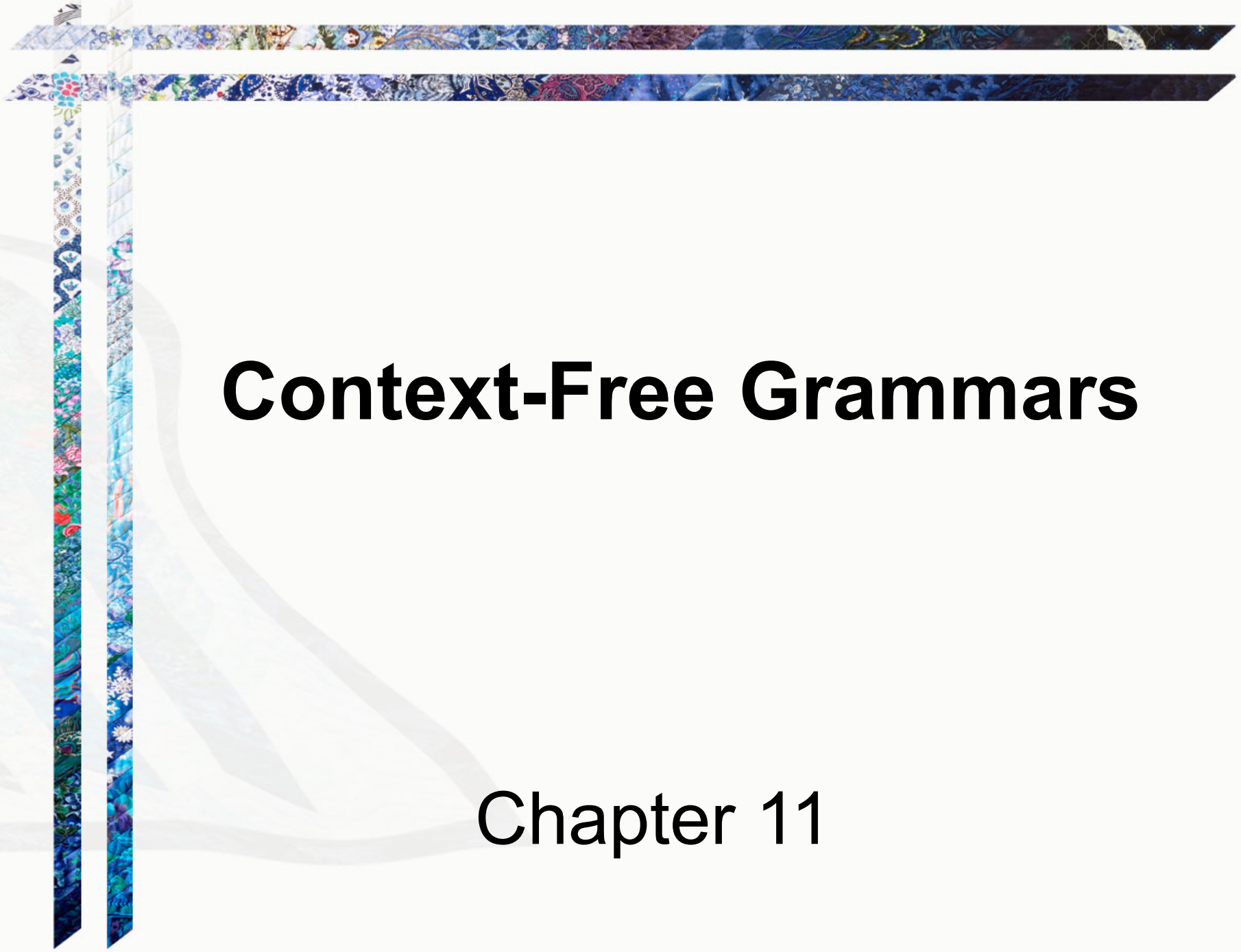
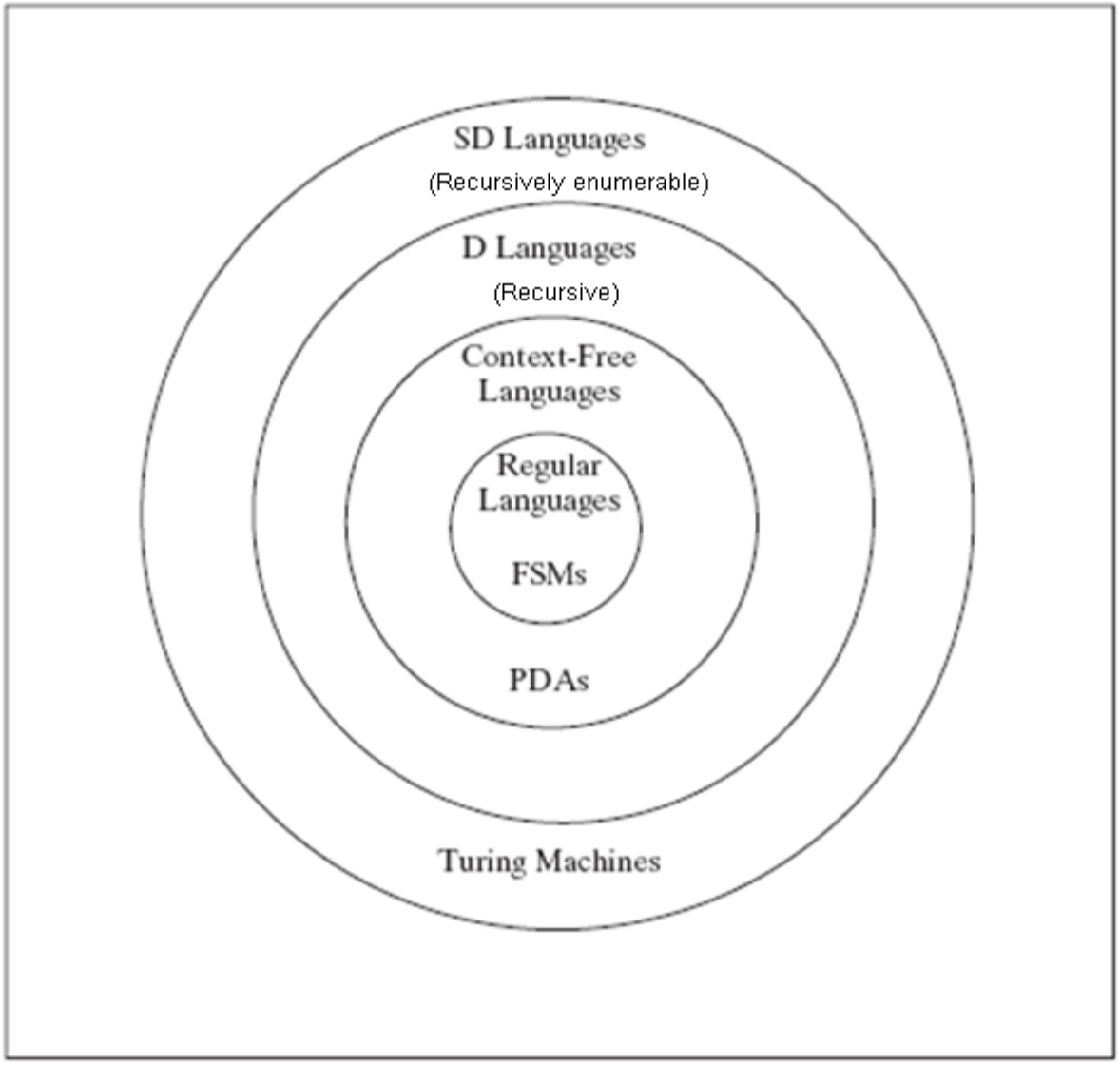


Context-Free Grammars

Chapter 11

Languages and Machines

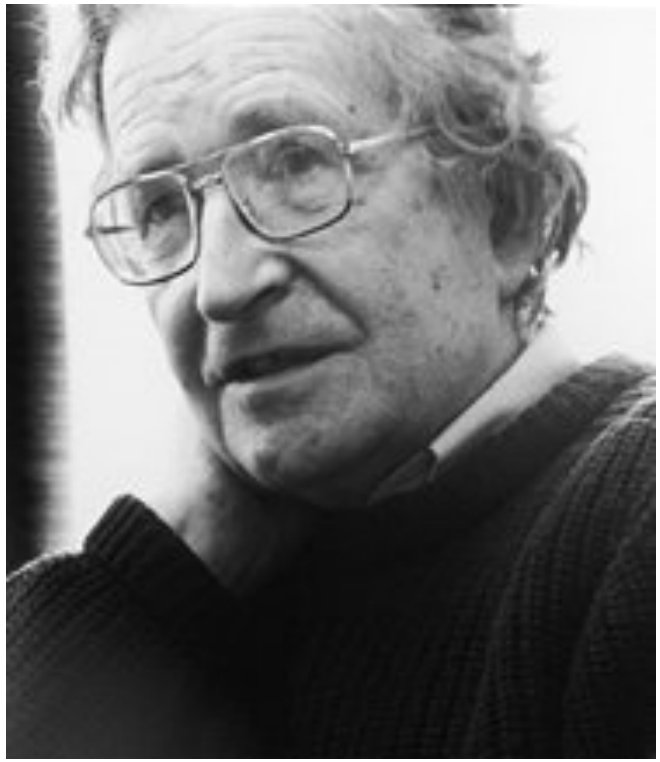


Background

• Context-free grammars play a central role in the description and design of programming languages and compilers
• They are also used for analyzing the syntax of natural languages.
• Developed by Noam Chomsky in mid 50's

•1928 –
• Professor emeritus at MIT
• Father of modern linguistics
• Still holds office

• Controversial political critic
• Often receives undercover police protection



Rewrite Systems and Grammars

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:

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

Simple-rewrite

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

1. Set *working-string* to *w*.

2. Until told by *R* to halt do:

   Match the lhs of some rule against some part of *working-string*.

   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 say that *R* can drive *s* from *w*

A Rewrite System Formalism

A rewrite system formalism specifies:

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

An Example

$w = S\mathtt{a}S$

Rules:

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

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

- What order to apply the rules?
- When to quit?

Rule Based Systems

- Expert systems
- Cognitive modeling
- Business practice modeling
- General models of computation
- **Grammars**
  - *G*
  - *L(G)*

Grammars Define Languages

A grammar has a set of rules, and works with an alphabet, that can be divided into two subsets:

• 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 ⇒ to indicate steps in a derivation.

A derivation could begin with:

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

Generating Many Strings

- Multiple rules may match.

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

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

Three choices at the next step:

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

Generating Many Strings

- One rule may match in more than one way.

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

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

Two choices at the next step:

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

When to Stop

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:

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

When to Stop

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]

When to Stop

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

Example:

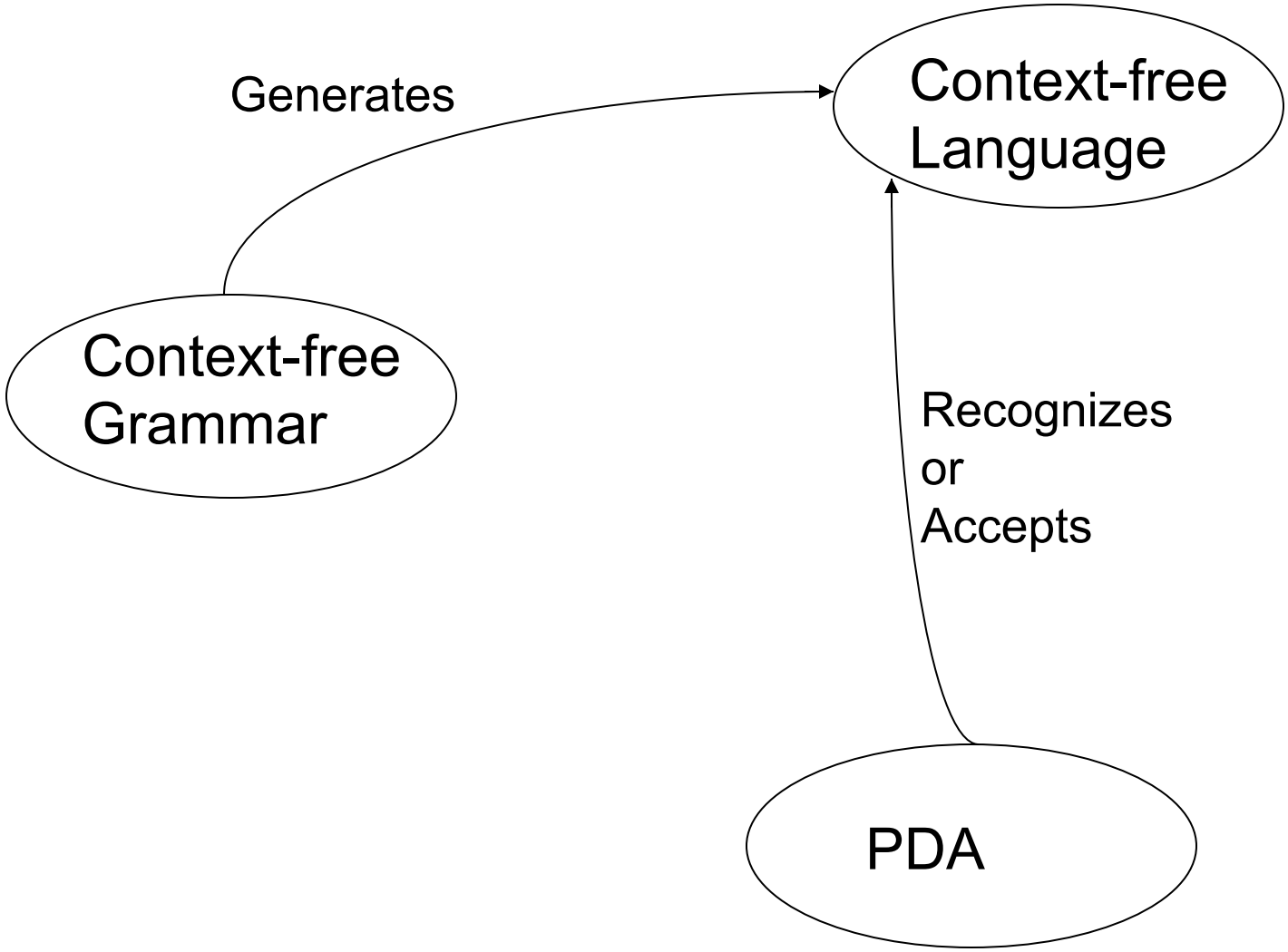
G contains only the rules $S \rightarrow B\mathtt{a}$ and $B \rightarrow \mathtt{b}B$, with *S* as the start symbol.

Then all derivations proceed as:

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

So the grammar generates the language ϕ

Context-free Grammars, Languages, and PDAs



Recall Regular Grammar

- Have a left-hand side that is a single nonterminal
- Have a right-hand side that is ε or a single terminal or a single terminal followed by a single nonterminal

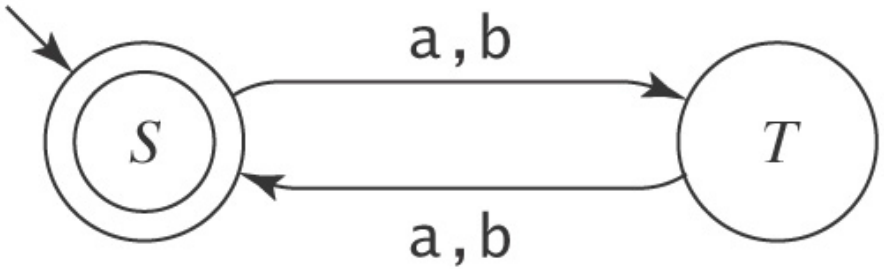
- Regular grammars must always produce strings one character at a time, moving left to right.

$L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$ $((aa) \cup (ab) \cup (ba) \cup (bb))^*$

G:

$S \rightarrow \varepsilon$
$S \rightarrow aT$
$S \rightarrow bT$
$T \rightarrow aS$
$T \rightarrow bS$

M:



Context-Free Grammars

No restrictions on the form of the right hand sides.

$S \rightarrow \texttt{ab}D\texttt{e}FG\texttt{ab}$

But require single non-terminal on left hand side as in regular grammars.

$S \rightarrow$

but not $ASB \rightarrow$

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:

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

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:

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

An Example Derivation

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

A language *L* is ***context-free*** iff it is generated by some context-free grammar *G*

- Why "context-free"?
  - Using these rules, the decision to replace a nonterminal by some other sequence is made without looking at the context in which the nonterminal occurs. Note by definition, lhs is a single nonterminal
  - There are less restrictive grammar formalisms (context-sensitive, unrestricted), where the lhs may contain several symbols
  - Context-sensitive grammar example: aSa -> aTa, where S can be replaced by T when it is surrounded by a's. Note that context is considered.
  - Unrestricted grammar is even less restrictive
- Context-free grammar = LBA = context sensitive language
- Unrestricted grammar = TM = SD
- Every regular language is also context-free

Balanced Parentheses

- Showed in Example 8.10 (p173) that Bal is not regular.
- Can we use regular grammar to define programming languages?

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

Some example derivations in *G*:

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

So,
$S \Rightarrow^*$ () and $S \Rightarrow^*$ (()())

A^nB^n

Showed in Example 8.8 (p171) that A^nB^n is not regular.

$S \rightarrow \varepsilon$
$S \rightarrow aSb$

Recursive and Self-Embedding 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 in V^*
- A grammar is recursive iff it contains at least one recursive rule.
- Recursive rules make it possible for a finite grammar to generate an infinite set of strings
- Examples: $S \rightarrow (S)$ $S \rightarrow \mathtt{a}S$

- 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^+
- It allows $X \Rightarrow^* w' X w''$ where neither w' nor w'' is ε
- A grammar is self-embedding iff it contains at least one self-embedding rule.
- Example: $S \rightarrow (S)$

Where Context-Free Grammars Get Their Power

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

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

Even length palindromes

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

$$R = \{ S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow \varepsilon \}.$$

BNF

Backus Naur Form: a notation for writing practical context-free grammars

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

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

- 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>;
```

```
{       while(x < 12) {
                hippo.pretend(x);
                x = x + 2;
        }}
```

Many other kinds of practical languages are also context-free. e.g., HTML

HTML

```
<ul>
    <li>Item 1, which will include a sublist</li>
        <ul>
            <li>First item in sublist</li>
            <li>Second item in sublist</li>
        </ul>
    <li>Item 2</li>
</ul>
```

A grammar:

/* Text is a sequence of elements.

HTMLtext → *Element* *HTMLtext* | ε

Element → *UL* | *LI* | … (and other kinds of elements that are allowed in the body of an HTML document)

/* The <ul> and tags must match.

UL → *HTMLtext*

/* The <li> and tags must match.

LI → *HTMLtext*

Designing Context-Free Grammars

Several simple strategies:

- Related regions must be generated in tandem.
  - otherwise, no way to enforce the necessary constraint

    A^nB^n

- For independent regions, use concatenation

    $A \rightarrow BC$

- Generate outside-in:
  - to generate

    $A \rightarrow aAb$

Concatenating Independent Sublanguages

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

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

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

$$R = \{ S \rightarrow NC$$
$$N \rightarrow aNb$$
$$N \rightarrow \varepsilon$$
$$C \rightarrow cC$$
$$C \rightarrow \varepsilon \}.$$

The Kleene star of a language

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

Examples of strings in L: ε, abab, aabbaaabbbabab

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

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

$R = \{$ $S \rightarrow MS$ // each M will generate one $\{a^n b^n : n \geq 0\}$
$S \rightarrow \varepsilon$
$M \rightarrow aMb$
$M \rightarrow \varepsilon\}$.

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:

$$R = \{\ S \rightarrow \mathtt{a}S\mathtt{b}$$
$$S \rightarrow \mathtt{b}S\mathtt{a}$$
$$S \rightarrow SS$$
$$S \rightarrow \varepsilon\ \}.$$

Another Ex.: Unequal a's and b's

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

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

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

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

$R =$

$S \rightarrow A$	/* more a's than b's
$S \rightarrow B$	/* more b's than a's
$A \rightarrow a$	/* at least one extra a generated
$A \rightarrow aA$	
$A \rightarrow aAb$	
$B \rightarrow b$	/* at least one extra b generated
$B \rightarrow Bb$	
$B \rightarrow aBb$	

Proving the Correctness of a Grammar

$A^nB^n = \{a^n b^n : n \geq 0\}$

$G = (\{S, a, b\}, \{a, b\}, R, S)$,

$$R = \{ \; S \rightarrow a \; S \; b$$
$$S \rightarrow \varepsilon \; \}$$

- Prove that *G* generates only strings in *L.*
- Prove that *G* generates all the strings in *L*.

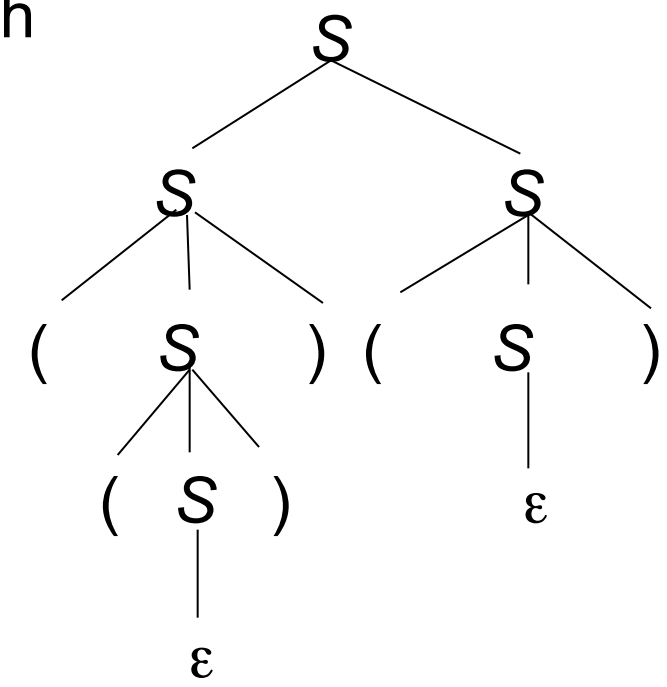
Derivations and Parse Trees

- Regular grammar: in most applications, we just want to describe the set of strings in a language.
- Context-free grammar: we also want to assign meanings to the strings in a language, for which we care about internal structure of the strings

Parse Trees

• A **parse tree** is an (ordered, rooted) tree that represents the syntactic structure of a string according to some formal grammar. In a parse tree, the interior nodes are labeled by nonterminals of the grammar, while the leaf nodes are labeled by terminals of the grammar or ε.

• A program that produces such trees is called a parser.
• Parse trees capture the essential grammatical structure of a string.



Parse Trees

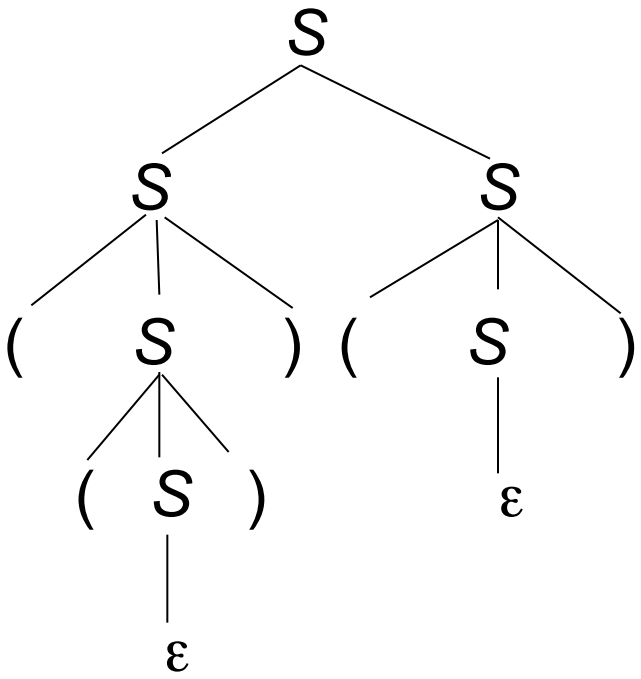
A parse tree, derived by a grammar $G = (V, \Sigma, R, S)$, is a rooted, ordered tree in which:

- 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
  $X \rightarrow x_1, x_2, \ldots, x_n$

Parse Trees

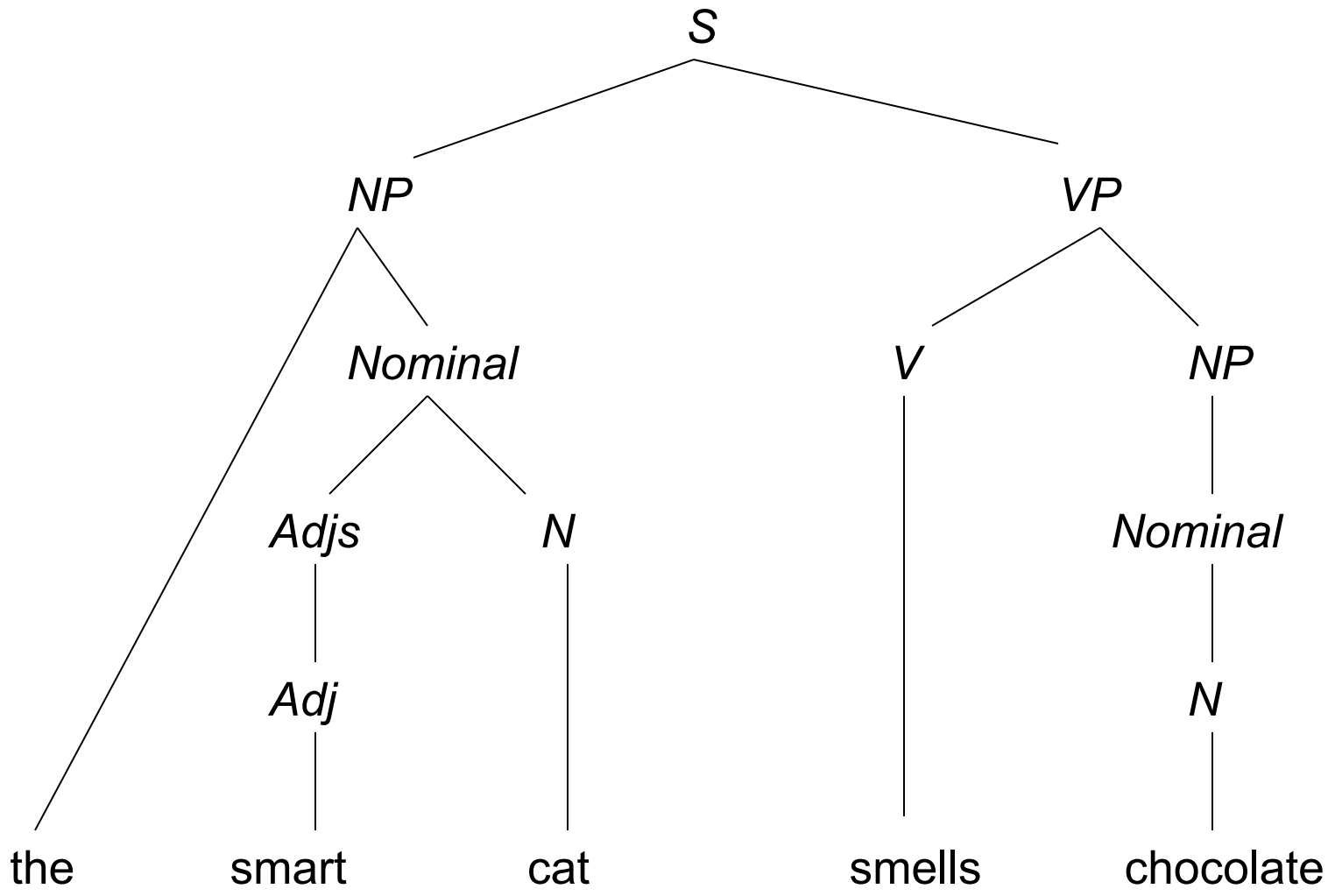
$$S \overset{1}{\Rightarrow} SS \overset{2}{\Rightarrow} (S)S \overset{3}{\Rightarrow} ((S))S \overset{4}{\Rightarrow} (())S \overset{5}{\Rightarrow} (())(S) \overset{6}{\Rightarrow} (())()$$

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



- A parse tree may correspond to multiple derivations.
- Parse trees are useful precisely because they capture the important structural facts about a derivation but throw away the details of the order in which the nonterminals were expanded.
- The order has no bearing on the structure we wish to assign to a string.

Structure in English



It is clear from the tree that the sentence is not about cat smells or smart cat smells.

Generative Capacity

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.

Which set is bigger?

A string may have multiple parse trees (leading to ambiguity)
A parse tree may correspond to multiple derivations

Another Example on Expansion Order

Look at the parse tree for

`the smart cat smells chocolate`

From the parse tree, we cannot tell which of the following is used in derivation:

$S \Rightarrow NP\ VP \Rightarrow$ the *Nominal VP* $\Rightarrow$
$S \Rightarrow NP\ VP \Rightarrow NP\ V\ NP \Rightarrow$

- Again, parse trees capture the important structural facts about a derivation but throw away the details of the nonterminal expansion order
  - The order has no bearing on the structure we wish to assign to a string.

Derivation Order

• However, the expansion order is important for algorithms.
• Algorithms for generation and recognition must be systematic.
• They typically use either the **leftmost derivation** or the **rightmost derivation.**

• A leftmost derivation is one in which, at each step, the **leftmost** nonterminal in the working string is chosen for expansion.
• A rightmost derivation is one in which, at each step, the **rightmost** nontermial in the working string is chosen for expansion.

Derivations of The Smart Cat

`the smart cat smells chocolate`

- A left-most derivation is:

  S ⇒ *NP VP* ⇒ the *Nominal VP* ⇒ the *Adjs N VP* ⇒
  the *Adj N VP* ⇒ the smart *N VP* ⇒ the smart cat *VP* ⇒
  the smart cat *V NP* ⇒ the smart cat smells *NP* ⇒
  the smart cat smells *Nominal* ⇒ the smart cat smells *N* ⇒
  the smart cat smells chocolate

- A right-most derivation is:

  S ⇒ *NP VP* ⇒ *NP V NP* ⇒ *NP V Nominal* ⇒ *NP V N* ⇒
  NP V chocolate ⇒ *NP* smells chocolate ⇒
  the *Nominal* smells chocolate ⇒
  the *Adjs N* smells chocolate ⇒
  the *Adjs* cat smells chocolate ⇒
  the *Adj* cat smells chocolate ⇒
  the smart cat smells chocolate

Ambiguity

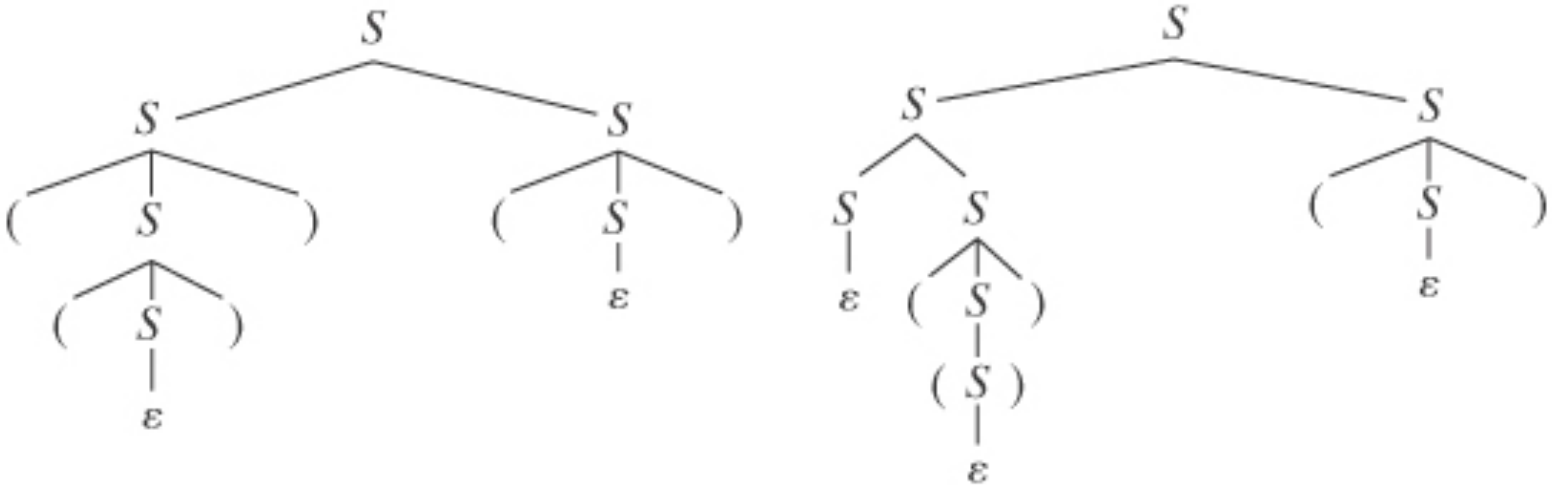
A grammar is ***ambiguous*** iff there is at least one string in $L(G)$ for which G produces more than one parse tree.

Even a very simple grammar can be highly ambiguous

$S \rightarrow \varepsilon$

$S \rightarrow SS$

$S \rightarrow (S)$



Regular expressions and grammars can be ambiguous too, but we do not care

Regular Expression

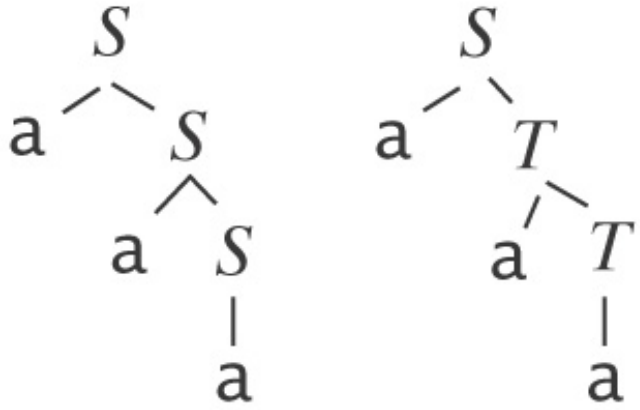
(a ∪ b)*a (a ∪ b)*

choose a from (a ∪ b)
choose a from (a ∪ b)
choose a

choose a
choose a from (a ∪ b)
choose a from (a ∪ b)

Regular Grammar

$S \rightarrow a$
$S \rightarrow bS$
$S \rightarrow aS$
$S \rightarrow aT$
$T \rightarrow a$
$T \rightarrow b$
$T \rightarrow aT$
$T \rightarrow bT$



Why Is Ambiguity a Problem?

• With regular languages, for most applications, we do not care about assigning internal structure to strings.

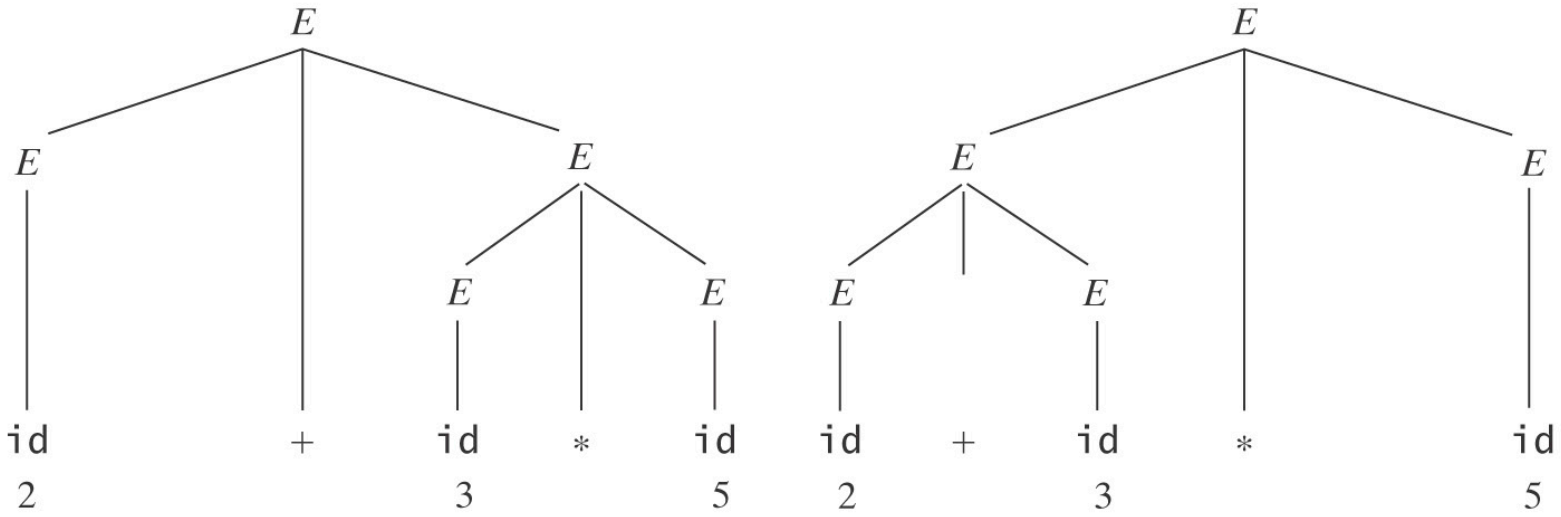
• With context-free languages, we usually do care about internal structure because, given a string *w*, we want to assign meaning to *w*.

• We almost always want to assign a unique such meaning.

• It is generally difficult, if not impossible, to assign a unique meaning without a unique parse tree.

An Ambiguous Expression Grammar

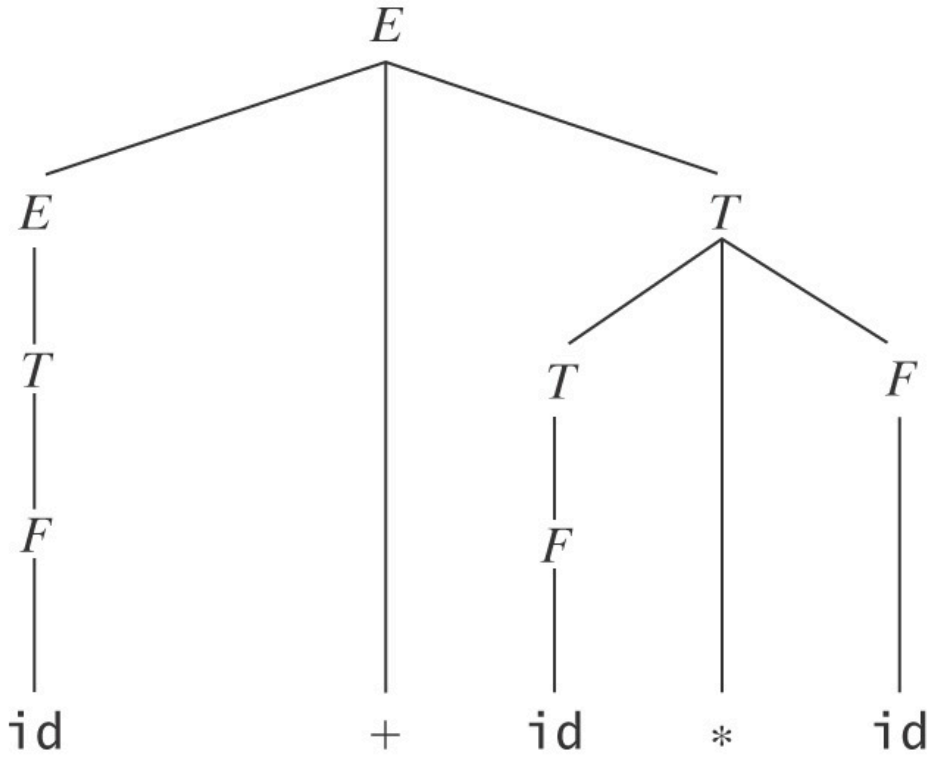
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow$ `id`



17 or 25?

Arithmetic Expressions - A Better Way

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow$ id



Inherent Ambiguity

In many cases, for an ambiguous grammar G, it is possible to construct a new grammar G' that generate L(G) with less or no ambiguity. However, not always.

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

Example:

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

Every string in *L* has either (or both) the same number of a's and b's or the same number of b's and c's.

Inherent Ambiguity

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

One grammar for L has the rules:

$S \rightarrow S_1 \mid S_2$

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

$S_2 \rightarrow aS_2 \mid B$ /* Generate all strings in $\{a^n b^m c^m\}$.
$B \rightarrow bBc \mid \varepsilon$

Consider any string of the form $a^n b^n c^n$.

- They have two distinct derivations, one through S_1 and the other through S_2
- It is possible to prove that L is inherently ambiguous: given any grammar G that generates L, there is at least on string with two derivations in G.

But We Can Often Reduce Ambiguity

We can get rid of:

- ε rules like $S \rightarrow \varepsilon$,
- rules with symmetric right-hand sides
  - A grammar is ambiguous if it is both left and right recursive.
  - Fix: remove right recursion

$$S \rightarrow SS \qquad \text{or} \qquad E \rightarrow E + E$$

- rule sets that lead to ambiguous attachment of optional postfixes.
  - dangling else problem: else goes with which if?
  - if *E* then if *E* then *S* else *S*

Proving that *G* is Unambiguous

• *G* is unambiguous iff, for all strings *w*, at every point in a leftmost or rightmost derivation of *w*, only one rule in *G* can be applied.

In other words,

• A grammar *G* is unambiguous iff every string derivable in *G* has a single leftmost (or rightmost) derivation.

Going Too Far

- Getting rid of ambiguity, but not at the expense of losing useful parse trees.
- In the arithmetic expression example and dangling else case, we were willing to force one interpretation. Sometimes, this is not acceptable.

```
Chris likes the girl with a cat.
```

```
Chris shot the bear with a rifle.
```

```
Chris shot the bear with a rifle.
```

A Testimonial

Also, you will be happy to know that I just made use of the context-free grammar skills I learned in your class! I am working on Firefox at IBM this summer and just found an inconsistency between how the native Firefox code and a plugin by Adobe parse SVG path data elements. In order to figure out which code base exhibits the correct behavior I needed to trace through the grammar http://www.w3.org/TR/SVG/paths.html#PathDataBNF. Thanks to your class I was able to determine that the bug is in the Adobe plugin. Go OpenSource!

Stochastic Context-Free Grammars

- Recall in Chapter 5, we introduced the idea of stochastic FSM: an NDFSM whose transitions have been augmented with probabilities that describe some phenomenon that we want to model.
- We can apply the same idea to context-free grammar.
- We can add probabilities to grammar rules and create a stochastic context-free grammar, also called probabilistic context-free grammar.

Stochastic Context-Free Grammars

A stochastic context-free grammar G is a quintuple:

(V, Σ, R, S, D):

- V is the rule alphabet,
- Σ is a subset of V,
- R is a finite subset of $(V - \Sigma) \times V^*$,
- S can be any element of $V - \Sigma$,
- D is a function from R to [0 - 1].

D assigns a probability to each rule in R.

D must satisfy the requirement that, for every nonterminal symbol X, the sum of the probabilities associated with all rules whose left-hand side is X must be 1.

Stochastic Context-Free Example

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

But now suppose we want to describe a special case:

- a's occur three times as often as b's do.

$G = (\{S, \mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}, R, S, D)$:

$S \rightarrow \mathtt{a}S\mathtt{a}$	[.72]
$S \rightarrow \mathtt{b}S\mathtt{b}$	[.24]
$S \rightarrow \varepsilon$	[.04]

Stochastic Context-Free Grammars

The probability of a particular parse tree *t*:

Let *C* be the collection (in which duplicates count) of rules *r* that were used to generate *t*. Then:

$$\Pr(t) = \prod_{r \in C} \Pr(r)$$

Example:

$S \rightarrow \mathtt{a}S\mathtt{a}$	[.72]
$S \rightarrow \mathtt{b}S\mathtt{b}$	[.24]
$S \rightarrow \varepsilon$	[.04]

$S \Rightarrow \mathtt{a}S\mathtt{a} \Rightarrow \mathtt{aa}S\mathtt{aa} \Rightarrow \mathtt{aab}S\mathtt{baa} \Rightarrow \mathtt{aabbaa}$

.72 .72 .24 .04 = .00497664