Principles of Compiler Construction

Lecture 8 Syntax Analysis (IV)

Lecturer: CHANG HUIYOU

Note that most of these slides were created by:

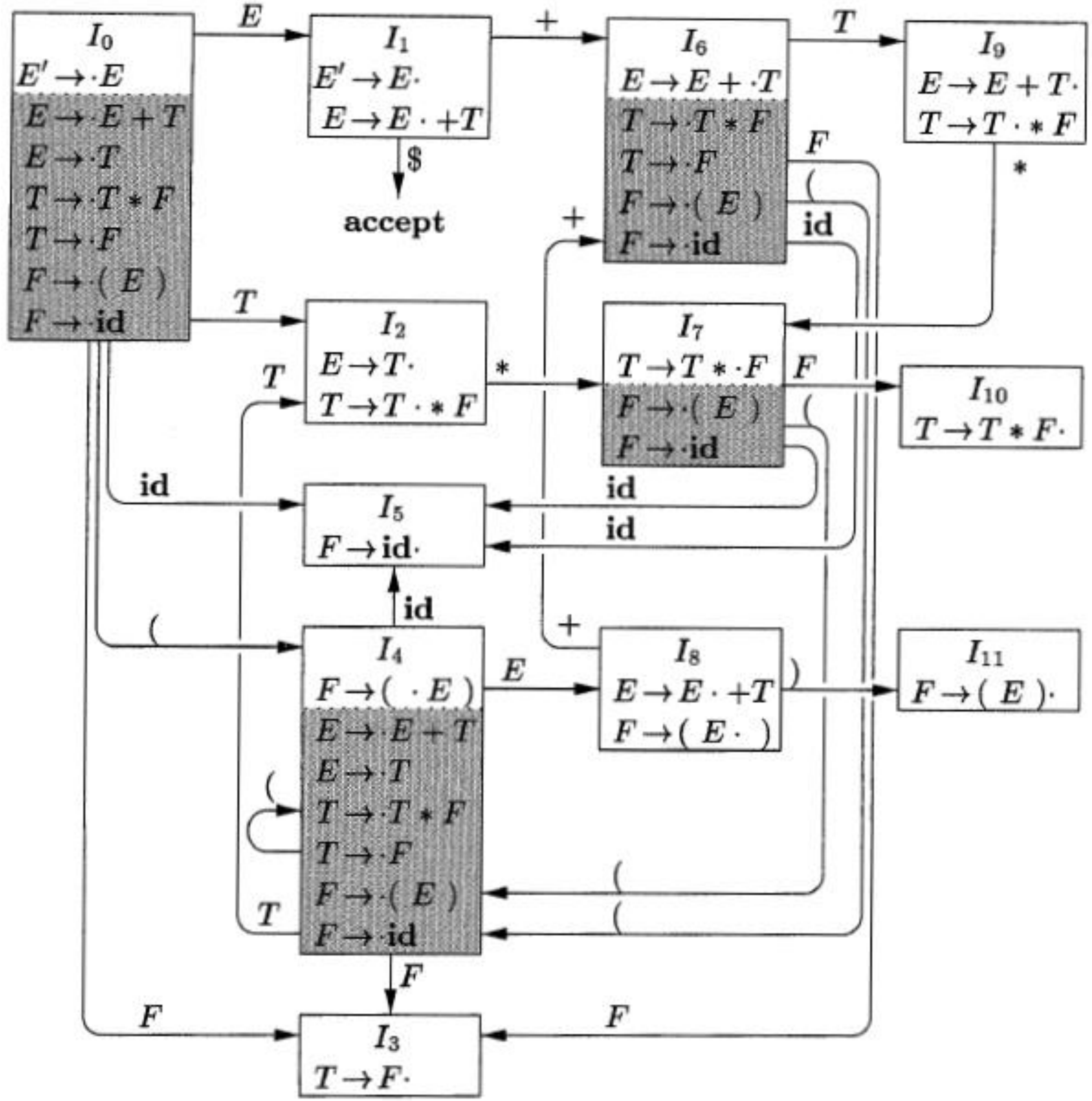
Prof. Wen-jun LI (School of Software)

回顾: *Shift-Reduce*

STACK	INPUT	ACTION
$\$$	$\mathbf{id}_1 * \mathbf{id}_2\,\$$	shift
$\$\,\mathbf{id}_1$	$* \mathbf{id}_2\,\$$	reduce by $F \to \mathbf{id}$
$\$\,F$	$* \mathbf{id}_2\,\$$	reduce by $T \to F$
$\$\,T$	$* \mathbf{id}_2\,\$$	shift
$\$\,T *$	$\mathbf{id}_2\,\$$	shift
$\$\,T * \mathbf{id}_2$	$\$$	reduce by $F \to \mathbf{id}$
$\$\,T * F$	$\$$	reduce by $T \to T * F$
$\$\,T$	$\$$	reduce by $E \to T$
$\$\,E$	$\$$	accept

中山大學

回顾：$LR(0)$ 自动机



回顾: *LR Parsing Table*

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$

(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$

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			

回顾：非SLR文法

$$
\begin{aligned}
S &\rightarrow L = R \mid R \\
L &\rightarrow *R \mid \mathbf{id} \\
R &\rightarrow L
\end{aligned}
$$

I_0

S' → • S
S → • L = R
S → • R
L → • * R
L → • **id**
R → • L

$I_0 \xrightarrow{L} I_2$

...

I_2

S → L • = R
R → L •

A **shift/reduce conflict** on input symbol '**=**', since '**=**' ∈ FOLLOW(R)

问题：如何避免这种移进/归约冲突？

思考

凡事预则立，不预则废。

——《礼记·中庸》

如果不是等到可能要归约的时候才来根据FOLLOW集判断能否归约，而是一开始就做好准备，会不会好一些?

LR(1) Items

[A→α·β, a]

A→αβ is a production

a is a terminal or $

A→α·β is called the core of this item

CLOSURE and GOTO

```
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;
}

SetOfItems GOTO(I, X) {
    initialize J to be the empty set;
    for ( each item [A → α·Xβ, a] in I )
        add item [A → αX·β, a] to set J;
    return CLOSURE(J);
}
```

An Example

Consider the following grammar:

(0) S' $\rightarrow$ S

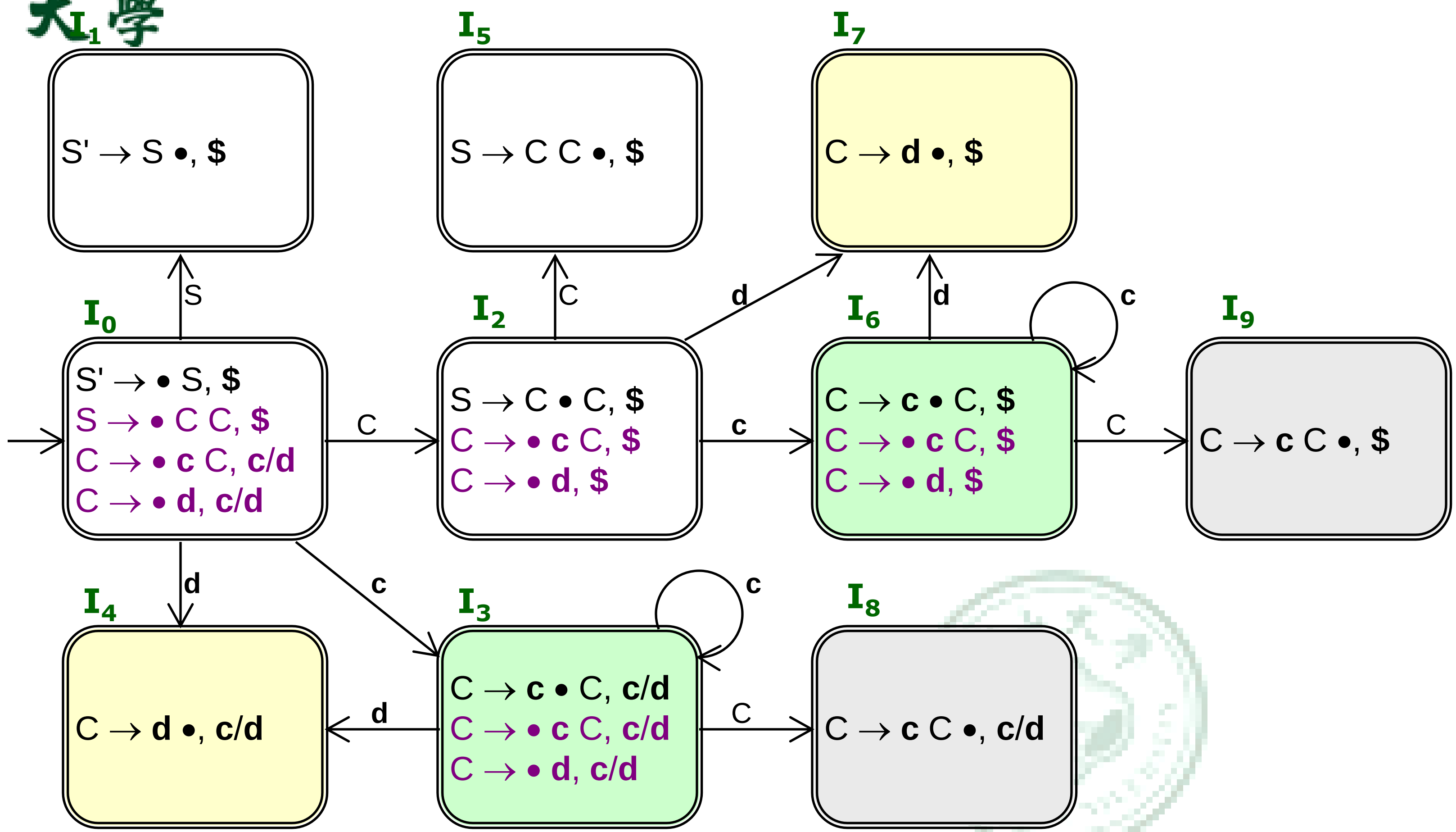
(1) S $\rightarrow$ C C

(2) C $\rightarrow$ **c** C

(3) C $\rightarrow$ **d**

It is easy to calculate the following sets:

FIRST(S) = FIRST(C) = { c, d }



DFA recognizing all viable prefixes

LR(1) Parsing Table

State	ACTION			GOTO	
	c	**d**	**$**	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

LR(1) Grammar

The table produced by this algorithm is called the *canonical LR(1) parsing table*.

An LR parser using this table is called a canonical-LR(1) parser.

If the parsing action function has no multiply defined entries, then the given grammar is called an *LR(1) grammar*.

Disadvantage of LR(1) Parsing

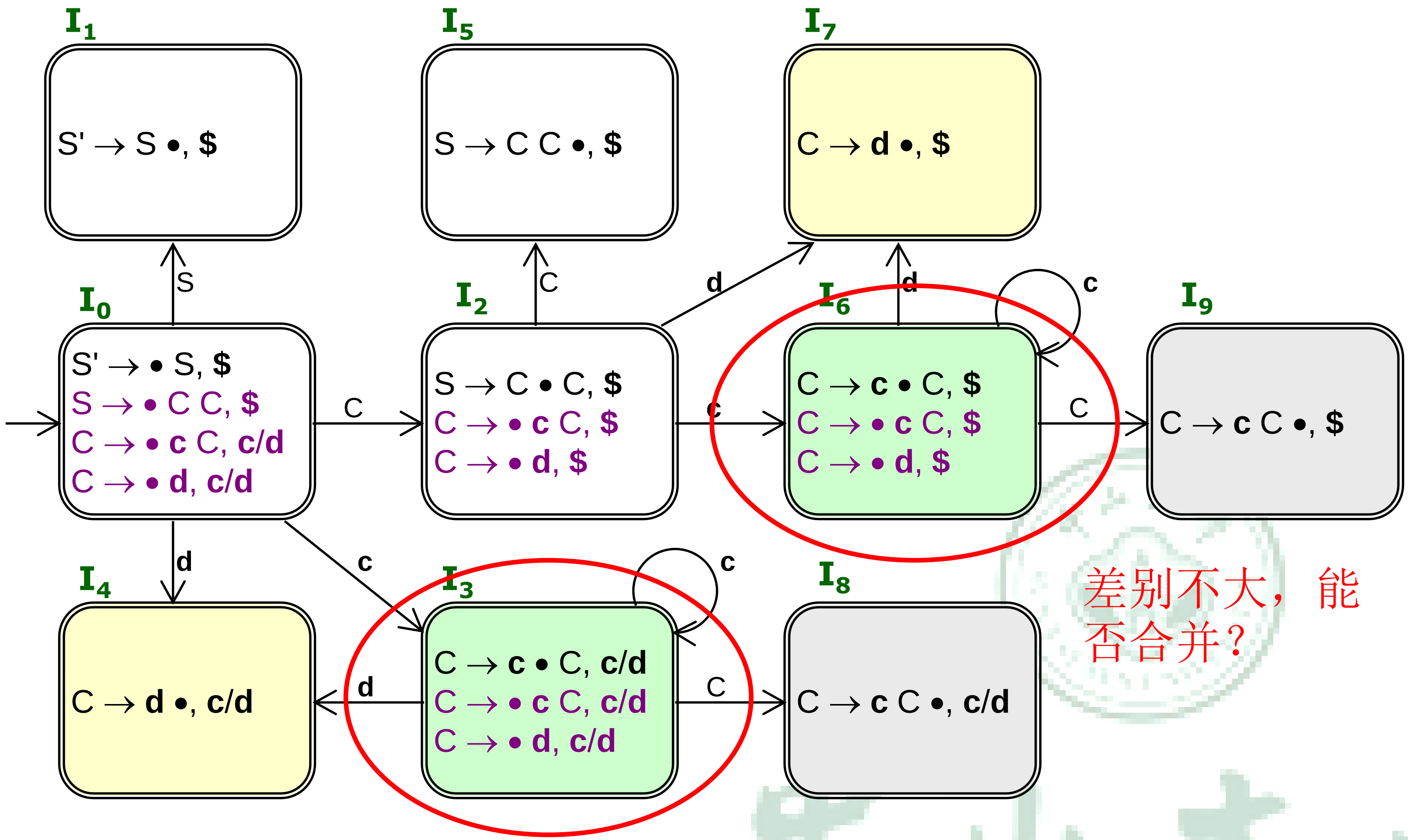
Too many states!

- **Number of states, such as C or Pascal**
  - LR(0) = SLR(1): several hundreds.
  - LR(1): several thousands (10 times).

LALR(Lookahead LR) Parsing

- Often used in practice
- LALR tables are considerably smaller than the canonical LR tables
- Most common syntactic constructs of programming languages can be expressed conveniently by an LALR grammar

思考



I1
S' → S •, $
I5
S → C C •, $
I7
C → d •, $
I0
S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d
I2
S → C • C, $
C → • c C, $
C → • d, $
I6
C → c • C, $
C → • c C, $
C → • d, $
I9
C → c C •, $
I4
C → d •, c/d
I3
C → c • C, c/d
C → • c C, c/d
C → • d, c/d
I8
C → c C •, c/d
S
C
d
c

差别不大，能否合并？

LALR Parsing

Strategy for LALR(1): merge the states with the same core.

E.g. I_3 and I_6, I_4 and I_7, I_8 and I_9

Lookaheads of the same item are merged.

GOTO() depends only on the core.

LALR vs. LR(1)

The merge will never produce new shift-reduce conflicts

Suppose in the merged state

[$A \rightarrow \alpha \bullet$, a] calls for a reduction

[$B \rightarrow \beta \bullet$ a γ, b] calls for a shift

Since the original states have the same core, there must be some state have

[$A \rightarrow \alpha \bullet$, a] calls for a reduction

[$B \rightarrow \beta \bullet$ a γ, c] calls for a shift (for some c)

Then the original state already has conflicts.

LALR vs. LR(1) (cont')

But the merge will produce new reduce-reduce conflicts

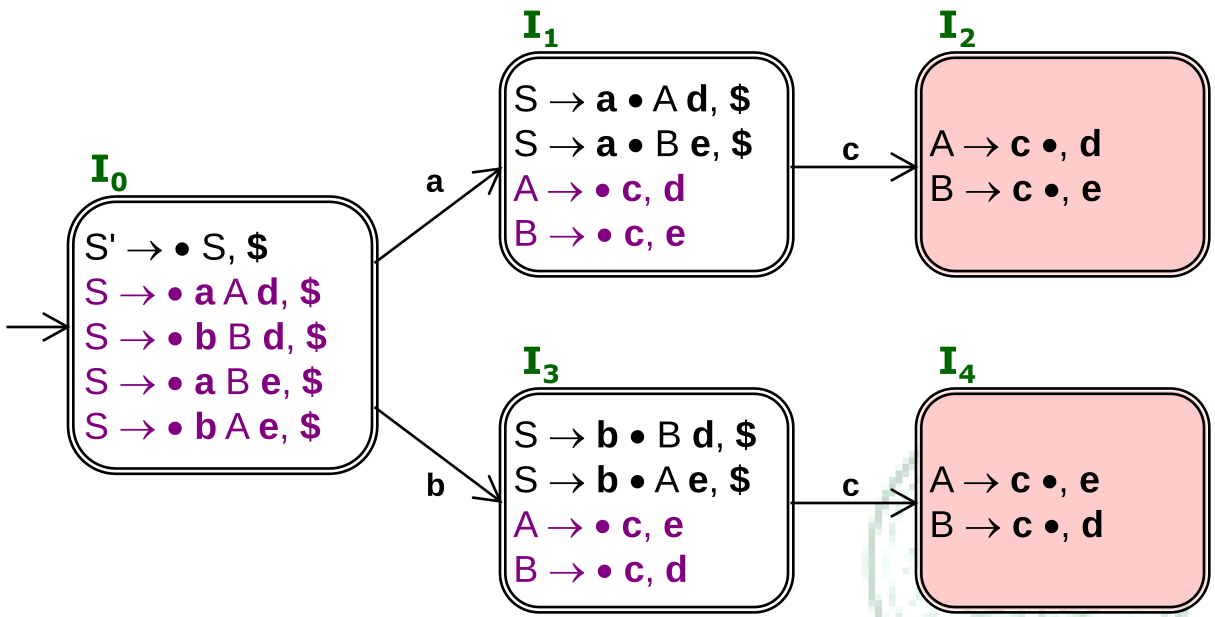
For example

- S' $\rightarrow$S
- S $\rightarrow$**a** A **d** | **b** B **d** | **a** B **e** | **b** A **e**
- A $\rightarrow$**c**
- B $\rightarrow$**c**

{[A $\rightarrow$ c •, d], [B $\rightarrow$ c •, e]} is valid for viable prefix ac, {[A $\rightarrow$ c •, e], [B $\rightarrow$ c •, d]} is valid for viable prefix bc. But the merge has conflicts:

{[A $\rightarrow$ **c •**, **d/e**], [B $\rightarrow$ **c •**, **d/e**]}

Example: New Conflicts in LALR

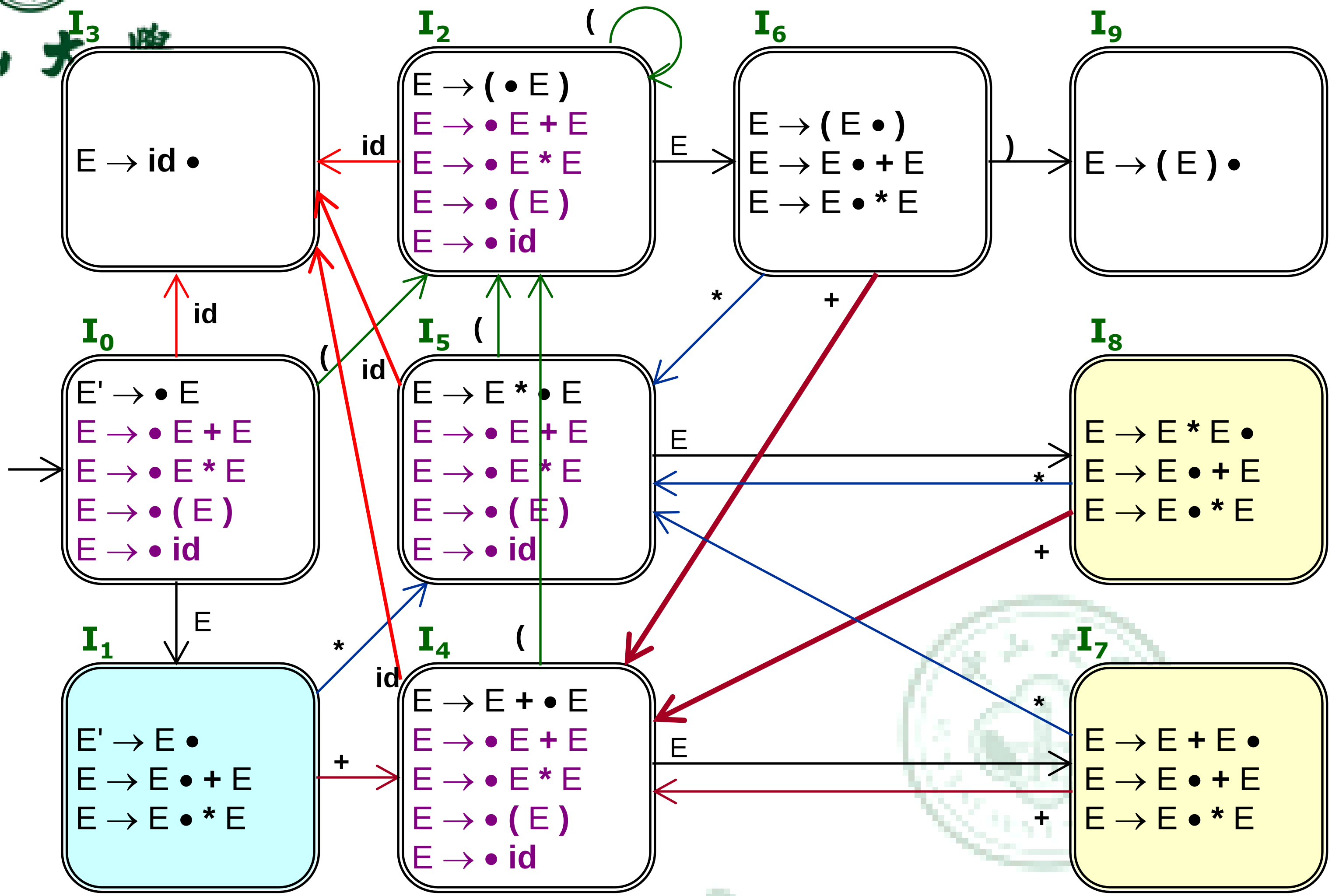


Part of the DFA recognizing all viable prefixes

Ambiguous Expression Grammar

Given the ambiguous grammar

- (0) E' $\rightarrow$ E
- (1) E $\rightarrow$ E **+** E
- (2) E $\rightarrow$ E ***** E
- (3) E $\rightarrow$ **(** E **)**
- (4) E $\rightarrow$ **id**



DFA recognizing all viable prefixes

shift-reduce conflict

中山大學

SLR(1) Parsing Table

State	ACTION						GOTO
	id	**+**	*****	**(**	**)**	**$**	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1/s4	r1/s5		r1	r1	
8		r2/s4	r2/s5		r2	r2	
9		r3	r3		r3	r3	

Resolve ambiguities at the parsing table level

*Dangling-**else** Grammar*

Given the ambiguous grammar

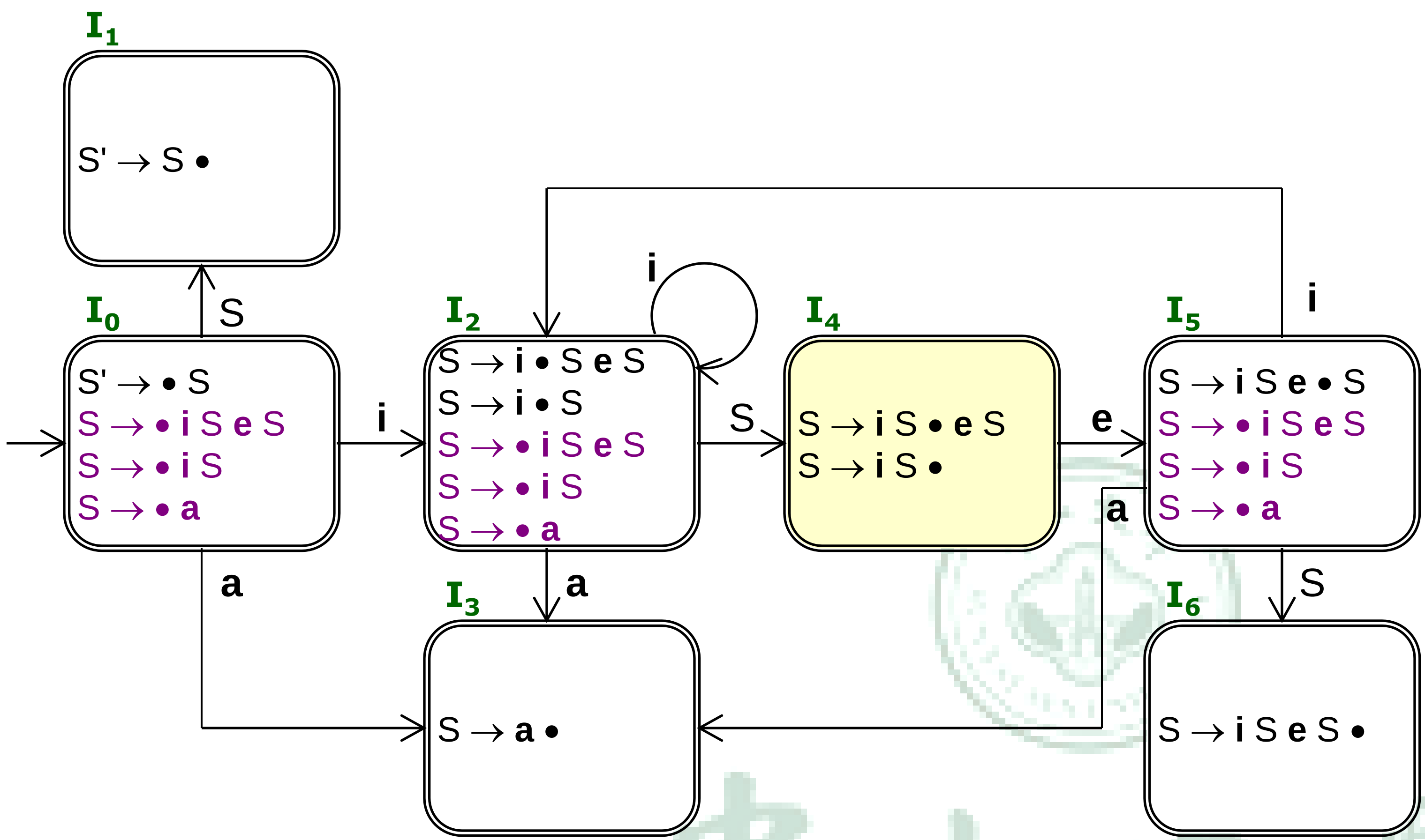
❿(0) S' →S

❿(1) S →**i** S **e** S

❿(2) S →**i** S

❿(3) S →**a**

DFA Recognizing All Viable Prefixes



shift-reduce conflict

SLR(1) Parsing Table

State	ACTION				GOTO
	i	**e**	**a**	**$**	S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		r2/s5		r2	
5	s2		s3		6
6		r1		r1	

Resolve ambiguities at the parsing table level

Parsing a Sentence

Step	Symbol	State	Input	Reference	Action	Output
1	**$**	0	**i i a e a $**	a[0, **i**] = s2	shift	
2	**$ i**	0 2	**i a e a $**	a[2, **i**] = s2	shift	
3	**$ i i**	0 2 2	**a e a $**	a[2, **a**] = s3	shift	
4	**$ i i a**	0 2 2 3	**e a $**	a[3, **e**] = r3 g[2, S] = 4	reduce	S → **a**
5	**$ i i** S	0 2 2 4	**e a $**	a[4, **e**] = s5	shift	
6	**$ i i** S **e**	0 2 2 4 5	**a $**	a[5, **a**] = s3	shift	
7	**$ i i** S **e a**	0 2 2 4 5 3	**$**	a[3, **$**] = r3 g[5, S] = 6	reduce	S → **a**
8	**$ i i** S **e** S	0 2 2 4 5 6	**$**	a[6, **$**] = r1 g[2, S] = 4	reduce	S → **i** S **e** S
9	**$ i** S	0 2 4	**$**	a[4, **$**] = r2 g[0, S] = 1	reduce	S → **i** S
10	**$** S	0 1	**$**	a[1, **$**] = acc	accept	

Error Recovery

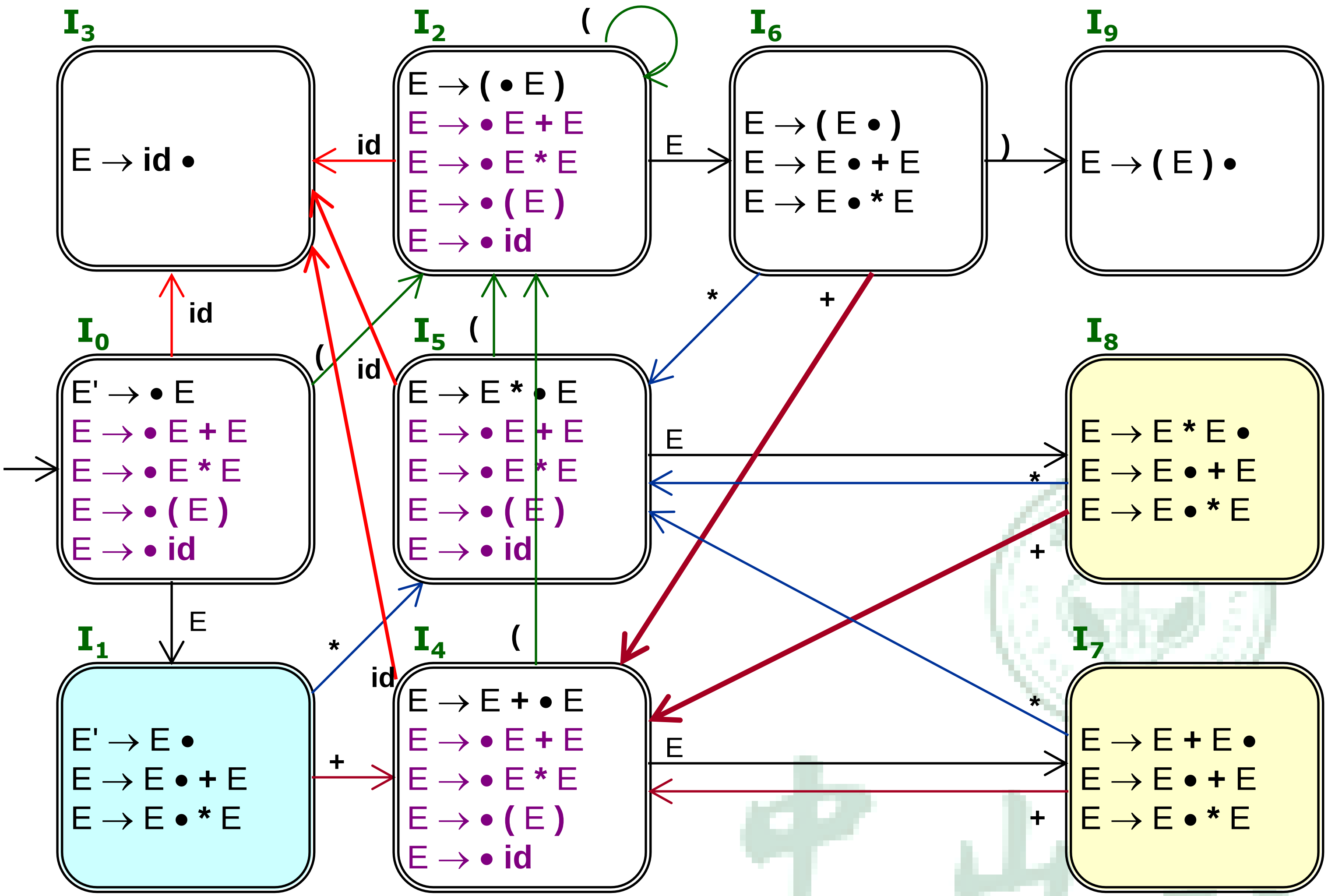
Panic-mode error recovery

Phrase-level recovery

Error-Productions

Global-Correction

Example


I3
E → id •
I2
E → (• E)
E → • E + E
E → • E * E
E → • (E)
E → • id
I6
E → (E •)
E → E • + E
E → E • * E
I9
E → (E) •
I0
E' → • E
E → • E + E
E → • E * E
E → • (E)
E → • id
I5
E → E * • E
E → • E + E
E → • E * E
E → • (E)
E → • id
I8
E → E * E •
E → E • + E
E → E • * E
I1
E' → E •
E → E • + E
E → E • * E
I4
E → E + • E
E → • E + E
E → • E * E
E → • (E)
E → • id
I7
E → E + E •
E → E • + E
E → E • * E
id
(
E
)
*
+

Example

State	ACTION						GOTO
	id	**+**	*****	**(**	**)**	**$**	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

Postpone error detection until one or more reductions are made

Error-Handling Routines

e1: an operand 'id' or '(' is expected.

push state 3; // add a symbol 'id'

e2: unbalanced right parenthesis.

drop one lookahead; // remove ')'

e3: an operator is expected.

push state 4; // add a symbol '+'

e4: a right parenthesis is expected.

push state 9; // add a symbol ')'

中山大學

Parsing an Erroneous Input

Step	Symbol	State	Input	Reference	Action	Output
1	**$**	0	**id +) $**	a[0, **id**] = s3	shift	
2	**$ id**	0 3	**+) $**	a[3, **+**] = r4 g[0, E] = 1	reduce	E → **id**
3	**$** E	0 1	**+) $**	a[1, **+**] = s4	shift	
4	**$** E **+**	0 1 4	**) $**	a[4, **)**] = e2	drop	Unbalanced ')'
5	**$** E **+**	0 1 4	**$**	a[4, **$**] = e1	push 3	Operand expected
6	**$** E **+ id**	0 1 4 3	**$**	a[3, **$**] = r4 g[4, E] = 7	reduce	E → **id**
7	**$** E **+** E	0 1 4 7	**$**	a[7, **$**] = r1 g[0, E] = 1	reduce	E → E **+** E
8	**$** E	0 1	**$**	a[1, **$**] = acc	end	

See you next time!