



## Parsing terminology

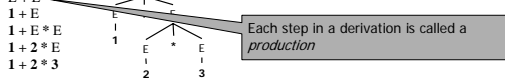
**Grammar rules (חוקי דקדוק):**  
 $E \rightarrow id$   
 $E \rightarrow num$   
 $E \rightarrow E + E$   
 $E \rightarrow E * E$   
 $E \rightarrow ( E )$

**Symbols (סימנים):**  
 terminals (tokens) + \* ( ) id num  
 non-terminals E

Convention: the non-terminal appearing in the first derivation rule is defined to be the **initial non-terminal**

**Derivation (גזירה):**  $E \Rightarrow E + E \Rightarrow 1 + E \Rightarrow 1 + E * E \Rightarrow 1 + 2 * E \Rightarrow 1 + 2 * 3$

**Parse tree (גזירה):**



Each step in a derivation is called a **production**

## Ambiguity

**Grammar rules:**  
 $E \rightarrow id$   
 $E \rightarrow num$   
 $E \rightarrow E + E$   
 $E \rightarrow E * E$   
 $E \rightarrow ( E )$

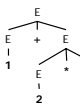
**Definition:** a grammar is *ambiguous* (ב-משמע) if there exists an input string that has two different derivations

**Leftmost derivation**

**Derivation:**

$E \Rightarrow E + E \Rightarrow 1 + E \Rightarrow 1 + E * E \Rightarrow 1 + 2 * E \Rightarrow 1 + 2 * 3$

**Parse tree:**



**Rightmost derivation**

**Derivation:**

$E \Rightarrow E * E \Rightarrow E * 3 \Rightarrow E + E * 3 \Rightarrow E + 2 * 3 \Rightarrow 1 + 2 * 3$

**Parse tree:**



## Grammar rewriting

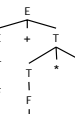
**Ambiguous grammar:**  
 $E \rightarrow id$   
 $E \rightarrow num$   
 $E \rightarrow E + E$   
 $E \rightarrow E * E$   
 $E \rightarrow ( E )$

**Unambiguous grammar:**  
 $E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow id$   
 $F \rightarrow num$   
 $F \rightarrow ( E )$

**Derivation:**

$E \Rightarrow E + T \Rightarrow 1 + T \Rightarrow 1 + T * F \Rightarrow 1 + F * F \Rightarrow 1 + 2 * F \Rightarrow 1 + 2 * 3$

**Parse tree:**



Sometimes, a grammar can be changed to an **unambiguous grammar** (while preserving the same language)

Note the difference between a language and a grammar. A grammar represents a language. A language can be represented by many grammars.

## Parsing methods

- Top-down / predictive / recursive descent without backtracking : LL(1)
  - "L" – left-to-right scan of input
  - "L" – leftmost derivation
  - "1" – predict based on one look-ahead token
    - For every non-terminal and token **predict** the next production
- Bottom-up : LR(0), SLR(1), LR(1), LALR(1)
  - "L" – left-to-right scan of input
  - "R" – rightmost derivation (in the reversed order)
    - For every potential right hand side and token decide when a production is found

## Top-down parsing

- Builds parse tree in preorder
- Also called *predictive parsing*
- LL(1) example

**Grammar:**  
 $S \rightarrow \text{if } E \text{ then } S \text{ else } S$   
 $S \rightarrow \text{begin } S L$   
 $S \rightarrow \text{print } E$   
 $L \rightarrow \text{end}$   
 $L \rightarrow ; S L$   
 $E \rightarrow \text{num}$

if 5 then print 8 else...  
**Token** : rule  
 S  
**if** : S  $\rightarrow$  if E then S else S  
 if E then S else S  
**S** : E  $\rightarrow$  num  
 if 5 then S else S  
**print** : print E  
 if 5 then print E else S  
 ...

## Problem: left recursion

- Left recursion:  $E \rightarrow E + T$ 
  - Symbol on left also first symbol on right
- Predictive parsing fails when two rules can start with same token  
 $E \rightarrow E + T$   
 $E \rightarrow T$
- Rewrite grammar using left-factoring
- Nullable, FIRST, FOLLOW sets

**Arithmetic expressions:**  
 $E \rightarrow E + T$   
 $E \rightarrow T$   
 $T \rightarrow T * F$   
 $T \rightarrow F$   
 $F \rightarrow id$   
 $F \rightarrow ( E )$

**Left-factored grammar:**  
 $E \rightarrow T E^p$   
 $E^p \rightarrow + T E^p$   
 $E^p \rightarrow \epsilon$   
 $T \rightarrow F T^p$   
 $T^p \rightarrow * F T^p$   
 $T^p \rightarrow \epsilon$   
 $F \rightarrow id$   
 $F \rightarrow ( E )$

## More on left recursion

- Non-terminal with two rules starting with same prefix

**Grammar:**  
 $S \rightarrow \text{if } E \text{ then } S \text{ else } S$   
 $S \rightarrow \text{if } E \text{ then } S$

**Left-factored grammar:**  
 $S \rightarrow \text{if } E \text{ then } S X$   
 $X \rightarrow$   
 $X \rightarrow \text{else } S$

13

## Bottom-up parsing

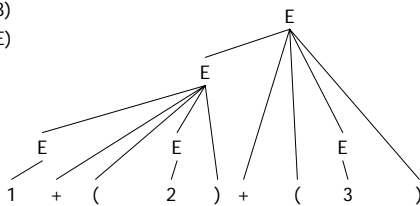
- No problem with left recursion
- Widely used in practice
- LR(0), SLR(1), LR(1), LALR(1)
  - We will focus only on the theory of LR(0)
- JavaCup implements LALR(1)

14

## Bottom-up parsing

1 + (2) + (3)  
 E + (2) + (3)  
 E + (E) + (3)  
 E + (3)  
 E + (E)  
 E

$E \rightarrow E + (E)$   
 $E \rightarrow i$



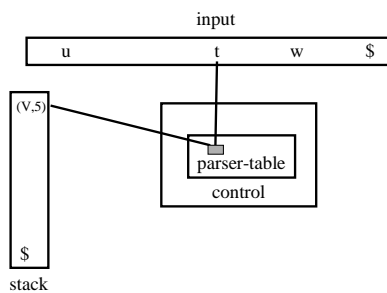
15

## Shift-reduce parsing

- Parser stack: symbols (terminal and non-terminals) + automaton states
- Parsing actions: sequence of shift and reduce operations
- Action determined by top of stack and  $k$  input tokens
- Shift: move next token to top of stack
- Reduce: for rule  $X \rightarrow A B C$  pop C, B, A then push X
- Convention: \$ stands for end of file

16

## Pushdown automaton



17

## LR parsing table

state	terminals	non-terminals
0		
1	$r k$	$g m$
⋮		
⋮	shift/reduce actions	goto part
⋮	$s n$	

Shift and move to state  $n$     Reduce by rule  $k$     Goto state  $m$

18

## Parsing table example

STATE	i	+	(	)	\$	E	T
0	s5	err	s7	err	err	1	6
1	err	s3	err	err	s2		
2	accept						
3	s5	err	s7	err	err		4
4	reduce E→E+T						
5	reduce T→i						
6	reduce E→T						
7	s5	err	s7	err	err	8	6
8	err	s3	err	err	s9		
9	reduce T→(E)						

$S \rightarrow ES$   
 $E \rightarrow T$   
 $E \rightarrow E + T$   
 $T \rightarrow i$   
 $T \rightarrow (E)$

19

## Items

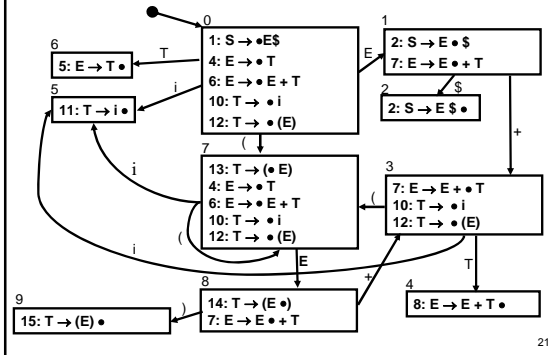
Items indicate the position inside a rule:  
 LR(0) items are of the form  $A \rightarrow \alpha \bullet t$   
 (LR(1) items are of the form  $A \rightarrow \alpha \bullet t, \beta$ )

Grammar:  
 $S \rightarrow ES$   
 $E \rightarrow T$   
 $E \rightarrow E + T$   
 $T \rightarrow i$   
 $T \rightarrow (E)$

1:  $S \rightarrow \bullet ES$   
 2:  $S \rightarrow E \bullet S$   
 3:  $S \rightarrow E S \bullet$   
 4:  $E \rightarrow \bullet T$   
 5:  $E \rightarrow T \bullet$   
 6:  $E \rightarrow \bullet E + T$   
 7:  $E \rightarrow E \bullet + T$   
 8:  $E \rightarrow E + \bullet T$   
 9:  $E \rightarrow E + T \bullet$   
 10:  $T \rightarrow \bullet i$   
 11:  $T \rightarrow i \bullet$   
 12:  $T \rightarrow \bullet (E)$   
 13:  $T \rightarrow ( \bullet E)$   
 14:  $T \rightarrow (E \bullet)$   
 15:  $T \rightarrow (E) \bullet$

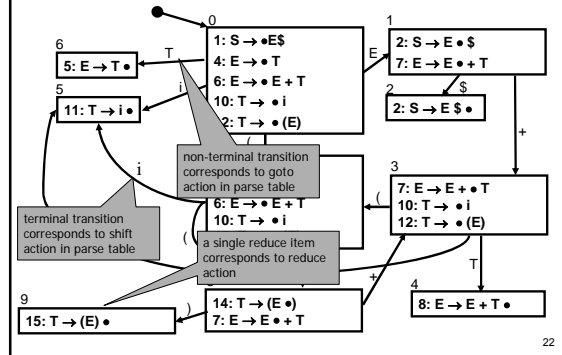
20

## Automaton states



21

## Automaton states



22

## Identifying handles

- Create a finite state automaton over grammar symbols
  - Sets of LR(0) items
- Use automaton to build parser tables
  - shift For items  $A \rightarrow \alpha \bullet t$  on token  $t$
  - reduce For items  $A \rightarrow \alpha \bullet$  on every token
- Any grammar has
  - Transition diagram
  - GOTO table
- Not every grammar has deterministic action table
- When no conflicts occur we can use a DPDA which pushes states on the stack

23

## Non-LR(0) grammars

- When conflicts occur the grammar is not LR(0)
  - Parsing table contains non-determinism
  - shift-reduce conflicts
  - reduce-reduce conflicts
  - shift-shift conflicts?
- Known cases
  - Operator precedence
  - Operator associativity
  - Dangling if-then-else
  - Unary minus
- Solutions
  - Develop equivalent non-ambiguous grammar
  - Patch parsing table to shift/reduce
  - Precedence and associativity of tokens
  - Stronger parser algorithm: SLR/LR(1)/LALR(1)

24

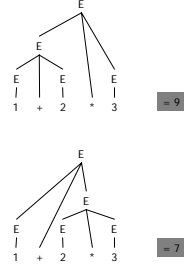
## Precedence and associativity

- Precedence
  - $E \rightarrow E+E \bullet *E$
  - $E \rightarrow E+E \bullet$
- Reduce
  - + precedes \*
- Shift
  - \* precedes +

25

## Precedence and associativity

- Precedence
  - $E \rightarrow E+E \bullet *E$
  - $E \rightarrow E+E \bullet$
- Reduce
  - + precedes \*
- Shift
  - \* precedes +



26

## Precedence and associativity

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>■ Precedence                             <ul style="list-style-type: none"> <li>■ <math>E \rightarrow E+E \bullet *E</math></li> <li>■ <math>E \rightarrow E+E \bullet</math></li> </ul> </li> <li>■ Reduce                             <ul style="list-style-type: none"> <li>+ precedes *</li> </ul> </li> <li>■ Shift                             <ul style="list-style-type: none"> <li>* precedes +</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>■ Associativity                             <ul style="list-style-type: none"> <li>■ <math>E \rightarrow E+E \bullet +E</math></li> <li>■ <math>E \rightarrow E+E \bullet</math></li> </ul> </li> <li>■ Shift                             <ul style="list-style-type: none"> <li>+ right-associative</li> </ul> </li> <li>■ Reduce                             <ul style="list-style-type: none"> <li>+ left-associative</li> </ul> </li> </ul> |
|--|--|

27

## Dangling else/if-else ambiguity

**Grammar:**  
 $S \rightarrow \text{if } E \text{ then } S \text{ else } S$   
 $S \rightarrow \text{if } E \text{ then } S$   
 $S \rightarrow \text{other}$

if a then if b then e1 else e2  
 which interpretation should we use?

- (1) if a then { if b then e1 else e2 } -- standard interpretation
- (2) if a then { if b then e1 } else e2

shift/reduce conflict

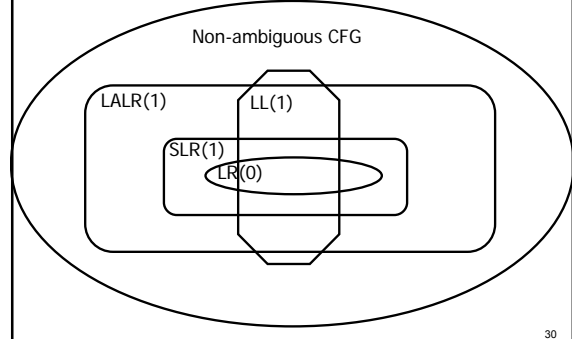
LR(1) items:	token:
$S \rightarrow \text{if } E \text{ then } S \bullet$	else
$S \rightarrow \text{if } E \text{ then } S \bullet \text{ else } S$	(any)

28

See you next week

29

## Grammar hierarchy



30