THE UNIVERSITY OF NEWCASTLE AUSTRALIA
FACULTY OF ENGINEERING AND BUILT ENVIRONMENT
www.newcastle.edu.au

Theory of Computation
Week 6

Much of the material on this slides comes from the recommended textbook by Elaine Rich

Announcements

- Midterm
  - **Schedule:** 06/04/2020 (Tuesday) 8:00 ~ 9:00 (online)
  - **Syllabus:** Topics covered in Week 01 to Week 05 (lecture/tutorial)

- Pumping Lemma Session
  - Collaborate Session on 01/04/2020 (Wednesday) 11:00 ~ 12:00

Detailed content

Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques
- ✓ Week 2 – Languages and strings. Hierarchies. Computation. Closure properties
- ✓ Week 3 – Finite State Machines: non-determinism vs. determinism
- ✓ Week 4 – Regular languages: expressions and grammars
- ✓ Week 5 – Non regular languages: pumping lemma. Closure

Week 6 – Context-free languages: grammars and parse trees

- ❑ Week 7 – Pushdown automata
- ❑ Week 8 – Non context-free languages: pumping lemma and decidability. Closure
- ❑ Week 9 – Decidable languages: Turing Machines
- ❑ Week 10 – Church-Turing thesis and the unsolvability of the Halting Problem
- ❑ Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)
- ❑ Week 12 – Revision of the hierarchy. Safety-critical systems
- ❑ Week 13 – Extra revision (if needed)

Week 06 Videos

You already know

- ❑ Context Free Grammar
  - ❑ How it is different from Regular Grammar
  - ❑ Why it is called context free
  - ❑ Formal definition and example
  - ❑ Regular languages are proper subset of context free languages
- ❑ Parse Tree
  - ❑ Derivation
  - ❑ Weak/Strong generative capacity

Videos to watch before lecture

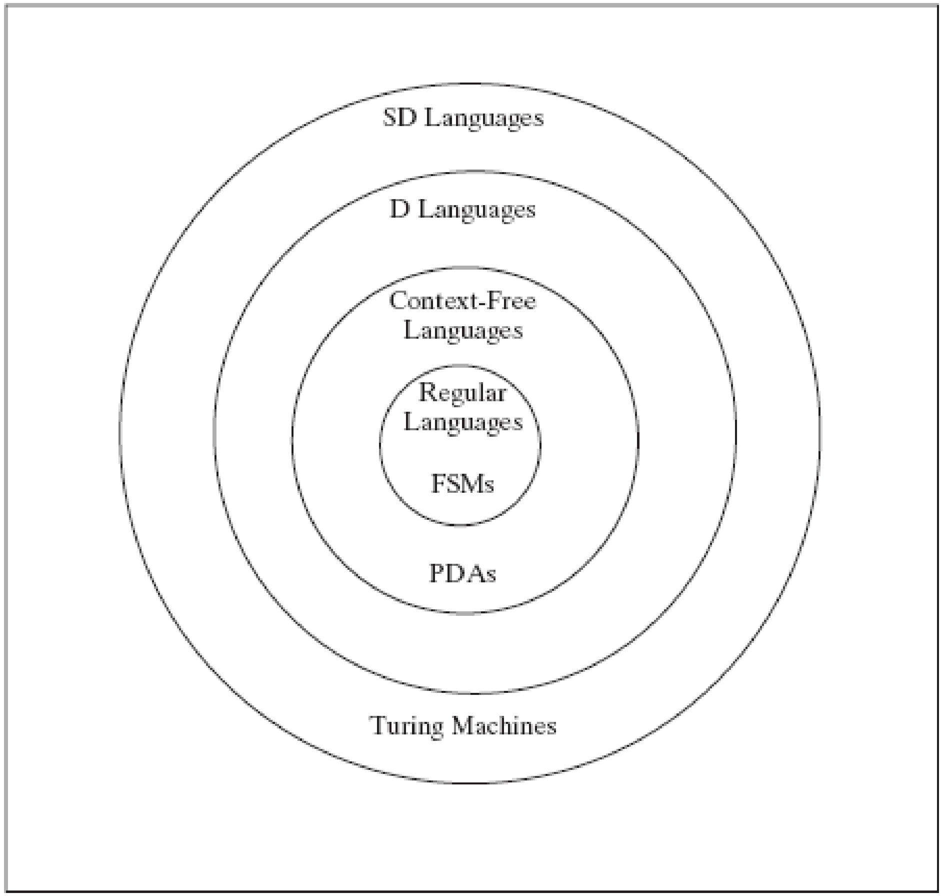
Additional videos to watch for this week

Week 06 Lecture Outline

Context-free languages: grammars and parse trees

- ❑ Rewrite Systems
- ❑ Grammars
  - ❑ Context Free Grammars
- ❑ Context Free Languages
- ❑ Properties of CFG
  - ❑ Recursion
  - ❑ Self-Embedding
- ❑ Backus Naur Form (BNF)
- ❑ Designing CFGs
- ❑ Simplifying CFGs
- ❑ Parse Tree
- ❑ Ambiguity

THE HIERARCHY



REWRITE SYSTEMS

A ***rewrite system*** (or ***production system*** or ***rule-based system***) is:

- a list of rules, and
- an algorithm for applying them.

Each rule has a left-hand side and a right hand side.

Example rules:

$$S \rightarrow \mathtt{a}S\mathtt{b}$$
$$\mathtt{a}S \rightarrow \varepsilon$$
$$\mathtt{a}S\mathtt{b} \rightarrow \mathtt{b}S\mathtt{ab}S\mathtt{a}$$

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

SIMPLE REWRITE

simple-rewrite(*R*: *rewrite system*, *w*: *initial string*) =

1. Set *working-string* to *w*.
2. Until told by *R* to halt do:
   a. Match the LHS of some rule against some part of *working-string*.
   b. Replace the matched part of *working-string* with the RHS of the rule that was matched.
3. Return *working-string*.

If *simple-rewrite*(R,w) can return some string *s* then we will say that R

can **derive** *s* from w or there exists a **derivation** in R of s from w.

SIMPLE REWRITE

A rewrite system formalism specifies:

- The form of the rules
- How *simple-rewrite* works:
  - How to choose rules?
  - When to quit?

SIMPLE REWRITE

$w = S\mathtt{a}S$

Rules:

[1] $S \rightarrow \mathtt{a}S\mathtt{a}$

[2] $S \rightarrow \mathtt{b}S\mathtt{b}$

[3] $\mathtt{a}S \rightarrow \varepsilon$

What order to apply the rules?

When to quit?

RULE BASED SYSTEMS

- Expert systems
- Cognitive modeling
- Business practice modeling
- General models of computation
- **Grammars**

[Source: Wikipedia]

GRAMMARS

A grammar is a set of rules that are stated in terms of two alphabets:

- a ***terminal alphabet***, Σ, that contains the symbols that make up the strings in $L(G)$, and
- a ***nonterminal alphabet***, the elements of which will function as working symbols that will be used while the grammar is operating. These symbols will disappear by the time the grammar finishes its job and generates a string.

A grammar has a unique **start symbol**, often called S.

USING A GRAMMAR TO DERIVE A STRING

Simple-rewrite (G, S) will generate the strings in $L(G)$.

We will use the symbol $\Rightarrow$ to indicate steps in a derivation.

A derivation could begin with:

$$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow \ldots$$

GENERATING MANY STRINGS

- Multiple rules may match.

Given: $S \rightarrow \mathtt{a}S\mathtt{b}$, $S \rightarrow \mathtt{b}S\mathtt{a}$, and $S \rightarrow \varepsilon$

Derivation so far: $S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow$

Three choices at the next step:

$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow \mathtt{aaa}S\mathtt{bbb}$ (using rule 1),
$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow \mathtt{aab}S\mathtt{abb}$ (using rule 2),
$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow \mathtt{aabb}$ (using rule 3).

GENERATING MANY STRINGS

- One rule may match in more than one way.

Given: $S \rightarrow \mathtt{a}TT\mathtt{b}$, $T \rightarrow \mathtt{b}T\mathtt{a}$, and $T \rightarrow \varepsilon$

Derivation so far: $S \Rightarrow \mathtt{a}TT\mathtt{b} \Rightarrow$

Two choices at the next step:

$S \Rightarrow \mathtt{a}\underline{T}T\mathtt{b} \Rightarrow \mathtt{ab}T\mathtt{a}T\mathtt{b} \Rightarrow$
$S \Rightarrow \mathtt{a}T\underline{T}\mathtt{b} \Rightarrow \mathtt{a}T\mathtt{b}T\mathtt{ab} \Rightarrow$

GENERATING MANY STRINGS

May stop when:

1. The working string no longer contains any nonterminal symbols (including, when it is ε).

In this case, we say that the working string is ***generated*** by the grammar.

Example:

$$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow \mathtt{aabb}$$

GENERATING MANY STRINGS

May stop when:

2. There are nonterminal symbols in the working string but none of them appears on the left-hand side of any rule in the grammar.

In this case, we have a blocked or non-terminated derivation but no generated string.

Example:

Rules: $S \rightarrow \mathtt{a}S\mathtt{b}$, $S \rightarrow \mathtt{b}T\mathtt{a}$, and $S \rightarrow \varepsilon$

Derivations: $S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{ab}T\mathtt{ab} \Rightarrow$ [blocked]

GENERATING MANY STRINGS

It is possible that neither (1) nor (2) is achieved.

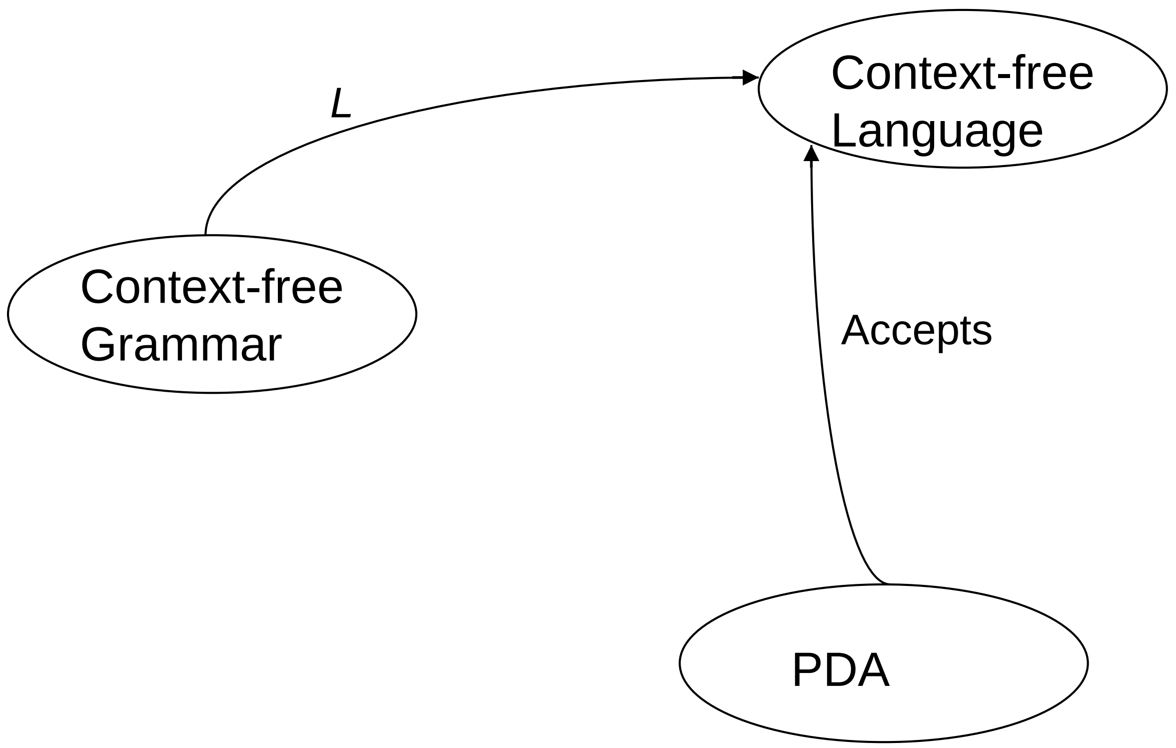
Example:

G contains only the rules $S \to B\mathtt{a}$ and $B \to \mathtt{b}B$, with S the start symbol.

Then all derivations proceed as:

$$S \Rightarrow B\mathtt{a} \Rightarrow \mathtt{b}B\mathtt{a} \Rightarrow \mathtt{bb}B\mathtt{a} \Rightarrow \mathtt{bbb}B\mathtt{a} \Rightarrow \mathtt{bbbb}B\mathtt{a} \Rightarrow \ldots$$

CONTEXT-FREE GRAMMARS, LANGUAGES, AND PDAS



MORE POWERFUL GRAMMARS

Regular grammars must always produce strings one character at a time, moving left to right.

But it may be more natural to describe generation more flexibly.

Example 1: $L = \mathtt{ab^*a}$

$S \to \mathtt{a}B\mathtt{a}$		$S \to \mathtt{a}B$
$B \to \mathtt{b}B$	vs.	$B \to \mathtt{b}B$
$B \to \varepsilon$		$B \to \mathtt{a}$

Example 2: $L = \{\mathtt{a}^n\mathtt{b}^*\mathtt{a}^n,\ n \geq 0\}$

$S \to B$
$S \to \mathtt{a}S\mathtt{a}$
$B \to \mathtt{b}B$
$B \to \varepsilon$

CONTEXT-FREE GRAMMARS

No restrictions on the form of the right hand sides.

$$S \to \mathtt{ab}D\mathtt{e}FG\mathtt{ab}$$

But require single non-terminal on left hand side.

$$S \to$$

but not $ASB \to$

A^nB^n

A^nB^n

$$S \rightarrow \varepsilon$$
$$S \rightarrow \mathtt{a}S\mathtt{b}$$

BALANCED PARENTHESES

BALANCED PARENTHESES

$$S \rightarrow \varepsilon$$
$$S \rightarrow SS$$
$$S \rightarrow (S)$$

CONTEXT-FREE GRAMMARS

A context-free grammar G is a quadruple, (V, Σ, R, S), where:

- V is the rule alphabet, which contains nonterminals and terminals.
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V - \Sigma) \times V^*$,
- S (the start symbol) is an element of $V - \Sigma$.

Example:

$(\{S, \mathtt{a}, \mathtt{b}\}, \ \{\mathtt{a}, \mathtt{b}\}, \ \{S \rightarrow \mathtt{a}\, S\, \mathtt{b}, S \rightarrow \varepsilon\}, \ S)$

DERIVATIONS

$x \Rightarrow_G y$ iff $x = \alpha A \beta$

and $A \rightarrow \gamma$ is in R

$y = \alpha \gamma \beta$

$w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_n$ is a derivation in G.

Let $\Rightarrow_G{}^*$ be the reflexive, transitive closure of $\Rightarrow_G$.

Then the language generated by G, denoted $L(G)$, is:

$\{w \in \Sigma^* : S \Rightarrow_G{}^* w\}$.

DERIVATIONS

Example:

Let $G = (\{S, \mathtt{a}, \mathtt{b}\}, \ \{\mathtt{a}, \mathtt{b}\}, \ \{S \rightarrow \mathtt{a}\, S\, \mathtt{b},\ S \rightarrow \varepsilon\},\ S)$

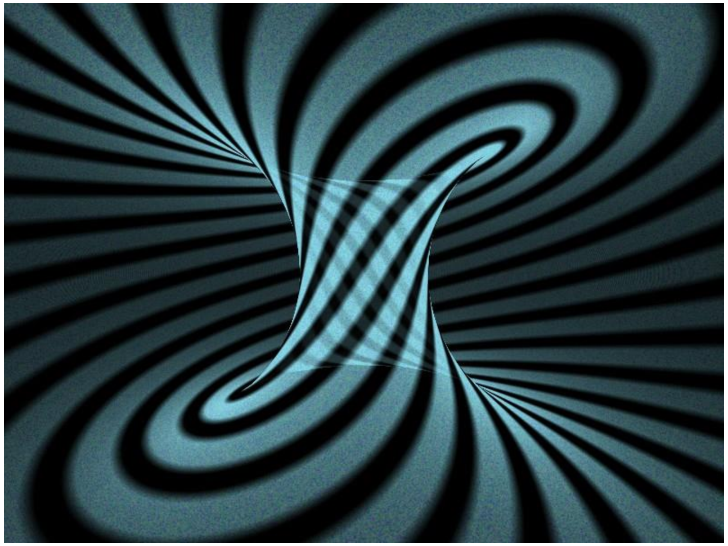
$S \Rightarrow \mathtt{a}\, S\, \mathtt{b} \Rightarrow \mathtt{aa}\, S\, \mathtt{bb} \Rightarrow \mathtt{aaa}\, S\, \mathtt{bbb} \Rightarrow \mathtt{aaabbb}$

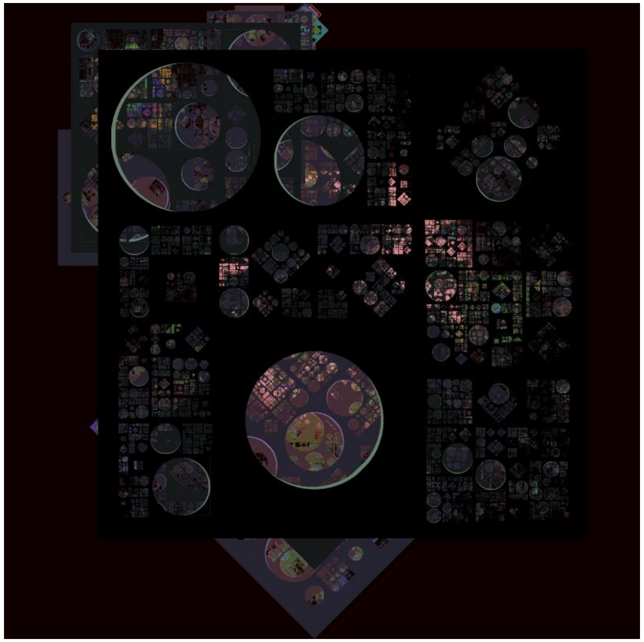
$S \Rightarrow^* \mathtt{aaabbb}$

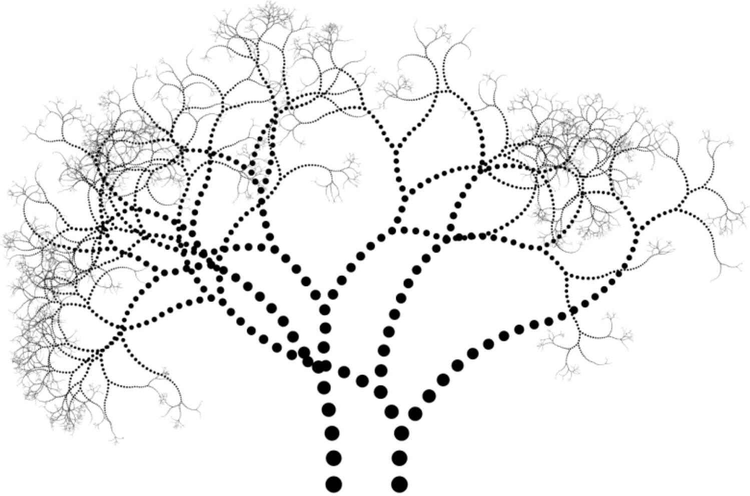
CONTEXT-FREE LANGUAGES

A language *L* is ***context-free*** if and only if it is generated by some context-free grammar *G*.

Context Free Art: CFDG



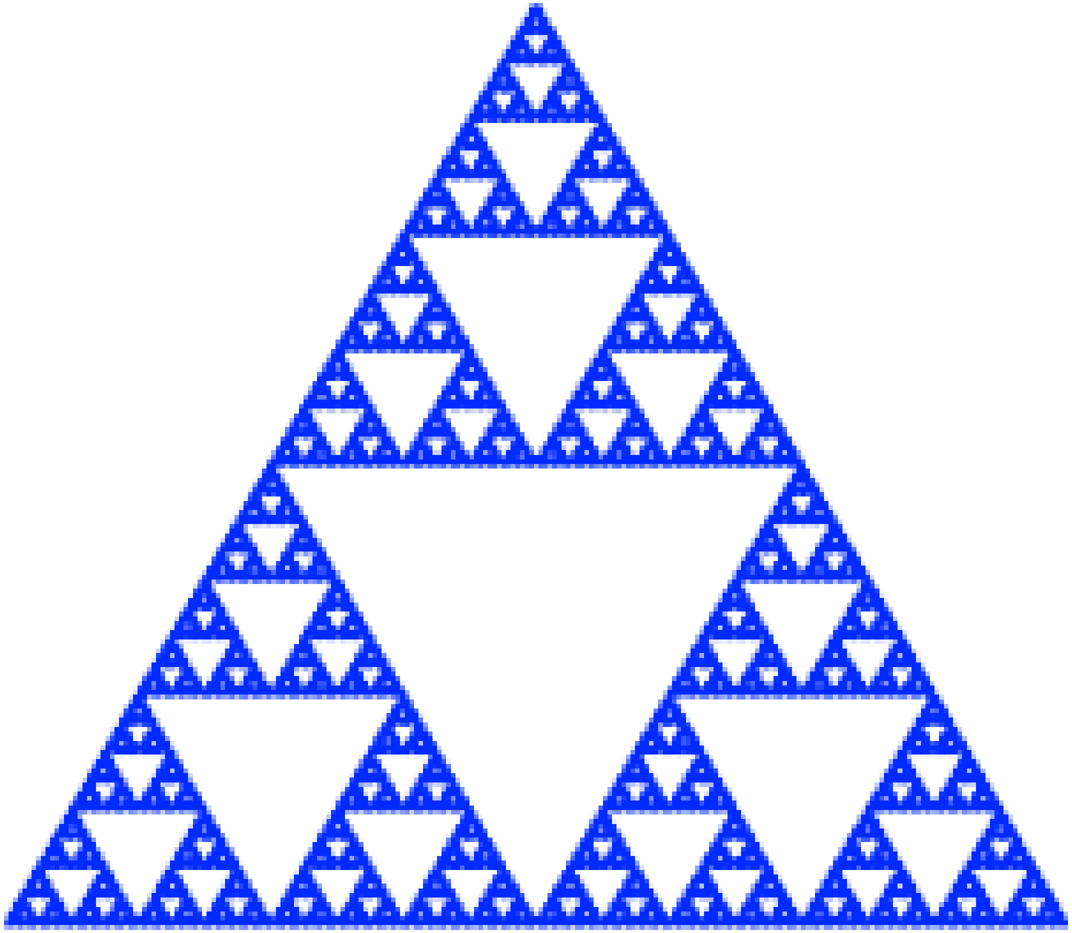




Context Free
http://www.contextfreeart.org/

Source:http://www.contextfreeart.org/

Fractals:



Lindenmayer Systems

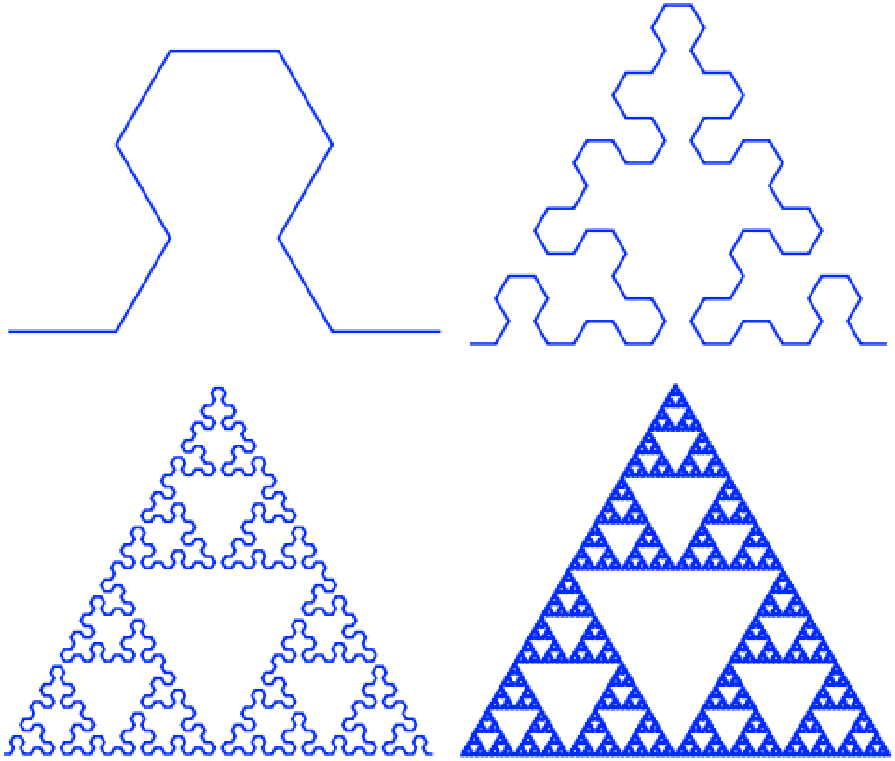
L-system:

Rules:

A-> B-A-B
B-> A+B+A

Start Symbol: A

A and B both mean "draw forward",
+ means "turn to the left 60°",
− means "turn to the right 60°"



RECURSIVE GRAMMAR RULES

- A rule is ***recursive*** iff it is $X \to w_1 Y w_2$, where:
  $Y \Rightarrow^* w_3 X w_4$ for some w_1, w_2, w_3, and w_4 are in V^*.

- A grammar is recursive iff it contains at least one recursive rule.

- Examples: $S \to (S)$

RECURSIVE GRAMMAR RULES

- A rule is ***recursive*** iff it is $X \rightarrow w_1 Y w_2$, where:
  $Y \Rightarrow^* w_3 X w_4$ for some w_1, w_2, w_3, and w_4 are in V^*.
- A grammar is recursive iff it contains at least one recursive rule.
- Examples: $S \rightarrow (S)$ $S \rightarrow (T)$

RECURSIVE GRAMMAR RULES

- A rule is ***recursive*** iff it is $X \to w_1 Y w_2$, where:
  $Y \Rightarrow^* w_3 X w_4$ for some w_1, w_2, w_3, and w_4 are in V^*.

- A grammar is recursive iff it contains at least one recursive rule.

- Examples: $S \to (S)$ $\quad\quad S \to (T)$
  $T \to (S)$

SELF-EMBEDDING GRAMMAR RULES

- A rule in a grammar G is ***self-embedding*** iff it is :

  $X \rightarrow w_1 Y w_2$, where $Y \Rightarrow^* w_3 X w_4$ and

  both $w_1 w_3$ and $w_4 w_2$ are in Σ^+.

- A grammar is self-embedding iff it contains at least one self-embedding rule.

- Example: $S \rightarrow \mathtt{a}S\mathtt{a}$ is self-embedding

- Self-embedding grammars are able to define languages like

  Bal, A^nB^n and of the form uv^ixy^iz

RECURSIVE AND SELF-EMBEDDING GRAMMAR RULES

- A rule in a grammar G is ***self-embedding*** iff it is :

  $X \rightarrow w_1 Y w_2$, where $Y \Rightarrow^* w_3 X w_4$ and

  both $w_1 w_3$ and $w_4 w_2$ are in Σ^+.

- A grammar is self-embedding iff it contains at least one self-embedding rule.

- Example:

$S \rightarrow \mathtt{a}S\mathtt{a}$	is self-embedding
$S \rightarrow \mathtt{a}S$	is recursive but not self-embedding
$S \rightarrow \mathtt{a}T$	
$T \rightarrow S\mathtt{a}$	is self-embedding

(we require that a nonempty string be generated on each side of the nested X)

RECURSIVE AND SELF-EMBEDDING GRAMMAR RULES

- If a grammar *G* is **not** self-embedding then *L*(*G*) is regular.

- A grammar G is self-embedding does not guarantee that L(G) isn't regular. Another different grammar G′ that also defines L(G) is not self embedding.

  e.g.: G_1=({S,a},{a},{S$\rightarrow\varepsilon$, S$\rightarrow$a, S$\rightarrow$aSa}, S)

- If a language *L* has the property that every grammar that defines it is self-embedding, then *L* is **not** regular.

PalEven = $\{ww^R : w \in \{\mathtt{a}, \mathtt{b}\}^*\}$

PalEven = $\{ww^R : w \in \{\mathtt{a}, \mathtt{b}\}^*\}$

$G = \{\{S, \mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}, R, S\}$, where:

$$R = \{\, S \to \mathtt{a}S\mathtt{a}$$
$$S \to \mathtt{b}S\mathtt{b}$$
$$S \to \varepsilon \,\}.$$

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

EQUAL NUMBERS OF a's AND b's

Let $L = \{w \in \{\texttt{a}, \texttt{b}\}^* : \#_{\texttt{a}}(w) = \#_{\texttt{b}}(w)\}$.

EQUAL NUMBERS OF a's AND b's

Let $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*: \#_{\mathtt{a}}(w) = \#_{\mathtt{b}}(w)\}$.

$G = \{\{S, \mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}, R, S\}$, where:

$$
\begin{aligned}
R = \{\ & S \to \mathtt{a}S\mathtt{b} \\
& S \to \mathtt{b}S\mathtt{a} \\
& S \to SS \\
& S \to \varepsilon\ \}.
\end{aligned}
$$

ARITHMETIC EXPRESSIONS

$G = (V, \Sigma, R, E)$, where

$V = \{+, *, (,), \texttt{id}, E\}$,

$\Sigma = \{+, *, (,), \texttt{id}\}$,

$R = \{$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow \texttt{id} \ \}$

BACKUS NAUR FORM (BNF)

A notation for writing practical context-free grammars

- The symbol | should be read as “or”.

  Example: $S \rightarrow \mathtt{a}S\mathtt{b} \mid \mathtt{b}S\mathtt{a} \mid SS \mid \varepsilon$

- Allow a nonterminal symbol to be any sequence of characters surrounded by angle brackets.

  Examples of nonterminals:

  <program>
  <variable>

BNF for a Java Fragment


```
<block> ::= {<stmt-list>} | {}
<stmt-list> ::= <stmt> | <stmt-list> <stmt>
<stmt> ::= <block> | while (<cond>) <stmt> |
          if (<cond>) <stmt> |
          do <stmt> while (<cond>); |
          <assignment-stmt>; |
          return | return <expression> |
          <method-invocation>;
```

ENGLISH

$S \rightarrow NP\ VP$

$NP \rightarrow$ `the` $Nominal$ | `a` $Nominal$ | $Nominal$ |
$ProperNoun$ | $NP\ PP$

$Nominal \rightarrow N$ | $Adjs\ N$

$N \rightarrow$ `cat` | `dogs` | `bear` | `girl` | `chocolate` | `rifle`

$ProperNoun \rightarrow$ `Chris` | `Fluffy`

$Adjs \rightarrow Adj\ Adjs$ | Adj

$Adj \rightarrow$ `young` | `older` | `smart`

$VP \rightarrow V$ | $V\ NP$ | $VP\ PP$

$V \rightarrow$ `like` | `likes` | `thinks` | `shots` | `smells`

$PP \rightarrow Prep\ NP$

$Prep \rightarrow$ `with`

DESIGNING CONTEXT-FREE GRAMMARS

- Generate related regions together.

  $A^n B^n$

- Generate concatenated regions:

  $$A \to BC$$

- Generate outside in:

  $$A \to \mathrm{a}A\mathrm{b}$$

CONCATENATING INDEPENDENT LANGUAGES

Let $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m : n, m \geq 0\}$.

The $\mathtt{c}^m$ portion of any string in L is completely independent of the $\mathtt{a}^n\mathtt{b}^n$ portion, so we should generate the two portions separately and concatenate them together.

CONCATENATING INDEPENDENT LANGUAGES

Let $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m : n, m \geq 0\}$.

The $\mathtt{c}^m$ portion of any string in L is completely independent of the $\mathtt{a}^n\mathtt{b}^n$ portion, so we should generate the two portions separately and concatenate them together.

$G = (\{S, N, C, \mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, S\}$ where:

$R = \{$ $S \rightarrow NC$ /* Generate two independent portions*/
$N \rightarrow \mathtt{a}N\mathtt{b}$ /* Generate a^nb^n portion, from outside in*/
$N \rightarrow \varepsilon$
$C \rightarrow \mathtt{c}C$ /* Generate c^m portion */
$C \rightarrow \varepsilon$ $\}$.

$L = \{a^{n_1}b^{n_1}a^{n_2}b^{n_2}\dots a^{n_k}b^{n_k}$: $k \geq 0$ and $\forall i\ (n_i \geq 0)\}$

Examples of strings in L: ε, `abab`, `aabbaaabbbabab`

Note that $L = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}^*$.

$L = \{a^{n_1}b^{n_1}a^{n_2}b^{n_2}\ldots a^{n_k}b^{n_k} : k \geq 0 \text{ and } \forall i\ (n_i \geq 0)\}$

Examples of strings in L: ε, `abab`, `aabbaaabbbabab`

Note that $L = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}^*$.

$G = (\{S, M, \mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}, R, S\}$ where:

$$
\begin{aligned}
R = \{\ & S \to MS \\
& S \to \varepsilon \\
& M \to \mathtt{a}M\mathtt{b} \\
& M \to \varepsilon\}.
\end{aligned}
$$

UNEQUAL a's AND b's

$L = \{a^n b^m : n \neq m\}$

$G = (V, \Sigma, R, S)$, where

$V = \{a, b, S, \quad\quad \}$,

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

$R =$

UNEQUAL a's AND b's

$L = \{\mathtt{a}^n\mathtt{b}^m : n \neq m\}$

$G = (V, \Sigma, R, S)$, where

$V = \{\mathtt{a}, \mathtt{b}, S, A, B\}$,

$\Sigma = \{\mathtt{a}, \mathtt{b}\}$,

$R =$

$S \rightarrow A$ /* more a's than b's

$S \rightarrow B$ /* more b's than a's

$A \rightarrow \mathtt{a}$ /* at least one extra a generated

$A \rightarrow \mathtt{a}A$

$A \rightarrow \mathtt{a}A\mathtt{b}$

$B \rightarrow \mathtt{b}$ /* at least one extra b generated

$B \rightarrow B\mathtt{b}$

$B \rightarrow \mathtt{a}B\mathtt{b}$

SIMPLIFYING CONTEXT-FREE GRAMMARS

$G = (\{S, A, B, C, D, \mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}, R, S)$, where

$R =$

$$\begin{aligned}
\{\quad & S \rightarrow AB \mid AC \\
& A \rightarrow \mathtt{a}A\mathtt{b} \mid \varepsilon \\
& B \rightarrow \mathtt{a}A \\
& C \rightarrow \mathtt{b}C\mathtt{a} \\
& D \rightarrow AB \;\}
\end{aligned}$$

SIMPLIFYING CONTEXT-FREE GRAMMARS

removeunproductive(G: CFG) =

1. $G' = G$.
2. Mark every nonterminal symbol in G' as unproductive.
3. Mark every terminal symbol in G' as productive.
4. Until one entire pass has been made without any new symbol being marked do:

   For each rule $X \rightarrow \alpha$ in R do:

   If every symbol in α has been marked as productive and X has not yet been marked as productive then:

   Mark X as productive.
5. Remove from G' every unproductive symbol.
6. Remove from G' every rule that contains an unproductive symbol.
7. Return G'.

SIMPLIFYING CONTEXT-FREE GRAMMARS

removeunreachable(G: CFG) =

1. $G' = G$.
2. Mark S as reachable.
3. Mark every other nonterminal symbol as unreachable.
4. Until one entire pass has been made without any new symbol being marked do:

   For each rule $X \rightarrow \alpha A \beta$ (where $A \in V - \Sigma$) in R do:

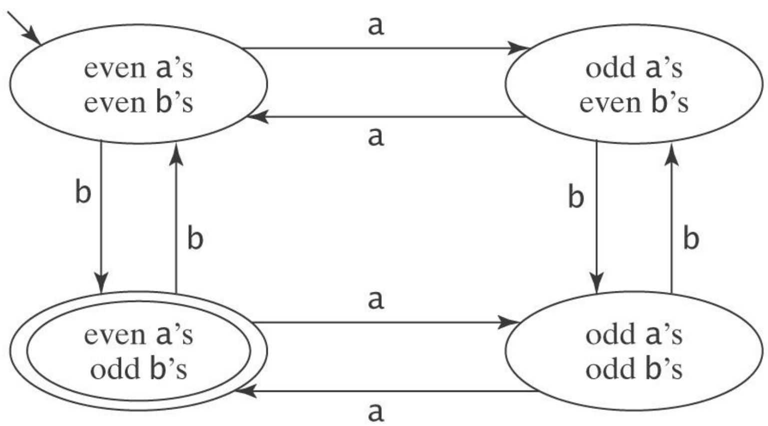
   If X has been marked as reachable and A has not then:

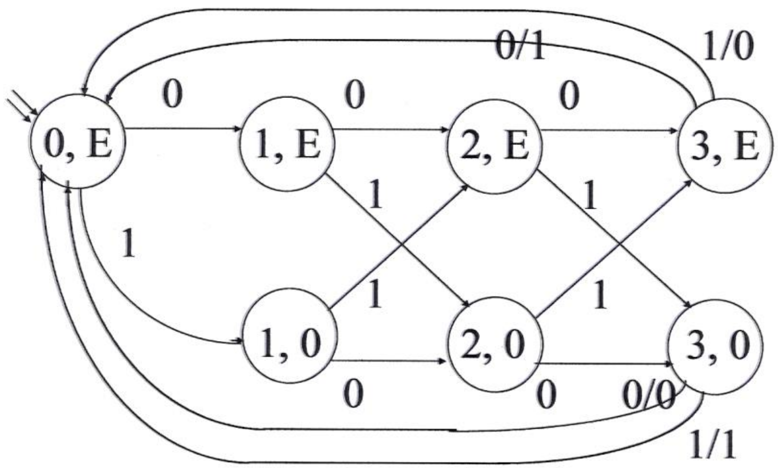
   Mark A as reachable.
5. Remove from G' every unreachable symbol.
6. Remove from G' every rule with an unreachable symbol on the left-hand side.
7. Return G'.

REGULAR VS. CONTEXT-FREE LANGUAGES

Regular languages:

We care about recognizing patterns and taking appropriate actions.

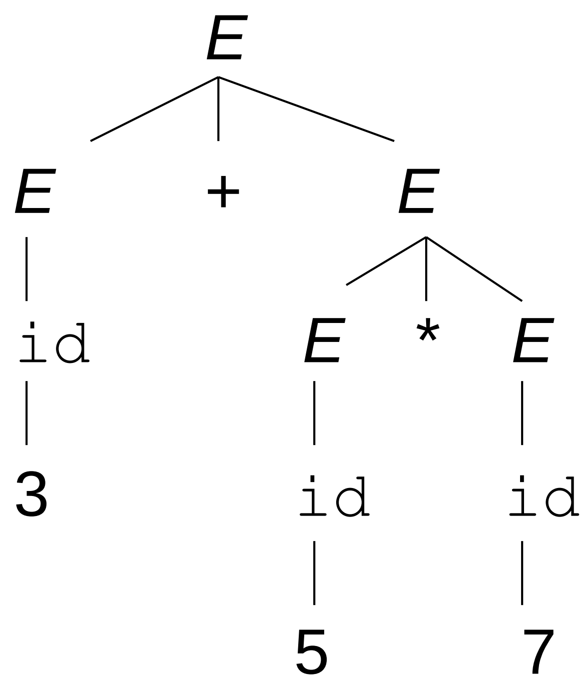




REGULAR VS. CONTEXT-FREE LANGUAGES

Context free languages:

We care about structure.



DERIVATIONS

To capture structure, we must capture the path we took through the grammar. **Derivations** do that.

Example:

$$
\begin{aligned}
S &\to \varepsilon \\
S &\to SS \\
S &\to (S)
\end{aligned}
$$

$$S \overset{1}{\Rightarrow} SS \overset{2}{\Rightarrow} (S)S \overset{3}{\Rightarrow} ((S))S \overset{4}{\Rightarrow} (())S \overset{5}{\Rightarrow} (())(S) \overset{6}{\Rightarrow} (())()$$

$$S \underset{1}{\Rightarrow} SS \underset{2}{\Rightarrow} (S)S \underset{3}{\Rightarrow} ((S))S \underset{5}{\Rightarrow} ((S))(S) \underset{4}{\Rightarrow} (())(S) \underset{6}{\Rightarrow} (())()$$

But the order of rule application doesn’t matter.

DERIVATIONS

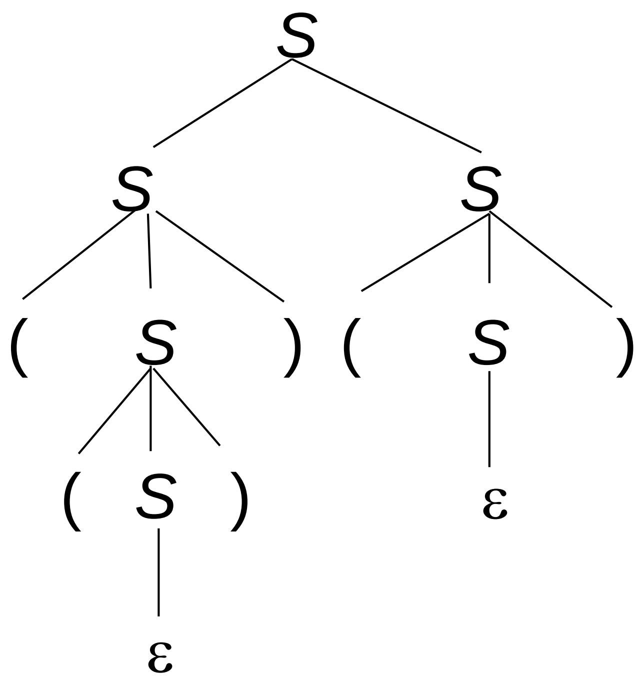
Parse trees capture essential structure:

1 2 3 4 5 6

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$$

1 2 3 5 4 6



PARSE TREES

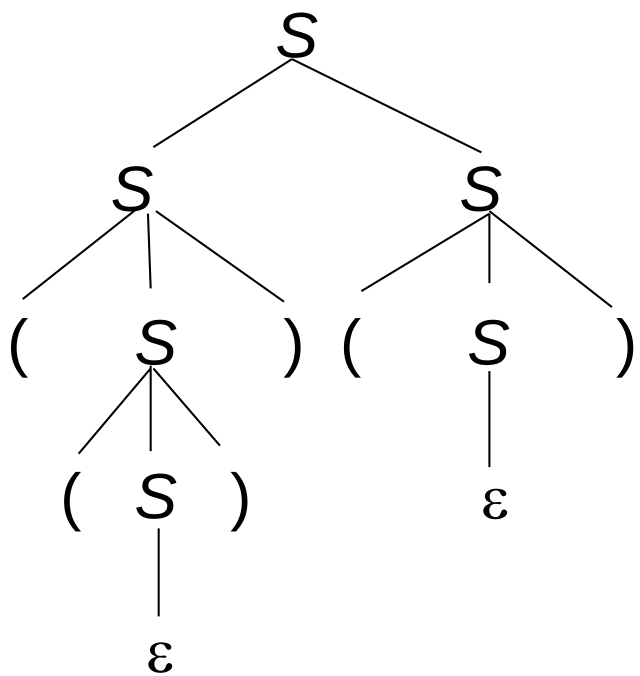
A parse tree, derived by a grammar $G = (V, \Sigma, R, S)$, is a rooted, ordered tree in which:

- Every leaf node is labeled with an element of $\Sigma \cup \{\varepsilon\}$,
- The root node is labeled S,
- Every other node is labeled with some element of:
  $V - \Sigma$, and
- If m is a nonleaf node labeled X and the children of m are labeled $x_1, x_2, \ldots, x_n$, then R contains the rule
  $X \to x_1, x_2, \ldots, x_n$.

DERIVATIONS

The process of applying a sequence of grammar rules to generate a string starting from the start symbol of the grammar.

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$



Derivation

Left-most derivation: At each step of derivation, the leftmost nonterminal in the working string is chose for expansion.

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

Right-most derivation: At each step of derivation, the rightmost nonterminal in the working string is chose for expansion.

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$

Branching Factor

Branching factor of a grammar G to be length (the number of symbols) of the longest right-hand side of any rule in G

Branching factor of any parse tree generated by G is less than or equal to the branching factor of G.

PARSE TREES

Because parse trees matter, it makes sense, given a grammar G, to distinguish between:

- G's ***weak generative capacity***, defined to be the set of strings, $L(G)$, that G generates, and
- G's ***strong generative capacity***, defined to be the set of parse trees that G generates.

AMBIGUITY

A grammar is ***ambiguous*** iff there is at least one string in $L(G)$ for which G produces more than one parse tree.

For most applications of context-free grammars, this is a problem.

ARITHMETIC EXPRESSIONS

$G = (V, \Sigma, R, E)$, where

$$V = \{+, *, (,), \texttt{id}, E\},$$

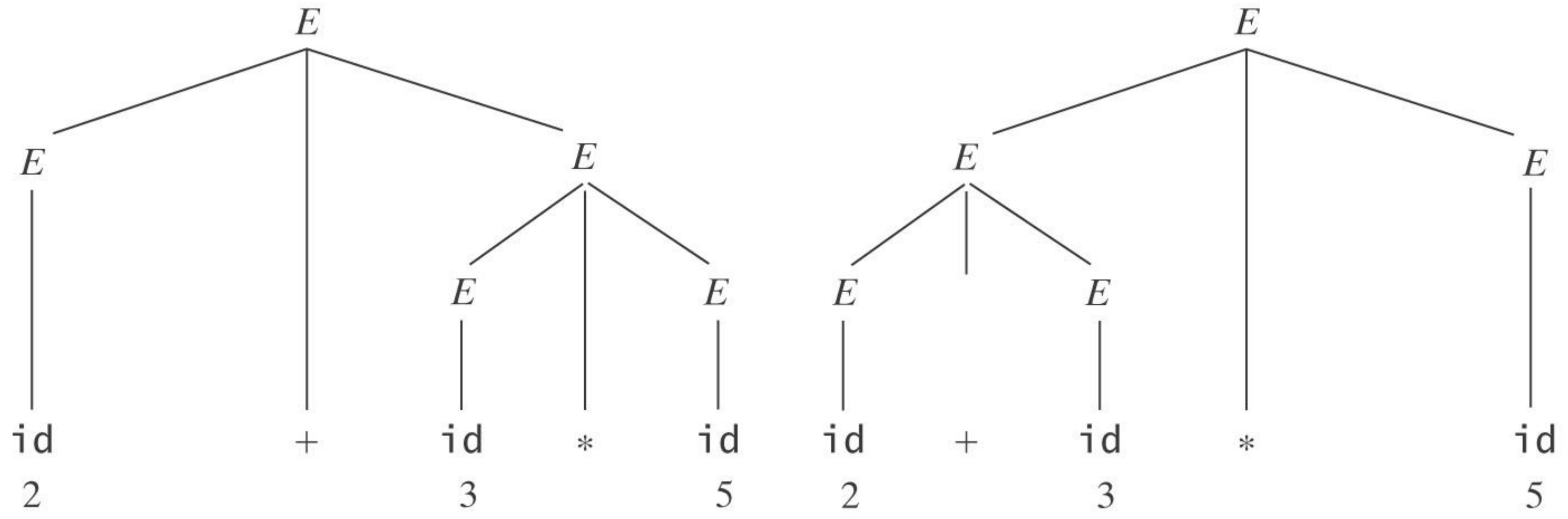
$$\Sigma = \{+, *, (,), \texttt{id}\},$$

$$\begin{aligned} R = \{ \; & E \to E + E \\ & E \to E * E \\ & E \to (E) \\ & E \to \texttt{id} \; \} \end{aligned}$$

ARITHMETIC EXPRESSIONS

$$E \to E + E$$
$$E \to E * E$$
$$E \to (E)$$
$$E \to \texttt{id}$$

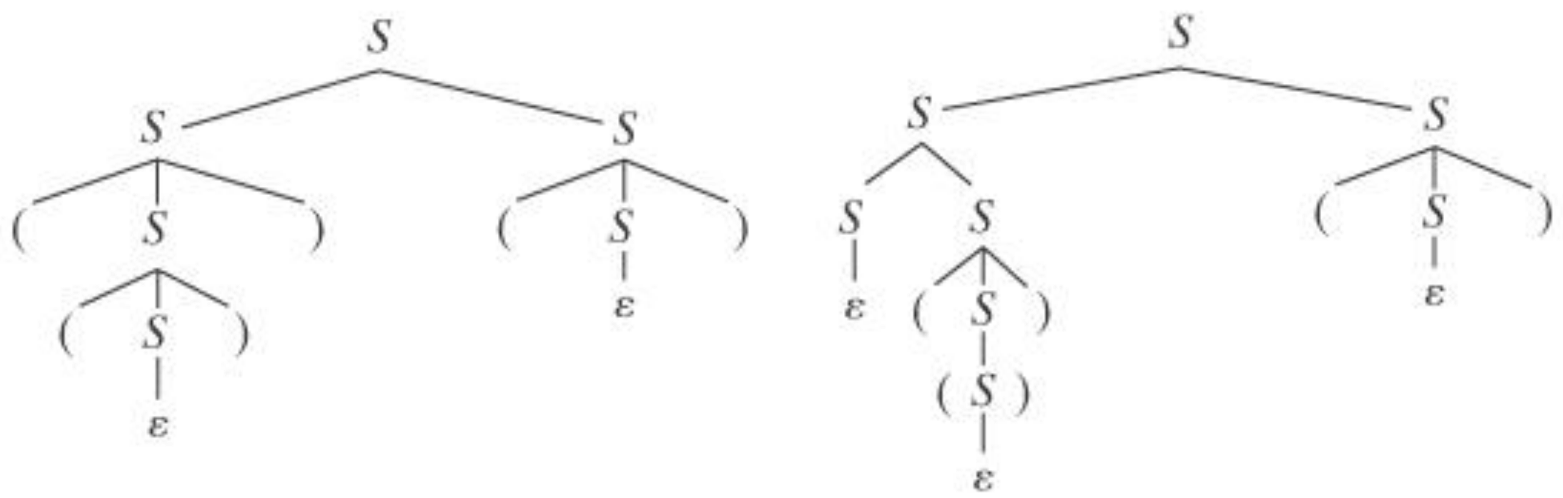
2 + 3 * 5



AMBIGUITY

$S \rightarrow \varepsilon$
$S \rightarrow SS$
$S \rightarrow (S)$

(( )) ( )



Even a Very Simple Grammar Can be Highly Ambiguous

INHERENT AMBIGUITY

Some languages have the property that every grammar for them is ambiguous. We call such languages ***inherently ambiguous***.

Example:

$L = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k: i, j, k \geq 0, i{=}j \text{ or } j{=}k\}$

$\quad = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m: n, m \geq 0\} \cup \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m: n, m \geq 0\}.$

INHERENT AMBIGUITY

$L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m\colon n, m \geq 0\} \cup \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m\colon n, m \geq 0\}$.

One grammar for L has the rules:

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow S_1\mathtt{c} \mid A$ /* Generate all strings in $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m\}$.
$A \rightarrow \mathtt{a}A\mathtt{b} \mid \varepsilon$

$S_2 \rightarrow \mathtt{a}S_2 \mid B$ /* Generate all strings in $\{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m\}$.
$B \rightarrow \mathtt{b}B\mathtt{c} \mid \varepsilon$

Consider any string of the form $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$.

L is inherently ambiguous.

INHERENT AMBIGUITY

Both of the following problems are **<u>undecidable:</u>**

- Given a context-free grammar *G*, is *G* ambiguous?
- Given a context-free language *L*, is *L* inherently ambiguous?

REDUCING AMBIGUITY

We can get rid of:

- ε rules like $S \to \varepsilon$,
- recursive rules with symmetric right-hand sides, e.g.,

  $S \to SS$

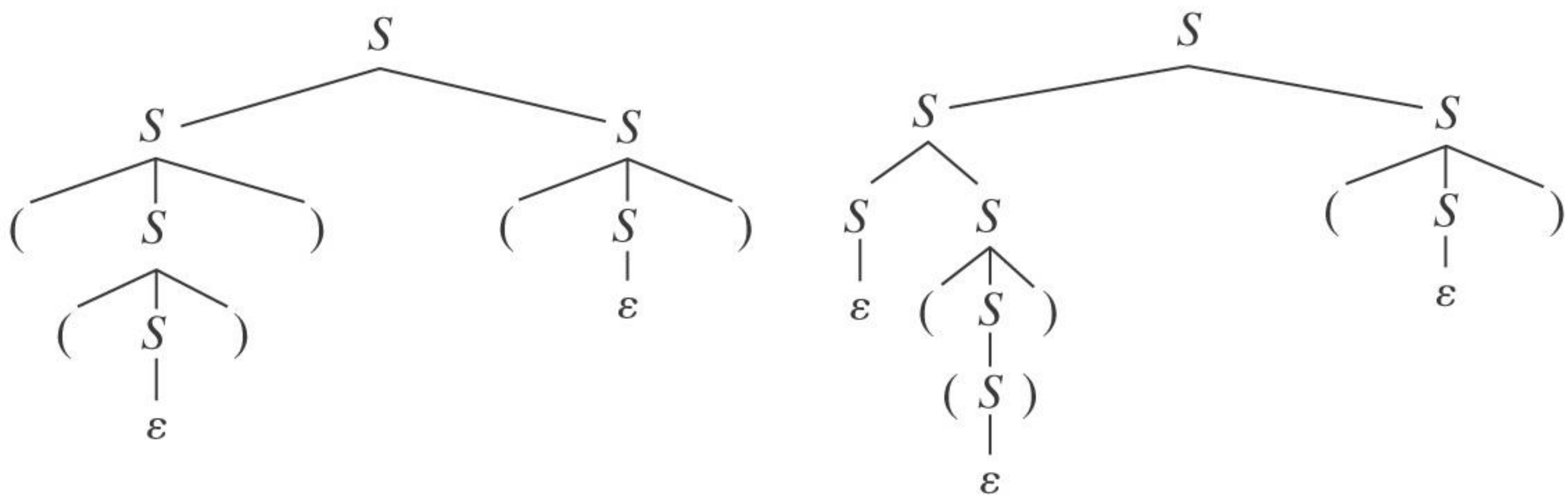
  $E \to E + E$

- rule sets that lead to ambiguous attachment of optional postfixes.

  if a then if b then s else s2

REDUCING AMBIGUITY

$S \rightarrow \varepsilon$
$S \rightarrow SS$
$S \rightarrow (S)$



REDUCING AMBIGUITY

A different grammar for the language of balanced parentheses:

$S^* \rightarrow \varepsilon$
$S^* \rightarrow S$
$S \rightarrow SS$
$S \rightarrow (S)$
$S \rightarrow ()$

The grammar is still ambiguous. Need to eliminate symmetric recursive rule S→ SS.
[See: Page 227-230]

References

- **Automata, Computability and Complexity. Theory and Applications**
  - By Elaine Rich
- Chapter 11:
  - Page : 203-224.

THE UNIVERSITY OF NEWCASTLE AUSTRALIA