This lecture:

- 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:

- One parsing function associated with each nonterminal:
  
  ```haskell
  parseX :: [Token] -> Maybe [Token]
  ```

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

- Functions for other nonterminals are invoked recursively as needed.

Recap: Handling Choice

We also need a way to handle choice, as in

\[
S \rightarrow AB \mid CD
\]

- Looking at the next input symbol is sometimes enough:
  
  \[
  S \rightarrow aB \mid cD
  \]

- If not, all alternatives could be explored through backtracking:
  
  ```haskell
  parseX :: [Token] -> [[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.

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

\[
S \rightarrow AB \mid CD
\]

\[
A \rightarrow a \mid b
\]

\[
C \rightarrow c \mid d
\]

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:

- \( X \rightarrow \alpha \mid \beta \)
- \( parseX :: [Token] -> Maybe [Token] \)

\[
| t ?? -> parse \alpha
| t ?? -> parse \beta
| otherwise -> Nothing
\]

Predictive Parsing (3)

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.

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

- \( parseX :: [Token] \to Maybe [Token] \)

\[
| t \in \text{first}(\alpha) -> parse \alpha
| t \in \text{first}(\beta) -> parse \beta
| \text{otherwise} -> Nothing
\]

Predictive Parsing (5)

Again, consider: \( X \rightarrow \alpha \mid \beta \)

What if \( \beta \Rightarrow \epsilon \)?

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

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

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

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

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

\[
\text{first}(\alpha) = \{a \in T \mid \alpha \Rightarrow^* a\beta\}
\]

\[
\text{follow}(A) = \{a \in T \mid S \Rightarrow^* a\alpha\beta\}
\]

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

### First and Follow Sets (2)

Consider:

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

\[
\text{first}(C) = \{c, d\}
\]

\[
\text{first}(B) = \{b\}
\]

\[
\text{first}(A) = \{a\}
\]

\[
\text{first}(S) = \text{first}(ABC) = \{a, b, c, d\}
\]

\[
\text{follow}(A) = \{c, d\}
\]

\[
\text{follow}(B) = \{$$\}
\]

\[
\text{follow}(C) = \{c\}
\]

\[
\text{follow}(S) = \text{first}(ABC) = \{a, b, c, d\}
\]

### Nullable Nonterminals (1)

The set $N_e$ is the smallest solution to the equation

\[
N_e = \{A \mid A \to \alpha \in P \land \forall X \in \text{sym}(\alpha) \cdot X \in N_e\}
\]

(Where $A \in N$, if $A \to \epsilon \in P$ because $\text{sym}(\epsilon) = \emptyset$ and $\forall X \in \emptyset \cdot \ldots$ is trivially true.)

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

\[
\begin{align*}
n\text{nullable} &\in (N \cup T)^* \to \text{Bool} \\
n\text{nullable}(\epsilon) &= \text{true} \\
n\text{nullable}(X\alpha) &= X \in N_e \land n\text{nullable}(\alpha)
\end{align*}
\]

### Nullable Nonterminals (2)

The equation for $N_e$ can be solved iteratively as follows:

1. Initialize $N_0$ to $\{A \mid A \to \epsilon \in P\}$.
2. If there is a production $A \to \alpha$ such that $\forall X \in \text{sym}(\alpha) \cdot X \in N_e$, then add $A$ to $N_e$.
3. Repeat step 2 until no further nullable nonterminals can be found.

### Nullable Nonterminals (3)

Consider the following grammar:

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

- Because $B \to b$ is a production, $B \in N_e$.
- Because $A \to BB$ is a production and $B \in N_e$, additionally $A \in N_e$.
- Because $S \to AB$ is a production, and $A, B \in N_e$, additionally $S \in N_e$.
- No more production with nullable RHS. The set of nullable symbols $N_e = \{S, A, B\}$. 

### Nullable Nonterminals (4)

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 $N_e \subseteq N$ of nullable nonterminals.

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

\[
\begin{align*}
\text{sym}\in (N \cup T)^* &\to \mathcal{P}(N \cup T) \\
\text{sym}(\epsilon) &= \emptyset \\
\text{sym}(X\alpha) &= \{X\} \cup \text{sym}(\alpha)
\end{align*}
\]
**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:

- $\text{first}(A) \subseteq T$
- $\text{first}(A) = \bigcup_{A \rightarrow \alpha \in P} \text{first}(\alpha)$

**Computing First Sets (2)**

For strings, first is defined as (note the overloaded notation):

- $\text{first}(A) \subseteq (N \cup T)^*$
- $\text{first}(\epsilon) = \emptyset$
- $\text{first}(aA) = \{a\}$
- $\text{first}(A\alpha) = \text{first}(A) \cup \{\text{first(\alpha)} | A \in N_i\}$

where $\alpha \in T$, $A \in N$, 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 $N_i$ 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):

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

First compute the nullable nonterminals: $N_i = \{A, B\}$.

Then compute first sets:

- $\text{first}(S) = \text{first}(A) \cup \text{first}(B) \cup \text{first}(C)$
- $\text{first}(A) = \{a\}$

**Computing First Sets (5)**

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

- $\text{first}(A) \subseteq T$
- $\text{first}(A) = \bigcup_{A \rightarrow \alpha \in P} \text{first}(\alpha)$

**Computing First Sets (6)**

For strings, first is defined as (note the overloaded notation):

- $\text{first}(A) \subseteq (N \cup T)^*$
- $\text{first}(\epsilon) = \emptyset$
- $\text{first}(aA) = \{a\}$
- $\text{first}(A\alpha) = \text{first}(A) \cup \{\text{first(\alpha)} | A \in N_i\}$

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

**Computing Follow Sets (1)**

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

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

**Computing Follow Sets (2)**

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

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

**Computing Follow Sets (3)**

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

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

**Computing Follow Sets (4)**

Constraints for follow($S$):

- $\{\}$ $\subseteq$ follow($S$)

Constraints for follow($A$) (note: nullable($B\alpha$)):

- $\text{first}(B\alpha) \subseteq$ follow($A$)
- $\text{first}(\epsilon) \subseteq$ follow($A$)
- follow($A$) $\subseteq$ follow($A$)
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:
\[
\begin{align*}
\{\$\} &\subseteq \text{follow}(S) \\
\{b, c, d\} &\subseteq \text{follow}(A) \\
\{c, d\} &\subseteq \text{follow}(B) \\
\text{follow}(S) &\subseteq \text{follow}(C)
\end{align*}
\]

Computing Follow Sets (6)

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

LL(1), Left-Recursion, Ambiguity (1)

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

Note that
\[
\begin{align*}
\text{first}(\beta) &\subseteq \text{first}(A) \\
\text{first}(A) &\subseteq \text{first}(Aa) \\
\text{first}(Aa) &\text{ if } A \not\Rightarrow \epsilon
\end{align*}
\]

Thus \(\text{first}(Aa) \cap \text{first}(\beta) \neq \emptyset\). Not LL(1)!

LL(1), Left-Recursion, Ambiguity (2)

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
\[
\begin{align*}
&\quad a \in \text{first}(Aa) \text{ because } A \Rightarrow \epsilon, \text{ and} \\
&\quad a \in \text{follow}(A) \text{ because } S \Rightarrow \alpha A \gamma \Rightarrow \alpha Aa \gamma
\end{align*}
\]

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:
\[
S \rightarrow aXbY \mid aXbY cZ
\]

Not suitable for predictive parsing!

But note common prefix! Let’s try to postpone the choice!

Left Factoring (2)

Before left factoring:
\[
S \rightarrow aXbY \mid aXbY cZ
\]

After left factoring:
\[
S \rightarrow aXbY S' \\
S' \rightarrow \epsilon \mid cZ
\]

Now suitable for predictive parsing!