Carnegie Mellon University in Qatar
How can we characterize these languages just outside the boundary of RLS?
We showed that \( L = \{a^n b^n \mid n \geq 0\} \) was not regular.

- No DFA
- No Regular Expression

How can we describe such languages?

Remember: the description has to be finite!
Consider $L = \{a^n b^n \mid n \geq 0\}$ again.

How can we generate such strings?

- Remember DFAs did recognition, not generation.

Consider the following inductive way to generate elements of $L$

- **Basis:** $\epsilon$ is in the language
- **Recursion:** If the string $w$ is in the language, then so is the string $awb$.

$\epsilon \rightarrow ab \rightarrow aabb \cdots \rightarrow a^{55} b^{55} \cdots$

- Looks like we have simple and finite length process to generate all the strings in $L$
- How can we generalize this kind of description?
Consider $L = \{ w \mid n_a(w) = n_b(w) \}$.

Now consider the following inductive way to generate elements of $L$

- **Basis**: $\epsilon$ is in the language
- **Recursion 1**: If the string $w$ is in the language, then so are $awb$ and $bwa$
- **Recursion 2**: If the strings $w$ and $v$ are in the language, so is $wv$.

The first recursion rules makes sure that the $a$’s and $b$’s are generated in the same number (regardless of order)

The second recursion takes any two strings each with equal number of $a$’s and $b$’s and generates a new such string by concatenating them.
Grammars provide the generative mechanism to generate all strings in a language.

A grammar is essentially a collection of substitution rules, called productions.

Each production rule has a left-hand-side and a right-hand-side.
Consider once again $L = \{a^n b^n \mid n \geq 0\}$

**Basis:** $\epsilon$ is in the language

- Production: $S \rightarrow \epsilon$

**Recursion:** If $w$ is in the language, then so is the string $awb$.

- Production: $S \rightarrow aSb$

- $S$ is called a variable or a nonterminal symbol
- $a, b$ etc., are called terminal symbols
- One variable is designated as the start variable or start symbol.
Consider the set of rules $R = \{ S \rightarrow \epsilon, S \rightarrow aSb \}$

Start with the start variable $S$

Apply the following until all remaining symbols are terminal.
- Choose a production in $R$ whose left-hand sides matches one of the variables.
- Replace the variable with the rule’s right hand side.

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaaabbbb$

The string $aaaaabbbb$ is in the language $L$

The sequence of rule applications above is called a derivation.
Parse Trees

Derivations can also be represented with a parse tree.

The leaves constitute the yield of the tree.

Terminal symbols can occur only at the leaves.

Variables can occur only at the internal nodes.

The terminals concatenated from left to right give us the string.
All strings generated this way starting with the start variable constitute the language of the grammar.

We write $L(G)$ for the language of the grammar $G$. 
A Grammar for a Fragment of English

Nomenclature:
- **S**: Sentence
- **NP**: Noun Phrase
- **CN**: Complex Noun
- **PP**: Prepositional Phrase
- **VP**: Verb Phrase
- **CV**: Complex Verb
- **P**: Preposition
- **DT**: Determiner

Production Rules:
- **S** → **NP** **VP**
- **NP** → **CN** | **CN PP**
- **VP** → **CV** | **CV PP**
- **PP** → **P** **NP**
- **CN** → **DT** **N**
- **CV** → **V** | **V NP**
- **DT** → **a** | **the**
- **N** → **boy** | **girl** | **flower** | **telescope**
- **V** → **touched** | **likes** | **sees** | **gives**
- **P** → **with** | **to**
A Grammar for a Fragment of English

\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow CN \mid CN \ PP \\
VP & \rightarrow CV \mid CV \ PP \\
PP & \rightarrow P \ NP \\
CN & \rightarrow DT \ N \\
CV & \rightarrow V \mid V \ NP \\
DT & \rightarrow a \mid the \\
N & \rightarrow boy \mid girl \mid flower \mid telescope \\
V & \rightarrow touches \mid likes \mid sees \mid gives \\
P & \rightarrow with \mid to \\
\end{align*}
\]

\[
S \Rightarrow NP \ VP \\
\Rightarrow CN \ PP \ VP \\
\Rightarrow DT \ N \ PP \ VP \\
\Rightarrow a \ N \ PP \ VP \\
\Rightarrow \ldots \\
\Rightarrow a \ boy \ with \ a \ flower \ VP \\
\Rightarrow a \ boy \ with \ a \ flower \ CV \ PP \\
\Rightarrow \ldots \\
\Rightarrow a \ boy \ with \ a \ flower \ sees \ a \ girl \ with \ a \ telescope
\]
This structure is for the interpretation where the boy is seeing with the telescope!
This is for the interpretation where the girl is carrying a telescope.
A set of rules can assign multiple structures to the same string.
Which rule one chooses determines the eventual structure.

- $VP \rightarrow CV \mid CV \, PP$
- $CV \rightarrow V \mid V \, NP$
- $NP \rightarrow CN \mid CN \, PP$
- $\cdots \left[ VP \left[ CV \, sees \left[ NP \, a \, girl \right] \left[ PP \, with \, a \, telescope \right]\right]\right]$
- $\cdots \left[ VP \left[ CV \, sees \right] \left[ NP \left[ CN \, a \, girl \right] \left[ PP \, with \, a \, telescope \right]\right]\right]$

(Not all brackets are shown!)
Other Examples of Grammar Applications

- **Programming Languages**
  - Users need to know how to “generate” correct programs.
  - Compilers need to know how to “check” and “translate” programs.

- **XML Documents**
  - Documents need to have a structure defined by a DTD grammar.

- **Natural Language Processing, Machine Translation**
A Grammar is a 4-tuple $G = (V, \Sigma, R, S)$ where
- $V$ is a finite set of variables
- $\Sigma$ is a finite set of terminals, disjoint from $V$.
- $R$ is a set of rules of the $X \to Y$
- $S \in V$ is the start variable

In general $X \in (V \cup \Sigma)^+$ and $Y \in (V \cup \Sigma)^*$

A context-free grammar is a grammar where all rules have $X \in V$ (remember $V \subset (V \cup \Sigma)^+$)
- The substitution is independent of the context $V$ appears in.

The right hand side of the rules can be any combination of variables and terminals, including $\epsilon$ (hence $Y \in (V \cup \Sigma)^*$).
**Formal Definition of a Grammar**

- If \( u, v \) and \( w \) are strings of variables and terminals and \( A \rightarrow w \) is a rule of the grammar, we say \( uAv \) yields \( uwv \), notated as \( uAv \Rightarrow uwv \).

- We say \( u \) derives \( v \), notated as, \( u \Rightarrow^* v \), if either:
  - \( u = v \), or
  - a sequence \( u_1, u_2, \ldots, u_k, k \geq 0 \) exists such that \( u \Rightarrow u_1 \Rightarrow u_2, \ldots, \Rightarrow u_k \Rightarrow v \).
  - We call \( u, v \), and all \( u_i \) as sentential forms.

- The language of the grammar is
  \[ \{ w \in \Sigma^* \mid S \Rightarrow^* w \} \]
Consider once again the language
\[ L = \{ w \mid n_a(w) = n_b(w) \} \].

The grammar for this language is
\[ G = (\{ S \}, \{ a, b \}, R, S) \] with \( R \) as follows:
1. \( S \rightarrow aSb \)
2. \( S \rightarrow bSa \)
3. \( S \rightarrow SS \)
4. \( S \rightarrow \epsilon \)

From now we will only list the productions, the others will be implicit.

We will also combine productions with the same left-hand side using | symbol.

\[ S \rightarrow aSb \mid bSa \mid SS \mid \epsilon \]
Designing Context Free Grammars

- \( L = \{ w \mid n_a(w) = n_b(w) \} \).

- \( S \rightarrow aSb \mid bSa \mid SS \mid \epsilon \)

Clearly the strings generated by \( G \) have equal number of \( a \)'s and \( b \)'s. (Obvious from the rules!)

We also have to show that all strings in \( L \) can be generated with this grammar.
DESIGNING CONTEXT FREE GRAMMARS

ASSERTION

Grammar G with $R = \{ S \rightarrow aSb \mid bSa \mid SS \mid \epsilon \}$ generates $L = \{ w \mid n_a(w) = n_b(w) \}$.

PROOF (BY INDUCTION)

- The grammar generates the basis strings of $\epsilon$, $ab$ and $ba$.
- All other strings in $L$ have even length and can be in one of the 4 possible forms:
  1. $awb$ ($w \in \Sigma^*$)
  2. $bwa$
  3. $awa$
  4. $bwb$
Proof (continued)

- Assume that $G$ generates all strings of equal number of $a$’s and $b$’s of (even) length $n$.
- Consider a string like $awb$ of length $n + 2$.
- $awb$ will be generated from $w$ by using the rule $S \rightarrow aSb$ provided $S \Rightarrow^* w$.
- But $w$ is of length $n$, so $S \Rightarrow^* w$ by the induction hypothesis.
- There is a symmetric argument for strings like $bwa$. 
Consider a string like $awa$. Clearly $w \not\in L$. Consider (symbols of) this string annotated as follows

$$0a_1 \cdots -1 a_0$$

where the subscripts after a prefix $v$ of $awa$ denotes $n_a(v) - n_b(v)$.

Think of this as count starting as 0, each $a$ adding one and each $b$ subtracting 1. We should end with 0 at the end.

Note that right after the first symbol we have 1 and right before the last $a$ we must have $-1$.

Somewhere along the string (in $w$) the counter crosses 0.
Somewhere along the string (in \( w \)) the counter crosses 0.

\[
\begin{align*}
\underbrace{u}_{0} a_{1} \cdots x_{0} y \cdots -1 & a_{0} \quad x, y \in \{a, b\} \\
\underbrace{v}_{v}
\end{align*}
\]

So \( u \) and \( v \) have equal numbers of \( a \)'s and \( b \)'s and are shorter.

\( u, v \in L \) by the induction hypothesis and the rule \( S \rightarrow SS \) generates \( awa = uv \), given \( S \not \rightarrow u \) and \( S \not \rightarrow v \)

There is a symmetric argument for strings like \( bwb \).