Syntactic Analysis: Bottom-up parsing

Key Ideas

- *Review*: Lexical analysis is done by the tokenizer, which constructs lexemes from the source file and categorizes them into tokens. The <token type, lexeme> pair is sent to the parser to build phrases.
- *Review*: The parser drives the tokenizer
- **Parsing**: check for grammatical correctness and determine a sentence's phrase structure
- General parsing strategies:
  - *Top-down*: from the start symbol down to the terminals
  - *Bottom-up*: from terminals up to the start symbol
- **Shift-reduce parsing**: a bottom-up parsing strategy
- The parsing strategy puts constraints on the form of the grammar

Bottom-up parsing

- Start with input string, end with start symbol
- Apply productions in reverse to the input, replacing the right-hand side (RHS) of a production with the left-hand-side (LHS) non-terminal.
  - This is called a *reduction*.
  - Loosely, a *handle* is the RHS to be replaced
- The result is a rightmost derivation, in reverse.
- Bottom-up parsers can use current stack and a lookahead symbol to choose the production to apply
  - Reads its target string from Left to right tracing a Right-most derivation (in reverse)
  - Often referred to as an LR parser

Bottom-up parsing example

Consider the grammar and input string abbcde.

1. S $\rightarrow$ a A B e
2. A $\rightarrow$ A b c
3. $\rightarrow$ b
4. B $\rightarrow$ d

Produce a rightmost derivation in reverse for abbcde using bottom-up parsing.

Production	**Sentential Form**	**Handle**
-	abbcde	A $\Leftarrow$ b
3	aAbcde	A $\Leftarrow$ A b c
2	aAde	B $\Leftarrow$ d
4	aABe	S $\Leftarrow$ a A B e
1	S	-

The problem is deciding when to reduce and which *RHS (handle)* to use

Idea

- Split the input into two parts
  - Left substring is our work area; all handles must be here
  - Right substring is input we have not yet processed

- **Shift**: move a terminal across the split
- **Reduce**: reduce a handle

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced. so *shift* 'a'

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
4	(X	↑	, a)	shift	No reducible handle, so *shift* ','

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
4	(X	↑	, a)	shift	No reducible handle, so *shift* ','
5	(X ,	↑	a)	shift	No reducible handle, so *shift* 'a'

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
4	(X	↑	, a)	shift	No reducible handle, so *shift* ','
5	(X ,	↑	a)	shift	No reducible handle, so *shift* 'a'
6	(X , a	↑	)	reduce	*reduce* 'a' to 'X' (grammar rule 3)

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
4	(X	↑	, a)	shift	No reducible handle, so *shift* ','
5	(X ,	↑	a)	shift	No reducible handle, so *shift* 'a'
6	(X , a	↑	)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
7	(X , X	↑	)	reduce	*reduce* 'X , X' to 'S' (grammar rule 2)

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
4	(X	↑	, a)	shift	No reducible handle, so *shift* ','
5	(X ,	↑	a)	shift	No reducible handle, so *shift* 'a'
6	(X , a	↑	)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
7	(X , X	↑	)	reduce	*reduce* 'X , X' to 'S' (grammar rule 2)
8	(S	↑	)	shift	No reducible handle, so *shift* ')'

Example

Grammar rules:

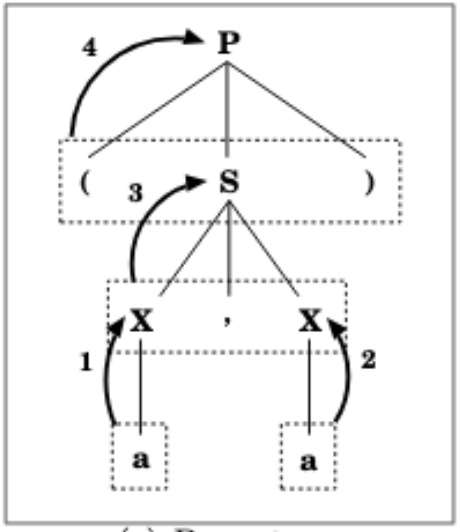
```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
4	(X	↑	, a)	shift	No reducible handle, so *shift* ','
5	(X ,	↑	a)	shift	No reducible handle, so *shift* 'a'
6	(X , a	↑	)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
7	(X , X	↑	)	reduce	*reduce* 'X , X' to 'S' (grammar rule 2)
8	(S	↑	)	shift	No reducible handle, so *shift* ')'
9	(S)	↑	ϵ	reduce	*reduce* '(S)' to 'P' (grammar rule 1)

Example

Grammar rules:

```
1 P → '(' S ')'
2 S → X ',' X
3 X → 'a'
4   → 'b'
```

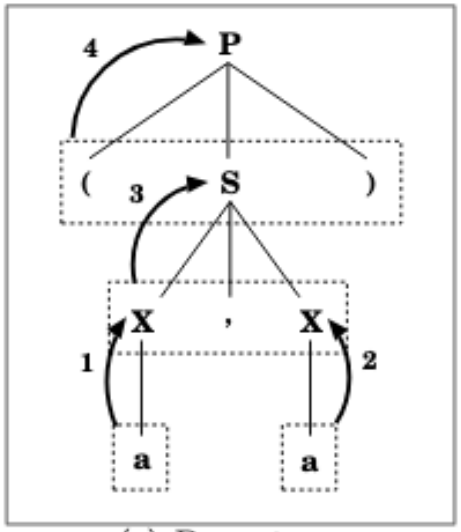


(c) Parse tree

Step	Rewritten string	↑	Action	Comment	
1	ϵ	↑	(a , a)	shift	No handle so *shift* '('
2	(	↑	a , a)	shift	'(' can't be reduced, so *shift* 'a'
3	(a	↑	, a)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
4	(X	↑	, a)	shift	No reducible handle, so *shift* ','
5	(X ,	↑	a)	shift	No reducible handle, so *shift* 'a'
6	(X , a	↑	)	reduce	*reduce* 'a' to 'X' (grammar rule 3)
7	(X , X	↑	)	reduce	*reduce* 'X , X' to 'S' (grammar rule 2)
8	(S	↑	)	shift	No reducible handle, so *shift* ')'
9	(S)	↑	ϵ	reduce	*reduce* '(S)' to 'P' (grammar rule 1)
10	P	↑	ϵ	accept	P is start symbol and input is empty

Pruning the leaves

- Consider the parse tree that is constructed
- We prune the tree by deleting the leaves corresponding to the right hand side of some production
- In the reverse of a rightmost derivation, we always prune the leftmost prunable node
- These leaves represent the handle of the immediate string obtained by concatenating the leaves of the tree



(c) Parse tree

LR(1) Grammars

- Informally, we say that a grammar G is LR(1) if we can find the sequence of handles for a reverse rightmost derivation using at most 1 token of lookahead past the end of the handle.

Properties

- Efficient shift-reduce parsers can be implemented for LR(1) grammars
- Disadvantage: hard to create by hand; LR parser generators exist

An LR(1) Grammar

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow \text{identifier}$

In-class: Get into assigned groups. Have one person open a word document and share their screen. As a group, use shift/reduce to parse the string id + id * id. Submit your document to Moodle.