Syntax Analysis

syntax analysis
covers
lecture
This.

- Words (tokens) need to appear in the right order to form correct sentences (programmes)

- Syntax analyser commonly called *parser*.
- Analyses if *token* sequence forms declarations, statements, etc..
- Essential tool and theory used here: *Context-free languages*.

Syntax Analysis

syntax analysis
covers
lecture
This.
Syntax error!

- Words (tokens) need to appear in the right order to form correct sentences (programmes)
- Syntax analyser commonly called *parser*.
- Analyses if *token* sequence forms declarations, statements, etc..
- Essential tool and theory used here: *Context-free languages*.

Syntax Analysis

syntax analysis	This analysis
covers	covers
lecture	lecture
This.	syntax.

Syntax error!

- Words (tokens) need to appear in the right order to form correct sentences (programmes) (but not necessarily meaningful ones)
- Syntax analyser commonly called *parser*.
- Analyses if *token* sequence forms declarations, statements, etc..
- Essential tool and theory used here: *Context-free languages*.

Syntax Analysis

syntax analysis covers lecture This. Syntax error!	This analysis covers lecture syntax. (semantic error)

- Words (tokens) need to appear in the right order to form correct sentences (programmes) (but not necessarily meaningful ones)
- Syntax analyser commonly called *parser*.
- Analyses if *token* sequence forms declarations, statements, etc..
- Essential tool and theory used here: *Context-free languages*.

Context-Free Grammars

Definition (Context-Free Grammar)

A *context-free grammar* is given by

- a set of *terminals* Σ (the alphabet of the resulting language),
- a set of *nonterminals* N,
- a start symbol $S \in N$
- a set P of *productions* $X \to \alpha$ with a single nonterminal $X \in N$ on the left and a (possibly empty) right-hand side $\alpha \in (\Sigma \cup N)^*$ of terminals and nonterminals.

$$\begin{aligned} G : S &\to aSB \\ S &\to \varepsilon \\ B &\to Bb \\ B &\to b \end{aligned}$$

- Context-free grammars describe (context-free) languages over their terminal alphabet Σ.
- Each nonterminal describes a set of words.
- Nonterminals *recursively* refer to each other. (cannot do that with regular expressions)

Context-Free Grammars

Definition (Context-Free Grammar)

A *context-free grammar* is given by

- a set of *terminals* Σ (the alphabet of the resulting language),
- a set of *nonterminals* N,
- a start symbol $S \in N$
- a set P of *productions* $X \to \alpha$ with a single nonterminal $X \in N$ on the left and a (possibly empty) right-hand side $\alpha \in (\Sigma \cup N)^*$ of terminals and nonterminals.

$$
\begin{aligned}
G : S &\to aSB \\
S &\to \varepsilon \\
B &\to Bb \\
B &\to b
\end{aligned}
$$

- Context-free grammars describe (context-free) languages over their terminal alphabet Σ.
- Each nonterminal describes a set of words.
- Nonterminals *recursively* refer to each other. (cannot do that with regular expressions)

Context-Free Grammars

Definition (Context-Free Grammar)

A *context-free grammar* is given by

- a set of *terminals* Σ (the alphabet of the resulting language),
- a set of *nonterminals* N,
- a start symbol $S \in N$
- a set P of *productions* $X \to \alpha$ with a single nonterminal $X \in N$ on the left and a (possibly empty) right-hand side $\alpha \in (\Sigma \cup N)^*$ of terminals and nonterminals.

$$
\begin{aligned}
G : S &\to aSB \mid \varepsilon \\
B &\to Bb \mid b
\end{aligned}
$$

- Context-free grammars describe (context-free) languages over their terminal alphabet Σ.
- Each nonterminal describes a set of words.
- Nonterminals *recursively* refer to each other. (cannot do that with regular expressions)

Example, Derivation of Words

$$
\begin{array}{rcll}
G : S & \rightarrow & aSB & (1) \\
S & \rightarrow & \varepsilon & (2) \\
B & \rightarrow & Bb & (3) \\
B & \rightarrow & b & (4)
\end{array}
$$

$$
\begin{array}{rcl}
S & = & \underbrace{\{\varepsilon\}}_{(2)} \cup \underbrace{\{\mathrm{a} \cdot x \cdot y \mid x \in S, y \in B\}}_{(1)} \\
B & = & \underbrace{\{\mathrm{b}\}}_{(4)} \cup \underbrace{\{x \cdot \mathrm{b} \mid x \in B\}}_{(3)}
\end{array}
$$

- Starting from the start symbol S,...
- words of the language can be *derived*...
- by successively replacing nonterminals with right-hand sides.

$$
\begin{array}{l}
S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{4}{\Rightarrow} \mathrm{aa}S\underline{\mathrm{b}}B \overset{1}{\Rightarrow} \mathrm{aa}\underline{\mathrm{a}SB}\mathrm{b}B \\
\quad \overset{2}{\Rightarrow} \mathrm{aaa}\underline{\;}B\mathrm{b}B \overset{3}{\Rightarrow} \mathrm{aaa}\underline{B\mathrm{b}}\mathrm{b}B \overset{4}{\Rightarrow} \mathrm{aaa}B\mathrm{bb}\underline{\mathrm{b}} \overset{4}{\Rightarrow} \mathrm{aaa}\underline{\mathrm{b}}\mathrm{bbb}
\end{array}
$$

Example, Derivation of Words

$$\begin{aligned} G: S &\rightarrow aSB(1) \\ S &\rightarrow \varepsilon \quad (2) \\ B &\rightarrow Bb \ (3) \\ B &\rightarrow b \quad (4) \end{aligned}$$

$$S = \underbrace{\{\varepsilon\}}_{(2)} \cup \underbrace{\{\mathsf{a} \cdot x \cdot y \mid x \in S, y \in B\}}_{(1)}$$

$$B = \underbrace{\{\mathsf{b}\}}_{(4)} \cup \underbrace{\{x \cdot \mathsf{b} \mid x \in B\}}_{(3)}$$

- Starting from the start symbol S,...
- words of the language can be *derived*...
- by successively replacing nonterminals with right-hand sides.

$$S \overset{1}{\Rightarrow} \underline{\mathsf{a}SB} \overset{1}{\Rightarrow} \mathsf{a}\underline{\mathsf{a}SB}B \overset{4}{\Rightarrow} \mathsf{aa}S\underline{\mathsf{b}}B \overset{1}{\Rightarrow} \mathsf{aa}\underline{\mathsf{a}SB}\mathsf{b}B$$
$$\overset{2}{\Rightarrow} \mathsf{aaa}\_B\mathsf{b}B \overset{3}{\Rightarrow} \mathsf{aaa}\underline{B\mathsf{b}}\mathsf{b}B \overset{4}{\Rightarrow} \mathsf{aaa}B\mathsf{bb}\underline{\mathsf{b}} \overset{4}{\Rightarrow} \mathsf{aaa}\underline{\mathsf{b}}\mathsf{bbb}$$

Example, Derivation of Words

$$
\begin{aligned}
G : S &\rightarrow aSB\,(1)\\
S &\rightarrow \varepsilon \quad (2)\\
B &\rightarrow Bb \ (3)\\
B &\rightarrow b \quad (4)
\end{aligned}
$$

$$
\begin{aligned}
S &= \underbrace{\{\varepsilon\}}_{(2)} \cup \underbrace{\{\mathrm{a} \cdot x \cdot y \mid x \in S, y \in B\}}_{(1)}\\
B &= \underbrace{\{\mathrm{b}\}}_{(4)} \cup \underbrace{\{x \cdot \mathrm{b} \mid x \in B\}}_{(3)}
\end{aligned}
$$

- Starting from the start symbol S,...
- words of the language can be *derived*...
- by successively replacing nonterminals with right-hand sides.

$$
S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{4}{\Rightarrow} \mathrm{aa}S\underline{\mathrm{b}}B \overset{1}{\Rightarrow} \mathrm{aa}\underline{\mathrm{a}SB}\mathrm{b}B
$$
$$
\overset{2}{\Rightarrow} \mathrm{aaa}\_B\mathrm{b}B \overset{3}{\Rightarrow} \mathrm{aaa}\underline{B\mathrm{b}}\mathrm{b}B \overset{4}{\Rightarrow} \mathrm{aaa}B\mathrm{bb}\underline{\mathrm{b}} \overset{4}{\Rightarrow} \mathrm{aaa}\underline{\mathrm{b}}\mathrm{bbb}
$$

Example, Derivation of Words

$$
\begin{array}{rcll}
G : S & \rightarrow & aSB & (1) \\
S & \rightarrow & \varepsilon & (2) \\
B & \rightarrow & Bb & (3) \\
B & \rightarrow & b & (4)
\end{array}
\qquad
\begin{array}{rcl}
S & = & \underbrace{\{\varepsilon\}}_{(2)} \cup \underbrace{\{\mathrm{a} \cdot x \cdot y \mid x \in S, y \in B\}}_{(1)} \\
B & = & \underbrace{\{\mathrm{b}\}}_{(4)} \cup \underbrace{\{x \cdot \mathrm{b} \mid x \in B\}}_{(3)}
\end{array}
$$

- Starting from the start symbol S,...
- words of the language can be *derived*...
- by successively replacing nonterminals with right-hand sides.

$$
S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{4}{\Rightarrow} \mathrm{aa}S\underline{\mathrm{b}}B \overset{1}{\Rightarrow} \mathrm{aa}\underline{\mathrm{a}SB}\mathrm{b}B
$$

$$
\overset{2}{\Rightarrow} \mathrm{aaa}\underline{\ }B\mathrm{b}B \overset{3}{\Rightarrow} \mathrm{aaa}\underline{B\mathrm{b}}\mathrm{b}B \overset{4}{\Rightarrow} \mathrm{aaa}B\mathrm{bb}\underline{\mathrm{b}} \overset{4}{\Rightarrow} \mathrm{aaa}\underline{\mathrm{b}}\mathrm{bbb}
$$

Example, Derivation of Words

$$
\begin{aligned}
G: S &\rightarrow aSB (1)\\
S &\rightarrow \varepsilon \quad (2)\\
B &\rightarrow Bb \; (3)\\
B &\rightarrow b \quad (4)
\end{aligned}
$$

$$
\begin{aligned}
S &= \underbrace{\{\varepsilon\}}_{(2)} \cup \underbrace{\{\mathrm{a} \cdot x \cdot y \mid x \in S, y \in B\}}_{(1)}\\
B &= \underbrace{\{\mathrm{b}\}}_{(4)} \cup \underbrace{\{x \cdot \mathrm{b} \mid x \in B\}}_{(3)}
\end{aligned}
$$

- Starting from the start symbol S,...
- words of the language can be *derived*...
- by successively replacing nonterminals with right-hand sides.

$$
S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{4}{\Rightarrow} \mathrm{aa}S\underline{\mathrm{b}}B \overset{1}{\Rightarrow} \mathrm{aa}\underline{\mathrm{a}SB}\mathrm{b}B
$$

$$
\overset{2}{\Rightarrow} \mathrm{aaa}\_B\mathrm{b}B \overset{3}{\Rightarrow} \mathrm{aaa}\underline{B\mathrm{b}}\mathrm{b}B \overset{4}{\Rightarrow} \mathrm{aaa}B\mathrm{bb}\underline{\mathrm{b}} \overset{4}{\Rightarrow} \mathrm{aaa}\underline{\mathrm{b}}\mathrm{bbb}
$$

Example, Derivation of Words

$$
\begin{array}{rcll}
G: S & \rightarrow & aSB & (1) \\
S & \rightarrow & \varepsilon & (2) \\
B & \rightarrow & Bb & (3) \\
B & \rightarrow & b & (4)
\end{array}
$$

$$
\begin{array}{rcl}
S & = & \underbrace{\{\varepsilon\}}_{(2)} \cup \underbrace{\{\mathrm{a} \cdot x \cdot y \mid x \in S, y \in B\}}_{(1)} \\
B & = & \underbrace{\{\mathrm{b}\}}_{(4)} \cup \underbrace{\{x \cdot \mathrm{b} \mid x \in B\}}_{(3)}
\end{array}
$$

- Starting from the start symbol S,...
- words of the language can be *derived*...
- by successively replacing nonterminals with right-hand sides.

$$
\begin{array}{l}
S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{4}{\Rightarrow} \mathrm{aa}S\underline{\mathrm{b}}B \overset{1}{\Rightarrow} \mathrm{aa}\underline{\mathrm{a}SB}\mathrm{b}B \\
\quad \overset{2}{\Rightarrow} \mathrm{aaa}\underline{\ }B\mathrm{b}B \overset{3}{\Rightarrow} \mathrm{aaa}\underline{B\mathrm{b}}\mathrm{b}B \overset{4}{\Rightarrow} \mathrm{aaa}B\mathrm{bb}\underline{\mathrm{b}} \overset{4}{\Rightarrow} \mathrm{aaa}\underline{\mathrm{b}}\mathrm{bbb}
\end{array}
$$

Example, Derivation of Words

$$\begin{array}{rcll} G: S & \rightarrow & aSB & (1) \\ S & \rightarrow & \varepsilon & (2) \\ B & \rightarrow & Bb & (3) \\ B & \rightarrow & b & (4) \end{array}$$

$$\begin{array}{rcl} S & = & \underbrace{\{\varepsilon\}}_{(2)} \cup \underbrace{\{\mathrm{a} \cdot x \cdot y \mid x \in S, y \in B\}}_{(1)} \\ B & = & \underbrace{\{\mathrm{b}\}}_{(4)} \cup \underbrace{\{x \cdot \mathrm{b} \mid x \in B\}}_{(3)} \end{array}$$

- Starting from the start symbol S,...
- words of the language can be *derived*...
- by successively replacing nonterminals with right-hand sides.

$$\begin{array}{l} S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{4}{\Rightarrow} \mathrm{aa}S\underline{\mathrm{b}}B \overset{1}{\Rightarrow} \mathrm{aa}\underline{\mathrm{a}SB}\mathrm{b}B \\ \quad \overset{2}{\Rightarrow} \mathrm{aaa}\underline{\ }B\mathrm{b}B \overset{3}{\Rightarrow} \mathrm{aaa}\underline{B\mathrm{b}}\mathrm{b}B \overset{4}{\Rightarrow} \mathrm{aaa}B\mathrm{bb}\underline{\mathrm{b}} \overset{4}{\Rightarrow} \mathrm{aaa}\underline{\mathrm{b}}\mathrm{bbb} \end{array}$$

Derivation Relation

Definition (Derivation $\Rightarrow$)

Let $G = (\Sigma, N, S, P)$ be a grammar.
The *derivation relation* $\Rightarrow$ on $(\Sigma \cup N)^*$ is defined as follows:

- For an $X \in N$ and a production $(X \to \beta) \in P$ of the grammar, $\underline{\alpha_1 X \alpha_2 \Rightarrow \alpha_1 \beta \alpha_2}$ for all $\alpha_1, \alpha_2 \in (\Sigma \cup N)^*$.

- Describes one derivation step using one of the productions.
- If productions numbered: can add *used production* number ($\stackrel{k}{\Rightarrow}$).
- *left-most* (or *right-most*) derivation indicated by subscript, $\stackrel{k}{\Rightarrow}_l$.

$$
\begin{aligned}
G: S &\to aSB \quad (1) \\
S &\to \varepsilon \quad (2) \\
B &\to Bb \quad (3) \\
B &\to b \quad (4)
\end{aligned}
$$

$$
S \stackrel{1}{\Rightarrow} \underline{aSB} \stackrel{1}{\Rightarrow} a\underline{aSB}B \stackrel{2}{\Rightarrow} aa\underline{\ }BB \stackrel{3}{\Rightarrow} aa\underline{Bb}B \stackrel{4}{\Rightarrow} aa\underline{b}bB \stackrel{4}{\Rightarrow} aabb\underline{b}
$$

Derivation Relation

Definition (Derivation $\Rightarrow$)

Let $G = (\Sigma, N, S, P)$ be a grammar.
The *derivation relation* $\Rightarrow$ on $(\Sigma \cup N)^*$ is defined as follows:

- For an $X \in N$ and a production $(X \to \beta) \in P$ of the grammar, $\underline{\alpha_1 X \alpha_2 \Rightarrow \alpha_1 \beta \alpha_2}$ for all $\alpha_1, \alpha_2 \in (\Sigma \cup N)^*$.

- Describes one derivation step using one of the productions.
- If productions numbered: can add *used production* number ($\overset{k}{\Rightarrow}$).
- *left-most* (or *right-most*) derivation indicated by subscript, $\overset{k}{\Rightarrow}_l$.

$$
\begin{aligned}
G : S &\to aSB \quad (1)\\
S &\to \varepsilon \quad\;\; (2)\\
B &\to Bb \quad (3)\\
B &\to b \quad\;\; (4)
\end{aligned}
$$

$$S \overset{1}{\Rightarrow} \underline{aSB} \overset{1}{\Rightarrow} a\underline{aSB}B \overset{2}{\Rightarrow} aa\_BB$$
$$\overset{3}{\Rightarrow} aa\underline{Bb}B \overset{4}{\Rightarrow} aa\underline{b}bB \overset{4}{\Rightarrow} aabb\underline{b}$$

Derivation Relation

Definition (Derivation $\Rightarrow$)

Let $G = (\Sigma, N, S, P)$ be a grammar.
The *derivation relation* $\Rightarrow$ on $(\Sigma \cup N)^*$ is defined as follows:

- For an $X \in N$ and a production $(X \to \beta) \in P$ of the grammar, $\underline{\alpha_1 X \alpha_2 \Rightarrow \alpha_1 \beta \alpha_2}$ for all $\alpha_1, \alpha_2 \in (\Sigma \cup N)^*$.

- Describes one derivation step using one of the productions.
- If productions numbered: can add *used production* number ($\stackrel{k}{\Rightarrow}$).
- *left-most* (or *right-most*) derivation indicated by subscript, $\stackrel{k}{\Rightarrow}_l$.

$$
\begin{array}{rcll}
G : S & \to & aSB & (1) \\
S & \to & \varepsilon & (2) \\
B & \to & Bb & (3) \\
B & \to & b & (4)
\end{array}
$$

$$
\begin{array}{l}
S \stackrel{1}{\Rightarrow} \underline{aSB} \stackrel{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \stackrel{2}{\Rightarrow} \mathrm{aa}\_BB \\
\quad \stackrel{3}{\Rightarrow} \mathrm{aa}\underline{B\mathrm{b}}B \stackrel{4}{\Rightarrow} \mathrm{aa}\underline{\mathrm{b}}\mathrm{b}B \stackrel{4}{\Rightarrow} \mathrm{aabb}\underline{\mathrm{b}}
\end{array}
$$

Extended Derivation Relation (Transitive Closure)

Definition (Transitive Derivation Relation $\Rightarrow^*$)

Let $G = (\Sigma, N, S, P)$ be a grammar and $\Rightarrow$ its derivation relation. The *transitive derivation relation* of G is defined as:

- $\alpha \Rightarrow^* \alpha$ for all $\alpha \in (\Sigma \cup N)^*$ (derived in 0 steps).
- For $\alpha, \beta \in (\Sigma \cup N)^*$, $\alpha \Rightarrow^* \beta$ if there exists a $\gamma \in (\Sigma \cup N)^*$ such that $\alpha \Rightarrow \gamma$ and $\gamma \Rightarrow^* \beta$ (derived in at least one step).

More generally, this is known as the *transitive closure* of a relation.
In our previous examples, we saw $S \Rightarrow^*$ aaabbbb and $S \Rightarrow^*$ aabbb. That means, both words are *in the language of G*.

Definition (Language of a Grammar)

Let $G = (\Sigma, N, S, P)$ be a grammar and $\Rightarrow$ its derivation relation. The language of the grammar is $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

Extended Derivation Relation (Transitive Closure)

Definition (Transitive Derivation Relation $\Rightarrow^*$)

Let $G = (\Sigma, N, S, P)$ be a grammar and $\Rightarrow$ its derivation relation. The <u>*transitive derivation relation* of G</u> is defined as:

- $\alpha \Rightarrow^* \alpha$ for all $\alpha \in (\Sigma \cup N)^*$ (derived in 0 steps).
- For $\alpha, \beta \in (\Sigma \cup N)^*$, $\alpha \Rightarrow^* \beta$ if there exists a $\gamma \in (\Sigma \cup N)^*$ such that $\alpha \Rightarrow \gamma$ and $\gamma \Rightarrow^* \beta$ (derived in at least one step).

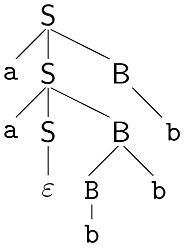
More generally, this is known as the *transitive closure* of a relation. In our previous examples, we saw $S \Rightarrow^*$ aaabbbb and $S \Rightarrow^*$ aabbb. That means, both words are *in the language of G*.

Definition (Language of a Grammar)

Let $G = (\Sigma, N, S, P)$ be a grammar and $\Rightarrow$ its derivation relation. The language of the grammar is $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

Syntax Tree and Directed Derivation

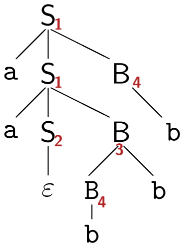
$$
\begin{array}{rcll}
G : S & \rightarrow & aSB & (1) \\
S & \rightarrow & \varepsilon & (2) \\
B & \rightarrow & Bb & (3) \\
B & \rightarrow & b & (4)
\end{array}
$$



- Syntax trees describe the derivation independent of the direction.
- Left-most derivation: *depth-first left-to-right* tree *traversal*.
- $S \overset{1}{\Rightarrow} \underline{aSB} \overset{1}{\Rightarrow} a\underline{aSB}B \overset{2}{\Rightarrow} aa\_BB \overset{3}{\Rightarrow} aa\underline{Bb}B \overset{4}{\Rightarrow} aa\underline{b}bB \overset{4}{\Rightarrow} aabb\underline{b}$

Syntax Tree and Directed Derivation

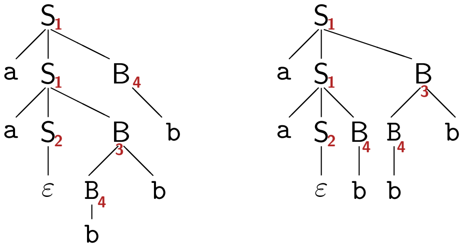
$$\begin{array}{lcll} G: S & \rightarrow & aSB & (1) \\ S & \rightarrow & \varepsilon & (2) \\ B & \rightarrow & Bb & (3) \\ B & \rightarrow & b & (4) \end{array}$$



- Syntax trees describe the derivation independent of the direction.
- Left-most derivation: *depth-first left-to-right* tree *traversal*.
- $S \stackrel{1}{\Rightarrow} \underline{\mathrm{a}SB} \stackrel{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \stackrel{2}{\Rightarrow} \mathrm{aa}\_BB \stackrel{3}{\Rightarrow} \mathrm{aa}\underline{B\mathrm{b}}B \stackrel{4}{\Rightarrow} \mathrm{aa}\underline{\mathrm{b}}\mathrm{b}B \stackrel{4}{\Rightarrow} \mathrm{aabb}\underline{\mathrm{b}}$

Syntax Tree and Directed Derivation

$$
\begin{array}{llllr}
G: S & \rightarrow & aSB & (1) \\
S & \rightarrow & \varepsilon & (2) \\
B & \rightarrow & Bb & (3) \\
B & \rightarrow & b & (4)
\end{array}
$$



- Syntax trees describe the derivation independent of the direction.
- Left-most derivation: *depth-first left-to-right* tree *traversal*.
- $S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{2}{\Rightarrow} \mathrm{aa}\underline{\ }BB \overset{3}{\Rightarrow} \mathrm{aa}\underline{B\mathrm{b}}B \overset{4}{\Rightarrow} \mathrm{aa}\underline{\mathrm{b}}\mathrm{b}B \overset{4}{\Rightarrow} \mathrm{aabb}\underline{\mathrm{b}}$

Nevertheless: $S \Rightarrow^{*}$ aabbb can be derived in two ways.

- $S \overset{1}{\Rightarrow} \underline{\mathrm{a}SB} \overset{1}{\Rightarrow} \mathrm{a}\underline{\mathrm{a}SB}B \overset{2}{\Rightarrow} \mathrm{aa}\underline{\ }BB \overset{4}{\Rightarrow} \mathrm{aa}\underline{\mathrm{b}}B \overset{3}{\Rightarrow} \mathrm{aab}\underline{\mathrm{b}B} \overset{4}{\Rightarrow} \mathrm{aabb}\underline{\mathrm{b}}$

The grammar G is said to be *ambiguous*.

Handling or Removing Ambiguity

$$
\begin{array}{lcl}
E & \rightarrow & E + E \mid E - E \\
E & \rightarrow & E * E \mid E / E \\
E & \rightarrow & \mathsf{a} \mid (E)
\end{array}
$$

- In many cases, grammars are rewritten to remove ambiguity.
- Sometimes, ambiguity is resolved by changes in the parser.

- In both cases: *Precedence* and *associativity* guide decisions.

Handling or Removing Ambiguity

$$
\begin{array}{lcl}
E & \rightarrow & E + E \mid E - E \\
E & \rightarrow & E * E \mid E / E \\
E & \rightarrow & \mathsf{a} \mid (E)
\end{array}
$$

- In many cases, grammars are rewritten to remove ambiguity.
- Sometimes, ambiguity is resolved by changes in the parser.

- In both cases: *Precedence* and *associativity* guide decisions.

Problems with this grammar:

1. Ambiguous derivation of `a - a - a`.
   Want a *left-associative* interpretation, $(a - a) - a$.
2. Ambiguous derivation of `a - a * a`.
   Want *precedence* of `*` over `+`, $a + (a \cdot a)$.

Establishing the Intended Operator Precedence

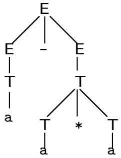
- Introduce *precedence levels* to get operator priorities
- New Grammar: *own nonterminal for each level*
- Here: 2 levels, mathematical interpretation:
  $a - a \cdot a = a - (a \cdot a)$ *Precedence* of * and / over + and -.
  More precedence levels could be added (exponentiation).

$$
\begin{array}{lcl}
E & \rightarrow & E + E \mid E - E \\
E & \rightarrow & E * E \mid E / E \\
E & \rightarrow & \mathrm{a} \mid (E)
\end{array}
$$

Establishing the Intended Operator Precedence

- Introduce *precedence levels* to get operator priorities
- New Grammar: *own nonterminal for each level*
- Here: 2 levels, mathematical interpretation:
  $a - a \cdot a = a - (a \cdot a)$ *Precedence* of * and / over + and -.
  More precedence levels could be added (exponentiation).

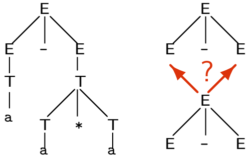
$$
\begin{aligned}
E &\rightarrow E + E \mid E - E \\
E &\rightarrow E * E \mid E / E \\
E &\rightarrow \text{a} \mid (E)
\end{aligned}
\qquad\qquad
\begin{aligned}
E &\rightarrow E + E \mid E - E \mid T \\
T &\rightarrow T * T \mid T / T \\
T &\rightarrow \text{a} \mid (E)
\end{aligned}
$$



Establishing the Intended Operator Precedence

- Introduce *precedence levels* to get operator priorities
- New Grammar: *own nonterminal for each level*
- Here: 2 levels, mathematical interpretation:
  $a - a \cdot a = a - (a \cdot a)$ *Precedence* of * and / over + and -.
  More precedence levels could be added (exponentiation).

$$
\begin{array}{lcl}
E & \rightarrow & E + E \mid E - E \\
E & \rightarrow & E * E \mid E / E \\
E & \rightarrow & \mathrm{a} \mid (E)
\end{array}
\qquad\qquad
\begin{array}{lcl}
E & \rightarrow & E + E \mid E - E \mid T \\
T & \rightarrow & T * T \mid T / T \\
T & \rightarrow & \mathrm{a} \mid (E)
\end{array}
$$



About Operator Associativity

Definition (Operator Associativity)

A binary operator $\oplus$ is called

- *left-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from left to right, as $(a \oplus b) \oplus c$.
- *right-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from right to left, as $a \oplus (b \oplus c)$.
- *non-associative*, if expressions $a \oplus b \oplus c$ are disallowed, (and *associative*, if both directions lead to the same result).

Examples:

- Arithmetic operators like `-` and `/`: left-associative.
- List constructors in functional languages: right-associative.
- Function arrows in types: right-associative.
- 'less-than' ($<$) in C:

```
if (3 < 2 < 1) { fprintf(stdout, "Awesome!\n"); }
```

About Operator Associativity

Definition (Operator Associativity)

A binary operator $\oplus$ is called

- *left-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from left to right, as $(a \oplus b) \oplus c$.
- *right-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from right to left, as $a \oplus (b \oplus c)$.
- *non-associative*, if expressions $a \oplus b \oplus c$ are disallowed, (and *associative*, if both directions lead to the same result).

Examples:

- Arithmetic operators like `-` and `/`: left-associative.
- List constructors in functional languages: right-associative.
- Function arrows in types: right-associative.
- 'less-than' ($<$) in C:

```
if (3 < 2 < 1) { fprintf(stdout, "Awesome!\n"); }
```

About Operator Associativity

Definition (Operator Associativity)

A binary operator $\oplus$ is called

- *left-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from left to right, as $(a \oplus b) \oplus c$.
- *right-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from right to left, as $a \oplus (b \oplus c)$.
- *non-associative*, if expressions $a \oplus b \oplus c$ are disallowed, (and *associative*, if both directions lead to the same result).

Examples:

- Arithmetic operators like `-` and `/`: left-associative.
- List constructors in functional languages: right-associative.
- Function arrows in types: right-associative.
- 'less-than' ($<$) in C:

```
if (3 < 2 < 1) { fprintf(stdout, "Awesome!\n"); }
```

About Operator Associativity

Definition (Operator Associativity)

A binary operator $\oplus$ is called

- *left-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from left to right, as $(a \oplus b) \oplus c$.
- *right-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from right to left, as $a \oplus (b \oplus c)$.
- *non-associative*, if expressions $a \oplus b \oplus c$ are disallowed, (and *associative*, if both directions lead to the same result).

Examples:

- Arithmetic operators like `-` and `/`: left-associative.
- List constructors in functional languages: right-associative.
- Function arrows in types: right-associative.
- 'less-than' ($<$) in C:

```
if (3 < 2 < 1) { fprintf(stdout, "Awesome!\n"); }
```

About Operator Associativity

Definition (Operator Associativity)

A binary operator $\oplus$ is called

- *left-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from left to right, as $(a \oplus b) \oplus c$.
- *right-associative*, if the expression $a \oplus b \oplus c$ should be evaluated from right to left, as $a \oplus (b \oplus c)$.
- *non-associative*, if expressions $a \oplus b \oplus c$ are disallowed, (and *associative*, if both directions lead to the same result).

Examples:

- Arithmetic operators like `-` and `/`: left-associative.
- List constructors in functional languages: right-associative.
- Function arrows in types: right-associative.
- 'less-than' ($<$) in C: left-associative

```
if (3 < 2 < 1) { fprintf(stdout, "Awesome!\n"); }
```

Establishing the Intended Associativity

- *limit recursion* to the intended side
- When operators are indeed *associative*, use same associativity as comparable operators.
- *Cannot mix* left- and right-associative operators at same precedence level.

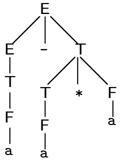
$$
\begin{aligned}
E &\rightarrow E + E \mid E - E \mid T \\
T &\rightarrow T * T \mid T / T \\
T &\rightarrow \mathrm{a} \mid (E)
\end{aligned}
$$

Establishing the Intended Associativity

- *limit recursion* to the intended side
- When operators are indeed *associative*, use same associativity as comparable operators.
- *Cannot mix* left- and right-associative operators at same precedence level.

$$
\begin{aligned}
E &\rightarrow E + E \mid E - E \mid T \\
T &\rightarrow T * T \mid T / T \\
T &\rightarrow \mathrm{a} \mid (E)
\end{aligned}
$$

$$
\begin{aligned}
E &\rightarrow E + T \mid E - T \mid T \\
T &\rightarrow T * F \mid T / F \mid F \\
F &\rightarrow \mathrm{a} \mid (E)
\end{aligned}
$$



Parsing

Token sequence

↓

Syntax analysis

↓

Syntax tree

- Producing a *syntax tree* from a token sequence.
- Representation of the tree: left-most or right-most derivation

Two approaches

- *Top-Down Parsing:* Builds syntax tree from the *root*.
  - Called *predictive parsing*: needs to "guess" used productions.
  - Builds a left-most derivation sequence
- *Bottom-Up Parsing:* Builds syntax tree from the *leaves*.
  - Called *shift/reduce parsing*: shifts input to stack,
  - reduces right-hand side to left-hand nonterminal,
  - until start symbol reached.
  - Builds a reversed right-most derivation sequence
- Both: use stack to keep track of derivation.

Parsing

Token sequence

$\downarrow$

$\boxed{\text{Syntax analysis}}$

$\downarrow$

Syntax tree

- Producing a *syntax tree* from a token sequence.
- Representation of the tree: left-most or right-most derivation

Two approaches

- *Top-Down Parsing:* Builds syntax tree from the *root*.
  - Called *predictive parsing*: needs to "guess" used productions.
  - Builds a left-most derivation sequence
- *Bottom-Up Parsing:* Builds syntax tree from the *leaves*.
  - Called *shift/reduce parsing*: shifts input to stack,
  - reduces right-hand side to left-hand nonterminal,
  - until start symbol reached.
  - Builds a reversed right-most derivation sequence
- Both: use stack to keep track of derivation.

Top-Down Parsing with look-ahead 1: "LL(1)"

- First, make the *grammar unambiguous*.

$$G' : S \rightarrow aSB \mid \varepsilon \qquad B \rightarrow Bb \mid b$$

$$G' : S \rightarrow aSb \mid B \quad (1,2) \qquad B \rightarrow bB \mid \varepsilon \quad (3,4)$$

Ambiguous!

OK (and same language).

- Compute information to *choose the right production*. For each right-hand side: *What input token can come first?*
- Special attention to empty right-hand sides. *What can follow?*
- Then, we can decide from a *look-ahead of one token*, which production to use. Choose a production $N \rightarrow \alpha$ if:
  - look-ahead c and $\alpha \Rightarrow^* $ something that starts with c.
  - look-ahead c , $\alpha \Rightarrow^* \varepsilon$ and c can follow N.

Top-Down Parsing with look-ahead 1: "LL(1)"

- First, make the *grammar unambiguous*.

$$G' : S \to aSB \mid \varepsilon \qquad B \to Bb \mid b$$

$$G' : S \to aSb \mid B \quad (1,2) \qquad B \to bB \mid \varepsilon \quad (3,4)$$

Ambiguous! OK (and same language).

- Compute information to *choose the right production*. For each right-hand side: *What input token can come first?*
- Special attention to empty right-hand sides. *What can follow?*
- Then, we can decide from a *look-ahead of one token*, which production to use. Choose a production $N \to \alpha$ if:
  - look-ahead c and $\alpha \Rightarrow^* $ something that starts with c.
  - look-ahead c , $\alpha \Rightarrow^* \varepsilon$ and c can follow N.

Top-Down Parsing with look-ahead 1: "LL(1)"

- First, make the *grammar unambiguous*.

$$G' : S \rightarrow aSB \mid \varepsilon \qquad B \rightarrow Bb \mid b$$

$$G' : S \rightarrow aSb \mid B \quad (1,2) \qquad B \rightarrow bB \mid \varepsilon \quad (3,4)$$

Ambiguous! OK (and same language).

- Compute information to *choose the right production*.
  For each right-hand side: *What input token can come first?*
- Special attention to empty right-hand sides. *What can follow?*
- Then, we can decide from a *look-ahead of one token*, which production to use. Choose a production $N \rightarrow \alpha$ if:
  - look-ahead c and $\alpha \Rightarrow^* $ something that starts with c.
  - look-ahead c , $\alpha \Rightarrow^* \varepsilon$ and c can follow N.

Top-Down Parsing with look-ahead 1: “LL(1)”

- First, make the *grammar unambiguous*.

$G' : S \;\rightarrow\; aSB \mid \varepsilon$ $\quad\; B \;\rightarrow\; Bb \mid b$	$G' : S \;\rightarrow\; aSb \mid B \quad (1,2)$ $\quad\; B \;\rightarrow\; bB \mid \varepsilon \quad (3,4)$
Ambiguous!	OK (and same language).

- Compute information to *choose the right production*.
  For each right-hand side: *What input token can come first?*
- Special attention to empty right-hand sides. *What can follow?*
- Then, we can decide from a *look-ahead of one token*, which production to use. Choose a production $N \rightarrow \alpha$ if:
  - look-ahead c and $\alpha \Rightarrow^* $ something that starts with c.
  - look-ahead c , $\alpha \Rightarrow^* \varepsilon$ and c can follow N.

Top-Down Parsing with look-ahead 1: "LL(1)"

- First, make the *grammar unambiguous*.

$$G' : S \rightarrow aSB \mid \varepsilon \qquad\qquad G' : S \rightarrow aSb \mid B \quad (1,2)$$
$$B \rightarrow Bb \mid b \qquad\qquad B \rightarrow bB \mid \varepsilon \quad (3,4)$$

Ambiguous! OK (and same language).

- Compute information to *choose the right production*.
  For each right-hand side: *What input token can come first?*
- Special attention to empty right-hand sides. *What can follow?*
- Then, we can decide from a *look-ahead of one token*, which production to use. Choose a production $N \rightarrow \alpha$ if:
  - look-ahead c and $\alpha \Rightarrow^* $ something that starts with c.
  - look-ahead c , $\alpha \Rightarrow^* \varepsilon$ and c can follow N.

Input: aabbb: $S \stackrel{1}{\Rightarrow} \underline{a}Sb \stackrel{1}{\Rightarrow} a\underline{a}Sbb \stackrel{2}{\Rightarrow} aa\underline{B}bb \stackrel{3}{\Rightarrow} aa\underline{b}Bbb \stackrel{4}{\Rightarrow} aabbb$

First Sets and Property Nullable

Definition (First set and Nullable)

Let $G = (\Sigma, N, S, P)$ a grammar and $\Rightarrow$ its derivation relation.
For all sequences of grammar symbols $\alpha \in (\Sigma \cup N)^*$, define

- $\text{FIRST}(\alpha) = \{c \in \Sigma \mid \exists_{\beta \in (\Sigma \cup N)^*} : \alpha \Rightarrow^* c\beta\}$
  (all input tokens at the start of what can be derived from α)
- $\text{NULLABLE}(\alpha) = \begin{cases} \textit{true} & \text{, if } \alpha \Rightarrow^* \varepsilon \\ \textit{false} & \text{, otherwise} \end{cases}$

Computing Nullable and First for right-hand sides:

- Set equations recursively use results for nonterminals.
- Smallest solution found by computing a smallest fixed-point.
- Solved simultaneously for *all right-hand sides* of the productions.

First Sets and Property Nullable

Definition (First set and Nullable)

Let $G = (\Sigma, N, S, P)$ a grammar and $\Rightarrow$ its derivation relation.
For all sequences of grammar symbols $\alpha \in (\Sigma \cup N)^*$, define

- $\textsc{First}(\alpha) = \{c \in \Sigma \mid \exists_{\beta \in (\Sigma \cup N)^*} : \ \alpha \Rightarrow^* c\beta\}$
  (all input tokens at the start of what can be derived from α)
- $\textsc{Nullable}(\alpha) = \begin{cases} \textit{true} & , \text{ if } \alpha \Rightarrow^* \varepsilon \\ \textit{false} & , \text{ otherwise} \end{cases}$

Computing Nullable and First for right-hand sides:

- Set equations recursively use results for nonterminals.
- Smallest solution found by computing a smallest fixed-point.
- Solved simultaneously for *all right-hand sides* of the productions.

Computing NULLABLE by Set Equations

$$
\begin{array}{lcl}
\text{NULLABLE}(\varepsilon) & = & \textit{true} \\
\text{NULLABLE}(\mathsf{a}) & = & \textit{false} \text{ for } \mathsf{a} \in \Sigma \\
\text{NULLABLE}(\alpha\beta) & = & \text{NULLABLE}(\alpha) \wedge \text{NULLABLE}(\beta) \text{ for } \alpha, \beta \in (\Sigma \cup N)^* \\
\text{NULLABLE}(N) & = & \text{NULLABLE}(\alpha_1) \vee \ldots \vee \text{NULLABLE}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{array}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl}
G' : S & \to & aSb \mid B \\
B & \to & bB \mid \varepsilon
\end{array}
\qquad
\begin{array}{lcl}
\text{NULLABLE}(S) & = & \text{NULLABLE}(aSb) \vee \text{NULLABLE}(B) \\
\text{NULLABLE}(B) & = & \text{NULLABLE}(bB) \vee \text{NULLABLE}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{array}{lcl}
\text{NULLABLE}(aSB) & = & \text{NULLABLE}(a) \wedge \text{NULLABLE}(S) \wedge \text{NULLABLE}(B) \\
\text{NULLABLE}(B) & = & \text{NULLABLE}(B) \\
\text{NULLABLE}(bB) & = & \text{NULLABLE}(b) \wedge \text{NULLABLE}(B) \\
\text{NULLABLE}(\varepsilon) & = & \textit{true}
\end{array}
$$

Compute smallest solution of system, starting by *false* for all.

Computing NULLABLE by Set Equations

$$
\begin{array}{lcl}
\textsc{Nullable}(\varepsilon) & = & \textit{true} \\
\textsc{Nullable}(\mathrm{a}) & = & \textit{false} \text{ for a} \in \Sigma \\
\textsc{Nullable}(\alpha\beta) & = & \textsc{Nullable}(\alpha) \wedge \textsc{Nullable}(\beta) \text{ for } \alpha, \beta \in (\Sigma \cup N)^* \\
\textsc{Nullable}(N) & = & \textsc{Nullable}(\alpha_1) \vee \ldots \vee \textsc{Nullable}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{array}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl}
G' : S & \to & aSb \mid B \\
B & \to & bB \mid \varepsilon
\end{array}
\qquad
\begin{array}{lcl}
\textsc{Nullable}(S) & = & \textsc{Nullable}(aSb) \vee \textsc{Nullable}(B) \\
\textsc{Nullable}(B) & = & \textsc{Nullable}(bB) \vee \textsc{Nullable}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{array}{lcl}
\textsc{Nullable}(aSB) & = & \textsc{Nullable}(a) \wedge \textsc{Nullable}(S) \wedge \textsc{Nullable}(B) \\
\textsc{Nullable}(B) & = & \textsc{Nullable}(B) \\
\textsc{Nullable}(bB) & = & \textsc{Nullable}(b) \wedge \textsc{Nullable}(B) \\
\textsc{Nullable}(\varepsilon) & = & \textit{true}
\end{array}
$$

Compute smallest solution of system, starting by *false* for all.

Computing $\textsc{Nullable}$ by Set Equations

$$
\begin{array}{lcl}
\textsc{Nullable}(\varepsilon) & = & \textit{true} \\
\textsc{Nullable}(\mathsf{a}) & = & \textit{false} \text{ for } \mathsf{a} \in \Sigma \\
\textsc{Nullable}(\alpha\beta) & = & \textsc{Nullable}(\alpha) \wedge \textsc{Nullable}(\beta) \text{ for } \alpha, \beta \in (\Sigma \cup N)^* \\
\textsc{Nullable}(N) & = & \textsc{Nullable}(\alpha_1) \vee \ldots \vee \textsc{Nullable}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{array}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl}
G' : S & \to & aSb \mid B \\
B & \to & bB \mid \varepsilon
\end{array}
\qquad
\begin{array}{lcl}
\textsc{Nullable}(S) & = & \textsc{Nullable}(aSb) \vee \textsc{Nullable}(B) \\
\textsc{Nullable}(B) & = & \textsc{Nullable}(bB) \vee \textsc{Nullable}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{array}{lcl}
\textsc{Nullable}(aSB) & = & \textsc{Nullable}(a) \wedge \textsc{Nullable}(S) \wedge \textsc{Nullable}(B) \\
\textsc{Nullable}(B) & = & \textsc{Nullable}(B) \\
\textsc{Nullable}(bB) & = & \textsc{Nullable}(b) \wedge \textsc{Nullable}(B) \\
\textsc{Nullable}(\varepsilon) & = & \textit{true}
\end{array}
$$

Compute smallest solution of system, starting by *false* for all.

Computing NULLABLE by Set Equations

$$
\begin{array}{lcl}
\textsc{Nullable}(\varepsilon) & = & \textit{true} \\
\textsc{Nullable}(\mathsf{a}) & = & \textit{false} \text{ for } \mathsf{a} \in \Sigma \\
\textsc{Nullable}(\alpha\beta) & = & \textsc{Nullable}(\alpha) \wedge \textsc{Nullable}(\beta) \text{ for } \alpha, \beta \in (\Sigma \cup N)^* \\
\textsc{Nullable}(N) & = & \textsc{Nullable}(\alpha_1) \vee \ldots \vee \textsc{Nullable}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{array}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl@{\qquad}lcl}
G' : S & \to & aSb \mid B & \textsc{Nullable}(S) & = & \textsc{Nullable}(aSb) \vee \textsc{Nullable}(B) \\
B & \to & bB \mid \varepsilon & \textsc{Nullable}(B) & = & \textsc{Nullable}(bB) \vee \textsc{Nullable}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{array}{lcl}
\textsc{Nullable}(aSB) & = & \textsc{Nullable}(a) \wedge \textsc{Nullable}(S) \wedge \textsc{Nullable}(B) = \textit{false} \\
\textsc{Nullable}(B) & = & \textsc{Nullable}(B) \text{ (does not contribute)} \\
\textsc{Nullable}(bB) & = & \textsc{Nullable}(b) \wedge \textsc{Nullable}(B) = \textit{false} \\
\textsc{Nullable}(\varepsilon) & = & \textit{true}
\end{array}
$$

Compute smallest solution of system, starting by *false* for all.

Computing $\textsc{First}$ by Set Equations

$$
\begin{array}{lcl}
\textsc{First}(\varepsilon) & = & \emptyset \\
\textsc{First}(\mathsf{a}) & = & \mathsf{a} \text{ for } \mathsf{a} \in \Sigma \\
\textsc{First}(\alpha\beta) & = & \begin{cases} \textsc{First}(\alpha) \cup \textsc{First}(\beta) & \text{, if } \textsc{Nullable}(\alpha) \\ \textsc{First}(\alpha) & \text{, otherwise} \end{cases} \\
\textsc{First}(N) & = & \textsc{First}(\alpha_1) \cup \ldots \cup \textsc{First}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \rightarrow \alpha_i\ (i \in \{1..n\})
\end{array}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl}
G' : S & \rightarrow & aSb \mid B \\
B & \rightarrow & bB \mid \varepsilon
\end{array}
\qquad
\begin{array}{lcl}
\textsc{First}(S) & = & \textsc{First}(aSb) \cup \textsc{First}(B) \\
\textsc{First}(B) & = & \textsc{First}(bB) \cup \textsc{First}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{array}{lcl}
\textsc{First}(aSB) & = & \textsc{First}(a) \\
\textsc{First}(B) & = & \textsc{First}(B) \\
\textsc{First}(bB) & = & \textsc{First}(b) \\
\textsc{First}(\varepsilon) & = & \emptyset
\end{array}
$$

Compute smallest solution of system, starting by $\emptyset$ for all sets.

Computing First by Set Equations

$$
\begin{aligned}
\textsc{First}(\varepsilon) &= \emptyset \\
\textsc{First}(\mathsf{a}) &= \mathsf{a} \text{ for } \mathsf{a} \in \Sigma \\
\textsc{First}(\alpha\beta) &= \begin{cases} \textsc{First}(\alpha) \cup \textsc{First}(\beta) & \text{, if } \textsc{Nullable}(\alpha) \\ \textsc{First}(\alpha) & \text{, otherwise} \end{cases} \\
\textsc{First}(N) &= \textsc{First}(\alpha_1) \cup \ldots \cup \textsc{First}(\alpha_n), \\
&\quad \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{aligned}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl}
G' : S & \to & aSb \mid B \\
B & \to & bB \mid \varepsilon
\end{array}
\qquad
\begin{array}{lcl}
\textsc{First}(S) & = & \textsc{First}(aSb) \cup \textsc{First}(B) \\
\textsc{First}(B) & = & \textsc{First}(bB) \cup \textsc{First}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{aligned}
\textsc{First}(aSB) &= \textsc{First}(a) \\
\textsc{First}(B) &= \textsc{First}(B) \\
\textsc{First}(bB) &= \textsc{First}(b) \\
\textsc{First}(\varepsilon) &= \emptyset
\end{aligned}
$$

Compute smallest solution of system, starting by $\emptyset$ for all sets.

Computing First by Set Equations

$$
\begin{array}{lcl}
\textsc{First}(\varepsilon) & = & \emptyset \\
\textsc{First}(\mathsf{a}) & = & \mathsf{a} \text{ for } \mathsf{a} \in \Sigma \\
\textsc{First}(\alpha\beta) & = & \begin{cases} \textsc{First}(\alpha) \cup \textsc{First}(\beta) & \text{, if } \textsc{Nullable}(\alpha) \\ \textsc{First}(\alpha) & \text{, otherwise} \end{cases} \\
\textsc{First}(N) & = & \textsc{First}(\alpha_1) \cup \ldots \cup \textsc{First}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{array}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl@{\qquad}lcl}
G' : S & \to & aSb \mid B & \textsc{First}(S) & = & \textsc{First}(aSb) \cup \textsc{First}(B) \\
B & \to & bB \mid \varepsilon & \textsc{First}(B) & = & \textsc{First}(bB) \cup \textsc{First}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{array}{lcl}
\textsc{First}(aSB) & = & \textsc{First}(a) \\
\textsc{First}(B) & = & \textsc{First}(B) \\
\textsc{First}(bB) & = & \textsc{First}(b) \\
\textsc{First}(\varepsilon) & = & \emptyset
\end{array}
$$

Compute smallest solution of system, starting by $\emptyset$ for all sets.

Computing $\textsc{First}$ by Set Equations

$$
\begin{array}{lcl}
\textsc{First}(\varepsilon) & = & \emptyset \\
\textsc{First}(\mathsf{a}) & = & \mathsf{a} \text{ for } \mathsf{a} \in \Sigma \\
\textsc{First}(\alpha\beta) & = & \begin{cases} \textsc{First}(\alpha) \cup \textsc{First}(\beta) & \text{, if } \textsc{Nullable}(\alpha) \\ \textsc{First}(\alpha) & \text{, otherwise} \end{cases} \\
\textsc{First}(N) & = & \textsc{First}(\alpha_1) \cup \ldots \cup \textsc{First}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{array}
$$

- Equations for nonterminals of the grammar:

$$
\begin{array}{lcl}
G' : S & \to & aSb \mid B \\
B & \to & bB \mid \varepsilon
\end{array}
\qquad
\begin{array}{lcl}
\textsc{First}(S) & = & \textsc{First}(aSb) \cup \textsc{First}(B) \\
\textsc{First}(B) & = & \textsc{First}(bB) \cup \textsc{First}(\varepsilon)
\end{array}
$$

- Equations for the right-hand side

$$
\begin{array}{lcl}
\textsc{First}(aSB) & = & \textsc{First}(a) = \{a\} \\
\textsc{First}(B) & = & \textsc{First}(B) \\
\textsc{First}(bB) & = & \textsc{First}(b) = \{b\} \\
\textsc{First}(\varepsilon) & = & \emptyset
\end{array}
$$

Compute smallest solution of system, starting by $\emptyset$ for all sets.

Computing $\textsc{First}$ by Set Equations

$$\begin{array}{lcl}
\textsc{First}(\varepsilon) & = & \emptyset \\
\textsc{First}(\mathsf{a}) & = & \mathsf{a} \text{ for } \mathsf{a} \in \Sigma \\
\textsc{First}(\alpha\beta) & = & \begin{cases} \textsc{First}(\alpha) \cup \textsc{First}(\beta) & \text{, if } \textsc{Nullable}(\alpha) \\ \textsc{First}(\alpha) & \text{, otherwise} \end{cases} \\
\textsc{First}(N) & = & \textsc{First}(\alpha_1) \cup \ldots \cup \textsc{First}(\alpha_n), \\
 & & \text{using all productions for } N,\ N \to \alpha_i\ (i \in \{1..n\})
\end{array}$$

- Equations for nonterminals of the grammar:

$$\begin{array}{lcllcl}
G' : S & \to & aSb \mid B & \textsc{First}(S) & = & \textsc{First}(aSb) \cup \textsc{First}(B) = \{a, b\} \\
B & \to & bB \mid \varepsilon & \textsc{First}(B) & = & \textsc{First}(bB) \cup \textsc{First}(\varepsilon) = \{b\}
\end{array}$$

- Equations for the right-hand side

$$\begin{array}{lcl}
\textsc{First}(aSB) & = & \textsc{First}(a) = \{a\} \\
\textsc{First}(B) & = & \textsc{First}(B) = \{b\} \\
\textsc{First}(bB) & = & \textsc{First}(b) = \{b\} \\
\textsc{First}(\varepsilon) & = & \emptyset
\end{array}$$

Compute smallest solution of system, starting by $\emptyset$ for all sets.

FOLLOW Sets for Nonterminals

FIRST Sets sometimes not sufficient for parsing.
Especially, suppose a production $X \to \alpha$ and $\text{NULLABLE}(\alpha)$. If the *look-ahead can follow* X, this production should be chosen. The FIRST set of α cannot provide this information.

Definition (FOLLOW Set of a Nonterminal)

Let $G = (\Sigma, N, S, P)$ a grammar and $\Rightarrow$ its derivation relation.
For each nonterminal $X \in N$, define

- $\text{FOLLOW}(X) = \{c \in \Sigma \mid \exists_{\alpha,\beta \in (\Sigma \cup N)^*} : \; S \Rightarrow^* \alpha \underline{Xc}\beta\}$
  (all input tokens that follow X in sequences derivable from S)

To recognise the end of the input, we also extend the grammar with a new start symbol S', a new character \$ and a production $S' \to S\$$.
Thereby, we always have $\$ \in \text{FOLLOW}(S)$.

FOLLOW Sets for Nonterminals

FIRST Sets sometimes not sufficient for parsing.
Especially, suppose a production $X \rightarrow \alpha$ and $\textsc{Nullable}(\alpha)$. If the *look-ahead can follow* X, this production should be chosen. The FIRST set of α cannot provide this information.

Definition (FOLLOW Set of a Nonterminal)

Let $G = (\Sigma, N, S, P)$ a grammar and $\Rightarrow$ its derivation relation.
For each nonterminal $X \in N$, define

- $\textsc{Follow}(X) = \{\mathsf{c} \in \Sigma \mid \exists_{\alpha,\beta \in (\Sigma \cup N)^*} : \ S \Rightarrow^* \alpha \underline{X\mathsf{c}} \beta\}$
  (all input tokens that follow X in sequences derivable from S)

To recognise the end of the input, we also extend the grammar with a new start symbol S', a new character \$ and a production $S' \rightarrow S\$$.
Thereby, we always have $\$ \in \textsc{Follow}(S)$.

Set Equations for FOLLOW Sets

FOLLOW sets solve a *collection of set constraints*.

Constraints derived from *right-hand sides* of grammar productions

For $X \in N$, consider all productions $Y \to \alpha X \beta$ where X occurs on the right.

- $\underline{\text{FIRST}(\beta) \subseteq \text{FOLLOW}(X)}$
- If $\text{NULLABLE}(\beta)$ or $\beta = \varepsilon$: $\underline{\text{FOLLOW}(Y) \subseteq \text{FOLLOW}(X)}$

If X occurs several times, each occurrence contributes separate equations.

Simple Example:

$$\begin{array}{llrl} S' \to S\$ & \ldots & \text{FIRST}(\$) = \{\$\} & \subseteq \text{FOLLOW}(S) \\ S \to aSb & \ldots & \text{FIRST}(b) = \{b\} & \subseteq \text{FOLLOW}(S) \\ S \to B & \ldots & \text{FOLLOW}(S) & \subseteq \text{FOLLOW}(B) \\ B \to bB & \ldots & \text{FOLLOW}(B) & \subseteq \text{FOLLOW}(B) \\ B \to \varepsilon & \ldots & \text{(nothing)} & \end{array}$$

Solve iteratively, starting by $\emptyset$ for all nonterminals.

Set Equations for $\textsc{Follow}$ Sets

$\textsc{Follow}$ sets solve a *collection of set constraints*.

Constraints derived from *right-hand sides* of grammar productions

For $X \in N$, consider all productions $Y \to \alpha X \beta$ where X occurs on the right.

- $\underline{\textsc{First}(\beta) \subseteq \textsc{Follow}(X)}$
- If $\textsc{Nullable}(\beta)$ or $\beta = \varepsilon$: $\underline{\textsc{Follow}(Y) \subseteq \textsc{Follow}(X)}$

If X occurs several times, each occurrence contributes separate equations.

Simple Example:

$$
\begin{array}{llrl}
S' \to S\$ & \ldots & \textsc{First}(\$) = \{\$\} & \subseteq \textsc{Follow}(S) \\
S \to aSb & \ldots & \textsc{First}(b) = \{b\} & \subseteq \textsc{Follow}(S) \\
S \to B & \ldots & \textsc{Follow}(S) & \subseteq \textsc{Follow}(B) \\
B \to bB & \ldots & \textsc{Follow}(B) & \subseteq \textsc{Follow}(B) \\
B \to \varepsilon & \ldots & \text{(nothing)} &
\end{array}
$$

Solve iteratively, starting by $\emptyset$ for all nonterminals.

Set Equations for FOLLOW Sets

FOLLOW sets solve a *collection of set constraints*.

Constraints derived from *right-hand sides* of grammar productions

For $X \in N$, consider all productions $Y \to \alpha X \beta$ where X occurs on the right.

- $\underline{\text{FIRST}(\beta) \subseteq \text{FOLLOW}(X)}$
- If $\text{NULLABLE}(\beta)$ or $\beta = \varepsilon$: $\underline{\text{FOLLOW}(Y) \subseteq \text{FOLLOW}(X)}$

If X occurs several times, each occurrence contributes separate equations.

Simple Example:

$$\begin{array}{llrl}
S' \to S\$ & \ldots & \text{FIRST}(\$) = \{\$\} & \subseteq \text{FOLLOW}(S) \\
S \to aSb & \ldots & \text{FIRST}(b) = \{b\} & \subseteq \text{FOLLOW}(S) \\
S \to B & \ldots & \text{FOLLOW}(S) & \subseteq \text{FOLLOW}(B) \\
B \to bB & \ldots & \text{FOLLOW}(B) & \subseteq \text{FOLLOW}(B) \\
B \to \varepsilon & \ldots & \text{(nothing)} &
\end{array}$$

Solve iteratively, starting by $\emptyset$ for all nonterminals.

$$\curvearrowright \text{FOLLOW}(S) = \text{FOLLOW}(B) = \{\$, b\}$$

Putting it Together: Look-ahead Sets and LL(1)

After computing $\textsc{Nullable}$ and $\textsc{First}$ for all right-hand sides and $\textsc{Follow}$ for all nonterminals, a parser can be constructed.

Definition (Look-ahead Sets of a Grammar)

For every production $X \rightarrow \alpha$ of a grammar, we define the ***Look-ahead set*** of the production as:

- $la(X \rightarrow \alpha) =$
$$\begin{cases} \textsc{First}(\alpha) \cup \textsc{Follow}(X) & \text{, if } \textsc{Nullable}(\alpha) \\ \textsc{First}(\alpha) & \text{, otherwise} \end{cases}$$

LL(1) Grammars

If for each nonterminal $X \in N$, the productions' look-ahead sets are disjoint, a syntax tree can be built with one token look-ahead. The grammar is then called LL(1) (*left-to-right, left-most, look-ahead 1*).

The parser can then predict the next production from look-ahead.

Putting it Together: Look-ahead Sets and LL(1)

After computing $\textsc{Nullable}$ and $\textsc{First}$ for all right-hand sides and $\textsc{Follow}$ for all nonterminals, a parser can be constructed.

Definition (Look-ahead Sets of a Grammar)

For every production $X \to \alpha$ of a grammar, we define the *Look-ahead set* of the production as:

- $la(X \to \alpha) = \begin{cases} \textsc{First}(\alpha) \cup \textsc{Follow}(X) & \text{, if } \textsc{Nullable}(\alpha) \\ \textsc{First}(\alpha) & \text{, otherwise} \end{cases}$

LL(1) Grammars

If for each nonterminal $X \in N$, the productions' look-ahead sets are disjoint, a syntax tree can be built with one token look-ahead. The grammar is then called LL(1) (*left-to-right, left-most, look-ahead 1*).

The parser can then predict the next production from look-ahead.

Examples

The grammar we have used in our example is <u>not</u> LL(1):

$$
\begin{array}{rcl@{\qquad}rcl}
G' : S & \to & aSb & la(S \to aSb) & = & \textsc{First}(aSB) = \{a\} \\
S & \to & B & la(S \to B) & = & \textsc{First}(B) \cup \textsc{Follow}(B) = \{b\} \\
B & \to & bB & la(B \to bB) & = & \textsc{First}(bB) = \{b\} \\
B & \to & \varepsilon & la(B \to \varepsilon) & = & \textsc{First}(\varepsilon) \cup \textsc{Follow}(B) = \{\$, b\}
\end{array}
$$

Solution: Separate generation of additional bs at the end.

Examples

The grammar we have used in our example is <u>not</u> LL(1):

$$
\begin{array}{rcl@{\qquad}rcl}
G' : S & \to & aSb & la(S \to aSb) & = & \textsc{First}(aSB) = \{a\} \\
S & \to & B & la(S \to B) & = & \textsc{First}(B) \cup \textsc{Follow}(B) = \{b\} \\
B & \to & bB & la(B \to bB) & = & \textsc{First}(bB) = \{b\} \\
B & \to & \varepsilon & la(B \to \varepsilon) & = & \textsc{First}(\varepsilon) \cup \textsc{Follow}(B) = \{\$, b\}
\end{array}
$$

Solution: Separate generation of additional bs at the end.

Examples

The grammar we have used in our example is not LL(1):

$$\begin{array}{rcl} G' : S & \to & aSb \\ S & \to & B \\ B & \to & bB \\ B & \to & \varepsilon \end{array} \qquad \begin{array}{rcl} la(S \to aSb) & = & \textsc{First}(aSB) = \{a\} \\ la(S \to B) & = & \textsc{First}(B) \cup \textsc{Follow}(B) = \{b\} \\ la(B \to bB) & = & \textsc{First}(bB) = \{b\} \\ la(B \to \varepsilon) & = & \textsc{First}(\varepsilon) \cup \textsc{Follow}(B) = \{\$, b\} \end{array}$$

Solution: Separate generation of additional bs at the end.

$$\begin{array}{rcl} G' : S & \to & AB \\ A & \to & aAb \\ A & \to & \varepsilon \\ B & \to & bB \\ B & \to & \varepsilon \end{array}$$

Examples

The grammar we have used in our example is <u>not</u> LL(1):

$$
\begin{array}{llllcl}
G' : S & \rightarrow & aSb & la(S \rightarrow aSb) & = & \textsc{First}(aSB) = \{a\} \\
S & \rightarrow & B & la(S \rightarrow B) & = & \textsc{First}(B) \cup \textsc{Follow}(B) = \{b\} \\
B & \rightarrow & bB & la(B \rightarrow bB) & = & \textsc{First}(bB) = \{b\} \\
B & \rightarrow & \varepsilon & la(B \rightarrow \varepsilon) & = & \textsc{First}(\varepsilon) \cup \textsc{Follow}(B) = \{\$, b\}
\end{array}
$$

Solution: Separate generation of additional bs at the end.

$$
\begin{array}{llllcl}
G' : S & \rightarrow & AB & la(S \rightarrow AB) & = & \textsc{First}(AB) \cup \textsc{Follow}(S) = \{a, b, \$\} \\
A & \rightarrow & aAb & la(A \rightarrow aAb) & = & \textsc{First}(aAB) = \{a\} \\
A & \rightarrow & \varepsilon & la(A \rightarrow \varepsilon) & = & \textsc{First}(\varepsilon) \cup \textsc{Follow}(A) = \{b, \$\} \\
B & \rightarrow & bB & la(B \rightarrow bB) & = & \textsc{First}(bB) = \{b\} \\
B & \rightarrow & \varepsilon & la(B \rightarrow \varepsilon) & = & \textsc{First}(\varepsilon) \cup \textsc{Follow}(B) = \{\$\}
\end{array}
$$

Recursive Descent Parsing (Pseudo-Code)

Recursive Procedures keep track of the parse in the *call stack*.
Each procedure recursively calls parsers for the right-hand side.
Whenever a production is used, its number is added to the output.

```
fun parseS () =
    (* check look-ahead set for S->AB *)
    if next = 'a' or next = 'b' or next = '$' then
        (* Follow the production *)
        print("1"); parseA(); parseB(); match('$')
    else error("unexpected input " ^ next)
and parseA () =
    (* branch on look-ahead a: A->aAb, b$: A-> *)
    if next = 'a' then (* Follow the production *)
        print("2"); match('a'); parseA(); match('b')
    else if next = 'b' or next = '$' then print("3");
         else error("unexpected input " ^ next)
and parseB () =
    (* branch on look-ahead b: B->bB, $: B-> *)
    if next = 'b' then
        (* Follow the production *)
        print("4"); match('b'); parseB()
    else if next = '$' then print("5")
         else error("unexpected input " ^ next)
```

Table-Driven LL(1) Parsing

- *Stack*, contains unprocessed part of production, initially: $S\$$.
- *Parser Table*: action to take, depending on stack and next input
- *Actions* (pop consumes inut, derivation only reads it)

Pop: remove terminal from stack (on matching input).
Derive: pop nonterminal from stack, push right-hand side in table.

- Accept input when stack and end of input.

Stack:	Look-ahead/Input: a	b	$
S	AB, 1	AB, 1	AB, 1
A	aAb, 2	ε, 3	ε, 3
B	*error*	*bB*, 4	ε, 5
a	*pop*	*error*	*error*
b	*error*	*pop*	*error*
$	*error*	*error*	*accept*

Example run (input `aabbb`):

Input	Stack	Action	Output
aabbb$	$S\$$	derive	ε
aabbb$	$AB\$$	derive	1
aabbb$	$aAbB\$$	pop	12
abbb$	$AbB\$$	derive	12
abbb$	$aAbbB\$$	pop	122
bbb$	$AbbB\$$	derive	122
bbb$	$bbB\$$	pop	1223
bb$	$bB\$$	pop	1223
b$	$B\$$	derive	1223
b$	$bB\$$	pop	12234
$	$B\$$	derive	12234
$	$	accept	122345

Eliminating Left-Recursion and Left-Factorisation

Problems that often occur when constructing LL(1) parsers:

- *Identical prefixes*: Productions $X \to \alpha\beta \mid \alpha\gamma$.
  Requires look-ahead longer than the common prefix α.

  Solution: *Left-Factorisation*, introducing new productions
  $X \to \alpha Y$ and $Y \to \beta \mid \gamma$.

- *Left-Recursion*: a nonterminal reproducing itself on the left.
  Direct: production $X \to X\alpha \mid \beta$, or indirect: $X \Rightarrow^* X\alpha$.
  Cannot be analised with finite look-ahead!
  $X \to X\alpha \mid \beta$, thus $\text{FIRST}(X) \subset \text{FIRST}(X\alpha) \cup \text{FIRST}(\beta)$

  Solution: new (nullable) nonterminals and swapped recursion.
  $X \to \beta X'$ and $X' \to \alpha X' \mid \varepsilon$

  Also works in case of multiple left-recursive productions.
  For indirect recursion: first transform into direct recursion.

Eliminating Left-Recursion and Left-Factorisation

Problems that often occur when constructing LL(1) parsers:

- *Identical prefixes*: Productions $X \to \alpha\beta \mid \alpha\gamma$.
  Requires look-ahead longer than the common prefix α.

  Solution: *Left-Factorisation*, introducing new productions

  $X \to \alpha Y$ and $Y \to \beta \mid \gamma$.

- *Left-Recursion*: a nonterminal reproducing itself on the left.
  Direct: production $X \to X\alpha \mid \beta$, or indirect: $X \Rightarrow^* X\alpha$.
  Cannot be analised with finite look-ahead!

  $X \to X\alpha \mid \beta$, thus $\textsc{first}(X) \subset \textsc{first}(X\alpha) \cup \textsc{First}(\beta)$

  Solution: new (nullable) nonterminals and swapped recursion.

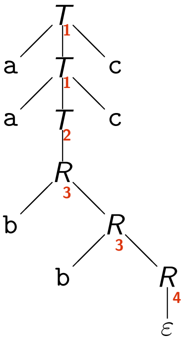
  $X \to \beta X'$ and $X' \to \alpha X' \mid \varepsilon$

  Also works in case of multiple left-recursive productions.
  For indirect recursion: first transform into direct recursion.

Bottom-Up Parsing

LL(1) Parser works top-down. *Needs to guess* used productions.
Bottom-Up approach: build syntax tree from leaves.

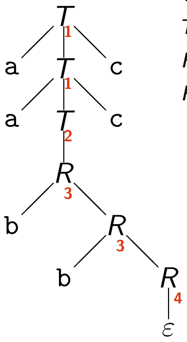
$$
\begin{array}{cclc}
T' & \rightarrow & T\$ & (0) \\
T & \rightarrow & \mathtt{a}T\mathtt{c} & (1) \\
T & \rightarrow & R & (2) \\
R & \rightarrow & \mathtt{b}R & (3) \\
R & \rightarrow & \varepsilon & (4)
\end{array}
$$



$$T \overset{1}{\Rightarrow} \mathtt{a}T\mathtt{c} \overset{1}{\Rightarrow} \mathtt{aa}T\mathtt{cc} \overset{2}{\Rightarrow} \mathtt{aa}R\mathtt{cc} \overset{3}{\Rightarrow} \mathtt{aab}R\mathtt{cc} \overset{3}{\Rightarrow} \mathtt{aabb}R\mathtt{cc} \overset{4}{\Rightarrow} \mathtt{aabbcc}$$

Bottom-Up Parsing: Idea for a Machine

$$
\begin{array}{rcll}
T' & \to & T\$ & (0) \\
T & \to & \mathtt{a}\,T\,\mathtt{c} & (1) \\
T & \to & R & (2) \\
R & \to & \mathtt{b}R & (3) \\
R & \to & \varepsilon & (4)
\end{array}
$$

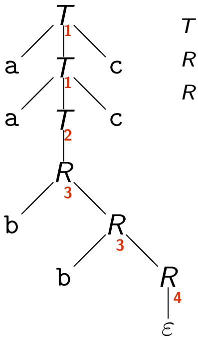


Questions:

- When to accept (solved: use separate start production with T')
- When to shift, when to reduce? Especially $R \to \varepsilon$.

Bottom-Up Parsing: Idea for a Machine

Stack	Input	Action
ε	aabbcc\$	**shift**
a	abbcc\$	**shift**
aa	bbcc\$	**shift**
aab	bcc\$	**shift**
aabb	cc\$	**reduce 4**
aab$\underline{\mathtt{b}R}$	cc\$	**reduce 3**
aa$\underline{\mathtt{b}R}$	cc\$	**reduce 3**
aa$\underline{R}$	cc\$	**reduce 2**
aa$T$	cc\$	**shift**
a$\underline{\mathtt{a}T\mathtt{c}}$	c\$	**reduce 1**
aT	c\$	**shift**
$\underline{\mathtt{a}T\mathtt{c}}$	\$	**reduce 1**
$\underline{T}$	\$	**accept**



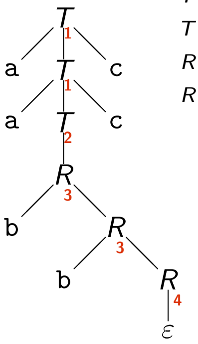
$$
\begin{array}{llll}
T' & \to & T\$ & (0) \\
T & \to & \mathtt{a}T\mathtt{c} & (1) \\
T & \to & R & (2) \\
R & \to & \mathtt{b}R & (3) \\
R & \to & \varepsilon & (4)
\end{array}
$$

Questions:

- When to accept (solved: use separate start production with T')
- When to shift, when to reduce? Especially $R \to \varepsilon$.

Bottom-Up Parsing: Idea for a Machine

Stack	Input	Action
ε	aabbcc\$	**shift**
a	abbcc\$	**shift**
aa	bbcc\$	**shift**
aab	bcc\$	**shift**
aabb	cc\$	**reduce 4**
aab$\underline{\text{b}R}$	cc\$	**reduce 3**
aa$\underline{\text{b}R}$	cc\$	**reduce 3**
aa$\underline{R}$	cc\$	**reduce 2**
aa$T$	cc\$	**shift**
a$\underline{\text{a}T\text{c}}$	c\$	**reduce 1**
aT	c\$	**shift**
$\underline{\text{a}T\text{c}}$	\$	**reduce 1**
$\underline{T}$	\$	**accept**



$$
\begin{array}{lcll}
T' & \rightarrow & T\$ & (0) \\
T & \rightarrow & \mathrm{a}T\mathrm{c} & (1) \\
T & \rightarrow & R & (2) \\
R & \rightarrow & \mathrm{b}R & (3) \\
R & \rightarrow & \varepsilon & (4)
\end{array}
$$

Questions:

- When to accept (solved: use separate start production with T')
- When to shift, when to reduce? Especially $R \rightarrow \varepsilon$.

Constructing an SLR Parser: Items

Each production in the grammar leads to a number of *items*:

Shift Items and Reduce Items of a Production

Let $X \to \alpha$ be a production in a grammar. The production implies:

- *Shift items*: $[X \to \alpha_1 \bullet \alpha_2]$ for every decomposition $\alpha = \alpha_1 \alpha_2$
- One *reduce item*: $[X \to \alpha \bullet]$ per production.

Items give information about the next action:

- *shift* input to the stack
- *reduce* part of the stack using a production

- Stack of the parser will contain *items*, not grammar symbols.
- Therefore, no need to read the stack for reductions.

Constructing an SLR Parser: Items

Each production in the grammar leads to a number of *items*:

Shift Items and Reduce Items of a Production

Let $X \to \alpha$ be a production in a grammar. The production implies:

- *Shift items*: $[X \to \alpha_1 \bullet \alpha_2]$ for every decomposition $\alpha = \alpha_1\alpha_2$
- One *reduce item*: $[X \to \alpha\bullet]$ per production.

Items give information about the next action:

- *shift* input to the stack
- *reduce* part of the stack using a production

- Stack of the parser will contain *items*, not grammar symbols.
- Therefore, no need to read the stack for reductions.

Constructing an SLR Parser: Production DFAs

Each production $X \to \alpha$ leads to a DFA with the following transitions:

- From $[X \to \alpha \bullet \mathrm{a}\beta]$ to $[X \to \alpha\mathrm{a} \bullet \beta]$ for input tokens a. These will be *shift* action that read input later.
- From $[X \to \alpha \bullet Y\beta]$ to $[X \to \alpha Y \bullet \beta]$ for nonterminals Y. These will be *go* actions later, without consuming input.

All items are states, start state is the first item $[X \to \bullet\alpha]$.

$T \to \mathrm{a}T\mathrm{c}$ $\quad \boxed{[T \to \bullet \mathrm{a}T\mathrm{c}]} \xrightarrow{\mathrm{a}} \boxed{[T \to \mathrm{a} \bullet T\mathrm{c}]} \xrightarrow{T} \boxed{[T \to \mathrm{a}T \bullet \mathrm{c}]} \xrightarrow{\mathrm{c}} \boxed{[T \to \mathrm{a}T\mathrm{c}\bullet]}$

$T \to R$ $\quad \boxed{[T \to \bullet R]} \xrightarrow{R} \boxed{[T \to R\bullet]}$

While *traversing* the DFA: items pushed on the stack.
When reaching a *reduce item*: use stack to back-track (later).

Constructing an SLR Parser: Production DFAs

Each production $X \to \alpha$ leads to a DFA with the following transitions:

- From $[X \to \alpha \bullet \mathrm{a}\beta]$ to $[X \to \alpha\mathrm{a} \bullet \beta]$ for input tokens a.
  These will be *shift* action that read input later.
- From $[X \to \alpha \bullet Y\beta]$ to $[X \to \alpha Y \bullet \beta]$ for nonterminals Y.
  These will be *go* actions later, without consuming input.

All items are states, start state is the first item $[X \to \bullet\alpha]$.

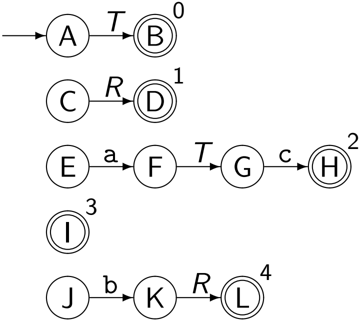
$$T \to \mathrm{a}T\mathrm{c} \qquad \begin{tikzcd} \boxed{[T \to \bullet \mathrm{a}T\mathrm{c}]} \arrow[r, "\mathrm{a}"] & \boxed{[T \to \mathrm{a} \bullet T\mathrm{c}]} \arrow[r, "T"] & \boxed{[T \to \mathrm{a}T \bullet \mathrm{c}]} \arrow[r, "\mathrm{c}"] & \boxed{\boxed{[T \to \mathrm{a}T\mathrm{c}\bullet]}} \end{tikzcd}$$

$$T \to R \qquad \begin{tikzcd} \boxed{[T \to \bullet R]} \arrow[r, "R"] & \boxed{\boxed{[T \to R\bullet]}} \end{tikzcd}$$

While *traversing* the DFA: items pushed on the stack.
When reaching a *reduce item*: use stack to back-track (later).

SLR Parser Construction: Example

Productions	NFA
$T' \to T$	
$T \to R$	
$T \to \mathtt{a}T\mathtt{c}$	
$R \to$	
$R \to \mathtt{b}R$	



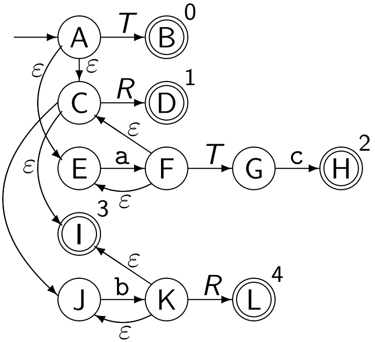
Extra ε-transitions connect the DFAs for all productions:

- From $[X \to \alpha \bullet Y\beta]$ to $[Y \to \bullet\gamma]$ for all productions $Y \to \gamma$

  When in front of a nonterminal Y in a production DFA:
  try to run the DFA for one of the right-hand sides of Y productions.

SLR Parser Construction: Example

Productions	NFA
$T' \to T$	
$T \to R$	
$T \to \mathtt{a}T\mathtt{c}$	
$R \to$	
$R \to \mathtt{b}R$	



Extra ε-transitions connect the DFAs for all productions:

- From $[X \to \alpha \bullet Y\beta]$ to $[Y \to \bullet\gamma]$ for all productions $Y \to \gamma$

When in front of a nonterminal Y in a production DFA:
try to run the DFA for one of the right-hand sides of Y productions.

SLR Parser Construction: Example

Productions	NFA

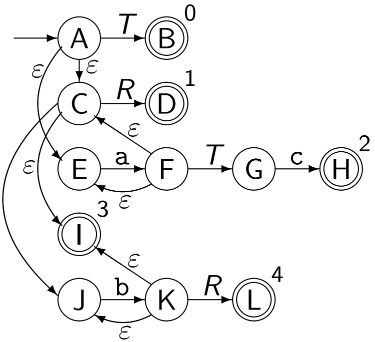
$T' \to T$

$T \to R$

$T \to \mathtt{a}T\mathtt{c}$

$R \to$

$R \to \mathtt{b}R$



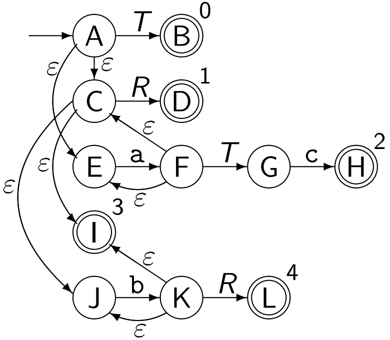
Extra ε-transitions connect the DFAs for all productions:

- From $[X \to \alpha \bullet Y\beta]$ to $[Y \to \bullet\gamma]$ for all productions $Y \to \gamma$

  When in front of a nonterminal Y in a production DFA:
  try to run the DFA for one of the right-hand sides of Y productions.

SLR Parser Construction: Example(2)

Productions	NFA
$T' \to T$	
$T \to R$	
$T \to \mathtt{a}T\mathtt{c}$	
$R \to$	
$R \to \mathtt{b}R$	

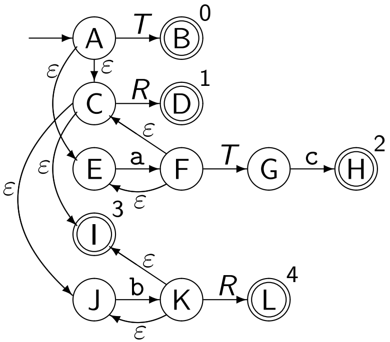


Next step: *Subset construction* of a combined DFA.

Blackboard. . .

SLR Parser Construction: Example(2)

Productions	NFA
$T' \to T$	
$T \to R$	
$T \to \mathtt{a}T\mathtt{c}$	
$R \to$	
$R \to \mathtt{b}R$	



Next step: *Subset construction* of a combined DFA.

Blackboard...

SLR Parsing: Internal DFA and Stack

- *Transitions*: *Shift actions* (terminals) and *Go actions* (nonterminals).
- Final DFA states: contain *reduce items*.
  Reduce actions need to be added to the transition table.
- Reduce: remove as many items from the stack as the length of the right-hand side. Followed by a *Go*.

- SLR Parser Table: actions indexed by symbols and DFA states.

Shift n	Read next input symbol, push state n on stack
Go n	Push state n on stack (consume nonterminal, do not read input)
Reduce p	Reduce with production p
Accept	Parsing has succeeded (reduce with production 0).