These lectures:

- The problem of choice revisited.
- Predictive Parsing and LL(1) grammars.
- Computation of First and Follow Sets.
- Left factoring
Recap: Recursive-Descent Parsing

*Recursive-descent parsing* is an example of the top-down parsing method:
Recap: Recursive-Descent Parsing

*Recursive-descent parsing* is an example of the top-down parsing method:

- One *parsing function* associated with each nonterminal:

\[
\text{parseX :: [Token] -> Maybe [Token]}
\]
Recap: Recursive-Descent Parsing

*Recursive-descent parsing* is an example of the top-down parsing method:

- One *parsing function* associated with each nonterminal:
  
  \[
  \text{parseX :: [Token] \rightarrow Maybe [Token]}
  \]

- Each function tries to derive a prefix of the current input according to the productions for the nonterminal in question.
Recap: Recursive-Descent Parsing

Recursive-descent parsing is an example of the top-down parsing method:

- One *parsing function* associated with each nonterminal:

  \[
  \text{parseX} :: [\text{Token}] \rightarrow \text{Maybe} \ [\text{Token}]
  \]

- Each function tries to derive a prefix of the current input according to the productions for the nonterminal in question.

- Function for other nonterminals are invoked recursively as needed.
Recap: Handling Choice

We also need a way to handle *choice*, as in

\[ S \rightarrow AB \mid CD \]
Recap: Handling Choice

We also need a way to handle *choice*, as in

\[ S \rightarrow AB \mid CD \]

- Looking at the *next input symbol* is sometimes enough:

\[ S \rightarrow aB \mid cD \]
Recap: Handling Choice

We also need a way to handle *choice*, as in

\[ S \rightarrow AB \mid CD \]

- Looking at the *next input symbol* is sometimes enough:

\[ S \rightarrow aB \mid cD \]

- If not, *all alternatives* could be explored through *backtracking*:

\[
\text{parseX :: } [\text{Token}] \rightarrow [[[\text{Token}]]]
\]
Predictive Parsing (1)

Today, we are going to look into exactly when the next input symbol, a one symbol *lookahead*, can be used to make all parsing decisions.
Today, we are going to look into exactly when the next input symbol, a one symbol \textit{lookahead}, can be used to make \textit{all} parsing decisions.

We note that this can be the case even if the RHSs start with nonterminals:

\[
\begin{align*}
S & \rightarrow AB \mid CD \\
A & \rightarrow a \mid b \\
C & \rightarrow c \mid d
\end{align*}
\]
Predictive Parsing (2)

- **Predictive parsing** is an example of recursive descent parsing where no backtracking is needed.

- The grammar must be such that the next input symbol uniquely determines the next production to use.

**Productions:** \( X \rightarrow \alpha \mid \beta \)

\[
\text{parseX (t : ts)} = \\
\text{t ??} \quad \rightarrow \text{parse } \alpha \\
\text{t ??} \quad \rightarrow \text{parse } \beta \\
\text{otherwise } \quad \rightarrow \text{Nothing}
\]
Predictive Parsing (3)

How to make the choices? Idea:
Predictive Parsing (3)

How to make the choices? Idea:

- Compute the set of terminal symbols that can start strings derived from each alternative, the \textit{first set}.
How to make the choices? Idea:

- Compute the set of terminal symbols that can start strings derived from each alternative, the *first set*.
- If there is a choice between two or more alternatives, insist that the first sets for those are *disjoint*. 
How to make the choices? Idea:

- Compute the set of terminal symbols that can start strings derived from each alternative, the \textit{first set}.
- If there is a choice between two or more alternatives, insist that the first sets for those are \textit{disjoint}.
- The right choice can now be made simply by determining to which alternative’s first set the next input symbol belongs.
Predictive Parsing (4)

Productions: \( X \rightarrow \alpha \mid \beta \)

\[
\text{parse}_X (t : ts) = \begin{cases} 
  t \in \text{first}(\alpha) & \rightarrow \text{parse} \ \alpha \\
  t \in \text{first}(\beta) & \rightarrow \text{parse} \ \beta \\
  \text{otherwise} & \rightarrow \text{Nothing}
\end{cases}
\]
Again, consider: \( X \rightarrow \alpha | \beta \)

What if e.g. \( \beta \Rightarrow^* \epsilon \)?

Clearly, the next input symbol could be a terminal that can follow a string derivable form \( X \)!

\[
\text{parse}_X (t : ts) = \\
| t \in \text{first}(\alpha) \rightarrow \text{parse} \ \alpha \\
| t \in \text{first}(\beta) \cup \text{follow}(X) \rightarrow \text{parse} \ \beta \\
| \text{otherwise} \rightarrow \text{Nothing}
\]

The branches must be mutually exclusive!
First and Follow Sets (1)

Following (roughly) “the Dragon Book” [ASU86]

For a CFG \( G = (N, T, P, S) \):

\[
\text{first}(\alpha) = \{ a \in T \mid \alpha \xrightarrow{\ast}_G a\beta \}
\]

\[
\text{follow}(A) = \{ a \in T \mid S \xrightarrow{\ast}_G \alpha Aa\beta \}
\]

\[
\cup \{ \$ \mid S \xrightarrow{\ast}_G \alpha A \}
\]

where we assume \( \alpha, \beta \in (N \cup T)^\ast, A \in N \), and where \$ is a special “end of input” marker.
First and Follow Sets (2)

Consider:

\[ S \rightarrow ABC \]
\[ A \rightarrow aA \mid \epsilon \]
\[ B \rightarrow b \mid \epsilon \]
\[ C \rightarrow c \mid d \]

\[ \text{first}(C) = \{c, d\} \]
\[ \text{first}(B) = \{b\} \]
\[ \text{first}(A) = \{a\} \]
\[ \text{first}(S) = \text{first}(ABC) \]
\[ = [\text{because } A \Rightarrow^* \epsilon \text{ and } B \Rightarrow^* \epsilon] \]
\[ \text{first}(A) \cup \text{first}(B) \cup \text{first}(C) \]
\[ = \{a, b, c, d\} \]
First and Follow Sets (3)

Same grammar:

\[ S \rightarrow ABC \]
\[ A \rightarrow aA | \epsilon \]
\[ B \rightarrow b | \epsilon \]
\[ C \rightarrow c | d \]

Follow sets:

\[ \text{follow}(C) = \{\$\} \]
\[ \text{follow}(B) = \text{first}(C) = \{c, d\} \]
\[ \text{follow}(A) = [\text{because } B \Rightarrow^* \epsilon] \]
\[ \text{first}(B) \cup \text{first}(C) \]
\[ = \{b, c, d\} \]
Consider all productions for a nonterminal $A$ in some grammar:

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$
Consider all productions for a nonterminal $A$ in some grammar:

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

In the parsing function for $A$, on input symbol $t$, we parse according to $\alpha_i$ if $t \in \text{first}(\alpha_i)$. 
LL(1) Grammars (1)

Consider all productions for a nonterminal $A$ in some grammar:

$$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$$

In the parsing function for $A$, on input symbol $t$, we parse according to $\alpha_i$ if $t \in \text{first}(\alpha_i)$.

If $\alpha_i \Rightarrow^* \epsilon$, we should parse according to $\alpha_i$ also if $t \in \text{follow}(A)$!
Thus, if:
Thus, if:

- \( \text{first}(\alpha_i) \cap \text{first}(\alpha_j) = \emptyset \) for \( 1 \leq i < j \leq n \), and
Thus, if:

- \( \text{first}(\alpha_i) \cap \text{first}(\alpha_j) = \emptyset \) for \( 1 \leq i < j \leq n \), and

- if \( \alpha_i \Rightarrow^* \epsilon \) for some \( i \), then, for all \( 1 \leq j \leq n, j \neq i \).
Thus, if:

- \( \text{first}(\alpha_i) \cap \text{first}(\alpha_j) = \emptyset \) for \( 1 \leq i < j \leq n \), and
- if \( \alpha_i \xrightarrow{*} \epsilon \) for some \( i \), then, for all \( 1 \leq j \leq n, j \neq i \),
  - \( \alpha_j \xrightarrow{*} \epsilon \), and
Thus, if:

- \( \text{first}(\alpha_i) \cap \text{first}(\alpha_j) = \emptyset \) for \( 1 \leq i < j \leq n \), and
- if \( \alpha_i \Rightarrow^* \epsilon \) for some \( i \), then, for all \( 1 \leq j \leq n, j \neq i \),
  - \( \alpha_j \not\Rightarrow^* \epsilon \), and
  - \( \text{follow}(A) \cap \text{first}(\alpha_j) = \emptyset \)
Thus, if:

1. \( \text{first}(\alpha_i) \cap \text{first}(\alpha_j) = \emptyset \) for \( 1 \leq i < j \leq n \), and
2. if \( \alpha_i \Rightarrow^* \epsilon \) for some \( i \), then, for all \( 1 \leq j \leq n, j \neq i \),
   - \( \alpha_j \not\Rightarrow^* \epsilon \), and
   - \( \text{follow}(A) \cap \text{first}(\alpha_j) = \emptyset \)

then it is always clear what do do!
Thus, if:

- \( \text{first}(\alpha_i) \cap \text{first}(\alpha_j) = \emptyset \) for \( 1 \leq i < j \leq n \), and
- if \( \alpha_i \Rightarrow^* \epsilon \) for some \( i \), then, for all \( 1 \leq j \leq n, j \neq i \),
  - \( \alpha_j \not\Rightarrow^* \epsilon \), and
  - \( \text{follow}(A) \cap \text{first}(\alpha_j) = \emptyset \)

then it is always clear what to do!

A grammar satisfying these conditions is said to be an **LL(1)** grammar.
Nullable Nonterminals (1)

In order to compute the first and follow sets for a grammar $G = (N, T, P, S)$, we first need to know all nonterminals $A \in N$ such that $A \Rightarrow^* \epsilon$; i.e. the set $N_\epsilon \subseteq N$ of **nullable** nonterminals.

Let $\text{syms}(\alpha)$ denote the **set** of symbols in a string $\alpha$:

\[
\begin{align*}
\text{syms} & \in (N \cup T)^* \rightarrow \mathcal{P}(N \cup T) \\
\text{syms}(\epsilon) & = \emptyset \\
\text{syms}(X\alpha) & = \{X\} \cup \text{syms}(\alpha)
\end{align*}
\]
Nullable Nonterminals (2)

The set $N_\epsilon$ is the **smallest** solution to the equation

$$N_\epsilon = \{ A \mid A \rightarrow \alpha \in P \land \forall X \in \text{syms}(\alpha). X \in N_\epsilon \}$$

(Note that $A \in N_\epsilon$ if $A \rightarrow \epsilon \in P$.)

We can then define a predicate **nullable** on **strings** of grammar symbols:

- $\text{nullable} \in (N \cup T)^* \rightarrow \text{Bool}$
- $\text{nullable}(\epsilon) = \text{true}$
- $\text{nullable}(X\alpha) = X \in N_\epsilon \land \text{nullable}(\alpha)$
Nullable Nonterminals (3)

The equation for $N_\epsilon$ can be solved iteratively as follows:

1. Initialize $N_\epsilon$ to $\{A \mid A \rightarrow \epsilon \in P\}$.

2. If there is a production $A \rightarrow \alpha$ such that $\forall X \in \text{syms}(\alpha) \cdot X \in N_\epsilon$, then add $A$ to $N_\epsilon$.

3. Repeat step 2 until no further nullable nonterminals can be found.
Nullable Nonterminals (4)

Consider the following grammar:

\[
S \rightarrow ABC \mid AB \\
A \rightarrow aA \mid BB \\
B \rightarrow b \mid \epsilon \\
C \rightarrow c \mid d
\]
Nullable Nonterminals (4)

Consider the following grammar:

\[
\begin{align*}
S & \rightarrow ABC \mid AB \\
A & \rightarrow aA \mid BB \\
B & \rightarrow b \mid \epsilon \\
C & \rightarrow c \mid d
\end{align*}
\]

- Because \( B \rightarrow \epsilon \) is a production, \( B \in N_\epsilon \).
Nullable Nonterminals (4)

Consider the following grammar:

\[
S \rightarrow ABC \mid AB \\
A \rightarrow aA \mid BB \\
B \rightarrow b \mid \epsilon \\
C \rightarrow c \mid d
\]

- Because \( B \rightarrow \epsilon \) is a production, \( B \in N_\epsilon \).
- Because \( A \rightarrow BB \) is a production and \( B \in N_\epsilon \), additionally \( A \in N_\epsilon \).
Nullable Nonterminals (4)

Consider the following grammar:

\[
\begin{align*}
S & \rightarrow ABC \mid AB \\
A & \rightarrow aA \mid BB \\
B & \rightarrow b \mid \epsilon \\
C & \rightarrow c \mid d
\end{align*}
\]

- Because \( B \rightarrow \epsilon \) is a production, \( B \in N_\epsilon \).
- Because \( A \rightarrow BB \) is a production and \( B \in N_\epsilon \), additionally \( A \in N_\epsilon \).
- Because \( S \rightarrow AB \) is a production, and \( A, B \in N_\epsilon \), additionally \( S \in N_\epsilon \).
Consider the following grammar:

\[
\begin{align*}
S & \rightarrow ABC \mid AB \\
A & \rightarrow aA \mid BB \\
B & \rightarrow b \mid \epsilon \\
C & \rightarrow c \mid d
\end{align*}
\]

- Because \( B \rightarrow \epsilon \) is a production, \( B \in N_\epsilon \).
- Because \( A \rightarrow BB \) is a production and \( B \in N_\epsilon \), additionally \( A \in N_\epsilon \).
- Because \( S \rightarrow AB \) is a production, and \( A, B \in N_\epsilon \), additionally \( S \in N_\epsilon \).
- No more production with nullable RHS. The set of nullable symbols \( N_\epsilon = \{ S, A, B \} \).
Computing First Sets (1)

For a CFG $G = (N, T, P, S)$, the sets $\text{first}(A)$ for $A \in N$ are the smallest sets satisfying:

$$\text{first}(A) \subseteq T$$

$$\text{first}(A) = \bigcup_{A \rightarrow \alpha \in P} \text{first}(\alpha)$$
For strings, \( \text{first} \) is defined as (note the \textit{overloaded} notation):

\[
\begin{align*}
\text{first} & \in (N \cup T)^* \rightarrow \mathcal{P}(T) \\
\text{first}(\epsilon) & = \emptyset \\
\text{first}(a\alpha) & = \{a\} \\
\text{first}(A\alpha) & = \text{first}(A) \cup \begin{cases} 
\text{first}(\alpha), & \text{if } A \in N_\epsilon \\
\emptyset, & \text{if } A \notin N_\epsilon 
\end{cases}
\end{align*}
\]

where \( a \in T, A \in N, \) and \( \alpha \in (N \cup T)^* \).
Computing First Sets (3)

The solutions can often be obtained directly by expanding out all definitions.

If necessary, the equations can be solved by iteration in a similar way to how $N_\varepsilon$ is computed.

However, note that the smallest solution to set equations of the type

$$A = A \cup B$$

is simply

$$A = B$$
Computing First Sets (4)

Consider (again):

\[ S \rightarrow ABC \quad B \rightarrow b \mid \epsilon \]
\[ A \rightarrow aA \mid \epsilon \quad C \rightarrow c \mid d \]

First compute the nullable nonterminals:
\[ N_\epsilon = \{ A, B \} \]

Then compute first sets:

\[
\begin{align*}
\text{first}(A) &= \text{first}(aA) \cup \text{first}(\epsilon) \\
&= \{ a \} \cup \emptyset = \{ a \}
\end{align*}
\]
Computing First Sets (5)

\[
\begin{align*}
S & \rightarrow ABC \\
A & \rightarrow aA | \epsilon \\
B & \rightarrow b | \epsilon \\
C & \rightarrow c | d
\end{align*}
\]

\[
\begin{align*}
\text{first}(B) &= \text{first}(b) \cup \text{first}(\epsilon) \\
&= \{b\} \cup \emptyset = \{b\}
\end{align*}
\]

\[
\begin{align*}
\text{first}(C) &= \text{first}(c) \cup \text{first}(d) \\
&= \{c\} \cup \{d\} = \{c, d\}
\end{align*}
\]
Computing First Sets (6)

\[ S \rightarrow ABC \quad B \rightarrow b | \epsilon \]
\[ A \rightarrow aA | \epsilon \quad C \rightarrow c | d \]

\[
\text{first}(S') = \text{first}(ABC') \\
= [A \in N_\epsilon] \\
\text{first}(A) \cup \text{first}(BC') \\
= [B \in N_\epsilon \land C \notin N_\epsilon] \\
\text{first}(A) \cup \text{first}(B) \cup \text{first}(C') \cup \emptyset \\
= \{a\} \cup \{b\} \cup \{c, d\} = \{a, b, c, d\}
\]
Computing Follow Sets (1)

For a CFG $G = (N, T, P, S)$, the sets $\text{follow}(A)$ are the smallest sets satisfying:

- $\{\$\} \subseteq \text{follow}(S)$
- If $A \rightarrow \alpha B \beta \in P$, then $\text{first}(\beta) \subseteq \text{follow}(B)$
- If $A \rightarrow \alpha B \beta \in P$, and $\text{nullable}(\beta)$ then $\text{follow}(A) \subseteq \text{follow}(B)$

$A, B \in N$, and $\alpha, \beta \in (N \cup T)^*$.

(It is assumed that there are no *useless* symbols; i.e., all symbols can appear in the derivation of some sentence.)
Computing Follow Sets (2)

\[ S \rightarrow ABC \quad B \rightarrow b \mid \epsilon \]
\[ A \rightarrow aA \mid \epsilon \quad C \rightarrow c \mid d \]

Constraints for \( \text{follow}(S) \):

\[ \{\$\} \subseteq \text{follow}(S) \]
Computing Follow Sets (2)

\[
S \rightarrow ABC \\
A \rightarrow aA | \epsilon \\
B \rightarrow b | \epsilon \\
C \rightarrow c | d
\]

Constraints for \( \text{follow}(S) \):

\[
\{ \$ \} \subseteq \text{follow}(S)
\]

Constraints for \( \text{follow}(A) \) (note: \( \neg \text{nullable}(BC) \)):

\[
\text{first}(BC) \subseteq \text{follow}(A) \\
\text{first}(\epsilon) \subseteq \text{follow}(A) \\
\text{follow}(A) \subseteq \text{follow}(A)
\]
Computing Follow Sets (3)

\[ S \rightarrow ABC \quad B \rightarrow b \mid \epsilon \]
\[ A \rightarrow aA \mid \epsilon \quad C \rightarrow c \mid d \]

Constraints for \text{follow}(B) \text{ (note: } \neg \text{nullable}(C)\text{):}

\text{first}(C) \subseteq \text{follow}(B)
Computing Follow Sets (3)

\[ S \rightarrow ABC, \quad B \rightarrow b | \epsilon \]
\[ A \rightarrow aA | \epsilon, \quad C \rightarrow c | d \]

**Constraints for** \texttt{follow}(B) \textbf{(note: \neg \text{nullable}(C'))}:

\[ \text{first}(C') \subseteq \text{follow}(B) \]

**Constraints for** \texttt{follow}(C') \textbf{(note: \text{nullable}(\epsilon))}:

\[ \text{first}(\epsilon) \subseteq \text{follow}(C') \]
\[ \text{follow}(S') \subseteq \text{follow}(C') \]
Computing Follow Sets (4)

In general:

\[ A \subseteq C \land B \subseteq C \iff A \cup B \subseteq C \]

Also, constraints like \( A \subseteq A \) are trivially satisfied and can be omitted. The constraints can thus be written as:

\[
\{ \$ \} \subseteq \text{follow}(S') \\
\text{first}(BC') \cup \text{first}(\epsilon) \subseteq \text{follow}(A) \\
\text{first}(C') \subseteq \text{follow}(B) \\
\text{first}(\epsilon) \cup \text{follow}(S') \subseteq \text{follow}(C')
\]
Computing Follow Sets (5)

Using

\[
\begin{align*}
\text{first}(\epsilon) &= \emptyset \\
\text{first}(C) &= \{c, d\} \\
\text{first}(BC) &= \text{first}(B) \cup \text{first}(C) \cup \emptyset \\
&= \{b\} \cup \{c, d\} = \{b, c, d\}
\end{align*}
\]

the constraints can be simplified further:

\[
\begin{align*}
\{\$\} &\subseteq \text{follow}(S) \\
\{b, c, d\} &\subseteq \text{follow}(A) \\
\{c, d\} &\subseteq \text{follow}(B) \\
\text{follow}(S) &\subseteq \text{follow}(C)
\end{align*}
\]
Looking for the smallest sets satisfying these constraints, we get:

\[
\begin{align*}
\text{follow}(S) &= \{\$\} \\
\text{follow}(A) &= \{b, c, d\} \\
\text{follow}(B) &= \{c, d\} \\
\text{follow}(C) &= \text{follow}(S) = \{\$\}
\end{align*}
\]
No left-recursive or ambiguous grammar can be LL(1)!
No left-recursive or ambiguous grammar can be LL(1)! For example, consider:

\[ A \rightarrow Aa \mid \beta \]

First assume \( \text{first}(\beta) \neq \emptyset \).
No left-recursive or ambiguous grammar can be LL(1)! For example, consider:

\[ A \rightarrow Aa \mid \beta \]

First assume \( \text{first}(\beta) \neq \emptyset \).

Note that

- \( \text{first}(\beta) \subseteq \text{first}(A) \)
- \( \text{first}(A) \subseteq \text{first}(Aa) \)
  
  \((\text{first}(A) = \text{first}(Aa) \text{ if } A \not\Rightarrow \varepsilon)\)
No left-recursive or ambiguous grammar can be LL(1)! For example, consider:

\[ A \rightarrow Aa \mid \beta \]

First assume \( \text{first}(\beta) \neq \emptyset \).

Note that

- \( \text{first}(\beta) \subseteq \text{first}(A) \)
- \( \text{first}(A) \subseteq \text{first}(Aa) \)
  
  (\( \text{first}(A) = \text{first}(Aa) \) if \( A \not\Rightarrow \epsilon \))
- **Thus** \( \text{first}(Aa) \cap \text{first}(\beta) \neq \emptyset \). Not LL(1)!
Now assume $\text{first}(\beta) = \emptyset$
Now assume $\text{first}(\beta) = \emptyset$

This can only be the case if $\beta \Rightarrow^* \epsilon$ and nothing else.
Now assume $\text{first}(\beta) = \emptyset$

This can only be the case if $\beta \Rightarrow^* \epsilon$ and nothing else.

Assuming $S \Rightarrow^* \alpha A \gamma$, we note

- $a \in \text{first}(Aa)$ because $A \Rightarrow \beta \Rightarrow^* \epsilon$, and
- $a \in \text{follow}(A)$ because $A \Rightarrow^* \alpha A \gamma \Rightarrow \alpha Aa \gamma$
Now assume $\text{first}(\beta) = \emptyset$

This can only be the case if $\beta \Rightarrow^* \epsilon$ and nothing else.

Assuming $S \Rightarrow^* \alpha A \gamma$, we note

- $a \in \text{first}(Aa)$ because $A \Rightarrow \beta \Rightarrow^* \epsilon$, and
- $a \in \text{follow}(A)$ because $A \Rightarrow^* \alpha A \gamma \Rightarrow \alpha Aa \gamma$
- Because $\beta \Rightarrow^* \epsilon$, the LL(1) conditions require that $\text{first}(Aa)$ and $\text{follow}(A)$ be disjoint. But that is clearly not the case!
Left Factoring (1)

*Left factoring* means factoring out a common prefix among a group of productions. This can help making a grammar suitable for predictive recursive descent parsing.

Example:

\[ S' \rightarrow aXbY \mid aXbYcZ \]

Not suitable for predictive parsing!

But note common prefix! Let’s try to postpone the choice!
Before left factoring:

\[ S' \rightarrow aXbY \mid aXbYcZ \]

After left factoring:

\[ S' \rightarrow aXbYS' \]
\[ S' \rightarrow \epsilon \mid cZ \]

Now suitable for predictive parsing!