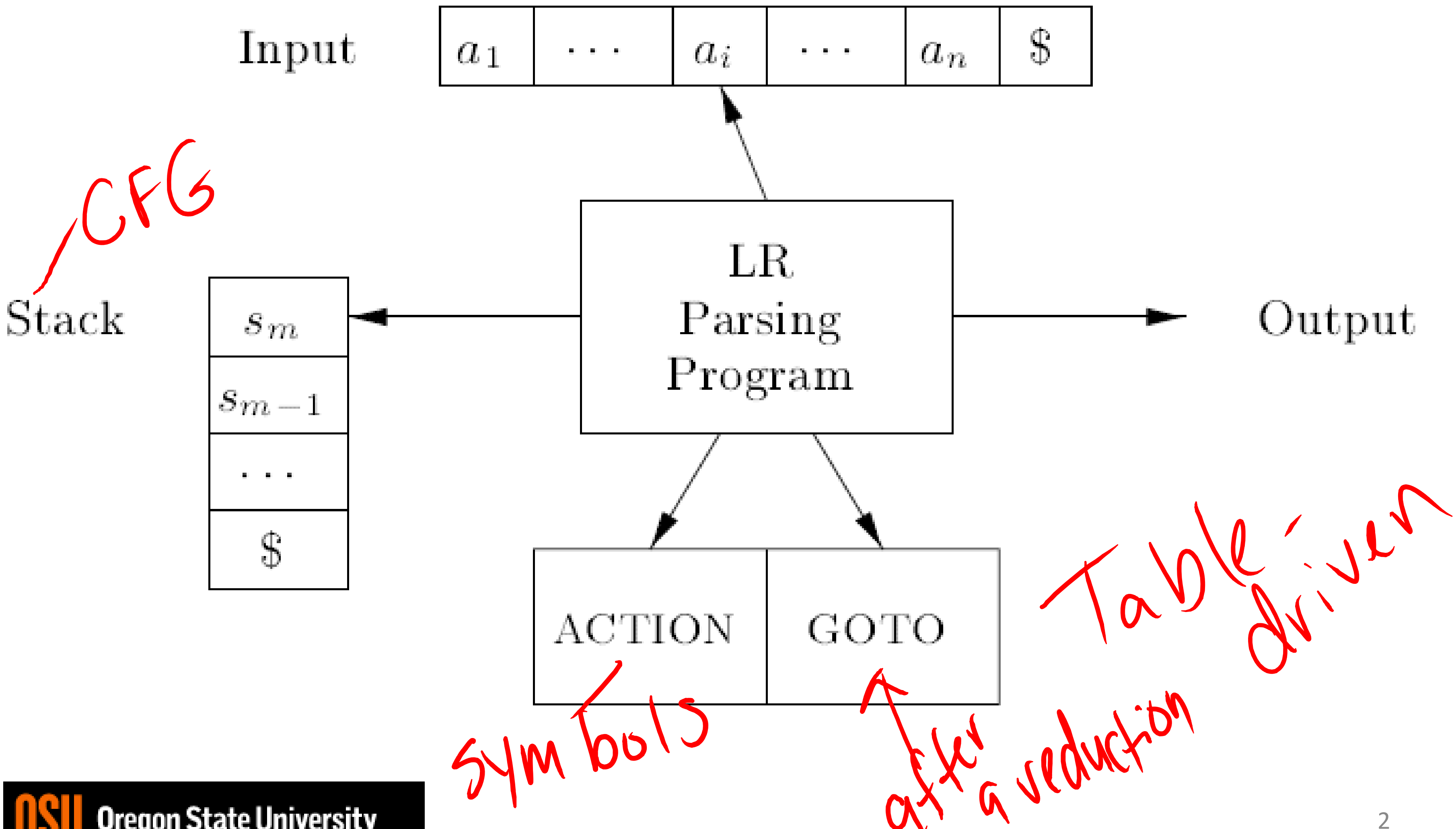
CS480
Translators

What is Bottom Up Parsing?

Chap. 4

Oregon State University

LR Parsing

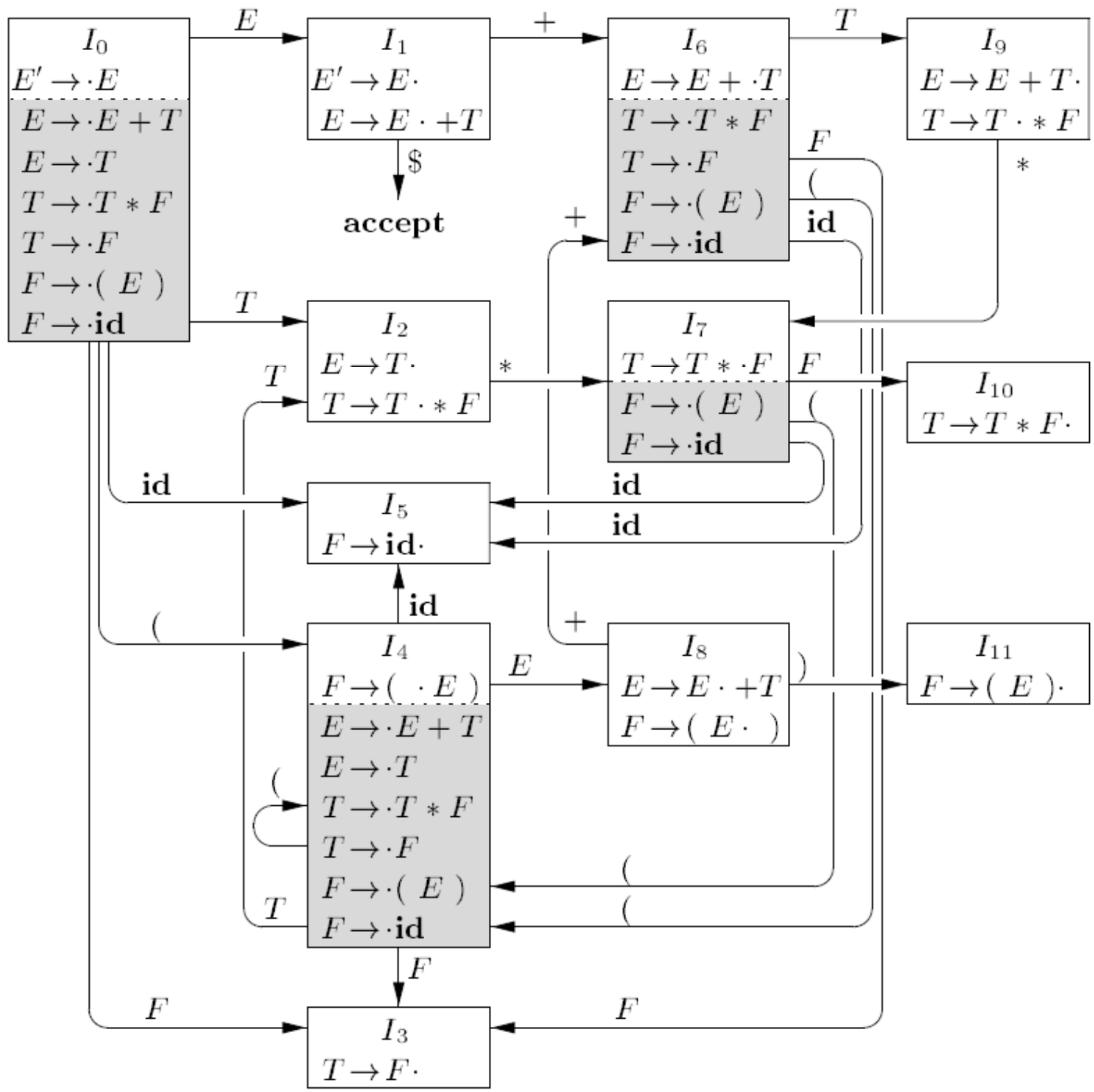

Input
a_1 ... a_i ... a_n \$
CFG
Stack
s_m
s_{m-1}
...
\$
LR Parsing Program
Output
ACTION
GOTO
Table-driven
symbols
after a reduction

LR Parse Table

- Build NFA out of productions
- Convert NFA to DFA

Theory

- Create action table (states, terminals)
- Create goto table (states, nonterms)
- Let's build a parse table…



I0
E' → ·E
E → ·E + T
E → ·T
T → ·T * F
T → ·F
F → ·(E)
F → ·id
I1
E' → E·
E → E · +T
$
accept
I6
E → E + ·T
T → ·T * F
T → ·F
F → ·(E)
F → ·id
I9
E → E + T·
T → T · * F
I2
E → T·
T → T · * F
I7
T → T * ·F
F → ·(E)
F → ·id
I10
T → T * F·
I5
F → id·
I4
F → (·E)
E → ·E + T
E → ·T
T → ·T * F
T → ·F
F → ·(E)
F → ·id
I8
E → E · +T
F → (E ·)
I11
F → (E)·
I3
T → F·

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			

Figure 4.37: Parsing table for expression grammar

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Figure 4.36: LR-parsing program

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\ \$$	shift
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\ \$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\ \$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\ \$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\ \$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\ \$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\ \$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\ \$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\ \$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\ \$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept

Figure 4.38: Moves of an LR parser on $\mathbf{id} * \mathbf{id} + \mathbf{id}$

LR Parse Table

- Time Consuming to construct by hand
- Parser Generator used, i.e. Yacc

telling us when to shift & when to reduce

Oregon State University

Example Grammar

E -> E**+**T

E -> T

T -> **(**E**)**

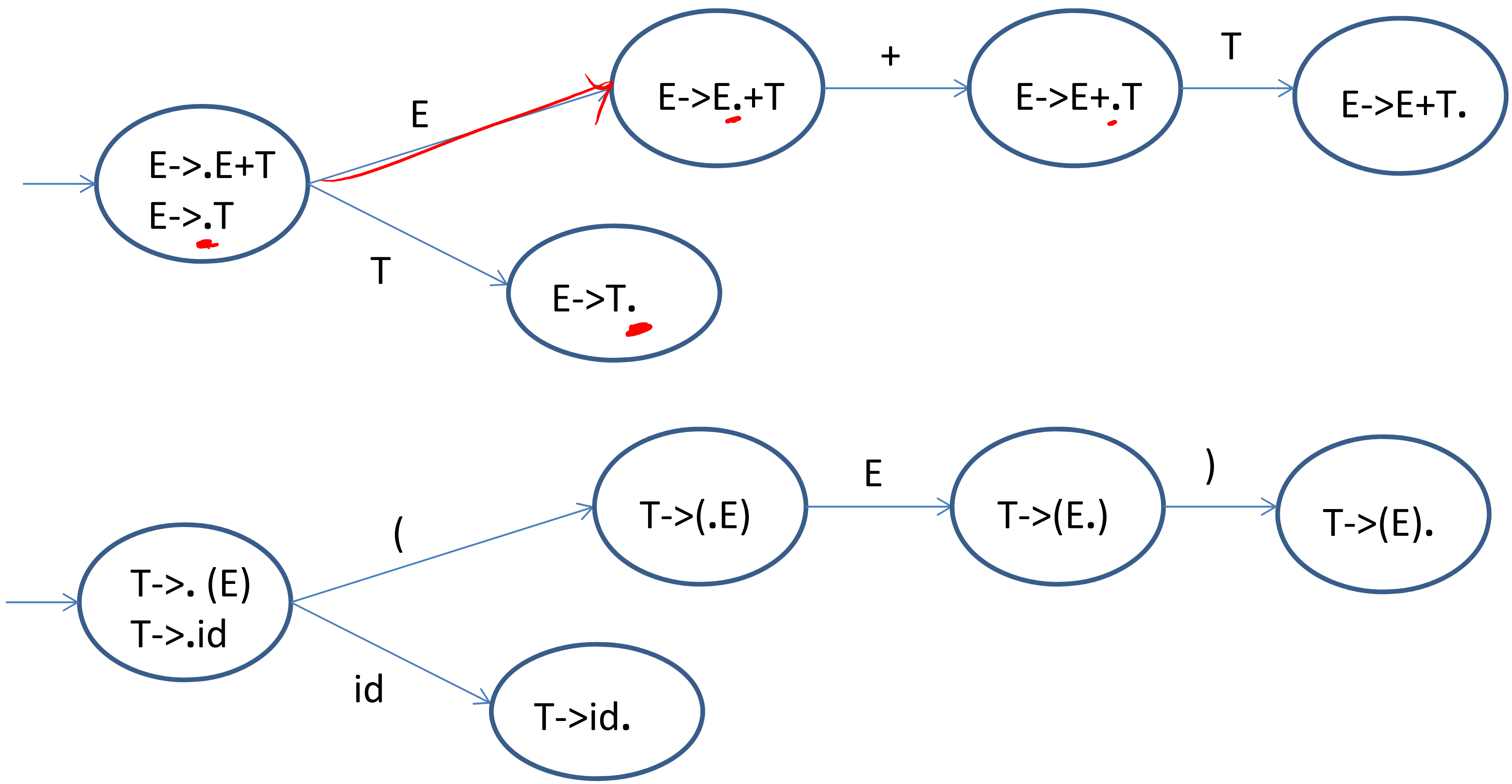
T -> **id**

- How might we construct a DFA for this?

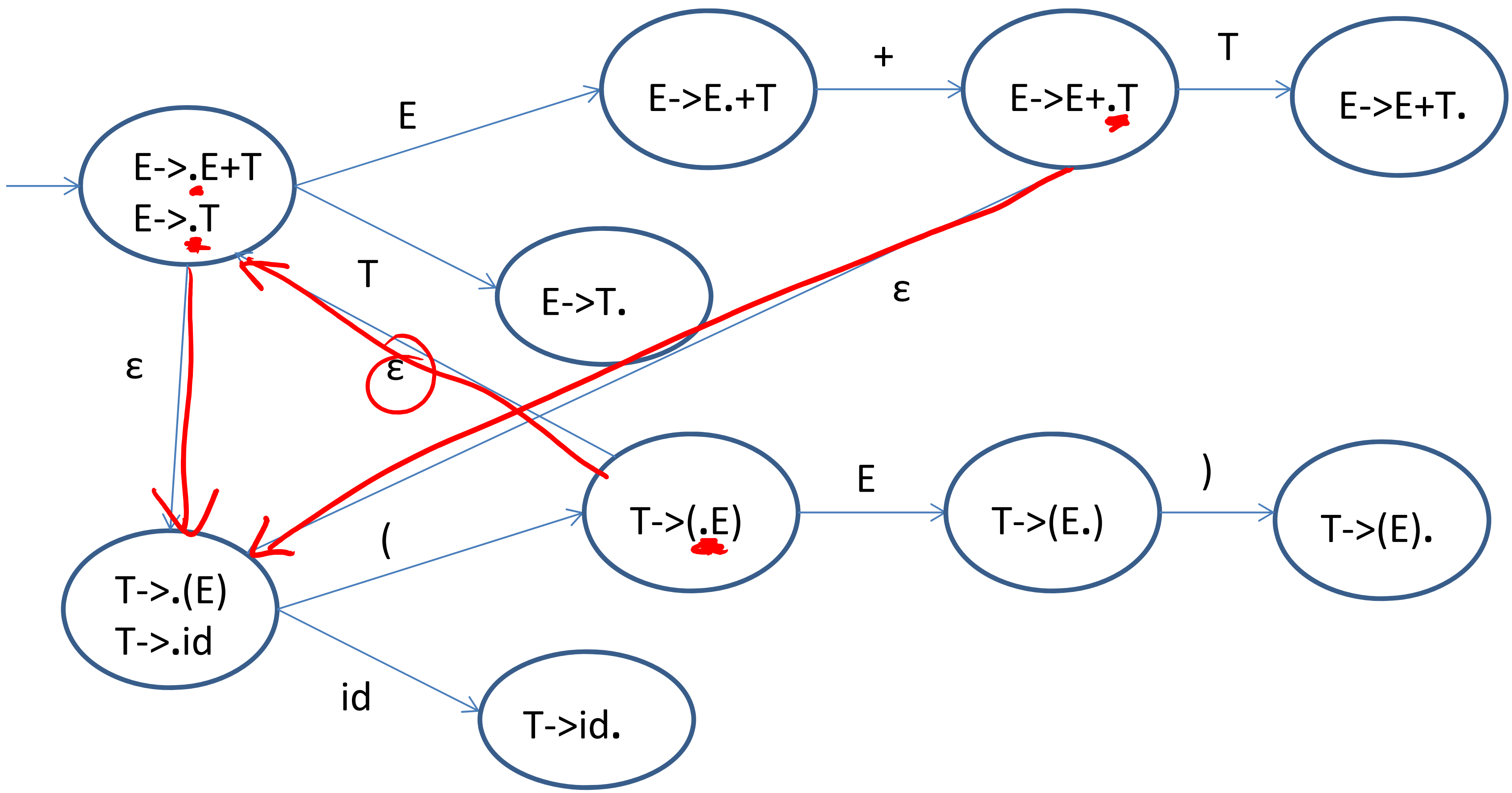
DFAs to NFA to DFA

- Create a DFA for each production
- Connect them together w/ ε
- Use subset construction to create big DFA
- Example…

Create DFAs



Create NFA w/ ε

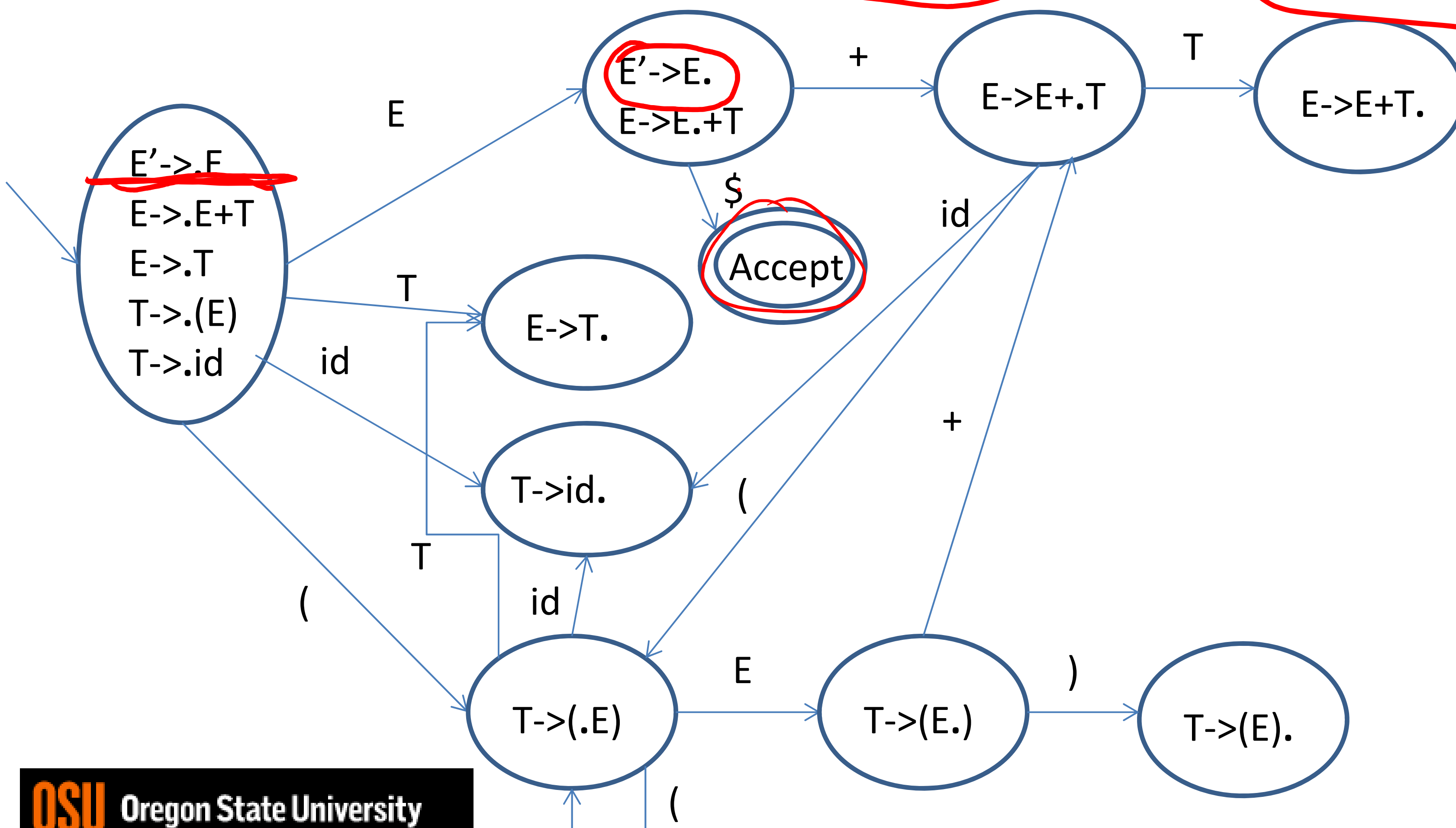


OSU Oregon State University

Subset Construction for DFA

shift / reduce

- Augment Grammar: add E'->E



OSU Oregon State University

Use Closure for DFA…

SetOfItems $\textsc{closure}(I)$ {
    $J = I$;
    repeat
        for (each item $A \to \alpha{\cdot}B\beta$ in J)
            for (each production $B \to \gamma$ of G)
                if ($B \to \cdot\gamma$ is not in J)
                    add $B \to \cdot\gamma$ to J;
    until no more items are added to J on one round;
    return J;
}

Figure 4.32: Computation of $\textsc{closure}$

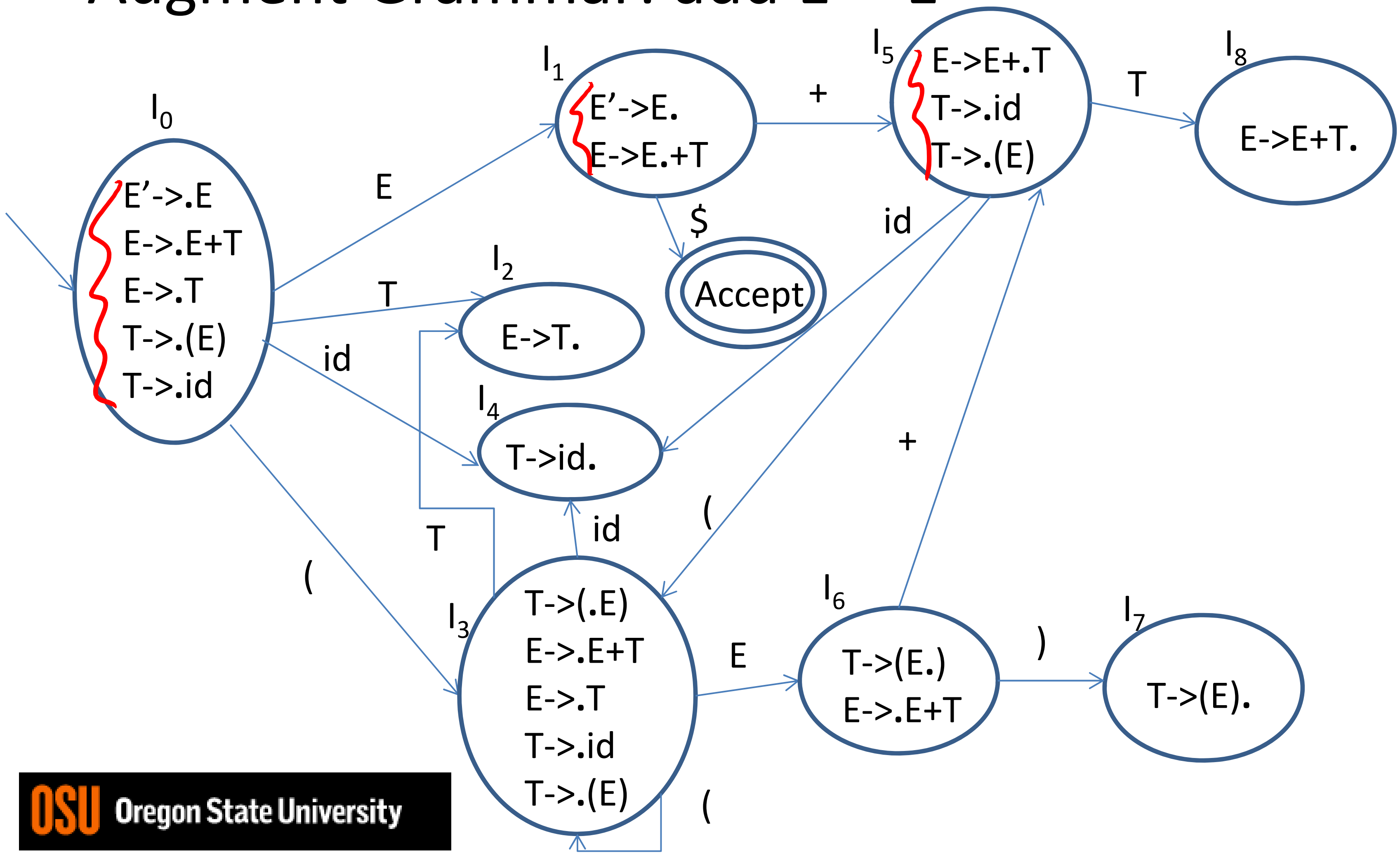
Closure Example…

nonterm

- I_0 {E'->.E, E->.E+T, E->.T, T->.id, T->.(E)}
- I_1 {E'->E., E->E.+T}
- I_2 {E->T.}
- I_3 {T->(.E), E->.E+T, E->.T, T->.id, T->.(E)}
- I_4 {T->id.}
- I_5 {E->E+.T, T->.id, T->.(E)}
- I_6 {T->(E.)}
- I_7 {T->(E).}
- I_8 {E->E+T.}

Closure Example

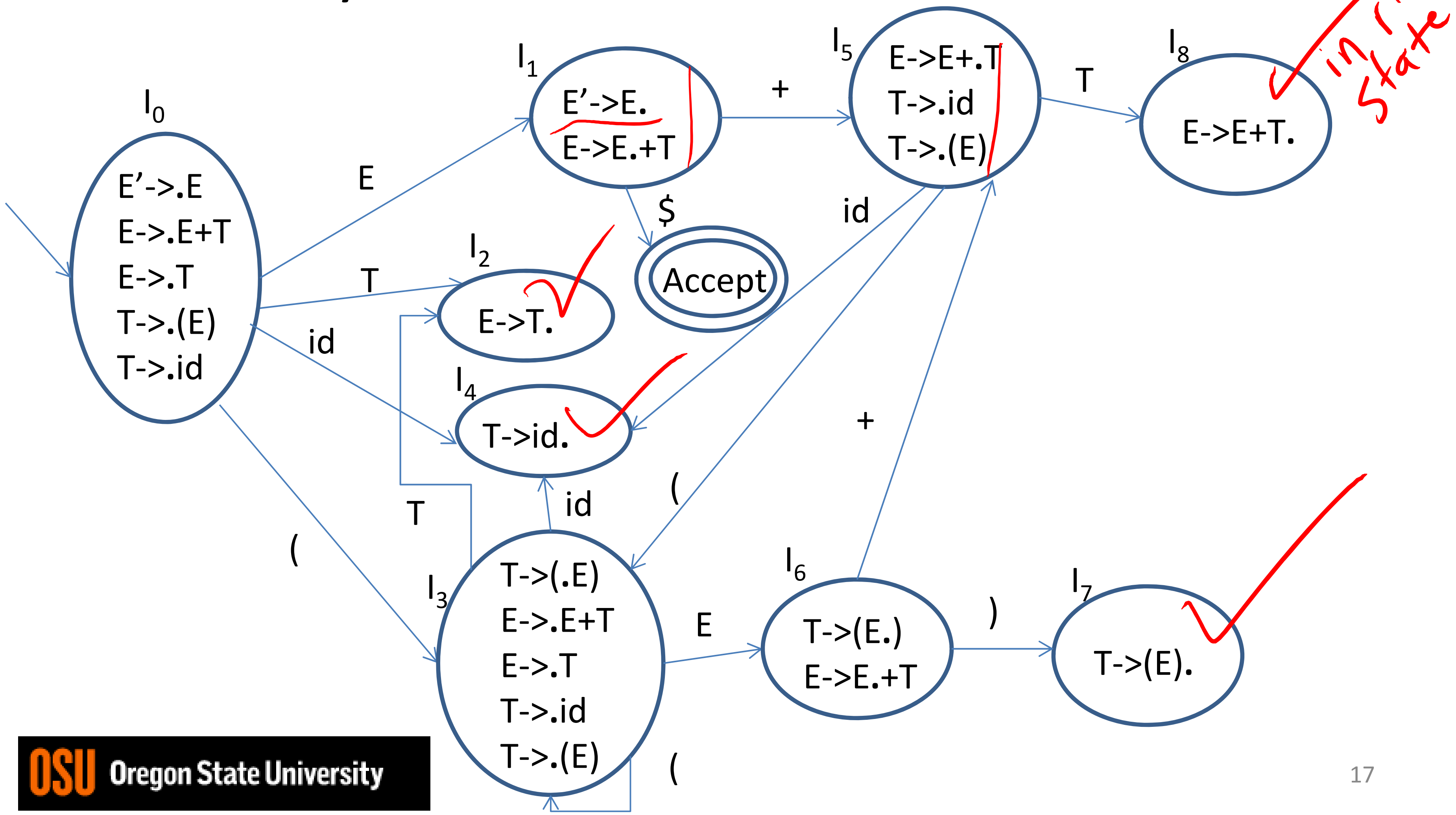
- Augment Grammar: add E'->E



OSU Oregon State University

LR(0) Automaton

- What do you notice about the reduce states



Collection of Configurating Sets

if dot in front of Non-term

Configurating set	*Successor*
I0: E' –> •E	I1
E –> •E+T	I1
E –> •T	I2
T –> •(E)	I3
T –> •id	I4
I1: E' –> E•	Accept
E –> E•+T	I5
I2: E –> T•	Reduce 2
I3: T –> (•E)	I6
E –> •E+T	I6
E –> •T	I2
T –> •(E)	I3
T –> •id	I4
I4: T –> id•	Reduce 4

Do closure

2 GoTos

3 Shifts in action

which production

Configurating set	*Successor*
I5: E –> E+•T	I8
T –> •(E)	I3
T –> •id	I4
I6: T –> (E•)	I7
E –> E•+T	I5
I7: T –> (E)•	Reduce 3
I8: E –> E+T•	Reduce 1

Construct LR(0) Table

1. Construct $F = \{I_0, I_1, \ldots I_n\}$, the collection of configurating sets for G'.

2. State i is determined from I_i. The parsing actions for the state are determined as follows:

   a) If A –> <u>u</u>• is in I_i then set Action[i,a] to reduce A –> <u>u</u> for all input. (A not equal to S').

   b) If S' –> S• is in I_i then set Action[i,$] to accept.

   c) If A –> <u>u</u>•a<u>v</u> is in I_i and successor(I_i, a) = I_j, then set Action[i,a] to shift j (a is a terminal).

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If successor(I_i, A) = I_j, then Goto [i, A] = j.

4. All entries not defined by rules 2 and 3 are errors.

5. The initial state is the one constructed from the configurating set containing S' –>S.

E→E+T ①
E→T ②
T→(E) ③
T→id ④

Create LR(0) Parse Table

State on Stack	Id	+	(	)	$	E	T
	Action					GoTo	
0	S4		S3			1	2
1		S5			Accept		
2	R2	R2	R2	R2	R2		
3	S4		S3			6	2
4	R4	R4	R4	R4	R4		
5	S4		S3				8
6		S5		S7			
7	R3	R3	R3	R3	R3		
8	R1	R1	R1	R1	R1		

- Let's parse: id + (id) $

Stack

Oregon State University

LR(0) Conditions

shifting

shift

state

reducing

1. For any configurating set containing the item $\boxed{A \rightarrow \underline{u} \bullet x \underline{v}}$ there is no complete item $\boxed{B \rightarrow \underline{w} \bullet}$ in that set. In the tables, this translates to no shift-reduce conflict on any state. This means the successor function from that set either shifts to a new state or reduces, but not both.

have 1 reduce

2. There is at most one complete item $\boxed{A \rightarrow \underline{u} \bullet}$ in each configurating set. This translates to no reduce-reduce conflict on any state. The successor function has at most one reduction.

Oregon State University

Quiz #$\cancel{7}$6

- S->0S1 | 01 Indicate the handle for:
  - 000111
  - 00S11
- S->SS+ | SS* | a Indicate the handle for:
  - SSS+a*+
  - SS+a*a+
  - aaa*a++
- Give a bottom up parse for 000111 and aaa*a++