Top Down Parsing

- Start at the root of the parse tree and grow toward leaves.
- Pick a production and try to match the input.
- Repeat until the fringe of the parse tree matches the input string.

Grammars and Parsers

LL(1) parsers

- **L**eft-to-right input
- **L**eftmost derivation
- **1** symbol of look-ahead

LL(1) Grammars

- **Def:** a grammar is LL(1) iff

  $A \rightarrow \alpha$ and $A \rightarrow \beta$ and
  $\text{FIRST+}(A \rightarrow \alpha) \cap \text{FIRST+}(A \rightarrow \beta) = \varnothing$

  LL(1) grammars are:

  - not ambiguous and
  - not left-recursive

Example

#	Production rule
1	Tern -> '0'..'9' '?' Tern ':' Tern
2	| '0'..'9'

- Problem?
- How do we predict which production to use?

Left factoring

#	Production rule
1	Tern -> '0'..'9' TernTail
2	TernTail -> '?' Tern ':' Tern
3	| e

FIRST and FOLLOW sets

#	Production rule
1	Tern -> '0'..'9' TernTail
2	TernTail -> '?' Tern ':' Tern
3	| e

FIRST(Tern) = { '0' .. '9'}
FIRST(TernTail) = { '?', e }

FOLLOW(Tern) = { ':', EOF }
FOLLOW(TernTail) = FOLLOW(Tern) = { ':', EOF }

FIRST+ sets

#	Production rule
1	Tern -> '0'..'9' TernTail
2	TernTail -> '?' Tern ':' Tern
3	| e

FIRST+(Tern) = { '0' .. '9'}

FIRST+(TernTail) = FIRST(TernTail) U
FOLLOW(TernTail) = { '?', e, ':', EOF }

Table-driven approach

	'0' .. '9'	':'	'?'	EOF
Tern	'0'..'9' TernTail	error	error	error
TernTail	error	e	'?' Tern ':' Tern	e

Recursive descent

- Define a function for each nonterminal.
- Have these functions call each other based on the lookahead token.
- The term *descent* refers to the direction in which the parse tree is built.