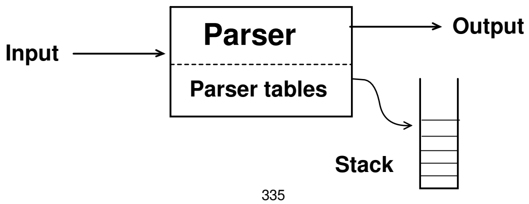
Reading Assignment

```
Fischer & LeBlanc

      Chapter  5
         Omit  5.8
```


LL(1) Parsing

- LL(1) is a Top Down parsing technique.
  An LL(1) parser is a *deterministic push down automata (DPDA)*
- Top down parsers are *predictive* parsers.
  Parse stack represents what the parser expects to see. As the parser encounters token that it expected to see, the parse stack gets modified to record this fact.
- LL(1) parser is controled by the incoming token and the top item in the parse stack. (LR parsers use more of the stack).
- If the LL(1) constructor algorithm in Slides 342 **..** 343 succeeds, i.e. there are no conflicting table entries for some particular row,column then the language is LL(1).



LL(1) Example

Grammar

S $\rightarrow$ d S A
  $\rightarrow$ b A c
A $\rightarrow$ d A
  $\rightarrow$ c

Input Tokens

d b c c d c ⊢

LL(1) Table

	d	b	c	⊢
S	pop S push dSa	pop S push bAc	Error	Error
A	pop A push dA	Error	pop A push c	Error
d	pop d Next	Error	Error	Error
b	Error	Pop b Next	Error	Error
c	Error	Error	pop c Next	Error
▽	Error	Error	Error	Accept

push xYz = push z push Y push x

Stack Snapshots

S ▽	
d S A ▽	
S A ▽	Next b
b A c A ▽	
A c A ▽	Next c
c c A ▽	
c A ▽	Next c
A ▽	Next d
d A ▽	
A ▽	Next c
c ▽	
▽	Next ⊢
	Accept

LL(1) - First and Follow Sets

- Notation

  $A, B, S, \ldots$ Nonterminal symbols
  $a, b, c, \ldots$ Terminal symbols
  $\alpha, \beta, \gamma, \ldots$ Strings of terminal and nonterminal symbols.

- Define: $first(\ \alpha\ )$
  For a string α ($\alpha\ \epsilon\ (\ V_n\ \cup\ V_t\ )^*$) the set $first(\ \alpha\ )$ is the set of *terminal* symbols that can occur at the beginning of strings derived from α. Examples

  $first(\ a\ \beta\ )$ = $\{a\}$ a is a terminal symbol
  $first(\ \lambda\ )$ = $\{\ \ \}$
  $first(\ B\ \alpha\ )$ = $first(\ B\ )$ B not nullable
   = $first(\ B\ )\ \cup\ first(\ \alpha\ )$ B nullable

  NOTE Fischer & LeBlanc include λ in $first$, these notes do not.

- Define: $follow(\ A\ )$
  For a nonterminal symbol A, the set $follow(\ A\ )$ is the set of *terminal symbols* that can immediately follow A in any intermediate string derived from the goal symbol S

  $follow(\ A\ ) = \{\ w\ |S \rightarrow^*\ \alpha\ A\ \beta\ w\ \gamma\ \wedge\ \beta\ is\ nullable\}$

LL(1) - Director Sets

- The LL(1) director sets are the decision mechanism that is used to decide among various alternatives for rewriting a nonterminal symbol.
- Define: *director set*

  Given a nonterminal A with several alternative definitions

  $$\begin{aligned} A &\rightarrow \alpha_1 \\ &\rightarrow \alpha_2 \\ &\quad \ldots \\ &\rightarrow \alpha_n \end{aligned}$$

  The director set for $A \rightarrow \alpha_i$ is

  $dir(A \rightarrow \alpha_i) = first(\alpha_i)$ α_i not nullable

  $dir(A \rightarrow \alpha_i) = first(\alpha_i) \cup follow(A)$ α_i is nullable
- For each nonterminal symbol in the grammar, the director sets for the nonterminal must be disjoint for the language to be LL(1).

Errors in Grammars

- Define: *extraneous nonterminals*

  Nonterminal symbols in a grammar are extraneous if they are

  a) *dead* - they do not produce any terminal strings

  b) *unreachable* - cannot be derived from the goal symbol S.
- Define: *unreachable non-terminal*

  The goal symbol S is reachable.

  If $A \rightarrow \alpha$ and A is reachable then all nonterminals in α are reachable.

  Iterate (transitive closure) until all reachable nonterminals have been detected. Any remaining nonterminals are unreachable.
- Define: *dead nonterminals*

  Dead non-terminals never produce a complete terminal string. Example:

$$\begin{array}{lll@{\qquad}lll} A & \rightarrow & a\ A & B & \rightarrow & b\ A \\ & \rightarrow & B & & \rightarrow & A \end{array}$$

Removing Left Recursion

- LL(1) parsers cannot handle production rules that are left recursive, for example: $A \rightarrow A\,\alpha$.
- Usually left recursion can be removed by introducing new non-terminal symbols and factoring the rules so that the revised rules satisfy the LL(1) property. Example:

$$\begin{array}{lll@{\qquad}lll} E & \rightarrow & E + T & E & \rightarrow & T\,Etail \\ & \rightarrow & T & Etail & \rightarrow & +T\,Etail \\ T & \rightarrow & T * P & & \rightarrow & \lambda \\ & \rightarrow & P & T & \rightarrow & P\,Ttail \\ P & \rightarrow & ID & Ttail & \rightarrow & *P\,Ttail \\ & & & & \rightarrow & \lambda \\ & & & P & \rightarrow & ID \end{array}$$

Fixing Director Set Conflicts

- The director sets for each non-terminal in the grammar must be disjoint for the grammar to be LL(1).
- Usually non disjoint director sets can be fixed by introducing extra non-terminal symbols to give the parser more context. (In effect, locally increasing the amount of lookahead.) Example:

$$\begin{array}{lll@{\qquad}lll} S & \rightarrow & a\ B & S & \rightarrow & a\ E \\ & \rightarrow & a\ c\ a & & \rightarrow & d \\ & \rightarrow & d & E & \rightarrow & b\ c \\ B & \rightarrow & b\ c & & \rightarrow & c\ F \\ & \rightarrow & c\ b & F & \rightarrow & b \\ & & & & \rightarrow & a \end{array}$$

LL(1) Table Construction Algorithm

0 Clean up the grammar.

- Remove ambiguities and left recursion.
- Delete extraneous nonterminal symbols.

1 The input set for the input state control of the DPDA (table column indices) is the set of terminal symbols plus the end marker $\vdash$

The stack symbol set for the stack top control of the DPDA (table row indices) is the set of nonterminal symbols plus the stack bottom marker $\bigtriangledown$ plus any terminal symbols that occur in the right hand side of productions in positions other than the extreme left.

2 The parser initial stack contents is the $S \ \bigtriangledown$ where S is the goal symbol for the grammar and $\bigtriangledown$ is the bottom of stack marker.

LL(1) Table Construction Algorithm

3 Construct the DPDA parser table.

3a row $\bigtriangledown$, col $\vdash$ $\leftarrow$ ACCEPT

3b if terminal symbol a is a stack symbol
  row a, col a $\leftarrow$ POP ; NEXT

3c if $A \rightarrow a \ \beta$ is a production
  row A, col a $\leftarrow$ REPLACE(β) NEXT

3d if $A \rightarrow B \ \alpha$ is a production and $B \ \alpha$ is not nullable
  for each b in $first(\ B \ \alpha \ )$
    row A, col b $\leftarrow$ REPLACE($B \ \alpha$)

3e if $A \rightarrow B \ \alpha$ is a production and $B \ \alpha$ is nullable
  for each b in $first(\ B \ \alpha) \ \cup \ follow(\ A \ )$
    row A, col b $\leftarrow$ REPLACE($B \ \alpha$)

3f if $A \rightarrow \lambda$ is a production
  for each b in $follow(\ A \ )$
    row A, col b $\leftarrow$ POP

3g All other entries in the table $\leftarrow$ ERROR

LL(1) Table Construction Example

- Grammar

$$
\begin{array}{lcl}
S & \rightarrow & B \ C \ c \\
 & | & e \ D \ B \\
B & \rightarrow & \lambda \\
 & | & b \ C \ D \ E \\
C & \rightarrow & D \ a \ B \\
 & | & c \ a \\
D & \rightarrow & \lambda \\
 & | & d \ S \\
E & \rightarrow & e \ S \ f \\
 & | & c
\end{array}
$$

- Define: *Begins Directly With (BDW)*
  A begins directly with x if $A \rightarrow \alpha \ x \ \beta$ is a production and α is nullable.
- Define: *Begins With (BW)*
  A begins with x if $A \ \rightarrow^* \ \alpha \ x \ \beta$ and α is nullable.

LL(1) Example - Begins Directly With

Construct Begins Directly With boolean matrix by inspection of the grammar.

	S	B	C	D	E	a	b	c	d	e	f
S		1	1							1	
B							1				
C				1		1		1			
D									1		
E								1		1	
a						1					
b							1				
c								1			
d									1		
e										1	
f											1

LL(1) Example - Begins With

The Begins With matrix is the symmetric transitive closure of the Begins Directly With relation. Compute the Begins With matrix from the BDW matrix.

$$BW = I + BDW + BDW^2 + BDW^3 + \cdots + BDW^{\infty}$$

	S	B	C	D	E	a	b	c	d	e	f
S	1	1	1	1		1	1	1	1	1	
B		1					1				
C			1	1		1		1	1		
D				1					1		
E					1			1		1	
a						1					
b							1				
c								1			
d									1		
e										1	
f											1

LL(1) Example - First & Follow Definitions

- $first(\alpha)$ comes directly from the rows of the begins with matrix.
  Examples:
  $first(D) = \{\, d \,\}$
  $first(B\ C\ c) = first(B) \cup first(C)$ B nullable
  $= \{\, b \,\} \cup \{\, a, c, d \,\}$
  $= \{\, a, b, c, d \,\}$
- Define: Is Followed Directly By (IFDB)
  A nonterminal symbol X is directly followed by a terminal symbol y if there is a production $G \rightarrow \alpha\ X\ \beta\ y\ \gamma$ and β is nullable.
- Define: Is Followed By (IFB)
  A nonterminal symbol X is followed by terminal symbol y if $S \rightarrow^* \alpha\ X\ \beta\ y\ \gamma$ and β is nulable.

LL(1) Example - Is Followed Directly By

Construct Is Followed Directly By matrix by inspection of the grammar.

	S	B	C	D	E	a	b	c	d	e	f
S											1
B			1								
C				1	1			1			
D		1			1	1					
E											
a		1									
b			1								
c						1					
d	1										
e	1	1		1							
f											

LL(1) Example - End Of

- Define: *Is Direct End Of (IDEO)*
  y is a direct end of a nonterminal X if there is a production
  $X \rightarrow \alpha\ y\ \beta$ and β is nullable.
  terminal symbol y is a direct end of itself.
- Define: *Is End Of* (IEO)
  y is an end of a nonterminal X if
  $S \rightarrow^* \alpha\ X\ \beta\ y$ and β is nullable.
- Redefine: Is Followed By (IFB)
  X is followed by y if there exist z, w such that
  X is end of z
  z is followed directly by w
  w begins with y

Read follow set from the IFB matrix.
Add X is followed by $\vdash$ whenever X is end of the goal symbol S.

LL(1) Example - Is Direct End Of

Construct Is Direct End Of matrix by inspection of the grammar.

	S	B	C	D	E	a	b	c	d	e	f
S		1		1				1			
B					1						
C		1				1					
D	1										
E								1			1
a						1					
b							1				
c								1			
d									1		
e										1	
f											1

LL(1) Example - Is End Of

The Is End Of matrix is the symmetric transitive closure of the Is Direct End Of relation. Compute the Is End Of matrix from the IDEO matrix.

$$BW = I + IDEO + IDEO^2 + IDEO^3 + \cdots + IDEO^\infty$$

	S	B	C	D	E	a	b	c	d	e	f
S	1	1		1	1			1			1
B		1			1			1			1
C		1	1		1	1		1			1
D	1	1		1	1			1			1
E					1			1			1
a						1					
b							1				
c								1			
d									1		
e										1	
f											1

LL(1) Example - Is Followed By

$$IFB = IEO \times IFDB \times BW$$

	S	B	C	D	E	a	b	c	d	e	f
S		1	1	1		1	1	1	1	1	1
B			1	1		1		1	1		
C		1	1	1		1	1	1	1	1	
D		1	1	1		1		1		1	1
E						1		1			
a		1					1				
b			1	1		1		1	1		
c						1					
d	1	1	1	1		1	1	1	1	1	
e	1	1	1	1		1	1	1	1	1	
f											

The upper right quadrant is $follow(\, X\,)$

LL(1) Example - First & Follow Sets

- First

$first(\, S\,)$	$\{\, a, b, c, d, e\, \}$
$first(\, B\,)$	$\{\, b\}$
$first(\, C\,)$	$\{\, a, c, d\}$
$first(\, D\,)$	$\{\, d\, \}$
$first(\, E\,)$	$\{\, c, e\}$

- Follow

$follow(\, S\,)$	$\{\, a, b, c, d, e, f, \vdash\}$
$follow(\, B\,)$	$\{\, a, c, d, \vdash\}$
$follow(\, C\,)$	$\{\, a, b, c, d, e\, \}$
$follow(\, D\,)$	$\{\, a, c, e, f, \vdash\}$
$follow(\, E\,)$	$\{\, a, c, \vdash\}$

LL(1) Example - Director Sets

- Director sets

S	$\rightarrow$	$B\,C\,c$	$\{\,a,\ b,\ c,\ d\,\}$
		$e\,D\,B$	$\{\,e\,\}$
B	$\rightarrow$	λ	$\{\,a,\ c,\ d,\ \vdash\}$
		$b\,C\,D\,E$	$\{\,b\,\}$
C	$\rightarrow$	$D\,a\,B$	$\{\,a,\ d\,\}$
		$c\,a$	$\{\,c\,\}$
D	$\rightarrow$	λ	$\{\,a,\ c,\ e,\ f,\ \vdash\}$
		$d\,S$	$\{\,d\,\}$
E	$\rightarrow$	$e\,S\,f$	$\{\,e\,\}$
		c	$\{\,c\,\}$

LL(1) Example - Director Set Examples

S	$\rightarrow$	$B\,C\,c$	
		$first(\ B\,C\,c\ )$	B nullable
		$first(\ B\ )\ \cup\ first(\ C\,c\ )$	
		$\{\,b\,\}\ \cup first(\ C\ )$	C not nullable
		$\{\,b\,\}\ \cup\ \{\,a,\ c,\ d\ \}$	
B	$\rightarrow$	λ	
		$first(\ \lambda\ )\ \cup\ follow(\ B\ )$	
		$\{\ \}\ \cup\ \{\,a,\ ,\ d,\ \vdash\}$	
C	$\rightarrow$	$D\,a\,B$	
		$first(\ D\ )\ \cup\ first(\ a\,B\ )$	D nullable
		$\{\,d\,\}\ \cup\ \{\,a\,\}$	
D	$\rightarrow$	λ	
		$first(\ \lambda\ )\ \cup\ follow(\ D\ )$	
		$\{\ \ \}\ \cup\ \{\,a,\ c,\ e,\ f,\ \vdash\}$	

LL(1) Example - Parse Table

	a	b	c	d	e	f	$\vdash$
S	Rep(BCc)	Rep(BCc)	Rep(BCc)	Rep(BCc)	Rep(DB) Advance		
B	Pop	Rep(CDE) Advance	Pop	Pop			
C			Rep(a) Advance	Rep(DaB) Advance	Pop	Pop	
D	Pop		Pop	Rep(S) Advance	Pop	Pop	
E			Pop		Rep(Sf) Advance		
$\bigtriangledown$							Accept
a	Pop Advance						
c			Pop Advance				
f						Pop Advance	

LL(1) Error Detection

- At first invalid input token can generate specific error message from the parse table:

  While looking for one of the following *list of terminal symbols*
  instead I found a *input token*.

- Can use information from the parse table to attempt a recovery from a syntax error.

- LL(k)

  Column indices become k-tuples of input symbols
  $(number\ of\ tokens)^k$ distinct columns

LL(1) vs LALR(1)

Attribute	LL(1)	LALR(1)
Simplicity	Easier to understand	More complex algorithms
Generality	Restrictions on grammar	More general
	non LL(1) constructs rare	Handles all modern languages
Action symbols	Anywhere on RHS	Only at right end of rule
		Fixed in yacc.
Error repair	Simpler	More complex
	Parser stack more useful	
Table size	$\mid V_n \mid \times \mid V_t \mid + \mid G \mid$	$\mid States \mid \times (\mid V_n \mid + \mid V_t \mid)$
		$+2\times \mid P \mid$
		worst case $\mid States \mid \sim\ 2^{\mid G \mid}$

Tables for Real Grammars

- Some observations from grammars for real programming languages:

$$\begin{aligned} \mid V_t \mid &\sim \tfrac{1}{2} \mid V_n \mid \\ \mid P \mid &\sim 2 \mid V_n \mid \\ \mid G \mid \approx 7 \mid V_n \mid &\approx 3\tfrac{1}{2} \mid P \mid \\ \mid States \mid \approx \mid P \mid &\approx \mid V_n \mid \end{aligned}$$

- About 10% non error table entries for LL(1)
  About 5% non error entries for LALR(1).
- Can prove the limiting case

$$\lim_{\mid V_n \mid \to \infty} \frac{\mid LALR(1) \mid}{\mid LL(1) \mid} \to 3$$

- For $\mid V_n \mid = 100$

$$\frac{\mid LALR(1) \mid}{\mid LL(1) \mid} \approx 1.58$$

User Interfaces for Compilers

- The importance of good *Human Engineering* in designing the interface between the compiler and the user cannot be over emphasized.
- **The compiler should describe problems with a program in terms that the user can understand.**
  - **Good**: Syntax error on line 100 of file myProgram.c
    The reserved word **if** cannot be followed by the identifier *myVar*
  - **Bad**: Illegal symbol pair **if** *identifier* . Parse stack dump:
    123 <identifier>
    122 **if**
    121 <statement>
    120 <block head>
    ...
  - **Really Bad:** Syntax error in program. Compilation terminated.

What the Compiler Should Do

- Determine if the program is correct.
- Describe the statically detectable errors at compile time.
- Translate the source program into an equivalent object program.
- Emit code to detect and describe dynamically detectable errors during program execution.
- Language designer (should) specify the possible errors in a language.
  Note the difference between error, implementation defined and unspecified.
- The language implementor decides between static and dynamic detection of errors.
- All errors *should be* detectable.

Error Detection Tradeoffs

- Static detection of errors may greatly increase
  - Complexity of the compilers internal data structures and algorithms
  - The time required to compile all programs.
- Example: To detect aliasing of variables the compiler needs links between all calls of a routine and the definition of the routine.
  The time to do aliasing detection is quadratic in the size of the program.
- Dynamic detection of errors at runtime may greatly increase
  - The size of the program due to extra chacking code (e.g. array subscript checking).
  - The execution time of the program due to the time required to execute checking code.
- Examples of expensive run time checking:
  - Uninitialized variable checking .
  - Array subscript checking .
  - Dangling and Nil pointer errors .

How the Compiler Can Help the User

- Use symbol table to produce information for the programmer
  - Cross reference list for identifier definition and use.
  - Frequency of usage information for identifiers.
  - Diagnostics for possible errors
    Variable/constant declared but never used.
    Variable assigned to but never used.
  - List sizes of data structures. Warn of excessive fill in data structures.
  - List code size and usage information for routines.
- Identify potentially inefficient constructs for the user.
  Statement 415 in yourProg.c generated 600 bytes of code.
  Statement 311 in utility.c implemented by 10 calls to library routines.
- Optionally summarize compiler internal resource usage.
  Used 400 of 511 symbol table entries.
  Provides warning of possible overrun of compiler limits.

Compiler Reaction to Errors in Programs

- Crash or Infinite Loop.
- Generate incorrect object program and no error messages.
- Stop the compilation
- Recover from the error and continue compilation.
  Hard to do well. Error cascades and false errors are problems.
- Repair error and continue the compilation
  Perfect correction is undecidable.
  Use heuristics and systematic strategies.
- **Errors should be detected as soon as possible.**
  Localize (in a routine) the production of error messages.
  Optionally print error summary at end of compilation
  13 Errors detected, Last error on line 219 of file urProg.c

Compiler Should be Self Diagnostic

- Compiler should be *error imune*, i.e. never crash for *any* possible input.
- Compiler must be programmed to detect *all* violations of compiler internal limits.
- Do internal consistency checking of compiler data structures.
- Use exception handlers to trap programming errors and shut down gracefully.
- Optionally print out compiler internal performance statistics for use in compiler tuning
  10,301 identifier lookups performed. 73% were local scope.
  4,219 scanner tokens produced, 3745 nodes in parse tree.
  143,216 bytes of code generated, 40,219 instructions.
  10.37 seconds in semantic analysis, 21.7 seconds in code generation.