Bottom-Up Parsing

Quiz

What are the first and follow sets for the nonterminals and terminals of the grammar below?

0	*Goal*	→	a *A B* e
1	*A*	→	*A* b c
2		|	b
3	*B*	→	d

Recap of Top-down Parsing

- Top-down parsers build syntax tree from root to leaves
- Left-recursion causes non-termination in top-down parsers
  - Transformation to eliminate left recursion
  - Transformation to eliminate common prefixes in right recursion

Recap of Top-down Parsing (cont'd)

- FIRST, FIRST$^+$, & FOLLOW sets + LL(1) condition
  - —LL(1) uses left-to-right scan of the input, leftmost derivation of the sentence, and 1 word lookahead
  - —LL(1) condition means grammar works for predictive parsing
- Given an LL(1) grammar, we can
  - —Build a table-driven LL(1) parser
- LL(1) parser keeps lower fringe of partially complete tree on the stack

Parsing Techniques

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

- Start at root of the parse tree and grow toward leaves
- Pick a production & try to match the input
- Bad "pick" ⇒ may need to backtrack
- Some grammars are backtrack-free *(predictive parsing)*

Bottom-up parsers *(LR(1), operator precedence)*

- Start at the leaves and grow toward root
- As input consumed, encode possibilities in internal state
- Start in a state valid for legal first tokens
- Bottom-up parsers handle a large class of grammars

Bottom-up Parsin (definitions)

The point of parsing is to construct a derivation

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 γ contains only terminal symbols, γ is a sentence in *L(G)*
  - If γ contains 1 or more non-terminals, γ is a sentential form

Bottom-up Parsing (definitions)

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

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

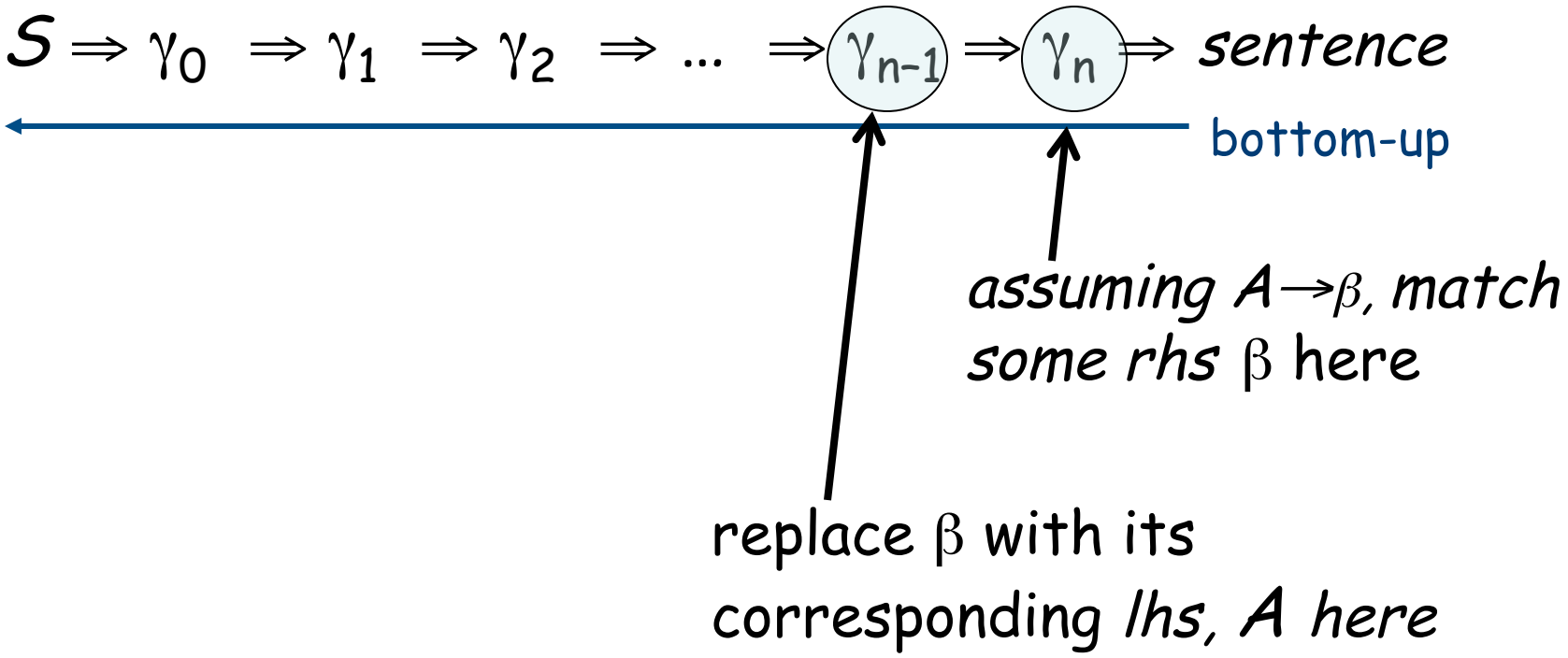
A *left-sentential form* occurs in a *leftmost* derivation

A *right-sentential form* occurs in a *rightmost* derivation

Bottom-up parsers build rightmost derivation in reverse

Bottom-up Parsing (definitions)

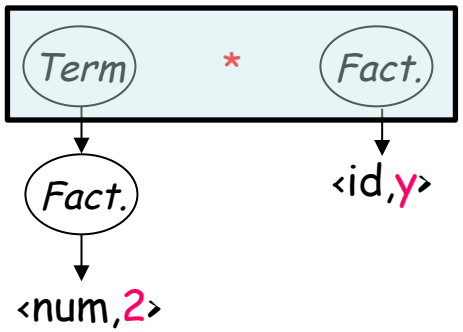
A bottom-up parser builds derivation by working from input sentence back toward the start symbol *S*



Bottom-up Parsing (definitions)

In terms of parse tree, it works from leaves to root

- Nodes with no parent in partial tree form *upper fringe*
- Each replacement of β with *A* shrinks the upper fringe, we call this a *reduction*.
- "Rightmost derivation in reverse" processes words *left to right*

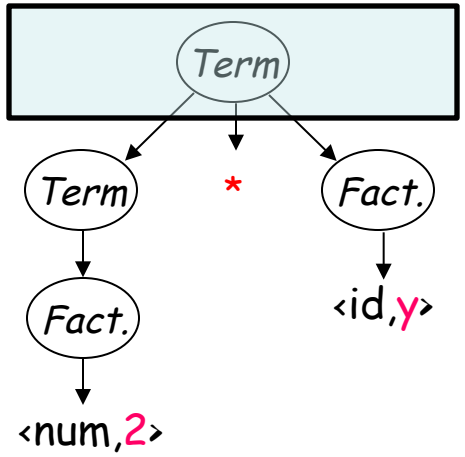


upper fringe

Bottom-up Parsing (definitions)

In terms of parse tree, it works from leaves to root

- Nodes with no parent in partial tree form *upper fringe*
- Each replacement of β with *A* shrinks the upper fringe, we call this a *reduction*.
- "Rightmost derivation in reverse" processes words *left to right*



reduction

Finding Reductions

Consider the grammar

0	*Goal*	→	a *A B* e
1	*A*	→	*A* b c
2		|	b
3	*B*	→	d

And the input string abbcde

Sentential Form	*Next Reduction* *Prod'n*	*Pos'n*
abbcde	2	2
a *A* bcde	1	4
a *A* de	3	3
a *A B* e	0	4
Goal	—	—

"Position" specifies where the right end of β occurs in the current sentential form. We call this position k.

Finding Reductions (Handles)

Parser must find substring β at parse tree's frontier that *matches some production* $A \rightarrow \beta$

($\Rightarrow \beta \rightarrow A$ is in Reverse Rightmost Derivation)

We call substring β a *handle*

Finding Reductions (Handles)

Formally,

A *handle* of a right-sentential form γ is a pair $\langle A \rightarrow \beta, k \rangle$ where

$A \rightarrow \beta \in P$ and k is the position in γ of β's rightmost symbol.

If $\langle A \rightarrow \beta, k \rangle$ is a handle, then replacing β at k with A produces the right sentential form from which γ is derived in the rightmost derivation.

Example

0	*Goal*	→	*Expr*
1	*Expr*	→	*Expr* + *Term*
2		|	*Expr* - *Term*
3		|	*Term*
4	*Term*	→	*Term* * *Factor*
5		|	*Term* / *Factor*
6		|	*Factor*
7	*Factor*	→	number
8		|	id
9		|	(*Expr*)

A simple left-recursive form of the classic expression grammar

Bottom up parsers can handle either left-recursive or right-recursive grammars.

Example

0	*Goal*	→	*Expr*
1	*Expr*	→	*Expr* + *Term*
2		|	*Expr* - *Term*
3		|	*Term*
4	*Term*	→	*Term* * *Factor*
5		|	*Term* / *Factor*
6		|	*Factor*
7	*Factor*	→	number
8		|	id
9		|	(*Expr*)

A simple left-recursive form of the classic expression grammar

Prod'n	*Sentential Form*	*Handle*
8	<id,x> - <num,2> * <id,y>	8,1
6	*Factor* - <num,2> * <id,y>	6,1
3	*Term* - <num,2> * <id,y>	3,1
7	*Expr* - <num,2> * <id,y>	7,3
6	*Expr* - *Factor* * <id,y>	6,3
8	*Expr* - *Term* * <id,y>	8,5
4	*Expr* - *Term* * *Factor*	4,5
2	*Expr* - *Term*	2,3
0	Expr	0,1
-	*Goal*	-

parse

Handles for rightmost derivation of x - 2 * y

Bottom-up Parsing (Abstract View)

A bottom-up parser repeatedly finds a handle $A \rightarrow \beta$ in current right-sentential form and replaces β with A.

To construct a rightmost derivation

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

Apply the following conceptual algorithm

for i ← n to 1 by -1

Find the handle $\langle A_i \rightarrow \beta_i , k_i \rangle$ in γ_i

Replace β_i *with* A_i *to generate* γ_{i-1}

This takes *2n* steps

of course, *n* is unknown until the derivation is built

More on Handles

Bottom-up parsers finds rightmost derivation

- Process input left to right
- Handle always appears at upper fringe of partially completed parse tree

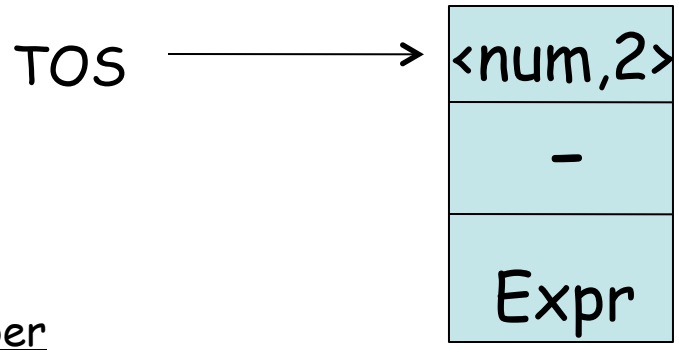
- We can keep the prefix of the upper fringe of the partially completed parse tree on a **stack**
  - The stack makes the position information irrelevant
  - Handles appear at the top of the stack

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

More on Handles

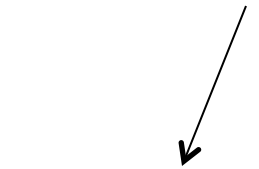
- Handles appear at the top of the stack

Prod'n	*Sentential Form*	*Handle*
8	<id,x> - <num,2> * <id,y>	8,1
6	*Factor* - <num,2> * <id,y>	6,1
3	*Term* - <num,2> * <id,y>	3,1
7	*Expr* - <num,2> * <id,y>	7,3



stack

Rest of input from scanner



* <id,y>

7 *Factor* → number