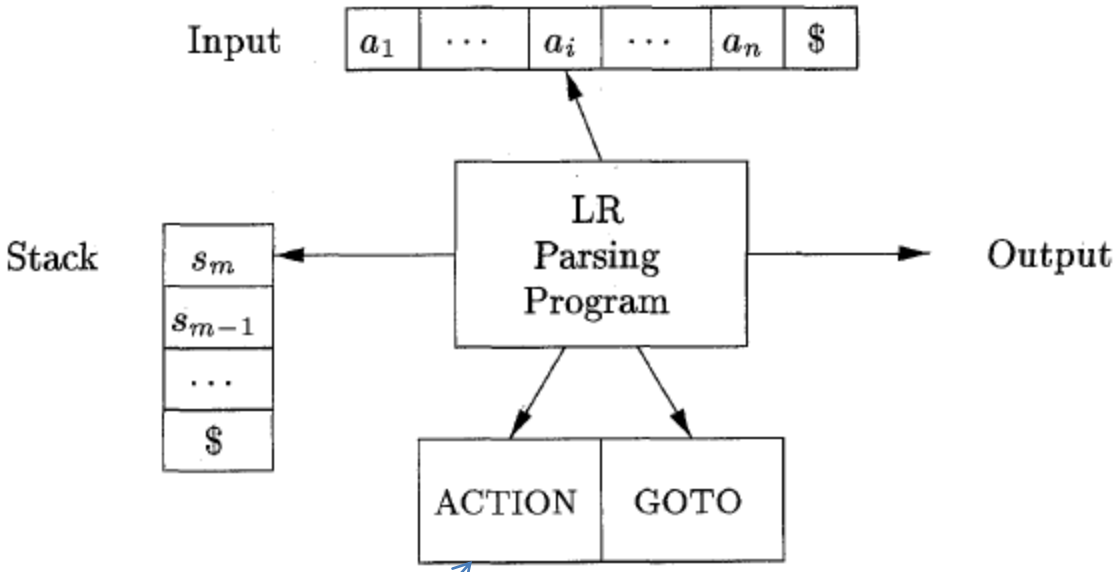


- When there is no transition we reduce.
- When we reduce:
  - pop state from top of stack
  - The new top of stack will tell you the next state based on the symbol.

Parse for ***id * id***

LINE	STACK	SYMBOLS	INPUT	ACTION
(1)	0	\$	**id** $*$ **id** \$	shift to 5
(2)	0 5	\$ **id**	$*$ **id** \$	reduce by $F \rightarrow \mathbf{id}$
(3)	0 3	\$ F	$*$ **id** \$	reduce by $T \rightarrow F$
(4)	0 2	\$ T	$*$ **id** \$	shift to 7
(5)	0 2 7	\$ T $*$	**id** \$	shift to 5
(6)	0 2 7 5	\$ T $*$ **id**	\$	reduce by $F \rightarrow \mathbf{id}$
(7)	0 2 7 10	\$ T $*$ F	\$	reduce by $T \rightarrow T * F$
(8)	0 2	\$ T	\$	reduce by $E \rightarrow T$
(9)	0 1	\$ E	\$	accept

The *Real* Implementation



- Shift
- Reduce
- Accept
- Error

GOTO(I,X)

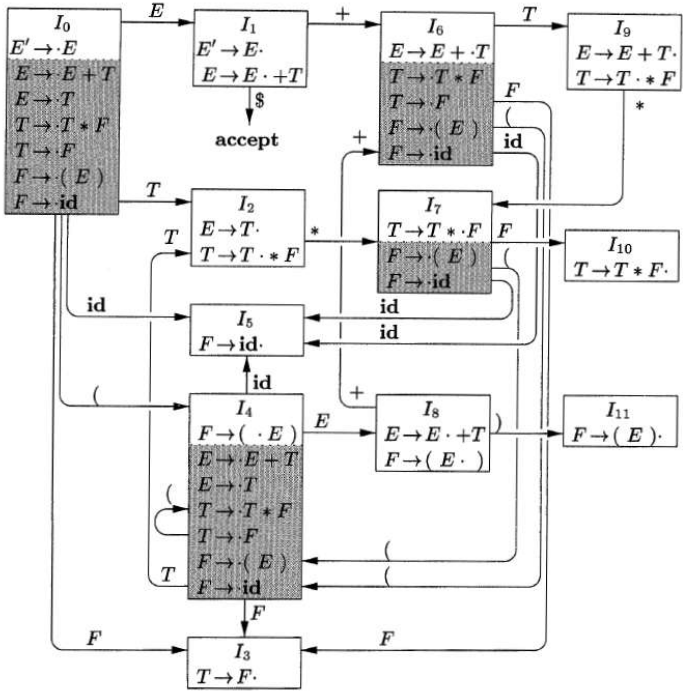
- I is a set of items
- X is a grammar symbol
- GOTO(I,X) is defined to be:
  The closure of the set of all items

$[A \to \alpha X \cdot \beta]$ such that $[A \to \alpha \cdot X\beta]$ is in I

Example: If I is the set of two items $\{[E' \to E\cdot],\ [E \to E\cdot + T]\}$
GOTO$(I, +)$ contains the items

$$
\begin{array}{lcl}
E' & \to & E \\
E & \to & E + T \mid T \\
T & \to & T * F \mid F \\
E & \to & (E) \mid \mathbf{id}
\end{array}
$$

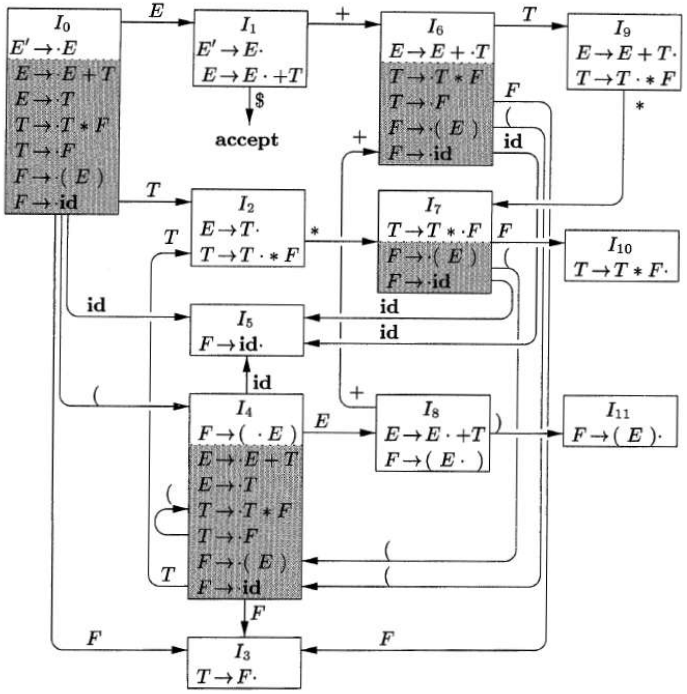
$$
\begin{array}{l}
E \to E + \cdot T \\
T \to \cdot T * F \\
T \to \cdot F \\
F \to \cdot (E) \\
F \to \cdot \mathbf{id}
\end{array}
$$



(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$

SLR Table

STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$

SLR Table

STATE	ACTION						GOTO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

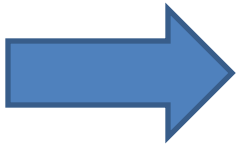
Means shift and move to state 5

Means reduce using production number 1

A Tricky Problem

What is the LR(0) DFA of the following grammar?

S-> L=R
S -> R
L-> *R
L-> ID
R-> L



I_0: $S' \to \cdot S$
$S \to \cdot L = R$
$S \to \cdot R$
$L \to \cdot * R$
$L \to \cdot \mathbf{id}$
$R \to \cdot L$

I_1: $S' \to S\cdot$

I_2: $S \to L\cdot = R$
$R \to L\cdot$

I_3: $S \to R\cdot$

I_4: $L \to *\cdot R$
$R \to \cdot L$
$L \to \cdot * R$
$L \to \cdot \mathbf{id}$

I_5: $L \to \mathbf{id}\cdot$

I_6: $S \to L = \cdot R$
$R \to \cdot L$
$L \to \cdot * R$
$L \to \cdot \mathbf{id}$

I_7: $L \to *R\cdot$

I_8: $R \to L\cdot$

I_9: $S \to L = R\cdot$

Suppose we are at state 2 and encounter '='

Shift to 6
Reduce by R-> L

Shift/Reduce conflict !!

Do you see any problems?

Is SLR-Parsing Powerful Enough?

- Every SLR grammar is unambiguous
- There are many unambiguous grammars that are not SLR
- What if in a state we can do both reduction and shift? Which one to choose?
- [S->C,d] means: use production S->C if next terminal is d
- Lookahead LR (LALR) method

LALR

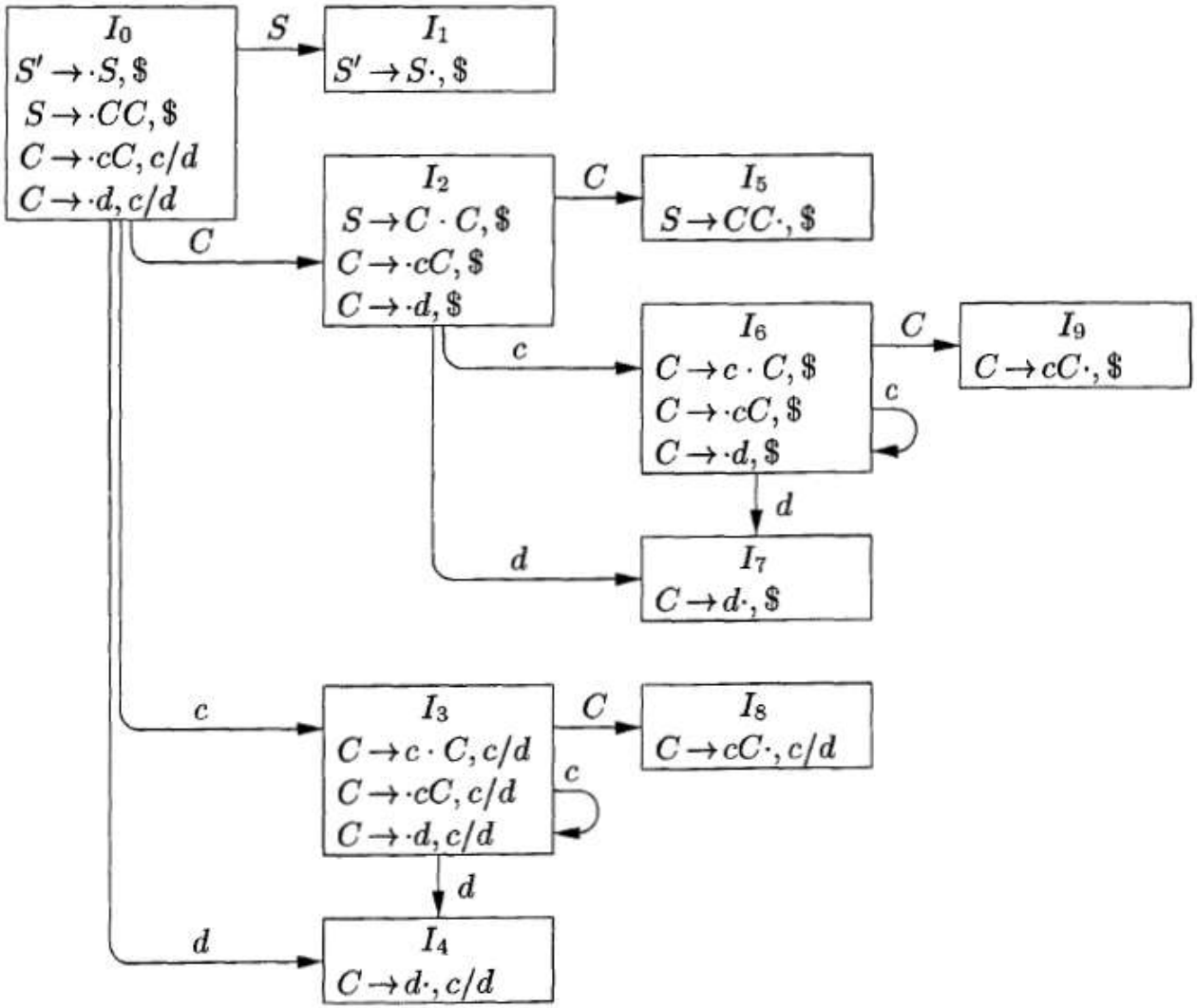
- There are two very powerful parsers:
  - canonical-LR
  - LALR
- LALR is the one used in practice
  - considerably smaller tables than canonical-LR
  - Can express most programming languages constructs

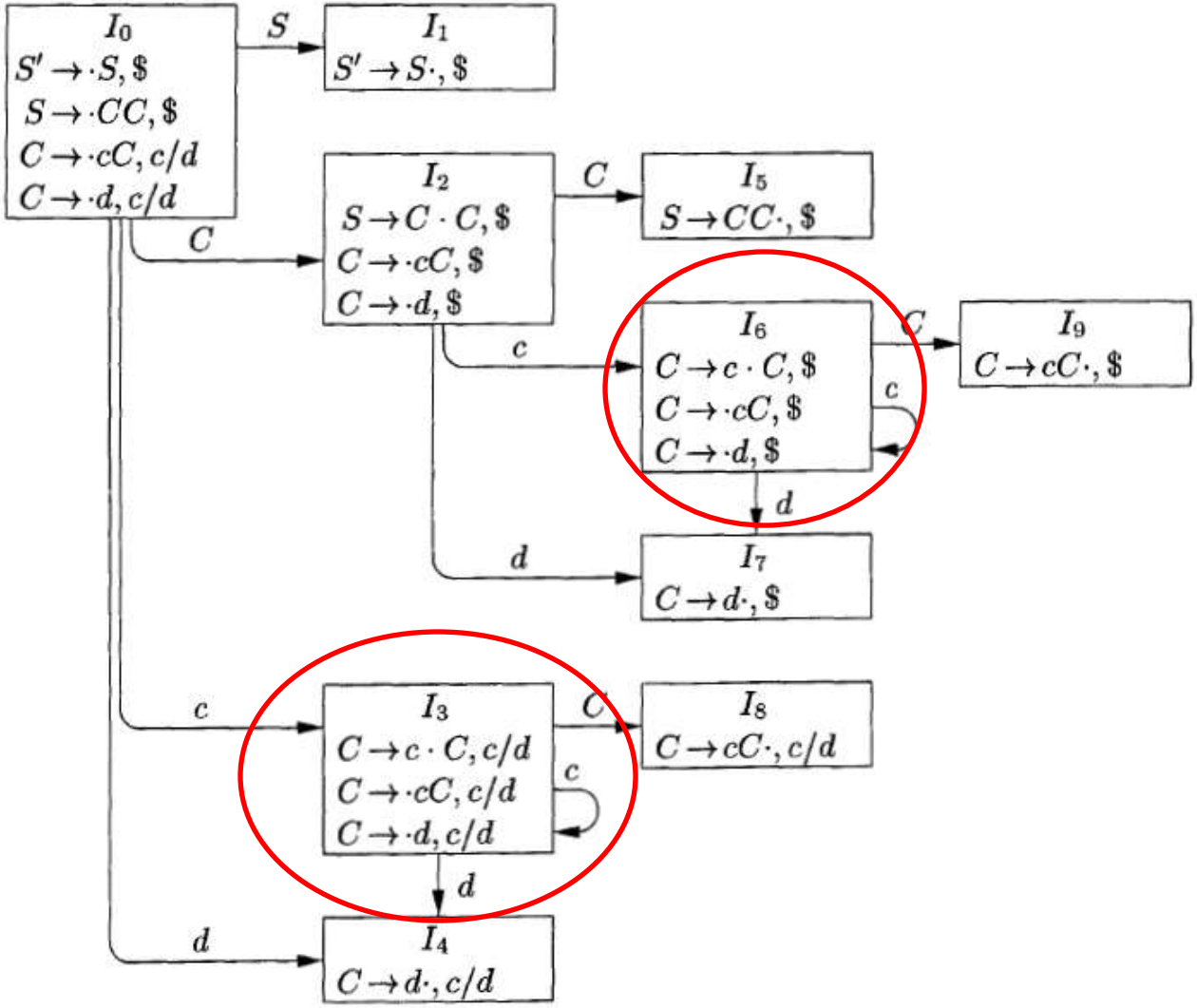
LALR Steps

- Get grammar G' from G
- Build different states
- start I_0 with {closure[S'->.S,$]}
- Then for each grammar symbol
  - Get new state
  - Get closure of the new state
- Reduce states by merging items with same cores
- Build ACTION and GOTO tables

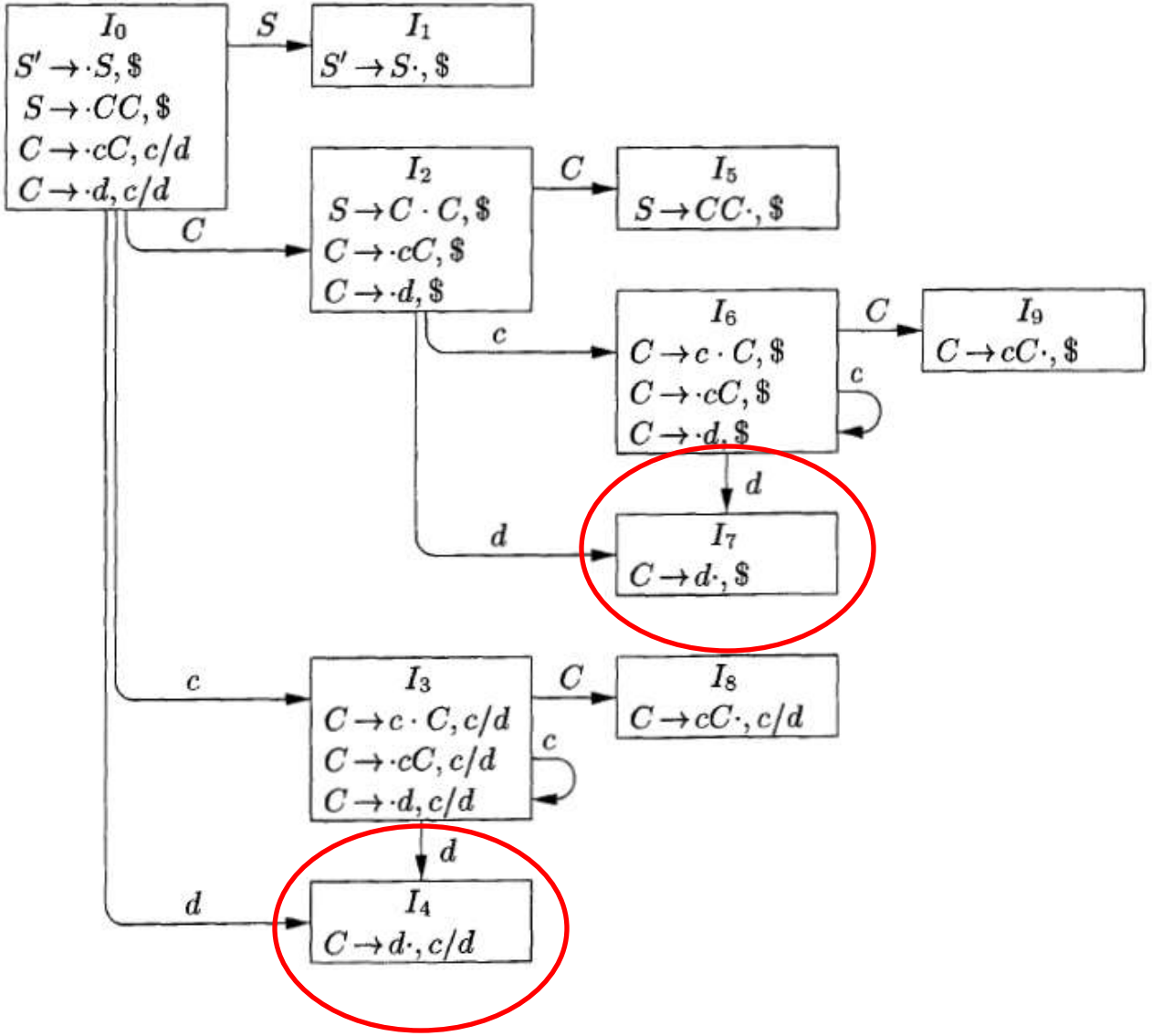
$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & C\ C \\
C & \rightarrow & c\ C \mid d
\end{array}
$$

```
SetOfItems CLOSURE(I) {
    repeat
        for ( each item [A → α·Bβ, a] in I )
            for ( each production B → γ in G' )
                for ( each terminal b in FIRST(βa) )
                    add [B → ·γ, b] to set I;
    until no more items are added to I;
    return I;
}
```

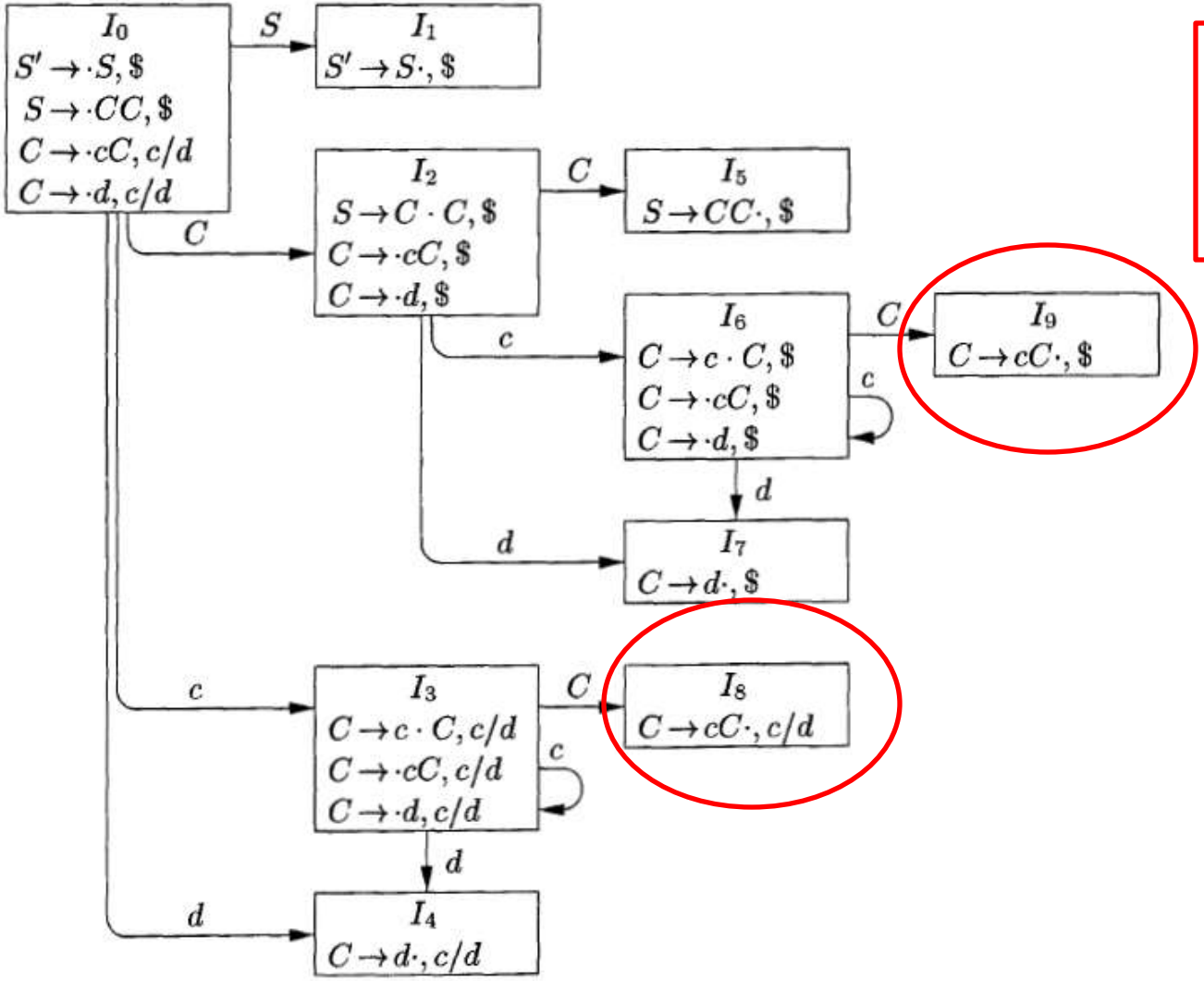




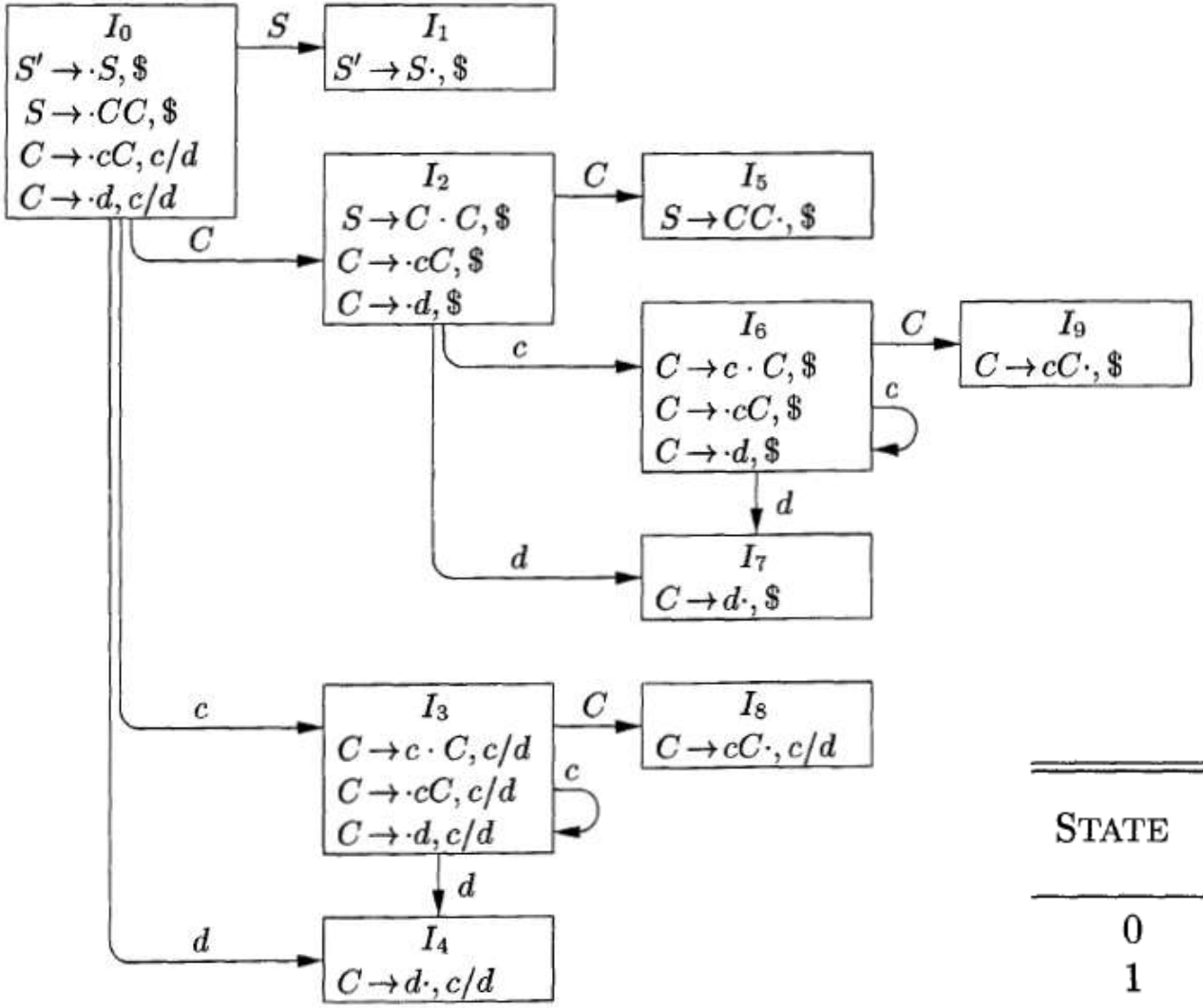
I_{36}: $C \to c\cdot C,\ c/d/\$$
$C \to \cdot cC,\ c/d/\$$
$C \to \cdot d,\ c/d/\$$



I_{47}: $C \to d\cdot,\ c/d/\$$



I_{89}: $C \to cC\cdot,\ c/d/\$$



I_{36}: $C \to c\cdot C,\ c/d/\$$
$C \to \cdot cC,\ c/d/\$$
$C \to \cdot d,\ c/d/\$$

I_{47}: $C \to d\cdot,\ c/d/\$$

I_{89}: $C \to cC\cdot,\ c/d/\$$

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Conclusion

- skim: 4.6.5, 4.7.5, 4.7.6, 4.8
- Read rest of: 4.6->4.8
- We will need 4.9 (except 4.9.2) for project part 2