This Lecture
- Parsing strategies: top-down and bottom-up.
- Shift-Reduce parsing theory.
- LR(0) parsing.
- LR(0), LR(k), and LALR(k) grammars

Top-Down: Leftmost Derivation
Consider the grammar:

\[ S \rightarrow aABe \quad A \rightarrow bcA \mid c \quad B \rightarrow d \]

Call sequence for predictive parser on \text{abcde}:

\[
\begin{align*}
\text{parse}S &\Rightarrow \text{lm} \quad aABe \\
\text{read a} &\Rightarrow \text{abcABe} \\
\text{parse}A &\Rightarrow \text{abccBe} \\
\text{read c} &\Rightarrow \text{abccde} \\
\text{parse}B &\Rightarrow \text{abccde} \\
\text{read d} &\Rightarrow \text{abccde} \\
\end{align*}
\]

Shift-Reduce Parsing
Shift-reduce parsing is a general style of bottom-up syntax analysis:
- Works from the leaves toward the root of the parse tree.
- Has two basic actions:
  - \text{Shift} (read) next terminal symbol.
  - \text{Reduce} a sequence of read terminals and previously reduced nonterminals corresponding to the RHS of a production to LHS nonterminal of that production.

Shift-Reduce Parsing: Idea
How can we know when and what to reduce???
Idea:
- Construct a DFA where each state is labelled by “all possibilities” given the input and reductions thus far. (Similar to how an NFA is turned into a DFA.)
- Whenever reduction is possible, if there is only one possible reduction, then it is always clear what to do.
Will make this more precise in the following.

LL, LR, and LALR parsing (1)
Three important classes of parsing methods:
- **LL(k):**
  - input scanned \text{Left to right}
  - \text{Leftmost derivation}
  - \(k\) symbols of lookahead
- **LR(k):**
  - input scanned \text{Left to right}
  - \text{Rightmost derivation in reverse}
  - \(k\) symbols of lookahead
- **LALR(k):** Look Ahead LR, simplified LR parsing

Parsing Strategies
There are two basic strategies for parsing:
- **Top-down parsing:**
  - Attempts to construct the parse tree \text{from the root downward.}
  - E.g. Recursive-Descent Parsing (see G52MAL).
- **Bottom-up parsing:**
  - Attempts to construct the parse tree \text{from the leaves working up toward the root.}
  - Traces out a \text{rightmost derivation in reverse}.

Bottom-Up: Rightmost Der. in Reverse
Consider (again) the grammar:

\[ S \rightarrow aABe \quad A \rightarrow bcA \mid c \quad B \rightarrow d \]

Reduction steps for the sentence \text{abcde} to \(S\):

\[
\begin{align*}
\text{abcdef} &\Rightarrow \text{abcde} &\Rightarrow &\text{abcde} \\
\text{abcA} &\Rightarrow &\text{abcA} &\Rightarrow &\text{abcA} \\
\text{aA} &\Rightarrow &\text{aA} &\Rightarrow &\text{aA} \\
\text{S} &\Rightarrow &\text{S} &\Rightarrow &\text{S} \\
\end{align*}
\]
Trace out rightmost derivation in reverse:

\[ S \Rightarrow aABe \Rightarrow aAde \Rightarrow abcAde \Rightarrow \text{abcdef} \]

How can we know when and what to reduce???

LL, LR, and LALR parsing (2)
By extension, the classes of grammars these methods can handle are also classified as LL(k), LR(k), and LALR(k).
Why study LR and LALR parsing?

- These methods handle a wide class of grammars of practical significance.
- In particular, handles left- and right-recursive grammars (but left rec. needs less stack).
- LALR is a good compromise between expressiveness and space cost of implementation.
- Consequently, many parser generator tools based on LALR.
- We will mainly study LR(0) parsing because it is the simplest, yet uses the same fundamental principles as LR(1) and LALR(1).

Shift-Reduce Parsing Theory (1)

Some terminology:
- An item for a CFG is a production with a dot anywhere in the RHS.
For example, the items for the grammar
\[ S \rightarrow aAc \quad A \rightarrow Ab | \epsilon \]
are:
- \[ S \rightarrow aAc \]
- \[ A \rightarrow Ab | \epsilon \]
- \[ S \rightarrow aA \cdot c \quad A \rightarrow Ab \cdot \]
- \[ S \rightarrow aAc \cdot A \rightarrow \cdot \]

Shift-Reduce Parsing Theory (2)

- Recap: Given a CFG \( G = (N, T, P, S) \), a string \( \phi \in (N \cup T)^* \) is a sentential form for \( G \) iff \( S \xrightarrow{r_m} \phi \).
- A right-sentential form is a sentential form that can be derived by a rightmost derivation.
- A handle of a right-sentential form \( \phi \) is a substring \( \alpha \) of \( \phi \) such that \( S \xrightarrow{r_m} \delta \cdot \alpha w \Rightarrow \delta \cdot \alpha w \) and \( \delta \cdot \alpha w = \phi \), where \( \alpha, \delta, \phi \in (N \cup T)^* \), and \( w \in T^* \).

Shift-Reduce Parsing Theory (3)

For example, consider the grammar:
\[ S \rightarrow aABc \quad A \rightarrow bcA \mid c \quad B \rightarrow d \]
The following is a rightmost derivation:
\[ S \Rightarrow aABc \Rightarrow aAdc \Rightarrow abcAdc \]
\( aABc, AAdc \) and \( abcAdc \) are right-sentential forms. Handle for each? \( aABc, A, \) and \( bcA \).
For an unambiguous grammar, the rightmost derivation is unique. Thus we can talk about “the handle” rather than merely “a handle”.

Shift-Reduce Parsing Theory (4)

- A viable prefix of a right-sentential form \( \phi \) is any prefix \( \gamma \) of \( \phi \) ending no farther right than the right end of the handle of \( \phi \).
- An item \( A \rightarrow \alpha \cdot \beta \) is complete valid if \( \gamma \) there is a rightmost derivation
\[ \begin{align*}
S &\xrightarrow{r_m} \delta Aw \\
\Rightarrow &\xrightarrow{r_m} \delta \alpha w
\end{align*} \]
and \( \delta = \gamma \).
- An item is complete if the the dot is the rightmost symbol in the item.

Shift-Reduce Parsing Theory (5)

Consider the grammar
\[ S \rightarrow aABc \quad A \rightarrow bcA \mid c \quad B \rightarrow d \]
and the rightmost derivation
\[ S \Rightarrow aABc \Rightarrow aAdc \Rightarrow abcAdc \]
The right-sentential form \( abcAdc \) has handle \( bcA \). Viable prefixes? \( \epsilon, a, ab, abc, aAc \).

Shift-Reduce Parsing Theory (6)

Last derivation step \( aAde \Rightarrow abcAdc \) by
production \( A \rightarrow bcA \), meaning the handle is \( bcA \).
Valid item for each non-\( c \) viable prefix of \( abcAdc \) considering this particular derivation only?

<table>
<thead>
<tr>
<th>Viable prefix</th>
<th>Valid item</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( A \rightarrow bcA )</td>
</tr>
<tr>
<td>( ab )</td>
<td>( A \rightarrow b \cdot cA )</td>
</tr>
<tr>
<td>( abc )</td>
<td>( A \rightarrow bc \cdot A )</td>
</tr>
<tr>
<td>( abcA )</td>
<td>( A \rightarrow bcA )</td>
</tr>
</tbody>
</table>

Any complete valid item?

Shift-Reduce Parsing Theory (7)

Knowing the valid items for a viable prefix allows a rightmost derivation in reverse to be found:
- If \( A \rightarrow \alpha \cdot \beta \) is a complete valid item for a viable prefix \( \gamma = \delta \alpha \cdot \beta \) of a right-sentential form \( \gamma w \) \( (w \in T^*) \), then it appears that \( A \rightarrow \alpha \) can be used at the last step, and that the previous right-sentential form is \( \delta Aw \).
- If this indeed always is the case for a CFG \( G \), then for any \( x \in L(G) \), since \( x \) is a right-sentential from, previous right-sentential forms can be determined until \( S \) is reached, giving a right-most derivation of \( x \).

Shift-Reduce Parsing Theory (8)

Of course, if \( A \rightarrow \alpha \cdot \beta \) is a complete valid item for a viable prefix \( \gamma = \delta \alpha \), in general, we only know it may be possible to use \( A \rightarrow \alpha \) to derive \( \gamma w \) from \( \delta Aw \). For example:
- \( A \rightarrow \alpha \cdot \) may be valid because of a different rightmost derivation \( S \Rightarrow \delta Aw' \Rightarrow \delta \alpha w' \).
- There could be two or more complete items valid for \( \gamma \).
- There could be a handle of \( \gamma w \) that includes symbols of \( w \).
A CFG for which knowing a complete valid item is enough to determine the previous right-sentential form is called \textit{LR(0)}.

For every CFG whatsoever, the set of viable prefixes is \textit{regular}.

Thus, an efficient parser can be developed for an LR(0) CFG based on a DFA for recognising viable prefixes and their valid items.

The states of the DFA are \textit{sets} of items valid for a recognised viable prefix.

A DFA recognising viable prefixes for the CFG
\[ S \rightarrow aABe \quad A \rightarrow cB \quad B \rightarrow d \]

\begin{itemize}
  \item Recall that item \( A \rightarrow bc \cdot A \) is a \textit{complete} valid item for the viable prefix \( abcA \). The corresponding DFA state indeed contains that item (and \textit{only} that item).
  \item In a state with a \textit{single complete item}: \textit{Reduce}
    \begin{itemize}
      \item The top of the parse stack contains the \textit{handle} of the current right-sentential form (since we have recognised a viable prefix for which a single \textit{complete} item is valid).
      \item The handle is just the \textit{RHS} of the valid item.
      \item Reduce to the previous right-sentential form by replacing the handle on the parse stack with the \textit{LHS} of the valid item.
      \item \textit{Move} to the state indicated by the new viable prefix on the parse stack.
    \end{itemize}
  \item If a state contains both complete and incomplete items, or if a state contains more than one complete item, then the grammar is \textit{not LR(0)}.
\end{itemize}

Given a DFA recognising viable prefixes, an LR(0) parser can be constructed as follows:
\begin{itemize}
  \item In a state \textit{without complete items}: \textit{Shift}
    \begin{itemize}
      \item Read next terminal symbol and push it onto an internal parse stack.
      \item Move to new state by following the edge labelled by the read terminal.
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item Recall that item \( A \rightarrow bcA \cdot \) is a complete valid item for the viable prefix \( abcA \).
  \item \textit{Reduce} to the previous right-sentential form by replacing the handle on the parse stack with the \textit{LHS} of the valid item.
  \item \textit{Move} to the state indicated by the new viable prefix on the parse stack.
\end{itemize}

\begin{itemize}
  \item If a state contains both complete and incomplete items, or if a state contains more than one complete item, then the grammar is \textit{not LR(0)}.
\end{itemize}

\begin{itemize}
  \item Recall that item \( A \rightarrow bcA \cdot \) is a complete valid item for the viable prefix \( abcA \).
  \item \textit{Reduce} to the previous right-sentential form by replacing the handle on the parse stack with the \textit{LHS} of the valid item.
  \item \textit{Move} to the state indicated by the new viable prefix on the parse stack.
\end{itemize}

\begin{itemize}
  \item If a state contains both complete and incomplete items, or if a state contains more than one complete item, then the grammar is \textit{not LR(0)}.
\end{itemize}

\begin{itemize}
  \item Recall that item \( A \rightarrow bcA \cdot \) is a complete valid item for the viable prefix \( abcA \).
  \item \textit{Reduce} to the previous right-sentential form by replacing the handle on the parse stack with the \textit{LHS} of the valid item.
  \item \textit{Move} to the state indicated by the new viable prefix on the parse stack.
\end{itemize}
### LR(0) Parsing (10)

<table>
<thead>
<tr>
<th>State</th>
<th>Stack (γ)</th>
<th>Input (w)</th>
<th>Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_0$</td>
<td>$\gamma$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_1$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_2$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_3$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_4$</td>
<td>$\alpha$</td>
<td>$\delta$</td>
<td>Reduce by $A \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$I_5$</td>
<td>$\alpha$</td>
<td>$\delta$</td>
<td>Reduce by $A \rightarrow \epsilon$</td>
</tr>
</tbody>
</table>

### LR(0) Parsing (11)

<table>
<thead>
<tr>
<th>State</th>
<th>Stack (γ)</th>
<th>Input (w)</th>
<th>Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_0$</td>
<td>$\gamma$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_1$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_2$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_3$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_4$</td>
<td>$\alpha$</td>
<td>$\delta$</td>
<td>Reduce by $A \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$I_5$</td>
<td>$\alpha$</td>
<td>$\delta$</td>
<td>Reduce by $A \rightarrow \epsilon$</td>
</tr>
</tbody>
</table>

### LR(0) Parsing (13)

Complete sequence ($\gamma w$ is right-sentential form):

<table>
<thead>
<tr>
<th>State</th>
<th>Stack (γ)</th>
<th>Input (w)</th>
<th>Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_0$</td>
<td>$\gamma$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_1$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_2$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_3$</td>
<td>$\alpha$</td>
<td>$\epsilon$</td>
<td>Shift</td>
</tr>
<tr>
<td>$I_4$</td>
<td>$\alpha$</td>
<td>$\delta$</td>
<td>Reduce by $A \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$I_5$</td>
<td>$\alpha$</td>
<td>$\delta$</td>
<td>Reduce by $A \rightarrow \epsilon$</td>
</tr>
</tbody>
</table>

Cf: $S \Rightarrow aABe \Rightarrow aAd \Rightarrow abcde \Rightarrow abcd e \Rightarrow abcd e \Rightarrow abcde$

### LR(0) Parsing (14)

Even more clear that the parser carries out the rightmost derivation in reverse if we look at the right-sentential forms $\gamma w$ of the reduction steps only:

$\gamma w = abcd e \Rightarrow abcd e \Rightarrow abcd e \Rightarrow abcd e \Rightarrow abcd e \Rightarrow abcd e \Rightarrow abcde \Rightarrow abcde$

### LR Parsing & Left/Right Recursion (1)

Remark: Note how the right-recursive production $A \rightarrow \delta$ causes symbols $bc$ to pile up on the parse stack until a reduction by

$A \rightarrow \epsilon$

can occur, in turn allowing the stacked symbols to be reduced away.

### LR Parsing & Left/Right Recursion (2)

Left-recursion allows reduction to happen sooner, thus keeping the size of the parse stack down. This is why left-recursive grammars are preferred for LR parsing.

### LR(1) Grammars (1)

- In practice, LR(0) tends to be a bit too restrictive.
- If we add one symbol of "lookahead" by determining the set of terminals that possibly could follow a handle being reduced by a production $A \rightarrow \beta$, then a wider class of grammars can be handled.
- Such grammars are called LR(1).

### LR(1) Grammars (2)

- Associate a lookahead set with items:

$A \rightarrow \alpha \cdot \beta, \{a_1, a_2, \ldots, a_n\}$

- On reduction, a complete item is only valid if the next input symbol belongs to its lookahead set.
- Thus it is OK to have two or more simultaneously valid complete items, as long as their lookahead sets are disjoint.
LR(\(k\)) Grammars

It is possible to have more than one symbol of lookahead:

- In general, a grammar that may be parsed with \(k\) symbols of lookahead is called LR(\(k\)).
- However, \(k > 1\) does not add to the class of languages that can be defined.

LALR Grammars

A problem with LR(\(k\)) parsers is that the DFAs become very large.

- \textit{LALR} ("lookahead LR") is a simplified construction that leads to much smaller DFAs.
- The basic idea is to reduce the number of states by merging sets of LR(1) items that are "similar".
- LALR places some additional constraints on a grammar, but those constraints are not too severe in practice. Most programming languages have LALR grammars.