

COMP2012/G52LAC  
Languages and Computation  
Lecture 14

*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  $X$ ,  $\text{parse}_X$ :  
 $\text{parse}_X :: [\text{Token}] \rightarrow \text{Maybe } [\text{Token}]$

# 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  $X$ ,  $\text{parse}_X$ :  
 $\text{parse}_X :: [\text{Token}] \rightarrow \text{Maybe } [\text{Token}]$
- 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  $X$ ,  $\text{parse}_X$ :  
$$\text{parse}_X :: [\text{Token}] \rightarrow \text{Maybe} [\text{Token}]$$
- 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  $\alpha\beta$ ,  $X \xRightarrow{*} \alpha$ , and `parseX` could carry out this derivation, then:

$$\text{parseX } \alpha\beta = \text{Just } \beta$$



# Recap: Recursive-Descent Parsing (2)

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

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

$$\text{parseX } \alpha\beta = \text{Just } \beta$$

- 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

$$S \rightarrow AB \mid CD$$

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

$$S \rightarrow aB \mid cD$$



# Recap: Handling Choice (2)

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

$$S \rightarrow AB \mid CD$$

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

$$S \rightarrow aB \mid cD$$

- 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:

$$S \rightarrow AB \mid CD$$

$$A \rightarrow a \mid b$$

$$C \rightarrow c \mid d$$

# 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 (a grammar restriction).

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

```
parseX (t : ts) =  
  | t ??      -> parse  $\alpha$   
  | t ??      -> parse  $\beta$   
  | 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( $\alpha$ ) -> parse  $\alpha$`

`| t ∈ first( $\beta$ ) -> parse  $\beta$`

`| otherwise -> Nothing`

# Predictive Parsing (5)

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

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

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

$$\begin{aligned} \text{parse}_X (t : ts) = & \\ & \mid t \in \text{first}(\alpha) \quad \rightarrow \text{parse } \alpha \\ & \mid t \in \text{first}(\beta) \cup \text{follow}(X) \rightarrow \text{parse } \beta \\ & \mid \text{otherwise} \quad \rightarrow \text{Nothing} \end{aligned}$$

The branches must be mutually exclusive!

# First and Follow Sets (1)

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

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

$$\text{first}(\alpha) = \{a \in T \mid \alpha \xrightarrow[G]{*} a\beta\}$$

$$\text{follow}(A) = \{a \in T \mid S \xrightarrow[G]{*} \alpha A a \beta\}$$

$$\cup \{\$ \mid S \xrightarrow[G]{*} \alpha A\}$$

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:

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

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

$$\text{first}(B) = \{b\}$$

$$\text{first}(A) = \{a\}$$

$$\text{first}(S) = \text{first}(ABC)$$

$$= [\text{because } A \xRightarrow{*} \epsilon \text{ and } B \xRightarrow{*} \epsilon]$$

$$\text{first}(A) \cup \text{first}(B) \cup \text{first}(C)$$

$$= \{a, b, c, d\}$$

# First and Follow Sets (3)

Same grammar:

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

Follow sets:

$$\begin{aligned} \text{follow}(C) &= \{\$ \} \\ \text{follow}(B) &= \text{first}(C) = \{c, d\} \\ \text{follow}(A) &= [\text{because } B \xRightarrow{*} \epsilon] \\ &\quad \text{first}(B) \cup \text{first}(C) \\ &= \{b, c, d\} \end{aligned}$$

# LL(1) Grammars (1)

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

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

# LL(1) Grammars (1)

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

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

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:

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

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

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



- 
- 
- 

# 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  $\alpha_i \xRightarrow{*} \epsilon$  for some  $i$ , then, for all  $1 \leq j \leq n$ ,  $j \neq i$ ,

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

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

then it is always clear what to 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  $\alpha_i \xRightarrow{*} \epsilon$  for some  $i$ , then, for all  $1 \leq j \leq n$ ,  $j \neq i$ ,
  - $\alpha_j \not\xRightarrow{*} \epsilon$ , 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  $A \xRightarrow{*} \epsilon$ ; i.e. the set  $N_\epsilon \subseteq N$  of **nullable** nonterminals.

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

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



## Nullable Nonterminals (2)

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

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

(Note that  $A \in N_\epsilon$  if  $A \rightarrow \epsilon \in P$  because  $\text{syms}(\epsilon) = \emptyset$  and  $\forall X \in \emptyset . \dots$  is trivially true.)

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

$$\text{nullable} \in (N \cup T)^* \rightarrow \text{Bool}$$

$$\text{nullable}(\epsilon) = \text{true}$$

$$\text{nullable}(X\alpha) = X \in N_\epsilon \wedge \text{nullable}(\alpha)$$

# 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  $\forall X \in \text{syms}(\alpha) . X \in N_\epsilon$ , 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:

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

# Nullable Nonterminals (4)

Consider the following grammar:

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

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

# Nullable Nonterminals (4)

Consider the following grammar:

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

- 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:

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

- 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  $S \in N_\epsilon$ .

# Nullable Nonterminals (4)

Consider the following grammar:

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

- 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:

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

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



# Computing First Sets (2)

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

$$\text{first} \in (N \cup T)^* \rightarrow \mathcal{P}(T)$$

$$\text{first}(\epsilon) = \emptyset$$

$$\text{first}(a\alpha) = \{a\}$$

$$\text{first}(A\alpha) = \text{first}(A) \cup \begin{cases} \text{first}(\alpha), & \text{if } A \in N_\epsilon \\ \emptyset, & \text{if } A \notin N_\epsilon \end{cases}$$

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  $N_\epsilon$  is computed.

Note that the smallest solution to set equations of the type

$$X = X \cup Y$$

when there are no other constraints on  $X$  is simply

$$X = Y$$

# Computing First Sets (4)

Consider (again):

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

First compute the nullable nonterminals:

$$N_\epsilon = \{A, B\}.$$

Then compute first sets:

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

# Computing First Sets (5)

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

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

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

# Computing First Sets (6)

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

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

# Computing Follow Sets (1)

For a CFG  $G = (N, T, P, S)$ , the sets  $\text{follow}(A)$  for  $A \in N$  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)

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

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

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

# Computing Follow Sets (2)

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

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)

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

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

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

# Computing Follow Sets (3)

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

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

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

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

$$\begin{array}{l} \text{first}(\epsilon) \subseteq \text{follow}(C) \\ \text{follow}(S) \subseteq \text{follow}(C) \end{array}$$

# Computing Follow Sets (4)

In general:

$$X \subseteq Z \wedge Y \subseteq Z \iff X \cup Y \subseteq Z$$

Also, constraints like  $\emptyset \subseteq X$  and  $X \subseteq X$  are trivially satisfied and can be omitted.

The constraints can thus be written as:

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

# Computing Follow Sets (5)

Using

$$\text{first}(\epsilon) = \emptyset$$

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

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

the constraints can be simplified further:

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

$$\{b, c, d\} \subseteq \text{follow}(A)$$

$$\{c, d\} \subseteq \text{follow}(B)$$

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

# Computing Follow Sets (6)

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

$$\text{follow}(S) = \{\$\}$$

$$\text{follow}(A) = \{b, c, d\}$$

$$\text{follow}(B) = \{c, d\}$$

$$\text{follow}(C) = \text{follow}(S) = \{\$\}$$

# LL(1), Left-Recursion, Ambiguity

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

Proof: See the lecture notes.

Thus, left-recursion and any ambiguities first have to be eliminated (see previous lectures).

# 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:

$$S \rightarrow aXbYS'$$
$$S' \rightarrow \epsilon \mid cZ$$

Now suitable for predictive parsing!