Mohammad T. Irfan
CSCI 2320: Principles of Progr. Lang.
Bowdoin College

Table-driven LL(1) parsing algorithm

References: Michael Scott's Programming Languages, Tucker-Noonan's Programming Languages
Website: https://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1/

Grammar in "augmented form":

S is an artificial start state, whereas E is the actual start state. We treat $ as a terminal symbol denoting the end of input. Must remove left recursion from BNF and do left factoring. Example below: E stands for Expr, T for Term, and F for Factor. Removing left recursion from E → E + T gives us E → T E', E' → + T E', and E' → ε.

S → E $
E → T E'
E' → + T E'
E' → ε
T → F T'
T' → * F T'
T' → ε
F → (E)
F → id

$$\text{FIRST}(\alpha) \equiv \{a : \alpha \Rightarrow^* a\,\beta\} \cup (\text{ if } \alpha \Rightarrow^* \epsilon \text{ then } \{\epsilon\} \text{ else } \varnothing\,)$$

$$\text{FOLLOW}(A) \equiv \{a : S \Rightarrow^+ \alpha\, A\, a\,\beta\} \cup (\text{ if } S \Rightarrow^* \alpha\, A \text{ then } \{\epsilon\} \text{ else } \varnothing\,)$$

A: nonterminal, a: terminal, ε: empty string, other Greek letters: sentential form (i.e., a string of terminal and/or non-terminal symbols).

FIRST(α) = Set of terminal symbols (tokens) occurring on the left of any sentential form derived from α.
Special rules: Add ε to FIRST(α) if α is nullable (i.e., α derives ε). For a terminal symbol a, FIRST(a) = $\{a\}$.

FOLLOW(A) = Set of terminal symbols appearing after A in any sentential form derived from S.
Special rule: Add ε to FOLLOW(A) when S derives a sentential form ending with A.

Non-terminal	Nullable?	FIRST	FOLLOW
S	✗	(, id	
E	✗	(, id	), $
E'	✓	+, ε	), $
T	✗	(, id	+,), $
T'	✓	*, ε	+,), $
F	✗	(, id	+, *,), $

LL(1) parse table construction:

Consider Table[A, a] and each of A's production rules A → α.

- If FIRST(α) contains a, enter A → α into Table[A, a]
- If FIRST(α) contains ε, enter A → ε into Table[A, b] for each terminal b in FOLLOW(A).

	$	**+**	*****	**(**	**)**	**id**
S				S → E $		S → E $
E				E → T E'		E → T E'
E'	E' → ε	E' → + T E'			E' → ε	
T				T → F T'		T → F T'
T'	T' → ε	T' → ε	T' → * F T'		T' → ε	
F				F → (E)		F → id

Multiple productions in a cell means predictive parsing like LL(1) is impossible for the given grammar.

Parsing Algorithm:

First, push S into the stack. Repeat the following steps, where **X** is the current top of the stack and **a** is the current input token:

- If X = a = $ → Successful parsing!
- Else if X is a terminal symbol:
  - X = a: pop X off the stack and get the next input token.
  - X ≠ a: Error!
- Else if X is a nonterminal symbol:
  - Table[X, a] has nothing → Syntax Error!
  - Table[X, a] contains X → α: pop X and push α with the leftmost symbol of α on top.