Chapter 5

Top-Down Parsing

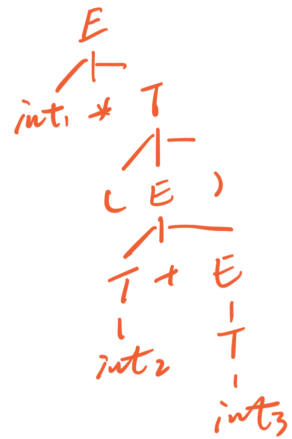
Recursive Descent Parsing

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid (E)$

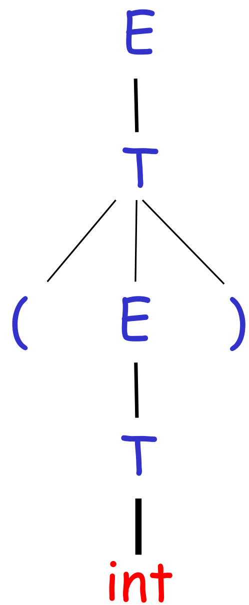
直短：深度为2
句柄：最左边的

$int_1 * (int_2 + int_3)$



短：int_2 , int_3
$int_2 + int_3$
$(int_2 + int_3)$
$int_1 * (int_2 + int_3)$

直短：int_2 , int_3
句柄：int_2



End of input, accept.

(int_5)

A (Limited) Recursive Descent Parser (3)

- For production $E \rightarrow T$

  ```
  bool E1() { return T(); }
  ```

- For production $E \rightarrow T + E$

  ```
  bool E2() { return T() && term(PLUS) && E(); }
  ```

- For all productions of E (with backtracking)

  ```
  bool E() {
    TOKEN *save = next;
    return    (next = save, E1())
           || (next = save,  E2());   }
  ```

A (Limited) Recursive Descent Parser (4)

- Functions for non-terminal T
- $T \rightarrow$ int | int * T | (E)

```
bool T1() { return term(INT); }
bool T2() { return term(INT) && term(TIMES) && T(); }
bool T3() { return term(OPEN) && E() && term(CLOSE); }
bool T() {
      TOKEN *save = next;
      return   (next = save, T1())
            || (next = save,  T2())
            || (next = save,  T3()); }
```

Practice

- A Recursive Descent Parser for Grammar S
- $S \rightarrow (\ )\ |\ (\ S\ )$

```
bool S1() {return term(OPEN) && term(CLOSE); }
bool S2() { return term(OPEN) && S() && term(CLOSE); }
bool S() {
        TOKEN *save = next;
        return    (next = save, S1())
                  || (next = save,  S2()); }
```

Example

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid (E)$

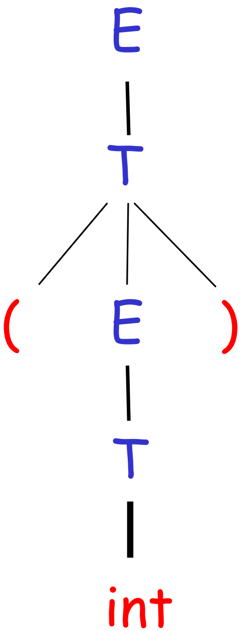
(int)

```
bool term(TOKEN tok) { return *next++ == tok; }

bool E1() { return T(); }
bool E2() { return T() && term(PLUS) && E(); }

bool E()  {TOKEN *save = next; return     (next = save, E1())
                                       || (next = save,  E2());   }
bool T1() { return term(INT); }
bool T2() { return term(INT) && term(TIMES) && T(); }
bool T3() { return term(OPEN) && E() && term(CLOSE); }

bool T() { TOKEN *save = next;  return     (next = save, T1())
                                        || (next = save,  T2())
                                        || (next = save, T3()); }
```



What happened?

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid (E)$

Input1: int

Input2: int*int

```
bool term(TOKEN tok) { return *next++ == tok; }

bool E1() { return T(); }
bool E2() { return T() && term(PLUS) && E(); }

bool E()  {TOKEN *save = next; return     (next = save, E1())
                                       || (next = save,  E2());   }
bool T1() { return term(INT); }
bool T2() { return term(INT) && term(TIMES) && T(); }
bool T3() { return term(OPEN) && E() && term(CLOSE); }

bool T() { TOKEN *save = next;  return    (next = save, T1())
                                       || (next = save,  T2())
                                       || (next = save, T3()); }
```

Elimination of Left Recursion

- Consider the left-recursive grammar

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

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

- Can rewrite using right-recursion

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

G[S]:

S->a|;|(T)

T->T,S|S

More Elimination of Left-Recursion

- In general

$$S \to S\ \alpha_1 \mid \ldots \mid S\ \alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

- All strings derived from S start with one of $\beta_1,\ldots,\beta_m$ and continue with several instances of $\alpha_1,\ldots,\alpha_n$
- Rewrite as

$$S \to \beta_1\, S' \mid \ldots \mid \beta_m\, S'$$
$$S' \to \alpha_1\, S' \mid \ldots \mid \alpha_n\, S' \mid \varepsilon$$

General Left Recursion

- The grammar

$$S \to A\ \alpha \mid \delta$$
$$A \to S\ \beta$$

  is also left-recursive because

$$S \to^{+} S\ \beta\ \alpha$$

- This left-recursion can also be eliminated

- See Dragon Book for general algorithm
  - Section 4.3

Practice

G[S]:

S->a|;|(T)

T->T,S|S

G'[S]:

S->a|;|(T)

T->ST'

T'->,ST'|ε

Predictive Parsing and Left Factoring

- Recall the grammar

$$E \rightarrow T + E \mid T$$

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

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

- We need to left-factor the grammar

Left-Factoring Example

- Recall the grammar

$$E \rightarrow T + E \mid T$$
$$T \rightarrow int \mid int * T \mid (E)$$

- Factor out common prefixes of productions

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

Practice
Left-recursion and Left-factoring Elimination

G[M]:
M->MaH|H
H->b(M)|(M)|b

G'[M]:
M->HM'
M'->aHM'|ε
H->bH'|(M)
H'->(M)|ε

LL(1) Parsing Table Example

- Left-factored grammar

$E \rightarrow T\ X$ $\qquad$ $X \rightarrow +\ E \mid \varepsilon$

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

- The LL(1) parsing table:

next input token

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

leftmost non-terminal

rhs of production to use

$E \to T\,X$ $\quad X \to +\,E \mid \varepsilon$
$T \to (\,E\,) \mid int\ Y$ $\quad Y \to *\ T \mid \varepsilon$

LL(1) Parsing Table Example

- Consider the [E, int] entry
  - "When current non-terminal is E and next input is int, use production $E \to T\,X$"
  - This can generate an int in the first position

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

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

LL(1) Parsing Tables Example

- Consider the [Y,+] entry
  - "When current non-terminal is Y and current token is +, get rid of Y"
  - Y can be followed by + only if $Y \rightarrow \varepsilon$

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

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

LL(1) Parsing Tables. Errors

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

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

LL(1) Parsing Example

$E \to T\,X \qquad T \to \text{int}\,Y \mid (\,E\,)$

$X \to +\,E \mid \varepsilon \qquad Y \to *\,T \mid \varepsilon$

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 Sets: Example 1

- Recall the grammar

$E \to T\,X$ $X \to +\,E \mid \varepsilon$

$T \to (\,E\,) \mid \text{int}\ Y$ $Y \to *\ T \mid \varepsilon$

- First sets

First(( ) = { (}

First( )) = {) }

First(int) = { int }

First(+) = { + }

First(*) = { * }

First(E) $\supseteq$ First(T)= First(T)

First(T) = { (,int }

First(X) = {+, ε}

First(Y) = {*, ε}

First Sets: Example2

G[E]: (1) E –> TE' (2) E' –> +TE' (3) E' –> ε
(4) T –> FT' (5) T' –> *FT' (6) T' –> ε
(7) F –> (E) (8) F –>i

- FIRST SETS:

N FIRST(E)=FIRST(T)=FIRST(F)= { (,i }

Y FIRST(E')= { +,ε }

N FIRST(T) =FIRST(F)= { (,i }

Y FIRST(T')= { *,ε }

N FIRST(F)= { (,i }

Computing Follow Sets

- Definition:

  $$\text{Follow}(X) = \{ t \mid S \Rightarrow^* \beta X t \delta \}$$

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

Computing Follow Sets (Cont.)

Algorithm sketch:

1. $\$ \in$ Follow(S)
2. For each production $A \rightarrow \alpha X \beta$
   - First(β) - $\{\varepsilon\} \subseteq$ Follow(X)
3. For each production $A \rightarrow \alpha X \beta$ where $\varepsilon \in$ First(β)
   - Follow(A) $\subseteq$ Follow(X)

Follow Sets: Example1

- Recall the grammar

$E \rightarrow T\ X$

$T \rightarrow (E)\ |\ \text{int}\ Y$

$X \rightarrow +\ E\ |\ \varepsilon$

$Y \rightarrow *\ T\ |\ \varepsilon$

Follow(E) ="\$"∪ ")"∪Follow(X)= {), \$}

Follow(X) = Follow(E)= {), \$ }

Follow(T) ={ First(X)-ε} ∪ Follow(E) ∪ Follow(Y)= {+,) , \$}

Follow(Y) = Follow(T) = {+,) , \$}

Follow Sets: Example2

G[E]: (1) E –> TE' (2) E' –> +TE' (3) E' –> ε
(4) T –> FT' (5) T' –> *FT' (6) T' –> ε
(7) F –> (E) (8) F –> a

- FOLLOW SETS:

FOLLOW(E)={),$}
FOLLOW(E')= FOLLOW(E)={), $}
FOLLOW(T)={FIRST(E')- ε } ∪ FOLLOW(E)= {+,), $}
FOLLOW(T')= FOLLOW(T)={+,), $}
FOLLOW(F)={FIRST(T')- ε } ∪ FOLLOW(T)= {*,+,),$}

Constructing LL(1) Parsing Tables

$E \rightarrow T\,X$ $\quad$ $X \rightarrow +\,E \mid \varepsilon$

$T \rightarrow (\,E\,) \mid \text{int}\;Y$ $\quad$ $Y \rightarrow *\,T \mid \varepsilon$

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

Follow(X) =Follow(E) = { $,) }

Follow(Y) =Follow(T) =First(X)-ε $\cup$ Follow(E) = { +, $,) }

Constructing LL(1) Parsing Tables

$S \rightarrow Sa \mid b$

First(S) = { b }
Follow(S) = { \$, a }

	a	b	\$
S		b Sa	

multiply defined !

3. 已知文法 G[S]：
$S \rightarrow MH \mid a$
$H \rightarrow LSo \mid \varepsilon$
$K \rightarrow dML \mid \varepsilon$
$L \rightarrow eHf$
$M \rightarrow K \mid bLM$
判断 G 是否是 LL(1)文法，如果是，构造 LL(1)分析表。

First　　　Follow

S　First(M) ∪ First(H) ∪ {a} = {a, b, d, e, ε}　{o, #}
H　First(L) ∪ {ε} = {e, ε}　{o, f, #}
K　{d, ε}　{e, o, #}
L　{e}　{a, b, d, e, o, #}
M　First(K) ∪ {b} = {b, d, ε}　{d, e, o, #}

	a	b	d	e	o	f	#
S	a	MH	MH	MH			MH
H				LSo	ε	ε	ε
K			dML	ε	ε		ε
L				eHf			
M		bLM	ε	ε	ε		ε

Constructing LL(1) Parsing Tables:Example2

G[E]:
(1) E –> TE’
(2) E’ –> +TE’
(3) E’ –> ε
(4) T –> FT’
(5) T’ –> *FT’
(6) T’ –> ε
(7) F –> (E)
(8) F –> a

FIRST & FOLLOW Sets

G[E]: (1) E –> TE' (2) E' –> +TE' (3) E' –> ε
(4) T –> FT' (5) T' –> *FT' (6) T' –> ε
(7) F –> (E) (8) F –>a

- FIRST SETS:

FIRST(E)= { (,a }
FIRST(E')= { +,ε }
FIRST(T) = { (,a }
FIRST(T')= { *,ε }
FIRST(F)= { (,a }

- FOLLOW SETS:

FOLLOW(E)={),$}
FOLLOW(E')= {), $}
FOLLOW(T)={+,), $}
FOLLOW(T')= {+,), $}
FOLLOW(F)={*,+,),$}

G[E]: (1) E –> TE' (2) E' –> +TE' (3) E' –> ε
(4) T –> FT' (5) T' –> *FT' (6) T' –> ε
(7) F –> (E) (8) F –> a

LL(1) Parsing Tables

	a	+	*	(	)	$
E	**TE'**			**TE'**		
E′		**+TE'**			**ε**	**ε**
T	**FT'**			**FT'**		
T′		**ε**	***FT'**		**ε**	**ε**
F	**a**			**(E)**		

PARSING PROCEDURE -- Input: $ a+a $

STACK	TOP	CURRENT	M[X,b]
1 $E	E	a	E –> TE’
2 $E’T	T	a	T –> FT’
3 $E’T’F	F	a	F –> a
4 $E’T’a	a	a	MATCH
5 $E’T’	T’	+	T’ –> ε
6 $E’	E’	+	E’ –> +TE’
7 $E’T+	+	+	MATCH
8 $E’T	T	a	T –> FT’
9 $E’T’F	F	a	F –> a
10 $E’T’a	a	a	MATCH
11 $E’T’	T’	$	T’ –> ε
12 $E’	E’	$	E ’ –> ε
13 $	$	$	ACCEPT