G52MAL
Machines and Their Langauges
Lecture 17
Recursive-Descent Parsing: Predictive Parsing

Henrik Nilsson

University of Nottingham, UK

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 (1)

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

Recap: Recursive-Descent Parsing (1)

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

- One ***parsing function*** associated with each nonterminal; e.g., for nonterminal X, `parseX`:

```
parseX :: [Token] -> Maybe [Token]
```

Recap: Recursive-Descent Parsing (1)

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

- One ***parsing function*** associated with each nonterminal; e.g., for nonterminal X, `parseX`:

  ```
  parseX :: [Token] -> Maybe [Token]
  ```

- A parsing function ***attempts*** to derive a prefix of the current input according to the grammar starting from the nonterminal.

Recap: Recursive-Descent Parsing (1)

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

- One ***parsing function*** associated with each nonterminal; e.g., for nonterminal X, `parseX`:

```
parseX :: [Token] -> Maybe [Token]
```

- A parsing function ***attempts*** to derive a prefix of the current input according to the grammar starting from the nonterminal.
- Other parsing functions invoked (recursively) as needed according to the RHS of the production(s) for the nonterminal.

Recap: Recursive-Descent Parsing (2)

- If successful, a parsing function returns the remainder of the input.

Recap: Recursive-Descent Parsing (2)

- If successful, a parsing function returns the remainder of the input.

  E.g. if input is $\alpha\beta$, $X \stackrel{*}{\Rightarrow} \alpha$, and `parseX` could carry out this derivation, then:

  $$\texttt{parseX}\,\alpha\beta = \texttt{Just}\;\beta$$

Recap: Recursive-Descent Parsing (2)

- If successful, a parsing function returns the remainder of the input.

  E.g. if input is $\alpha\beta$, $X \stackrel{*}{\Rightarrow} \alpha$, and `parseX` could carry out this derivation, then:

  $$\texttt{parseX}\,\alpha\beta = \texttt{Just}\ \beta$$

- If unsuccessful, a parsing function returns `Nothing`.

Recap: Handling Choice (1)

Of course, we want a parsing function to be successful ***exactly*** when a prefix of the input ***can*** be derived from the corresponding nonterminal.

Recap: Handling Choice (1)

Of course, we want a parsing function to be successful ***exactly*** when a prefix of the input ***can*** be derived from the corresponding nonterminal.

This can be achieved by:

Recap: Handling Choice (1)

Of course, we want a parsing function to be successful ***exactly*** when a prefix of the input ***can*** be derived from the corresponding nonterminal.

This can be achieved by:

- Adopting a suitable parsing strategy, specifically regarding how to handle ***choice*** between two or more productions for one nonterminal.

Recap: Handling Choice (1)

Of course, we want a parsing function to be successful ***exactly*** when a prefix of the input ***can*** be derived from the corresponding nonterminal.

This can be achieved by:

- Adopting a suitable parsing strategy, specifically regarding how to handle ***choice*** between two or more productions for one nonterminal.
- Impose ***restrictions*** on the grammar to ensure success of the chosen parsing strategy.

Recap: Handling Choice (1)

Of course, we want a parsing function to be successful ***exactly*** when a prefix of the input ***can*** be derived from the corresponding nonterminal.

This can be achieved by:

- Adopting a suitable parsing strategy, specifically regarding how to handle ***choice*** between two or more productions for one nonterminal.
- Impose ***restrictions*** on the grammar to ensure success of the chosen parsing strategy.

In particular, ***left recursion*** usually ***not allowed***.

Recap: Handling Choice (2)

Two strategies for handling ***choice***, as in

$$S \rightarrow AB \mid CD$$

Recap: Handling Choice (2)

Two strategies for handling ***choice***, as in

$$S \to AB \mid CD$$

- Looking at the ***next input symbol*** is sometimes enough; e.g.:

$$S \to aB \mid cD$$

Recap: Handling Choice (2)

Two strategies for handling ***choice***, as in

$$S \to AB \mid CD$$

- Looking at the ***next input symbol*** is sometimes enough; e.g.:

$$S \to aB \mid cD$$

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

```
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.

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:

$$
\begin{aligned}
S &\rightarrow AB \mid CD \\
A &\rightarrow a \mid b \\
C &\rightarrow c \mid d
\end{aligned}
$$

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 (a grammar restriction).

Productions: $X \rightarrow \alpha \mid \beta$

```
parseX (t : ts) =
      | t ??        -> parse α
      | t ??        -> parse β
      | otherwise -> 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 ***first set***.

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*** (a grammar restriction).

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*** (a grammar restriction).
- 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 (t : ts) =
      | t ∈ first(α) -> parse α
      | t ∈ first(β) -> parse β
      | otherwise    -> Nothing
```

Predictive Parsing (5)

Again, consider: $X \rightarrow \alpha \mid \beta$

What if e.g. $\beta \stackrel{*}{\Rightarrow} \epsilon$?

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

```
parseX (t : ts) =
    | t ∈ first(α)              -> parse α
    | t ∈ first(β) ∪ follow(X) -> parse β
    | otherwise  -> Nothing
```

The branches must be mutually exclusive!

First and Follow Sets (1)

Following (roughly) "the Dragon Book" [ASU86]

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

$$\begin{aligned}
\text{first}(\alpha) &= \{a \in T \mid \alpha \overset{*}{\underset{G}{\Rightarrow}} a\beta\} \\
\text{follow}(A) &= \{a \in T \mid S \overset{*}{\underset{G}{\Rightarrow}} \alpha A a \beta\} \\
&\quad \cup \{\$ \mid S \overset{*}{\underset{G}{\Rightarrow}} \alpha A\}
\end{aligned}$$

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:

$$\begin{array}{rclcrcl} S & \to & ABC & \quad & B & \to & b \mid \epsilon \\ A & \to & aA \mid \epsilon & & C & \to & c \mid d \end{array}$$

$$\begin{aligned} \text{first}(C) &= \{c,\ d\} \\ \text{first}(B) &= \{b\} \\ \text{first}(A) &= \{a\} \\ \text{first}(S) &= \text{first}(ABC) \\ &= [\text{because } A \stackrel{*}{\Rightarrow} \epsilon \text{ and } B \stackrel{*}{\Rightarrow} \epsilon] \\ &\ \text{first}(A) \cup \text{first}(B) \cup \text{first}(C) \\ &= \{a,\ b,\ c,\ d\} \end{aligned}$$

First and Follow Sets (3)

Same grammar:

$$\begin{array}{rclcrcl} S & \rightarrow & ABC & & B & \rightarrow & b \mid \epsilon \\ A & \rightarrow & aA \mid \epsilon & & C & \rightarrow & c \mid d \end{array}$$

Follow sets:

$$\begin{array}{rcl} \text{follow}(C) & = & \{\$\} \\ \text{follow}(B) & = & \text{first}(C) = \{c,\ d\} \\ \text{follow}(A) & = & [\text{because } B \stackrel{*}{\Rightarrow} \epsilon] \\ & & \text{first}(B) \cup \text{first}(C) \\ & = & \{b,\ c,\ d\} \end{array}$$

LL(1) Grammars (1)

Consider all productions for a nonterminal A in some grammar:

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

LL(1) Grammars (1)

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 α_i if $t \in \mathrm{first}(\alpha_i)$.

LL(1) Grammars (1)

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 α_i if $t \in \mathrm{first}(\alpha_i)$.

If $\alpha_i \stackrel{*}{\Rightarrow} \epsilon$, we should parse according to α_i also if $t \in \mathrm{follow}(A)$!

LL(1) Grammars (2)

Thus, if:

LL(1) Grammars (2)

Thus, if:

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

LL(1) Grammars (2)

Thus, if:

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

LL(1) Grammars (2)

Thus, if:

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

LL(1) Grammars (2)

Thus, if:

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

LL(1) Grammars (2)

Thus, if:

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

then it is always clear what do do!

LL(1) Grammars (2)

Thus, if:

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

then it is always clear what do 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 $A \stackrel{*}{\Rightarrow} \epsilon$; i.e. the set $N_\epsilon \subseteq N$ of ***nullable*** nonterminals.

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

$$\begin{aligned} \mathrm{syms} &\in (N \cup T)^* \to \mathcal{P}(N \cup T) \\ \mathrm{syms}(\epsilon) &= \emptyset \\ \mathrm{syms}(X\alpha) &= \{X\} \cup \mathrm{syms}(\alpha) \end{aligned}$$

Nullable Nonterminals (2)

The set N_ϵ is the ***smallest*** solution to the equation

$$N_\epsilon \;=\; \{A \mid A \to \alpha \in P \;\wedge\; \forall X \in \mathrm{syms}(\alpha)\,.\, X \in N_\epsilon\}$$

(Note that $A \in N_\epsilon$ if $A \to \epsilon \in P$ because $\mathrm{syms}(\epsilon) = \emptyset$ and $\forall X \in \emptyset\,.\;\ldots$ is trivially true.)

We can now define a predicate $\mathrm{nullable}$ on ***strings*** of grammar symbols:

$$\begin{aligned} \mathrm{nullable} &\in (N \cup T)^* \to \mathrm{Bool} \\ \mathrm{nullable}(\epsilon) &= \mathrm{true} \\ \mathrm{nullable}(X\alpha) &= X \in N_\epsilon \wedge \mathrm{nullable}(\alpha) \end{aligned}$$

Nullable Nonterminals (3)

The equation for N_ϵ can be solved iteratively as follows:

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

Nullable Nonterminals (4)

Consider the following grammar:

$$
\begin{array}{lll}
S & \rightarrow & ABC \mid AB \\
A & \rightarrow & aA \mid BB
\end{array}
\qquad
\begin{array}{lll}
B & \rightarrow & b \mid \epsilon \\
C & \rightarrow & c \mid d
\end{array}
$$

Nullable Nonterminals (4)

Consider the following grammar:

$$\begin{array}{rclcrcl} S & \to & ABC \mid AB & \quad & B & \to & b \mid \epsilon \\ A & \to & aA \mid BB & \quad & C & \to & c \mid d \end{array}$$

- Because $B \to \epsilon$ is a production, $B \in N_\epsilon$.

Nullable Nonterminals (4)

Consider the following grammar:

$$\begin{array}{rclcrcl} S & \to & ABC \mid AB & \quad & B & \to & b \mid \epsilon \\ A & \to & aA \mid BB & & C & \to & c \mid d \end{array}$$

- Because $B \to \epsilon$ is a production, $B \in N_\epsilon$.
- Because $A \to BB$ is a production and $B \in N_\epsilon$, additionally $A \in N_\epsilon$.

Nullable Nonterminals (4)

Consider the following grammar:

$$
\begin{array}{lllllll}
S & \to & ABC \mid AB & \quad & B & \to & b \mid \epsilon \\
A & \to & aA \mid BB & \quad & C & \to & c \mid d
\end{array}
$$

- Because $B \to \epsilon$ is a production, $B \in N_\epsilon$.
- Because $A \to BB$ is a production and $B \in N_\epsilon$, additionally $A \in N_\epsilon$.
- Because $S \to AB$ is a production, and $A, B \in N_\epsilon$, additionally $S \in N_\epsilon$.

Nullable Nonterminals (4)

Consider the following grammar:

$$
\begin{array}{rclcrcl}
S & \to & ABC \mid AB & \quad & B & \to & b \mid \epsilon \\
A & \to & aA \mid BB & \quad & C & \to & c \mid d
\end{array}
$$

- Because $B \to \epsilon$ is a production, $B \in N_\epsilon$.
- Because $A \to BB$ is a production and $B \in N_\epsilon$, additionally $A \in N_\epsilon$.
- Because $S \to AB$ 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 $\mathrm{first}(A)$ for $A \in N$ are the smallest sets satisfying:

$$\begin{aligned} \mathrm{first}(A) &\subseteq T \\ \mathrm{first}(A) &= \bigcup_{A \to \alpha \in P} \mathrm{first}(\alpha) \end{aligned}$$

Computing First Sets (2)

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

$$
\begin{aligned}
\mathrm{first} &\in (N \cup T)^* \to \mathcal{P}(T) \\
\mathrm{first}(\epsilon) &= \emptyset \\
\mathrm{first}(a\alpha) &= \{a\} \\
\mathrm{first}(A\alpha) &= \mathrm{first}(A) \cup \begin{cases} \mathrm{first}(\alpha), & \text{if } A \in N_\epsilon \\ \emptyset, & \text{if } A \notin N_\epsilon \end{cases}
\end{aligned}
$$

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_ϵ is computed.

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):

$$\begin{array}{rclcrcl} S & \rightarrow & ABC & \qquad & B & \rightarrow & b \mid \epsilon \\ A & \rightarrow & aA \mid \epsilon & & C & \rightarrow & c \mid d \end{array}$$

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

Then compute first sets:

$$\begin{aligned} \mathrm{first}(A) &= \mathrm{first}(aA) \cup \mathrm{first}(\epsilon) \\ &= \{a\} \cup \emptyset = \{a\} \end{aligned}$$

Computing First Sets (5)

$$
\begin{aligned}
S &\rightarrow ABC & \qquad B &\rightarrow b \mid \epsilon \\
A &\rightarrow aA \mid \epsilon & \qquad C &\rightarrow c \mid d
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{first}(B) &= \mathrm{first}(b) \cup \mathrm{first}(\epsilon) \\
&= \{b\} \cup \emptyset = \{b\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{first}(C) &= \mathrm{first}(c) \cup \mathrm{first}(d) \\
&= \{c\} \cup \{d\} = \{c, d\}
\end{aligned}
$$

Computing First Sets (6)

$$\begin{array}{lll} S \rightarrow ABC & & B \rightarrow b \mid \epsilon \\ A \rightarrow aA \mid \epsilon & & C \rightarrow c \mid d \end{array}$$

$$\begin{aligned} \text{first}(S) &= \text{first}(ABC) \\ &= [A \in N_\epsilon] \\ &\quad \text{first}(A) \cup \text{first}(BC) \\ &= [B \in N_\epsilon \wedge C \notin N_\epsilon] \\ &\quad \text{first}(A) \cup \text{first}(B) \cup \text{first}(C) \cup \emptyset \\ &= \{a\} \cup \{b\} \cup \{c, d\} = \{a, b, c, d\} \end{aligned}$$

Computing Follow Sets (1)

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

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

$A, B \in N$, 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)

$$
\begin{array}{rclcrcl}
S & \to & ABC & \qquad & B & \to & b \mid \epsilon \\
A & \to & aA \mid \epsilon & & C & \to & c \mid d
\end{array}
$$

Constraints for $\mathrm{follow}(S)$:

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

Computing Follow Sets (2)

$$\begin{array}{rcl} S & \to & ABC \\ A & \to & aA \mid \epsilon \end{array} \qquad \begin{array}{rcl} B & \to & b \mid \epsilon \\ C & \to & c \mid d \end{array}$$

Constraints for $\mathrm{follow}(S)$:

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

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

$$\begin{array}{rcl} \mathrm{first}(BC) & \subseteq & \mathrm{follow}(A) \\ \mathrm{first}(\epsilon) & \subseteq & \mathrm{follow}(A) \\ \mathrm{follow}(A) & \subseteq & \mathrm{follow}(A) \end{array}$$

Computing Follow Sets (3)

$$S \rightarrow ABC \qquad B \rightarrow b \mid \epsilon$$
$$A \rightarrow aA \mid \epsilon \qquad C \rightarrow c \mid d$$

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

$$\mathrm{first}(C) \subseteq \mathrm{follow}(B)$$

Computing Follow Sets (3)

$$S \rightarrow ABC \qquad B \rightarrow b \mid \epsilon$$
$$A \rightarrow aA \mid \epsilon \qquad C \rightarrow c \mid d$$

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

$$\text{first}(C) \subseteq \text{follow}(B)$$

Constraints for $\text{follow}(C)$ (note: $\text{nullable}(\epsilon)$):

$$\text{first}(\epsilon) \subseteq \text{follow}(C)$$
$$\text{follow}(S) \subseteq \text{follow}(C)$$

Computing Follow Sets (4)

In general:

$$A \subseteq C \;\wedge\; B \subseteq C \quad\Longleftrightarrow\quad A \cup B \subseteq C$$

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

$$\begin{aligned}
\{\$\} &\subseteq \mathrm{follow}(S)\\
\mathrm{first}(BC) \cup \mathrm{first}(\epsilon) &\subseteq \mathrm{follow}(A)\\
\mathrm{first}(C) &\subseteq \mathrm{follow}(B)\\
\mathrm{first}(\epsilon) \cup \mathrm{follow}(S) &\subseteq \mathrm{follow}(C)
\end{aligned}$$

Computing Follow Sets (5)

Using

$$\begin{aligned} \text{first}(\epsilon) &= \emptyset \\ \text{first}(C) &= \{c, d\} \\ \text{first}(BC) &= \text{first}(B) \cup \text{first}(C) \cup \emptyset \\ &= \{b\} \cup \{c, d\} = \{b, c, d\} \end{aligned}$$

the constraints can be simplified further:

$$\begin{aligned} \{\$\} &\subseteq \text{follow}(S) \\ \{b, c, d\} &\subseteq \text{follow}(A) \\ \{c, d\} &\subseteq \text{follow}(B) \\ \text{follow}(S) &\subseteq \text{follow}(C) \end{aligned}$$

Computing Follow Sets (6)

Looking for the smallest sets satisfying these constraints, we get:

$$\begin{aligned}
\text{follow}(S) &= \{\$\} \\
\text{follow}(A) &= \{b, c, d\} \\
\text{follow}(B) &= \{c, d\} \\
\text{follow}(C) &= \text{follow}(S) = \{\$\}
\end{aligned}$$

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

No left-recursive or ambiguous grammar can be LL(1)!

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

No left-recursive or ambiguous grammar can be LL(1)! For example, consider:

$$A \to Aa \mid \beta$$

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

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

No left-recursive or ambiguous grammar can be LL(1)! For example, consider:

$$A \to Aa \mid \beta$$

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

Note that

- $\mathrm{first}(\beta) \subseteq \mathrm{first}(A)$
- $\mathrm{first}(A) \subseteq \mathrm{first}(Aa)$
  ($\mathrm{first}(A) = \mathrm{first}(Aa)$ if $A \not\overset{*}{\Rightarrow} \epsilon$)

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 $\mathrm{first}(\beta) \neq \emptyset$.

Note that

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

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

Now assume $\text{first}(\beta) = \emptyset$

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

Now assume $\mathrm{first}(\beta) = \emptyset$

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

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

Now assume $\mathrm{first}(\beta) = \emptyset$

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

Assuming $S \stackrel{*}{\Rightarrow} \alpha A \gamma$, we note

- $a \in \mathrm{first}(Aa)$ because $A \Rightarrow \beta \stackrel{*}{\Rightarrow} \epsilon$, and
- $a \in \mathrm{follow}(A)$ because $S \stackrel{*}{\Rightarrow} \alpha A \gamma \Rightarrow \alpha A a \gamma$

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

Now assume $\text{first}(\beta) = \emptyset$

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

Assuming $S \stackrel{*}{\Rightarrow} \alpha A \gamma$, we note

- $a \in \text{first}(Aa)$ because $A \Rightarrow \beta \stackrel{*}{\Rightarrow} \epsilon$, and
- $a \in \text{follow}(A)$ because $S \stackrel{*}{\Rightarrow} \alpha A \gamma \Rightarrow \alpha A a \gamma$
- Because $\beta \stackrel{*}{\Rightarrow} \epsilon$, 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!

Left Factoring (2)

Before left factoring:

$$S \rightarrow aXbY \mid aXbYcZ$$

After left factoring:

$$\begin{aligned} S &\rightarrow aXbYS' \\ S' &\rightarrow \epsilon \mid cZ \end{aligned}$$

Now suitable for predictive parsing!