

CS 432

Fall 2025

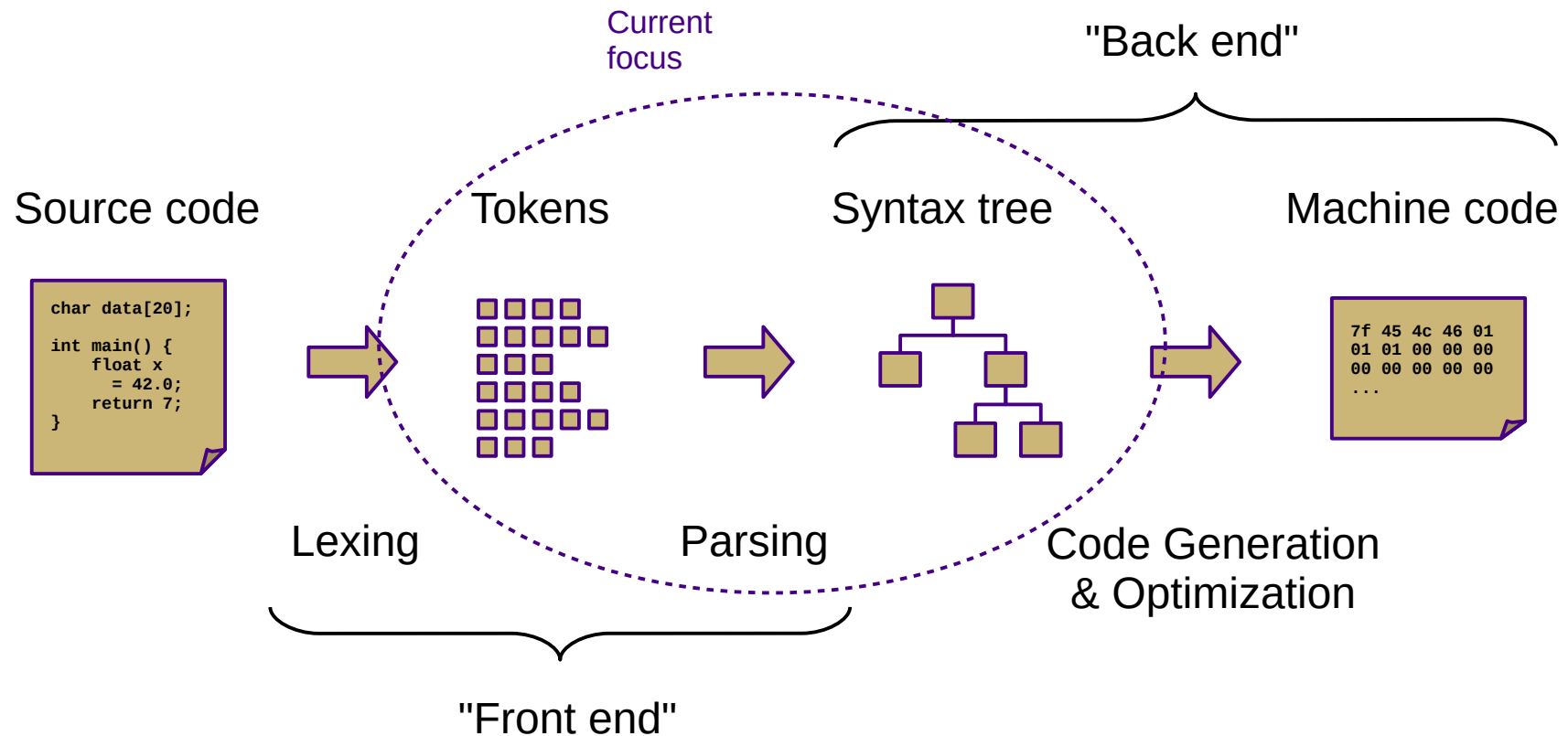
Mike Lam, Professor

(AN UNMATCHED LEFT PARENTHESIS
CREATES AN UNRESOLVED TENSION
THAT WILL STAY WITH YOU ALL DAY.

<https://xkcd.com/859/>

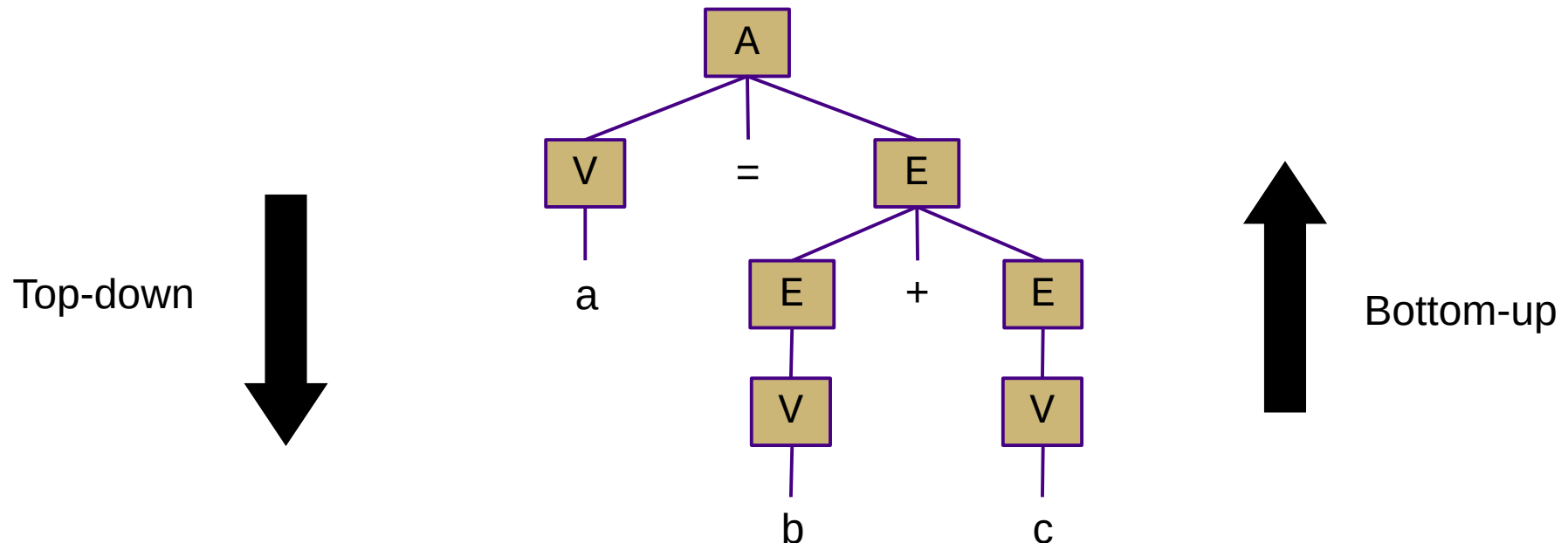
Bottom-Up (LR) Parsing

Compilation



Overview

- Two general parsing approaches
 - Top-down: begin with start symbol (root of parse tree), and gradually expand non-terminals
 - Bottom-up: begin with terminals (leaves of parse tree), and gradually connect using non-terminals



Shift-Reduce Parsing

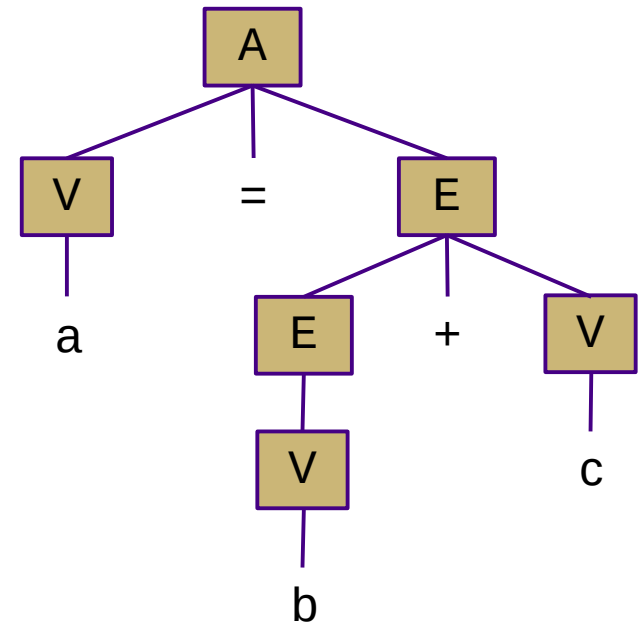
- Top-down (**LL**) parsers
 - Left-to-right scan, Leftmost derivation
 - Recursive routines, one per non-terminal (*recursive descent*)
 - Implicit stack (system call stack)
 - Requires more restrictive grammars
 - Simpler to understand and possible to hand-code
- Bottom-up (**LR**) parsers
 - Left-to-right scan, (reverse) Rightmost derivation
 - "Shift"/push terminals and non-terminals onto a stack
 - "Reduce"/pop to replace *handles* with non-terminals
 - Less restrictive grammars
 - Harder to understand and nearly always auto-generated; very efficient!

Shift-Reduce Parsing

- - shift 'a'
- a
 - reduce ($V \rightarrow a$)
- V
 - shift '='
- V =
 - shift 'b'
- V = b
 - reduce ($V \rightarrow b$)
- V = V
 - reduce ($E \rightarrow V$)
- $V = E$
 - shift '+'
- $V = E +$
 - shift 'c'
- $V = E + \underline{c}$
 - reduce ($V \rightarrow c$)
- $V = \underline{E + V}$
 - reduce ($E \rightarrow E + V$)
- $V = E$
 - reduce ($A \rightarrow V = E$)
- A
 - accept

(handles are underlined)

shift = push, reduce = popN



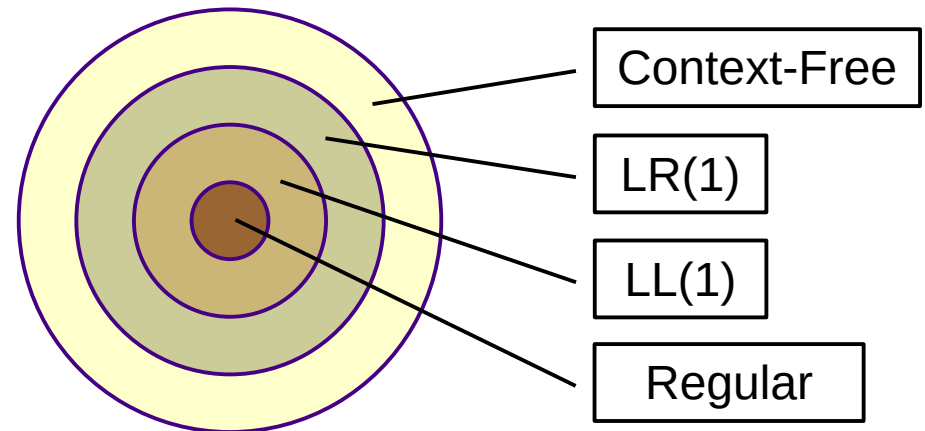
A	→	V	=	E	
E	→	E	+	V	
				V	
V	→	a		b	
				c	

"a = b + c"

LR Parsing

- **LR(1)** grammars and parsers
 - **Left-to-right** scan of the input string
 - **Rightmost** derivation
 - **1 symbol** of lookahead
 - Less restricted form of context-free grammar
 - Support for most language features
 - Efficient parsing

**Context-Free
Hierarchy**



LR Parser Variants

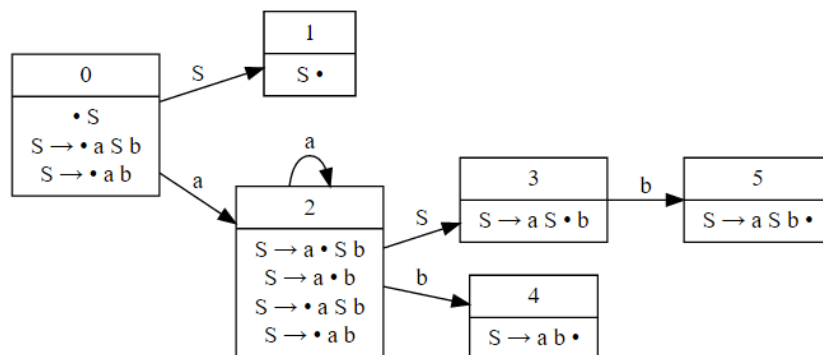
- LR(k) – multiple lookaheads (not necessary)
- LR(1) – single lookahead (*EAC covers this!*)
 - Very general and very powerful
 - Lots of item sets; tedious to construct by hand
 - Overkill for most practical languages
- LALR(1) – special case of LR(1) that merges some states
 - Less powerful, but easier to manage
- **SLR(1)** – special case of LR(1) w/o explicit lookahead
 - (*Dragon book covers this!*)
 - Uses **FOLLOW** sets to disambiguate
 - Even less powerful, but much easier to understand
 - Slightly counterintuitive: all LR(1) languages have SLR(1) grammars
 - So SLR(1) is sufficiently general for our purposes
 - Use LR(0) item sets and generate SLR(1) ACTION/GOTO tables
- LR(0) – no lookahead
 - Severely restricted; most "interesting" grammars aren't LR(0)

LR Parsing

- Creating an LR parser (**pushdown automaton**)
 - Build item sets from grammar productions
 - An **item** uses a dot (•) to represent parser status: " $A \rightarrow a \bullet S b$ "
 - Dots on the left end: "possibilities"
 - Dots in the middle: "partially-complete"
 - Dots on the right end: "complete"
 - **Item sets** represent multiple potential parser states
 - Build by taking closure
 - Similar to NFA state collections in subset construction
 - Build **ACTION** / **GOTO** tables
 - Encodes stack and transition decisions (like δ in FA)
 - **ACTION**(state, terminal) = { *shift/push*, *reduce/pop*, *accept* }
 - **GOTO**(state, non-terminal) = state

LR(0) Item Sets

- **LR(0) item sets and automaton**
 - Begin with an item representing “ $\bullet S$ ” or “ $S' \rightarrow \bullet S$ ”
 - “S” is the start symbol of the grammar
 - The latter is an **augmented** grammar
 - The Dragon book uses it; the online tool doesn’t
 - Take the **closure** to add more items if the dot lies immediately to the left of a non-terminal
 - Added items are **non-kernel** items, denoted here in blue
 - Form new sets by “moving the dot” (and take the closure)
 - Convert to finite automaton for recognizing handles by adding transitions
 - Each set becomes a state
 - “Moving the dot” = state transition + stack push



$$\begin{array}{c}
 S \rightarrow a \ S \ b \\
 | \ a \ b
 \end{array}$$

$$\begin{array}{l}
 I_0: \quad \bullet S \\
 \quad \quad S \rightarrow \bullet a \ S \ b \\
 \quad \quad S \rightarrow \bullet a \ b
 \end{array}$$

$$I_1: \quad S \bullet$$

$$\begin{array}{l}
 I_2: \quad S \rightarrow a \bullet S \ b \\
 \quad \quad S \rightarrow a \bullet b \\
 \quad \quad S \rightarrow \bullet a \ S \ b \\
 \quad \quad S \rightarrow \bullet a \ b
 \end{array}$$

$$I_3: \quad S \rightarrow a \ S \bullet b$$

$$I_4: \quad S \rightarrow a \ b \bullet$$

$$I_5: \quad S \rightarrow a \ S \ b \bullet$$

SLR(1) Tables

- Create **ACTION** and **GOTO** tables
 - For each item set i
 - If an item matches $A \rightarrow \beta \cdot c y$
 - **ACTION**(i, c) = "shift" to corresponding item set ("move the dot")
 - If an item matches $A \rightarrow \beta \cdot$
 - **ACTION**(i, x) = "reduce $A \rightarrow \beta$ " for all x in **FOLLOW**(A) ("backtrack in FA")
 - If an item matches $A \rightarrow \beta \cdot B y$
 - **GOTO**(i, B) = corresponding item set ("move the dot")
 - **ACTION**($\{S \cdot\}, \$$) = "accept"
 - Any empty **ACTION** entry = "error" (usually left blank)

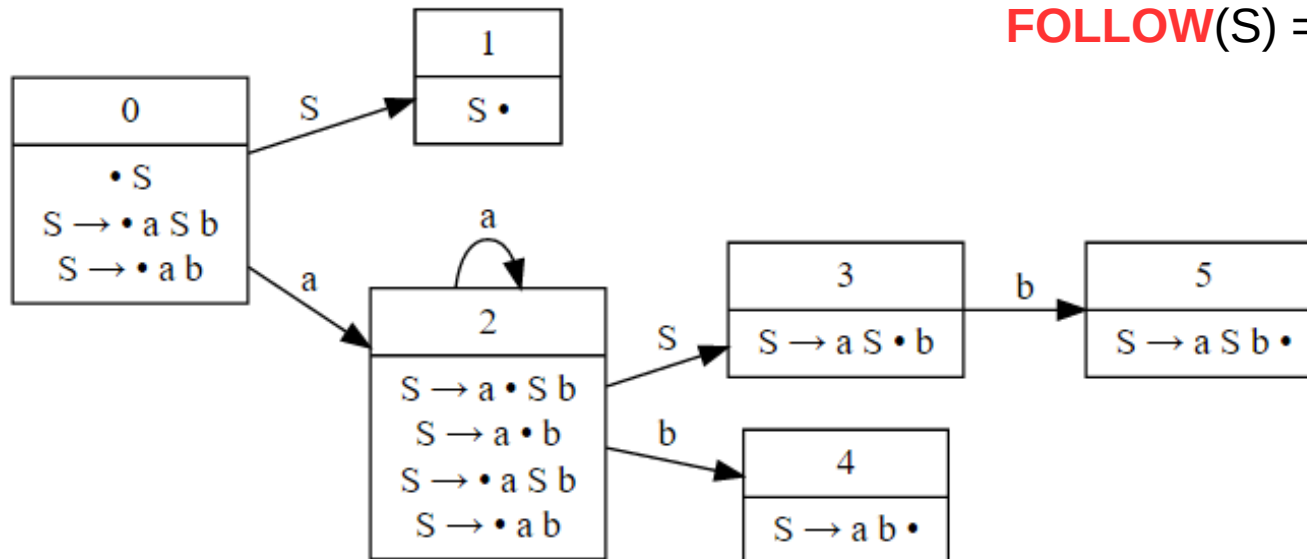
Example

$$S \rightarrow a S b$$

$$| a b$$

State	ACTION			GOTO
	a	b	$\$$	
0	shift(2)			1
1			accept	
2	shift(2)	shift(4)		3
3		shift(5)		
4		reduce($S \rightarrow a b$)	reduce($S \rightarrow a b$)	
5		reduce($S \rightarrow a S b$)	reduce($S \rightarrow a S b$)	

FOLLOW(S) = { b, \$ }



SLR(1) Parsing

- **Push** state 0 onto the stack
- Repeat until next action is accept or error:
 - Look up next action in **ACTION** table
 - Row is the current state (top of stack)
 - Column is the next input (terminal or \$)
 - If action is $\text{shift}(s)$:
 - **Push** state s onto stack
 - Consume one token from input
 - If action is $\text{reduce}(A \rightarrow \beta)$:
 - **Pop** one state for each terminal or non-terminal in β
 - Look up next state in **GOTO** table and **push** it onto the stack
 - Row is the current state (top of stack, *after* popping β)
 - Column is A (newly-reduced non-terminal)

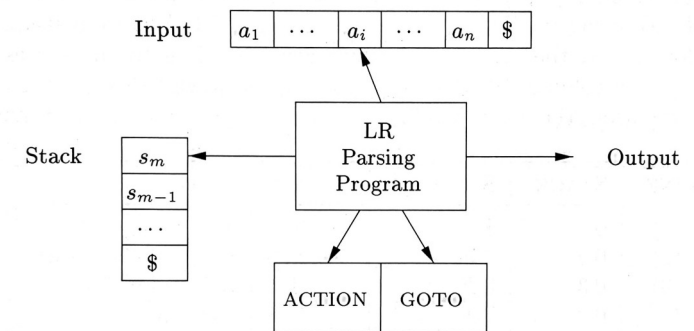


Figure 4.35: Model of an LR parser

Example

$$S \rightarrow a S b$$

$$| a b$$

Parsing for "a a b b":

Stack

\$ 0
\$ 0 2
\$ 0 2 2
\$ 0 2 2 4
\$ 0 2 3
\$ 0 2 3 5
\$ 0 1

Symbols

\$
\$ a
\$ a a
\$ a a b
\$ a S
\$ a S b
\$ S

Input

a a b b \$
a b b \$
b b \$
b \$
b \$
\$
\$

Action

shift(2)
shift(2)
shift(4)
reduce(S → a b)
shift(5)
reduce(S → a S b)
accept

ACTION

GOTO

State	a	b	\$	S
0	shift(2)			1
1			accept	
2	shift(2)	shift(4)		3
3		shift(5)		
4		reduce(S → a b)	reduce(S → a b)	
5		reduce(S → a S b)	reduce(S → a S b)	

(Dragon Book version)

Example

$$S \rightarrow a S b$$

$$| a b$$

State	ACTION			GOTO
	a	b	\$	S
0	shift(2)			1
1			accept	
2	shift(2)	shift(4)		3
3		shift(5)		
4		reduce($S \rightarrow a b$)	reduce($S \rightarrow a b$)	
5		reduce($S \rightarrow a S b$)	reduce($S \rightarrow a S b$)	

Parsing for "a a b b":

(cleaner version w/ goto)

<u>Stack</u>	<u>Input</u>	<u>Action</u>	<u>Goto</u>
<u>0</u>	<u>a</u> a b b \$	shift(2)	-
0 <u>2</u>	<u>a</u> b b \$	shift(2)	-
0 2 <u>2</u>	<u>b</u> b \$	shift(4)	-
0 2 2 <u>4</u>	<u>b</u> \$	reduce($S \rightarrow a b$)	3
0 2 <u>3</u>	<u>b</u> \$	shift(5)	-
0 2 3 <u>5</u>	<u>\$</u>	reduce($S \rightarrow a S b$)	1
0 <u>1</u>	<u>\$</u>	accept	

LR Conflicts

- Shift/reduce
 - Can be resolved by always shifting or by grammar modification
- Reduce/reduce
 - Requires grammar modification to fix

$A \rightarrow V = E$

$E \rightarrow E + V$

$E \rightarrow V$

$V \rightarrow a \mid b \mid c$

Shift/reduce conflict in LR(0)

$A \rightarrow x A x$

$A \rightarrow$

Shift/reduce conflict (all LR)

$A \rightarrow B \mid C$

$B \rightarrow x$

$C \rightarrow x$

Reduce/reduce conflict (all LR)

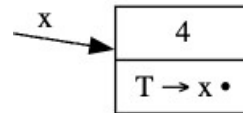
Observation: none of these languages are LL(1) either!

LR examples

- This grammar is LR(0):

$$S \rightarrow a T a$$

$$| a T b$$

$$T \rightarrow x$$


LR(0) Parsing Table

State	a	b	x	\$	S	T
0	shift(2)				1	
1	accept	accept	accept	accept		
2			shift(4)			3
3	shift(5)	shift(6)				
4	reduce($T \rightarrow x$)	reduce($T \rightarrow x$)	reduce($T \rightarrow x$)	reduce($T \rightarrow x$)		
5	reduce($S \rightarrow a T a$)	reduce($S \rightarrow a T a$)	reduce($S \rightarrow a T a$)	reduce($S \rightarrow a T a$)		
6	reduce($S \rightarrow a T b$)	reduce($S \rightarrow a T b$)	reduce($S \rightarrow a T b$)	reduce($S \rightarrow a T b$)		

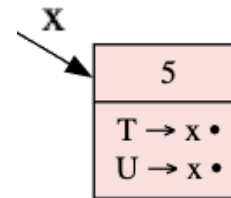
LR examples

- This grammar is SLR(1) but not LR(0):

$$S \rightarrow a T a$$

$$| a U b$$

$$T \rightarrow x$$

$$U \rightarrow x$$


$\text{FOLLOW}(T) = \{ a \}$
 $\text{FOLLOW}(U) = \{ b \}$

LR(0) Parsing Table

State	a	b	x	\$	S	T	U
0	shift(2)				1		
1	accept	accept	accept	accept			
2			shift(5)			3	4
3	shift(6)						
4		shift(7)					
5	reduce($T \rightarrow x$) reduce($U \rightarrow x$)	reduce($T \rightarrow x$) reduce($U \rightarrow x$)	reduce($T \rightarrow x$) reduce($U \rightarrow x$)	reduce($T \rightarrow x$) reduce($U \rightarrow x$)			
6	reduce($S \rightarrow a T a$)	reduce($S \rightarrow a T a$)	reduce($S \rightarrow a T a$)	reduce($S \rightarrow a T a$)			
7	reduce($S \rightarrow a U b$)	reduce($S \rightarrow a U b$)	reduce($S \rightarrow a U b$)	reduce($S \rightarrow a U b$)			

SLR(1) Parsing Table

State	a	b	x	\$	S	T	U
0	shift(2)				1		
1				accept			
2			shift(5)			3	4
3	shift(6)						
4		shift(7)					
5	reduce($T \rightarrow x$)	reduce($U \rightarrow x$)					
6				reduce($S \rightarrow a T a$)			
7				reduce($S \rightarrow a U b$)			

LR examples

- This grammar is LALR(1) but not SLR(1):

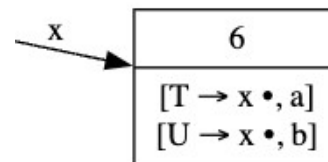
$S \rightarrow a T a$

$\quad \mid a U b$

$\quad \mid b T b$

$T \rightarrow x$

$U \rightarrow x$



$\text{FOLLOW}(T) = \{ a, b \}$
 $\text{FOLLOW}(U) = \{ b \}$

SLR(1) Parsing Table

State	a	b	x	\$	S	T	U
0	shift(2)	shift(3)			1		
1				accept			
2			shift(6)			4	5
3			shift(8)			7	
4	shift(9)						
5		shift(10)					
6	reduce($T \rightarrow x$)	reduce($T \rightarrow x$) reduce($U \rightarrow x$)					
7		shift(11)					
8	reduce($T \rightarrow x$)	reduce($T \rightarrow x$)					
9				reduce($S \rightarrow a T a$)			
10				reduce($S \rightarrow a U b$)			
11				reduce($S \rightarrow b T b$)			

LALR(1) Parsing Table

State	a	b	x	\$	S	T	U
0	shift(2)	shift(3)			1		
1				accept			
2			shift(6)			4	5
3			shift(8)			7	
4	shift(9)						
5		shift(10)					
6	reduce($T \rightarrow x$)	reduce($U \rightarrow x$)					
7		shift(11)					
8		reduce($T \rightarrow x$)					
9				reduce($S \rightarrow a T a$)			
10				reduce($S \rightarrow a U b$)			
11				reduce($S \rightarrow b T b$)			

LR examples

- This grammar is LR(1) but not LALR(1):

$S \rightarrow a T a$

| $a U b$

| $b T b$

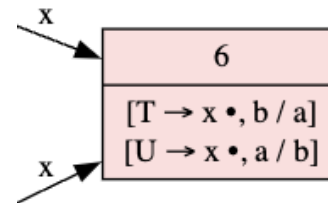
| $b U a$

$T \rightarrow x$

$U \rightarrow x$

LR(1) Parsing Table

...		
6	reduce($T \rightarrow x$)	reduce($U \rightarrow x$)
7		shift(12)
8	shift(13)	
9	reduce($U \rightarrow x$)	reduce($T \rightarrow x$)
...		



$\text{FOLLOW}(T) = \{ a, b \}$
 $\text{FOLLOW}(U) = \{ a, b \}$

LR(1) Automaton

