04 - Testing and Manipulating Grammars

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Outline

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):

$$
\begin{aligned}
S &\rightarrow E \\
E &\rightarrow T \mid E + T \\
T &\rightarrow F \mid T * F \\
F &\rightarrow U \mid (E) \\
U &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

General Top Down Parsing

1. Look at the next symbol of input. This is the **target** symbol.

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.

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, bactrack and select a different non-terminal.

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, bactrack 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!

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.

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 $E \rightarrow E + T$

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:

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:

$$E + T \qquad\qquad 1 + 2 * 3$$

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:

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

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:

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

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:

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

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:

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

Ordering Recursion

- Left recursion causes problems in candidate expansions.

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive choices to the end, maybe this would fix it!

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive 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?

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive 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?

$$T \qquad\qquad\qquad\qquad 1 + 2 * 3$$

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive 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?

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

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive 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	$1 + 2 * 3$

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive 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?

$$
\begin{array}{ll}
T & 1 + 2 * 3 \\
F & 1 + 2 * 3 \\
U & 1 + 2 * 3 \\
1 & 1 + 2 * 3
\end{array}
$$

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive 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?

$$
\begin{array}{ll}
T & 1 + 2 * 3 \\
F & 1 + 2 * 3 \\
U & 1 + 2 * 3 \\
1 & 1 + 2 * 3 \\
\lambda & +2 * 3
\end{array}
$$

Ordering Recursion

- Left recursion causes problems in candidate expansions.
- Perhaps we could organize a grammar to mitigate the expansion problem.
- If we move left recusrive 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?

$$\begin{array}{ll}
T & 1+2*3 \\
F & 1+2*3 \\
U & 1+2*3 \\
1 & 1+2*3 \\
\lambda & +2*3 \\
\text{Mismatch! Backtrack!} &
\end{array}$$

Ordering Recursion (Continued)

$T * F$ $\qquad\qquad$ $1 + 2 * 3$

Ordering Recursion (Continued)

$$T * F \qquad\qquad 1 + 2 * 3$$
$$F * F \qquad\qquad 1 + 2 * 3$$

Ordering Recursion (Continued)

$T * F$	$1 + 2 * 3$
$F * F$	$1 + 2 * 3$
$U * F$	$1 + 2 * 3$

Ordering Recursion (Continued)

$T * F$	$1 + 2 * 3$
$F * F$	$1 + 2 * 3$
$U * F$	$1 + 2 * 3$
$1 * F$	$1 + 2 * 3$

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!	

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!	
$T * F * F$	$1 + 2 * 3$

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!	
$T * F * F$	$1 + 2 * 3$
$\ldots$	

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!	
$T * F * F$	$1 + 2 * 3$
$\ldots$	
And there's the loop again...	

Outline

Deterministic Parsing

- Backtracking is parsing by "brute force".

Deterministic Parsing

- Backtracking is parsing by “brute force”.
- Backtracking essentially explores every possible production, searching for a match.

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.

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!

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.

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.

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.

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 derivaion.

$\mathrm{LL}(k)$ 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 $\mathrm{LL}(k)$ grammar is a grammar that is scanned from left to right and expands the left most derivaion.
- $\mathrm{RL}(k)$ scans input from right to left, exapanding left-most derivations.

$\mathrm{LL}(k)$ 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 $\mathrm{LL}(k)$ grammar is a grammar that is scanned from left to right and expands the left most derivaion.
- $\mathrm{RL}(k)$ scans input from right to left, exapanding left-most derivations.
- $\mathrm{LR}(k)$ scans from left to right, expanding left-most derivations.

$\mathrm{LL}(k)$ 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 $\mathrm{LL}(k)$ grammar is a grammar that is scanned from left to right and expands the left most derivaion.
- $\mathrm{RL}(k)$ scans input from right to left, exapanding left-most derivations.
- $\mathrm{LR}(k)$ scans from left to right, expanding left-most derivations.
- All of the above have a look-ahead buffer of k terminals.

$\mathrm{LL}(k)$ 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 $\mathrm{LL}(k)$ grammar is a grammar that is scanned from left to right and expands the left most derivaion.
- $\mathrm{RL}(k)$ scans input from right to left, exapanding left-most derivations.
- $\mathrm{LR}(k)$ 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 $\mathrm{LL}(1)$ grammars.

Defining LL(1) Grammars

- Suppose we have a target expansion $A \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$

Defining $\mathrm{LL}(1)$ Grammars

- Suppose we have a target expansion $A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$
- We must be able to select α_i by looking at the next symbol.

Defining $\mathrm{LL}(1)$ Grammars

- Suppose we have a target expansion $A \rightarrow \alpha_1|\alpha_2|\dots|\alpha_n$
- We must be able to select α_i by looking at the next symbol.
- For each production, we must have a disjoin **director set** $D(A \rightarrow \alpha_i)$.

Defining $\mathrm{LL}(1)$ Grammars

- Suppose we have a target expansion $A \to \alpha_1|\alpha_2|\ldots|\alpha_n$
- We must be able to select α_i by looking at the next symbol.
- For each production, we must have a disjoin **director set** $D(A \to \alpha_i)$.
- For lookup buffer s, $A \to \alpha_i$ iff $s \in D(A \to \alpha_i)$.

Defining $\mathrm{LL}(1)$ Grammars

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

Calculating Director Sets

- If $\alpha_i \overset{*}{\Rightarrow} t\gamma$ for some terminal t

Calculating Director Sets

- If $\alpha_i \stackrel{*}{\Rightarrow} t\gamma$ for some terminal t
- Then $t \in D(A \rightarrow \alpha_i)$

Calculating Director Sets

- If $\alpha_i \stackrel{*}{\Rightarrow} t\gamma$ for some terminal t
- Then $t \in D(A \to \alpha_i)$
- Because $A \stackrel{+}{\Rightarrow} t\gamma$ is a valid derivation.

Calculating Director Sets

- If $\alpha_i \overset{*}{\Rightarrow} t\gamma$ for some terminal t
- Then $t \in D(A \rightarrow \alpha_i)$
- Because $A \overset{+}{\Rightarrow} t\gamma$ is a valid derivation.
- Let $<<$ be an operator over $(N \cup T)$ such that $\beta << \alpha \iff \exists \alpha \rightarrow \beta$

Calculating Director Sets

- If $\alpha_i \overset{*}{\Rightarrow} t\gamma$ for some terminal t
- Then $t \in D(A \rightarrow \alpha_i)$
- Because $A \overset{+}{\Rightarrow} t\gamma$ 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

Calculating Director Sets

- If $\alpha_i \overset{*}{\Rightarrow} t\gamma$ for some terminal t
- Then $t \in D(A \to \alpha_i)$
- Because $A \overset{+}{\Rightarrow} t\gamma$ is a valid derivation.
- Let $<<$ be an operator over $(N \cup T)$ such that $\beta << \alpha \iff \exists \alpha \to \beta$
- The reflexive transitive closure $<<^*$ is therefore the "Can Start" relation
- The **start set** is $\mathrm{START}(\alpha) = \beta : \beta <<^* \alpha$

Calculating Director Sets

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

Start Sets of G

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

Start Sets of G

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

Start Sets of G

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

Start Sets of G

$$\begin{aligned}
\text{START}(U) &= \{U, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
\text{START}(F) &= \{\{F, (\} \cup \text{START}(U)\} \\
\text{START}(T) &= \{\{T\} \cup \text{START}(F)\} \\
\text{START}(E) &= \{\{E\} \cup \text{START}(T)\}
\end{aligned}$$

Start Sets of G

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

Start Sets of G

$$
\begin{aligned}
\mathrm{START}(U) &= \{U, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
\mathrm{START}(F) &= \{\{F, (\} \cup \mathrm{START}(U)\} \\
\mathrm{START}(T) &= \{\{T\} \cup \mathrm{START}(F)\} \\
\mathrm{START}(E) &= \{\{E\} \cup \mathrm{START}(T)\} \\
\mathrm{START}(S) &= \{\{S\} \cup \mathrm{START}(E)\}
\end{aligned}
$$

- Is G an $\mathrm{LL}(1)$ grammar?

Start Sets of G

$$
\begin{aligned}
\mathrm{START}(U) &= \{U, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
\mathrm{START}(F) &= \{\{F, (\} \cup \mathrm{START}(U)\} \\
\mathrm{START}(T) &= \{\{T\} \cup \mathrm{START}(F)\} \\
\mathrm{START}(E) &= \{\{E\} \cup \mathrm{START}(T)\} \\
\mathrm{START}(S) &= \{\{S\} \cup \mathrm{START}(E)\}
\end{aligned}
$$

- Is G an $\mathrm{LL}(1)$ grammar?
- NO! In fact, no grammar containing left-recursive rules is $\mathrm{LL}(1)$!

Start Sets of G

$$\begin{aligned}
\mathrm{START}(U) &= \{U, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
\mathrm{START}(F) &= \{\{F, (\} \cup \mathrm{START}(U)\} \\
\mathrm{START}(T) &= \{\{T\} \cup \mathrm{START}(F)\} \\
\mathrm{START}(E) &= \{\{E\} \cup \mathrm{START}(T)\} \\
\mathrm{START}(S) &= \{\{S\} \cup \mathrm{START}(E)\}
\end{aligned}$$

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