Principles of Compiler Construction

Prof. Wen-jun LI
School of Computer Science and Engineering
lnslwj@mail.sysu.edu.cn

Lecture 5. LL(1) Parsing

1. Predictive Parsing Table
2. Table-Driven Parser
3. Error Recovery in LL(1) Parsing

1. Predictive Parsing Table

- Making decision of actions based on a parsing table
  - Precondition: LL(1) grammars
- Four actions in a top-down parser
  - *Derive*
  - *Match*
  - *Accept*
  - *Error*

Review

- Given the following grammar

$$
\begin{aligned}
E &\rightarrow T\,E' \\
E' &\rightarrow +\,T\,E' \mid \varepsilon \\
T &\rightarrow F\,T' \\
T' &\rightarrow *\,F\,T' \mid \varepsilon \\
F &\rightarrow \mathbf{(}\,E\,\mathbf{)} \mid \mathbf{n}
\end{aligned}
$$

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

Parsing Table: An Example

- For the previous grammar

Non-terminal	lookahead					
	n	**+**	*****	**(**	**)**	**$**
E	T E'			T E'		
E'		**+** T E'			ε	ε
T	F T'			F T'		
T'		ε	***** F T'		ε	ε
F	**n**			**(** E **)**		

Construction of Parsing Table

- Algorithm: for each $A \rightarrow \alpha$,
  - Add $A \rightarrow \alpha$ to M[A, **a**] for each **a** $\in$ FIRST(α) $\bigcap$ Σ.
  - If $\varepsilon \in$ FIRST(α), add $A \rightarrow \alpha$ to M[A, **b**] for each **b** $\in$ FOLLOW(A).
    - Note that b may be the $ symbol, which indicates the end of input tokens.
- Discussions: conflicts in a parsing table
  - Is it possible that one single entry has multiple productions?
  - What does it mean?

Conflicts in Parsing Table

- The left factored dangling-else grammar

$$S \rightarrow \textbf{if}\ E\ \textbf{then}\ S\ S' \mid \textbf{other}$$

$$S' \rightarrow \textbf{else}\ S \mid \varepsilon$$

$$E \rightarrow \textbf{expr}$$

Non-terminal	lookahead					
	if	**then**	**else**	**other**	**expr**	**$**
S	**if** E **then** S S'			other		
S'			ε else S			ε
E					expr	

Discussions

- Where conflicts occur in the parsing table?
  - Left recursion

    $$A \rightarrow A\,\mathbf{a} \mid \mathbf{b}$$

  - Left factor

    $$A \rightarrow \mathbf{a}\,\mathbf{b} \mid \mathbf{a}\,\mathbf{c}$$

  - Ambiguity

2. Table-Driven Parser

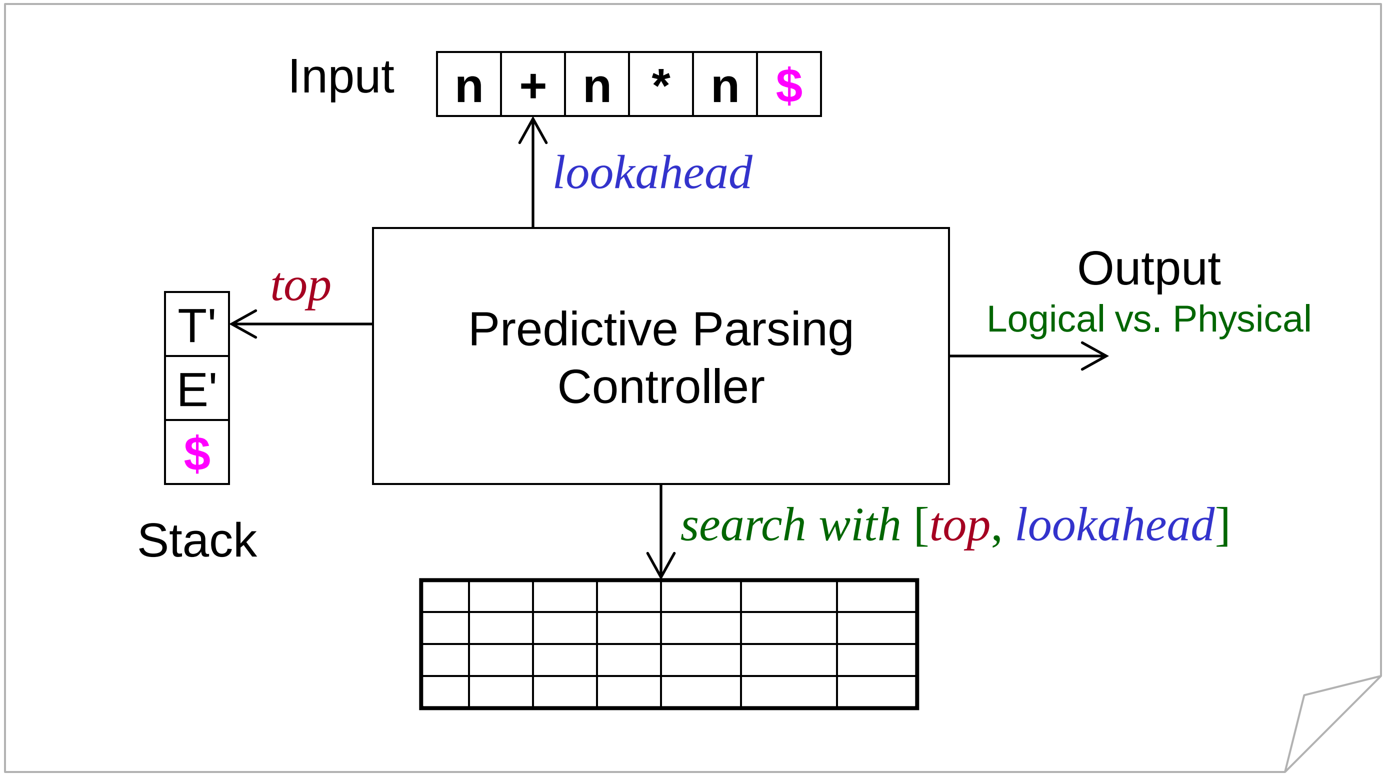
- No hard-codings of specific grammar
  - Parser is driven by a parsing table.
    - The controller is general enough to handle all LL(1) grammars.
  - The parsing table contains all information about the grammar.
    - Automatic generation of an LL(1) parser is the generation of a parsing table from the LL(1) grammar.

Model of Table-Driven Parsers

- Components of the parser
  - Specific to table-driven predictive parsers
    - Parsing Table
    - Controller
    - (Explicit) stack
  - General to all parsers
    - Input: can be abstracted as a lookahead.
    - Output: logical vs. physical

Model of Table-Driven Parsers (cont')

- A table-driven predictive parser



Initial and Accepting Configuration

- Initial configuration
  - Stack: **$ S**
  - Lookahead: points to the 1st input symbol and the input string ends with $
- Accepting configuration
  - Stack: $
  - Lookahead: points to $

Predictive Parsing Controller

```
initialize();
while (! stack.top().equals(new Symbol('$')) {
   top = stack.top();
   if (top instanceof Terminal) {  // match a terminal
      stack.pop();
      match(top);
   } else {  // derive a nonterminal
      if (table[top, lookahead] is X → Y1Y2...Yk) {
         output(X → Y1Y2...Yk);
         stack.pop();
         stack.push(Yk, Yk-1, ..., Y1);  // Y1 on top
      } else error();  // empty entry in parsing table
   }
}
if (lookahead.equals(new Symbol('$'))  accept();
else  error();
```

LL(1) Parsing: An Example

- Given the following CFG

$S \rightarrow \mathbf{a}\, B\, \mathbf{a}$

$B \rightarrow \mathbf{b}\, B \mid \varepsilon$

Non-terminal	lookahead		
	a	**b**	**$**
S	**a** B **a**		
B	ε	**b** B	

Non-terminal	lookahead		
	a	**b**	**$**
S	**a** B **a**		
B	ε	**b** B	

Parsing Process

Step	Stack	Input	Reference	Action	Output
0	**$** S	**a b b a $**	[S, **a**] = **a** B **a**	derive	S → **a** B **a**
1	**$ a** B **a**	**a b b a $**		match	
2	**$ a** B	**b b a $**	[B, **b**] = **b** B	derive	B → **b** B
3	**$ a** B **b**	**b b a $**		match	
4	**$ a** B	**b a $**	[B, **b**] = **b** B	derive	B → **b** B
5	**$ a** B **b**	**b a $**		match	
6	**$ a** B	**a $**	[B, **a**] = ε	derive	B → ε
7	**$ a**	**a $**		match	
8	**$**	**$**		accept	

LL(1) Parsing: More Examples

- Given an input string: **n + n * n**

Non-terminal	lookahead					
	n	**+**	*****	**(**	**)**	**$**
E	T E'			T E'		
E'		**+** T E'			ε	ε
T	F T'			F T'		
T'		ε	***** F T'		ε	ε
F	**n**			**(** E **)**		

Parsing Process

Non-terminal	lookahead					
	n	**+**	*****	**(**	**)**	**$**
E	T E'			T E'		
E'		**+** T E'			ε	ε
T	F T'			F T'		
T'		ε	***** F T'		ε	ε
F	**n**			**(** E **)**		

Step	Stack	Input	Reference	Action	Output
0	**$** E	**n + n * n $**	[E, **n**] = T E'	derive	E → T E'
1	**$** E' T	**n + n * n $**	[T, **n**] = F T'	derive	T → F T'
2	**$** E' T' F	**n + n * n $**	[F, **n**] = **n**	derive	F → **n**
3	**$** E' T' **n**	**n + n * n $**		match	
4	**$** E' T'	**+ n * n $**	[T', **+**] = ε	derive	T' → ε
5	**$** E'	**+ n * n $**	[E', **+**] = **+** T E'	derive	E' → **+** T E'
6	**$** E' T +	**+ n * n $**		match	
7	**$** E' T	**n * n $**	[T, **n**] = F T'	derive	T → F T'
8	**$** E' T' F	**n * n $**	[F, **n**] = **n**	derive	F → **n**

Parsing Process (cont')

Non-terminal	lookahead					
	n	**+**	*****	**(**	**)**	**$**
E	T E'			T E'		
E'		**+** T E'			ε	ε
T	F T'			F T'		
T'		ε	***** F T'		ε	ε
F	**n**			**(** E **)**		

Step	Stack	Input	Reference	Action	Output
9	**$** E' T' **n**	**n * n $**		match	
10	**$** E' T'	*** n $**	[T', *****] = ***** F T'	derive	T' → ***** F T'
11	**$** E' T' F *****	*** n $**		match	
12	**$** E' T' F	**n $**	[F, **n**] = **n**	derive	F → **n**
13	**$** E' T' **n**	**n $**		match	
14	**$** E' T'	**$**	[T', **$**] = ε	derive	T' → ε
15	**$** E'	**$**	[E', **$**] = ε	derive	E' → ε
16	**$**	**$**		accept	

Error Report: Missing Operand

Step	Stack	Input	Reference	Action	Output
0	**$** E	**n + * n $**	[E, **n**] = T E'	derive	E $\rightarrow$ T E'
1	**$** E' T	**n + * n $**	[T, **n**] = F T'	derive	T $\rightarrow$ F T'
2	**$** E' T' F	**n + * n $**	[F, **n**] = **n**	derive	F $\rightarrow$ **n**
3	**$** E' T' **n**	**n + * n $**		match	
4	**$** E' T'	**+ * n $**	[T', **+**] = ε	derive	T' $\rightarrow$ ε
5	**$** E'	**+ * n $**	[E', **+**] = **+** T E'	derive	E' $\rightarrow$ **+** T E'
6	**$** E' T +	**+ * n $**		match	
7	**$** E' T	*** n $**	[T, *****] = empty	error	

Missing operand

Error Report: Missing Operator

Step	Stack	Input	Reference	Action	Output
0	**$** E	**n n * n $**	[E, **n**] = T E'	derive	E $\rightarrow$ T E'
1	**$** E' T	**n n * n $**	[T, **n**] = F T'	derive	T $\rightarrow$ F T'
2	**$** E' T' F	**n n * n $**	[F, **n**] = **n**	derive	F $\rightarrow$ **n**
3	**$** E' T' **n**	**n n * n $**		match	
4	**$** E' T'	**n * n $**	[T', **n**] = empty	error	

Missing operator

Step	Stack	Input	Reference	Action	Output
0	**$** E	**n + n ($**	[E, **n**] = T E'	derive	E → T E'
1	**$** E' T	**n + n ($**	[T, **n**] = F T'	derive	T → F T'
2	**$** E' T' F	**n + n ($**	[F, **n**] = **n**	derive	F → **n**
3	**$** E' T' **n**	**n + n ($**		match	
4	**$** E' T'	**+ n ($**	[T', **+**] = ε	derive	T' → ε
5	**$** E'	**+ n ($**	[E', **+**] = **+** T E'	derive	E' → **+** T E'
6	**$** E' T +	**+ n ($**		match	
7	**$** E' T	**n ($**	[T, **n**] = F T'	derive	T → F T'
8	**$** E' T' F	**n ($**	[F, **n**] = **n**	derive	F → **n**
9	**$** E' T' **n**	**n ($**		match	
10	**$** E' T'	**($**	[T', **(**] = empty	error	

Missing operator

Step	Stack	Input	Reference	Action	Output
0	**$** E	**n + n) $**	[E, **n**] = T E'	derive	E → T E'
1	**$** E' T	**n + n) $**	[T, **n**] = F T'	derive	T → F T'
2	**$** E' T' F	**n + n) $**	[F, **n**] = **n**	derive	F → **n**
3	**$** E' T' **n**	**n + n) $**		match	
4	**$** E' T'	**+ n) $**	[T', **+**] = ε	derive	T' → ε
5	**$** E'	**+ n) $**	[E', **+**] = **+** T E'	derive	E' → **+** T E'
6	**$** E' T +	**+ n) $**		match	
7	**$** E' T	**n) $**	[T, **n**] = F T'	derive	T → F T'
8	**$** E' T' F	**n) $**	[F, **n**] = **n**	derive	F → **n**
9	**$** E' T' **n**	**n) $**		match	
10	**$** E' T'	**) $**	[T', *****] = ***** F T'	derive	T' → ***** F T'
11	**$** E' T'	**) $**	[T', **)**] = ε	derive	T' → ε
12	**$** E'	**) $**	[E', **)**] = ε	derive	E' → ε
13	**$**	**) $**		error	

3. Error Recovery in LL(1) Parsing

- What should the parser do in an error case?
  - Give an error message
    - Meaningful, as much as possible
  - Recover from that error case
    - Be able to continue the parsing with the rest of the input

Errors in LL(1) Parsing

- Three types of errors
  - Mismatch of terminals
    - Top of the stack is a terminal, but does not match with the lookahead.
  - [top, lookahead] = empty
    - There is no production candidate.
  - Empty stack with input remaining
    - The stack is empty, but the input string does not end.

Parsing Table with Synchronization

Non-terminal	lookahead					
	n	**+**	*****	**(**	**)**	**$**
E	T E'			T E'	synch	synch
E'		**+** T E'			ε	ε
T	F T'	synch		F T'	synch	synch
T'		ε	***** F T'		ε	ε
F	**n**	synch	synch	**(** E **)**	synch	synch

Drop the lookahead (more)

Drop the top (less)

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

Parsing with Error Recovery

- Error recovery strategies
  - If [*top*, *lookahead*] = empty, then
    - Skip the *lookahead*, i.e. forward the input and keep the stack unchanged.
  - If [*top*, *lookahead*] = synch, then
    - Skip the *top*, i.e. pop the stack and keep the input unchanged.
  - If *top* ≠ *lookahead*, then
    - Skip the *top*, i.e. pop the stack and keep the input unchanged.

Step	Stack	Input	Reference	Action	Output
0	**$** E	*** n * + n $**	[E, *****] = empty	skip input	error
1	**$** E	**n * + n $**	[E, **n**] = T E'	derive	E $\rightarrow$ T E'
2	**$** E' T	**n * + n $**	[T, **n**] = F T'	derive	T $\rightarrow$ F T'
3	**$** E' T' F	**n * + n $**	[F, **n**] = **n**	derive	F $\rightarrow$ **n**
4	**$** E' T' **n**	**n * + n $**		match	
5	**$** E' T'	*** + n $**	[T', *****] = ***** F T'	derive	T' $\rightarrow$ ***** F T'
6	**$** E' T' F *	*** + n $**		match	
7	**$** E' T' F	**+ n $**	[F, **+**] = synch	skip top	error
8	**$** E' T'	**+ n $**	[T', **+**] = ε	derive	T' $\rightarrow \varepsilon$
9	**$** E'	**+ n $**	[E', **+**] = **+** T E'	derive	E' $\rightarrow$ **+** T E'
10	**$** E' T **+**	**+ n $**		match	
11	**$** E' T	**n $**	[T, **n**] = F T'	derive	T $\rightarrow$ F T'
12	**$** E' T' F	**n $**	[F, **n**] = **n**	derive	F $\rightarrow$ **n**
13	**$** E' T' **n**	**n $**		match	
14	**$** E' T'	**$**	[T', **$**] = ε	derive	T' $\rightarrow \varepsilon$
15	**$** E'	**$**	[E', **$**] = ε	derive	E' $\rightarrow \varepsilon$
16	**$**	**$**		end	

Error Recovery Techniques

- Panic-Mode Error Recovery
  - Skip inputs until synchronizing token found.
- Phrase-Level Error Recovery
  - Assign each empty entry a specific error routine.
- Error-Productions
  - Suitable for common errors but not all errors.
- Global-Correction
  - Globally analyze the input to find the error.
  - Expensive and not in practice.

Exercise 5.1

- Given the following grammar

$$S \rightarrow (L) \mid a$$

$$L \rightarrow L, S \mid S$$

  - Construct an LL(1) parsing table for the grammar
    - Note: you must eliminate the left recursion first.
  - Draw the detailed process of the parsing of the sentence **`(a, (a, a))`**, follow the style in the previous slides.

Exercise 5.2 **

- Given the following grammar

$$A \rightarrow B \mid B\,C$$
$$B \rightarrow \mathbf{a}\,B \mid \varepsilon$$
$$C \rightarrow \mathbf{a}\,\mathbf{b}$$

  - Left factor the grammar.
  - After left factoring, is the grammar an LL(1) grammar? or is it an LL(k) grammar? and why?
    - Note: you may try the input string **ab**.

Further Reading

- Dragon Book, 2nd Edition (DBv2)
  - Comprehensive Reading:
    - Section 4.4.3–4.4.4 for LL(1) parsing.
    - Section 4.1.4 and 4.4.5 for error recovery in LL(1) parsing.
  - Skip Reading:
    - Section 4.2.7 for differences between CFGs and regular expressions.

Enjoy the Course!