

## 04 - Testing and Manipulating Grammars

Dr. Robert Lowe

Division of Mathematics and Computer Science  
Maryville College

# Outline

1 Grammars and Recursion

2 LL(1) Grammars

# Sample Grammar $G$

For this discussion, we will be using the following grammar (found on page 39 of your textbook):

$$S \rightarrow E$$

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

$$T \rightarrow F \mid T * F$$

$$F \rightarrow U \mid (E)$$

$$U \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

# General Top Down Parsing

- 1 Look at the next symbol of input. This is the **target** symbol.
- 2 Expand the next non-terminal in the sentence.
- 3 If the target symbol does not match, backtrack and select a different non-terminal.
- 4 Keep repeating the process until there are either no non-terminal candidates or until there are no non-terminals left in the sentence.

# The Backtracking Problem

- Even with the best of luck, a backtracking parser would be exponential in runtime!
- A left-recursive grammar could lead to an infinite number of candidates.
- Recall that the sample grammar in the textbook has the rule  $E \rightarrow E + T$
- Consider the following expansion for the grammar from the textbook:

$$\begin{array}{ll} E + T & 1 + 2 * 3 \\ E + T + T & 1 + 2 * 3 \\ E + T + T + T & 1 + 2 * 3 \\ E + T + T + T + T & 1 + 2 * 3 \\ \dots & \end{array}$$

# Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recursive choices to the end, maybe this would fix it!
- What if we took the grammar  $G$  and imposed the order  $\langle T, F, U \rangle$  on expansions?

|           |             |
|-----------|-------------|
| $T$       | $1 + 2 * 3$ |
| $F$       | $1 + 2 * 3$ |
| $U$       | $1 + 2 * 3$ |
| $1$       | $1 + 2 * 3$ |
| $\lambda$ | $+ 2 * 3$   |

Mismatch! Backtrack!

# Ordering Recursion (Continued)

|                               |             |
|-------------------------------|-------------|
| $T * F$                       | $1 + 2 * 3$ |
| $F * F$                       | $1 + 2 * 3$ |
| $U * F$                       | $1 + 2 * 3$ |
| $1 * F$                       | $1 + 2 * 3$ |
| Mismatch! Backtrack!          |             |
| $T * F * F$                   | $1 + 2 * 3$ |
| ...                           |             |
| And there's the loop again... |             |

# Deterministic Parsing

- Backtracking is parsing by “brute force”.
- Backtracking essentially explores every possible production, searching for a match.
- Generally, we want parse times to be proportional to the size of the input, not exponential.
- Undoing parsing is difficult!
- We need some way to determine what production we must have based on the symbols being examined.



# LL( $k$ ) Grammars

- Instead of guessing and checking, we maintain a buffer of terminals.
- If a grammar is decidable using  $k$  terminals, we call this a  $k$ -lookahead grammar.
- We can further classify the grammar by its scanning order and which production it expands first.
- An LL( $k$ ) grammar is a grammar that is scanned from left to right and expands the left most derivation.
- RL( $k$ ) scans input from right to left, expanding left-most derivations.
- LR( $k$ ) scans from left to right, expanding left-most derivations.
- All of the above have a look-ahead buffer of  $k$  terminals.
- We are really interested in LL(1) grammars.

# Defining LL(1) Grammars

- Suppose we have a target expansion  $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$
- We must be able to select  $\alpha_i$  by looking at the next symbol.
- For each production, we must have a disjoint **director set**  $D(A \rightarrow \alpha_i)$ .
- For lookup buffer  $s$ ,  $A \rightarrow \alpha_i$  iff  $s \in D(A \rightarrow \alpha_i)$ .
- We can also have a set of symbols which immediately identify as an error if they are encountered.

# Calculating Director Sets

- If  $\alpha_i \xRightarrow{*} t\gamma$  for some terminal  $t$
- Then  $t \in D(A \rightarrow \alpha_i)$
- Because  $A \xRightarrow{+} t\gamma$  is a valid derivation.
- Let  $<<$  be an operator over  $(N \cup T)$  such that
$$\beta << \alpha \iff \exists \alpha \rightarrow \beta$$
- The reflexive transitive closure  $<<^*$  is therefore the “Can Start” relation
- The **start set** is  $\text{START}(\alpha) = \beta : \beta <<^* \alpha$
- Considering  $\alpha_i = \beta_1\beta_2 \dots \beta_r$  then
$$t \in \text{START}(\beta_i) \implies t \in D(A \rightarrow \alpha_i)$$

# Start Sets of $G$

$$\text{START}(U) = \{U, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$\text{START}(F) = \{\{F, () \cup \text{START}(U)\}$$

$$\text{START}(T) = \{\{T\} \cup \text{START}(F)\}$$

$$\text{START}(E) = \{\{E\} \cup \text{START}(T)\}$$

$$\text{START}(S) = \{\{S\} \cup \text{START}(E)\}$$

- Is  $G$  an LL(1) grammar?
- NO! In fact, no grammar containing left-recursive rules is LL(1)!
- $D(A \rightarrow A\gamma) \subseteq \text{START}(A)$