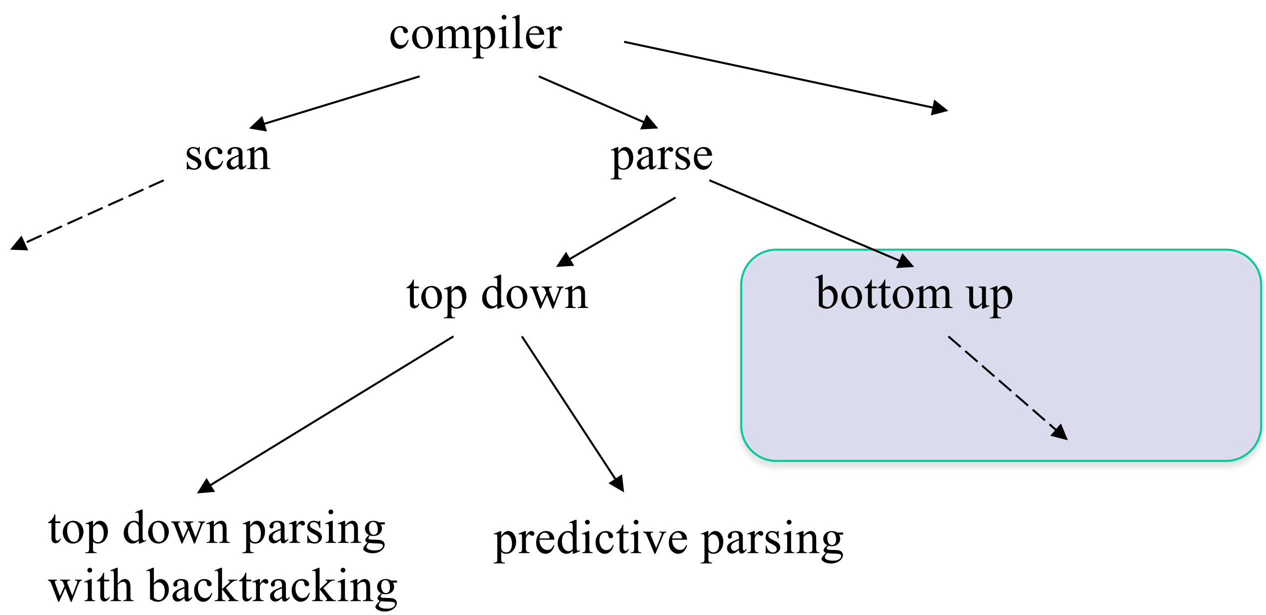
Computer Science 160
Translation of Programming Languages

Instructor: Christopher Kruegel

Introduction to Bottom-Up Parsing

Where are we in the process?



Parsing Techniques

Top-down parsers *(LL(1), recursive descent parsers)*

- Start at the root of the parse tree from the start symbol and grow toward leaves (similar to a derivation)
- Pick a production and try to match the input
- Bad "pick" $\Rightarrow$ may need to backtrack
- Some grammars are backtrack-free *(predictive parsing)*

Bottom-up parsers *(LR(1), shift-reduce parsers)*

- Start at the leaves and grow toward root
- We can think of the process as reducing the input string to the start symbol
- At each reduction step, a particular substring matching the right-side of a production is replaced by the symbol on the left-side of the production
- Bottom-up parsers handle a large class of grammars

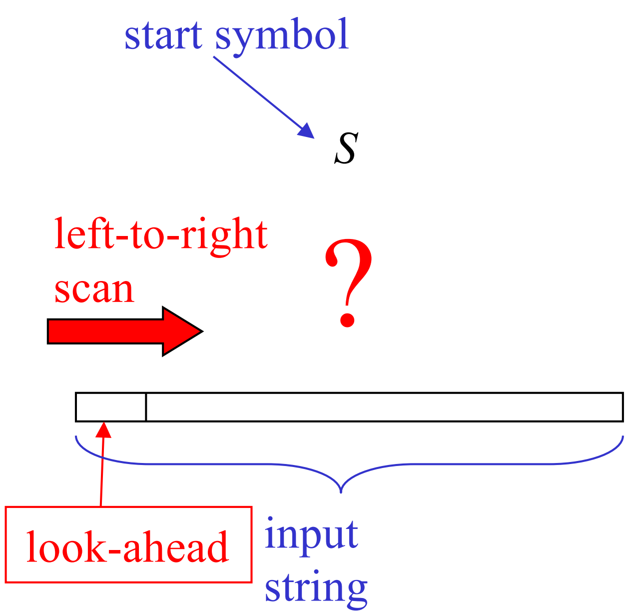
Bottom-Up Parsers

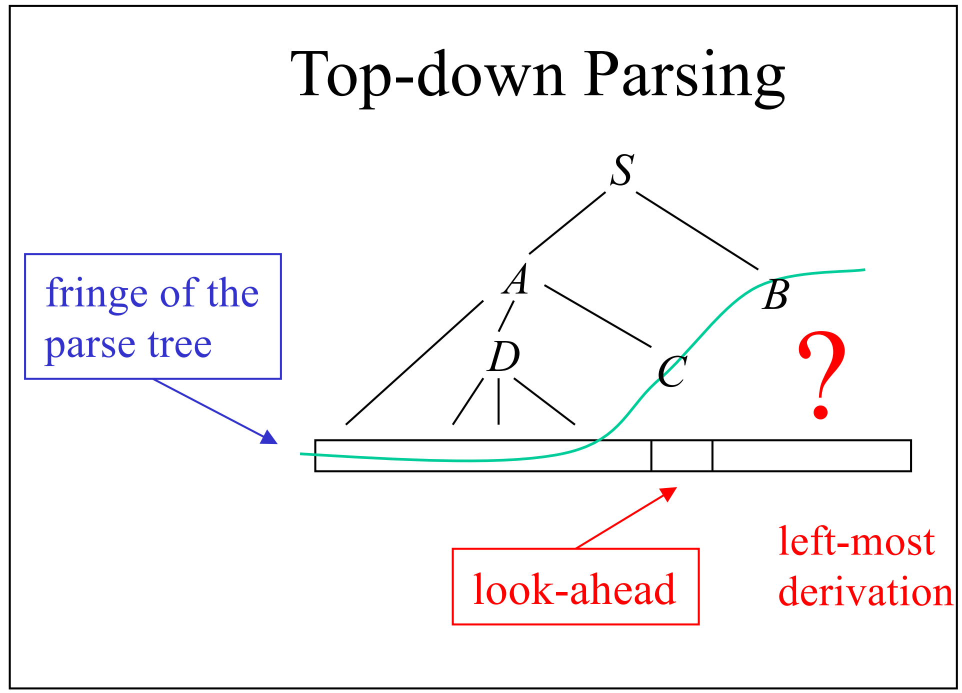
- Bottom up parsers handle a larger class of useful grammars
  Why is that?

- With both techniques, we are trying to build a tree that connects the input string to our start symbol (S) with a parse tree

- Let's look at what what each technique looks like halfway through parsing an input string

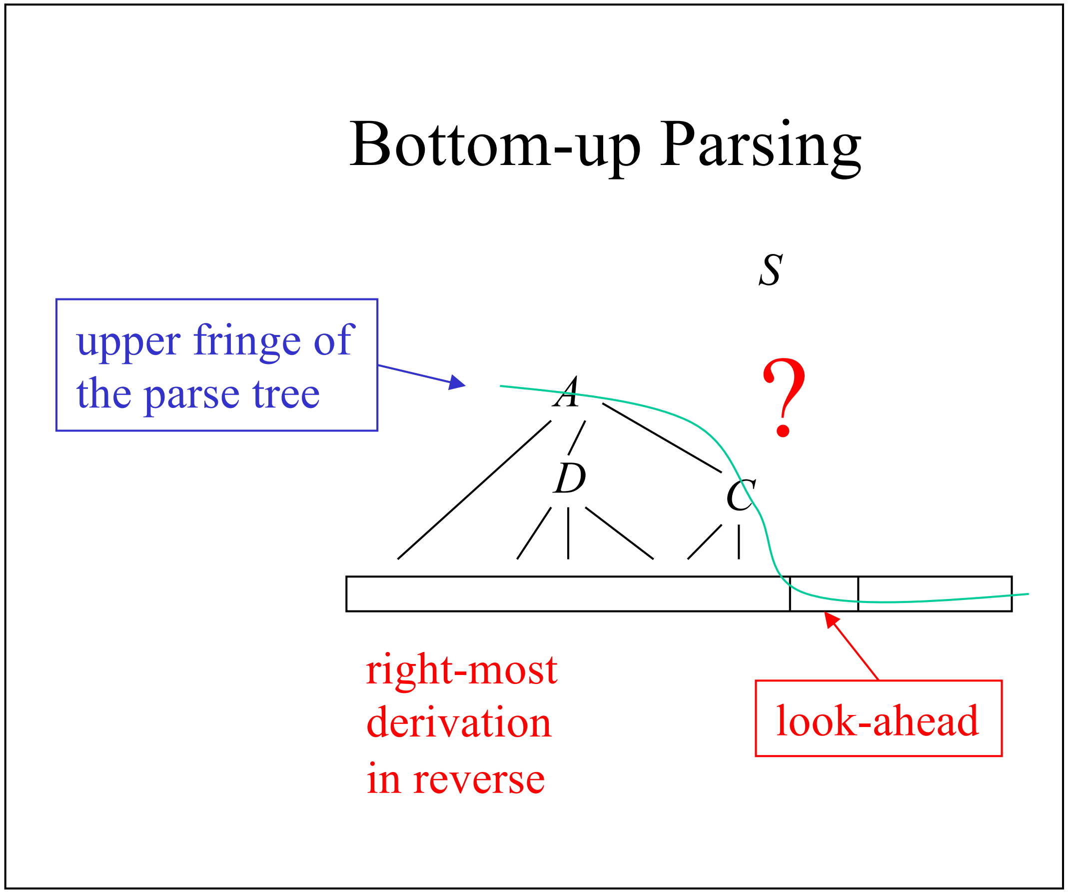
Top-Down Parsing





When we start with a predictive top down parser, the look-ahead symbol we read from our input string MUST fully specify the parse tree from S to the input symbol. In the example, we have to know that S→AB before we even see any of B

Bottom-Up Parsing



In a bottom up parser, we can delay this decision because we only need to build the tree up above the part of the input string we have examined so far.

In the graphical example on the left, you can see that even though we are at the same point in the input string, the production S→AB has not been specified yet. This delayed decision allows us to parse more grammars than predictive top-down parsing (LL).

As a nice side effect, bottom-up parsing allows us to handle left-recursive grammars without modification

LR(k) versus LL(k)

A k-look-ahead top-down predictive parser:

LL(*k*) ⇒ Parser must select the reduction (production) based on

1. The complete left context
2. The next *k* terminals

A k-look-ahead bottom-up parser:

LR(*k*) ⇒ Each reduction (production) in the parse is detectable with

1. the complete left context,
2. the reducible phrase, itself, and
3. the *k* terminal symbols to its right

Thus, LR(*k*) examines more context

From the Bottom Up

- How does a bottom-up parser work
  - The main idea of bottom-up parsing is to find the rightmost-derivation of a sentence, by running the productions backwards from sentence to the start symbol

A rightmost derivation

$$S \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n \Rightarrow w$$

- In the above, w is derived from S via the sentential forms $\gamma_0 \ldots \gamma_n$

  What we are going to do is start with w, and figure out what γ_n leads to w, and then we will replace w with γ_n. Then, we will figure out what γ_{n-1} leads to γ_n and so forth until we reach S.

Bottom-up Parsing

The point of parsing is to construct a <u>derivation</u>

A derivation consists of a series of rewrite steps

$$S \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n \Rightarrow \textit{sentence}$$

- Each γ_i is a sentential form
  - If α contains only terminal symbols, α is a sentence in $L(G)$
  - If α contains ≥ 1 non-terminals, α is just a sentential form

- To get γ_i from γ_{i-1}, expand non-terminal $\alpha \in \gamma_{i-1}$ by using a production $\alpha \rightarrow \beta$
  - Replace the occurrence of $\alpha \in \gamma_{i-1}$ with β to get γ_i
  - In a leftmost derivation, it would be the first $\alpha \in \gamma_{i-1}$

A *left-sentential form* occurs in a *<u>leftmost</u>* derivation
A *right-sentential form* occurs in a *<u>rightmost</u>* derivation

Bottom-up Parsing

A bottom-up parser builds a derivation by working from the input sentence back toward the start symbol *S*

$$S \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n \Rightarrow \textit{sentence}$$

To derive γ_{i-1} from γ_i, it matches some *rhs (right-hand side)* β against γ_i, then replace β with its corresponding *lhs (left-hand side)* α. *(assuming* $\alpha \rightarrow \beta$*)*

In terms of the parse tree, this is working from leaves to root

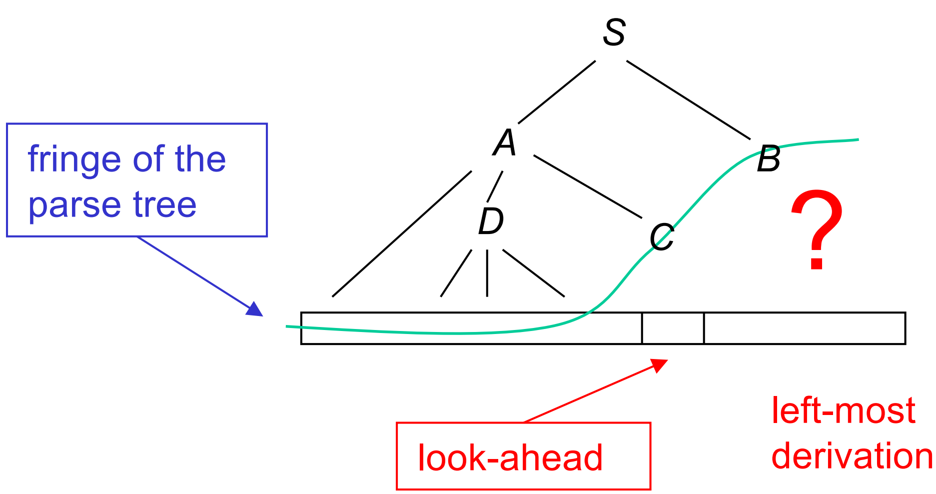
- Nodes with no parent in a partial tree form its *upper fringe*
- Since each replacement of β with α shrinks the upper fringe, we call it a *reduction*

The parse tree need not be built, it can be simulated

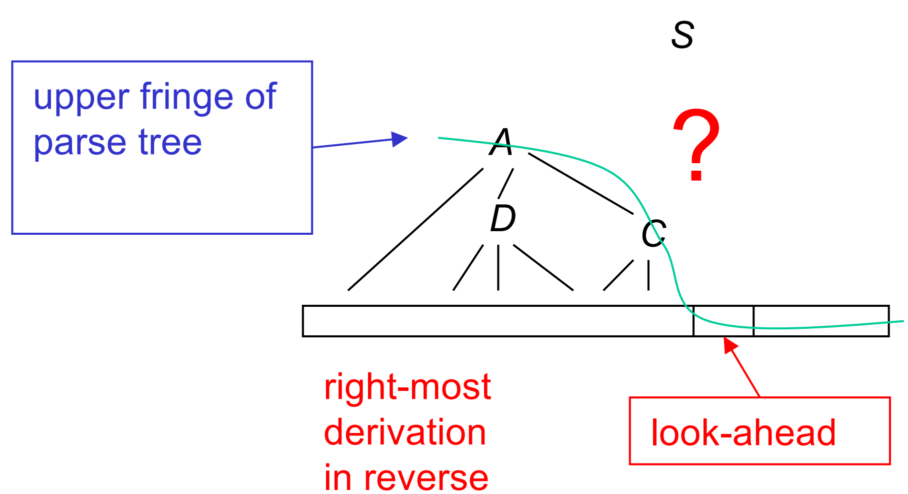
$$|\textit{parse tree nodes}| = |\textit{words}| + |\textit{reductions}|$$

Parsing Techniques

Top-down Parsing



Bottom-up Parsing



Finding Reductions

Consider the simple grammar

1	***S***	→	**a *A B* e**
2	***A***	→	***A* b c**
3		|	**b**
4	***B***	→	**d**

Sentential Form	*Next Reduction*	
	Production	*Position*
abbcde	3	2
a *A* bcde	2	4
a *A* de	4	3
a *A B* e	1	4
S	—	—

And the input string: abbcde

The trick is scanning the input and finding the next reduction

The mechanism for doing this must be efficient

Finding Reductions (Handles)

The parser must find a substring $\beta\mu\delta$ of the tree's fringe that matches some production $\alpha \rightarrow \beta\mu\delta$ that occurs as one step in the rightmost derivation. Informally, we call this substring $\beta\mu\delta$ a *handle*

Formally, **(IMPORTANT)**

- A *handle* of a right-sentential form γ is a pair $< \alpha \rightarrow \beta\mu\delta \, , \boldsymbol{k} >$ where $\alpha \rightarrow \beta\mu\delta$ is a production and *k* is the position in γ of $\beta\mu\delta$'s rightmost symbol.
- If $< \alpha \rightarrow \beta\mu\delta \, , \boldsymbol{k} >$ is a handle, then replacing $\beta\mu\delta$ at *k* with α produces the right sentential form preceding γ in the rightmost derivation.

Because γ is a right-sentential form, the substring to the right of a handle contains only terminal symbols

$\Rightarrow$ The parser doesn't need to scan past the handle (needs only a look-ahead)

Finding Reductions (Handles)

Insight

If G is unambiguous, then every right-sentential form has a unique handle.

If we can find those handles, we can build a derivation !

Sketch of Proof:

1 *G* is unambiguous $\Rightarrow$ rightmost derivation is unique

2 $\Rightarrow$ a unique production $\alpha \rightarrow \beta\mu\delta$ applied to take γ_{i-1} to γ_i

3 $\Rightarrow$ a unique position ***k*** at which $\alpha\rightarrow\beta\mu\delta$ is applied

4 $\Rightarrow$ a unique handle < $\alpha\rightarrow\beta\gamma\delta$, ***k*** >

This all follows from the definitions

Expression Example

1	*S*	→	*Expr*
2	*Expr*	→	*Expr + Term*
3		|	*Expr – Term*
4		|	*Term*
5	*Term*	→	*Term * Factor*
6		|	*Term / Factor*
7		|	*Factor*
8	*Factor*	→	num
9		|	id

The expression grammar

Sentential Form	*Handle Prod'n , Pos'n*
S	—
Expr	1,1
Expr – *Term*	3,3
Expr – *Term* * *Factor*	5,5
Expr – *Term* * <id,y>	9,5
Expr – *Factor* * <id,y>	7,3
Expr – <num,2> * <id,y>	8,3
Term – <num,2> * <id,y>	4,1
Factor – <num,2> * <id,y>	7,1
<id,x> – <num,2> * <id,y>	9,1

Handles for rightmost derivation of input:

x – 2 * y

If we start with our input string, we can work backwards to S