Ambiguous Grammars

For now, we are only working with CFG's.

Let G be a CFG over Σ. Recall:

G is **right-linear (RL)** if all productions of G are of the form:

$$A \longrightarrow xB \quad \text{or} \quad A \longrightarrow x$$

G is **left-linear (LL)** if all productions of G are of the form:

$$A \longrightarrow Bx \quad \text{or} \quad A \longrightarrow x$$

G is **linear** iff G is **RL** or **LL**.

Definition: G is **semilinear** iff
at most 1 non-terminal occurs on the r.h.s. of any production.

Note:
All **linear grammars** are **semilinear**, but **not conversely**.

Examples: (1) $\Sigma = \{a, b\}$, $L = \{a^n b^n \mid n = 0, 1, 2, \ldots\}$.

L is generated by grammar $G : S \longrightarrow aSb \mid \lambda$

G is **semilinear**, but **not linear**.

$Q.$ Is it possible that there is **also** a **linear** grammar for L?

$A.$ No, since L is not regular by the PL.

(2) $L = WN_{[\,]}$ = the set of **well-nested** strings over $\{ [\,,\,] \}$.

Generated by $G : S \longrightarrow [S] \mid SS \mid \lambda$

Not even semilinear!

$Q.$ Is L regular?

I.e. does L possibly also have a **linear** grammar?

$A.$ No, again by the PL

Note: We will see:
Non-semilinear grammars are associated with
ambiguity of syntax.

Leftmost & Rightmost Derivations

Suppose G is a **non-semilinear** CFG.

Then there are generally more than 1 derivation of a word in G,

e.g. Suppose G has productions

Production #	Production
1	$S \longrightarrow AB$
2,3	$A \longrightarrow aA \mid \lambda$
4,5	$bbB \mid \lambda$

Then for example:

(1) $S \stackrel{1}{\Longrightarrow} AB \stackrel{2}{\Longrightarrow} aAB \stackrel{2}{\Longrightarrow} aaAB \stackrel{3}{\Longrightarrow} aaB \stackrel{4}{\Longrightarrow} aabbB \stackrel{5}{\Longrightarrow} aabb$,

(2) $S \stackrel{1}{\Longrightarrow} AB \stackrel{4}{\Longrightarrow} AbbB \stackrel{5}{\Longrightarrow} Abb \stackrel{2}{\Longrightarrow} aAbb \stackrel{2}{\Longrightarrow} aaAbb \stackrel{3}{\Longrightarrow} aabb$,

(3) $S \stackrel{1}{\Longrightarrow} AB \stackrel{2}{\Longrightarrow} aAB \stackrel{4}{\Longrightarrow} aAbbB \stackrel{2}{\Longrightarrow} aaAbbB \stackrel{3}{\Longrightarrow} aabbB \stackrel{5}{\Longrightarrow} aabb$,

(etc.) are all derivations in G of a^2b^2.

They all use the **same productions**, just in a **different order**

Derivations of $\boldsymbol{a^2b^2}$ in $\boldsymbol{G}$ on p. 5-15:

(1) is a **leftmost** derivation,

i.e. at each step, **leftmost non-terminal** is used,

(2) is a **rightmost** derivation,

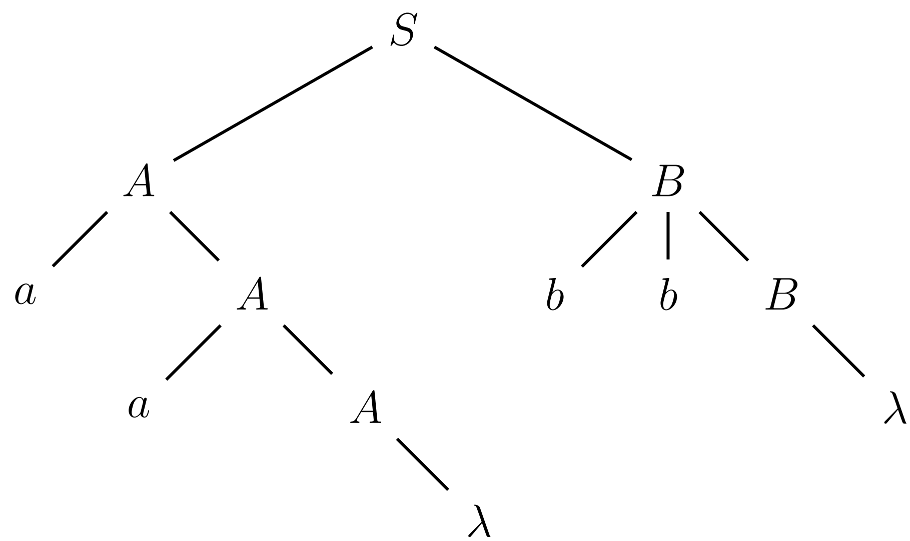
i.e. at each step, **rightmost non-terminal** is used,

(3) is **neither** of these.

Note:
Such choices are only possible for **non-semilinear** grammars.

Want to say: these derivations are “**really the same**” in some sense.
The **non-terminals** are just eliminated in a **different order**.
We can say: they have the same **derivation tree** or **parse tree**:

Parse tree for ***aabb*** in ***G***:



This shows a derivation of a word of ***G*** by **reading leaves left → right**.

Note:

For **parse trees**:

(1) The **root** is labeled ***S***

(2) Every **leaf** is labeled by a **terminal** or ***λ***

(3) Every **non-leaf** is labelled by a **non-terminal**

If (2) is replaced by:

(2') Every leaf is labelled by a **terminal**, ***λ***, or a **non-terminal**

- Have a **partial parse tree**:

  shows a derivation of a **sentential form** in ***G***.

Ambiguous grammars

Let G be the **non-semilinear** CFG

$$S \longrightarrow SS \mid [\,S\,] \mid [\,]$$

This generates all **non-empty well-nested** bracket strings.

Consider 2 derivations of $u = [\,]\,[\,]\,[\,]$.

(1) $S \Longrightarrow SS \Longrightarrow [\,]\,S \Longrightarrow [\,]\,SS \Longrightarrow [\,]\,[\,]\,S \Longrightarrow [\,]\,[\,]\,[\,]$

(2) $S \Longrightarrow SS \Longrightarrow S\,[\,] \Longrightarrow SS\,[\,] \Longrightarrow [\,]\,S\,[\,] \Longrightarrow [\,]\,[\,]\,[\,]$

These have **different parse trees**!

Grammars like G which have more than 1 parse tree for the same word are called **ambiguous**.

These are **bad** for programming languages!

Example:

Modified BNF (p. 1-14) is a form of CFG!

So e.g. consider grammar for **arithmetic expressions** $e, e', \ldots$

from **variables** $x, y, \ldots$,

numerals $\overline{m}, \overline{n}, \ldots$, and

arithmetic operators $+, \times, -$

$$e ::= x \mid \overline{n} \mid e_1 + e_2 \mid e_1 \times e_2 \mid\ - e$$

This grammar is ambiguous!

Can generate expressions:

$e_1 + e_2 + e_3$

$e_1 \times e_2 \times e_3$

What is their **parse tree**?

If you say it doesn't matter because of **associativity** of $+$ and $\times$, what about

$$e_1 + e_2 \times e_3$$

For this, we need **precedence rules**

('$\times$' has higher precedence than '$+$')

But better:

Rewrite the grammar s.t. only 1 parsing is possible:

Use... parentheses!

Read Linz §5.3:

"Context-Free Grammars and Programming Languages"

Warning!

Our terminology is not the same as Linz's:

- Our **linear** grammar is called **regular** by Linz;
- Our **semilinear** grammar is called **linear** by Linz.