Lecture 20 - Nov. 24

Syntactic Analysis

Bottom-Up Parsing: shift vs. reduce
Exercise: LL(1) Parser

Bottom-Up Parsing: Algorithm

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ``accept'' then
      succeed()
    elseif act = ``reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word + state */
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ``shift to s'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

	Goal → List
2	List → List Pair
3	| Pair
4	Pair → (Pair)
5	| ( )

State	Action Table			Goto Table	
	eof	(	)	List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Bottom-Up Parsing: Discovering Rightmost Derivations (1)

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ''accept'' then
      succeed()
    elseif act = ''reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word +
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ''shift to s_i'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

1	$Goal \rightarrow List$
2	$List \rightarrow List\ Pair$
3	$\mid Pair$
4	$Pair \rightarrow (\ Pair\ )$
5	$\mid (\ )$

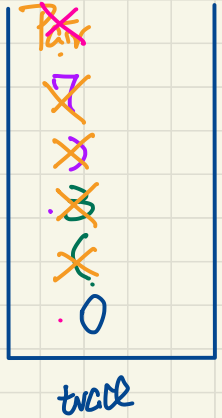
handle

State	Action Table: eof	(	)	Goto Table: List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Parse: ( )

word: "(" "(" eof

state: 0 3 3 6 8 7 1



Bottom-Up Parsing: Discovering Rightmost Derivations (2)

Parse: (( )) ( )

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success  or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ``accept'' then
      succeed()
    elseif act = ``reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word +
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ``shift to s_i'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

1	*Goal* → *List*
2	*List* → *List Pair*
3	| *Pair*
4	*Pair* → (*Pair*)
5	| ( )

	Action Table			Goto Table	
State	eof	(	)	List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Bottom-Up Parsing: Discovering Rightmost Derivations (3)

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success  or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ''accept'' then
      succeed()
    elseif act = ''reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word +
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ''shift to s_i'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

Parse: ( ))

1	*Goal* → *List*
2	*List* → *List Pair*
3	| *Pair*
4	*Pair* → (*Pair*)
5	| ( )

	Action Table			Goto Table	
State	eof	(	)	List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Exercise: LL(1) Parser

Consider the following grammar:

L	$\rightarrow$	R a	R	$\rightarrow$	aba	Q	$\rightarrow$	bbc
	|	Q ba		|	caba		|	bc
				|	R bc			

Q. Is it suitable for a ***top-down predictive*** parser?

- If so, show that it satisfies the *LL(1)* condition.
- If not, identify the problem(s) and correct it (them). Also show that the revised grammar satisfies the *LL(1)* condition.

- Given an arbitrary CFG as input to a ***top-down parser*** :
  - **Q.** How do we avoid a ***non-terminating*** parsing process?
    A. Convert left-recursions to right-recursion.
  - **Q.** How do we minimize the need of ***backtracking***?
    A. left-factoring & one-symbol lookahead using **START**
- ***Not*** every context-free language has a corresponding ***backtrack-free*** context-free grammar.

  Given a CFL l, the following is ***undecidable***:

$$\boxed{\exists cfg \mid L(cfg) = l \wedge isBacktrackFree(cfg)}$$

- Given a CFG $g = (V,\ \Sigma,\ R,\ S)$, whether or not g is ***backtrack-free*** is ***decidable***:

  For each $A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n \in R$:

$$\boxed{\forall i, j : 1 \le i, j \le n \wedge i \neq j \bullet \textbf{START}(\gamma_i) \cap \textbf{START}(\gamma_j) = \varnothing}$$

$$
\begin{array}{lcl}
L & \to & R\ a \\
 & | & Q\ ba
\end{array}
\qquad
\begin{array}{lcl}
R & \to & \cdot\ aba \\
 & | & \cdot\ caba \\
 & | & R\ bc
\end{array}
\qquad
\begin{array}{lcl}
Q & \to & bbc \\
 & | & bc
\end{array}
$$

direct left recursion

For each $A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n \in R$:

$$\forall i, j : 1 \le i, j \le n \wedge i \neq j \bullet \textsc{Start}(\gamma_i) \cap \textsc{Start}(\gamma_j) = \varnothing$$

aba
caba bc bc bc

Fix: Remove left Recursion

$$
\begin{array}{lcl}
R & \to & abaR' \\
 & | & cabaR'
\end{array}
$$

$$
\begin{array}{lcl}
R' & \to & bcR' \\
 & | & \varepsilon
\end{array}
$$

$$
\begin{array}{lcl}
L & \to & Ra \\
 & | & Qba
\end{array}
$$

$$
\begin{array}{lcl}
R & \to & abaR' \\
 & | & cabaR'
\end{array}
$$

$$
\begin{array}{lcl}
R' & \to & bcR' \\
 & | & \varepsilon
\end{array}
$$

$$
\begin{array}{lcl}
Q & \to & Qbbc \\
 & | & Qbc
\end{array}
$$

problematic

$L \to Ra$
$\quad \mid Qba$
$R \to abaR'$
$\quad \mid abaR'$
$R' \to bcR'$
$\quad \mid \varepsilon$

$Q \to bbc$
$\quad \mid bc$

left factoring →

$Q \to bQ'$
$Q' \to bc$
$\quad \mid c$

① left recursion

② common prefix

③

For each $A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n \in R$:

$$\forall i, j : 1 \le i, j \le n \wedge i \neq j \bullet \textbf{START}(\gamma_i) \cap \textbf{START}(\gamma_j) = \varnothing$$

$L \to Ra$
$\quad | \; Qba$

$R \to \dot{c}baR'$
$\quad | \; \dot{c}abaR'$

$R' \to bcR'$
$\quad | \; \varepsilon$

$Q \to bQ'$

$Q' \to bc$
$\quad | \; c$

Non-Terminal	Alternative	START Set	Intersection
Q'	bc	$\{b\}$	$\emptyset$
	c	$\{c\}$	
R	$\dot{c}baR'$	$\{a\}$	$\emptyset$
	$\dot{c}abaR'$	$\{c\}$	
L			
R'			
Q			

For each $A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n \in R$:

$\forall i, j : 1 \le i, j \le n \wedge i \neq j \bullet \textbf{START}(\gamma_i) \cap \textbf{START}(\gamma_j) = \varnothing$

F → true

trivially.