CS415 Compilers
Syntax Analysis
Top-Down Parsing

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University
• Regarding the instruction scheduling project
  → ILOC simulator updated: does not affect your code implementation & only affect your report (statistics) if you have started writing report
  → Longest latency weighted path: multiple instructions that have no successors, pick the longest path to any one of them
  → Memory reference instructions: do not change the relative order of the memory instructions
Top-Down Parsing
(Syntax Analysis)

EAC Chapters 3.1 – 3.3
**LL(1), recursive descent**

1 input symbol lookahead

construct leftmost derivation (forwards)

input: read left-to-right

\[
S \Rightarrow^* \beta \Rightarrow \delta \Rightarrow^* y
\]

- \( S \Rightarrow^*_{lm} \beta \Rightarrow_{lm} \delta \Rightarrow^*_{lm} y \)
**LL(1), recursive descent**

1 input symbol lookahead

Construct leftmost derivation (forwards)

Input: read left-to-right

Rule: $A \rightarrow \delta$

$$S \Rightarrow^*_{lm} x A \beta \Rightarrow_{lm} x \delta \beta \Rightarrow^*_{lm} x y$$

Diagram:

```
S
  /|
 / A
/ β
δ
```

Input: $x$ $y$
A top-down parser starts with the root of the parse tree
The root node is labeled with the goal symbol of the grammar

Top-down parsing algorithm:

1. Construct the root node of the parse tree
2. Repeat until the fringe of the parse tree matches the input string
   1. At a node labeled A, select a production with A on its LHS and, for each symbol on its RHS, construct the appropriate child
   2. When a terminal symbol is added to the fringe and it doesn’t match the fringe, backtrack
   3. Find the next node to be expanded

• The key is picking the right production in step 1
  → That choice should be guided by the input string
Remember the expression grammar?

Version with precedence

1. \( \text{Goal} \rightarrow \text{Expr} \)
2. \( \text{Expr} \rightarrow \text{Expr} + \text{Term} \)
3. \( \mid \text{Expr} - \text{Term} \)
4. \( \mid \text{Term} \)
5. \( \text{Term} \rightarrow \text{Term} \times \text{Factor} \)
6. \( \mid \text{Term} / \text{Factor} \)
7. \( \mid \text{Factor} \)
8. \( \text{Factor} \rightarrow \text{number} \)
9. \( \mid \text{id} \)

And the input: \( x - 2 \times y \)
Let's try $x - 2 * y$:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentential Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Goal</td>
<td>$\uparrow x - 2 * y$</td>
</tr>
<tr>
<td>1</td>
<td>Expr</td>
<td>$\uparrow x - 2 * y$</td>
</tr>
<tr>
<td>2</td>
<td>Expr + Term</td>
<td>$\uparrow x - 2 * y$</td>
</tr>
<tr>
<td>4</td>
<td>Term + Term</td>
<td>$\uparrow x - 2 * y$</td>
</tr>
<tr>
<td>7</td>
<td>Factor + Term</td>
<td>$\uparrow x - 2 * y$</td>
</tr>
<tr>
<td>9</td>
<td>&lt;id,x&gt; + Term</td>
<td>$\uparrow x - 2 * y$</td>
</tr>
<tr>
<td>9</td>
<td>&lt;id,x&gt; + Term</td>
<td>$x \uparrow - 2 * y$</td>
</tr>
</tbody>
</table>

*Leftmost derivation, choose productions in an order that exposes problems*
Let’s try \( x - 2 * y \):

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentential Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \text{Expr} )</td>
<td>( \uparrow x - 2 * y )</td>
</tr>
<tr>
<td>2</td>
<td>( \text{Expr} + \text{Term} )</td>
<td>( \uparrow x - 2 * y )</td>
</tr>
<tr>
<td>4</td>
<td>( \text{Term} + \text{Term} )</td>
<td>( \uparrow x - 2 * y )</td>
</tr>
<tr>
<td>7</td>
<td>( \text{Factor} + \text{Term} )</td>
<td>( \uparrow x - 2 * y )</td>
</tr>
<tr>
<td>9</td>
<td>( &lt;\text{id},x&gt; + \text{Term} )</td>
<td>( \uparrow x - 2 * y )</td>
</tr>
<tr>
<td>9</td>
<td>( &lt;\text{id},x&gt; + \text{Term} )</td>
<td>( x \uparrow - 2 * y )</td>
</tr>
</tbody>
</table>

This worked well, except that “-” doesn’t match “+”

The parser must backtrack to here
Continuing with $x - 2 \times y$:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentential Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Expr</td>
<td>$x - 2 \times y$</td>
</tr>
<tr>
<td>3</td>
<td>Expr – Term</td>
<td>$x - 2 \times y$</td>
</tr>
<tr>
<td>4</td>
<td>Term – Term</td>
<td>$x - 2 \times y$</td>
</tr>
<tr>
<td>7</td>
<td>Factor – Term</td>
<td>$x - 2 \times y$</td>
</tr>
<tr>
<td>9</td>
<td>&lt;id,x&gt; – Term</td>
<td>$x - 2 \times y$</td>
</tr>
<tr>
<td>9</td>
<td>&lt;id,x&gt; – Term</td>
<td>$x - 2 \times y$</td>
</tr>
<tr>
<td>—</td>
<td>&lt;id,x&gt; – Term</td>
<td>$x - 2 \times y$</td>
</tr>
</tbody>
</table>
Continuing with $x - 2 * y$:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentential Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>Goal</td>
<td>↑x – 2 * y</td>
</tr>
<tr>
<td>1</td>
<td>Expr</td>
<td>↑x – 2 * y</td>
</tr>
<tr>
<td>3</td>
<td>Expr – Term</td>
<td>↑x – 2 * y</td>
</tr>
<tr>
<td>4</td>
<td>Term – Term</td>
<td>↑x – 2 * y</td>
</tr>
<tr>
<td>7</td>
<td>Factor – Term</td>
<td>↑x – 2 * y</td>
</tr>
<tr>
<td>9</td>
<td>&lt;id,x&gt; – Term</td>
<td>↑x – 2 * y</td>
</tr>
<tr>
<td>9</td>
<td>&lt;id,x&gt; – Term</td>
<td>x ↑− 2 * y</td>
</tr>
<tr>
<td>—</td>
<td>&lt;id,x&gt; – Term</td>
<td>x – ↑2 * y</td>
</tr>
</tbody>
</table>

This time, “−” and “−” matched

⇒ Now, we need to expand $\text{Term}$ - the last $NT$ on the fringe
Example

Trying to match the “2” in \( x - 2 \cdot y \):

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentential Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle \text{id},x \rangle - \text{Term} )</td>
<td>( x \leftarrow 2 \cdot y )</td>
</tr>
<tr>
<td>7</td>
<td>( \langle \text{id},x \rangle - \text{Factor} )</td>
<td>( x \leftarrow 2 \cdot y )</td>
</tr>
<tr>
<td>9</td>
<td>( \langle \text{id},x \rangle - \langle \text{num},2 \rangle )</td>
<td>( x \leftarrow 2 \cdot y )</td>
</tr>
<tr>
<td></td>
<td>( \langle \text{id},x \rangle - \langle \text{num},2 \rangle )</td>
<td>( x \leftarrow 2 \cdot y )</td>
</tr>
</tbody>
</table>

Diagram:

```
Goal
  └───Expr
       └───Term
            │
            ├── Fact.
            │    │
            │    └───<num,2>
            └───<id,x>
```

GUESS

12
Example

Trying to match the “2” in $x - 2* y$:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentential Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$&lt;id, x&gt; - Term$</td>
<td>$x - \uparrow 2 * y$</td>
</tr>
<tr>
<td>7</td>
<td>$&lt;id, x&gt; - Factor$</td>
<td>$x - \uparrow 2 * y$</td>
</tr>
<tr>
<td>9</td>
<td>$&lt;id, x&gt; - &lt;num, 2&gt;$</td>
<td>$x - \uparrow 2 * y$</td>
</tr>
<tr>
<td></td>
<td>$&lt;id, x&gt; - &lt;num, 2&gt;$</td>
<td>$x - 2* y$</td>
</tr>
</tbody>
</table>

Where are we?

- “2” matches “2”
- We have more input, but no NTs left to expand
- The expansion terminated too soon
  $\Rightarrow$ Need to backtrack
Example

Trying again with "2" in $x - 2 \times y$:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentenceal Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$&lt;id,x&gt; - Term * Factor$</td>
<td>$x - \uparrow 2 \times y$</td>
</tr>
<tr>
<td>7</td>
<td>$&lt;id,x&gt; - Factor * Factor$</td>
<td>$x - \uparrow 2 \times y$</td>
</tr>
<tr>
<td>8</td>
<td>$&lt;id,x&gt; - &lt;num,2&gt; * Factor$</td>
<td>$x - \uparrow 2 \times y$</td>
</tr>
<tr>
<td>9</td>
<td>$&lt;id,x&gt; - &lt;num,2&gt; * &lt;id,y&gt;$</td>
<td>$x - \uparrow 2 \times y$</td>
</tr>
</tbody>
</table>

This time, we matched & consumed all the input

⇒ Success!
Other choices for expansion are possible

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentential Form</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>Goal</td>
<td>(\uparrow x - 2 \times y)</td>
</tr>
<tr>
<td>1</td>
<td>Expr</td>
<td>(\uparrow x - 2 \times y)</td>
</tr>
<tr>
<td>2</td>
<td>Expr + Term</td>
<td>(\uparrow x - 2 \times y)</td>
</tr>
<tr>
<td>2</td>
<td>Expr + Term + Term</td>
<td>(\uparrow x - 2 \times y)</td>
</tr>
<tr>
<td>2</td>
<td>Expr + Term + Term + Term</td>
<td>(\uparrow x - 2 \times y)</td>
</tr>
<tr>
<td>2</td>
<td>Expr + Term + Term + ... + Term</td>
<td>(\uparrow x - 2 \times y)</td>
</tr>
</tbody>
</table>

This doesn’t terminate

- Wrong choice of expansion leads to non-termination
- Non-termination is a bad property for a parser to have
- Parser must make the right choice
Top-down parsers cannot handle left-recursive grammars

Formally,

A grammar is left recursive if \( \exists A \in NT \) such that

\[ \exists \text{ a derivation } A \Rightarrow^+ A\alpha, \text{ for some string } \alpha \in (NT \cup T)^+ \]

Our expression grammar is left recursive

- This can lead to non-termination in a top-down parser
- For a top-down parser, any recursion must be right recursion
- We would like to convert the left recursion to right recursion

Non-termination is a bad property in any part of a compiler
To remove left recursion, we can transform the grammar

Consider a grammar fragment of the form

\[ Fee \rightarrow Fee \alpha \]
\[ \quad | \quad \beta \]

where neither \( \alpha \) nor \( \beta \) start with \( Fee \)

We can rewrite this as

\[ Fee \rightarrow \beta Fie \]
\[ Fie \rightarrow \alpha Fie \]
\[ \quad | \quad \varepsilon \]

where \( Fie \) is a new non-terminal

This accepts the same language, but uses only right recursion
The expression grammar contains two cases of left recursion

\[
\begin{align*}
\text{Expr} & \rightarrow \text{Expr} + \text{Term} & \text{Term} & \rightarrow \text{Term} \times \text{Factor} \\
& \mid \text{Expr} - \text{Term} & & \mid \text{Term} / \text{Factor} \\
& \mid \text{Term} & & \mid \text{Factor}
\end{align*}
\]

Applying the transformation yields

\[
\begin{align*}
\text{Expr} & \rightarrow \text{Term} \text{Expr'} \\
\text{Expr'} & \rightarrow + \text{Term} \text{Expr'} \\
& \mid - \text{Term} \text{Expr'} \\
& \mid \varepsilon
\end{align*}
\]

\[
\begin{align*}
\text{Term} & \rightarrow \text{Factor} \text{Term'} \\
\text{Term'} & \rightarrow * \text{Factor} \text{Term'} \\
& \mid / \text{Factor} \text{Term'} \\
& \mid \varepsilon
\end{align*}
\]

These fragments use only right recursion
Substituting them back into the grammar yields

- This grammar is correct, if somewhat non-intuitive.
- It is left associative, as was the original
- A top-down parser will terminate using it.
- A top-down parser may need to backtrack with it.
- General left recursion removal algorithm fig 3.6 EAC
We set out to study parsing

• Specifying syntax
  → Context-free grammars
  → Ambiguity

• Top-down parsers
  → Algorithm & its problem with left recursion
  → Left-recursion removal

• Predictive top-down parsing
  → The LL(1) condition
  → Table-driven LL(1) parsers
  → Recursive descent parsers
    ▪ Syntax directed translation (example)
Roadmap (Where are we?)

We set out to study parsing

• Specifying syntax
  → Context-free grammars
  → Ambiguity

• Top-down parsers
  → Algorithm & its problem with left recursion
  → Left-recursion removal

• Predictive top-down parsing
  → The LL(1) condition
  → Table-driven LL(1) parsers
  → Recursive descent parsers
    ▪ Syntax directed translation (example)
If it picks the wrong production, a top-down parser may backtrack

Alternative is to look ahead in input & use context to pick correctly

How much lookahead is needed?

• In general, an arbitrarily large amount
• Use the Cocke-Younger-Kasami algorithm or Earley’s algorithm

Fortunately,

• Large subclasses of CFGs can be parsed with limited lookahead
• Most programming language constructs fall in those subclasses

Among the interesting subclasses are $LL(1)$ and $LR(1)$ grammars
Basic idea

*Given* $A \rightarrow \alpha | \beta$, *the parser should be able to choose between* $\alpha$ *&* $\beta$

**FIRST sets**

For some rhs $\alpha \in G$, define $\text{FIRST}(\alpha)$ as the set of tokens that appear as the first (terminal) symbol in some string that derives from $\alpha$

That is, $a \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* a \gamma$, for some $\gamma$
The FIRST Set

\[ a \in \text{FIRST}(\alpha) \iff \alpha \Rightarrow^* a \gamma, \text{ for some } \gamma \]

To build \( \text{FIRST}(X) \) for all grammar symbols \( X \):

1. if \( X \) is a terminal (token), \( \text{FIRST}(X) := \{ X \} \)
2. if \( X ::= \epsilon \), then \( \epsilon \in \text{FIRST}(X) \)
3. iterate until no more terminals or \( \epsilon \) can be added to any \( \text{FIRST}(X) \):
   
   \[
   \text{if } X ::= Y_1 Y_2 \ldots Y_k \text{ then }
   \]
   \[
   a \in \text{FIRST}(X) \text{ if } a \in \text{FIRST}(Y_i) \text{ and } \\
   \epsilon \in \text{FIRST}(Y_j) \text{ for all } 1 \leq j < i \\
   \epsilon \in \text{FIRST}(X) \text{ if } \epsilon \in \text{FIRST}(Y_i) \text{ for all } 1 \leq i \leq k
   \]
   
   end iterate

Note: if \( \epsilon \not\in \text{FIRST}(Y_1) \), then \( \text{FIRST}(Y_i) \) is irrelevant, for \( 1 < i \)
Predictive Parsing

Basic idea

*Given* $A \rightarrow \alpha \mid \beta$, the parser should be able to choose between $\alpha$ & $\beta$

**FIRST** sets

For some rhs $\alpha \in G$, define $\text{FIRST}(\alpha)$ as the set of tokens that appear as the first symbol in some string that derives from $\alpha$

That is, $a \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* a \gamma$, for some $\gamma$

The LL(1) Property

If $A \rightarrow \alpha$ and $A \rightarrow \beta$ both appear in the grammar, we would like

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$$

This would allow the parser to make a correct choice with a lookahead of exactly one symbol!

Is this correct?
The FOLLOW Set

For a non-terminal $A$, define $\text{FOLLOW}(A)$ as

$$\text{FOLLOW}(A) := \text{the set of terminals that can appear immediately to the right of } A \text{ in some sentential form.}$$

Thus, a non-terminal’s FOLLOW set specifies the tokens that can legally appear after it; a terminal has no FOLLOW set.
The FOLLOW Set

To build FOLLOW(X) for all non-terminal X:

1. Place $eof$ in FOLLOW( $<goal>$ )
   iterate until no more terminals or $\varepsilon$ can be added to any FOLLOW(X):
2. If $A \rightarrow \alpha B \beta$ then
   put $\{FIRST(\beta) - \varepsilon\}$ in FOLLOW(B)
3. If $A \rightarrow \alpha B$ then
   put FOLLOW(A) in FOLLOW(B)
4. If $A \rightarrow \alpha B \beta$ and $\varepsilon \in$ FIRST(\beta) then
   put FOLLOW(A) in FOLLOW(B)
If $A \rightarrow \alpha$ and $A \rightarrow \beta$ and $\varepsilon \in \text{FIRST}(\alpha)$, then we need to ensure that $\text{FIRST}(\beta)$ is disjoint from $\text{FOLLOW}(A)$, too.

Define $\text{FIRST}^+(\delta)$ for rule $A \rightarrow \delta$ as:

- $(\text{FIRST}(\delta) - \{ \varepsilon \}) \cup \text{FOLLOW}(A)$, if $\varepsilon \in \text{FIRST}(\delta)$
- $\text{FIRST}(\delta)$, otherwise
More Syntax Analysis (bottom-up)

Read EaC: Chapter 3.4