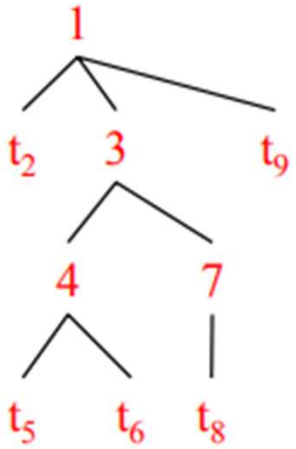
Top-down Parsing

Professor: Suman Saha

Top-down Parser

PennState

- The parse tree is constructed
  - From the top
  - From left to right
- Terminal are seen in order of appearance in the token stream:
  - $t_2\ t_5\ t_6\ t_8\ t_9$
- Top-down parsing methods
  - Recursive descent
  - Predictive parsing



Recursive Descent Parsing

- It is built from a set of mutually recursive procedures (or a non-recursive equivalent) where each such procedure implements one of the nonterminals of the grammar.

- Recursive descent parsing suffers from backtracking
  - Backtracking: It means one derivation of a production fails; the syntax analyzer restart the process using different rules of same production.
  - This technique may process the input string more than once to determine the right production

Backtracking

PennState

- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$

$$X \rightarrow oa \mid ea$$

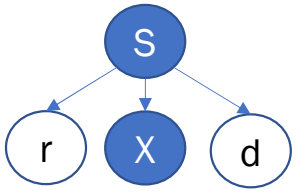
$$Z \rightarrow ai$$

- For input string $read$

Backtracking

- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$

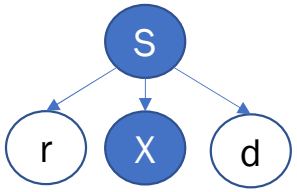


- For input string $read$

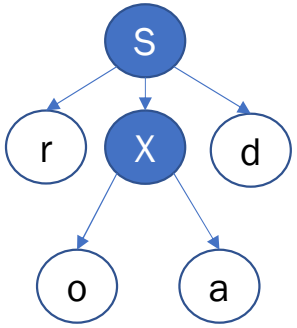
Backtracking

- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$



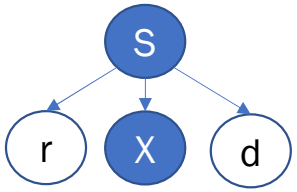
- For input string read

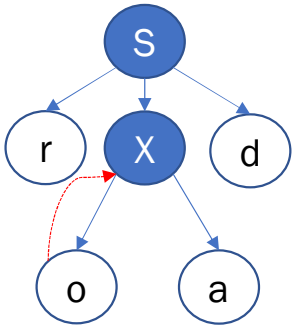


Backtracking

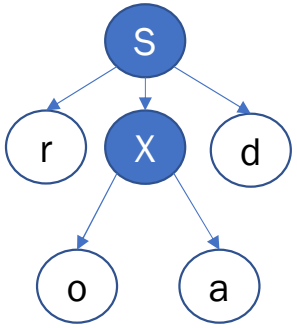
- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$





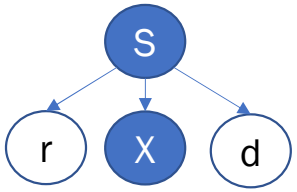
- For input string read

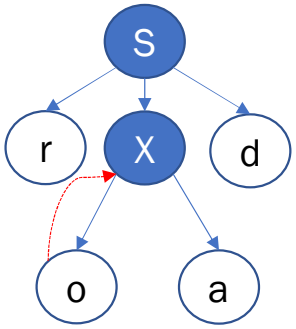


Backtracking

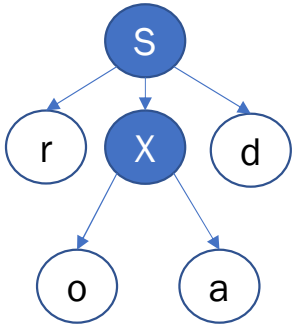
- Say, we have grammar

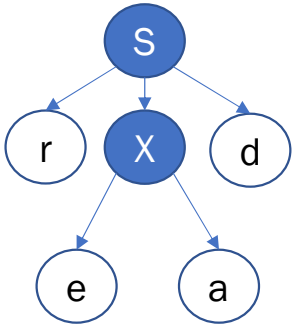
$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$





- For input string read





Recursive Descent Parsing

PennState

- Left-recursive grammar has a non-terminal S

$$S \rightarrow^{+} S\alpha \text{ for some } \alpha$$

- Recursive descent does not work in such case

Elimination of Left Recursion

PennState

- Consider the left-recursive grammar

$$S \rightarrow S\ \alpha \mid \beta$$

- S generates all string starting with a β and followed by a number of α

- Can rewrite using right-recursion

$$S \rightarrow \beta\ S'$$
$$S' \rightarrow \alpha\ S' \mid \varepsilon$$

Elimination of Left Recursion

PennState

- The grammar

$$S \rightarrow A\,\alpha \mid \delta$$
$$A \rightarrow S\,\beta$$

is also left-recursive because

$$S \rightarrow^{+} S\,\beta\,\alpha$$

- This left-recursion can also be eliminated? (Exercise)

Predictive Parsing

PennState

- It is like a recursive descent parser does not have left recursion but has the capability to predict which production is to be used to replace the input string. It does not suffer from backtracking.

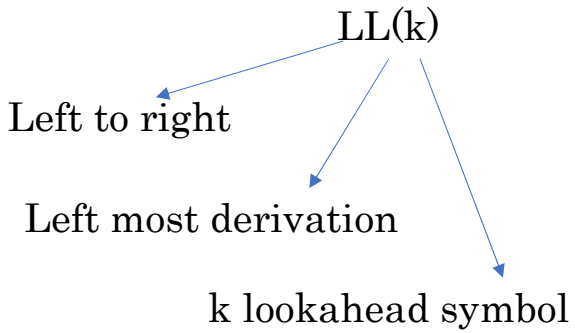
- It uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser accepts only a class of grammar known as LL(k) grammar
  - LL(k) grammar is a subset of context-free grammar

Predictive Parsing

PennState

- It is like a recursive descent parser does not have left recursion but has the capability to predict which production is to be used to replace the input string. It does not suffer from backtracking.

- It uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser accepts only a class of grammar known as LL(k) grammar
  - LL(k) grammar is a subset of context-free grammar



Predictive Parsing and Left Factoring

PennState

- Given grammar

$$\mathrm{E} \rightarrow \mathrm{T} + \mathrm{E} \mid \mathrm{T}$$
$$\mathrm{T} \rightarrow \mathrm{int} \mid \mathrm{int} * \mathrm{T} \mid (\mathrm{E})$$

- Hard to predict because
  - For T two productions start with int
  - For E it is not clear how to predict

- A grammar must be left-factored before use for predictive parsing

Predictive Parsing and Left Factoring

PennState

- Given grammar

$$E \rightarrow T + E \mid T$$
$$T \rightarrow \text{int} \mid \text{int} * T \mid (E)$$

- Factor out common prefixes of productions

$$E \rightarrow T\ X$$
$$X \rightarrow + E \mid \varepsilon$$
$$T \rightarrow (E) \mid \text{int}\ Y$$
$$Y \rightarrow * T \mid \varepsilon$$

LL(1) Parsing Table

PennState

- LL(1) means that for each non-terminal and token there is only one production
- Can be specified via 2D tables
  - One dimension for current non-terminal to expand
  - One dimension for next token
  - A table entry contains one production (We will talk about how to find table entry later)

Left-factored Grammar

$E \rightarrow T\ X$
$X \rightarrow +\ E \mid \varepsilon$
$T \rightarrow (\ E\ ) \mid \text{int}\ Y$
$Y \rightarrow *\ T \mid \varepsilon$

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

LL(1) Parsing Table

PennState

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Left-factored Grammar

$E \rightarrow T\ X$
$X \rightarrow +\ E \mid \varepsilon$
$T \rightarrow (\ E\ ) \mid \text{int}\ Y$
$Y \rightarrow *\ T \mid \varepsilon$

- Consider the $[E, \text{int}]$ entry
  - When current non-terminal is E and next input is int, use production $E \rightarrow T\ X$

LL(1) Parsing Table

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Left-factored Grammar

$E \rightarrow T\ X$
$X \rightarrow +\ E \mid \varepsilon$
$T \rightarrow (\ E\ ) \mid \text{int}\ Y$
$Y \rightarrow *\ T \mid \varepsilon$

- Consider the [Y,+] entry
  - When current non-terminal is Y and current token is +, get rid of Y

LL(1) Parsing Table

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Left-factored Grammar

$E \rightarrow T\ X$
$X \rightarrow +\ E \mid \varepsilon$
$T \rightarrow (\ E\ ) \mid \text{int}\ Y$
$Y \rightarrow *\ T \mid \varepsilon$

- Blank entries indicate error situations
  - Consider the [E,*] entry
  - There is no way to derive a string starting with * from non-terminal E

LL(1) Parsing Table

PennState

- We use a stack to keep track of pending non-terminals
- We reject when we encounter an error state
- We accept when we encounter end-of-input

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Stack	Input	Action
E $	int * int $	T X
T X $	int * int $	int Y
int Y X $	int * int $	terminal
Y X $	* int $	* T
* T X $	* int $	terminal
T X $	int $	int Y
int Y X $	int $	terminal
Y X $	$	ε
X $	$	ε
$	$	ACCEPT

FIRST() and FOLLOW()

PennState

- FIRST(X) for a grammar symbol X is the set of terminals that begin the strings derivable from X

$$\text{First}(X) = \{ t \mid X \rightarrow^* t\alpha \} \cup \{\varepsilon \mid X \rightarrow^* \varepsilon\}$$

- First of terminal is terminal itself

  - FIRST(E) = { (, int}
  - FIRST(X) = {+, ε }
  - FIRST(T) = { (, int}
  - FIRST(Y) = {*, ε }

Left-factored Grammar

E → T X
X → + E | ε
T → (E) | int Y
Y → * T | ε

FIRST() and FOLLOW()

- Follow(X) to be the set of terminals that can appear immediately to the right of Non-Terminal X in some sentential form.

- Rules:
  - If S is the start symbol then $\$ \in \text{Follow}(S)$
  - If $X \rightarrow A\ B$ then $\text{First}(B) \subseteq \text{Follow}(A)$ and $\text{Follow}(X) \subseteq \text{Follow}(B)$
  - Also, if $B \rightarrow^* \varepsilon$ then $\text{Follow}(X) \subseteq \text{Follow}(A)$

  - $\text{FOLLOW}(E) = \{\ ),\ \$\}$
  - $\text{FOLLOW}(X) = \{\ ),\ \$\ \}$
  - $\text{FOLLOW}(T) = \{\ +,\ ),\ \$\}$
  - $\text{FOLLOW}(Y) = \{+,\ ),\ \$\ \}$

Left-factored Grammar

$E \rightarrow T\ X$
$X \rightarrow +\ E \mid \varepsilon$
$T \rightarrow (\ E\ ) \mid \text{int}\ Y$
$Y \rightarrow *\ T \mid \varepsilon$

$\text{FIRST}(E) = \{\ (,\ \text{int}\}$
$\text{FIRST}(X) = \{+,\ \varepsilon\ \}$
$\text{FIRST}(T) = \{\ (,\ \text{int}\}$
$\text{FIRST}(Y) = \{*,\ \varepsilon\ \}$

Constructing LL(1) Parsing Table

- Construct a parsing table T for CFG G
- For each production $A \rightarrow \alpha$ in G do:
  - For each terminal $t \in First(\alpha)$ do
    - $T[A, t] = \alpha$
  - If $\varepsilon \in First(\alpha)$, for each $t \in Follow(A)$ do
    - $T[A, t] = \alpha$
  - If $\varepsilon \in First(\alpha)$ and $\$ \in Follow(A)$ do
    - $T[A, \$] = \alpha$

Left-factored Grammar

$E \rightarrow T\ X$
$X \rightarrow +\ E \mid \varepsilon$
$T \rightarrow (\ E\ ) \mid int\ Y$
$Y \rightarrow *\ T \mid \varepsilon$

FOLLOW(E) = {), $}
FOLLOW(X) = {), $ }
FOLLOW(T) = { +,), $}
FOLLOW(Y) = {+,), $ }

FIRST(E) = { (, int}
FIRST(X) = {+, ε }
FIRST(T) = { (, int}
FIRST(Y) = {*, ε }

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Reading and Exercises

PennState

Reading

- Chapter: 2.3.1, 2.3.2, and 2.3.3 (Michael Scott Book)

Exercises

- Exercises: 2.11 - 2.26 (Michael Scott Book)