

# Automata & Complexity

Jörg Endrullis

Vrije Universiteit Amsterdam

2018

# Looking Back

Last Lecture:

Not all languages are regular. For example

$$L = \{ a^n b^n \mid n \geq 0 \}$$

Proof: use the **pumping lemma!**

# Context-Free Languages

A grammar  $G = (V, T, S, P)$  is called **context-free** if all production rules are of the form

$$A \rightarrow u$$

where  $A \in V$  and  $u \in (V \cup T)^*$ .

That is, the left-hand side of all rules is a single variable.

# Context-Free Languages

A grammar  $G = (V, T, S, P)$  is called **context-free** if all production rules are of the form

$$A \rightarrow u$$

where  $A \in V$  and  $u \in (V \cup T)^*$ .

That is, the left-hand side of all rules is a single variable.

A language  $L$  is **context-free** if there is a context-free grammar with  $L = L(G)$ .

# Context-Free Languages

A grammar  $G = (V, T, S, P)$  is called **context-free** if all production rules are of the form

$$A \rightarrow u$$

where  $A \in V$  and  $u \in (V \cup T)^*$ .

That is, the left-hand side of all rules is a single variable.

A language  $L$  is **context-free** if there is a context-free grammar with  $L = L(G)$ .

Context-free grammars are used to

- describe the **syntax of programming languages**, and
- form the basis of **parsing algorithms**.

# Exercises

Show that the following language is context-free:

$$\{a^n b^m c^{2n} \mid n \geq 0, m \geq 0\}$$

# Exercises

Show that the following language is context-free:

$$\{a^n b^m c^{2n} \mid n \geq 0, m \geq 0\}$$

Show that the following language is context-free:

$$\{x \in \Sigma^+ \mid \forall w \in \Sigma^+. x \neq ww\}$$

where  $\Sigma = \{a, b\}$ .

# Exercises

Show that the following language is context-free:

$$\{a^n b^m c^{2n} \mid n \geq 0, m \geq 0\}$$

Show that the following language is context-free:

$$\{x \in \Sigma^+ \mid \forall w \in \Sigma^+. x \neq ww\}$$

where  $\Sigma = \{a, b\}$ .

Is the following language context-free?

$$\{ww \mid w \in \Sigma^+\}$$

where  $\Sigma = \{a, b\}$ .



## Rightmost and Leftmost

Let  $G$  be a grammar, and consider a derivation  $S \Rightarrow^* w$ .

- If  $G$  is **(right) linear**,  $w$  contains **at most one variable**.
- If  $G$  is **context-free**,  $w$  can contain **multiple variables**.

## Rightmost and Leftmost

Let  $G$  be a grammar, and consider a derivation  $S \Rightarrow^* w$ .

- If  $G$  is **(right) linear**,  $w$  contains **at most one variable**.
- If  $G$  is **context-free**,  $w$  can contain **multiple variables**.

Two strategies to choose which variable to expand:

- **leftmost**: always the leftmost variable
- **rightmost**: always the rightmost variable

$$S \rightarrow SaS \mid b$$

Two derivations of  $bab$ :

**Leftmost:**  $S \Rightarrow SaS \Rightarrow baS \Rightarrow bab$

**Rightmost:**  $S \Rightarrow SaS \Rightarrow Sab \Rightarrow bab$

# Rightmost and Leftmost

Let  $G$  be a grammar, and consider a derivation  $S \Rightarrow^* w$ .

- If  $G$  is **(right) linear**,  $w$  contains **at most one variable**.
- If  $G$  is **context-free**,  $w$  can contain **multiple variables**.

Two strategies to choose which variable to expand:

- **leftmost**: always the leftmost variable
- **rightmost**: always the rightmost variable

$$S \rightarrow SaS \mid b$$

Two derivations of  $bab$ :

**Leftmost:**  $S \Rightarrow SaS \Rightarrow baS \Rightarrow bab$

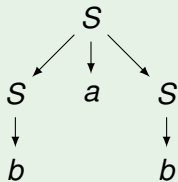
**Rightmost:**  $S \Rightarrow SaS \Rightarrow Sab \Rightarrow bab$

Result depends not on the strategy, but the choice of the rules.

# Derivation Trees

$$S \rightarrow SaS \mid b$$

A **derivation tree** for *bab* is:



# Derivation Trees

A **derivation tree** for a context-free grammar  $G = (V, T, S, P)$  has nodes with labels from  $V \cup T \cup \{\lambda\}$ .

- The **root** has the label  $S$ .
- If there is a production rule
  - $A \rightarrow x_1 \cdots x_n$  with  $n \geq 1$ , then  
a node labelled  $A$  can have children labelled  $x_1, \dots, x_n$ .
  - $A \rightarrow \lambda$ , then  
a node labelled  $A$  can have one child with label  $\lambda$ .
- Every node with a label from  $T \cup \{\lambda\}$  is a **leaf**.  
(That is, the node has no children).

# Derivation Trees

A **derivation tree** for a context-free grammar  $G = (V, T, S, P)$  has nodes with labels from  $V \cup T \cup \{\lambda\}$ .

- The **root** has the label  $S$ .
- If there is a production rule
  - $A \rightarrow x_1 \cdots x_n$  with  $n \geq 1$ , then a node labelled  $A$  can have children labelled  $x_1, \dots, x_n$ .
  - $A \rightarrow \lambda$ , then a node labelled  $A$  can have one child with label  $\lambda$ .
- Every node with a label from  $T \cup \{\lambda\}$  is a **leaf**. (That is, the node has no children).

The labels of the leaves of a derivation tree, read from left to right (skipping  $\lambda$ ), form a word in  $L(G)$ .

# Ambiguous Grammars

A context-free grammar  $G$  is **ambiguous** if there exists a word  $w \in L(G)$  for which there are multiple derivation trees.

# Ambiguous Grammars

A context-free grammar  $G$  is **ambiguous** if there exists a word  $w \in L(G)$  for which there are multiple derivation trees.

Is the following grammar ambiguous?

$$S \rightarrow S + S \mid 0$$



# Ambiguous Grammars

A context-free grammar  $G$  is **ambiguous** if there exists a word  $w \in L(G)$  for which there are multiple derivation trees.

Is the following grammar ambiguous?

$$S \rightarrow S + S \mid 0$$

Yes, there are two derivations trees for  $0 + 0 + 0$ :

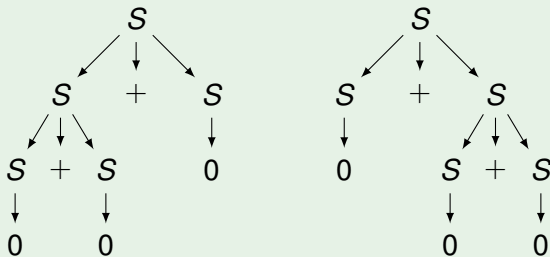
# Ambiguous Grammars

A context-free grammar  $G$  is **ambiguous** if there exists a word  $w \in L(G)$  for which there are multiple derivation trees.

Is the following grammar ambiguous?

$$S \rightarrow S + S \mid 0$$

Yes, there are two derivations trees for  $0 + 0 + 0$ :



# Ambiguous Grammars

Is the following grammar ambiguous?

$$S \rightarrow S + 0 \mid 0$$

# Ambiguous Grammars

Is the following grammar ambiguous?

$$S \rightarrow S + 0 \mid 0$$

No, every word has precisely one derivation tree.

# Ambiguous Grammars

Is the following grammar ambiguous?

$$S \rightarrow S + 0 \mid 0$$

No, every word has precisely one derivation tree.

Note that both grammars

$$S \rightarrow S + S \mid 0$$

$$S \rightarrow S + 0 \mid 0$$

generate the same language:

$$\{0(+0)^n \mid n \geq 0\}$$

# Ambiguity in Programming Languages

The following production rules (from ALGOL 60) became known as **dangling else problem**:

$$\begin{aligned} \textit{Statement} &\rightarrow \textit{if Condition then Statement} \mid \\ &\quad \textit{if Condition then Statement else Statement} \mid \\ &\quad \dots \\ \textit{Condition} &\rightarrow \dots \end{aligned}$$

# Ambiguity in Programming Languages

The following production rules (from ALGOL 60) became known as **dangling else problem**:

$$\begin{aligned} \textit{Statement} &\rightarrow \textit{if Condition then Statement} \mid \\ &\quad \textit{if Condition then Statement else Statement} \mid \\ &\quad \dots \\ \textit{Condition} &\rightarrow \dots \end{aligned}$$

There are two derivation trees for

```
if ... then if ... then ... else ...
```

# Ambiguity in Programming Languages

The following production rules (from ALGOL 60) became known as **dangling else problem**:

*Statement*  $\rightarrow$  if *Condition* then *Statement* |  
if *Condition* then *Statement* else *Statement* |  
...  
  
*Condition*  $\rightarrow$  ...

There are two derivation trees for

if ... then if ... then ... else ...

The production rules are **ambiguous**.



# Ambiguity Typically Unwanted

Ambiguity is typically unwanted:

- derivation trees often used to assign meaning to words,
- multiple derivation tree may result in double meaning.

# Ambiguity Typically Unwanted

Ambiguity is typically unwanted:

- derivation trees often used to assign meaning to words,
- multiple derivation tree may result in double meaning.

In practice, ambiguity is often resolved outside of the grammar.

For example, by a precedence on the rules:

- For example,  $0 + 2 * 1$  is parsed as  $0 + (2 * 1)$ .

# Ambiguity Typically Unwanted

Ambiguity is typically unwanted:

- derivation trees often used to assign meaning to words,
- multiple derivation tree may result in double meaning.

In practice, ambiguity is often resolved outside of the grammar. For example, by a precedence on the rules:

- For example,  $0 + 2 * 1$  is parsed as  $0 + (2 * 1)$ .

**Ambiguity is undecidable.** That is, there exists no algorithm that decides whether a context-free grammar is ambiguous.

# Inherently Ambiguous Languages

A language  $L$  is **inherently ambiguous** if

- $L$  is context-free, and
- **every** grammar  $G$  with  $L(G) = L$  is ambiguous.

# Inherently Ambiguous Languages

A language  $L$  is **inherently ambiguous** if

- $L$  is context-free, and
- **every** grammar  $G$  with  $L(G) = L$  is ambiguous.

The following context-free language is inherently ambiguous

$$\{ a^n b^m c^\ell \mid n = m \vee m = \ell \}$$

It is generated by the following (ambiguous) grammar

$$\begin{array}{lll} S \rightarrow A \mid B & A \rightarrow Ac \mid C & B \rightarrow aB \mid D \\ & C \rightarrow aCb \mid \lambda & D \rightarrow bDc \mid \lambda \end{array}$$

# Inherently Ambiguous Languages

A language  $L$  is **inherently ambiguous** if

- $L$  is context-free, and
- **every** grammar  $G$  with  $L(G) = L$  is ambiguous.

The following context-free language is inherently ambiguous

$$\{ a^n b^m c^\ell \mid n = m \vee m = \ell \}$$

It is generated by the following (ambiguous) grammar

$$\begin{array}{lll} S \rightarrow A \mid B & A \rightarrow Ac \mid C & B \rightarrow aB \mid D \\ & C \rightarrow aCb \mid \lambda & D \rightarrow bDc \mid \lambda \end{array}$$

(Groups of two, 2 minutes)

## Exercise

Find two derivation trees for the word  $abc$ .

# Parsing

**Parsing** is the search for a derivation tree for a given word.

# Parsing

**Parsing** is the search for a derivation tree for a given word.

## Theorem

For context-free grammars, parsing is possible in  $O(|w|^3)$  time.  
(Here  $|w|$  is the length of the input word.)



# Parsing

**Parsing** is the search for a derivation tree for a given word.

## Theorem

For context-free grammars, parsing is possible in  $O(|w|^3)$  time.  
(Here  $|w|$  is the length of the input word.)

For every context-free grammar  $G$ ,  
there exists an algorithm and  $C \geq 0$  such that for every

- word  $w$ ,

the algorithm determines in at most  $C \cdot |w|^3$  steps

- whether  $w \in L(G)$  (including a derivation tree if  $w \in L(G)$ ).

## Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

### Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

# Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be a context-free grammar with  $L(G) = L$ .

# Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be a context-free grammar with  $L(G) = L$ .

Determine all variables  $A$  in  $G$  with  $A \Rightarrow^* \lambda$ :

# Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be a context-free grammar with  $L(G) = L$ .

Determine all variables  $A$  in  $G$  with  $A \Rightarrow^* \lambda$ :

- If  $A \rightarrow \lambda$ , then  $A \Rightarrow^* \lambda$ .

# Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be a context-free grammar with  $L(G) = L$ .

Determine all variables  $A$  in  $G$  with  **$A \Rightarrow^* \lambda$** :

- If  $A \rightarrow \lambda$ , then  $A \Rightarrow^* \lambda$ .
- If  $A \rightarrow B_1 \cdots B_n$  and  $B_1 \Rightarrow^* \lambda, \dots, B_n \Rightarrow^* \lambda$ , then  $A \Rightarrow^* \lambda$ .

# Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be a context-free grammar with  $L(G) = L$ .

Determine all variables  $A$  in  $G$  with  $A \Rightarrow^* \lambda$ :

- If  $A \rightarrow \lambda$ , then  $A \Rightarrow^* \lambda$ .
- If  $A \rightarrow B_1 \cdots B_n$  and  $B_1 \Rightarrow^* \lambda, \dots, B_n \Rightarrow^* \lambda$ , then  $A \Rightarrow^* \lambda$ .

For every rule  $A \rightarrow xBy$  with  $B \Rightarrow^* \lambda$ , add a rule  $A \rightarrow xy$ .



# Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be a context-free grammar with  $L(G) = L$ .

Determine all variables  $A$  in  $G$  with  $A \Rightarrow^* \lambda$ :

- If  $A \rightarrow \lambda$ , then  $A \Rightarrow^* \lambda$ .
- If  $A \rightarrow B_1 \cdots B_n$  and  $B_1 \Rightarrow^* \lambda, \dots, B_n \Rightarrow^* \lambda$ , then  $A \Rightarrow^* \lambda$ .

For every rule  $A \rightarrow xBy$  with  $B \Rightarrow^* \lambda$ , add a rule  $A \rightarrow xy$ .

Remove all  $\lambda$ -production rules.

# Removal of $\lambda$ -Rules

A production rule  $A \rightarrow \lambda$  is called  **$\lambda$ -production rule**.

## Theorem

For every context-free language  $L$  there exists a context-free grammar  $G$  **without  $\lambda$ -rules** such that  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be a context-free grammar with  $L(G) = L$ .

Determine all variables  $A$  in  $G$  with  $A \Rightarrow^* \lambda$ :

- If  $A \rightarrow \lambda$ , then  $A \Rightarrow^* \lambda$ .
- If  $A \rightarrow B_1 \cdots B_n$  and  $B_1 \Rightarrow^* \lambda, \dots, B_n \Rightarrow^* \lambda$ , then  $A \Rightarrow^* \lambda$ .

For every rule  $A \rightarrow xBy$  with  $B \Rightarrow^* \lambda$ , add a rule  $A \rightarrow xy$ .

Remove all  $\lambda$ -production rules.

The resulting grammar  $G$  has the property  $L(G) = L \setminus \{\lambda\}$ .

# Exercise

Consider the following grammar

$$S \rightarrow ABaC$$

$$A \rightarrow BC$$

$$B \rightarrow b \mid \lambda$$

$$D \rightarrow d$$

$$C \rightarrow D \mid \lambda$$

# Exercise

Consider the following grammar

$$\begin{array}{llll} S \rightarrow ABaC & A \rightarrow BC & B \rightarrow b \mid \lambda & D \rightarrow d \\ & & C \rightarrow D \mid \lambda & \end{array}$$

Which variables can produce  $\lambda$ ?

# Exercise

Consider the following grammar

$$S \rightarrow ABaC \quad A \rightarrow BC \quad B \rightarrow b \mid \lambda \quad D \rightarrow d$$
$$C \rightarrow D \mid \lambda$$

Which variables can produce  $\lambda$ ?

- $A$ ,  $B$  and  $C$

# Exercise

Consider the following grammar

$$S \rightarrow ABaC \quad A \rightarrow BC \quad B \rightarrow b \mid \lambda \quad D \rightarrow d$$
$$C \rightarrow D \mid \lambda$$

Which variables can produce  $\lambda$ ?

- $A$ ,  $B$  and  $C$

Construct the resulting grammar after removing all  $\lambda$ -rules:

# Exercise

Consider the following grammar

$$S \rightarrow ABaC \quad A \rightarrow BC \quad B \rightarrow b \mid \lambda \quad D \rightarrow d \\ C \rightarrow D \mid \lambda$$

Which variables can produce  $\lambda$ ?

- $A, B$  and  $C$

Construct the resulting grammar after removing all  $\lambda$ -rules:

$$S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Ba \mid Aa \mid a \\ A \rightarrow BC \mid C \mid B \\ B \rightarrow b \quad C \rightarrow D \quad D \rightarrow d$$

## Removal of Unit Production Rules

A rule  $A \rightarrow B$  is called **unit production rule** (here  $B \in V$ ).



# Removal of Unit Production Rules

A rule  $A \rightarrow B$  is called **unit production rule** (here  $B \in V$ ).

## Theorem

For every context-free language  $L$  there is a context-free grammar  $G$  **without  $\lambda$ - and unit-productions** with  $L(G) = L \setminus \{\lambda\}$ .

# Removal of Unit Production Rules

A rule  $A \rightarrow B$  is called **unit production rule** (here  $B \in V$ ).

## Theorem

For every context-free language  $L$  there is a context-free grammar  $G$  **without  $\lambda$ - and unit-productions** with  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be context-free, without  $\lambda$ -rules, and  $L(G) = L \setminus \{\lambda\}$ .

# Removal of Unit Production Rules

A rule  $A \rightarrow B$  is called **unit production rule** (here  $B \in V$ ).

## Theorem

For every context-free language  $L$  there is a context-free grammar  $G$  **without  $\lambda$ - and unit-productions** with  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be context-free, without  $\lambda$ -rules, and  $L(G) = L \setminus \{\lambda\}$ .

Determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ .

# Removal of Unit Production Rules

A rule  $A \rightarrow B$  is called **unit production rule** (here  $B \in V$ ).

## Theorem

For every context-free language  $L$  there is a context-free grammar  $G$  **without  $\lambda$ - and unit-productions** with  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be context-free, without  $\lambda$ -rules, and  $L(G) = L \setminus \{\lambda\}$ .

Determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ .

Whenever  $A \Rightarrow^+ B$  and  $B \rightarrow y$  is a rule, add a rule  $A \rightarrow y$ .

# Removal of Unit Production Rules

A rule  $A \rightarrow B$  is called **unit production rule** (here  $B \in V$ ).

## Theorem

For every context-free language  $L$  there is a context-free grammar  $G$  **without  $\lambda$ - and unit-productions** with  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be context-free, without  $\lambda$ -rules, and  $L(G) = L \setminus \{\lambda\}$ .

Determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ .

Whenever  $A \Rightarrow^+ B$  and  $B \rightarrow y$  is a rule, add a rule  $A \rightarrow y$ .

Remove all unit production rules.

# Removal of Unit Production Rules

A rule  $A \rightarrow B$  is called **unit production rule** (here  $B \in V$ ).

## Theorem

For every context-free language  $L$  there is a context-free grammar  $G$  **without  $\lambda$ - and unit-productions** with  $L(G) = L \setminus \{\lambda\}$ .

## Construction

Let  $G$  be context-free, without  $\lambda$ -rules, and  $L(G) = L \setminus \{\lambda\}$ .

Determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ .

Whenever  $A \Rightarrow^+ B$  and  $B \rightarrow y$  is a rule, add a rule  $A \rightarrow y$ .

Remove all unit production rules.

The resulting grammar  $G$  has no  $\lambda$ - and unit-productions and it has the property  $L(G) = L \setminus \{\lambda\}$ .

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \qquad A \rightarrow a \mid bc \mid B \qquad B \rightarrow A \mid bb$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)



# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B$$

$$A \rightarrow a \mid bc \mid B$$

$$B \rightarrow A \mid bb$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a$$

$$A \rightarrow a \mid bc \mid B$$

$$B \rightarrow A \mid bb$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc$$

$$A \rightarrow a \mid bc \mid B$$

$$B \rightarrow A \mid bb$$



# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A$$

$$A \rightarrow a \mid bc \mid B$$

$$B \rightarrow A \mid bb$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A \mid bb$$

$$A \rightarrow a \mid bc \mid B$$

$$B \rightarrow A \mid bb$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A \mid bb$$

$$A \rightarrow a \mid bc \mid B \mid A$$

$$B \rightarrow A \mid bb$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A \mid bb$$

$$A \rightarrow a \mid bc \mid B \mid A \mid bb$$

$$B \rightarrow A \mid bb$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A \mid bb$$

$$A \rightarrow a \mid bc \mid B \mid A \mid bb$$

$$B \rightarrow A \mid bb \mid a$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A \mid bb$$

$$A \rightarrow a \mid bc \mid B \mid A \mid bb$$

$$B \rightarrow A \mid bb \mid a \mid bc$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A \mid bb$$

$$A \rightarrow a \mid bc \mid B \mid A \mid bb$$

$$B \rightarrow A \mid bb \mid a \mid bc \mid B$$

# Exercise

Remove all unit production rules from

$$S \rightarrow Aa \mid B \quad A \rightarrow a \mid bc \mid B \quad B \rightarrow A \mid bb$$

Note that there are no  $\lambda$ -productions.

(So no need to first remove  $\lambda$ -productions.)

We determine all pairs  $A \neq B$  with  $A \Rightarrow^+ B$ :

$$S \Rightarrow^+ B \quad A \Rightarrow^+ B \quad B \Rightarrow^+ A \quad S \Rightarrow^+ A$$

Thus we add the following rules:

$$S \rightarrow Aa \mid B \mid a \mid bc \mid A \mid bb$$

$$A \rightarrow a \mid bc \mid B \mid A \mid bb$$

$$B \rightarrow A \mid bb \mid a \mid bc \mid B$$

Removing all unit production rules yields the final result:

$$S \rightarrow a \mid bb \mid bc \mid Aa \quad A \rightarrow a \mid bb \mid bc \quad B \rightarrow a \mid bb \mid bc$$



# Chomsky Normal Form

In a grammar in **Chomsky normal form** all rules have the form

$$A \rightarrow BC \quad \text{or} \quad A \rightarrow a$$

# Chomsky Normal Form

In a grammar in **Chomsky normal form** all rules have the form

$$A \rightarrow BC \text{ or } A \rightarrow a$$

Note that a grammar in Chomsky normal form contains

- no  $\lambda$ -production rules,
- no unit production rules.

# Chomsky Normal Form

In a grammar in **Chomsky normal form** all rules have the form

$$A \rightarrow BC \text{ or } A \rightarrow a$$

Note that a grammar in Chomsky normal form contains

- no  $\lambda$ -production rules,
- no unit production rules.

## Theorem

For every context-free language  $L$  there is a grammar  $G$  in **Chomsky normal form** with  $L(G) = L \setminus \{\lambda\}$ .

# Chomsky Normal Form

## Construction

Let  $G$  be a context-free grammar without  $\lambda$ - and unit-productions and  $L(G) = L \setminus \{\lambda\}$ .

# Chomsky Normal Form

## Construction

Let  $G$  be a context-free grammar without  $\lambda$ - and unit-productions and  $L(G) = L \setminus \{\lambda\}$ .

Introduce variables  $C_a$  and rules  $C_a \rightarrow a$  for every  $a \in T$ .

# Chomsky Normal Form

## Construction

Let  $G$  be a context-free grammar without  $\lambda$ - and unit-productions and  $L(G) = L \setminus \{\lambda\}$ .

Introduce variables  $C_a$  and rules  $C_a \rightarrow a$  for every  $a \in T$ .

Replace every rule  $A \rightarrow x_1 \cdots x_n$  ( $x_i \in V \cup T$ ) with  $n \geq 2$  by

$$A \rightarrow B_1 \cdots B_n \quad \text{where} \quad B_i = \begin{cases} x_i, & \text{if } x_i \in V \\ C_{x_i}, & \text{if } x_i \in T \end{cases}$$

# Chomsky Normal Form

## Construction

Let  $G$  be a context-free grammar without  $\lambda$ - and unit-productions and  $L(G) = L \setminus \{\lambda\}$ .

Introduce variables  $C_a$  and rules  $C_a \rightarrow a$  for every  $a \in T$ .

Replace every rule  $A \rightarrow x_1 \cdots x_n$  ( $x_i \in V \cup T$ ) with  $n \geq 2$  by

$$A \rightarrow B_1 \cdots B_n \quad \text{where} \quad B_i = \begin{cases} x_i, & \text{if } x_i \in V \\ C_{x_i}, & \text{if } x_i \in T \end{cases}$$

Replace every  $A \rightarrow B_1 \cdots B_n$  with  $n \geq 3$  by

$$A \rightarrow B_1 \cdots B_{n-2} C \quad C \rightarrow B_{n-1} B_n$$

where  $C$  is a fresh variable.

# Chomsky Normal Form

## Construction

Let  $G$  be a context-free grammar without  $\lambda$ - and unit-productions and  $L(G) = L \setminus \{\lambda\}$ .

Introduce variables  $C_a$  and rules  $C_a \rightarrow a$  for every  $a \in T$ .

Replace every rule  $A \rightarrow x_1 \cdots x_n$  ( $x_i \in V \cup T$ ) with  $n \geq 2$  by

$$A \rightarrow B_1 \cdots B_n \quad \text{where} \quad B_i = \begin{cases} x_i, & \text{if } x_i \in V \\ C_{x_i}, & \text{if } x_i \in T \end{cases}$$

Replace every  $A \rightarrow B_1 \cdots B_n$  with  $n \geq 3$  by

$$A \rightarrow B_1 \cdots B_{n-2} C \quad C \rightarrow B_{n-1} B_n$$

where  $C$  is a fresh variable.

Repeat the last step until all rules are in Chomsky normal form.



# Exercise

(Groups of two, 2 minutes)

Transform the following context-free grammar

$$S \rightarrow aAbB$$

$$A \rightarrow SbBa \mid a$$

$$B \rightarrow AbBb \mid aba$$

into Chomsky normal form.

# Bottom-up Parsing

**Bottom-up parsing** applies rules backwards, it tries to construct the starting variable  $S$  from the input word.

# Bottom-up Parsing

**Bottom-up parsing** applies rules backwards, it tries to construct the starting variable  $S$  from the input word.

## Cocke-Younger-Kasami algorithm (1965)

The **CYK algorithm** is a bottom-up parsing technique for grammars in Chomsky normal form.

## Cocke-Younger-Kasami algoritme (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $a_1 \cdots a_n$  (with  $n > 0$ ) is in  $L(G)$ .

# Cocke-Younger-Kasami algoritme (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $a_1 \cdots a_n$  (with  $n > 0$ ) is in  $L(G)$ .

**Idea:** compute sets  $V_{ij}$  of variables such that

$$V_{ij} = \{A \in V \mid A \Rightarrow^+ a_i \cdots a_j\} \quad (1 \leq i \leq j \leq n)$$

# Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $a_1 \cdots a_n$  (with  $n > 0$ ) is in  $L(G)$ .

**Idea:** compute sets  $V_{ij}$  of variables such that

$$V_{ij} = \{A \in V \mid A \Rightarrow^+ a_i \cdots a_j\} \quad (1 \leq i \leq j \leq n)$$

as follows:

# Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $a_1 \cdots a_n$  (with  $n > 0$ ) is in  $L(G)$ .

**Idea:** compute sets  $V_{ij}$  of variables such that

$$V_{ij} = \{A \in V \mid A \Rightarrow^+ a_i \cdots a_j\} \quad (1 \leq i \leq j \leq n)$$

as follows:

- $V_{ij} = \{A \in V \mid A \rightarrow a_i \in P\}$  for every  $1 \leq i \leq n$ .

# Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $a_1 \cdots a_n$  (with  $n > 0$ ) is in  $L(G)$ .

**Idea:** compute sets  $V_{ij}$  of variables such that

$$V_{ij} = \{A \in V \mid A \Rightarrow^+ a_i \cdots a_j\} \quad (1 \leq i \leq j \leq n)$$

as follows:

- $V_{ii} = \{A \in V \mid A \rightarrow a_i \in P\}$  for every  $1 \leq i \leq n$ .
- $V_{ij}$  with  $i < j$  is the set of all  $A \in V$  such that

$$A \rightarrow BC \in P \quad \text{with} \quad B \in V_{ik} \text{ and } C \in V_{k+1j}$$

for some  $k$  with  $i \leq k < j$ .



# Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $a_1 \cdots a_n$  (with  $n > 0$ ) is in  $L(G)$ .

**Idea:** compute sets  $V_{ij}$  of variables such that

$$V_{ij} = \{A \in V \mid A \Rightarrow^+ a_i \cdots a_j\} \quad (1 \leq i \leq j \leq n)$$

as follows:

- $V_{ii} = \{A \in V \mid A \rightarrow a_i \in P\}$  for every  $1 \leq i \leq n$ .
- $V_{ij}$  with  $i < j$  is the set of all  $A \in V$  such that

$$A \rightarrow BC \in P \quad \text{with} \quad B \in V_{ik} \text{ and } C \in V_{k+1j}$$

for some  $k$  with  $i \leq k < j$ .

Finally,  $a_1 \cdots a_n \in L(G) \Leftrightarrow S \in V_{1n}$ .

# Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $a_1 \cdots a_n$  (with  $n > 0$ ) is in  $L(G)$ .

**Idea:** compute sets  $V_{ij}$  of variables such that

$$V_{ij} = \{A \in V \mid A \Rightarrow^+ a_i \cdots a_j\} \quad (1 \leq i \leq j \leq n)$$

as follows:

- $V_{ii} = \{A \in V \mid A \rightarrow a_i \in P\}$  for every  $1 \leq i \leq n$ .
- $V_{ij}$  with  $i < j$  is the set of all  $A \in V$  such that

$$A \rightarrow BC \in P \quad \text{with} \quad B \in V_{ik} \text{ and } C \in V_{k+1j}$$

for some  $k$  with  $i \leq k < j$ .

Finally,  $a_1 \cdots a_n \in L(G) \Leftrightarrow S \in V_{1n}$ .

The next slide presents the same algorithm but with better variable names.

## Cocke-Younger-Kasami algoritme (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $w \neq \lambda$  is in  $L(G)$ .

## Cocke-Younger-Kasami algoritme (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $w \neq \lambda$  is in  $L(G)$ .

**Idea:** compute sets  $V_u$  of variables ( $u$  subword of  $w$ ) such that

$$V_u = \{A \in V \mid A \Rightarrow^+ u\}$$

## Cocke-Younger-Kasami algoritme (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $w \neq \lambda$  is in  $L(G)$ .

**Idea:** compute sets  $V_u$  of variables ( $u$  subword of  $w$ ) such that

$$V_u = \{A \in V \mid A \Rightarrow^+ u\}$$

as follows:

## Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $w \neq \lambda$  is in  $L(G)$ .

**Idea:** compute sets  $V_u$  of variables ( $u$  subword of  $w$ ) such that

$$V_u = \{A \in V \mid A \Rightarrow^+ u\}$$

as follows:

- if  $|u| = 1$ , then  $V_u = \{A \in V \mid A \rightarrow u \in P\}$

## Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $w \neq \lambda$  is in  $L(G)$ .

**Idea:** compute sets  $V_u$  of variables ( $u$  subword of  $w$ ) such that

$$V_u = \{A \in V \mid A \Rightarrow^+ u\}$$

as follows:

- if  $|u| = 1$ , then  $V_u = \{A \in V \mid A \rightarrow u \in P\}$
- if  $|u| > 1$ , then  $V_u$  is the set of all  $A \in V$  such that
  - $u = u_1 u_2$  for some non-empty words  $u_1, u_2$ , and
  - $A \rightarrow BC \in P$  with  $B \in V_{u_1}$  and  $C \in V_{u_2}$ .

## Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $w \neq \lambda$  is in  $L(G)$ .

**Idea:** compute sets  $V_u$  of variables ( $u$  subword of  $w$ ) such that

$$V_u = \{A \in V \mid A \Rightarrow^+ u\}$$

as follows:

- if  $|u| = 1$ , then  $V_u = \{A \in V \mid A \rightarrow u \in P\}$
- if  $|u| > 1$ , then  $V_u$  is the set of all  $A \in V$  such that
  - $u = u_1 u_2$  for some non-empty words  $u_1, u_2$ , and
  - $A \rightarrow BC \in P$  with  $B \in V_{u_1}$  and  $C \in V_{u_2}$ .

Finally,  $w \in L(G) \Leftrightarrow S \in V_w$ .



## Cocke-Younger-Kasami algorithm (1965)

Let  $G$  be a grammar in Chomsky normal form.

**Goal:** determine whether word  $w \neq \lambda$  is in  $L(G)$ .

**Idea:** compute sets  $V_u$  of variables ( $u$  subword of  $w$ ) such that

$$V_u = \{A \in V \mid A \Rightarrow^+ u\}$$

as follows:

- if  $|u| = 1$ , then  $V_u = \{A \in V \mid A \rightarrow u \in P\}$
- if  $|u| > 1$ , then  $V_u$  is the set of all  $A \in V$  such that
  - $u = u_1 u_2$  for some non-empty words  $u_1, u_2$ , and
  - $A \rightarrow BC \in P$  with  $B \in V_{u_1}$  and  $C \in V_{u_2}$ .

Finally,  $w \in L(G) \Leftrightarrow S \in V_w$ .

**Worst-case time complexity:**  $O(n^3)$

(There are  $n(n+1)/2$  sets  $V_u$ , and computation of  $V_u$  is  $O(n)$ .)

# Exercise

(Groups of two, 2 minutes)

Use the CYK algorithm to check whether  $abbb$  is generated by

$$S \rightarrow AB$$

$$A \rightarrow BB \mid a$$

$$B \rightarrow AB \mid b$$

## Removal of Useless Variables

A variable  $A$  is **useless** for a context-free grammar if there exists **no** derivation of the form

$$S \Rightarrow^* uAv \Rightarrow^+ w \quad \text{with } w \in T^*.$$

# Removal of Useless Variables

A variable  $A$  is **useless** for a context-free grammar if there exists **no** derivation of the form

$$S \Rightarrow^* uAv \Rightarrow^+ w \quad \text{with } w \in T^*.$$

Removing production rules that contain a useless variable from a grammar does not change the generated language.

# Removal of Useless Variables

A variable  $A$  is **useless** for a context-free grammar if there exists **no** derivation of the form

$$S \Rightarrow^* uAv \Rightarrow^+ w \quad \text{with } w \in T^*.$$

Removing production rules that contain a useless variable from a grammar does not change the generated language.

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are useless?

# Removal of Useless Variables

A variable  $A$  is **useless** for a context-free grammar if there exists **no** derivation of the form

$$S \Rightarrow^* uAv \Rightarrow^+ w \quad \text{with } w \in T^*.$$

Removing production rules that contain a useless variable from a grammar does not change the generated language.

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are useless?

- $A$  because there is no derivation  $S \Rightarrow^* uAv$

# Removal of Useless Variables

A variable  $A$  is **useless** for a context-free grammar if there exists **no** derivation of the form

$$S \Rightarrow^* uAv \Rightarrow^+ w \quad \text{with } w \in T^*.$$

Removing production rules that contain a useless variable from a grammar does not change the generated language.

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are useless?

- $A$  because there is no derivation  $S \Rightarrow^* uAv$
- $C$  because there is no derivation  $C \Rightarrow^* w$  with  $w \in T^*$

# Removal of Useless Variables

A variable  $A$  is **useless** for a context-free grammar if there exists **no** derivation of the form

$$S \Rightarrow^* uAv \Rightarrow^+ w \quad \text{with } w \in T^*.$$

Removing production rules that contain a useless variable from a grammar does not change the generated language.

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are useless?

- $A$  because there is no derivation  $S \Rightarrow^* uAv$
- $C$  because there is no derivation  $C \Rightarrow^* w$  with  $w \in T^*$
- $B$  because  $B$  can be reached only together with  $C$



# Removal of Useless Variables

A variable  $A$  is **useless** for a context-free grammar if there exists **no** derivation of the form

$$S \Rightarrow^* uAv \Rightarrow^+ w \quad \text{with } w \in T^*.$$

Removing production rules that contain a useless variable from a grammar does not change the generated language.

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are useless?

- $A$  because there is no derivation  $S \Rightarrow^* uAv$
- $C$  because there is no derivation  $C \Rightarrow^* w$  with  $w \in T^*$
- $B$  because  $B$  can be reached only together with  $C$

The resulting grammar is  $S \rightarrow aSb \mid \lambda$ .

# Removal of Useless Variables

## Question

How to determine useless variables of a context-free grammar?

# Removal of Useless Variables

## Question

How to determine useless variables of a context-free grammar?

## Construction

A variable  $A$  is called **productive** if  $A \Rightarrow^+ w$  with  $w \in T^*$ .

# Removal of Useless Variables

## Question

How to determine useless variables of a context-free grammar?

## Construction

A variable  $A$  is called **productive** if  $A \Rightarrow^+ w$  with  $w \in T^*$ .

We determine all productive variables:

- If  $A \rightarrow y$  is a rule and all variables in  $y$  are productive, then  $A$  is productive.

# Removal of Useless Variables

## Question

How to determine useless variables of a context-free grammar?

## Construction

A variable  $A$  is called **productive** if  $A \Rightarrow^+ w$  with  $w \in T^*$ .

We determine all productive variables:

- If  $A \rightarrow y$  is a rule and all variables in  $y$  are productive, then  $A$  is productive.

Remove all rules that contain a **non-productive** variable.

# Removal of Useless Variables

## Question

How to determine useless variables of a context-free grammar?

## Construction

A variable  $A$  is called **productive** if  $A \Rightarrow^+ w$  with  $w \in T^*$ .

We determine all productive variables:

- If  $A \rightarrow y$  is a rule and all variables in  $y$  are productive, then  $A$  is productive.

Remove all rules that contain a **non-productive** variable.

We determine all **reachable** variables as follows:

- $S$  is reachable.
- If  $A \rightarrow y$  and  $A$  is reachable, then so are all variables in  $y$ .

# Removal of Useless Variables

## Question

How to determine useless variables of a context-free grammar?

## Construction

A variable  $A$  is called **productive** if  $A \Rightarrow^+ w$  with  $w \in T^*$ .

We determine all productive variables:

- If  $A \rightarrow y$  is a rule and all variables in  $y$  are productive, then  $A$  is productive.

Remove all rules that contain a **non-productive** variable.

We determine all **reachable** variables as follows:

- $S$  is reachable.
- If  $A \rightarrow y$  and  $A$  is reachable, then so are all variables in  $y$ .

Remove all rules that contain a **non-reachable** variable.

# Removal of Useless Variables

## Question

How to determine useless variables of a context-free grammar?

## Construction

A variable  $A$  is called **productive** if  $A \Rightarrow^+ w$  with  $w \in T^*$ .

We determine all productive variables:

- If  $A \rightarrow y$  is a rule and all variables in  $y$  are productive, then  $A$  is productive.

Remove all rules that contain a **non-productive** variable.

We determine all **reachable** variables as follows:

- $S$  is reachable.
- If  $A \rightarrow y$  and  $A$  is reachable, then so are all variables in  $y$ .

Remove all rules that contain a **non-reachable** variable.

A variable is **useless** if it is **not in one of the remaining rules**.



## Removal of Useless Variables

$S \rightarrow aSb \mid BC \mid \lambda$      $A \rightarrow Sb$      $B \rightarrow a$      $C \rightarrow C$

## Removal of Useless Variables

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are non-productive?

# Removal of Useless Variables

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are non-productive?

- $C$  is not productive

# Removal of Useless Variables

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are non-productive?

- $C$  is not productive

We remove all rules containing non-productive variables:

$$S \rightarrow aSb \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a$$

# Removal of Useless Variables

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are non-productive?

- $C$  is not productive

We remove all rules containing non-productive variables:

$$S \rightarrow aSb \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a$$

Which variables are reachable from  $S$ ?

# Removal of Useless Variables

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are non-productive?

- $C$  is not productive

We remove all rules containing non-productive variables:

$$S \rightarrow aSb \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a$$

Which variables are reachable from  $S$ ?

- only  $S$  is reachable

# Removal of Useless Variables

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are non-productive?

- $C$  is not productive

We remove all rules containing non-productive variables:

$$S \rightarrow aSb \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a$$

Which variables are reachable from  $S$ ?

- only  $S$  is reachable

We remove all rules containing non-reachable variables:

$$S \rightarrow aSb \mid \lambda$$

## Removal of Useless Variables

$$S \rightarrow aSb \mid BC \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a \quad C \rightarrow C$$

Which variables are non-productive?

- $C$  is not productive

We remove all rules containing non-productive variables:

$$S \rightarrow aSb \mid \lambda \quad A \rightarrow Sb \quad B \rightarrow a$$

Which variables are reachable from  $S$ ?

- only  $S$  is reachable

We remove all rules containing non-reachable variables:

$$S \rightarrow aSb \mid \lambda$$

Hence only  $S$  is useful, the variables  $A, B, C$  are not useful.



# Top-down Parsing

**Top-down parsing** tries to derive the input word from the starting variable  $S$ .

# Top-down Parsing

**Top-down parsing** tries to derive the input word from the starting variable  $S$ .

## **Simple leftmost strategy:**

- Always expand the leftmost variable  $A$ .  
(Replace  $A$  by  $u$  if there is a production  $A \rightarrow u$ )
- Backtrack when a mismatch with the input string is found.  
Then try another production rule  $A \rightarrow v$ .

# Top-down Parsing

**Top-down parsing** tries to derive the input word from the starting variable  $S$ .

## Simple leftmost strategy:

- Always expand the leftmost variable  $A$ .  
(Replace  $A$  by  $u$  if there is a production  $A \rightarrow u$ )
- Backtrack when a mismatch with the input string is found.  
Then try another production rule  $A \rightarrow v$ .

**Disadvantage:** backtracking is expensive and difficult.

# LL Parsing

## LL parsing

Parsing **top-down** with a **leftmost** strategy.

Backtracking is **not** allowed.

LL parsing does not work for every context-free grammar.

# LL Parsing

## LL parsing

Parsing **top-down** with a **leftmost** strategy.

Backtracking is **not** allowed.

LL parsing does not work for every context-free grammar.

Starting point is a context-free grammar  $G = (V, T, S, P)$ :

- **without useless variables**
- $\lambda$ -productions and unit productions are allowed  
(elimination often increases the size of the grammar)

# LL Parsing

## LL parsing

Parsing **top-down** with a **leftmost** strategy.

Backtracking is **not** allowed.

LL parsing does not work for every context-free grammar.

Starting point is a context-free grammar  $G = (V, T, S, P)$ :

- **without useless variables**
- $\lambda$ -productions and unit productions are allowed  
(elimination often increases the size of the grammar)

Steps of LL parsing:

- Construct sets  $\text{First}(A)$  and  $\text{Follow}(A)$  for every variable  $A$ .
- Construct a parsing table.
- Parse the input word using the parsing table.

# Erasurable Variables

A variable  $A$  is called **erasurable** if  $A \Rightarrow^+ \lambda$ .

# Erasurable Variables

A variable  $A$  is called **erasable** if  $A \Rightarrow^+ \lambda$ .

The set of erasurable variables is the smallest set such that:

- If  $A \rightarrow \lambda$ , then  $A$  is erasurable.
- If  $A \rightarrow B_1 \cdots B_n$  and  $B_1, \dots, B_n$  are erasurable, then  $A$  is erasurable.



# Erasurable Variables

A variable  $A$  is called **erasurable** if  $A \Rightarrow^+ \lambda$ .

The set of erasurable variables is the smallest set such that:

- If  $A \rightarrow \lambda$ , then  $A$  is erasurable.
- If  $A \rightarrow B_1 \cdots B_n$  and  $B_1, \dots, B_n$  are erasurable, then  $A$  is erasurable.

## Algorithm

The set of erasurable variables can be constructed as follows:

1. Let  $E = \emptyset$ .
2. Repeat the following until there is nothing fresh to add:
  - If  $A \rightarrow \lambda$ , then add  $A$  to  $E$ .
  - If  $A \rightarrow B_1 \cdots B_n$  and  $B_1, \dots, B_n$  are in  $E$ , then add  $A$  to  $E$ .
3. Finally,  $E$  is the set of erasurable variables.

# Erasurable Variables

Consider the following grammar:

$$\begin{array}{llll} S \rightarrow AcB & A \rightarrow CBC & B \rightarrow abB & C \rightarrow cCd \\ & & B \rightarrow \lambda & C \rightarrow BB \end{array}$$

We determine the set of erasurable variables:

# Erasurable Variables

Consider the following grammar:

$$\begin{array}{llll} S \rightarrow AcB & A \rightarrow CBC & B \rightarrow abB & C \rightarrow cCd \\ & & B \rightarrow \lambda & C \rightarrow BB \end{array}$$

We determine the set of erasurable variables:

- $B$  is erasurable because of the rule  $B \rightarrow \lambda$

# Erasurable Variables

Consider the following grammar:

$$\begin{array}{llll} S \rightarrow AcB & A \rightarrow CBC & B \rightarrow abB & C \rightarrow cCd \\ & & B \rightarrow \lambda & C \rightarrow BB \end{array}$$

We determine the set of erasurable variables:

- $B$  is erasurable because of the rule  $B \rightarrow \lambda$
- $C$  is erasurable because of  $C \rightarrow BB$  and  $B$  is erasurable

# Erasurable Variables

Consider the following grammar:

$$\begin{array}{llll} S \rightarrow AcB & A \rightarrow CBC & B \rightarrow abB & C \rightarrow cCd \\ & & B \rightarrow \lambda & C \rightarrow BB \end{array}$$

We determine the set of erasurable variables:

- $B$  is erasurable because of the rule  $B \rightarrow \lambda$
- $C$  is erasurable because of  $C \rightarrow BB$  and  $B$  is erasurable
- $A$  is erasurable because of  $A \rightarrow CBC$  and  $B, C$  are erasurable

# Erasurable Variables

Consider the following grammar:

$$\begin{array}{llll} S \rightarrow AcB & A \rightarrow CBC & B \rightarrow abB & C \rightarrow cCd \\ & & B \rightarrow \lambda & C \rightarrow BB \end{array}$$

We determine the set of erasurable variables:

- $B$  is erasurable because of the rule  $B \rightarrow \lambda$
- $C$  is erasurable because of  $C \rightarrow BB$  and  $B$  is erasurable
- $A$  is erasurable because of  $A \rightarrow CBC$  and  $B, C$  are erasurable

So the variables  $A, B, C$  are erasurable.

# First( $A$ )

We consider the first terminal letters derivable from a word:

$$\text{First}(w) = \{a \in T \mid w \Rightarrow^* a\dots\} \cup \{\lambda \mid w \Rightarrow^* \lambda\}$$

# First( $A$ )

We consider the first terminal letters derivable from a word:

$$\text{First}(w) = \{a \in T \mid w \Rightarrow^* a \dots\} \cup \{\lambda \mid w \Rightarrow^* \lambda\}$$

## Algorithm

Let  $\text{PreFirst}(w)$  be the smallest set such that:

- $w \in \text{PreFirst}(w)$
- $a \in \text{PreFirst}(w)$  if  $av \in \text{PreFirst}(w)$
- $B \in \text{PreFirst}(w)$  if  $Bv \in \text{PreFirst}(w)$
- $v \in \text{PreFirst}(w)$  if  $Bv \in \text{PreFirst}(w)$  and  $B$  erasable
- $v \in \text{PreFirst}(w)$  for every  $A \in \text{PreFirst}(w)$  and rule  $A \rightarrow v$

Then  $\text{First}(w)$  consists of

- all terminal letters  $a \in T$  such that  $a \in \text{PreFirst}(w)$ , and
- $\lambda$  if  $w = A_1 A_2 \dots A_n$  for erasable variables  $A_1, \dots, A_n$ .



# Exercise

$$S \rightarrow AA^c$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

# Exercise

$$S \rightarrow AAc$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:

# Exercise

$$S \rightarrow AAc$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:  $A$ .

# Exercise

$$S \rightarrow AA^c$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:  $A$ .

We determine  $\text{PreFirst}(A)$ ,  $\text{PreFirst}(B)$  and  $\text{PreFirst}(S)$ :

# Exercise

$$S \rightarrow AAc$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:  $A$ .

We determine  $\text{PreFirst}(A)$ ,  $\text{PreFirst}(B)$  and  $\text{PreFirst}(S)$ :

$$\text{PreFirst}(A) = \{ \underbrace{A}_{\text{from } A}, \underbrace{Ba}_{\text{from } A}, \underbrace{\lambda}_{\text{from } A}, \underbrace{B}_{\text{from } Ba}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab} \}$$

$$\begin{aligned} \text{PreFirst}(B) &= \{ \underbrace{B}_{\text{from } B}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab}, \underbrace{A}_{\text{from } Ab} \} \cup \text{PreFirst}(A) \\ &= \{ A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

$$\begin{aligned} \text{PreFirst}(S) &= \{ \underbrace{S}_{\text{from } S}, \underbrace{AAc}_{\text{from } AAc}, \underbrace{Ac}_{\text{from } Ac}, \underbrace{c}_{\text{from } Ac}, \underbrace{A}_{\text{from } AAc} \} \cup \text{PreFirst}(A) \\ &= \{ S, AAc, Ac, c, A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

# Exercise

$$S \rightarrow AAc$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:  $A$ .

We determine  $\text{PreFirst}(A)$ ,  $\text{PreFirst}(B)$  and  $\text{PreFirst}(S)$ :

$$\text{PreFirst}(A) = \{ \underbrace{A}_{\text{from } A}, \underbrace{Ba}_{\text{from } A}, \underbrace{\lambda}_{\text{from } A}, \underbrace{B}_{\text{from } Ba}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab} \}$$

$$\begin{aligned} \text{PreFirst}(B) &= \{ \underbrace{B}_{\text{from } B}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab}, \underbrace{A}_{\text{from } Ab} \} \cup \text{PreFirst}(A) \\ &= \{ A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

$$\begin{aligned} \text{PreFirst}(S) &= \{ \underbrace{S}_{\text{from } S}, \underbrace{AAc}_{\text{from } AAc}, \underbrace{Ac}_{\text{from } AAc}, \underbrace{c}_{\text{from } Ac}, \underbrace{A}_{\text{from } AAc} \} \cup \text{PreFirst}(A) \\ &= \{ S, AAc, Ac, c, A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

Thus we get

$$\text{First}(A) =$$

$$\text{First}(B) =$$

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

# Exercise

$$S \rightarrow AAc$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:  $A$ .

We determine  $\text{PreFirst}(A)$ ,  $\text{PreFirst}(B)$  and  $\text{PreFirst}(S)$ :

$$\text{PreFirst}(A) = \{ \underbrace{A}_{\text{from } A}, \underbrace{Ba}_{\text{from } A}, \underbrace{\lambda}_{\text{from } A}, \underbrace{B}_{\text{from } Ba}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab} \}$$

$$\begin{aligned} \text{PreFirst}(B) &= \{ \underbrace{B}_{\text{from } B}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab}, \underbrace{A}_{\text{from } Ab} \} \cup \text{PreFirst}(A) \\ &= \{ A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

$$\begin{aligned} \text{PreFirst}(S) &= \{ \underbrace{S}_{\text{from } S}, \underbrace{AAc}_{\text{from } AAc}, \underbrace{Ac}_{\text{from } AAc}, \underbrace{c}_{\text{from } Ac}, \underbrace{A}_{\text{from } AAc} \} \cup \text{PreFirst}(A) \\ &= \{ S, AAc, Ac, c, A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

Thus we get

$$\text{First}(A) = \{ b, d, \lambda \}$$

$$\text{First}(B) =$$

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

# Exercise

$$S \rightarrow AAc$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:  $A$ .

We determine  $\text{PreFirst}(A)$ ,  $\text{PreFirst}(B)$  and  $\text{PreFirst}(S)$ :

$$\text{PreFirst}(A) = \{ \underbrace{A}_{\text{from } A}, \underbrace{Ba}_{\text{from } A}, \underbrace{\lambda}_{\text{from } A}, \underbrace{B}_{\text{from } Ba}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab} \}$$

$$\begin{aligned} \text{PreFirst}(B) &= \{ \underbrace{B}_{\text{from } B}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab}, \underbrace{A}_{\text{from } Ab} \} \cup \text{PreFirst}(A) \\ &= \{ A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

$$\begin{aligned} \text{PreFirst}(S) &= \{ \underbrace{S}_{\text{from } S}, \underbrace{AAc}_{\text{from } AAc}, \underbrace{Ac}_{\text{from } AAc}, \underbrace{c}_{\text{from } Ac}, \underbrace{A}_{\text{from } AAc} \} \cup \text{PreFirst}(A) \\ &= \{ S, AAc, Ac, c, A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

Thus we get

$$\text{First}(A) = \{ b, d, \lambda \} \quad \text{First}(B) = \{ b, d \} \quad \text{First}(S) =$$



# Exercise

$$S \rightarrow AAc$$

$$A \rightarrow Ba \mid \lambda$$

$$B \rightarrow Ab \mid d$$

The erasable variables ( $V \Rightarrow^+ \lambda$ ) are:  $A$ .

We determine  $\text{PreFirst}(A)$ ,  $\text{PreFirst}(B)$  and  $\text{PreFirst}(S)$ :

$$\text{PreFirst}(A) = \{ \underbrace{A}_{\text{from } A}, \underbrace{Ba}_{\text{from } A}, \underbrace{\lambda}_{\text{from } A}, \underbrace{B}_{\text{from } Ba}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab} \}$$

$$\begin{aligned} \text{PreFirst}(B) &= \{ \underbrace{B}_{\text{from } B}, \underbrace{Ab}_{\text{from } B}, \underbrace{d}_{\text{from } B}, \underbrace{b}_{\text{from } Ab}, \underbrace{A}_{\text{from } Ab} \} \cup \text{PreFirst}(A) \\ &= \{ A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

$$\begin{aligned} \text{PreFirst}(S) &= \{ \underbrace{S}_{\text{from } S}, \underbrace{AAc}_{\text{from } AAc}, \underbrace{Ac}_{\text{from } Ac}, \underbrace{c}_{\text{from } Ac}, \underbrace{A}_{\text{from } AAc} \} \cup \text{PreFirst}(A) \\ &= \{ S, AAc, Ac, c, A, Ba, \lambda, B, Ab, d, b \} \end{aligned}$$

Thus we get

$$\text{First}(A) = \{ b, d, \lambda \} \quad \text{First}(B) = \{ b, d \} \quad \text{First}(S) = \{ b, c, d \}$$

## Follow( $A$ )

The sets  $\text{First}(A)$  are not yet sufficient for 'predictive' parsing, if there are derivations  $A \Rightarrow^+ \lambda$ .

## Follow( $A$ )

The sets  $\text{First}(A)$  are not yet sufficient for 'predictive' parsing, if there are derivations  $A \Rightarrow^+ \lambda$ .

We consider the terminal letters that can follow a variable:

$$\text{Follow}(A) = \{ a \in T \mid S \Rightarrow^* \dots Aa \dots \}$$

Intuition:  $a \in \text{Follow}(A)$  if  $A$  can be followed by  $a$  in a derivation.

# Follow( $A$ )

The sets  $\text{First}(A)$  are not yet sufficient for 'predictive' parsing, if there are derivations  $A \Rightarrow^+ \lambda$ .

We consider the terminal letters that can follow a variable:

$$\text{Follow}(A) = \{ a \in T \mid S \Rightarrow^* \dots Aa \dots \}$$

Intuition:  $a \in \text{Follow}(A)$  if  $A$  can be followed by  $a$  in a derivation.

We use  $\$$  as a special '**end of word**' symbol.

## Algorithm

# Follow( $A$ )

The sets  $\text{First}(A)$  are not yet sufficient for 'predictive' parsing, if there are derivations  $A \Rightarrow^+ \lambda$ .

We consider the terminal letters that can follow a variable:

$$\text{Follow}(A) = \{ a \in T \mid S \Rightarrow^* \dots Aa \dots \}$$

Intuition:  $a \in \text{Follow}(A)$  if  $A$  can be followed by  $a$  in a derivation.

We use  $\$$  as a special '**end of word**' symbol.

## Algorithm

- $\text{Follow}(S) \supseteq \{\$\}$

# Follow( $A$ )

The sets  $\text{First}(A)$  are not yet sufficient for ‘predictive’ parsing, if there are derivations  $A \Rightarrow^+ \lambda$ .

We consider the terminal letters that can follow a variable:

$$\text{Follow}(A) = \{a \in T \mid S \Rightarrow^* \dots Aa \dots\}$$

Intuition:  $a \in \text{Follow}(A)$  if  $A$  can be followed by  $a$  in a derivation.

We use  $\$$  as a special ‘**end of word**’ symbol.

## Algorithm

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$

# Follow( $A$ )

The sets  $\text{First}(A)$  are not yet sufficient for 'predictive' parsing, if there are derivations  $A \Rightarrow^+ \lambda$ .

We consider the terminal letters that can follow a variable:

$$\text{Follow}(A) = \{ a \in T \mid S \Rightarrow^* \dots Aa \dots \}$$

Intuition:  $a \in \text{Follow}(A)$  if  $A$  can be followed by  $a$  in a derivation.

We use  $\$$  as a special '**end of word**' symbol.

## Algorithm

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

# Example

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

If  $C \rightarrow AB$ , then:



# Example

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

If  $C \rightarrow AB$ , then:

- $\text{First}(B) \subseteq \text{Follow}(A)$

Example:  $C \Rightarrow AB \Rightarrow^* Aaw$  if  $B \rightarrow aw$

# Example

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

If  $C \rightarrow AB$ , then:

- $\text{First}(B) \subseteq \text{Follow}(A)$   
Example:  $C \Rightarrow AB \Rightarrow^* Aaw$  if  $B \rightarrow aw$
- $\text{Follow}(C) \subseteq \text{Follow}(B)$   
Example:  $S \Rightarrow Ca \Rightarrow ABa$  if  $S \rightarrow Ca$

# Example

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

If  $C \rightarrow AB$ , then:

- $\text{First}(B) \subseteq \text{Follow}(A)$

Example:  $C \Rightarrow AB \Rightarrow^* Aaw$  if  $B \rightarrow aw$

- $\text{Follow}(C) \subseteq \text{Follow}(B)$

Example:  $S \Rightarrow Ca \Rightarrow ABa$  if  $S \rightarrow Ca$

- $\text{Follow}(C) \subseteq \text{Follow}(A)$  if  $B \Rightarrow^* \lambda$

Example:  $S \Rightarrow Ca \Rightarrow ABa \Rightarrow Aa$  if  $S \rightarrow Ca$  and  $B \rightarrow \lambda$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$



# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$

$$\text{Follow}(A) \supseteq$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$

$$\text{Follow}(A) \supseteq (\text{First}(A) \setminus \{\lambda\})$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$

$$\text{Follow}(A) \supseteq (\text{First}(A) \setminus \{\lambda\}) \cup \{b\}$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$

$$\text{Follow}(A) \supseteq (\text{First}(A) \setminus \{\lambda\}) \cup \{b\} \cup \text{Follow}(D)$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$

$$\text{Follow}(A) \supseteq (\text{First}(A) \setminus \{\lambda\}) \cup \{b\} \cup \text{Follow}(D) \supseteq \{b, c, d\}$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$

$$\text{Follow}(A) \supseteq (\text{First}(A) \setminus \{\lambda\}) \cup \{b\} \cup \text{Follow}(D) \supseteq \{b, c, d\}$$

$$\text{Follow}(B) \supseteq$$

# Exercise

- $\text{Follow}(S) \supseteq \{\$\}$
- $\text{Follow}(A) \supseteq \text{First}(w) \setminus \{\lambda\}$  for every rule  $B \rightarrow vAw$
- $\text{Follow}(A) \supseteq \text{Follow}(B)$  for rules  $B \rightarrow vAw$  with  $\lambda \in \text{First}(w)$

$$S \rightarrow Dc$$

$$A \rightarrow Ba \mid \lambda$$

$$D \rightarrow AA$$

$$B \rightarrow Ab \mid d$$

We have

$$\text{First}(S) = \{b, c, d\}$$

$$\text{First}(A) = \{\lambda, b, d\}$$

$$\text{First}(D) = \{\lambda, b, d\}$$

$$\text{First}(B) = \{b, d\}$$

Determine  $\text{Follow}(S)$ ,  $\text{Follow}(D)$ ,  $\text{Follow}(A)$ ,  $\text{Follow}(B)$ :

$$\text{Follow}(S) \supseteq \{\$\}$$

$$\text{Follow}(D) \supseteq \{c\}$$

$$\text{Follow}(A) \supseteq (\text{First}(A) \setminus \{\lambda\}) \cup \{b\} \cup \text{Follow}(D) \supseteq \{b, c, d\}$$

$$\text{Follow}(B) \supseteq \{a\}$$

# Parser Tables

The **parser table** for a context-free grammar is a table with

- columns indexed by terminals  $T \cup \{\$\}$ ,
- rows indexed by variables  $V$ ,

At place  $[a \in T \cup \{\$\}, A \in V]$  it contains rules  $A \rightarrow u$  for which

- $a \in \text{First}(u)$ , or (never the case for  $a = \$$ )
- $\lambda \in \text{First}(u)$  and  $a \in \text{Follow}(A)$ .



# Parser Tables

The **parser table** for a context-free grammar is a table with

- columns indexed by terminals  $T \cup \{\$\}$ ,
- rows indexed by variables  $V$ ,

At place  $[a \in T \cup \{\$\}, A \in V]$  it contains rules  $A \rightarrow u$  for which

- $a \in \text{First}(u)$ , or (never the case for  $a = \$$ )
- $\lambda \in \text{First}(u)$  and  $a \in \text{Follow}(A)$ .

$$S \rightarrow aSb \mid \lambda$$

We have

- $\text{First}(aSb) = \{a\}$ ,  $\text{First}(\lambda) = \{\lambda\}$ ,  $\text{First}(S) = \{\lambda, a\}$
- $\text{Follow}(S) = \{b, \$\}$ ,

Thus the parser table is:

	$a$	$b$	$\$$
$S$			

# Parser Tables

The **parser table** for a context-free grammar is a table with

- columns indexed by terminals  $T \cup \{\$\}$ ,
- rows indexed by variables  $V$ ,

At place  $[a \in T \cup \{\$\}, A \in V]$  it contains rules  $A \rightarrow u$  for which

- $a \in \text{First}(u)$ , or (never the case for  $a = \$$ )
- $\lambda \in \text{First}(u)$  and  $a \in \text{Follow}(A)$ .

$$S \rightarrow aSb \mid \lambda$$

We have

- $\text{First}(aSb) = \{a\}$ ,  $\text{First}(\lambda) = \{\lambda\}$ ,  $\text{First}(S) = \{\lambda, a\}$
- $\text{Follow}(S) = \{b, \$\}$ ,

Thus the parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$		

# Parser Tables

The **parser table** for a context-free grammar is a table with

- columns indexed by terminals  $T \cup \{\$\}$ ,
- rows indexed by variables  $V$ ,

At place  $[a \in T \cup \{\$\}, A \in V]$  it contains rules  $A \rightarrow u$  for which

- $a \in \text{First}(u)$ , or (never the case for  $a = \$$ )
- $\lambda \in \text{First}(u)$  and  $a \in \text{Follow}(A)$ .

$$S \rightarrow aSb \mid \lambda$$

We have

- $\text{First}(aSb) = \{a\}$ ,  $\text{First}(\lambda) = \{\lambda\}$ ,  $\text{First}(S) = \{\lambda, a\}$
- $\text{Follow}(S) = \{b, \$\}$ ,

Thus the parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	

# Parser Tables

The **parser table** for a context-free grammar is a table with

- columns indexed by terminals  $T \cup \{\$\}$ ,
- rows indexed by variables  $V$ ,

At place  $[a \in T \cup \{\$\}, A \in V]$  it contains rules  $A \rightarrow u$  for which

- $a \in \text{First}(u)$ , or (never the case for  $a = \$$ )
- $\lambda \in \text{First}(u)$  and  $a \in \text{Follow}(A)$ .

$$S \rightarrow aSb \mid \lambda$$

We have

- $\text{First}(aSb) = \{a\}$ ,  $\text{First}(\lambda) = \{\lambda\}$ ,  $\text{First}(S) = \{\lambda, a\}$
- $\text{Follow}(S) = \{b, \$\}$ ,

Thus the parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

To parse  $a_1 \cdots a_n$ , we start with  $\langle S\$, a_1 \cdots a_n\$ \rangle$ .



# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

To parse  $a_1 \cdots a_n$ , we start with  $\langle S\$, a_1 \cdots a_n\$ \rangle$ .

From a state  $\langle v, w \rangle$  we can do the following steps:

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

To parse  $a_1 \cdots a_n$ , we start with  $\langle S\$, a_1 \cdots a_n\$ \rangle$ .

From a state  $\langle v, w \rangle$  we can do the following steps:

- $\langle av', aw' \rangle$  becomes  $\langle v', w' \rangle$

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

To parse  $a_1 \cdots a_n$ , we start with  $\langle S\$, a_1 \cdots a_n\$ \rangle$ .

From a state  $\langle v, w \rangle$  we can do the following steps:

- $\langle av', aw' \rangle$  becomes  $\langle v', w' \rangle$
- $\langle Av', aw' \rangle$  becomes  $\langle uv', aw' \rangle$  if  $A \rightarrow u$  at position  $[a, A]$

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

To parse  $a_1 \cdots a_n$ , we start with  $\langle S\$, a_1 \cdots a_n\$ \rangle$ .

From a state  $\langle v, w \rangle$  we can do the following steps:

- $\langle av', aw' \rangle$  becomes  $\langle v', w' \rangle$
- $\langle Av', aw' \rangle$  becomes  $\langle uv', aw' \rangle$  if  $A \rightarrow u$  at position  $[a, A]$
- $\langle Av', \$ \rangle$  becomes  $\langle v', \$ \rangle$  if  $A \rightarrow u$  at position  $[\$, A]$

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

To parse  $a_1 \cdots a_n$ , we start with  $\langle S\$, a_1 \cdots a_n\$ \rangle$ .

From a state  $\langle v, w \rangle$  we can do the following steps:

- $\langle av', aw' \rangle$  becomes  $\langle v', w' \rangle$
- $\langle Av', aw' \rangle$  becomes  $\langle uv', aw' \rangle$  if  $A \rightarrow u$  at position  $[a, A]$
- $\langle Av', \$ \rangle$  becomes  $\langle v', \$ \rangle$  if  $A \rightarrow u$  at position  $[\$, A]$
- $\langle \$, \$ \rangle$  results in **accept**

# LL(1) Grammars and Parsing

A grammar is **LL(1)** if its parser table contains in every cell  $[a \in T \cup \{\$, A \in V]$  at most one production rule.

An LL(1) parser reads from **L**eft to right, performs a **L**eftmost derivation, and looks always at **1** symbol of the input.

Given an LL(1)-grammar and parsing table.

To parse  $a_1 \cdots a_n$ , we start with  $\langle S\$, a_1 \cdots a_n\$ \rangle$ .

From a state  $\langle v, w \rangle$  we can do the following steps:

- $\langle av', aw' \rangle$  becomes  $\langle v', w' \rangle$
- $\langle Av', aw' \rangle$  becomes  $\langle uv', aw' \rangle$  if  $A \rightarrow u$  at position  $[a, A]$
- $\langle Av', \$ \rangle$  becomes  $\langle v', \$ \rangle$  if  $A \rightarrow u$  at position  $[\$, A]$
- $\langle \$, \$ \rangle$  results in **accept**
- In all other cases,  $\langle v, w \rangle$  results in **reject!**

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$\langle S\$, ab\$ \rangle$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle$$



# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	<i>a</i>	<i>b</i>	<i>\$</i>
<i>S</i>	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$ , ab\$ \rangle \rightarrow \langle aSb\$ , ab\$ \rangle \rightarrow \langle Sb\$ , b\$ \rangle \rightarrow \langle b\$ , b\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	<i>a</i>	<i>b</i>	<i>\$</i>
<i>S</i>	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\begin{aligned} \langle S\$ , ab\$ \rangle &\rightarrow \langle aSb\$ , ab\$ \rangle \rightarrow \langle Sb\$ , b\$ \rangle \rightarrow \langle b\$ , b\$ \rangle \\ &\rightarrow \langle \$ , \$ \rangle \end{aligned}$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\begin{aligned} \langle S\$, ab\$ \rangle &\rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ &\rightarrow \langle \$, \$ \rangle \quad \mathbf{accept} \end{aligned}$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	<i>a</i>	<i>b</i>	<i>\$</i>
<i>S</i>	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle$   
 $\rightarrow \langle \$, \$ \rangle$  **accept**

$\langle S\$, abb\$ \rangle$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\begin{aligned} \langle S\$, ab\$ \rangle &\rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ &\rightarrow \langle \$, \$ \rangle \quad \mathbf{accept} \end{aligned}$$

$$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	<i>a</i>	<i>b</i>	<i>\$</i>
<i>S</i>	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ \rightarrow \langle \$, \$ \rangle \quad \mathbf{accept}$$

$$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\begin{aligned} \langle S\$, ab\$ \rangle &\rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ &\rightarrow \langle \$, \$ \rangle \quad \mathbf{accept} \end{aligned}$$

$$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle$$



# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\begin{aligned} \langle S\$, ab\$ \rangle &\rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ &\rightarrow \langle \$, \$ \rangle \quad \mathbf{accept} \end{aligned}$$

$$\begin{aligned} \langle S\$, abb\$ \rangle &\rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle \\ &\rightarrow \langle \$, b\$ \rangle \end{aligned}$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle$   
 $\rightarrow \langle \$, \$ \rangle$  **accept**

$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle$   
 $\rightarrow \langle \$, b\$ \rangle$  **reject**

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle$   
 $\rightarrow \langle \$, \$ \rangle$  **accept**

$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle$   
 $\rightarrow \langle \$, b\$ \rangle$  **reject**

$\langle S\$, aab\$ \rangle$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ \rightarrow \langle \$, \$ \rangle \quad \mathbf{accept}$$

$$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle \\ \rightarrow \langle \$, b\$ \rangle \quad \mathbf{reject}$$

$$\langle S\$, aab\$ \rangle \rightarrow \langle aSb\$, aab\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ \rightarrow \langle \$, \$ \rangle \quad \mathbf{accept}$$

$$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle \\ \rightarrow \langle \$, b\$ \rangle \quad \mathbf{reject}$$

$$\langle S\$, aab\$ \rangle \rightarrow \langle aSb\$, aab\$ \rangle \rightarrow \langle Sb\$, ab\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ \rightarrow \langle \$, \$ \rangle \quad \mathbf{accept}$$

$$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle \\ \rightarrow \langle \$, b\$ \rangle \quad \mathbf{reject}$$

$$\langle S\$, aab\$ \rangle \rightarrow \langle aSb\$, aab\$ \rangle \rightarrow \langle Sb\$, ab\$ \rangle \rightarrow \langle aSbb\$, ab\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ \rightarrow \langle \$, \$ \rangle \quad \mathbf{accept}$$

$$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle \\ \rightarrow \langle \$, b\$ \rangle \quad \mathbf{reject}$$

$$\langle S\$, aab\$ \rangle \rightarrow \langle aSb\$, aab\$ \rangle \rightarrow \langle Sb\$, ab\$ \rangle \rightarrow \langle aSbb\$, ab\$ \rangle \\ \rightarrow \langle Sbb\$, b\$ \rangle$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\begin{aligned} \langle S\$, ab\$ \rangle &\rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ &\rightarrow \langle \$, \$ \rangle \quad \mathbf{accept} \end{aligned}$$

$$\begin{aligned} \langle S\$, abb\$ \rangle &\rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle \\ &\rightarrow \langle \$, b\$ \rangle \quad \mathbf{reject} \end{aligned}$$

$$\begin{aligned} \langle S\$, aab\$ \rangle &\rightarrow \langle aSb\$, aab\$ \rangle \rightarrow \langle Sb\$, ab\$ \rangle \rightarrow \langle aSbb\$, ab\$ \rangle \\ &\rightarrow \langle Sbb\$, b\$ \rangle \rightarrow \langle bb\$, b\$ \rangle \end{aligned}$$



# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$$\begin{aligned} \langle S\$, ab\$ \rangle &\rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle \\ &\rightarrow \langle \$, \$ \rangle \quad \mathbf{accept} \end{aligned}$$

$$\begin{aligned} \langle S\$, abb\$ \rangle &\rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle \\ &\rightarrow \langle \$, b\$ \rangle \quad \mathbf{reject} \end{aligned}$$

$$\begin{aligned} \langle S\$, aab\$ \rangle &\rightarrow \langle aSb\$, aab\$ \rangle \rightarrow \langle Sb\$, ab\$ \rangle \rightarrow \langle aSbb\$, ab\$ \rangle \\ &\rightarrow \langle Sbb\$, b\$ \rangle \rightarrow \langle bb\$, b\$ \rangle \rightarrow \langle b\$, \$ \rangle \end{aligned}$$

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	<i>a</i>	<i>b</i>	<i>\$</i>
<i>S</i>	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle$   
 $\rightarrow \langle \$, \$ \rangle$  **accept**

$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle$   
 $\rightarrow \langle \$, b\$ \rangle$  **reject**

$\langle S\$, aab\$ \rangle \rightarrow \langle aSb\$, aab\$ \rangle \rightarrow \langle Sb\$, ab\$ \rangle \rightarrow \langle aSbb\$, ab\$ \rangle$   
 $\rightarrow \langle Sbb\$, b\$ \rangle \rightarrow \langle bb\$, b\$ \rangle \rightarrow \langle b\$, \$ \rangle$  **reject**

# Example

$$S \rightarrow aSb \mid \lambda$$

The parser table is:

	$a$	$b$	$\$$
$S$	$S \rightarrow aSb$	$S \rightarrow \lambda$	$S \rightarrow \lambda$

$\langle S\$, ab\$ \rangle \rightarrow \langle aSb\$, ab\$ \rangle \rightarrow \langle Sb\$, b\$ \rangle \rightarrow \langle b\$, b\$ \rangle$   
 $\rightarrow \langle \$, \$ \rangle$  **accept**

$\langle S\$, abb\$ \rangle \rightarrow \langle aSb\$, abb\$ \rangle \rightarrow \langle Sb\$, bb\$ \rangle \rightarrow \langle b\$, bb\$ \rangle$   
 $\rightarrow \langle \$, b\$ \rangle$  **reject**

$\langle S\$, aab\$ \rangle \rightarrow \langle aSb\$, aab\$ \rangle \rightarrow \langle Sb\$, ab\$ \rangle \rightarrow \langle aSbb\$, ab\$ \rangle$   
 $\rightarrow \langle Sbb\$, b\$ \rangle \rightarrow \langle bb\$, b\$ \rangle \rightarrow \langle b\$, \$ \rangle$  **reject**

JavaCC (Java Compiler Compiler) automatically generates a parser from an LL(1) grammar.

# Left Factorisation

**Left factorisation:** rewrite rules  $A \rightarrow uv \mid uw$  ( $u \neq \lambda$ ) into

$$A \rightarrow uB \quad \text{and} \quad B \rightarrow v \mid w$$

where  $B$  is a fresh variable.

# Left Factorisation

**Left factorisation:** rewrite rules  $A \rightarrow uv \mid uw$  ( $u \neq \lambda$ ) into

$$A \rightarrow uB \quad \text{and} \quad B \rightarrow v \mid w$$

where  $B$  is a fresh variable.

The grammar  $S \rightarrow ab \mid ac$  is **not** LL(1):

	$a$	$b$	$c$	$\$$
$S$	$S \rightarrow ab$			
			$S \rightarrow ac$	

# Left Factorisation

**Left factorisation:** rewrite rules  $A \rightarrow uv \mid uw$  ( $u \neq \lambda$ ) into

$$A \rightarrow uB \quad \text{and} \quad B \rightarrow v \mid w$$

where  $B$  is a fresh variable.

The grammar  $S \rightarrow ab \mid ac$  is **not** LL(1):

	$a$	$b$	$c$	$\$$
$S$	$S \rightarrow ab$			
			$S \rightarrow ac$	

After left factorisation we get:  $S \rightarrow aA$   $A \rightarrow b \mid c$

The grammar  $S \rightarrow aA, A \rightarrow b \mid c$  **is** LL(1):

	$a$	$b$	$c$	$\$$
$S$	$S \rightarrow aA$			
$A$		$A \rightarrow b$	$A \rightarrow c$	

# LL( $k$ ) Grammars

The class of LL(1) grammars is often too restrictive in practice.

## LL( $k$ ) Grammars

The class of LL(1) grammars is often too restrictive in practice.

LL(1) parsers look at 1 symbol to decide which rule to use.



# LL( $k$ ) Grammars

The class of LL(1) grammars is often too restrictive in practice.

LL(1) parsers look at 1 symbol to decide which rule to use.

An LL( $k$ ) parser looks  $k$  symbols ahead to choose the rule.

# LL( $k$ ) Grammars

The class of LL(1) grammars is often too restrictive in practice.

LL(1) parsers look at 1 symbol to decide which rule to use.

An LL( $k$ ) parser looks  $k$  symbols ahead to choose the rule.

The parser table is constructed with  $k$  symbols look-ahead.

# LL( $k$ ) Grammars

The class of LL(1) grammars is often too restrictive in practice.

LL(1) parsers look at 1 symbol to decide which rule to use.

An LL( $k$ ) parser looks  $k$  symbols ahead to choose the rule.

The parser table is constructed with  $k$  symbols look-ahead.

A grammar is LL( $k$ ) if this table has in every cell  $\leq 1$  rule.

# LL( $k$ ) Grammars

The class of LL(1) grammars is often too restrictive in practice.

LL(1) parsers look at 1 symbol to decide which rule to use.

An LL( $k$ ) parser looks  $k$  symbols ahead to choose the rule.

The parser table is constructed with  $k$  symbols look-ahead.

A grammar is LL( $k$ ) if this table has in every cell  $\leq 1$  rule.

LL( $k$ ) is strictly contained in LL( $k + 1$ ).

# LL( $k$ ) Grammars

The class of LL(1) grammars is often too restrictive in practice.  
LL(1) parsers look at 1 symbol to decide which rule to use.

An LL( $k$ ) parser looks  $k$  symbols ahead to choose the rule.  
The parser table is constructed with  $k$  symbols look-ahead.  
A grammar is LL( $k$ ) if this table has in every cell  $\leq 1$  rule.

LL( $k$ ) is strictly contained in LL( $k + 1$ ).

**Disadvantage:** size of the parser table grows exponential in  $k$ .

# Exercises

Explain why these grammars cannot be  $LL(k)$  (for  $k \geq 1$ ):

- ambiguous grammars
- $S \rightarrow aSa \mid \lambda$

# Looking Back

- Context-free languages via context-free grammars
- Grammar simplifications & Chomsky normal form
- CYK parsing (bottom-up)
- LL and LL(1) parsing (top-down)

# Looking Forward

## Read:

- Linz 5.1–5.2, 6.1–6.3
- Grune & Jacobs H8 (LL parsing)

## Do the following exercises:

- Linz 5.1: 2, 7b,e, 8b,h, 21, 22
- Linz 5.2: 14
- Linz 6.1: 6, 7, 9, 15, 16
- Linz 6.2: 3, 4
- Linz 6.3: 1, 2, 3
- Extra exercise about LL parsing

## Following lecture:

- (Nondeterministic) pushdown automata