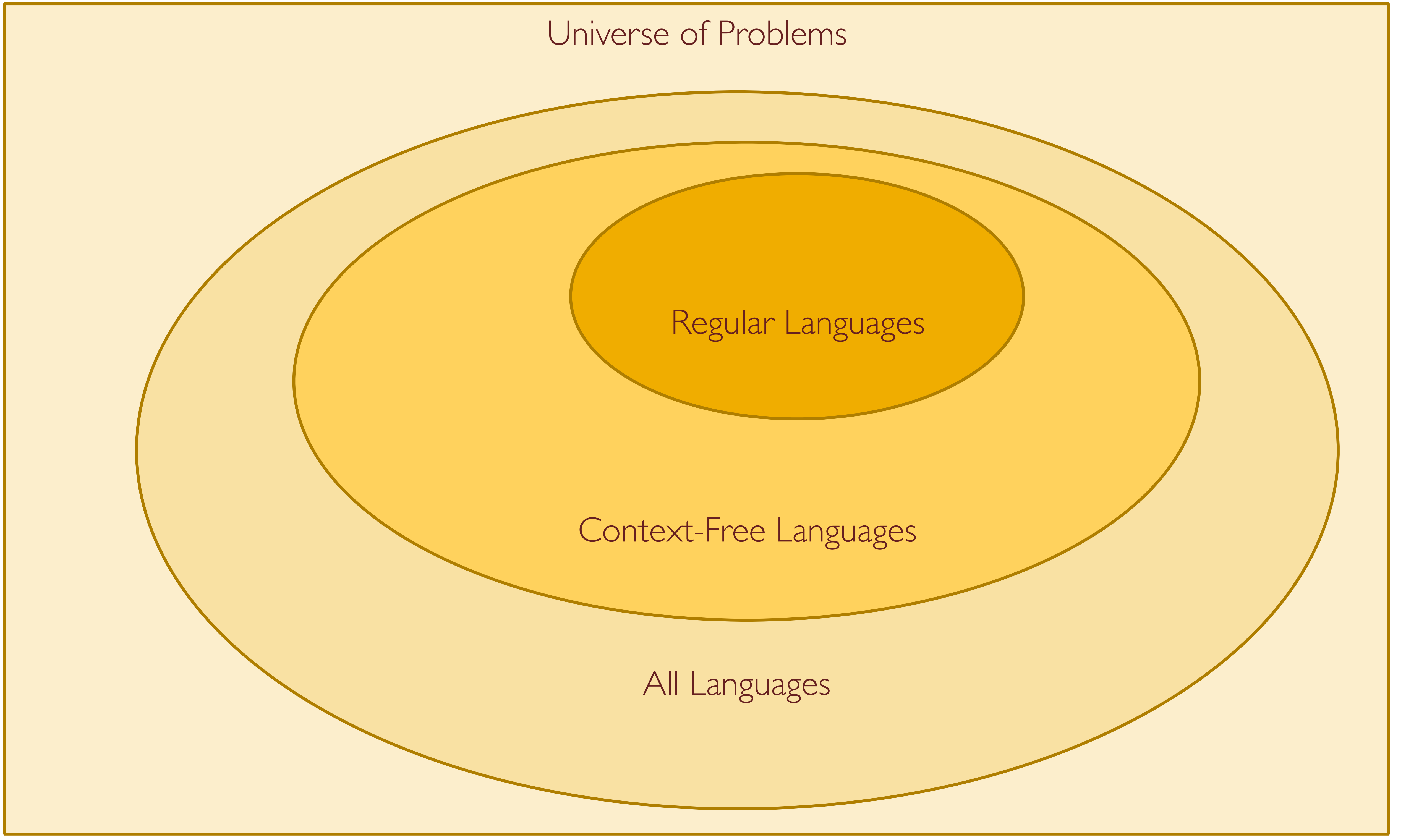
Computing Theory

COMP 147

The Space of Problems



Overview

- Context-free languages:
  - CFLs include languages that were excluded from regular languages due to bounded memory
- Context-free Grammars
  - CFGs are a notation for describing CFLs
  - Languages definable by CFG ⇔ CFLs
  - Analogous to regex for regular languages
  - Recursive definitions

Overview

- Pushdown Automata (PDA)
  - Automata that also have a stack
  - Stacks provide unbounded memory
  - Languages recognized by PDA = CFLs
- Pumping Lemma for CFLs
  - How do we show a language is not context-free?

General Grammars

- Grammars define the syntax (structure) of a language
    - Usually defined by rules that can generate legal strings (sentences) in the language
    - The set of strings that can be generated by the rules of the grammar is the language of the grammar

Context-Free Grammars

- A context-free grammar is a notation for describing languages.
- It is more powerful than finite automata or RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.

Context-Free Grammars

- Basic idea is to use “variables” to stand for sets of strings (i.e., languages).
- These variables are defined recursively, in terms of one another.
- Recursive rules a“productions”b involve only concatenation.

Example

CFG for $\{ 0^n1^n \mid n \geq 1\}$

- Productions:

S -> 01

S -> 0S1

- Basis: 01 is in the language.
- Induction: if w is in the language, then so is 0w1.

English Grammar

$$
\begin{aligned}
\langle\text{SENTENCE}\rangle &\rightarrow \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle \\
\langle\text{NOUN-PHRASE}\rangle &\rightarrow \langle\text{CMPLX-NOUN}\rangle \mid \langle\text{CMPLX-NOUN}\rangle\langle\text{PREP-PHRASE}\rangle \\
\langle\text{VERB-PHRASE}\rangle &\rightarrow \langle\text{CMPLX-VERB}\rangle \mid \langle\text{CMPLX-VERB}\rangle\langle\text{PREP-PHRASE}\rangle \\
\langle\text{PREP-PHRASE}\rangle &\rightarrow \langle\text{PREP}\rangle\langle\text{CMPLX-NOUN}\rangle \\
\langle\text{CMPLX-NOUN}\rangle &\rightarrow \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle \\
\langle\text{CMPLX-VERB}\rangle &\rightarrow \langle\text{VERB}\rangle \mid \langle\text{VERB}\rangle\langle\text{NOUN-PHRASE}\rangle \\
\langle\text{ARTICLE}\rangle &\rightarrow \texttt{a} \mid \texttt{the} \\
\langle\text{NOUN}\rangle &\rightarrow \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle\text{VERB}\rangle &\rightarrow \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle\text{PREP}\rangle &\rightarrow \texttt{with}
\end{aligned}
$$

Attempt to generate a sentence.

English Grammar

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\rightarrow \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\rightarrow \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\rightarrow \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\rightarrow \texttt{with}
\end{aligned}
$$

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a} \ \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy} \ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy} \ \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \texttt{a boy} \ \langle \text{VERB} \rangle \\
&\Rightarrow \texttt{a boy sees}
\end{aligned}
$$

Context-Free Grammars

- Terminals = symbols of the alphabet of the language being defined.
- Variables = nonterminals = a finite set of other symbols, each of which represents a language.
- Start symbol = the variable whose language is the one being defined.

Context-Free Grammars

- A production or substitution rule has the form variable head -> string of variables and terminals body.

- Convention:
  - A, B, C,… and also S are variables.
  - a, b, c,… are terminals.
  - …, X, Y, Z are either terminals or variables

Context-free Grammar

A CFG over $\Sigma = \{ \#, 0, 1 \}$:

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$

3 *substitution rules* (also called productions)

symbols left of → are *variables*

1st rule identifies the *start symbol*

symbols right of → are strings of *variables* or *terminals*

Σ = set of terminals

Generating strings from a CFG

- A CFG over $\Sigma = \{ \#, 0, 1 \}$:

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

- Start with start symbol
- Loop:
  - Replace any variable with the RHS of a rule for that variable

Attempt to generate some strings.

Derivations

- We derive strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the body of one of its productions.

    - That is, the “productions for A” are those that have head A.

Derivations

- We say $\alpha A\beta \Rightarrow \alpha\gamma\beta$ if $A \to \gamma$ is a production.

- Example:
  S -> 01;
  S -> 0S1.

- S => 0S1 => 00S11 => 000111.

Language of a Grammar

- If G is a CFG, then L(G), the language of G, is {w | S =>* w}.

- Example:
  G has productions S -> **ε** and S -> 0S1.

- $L(G) = \{0^n 1^n \mid n \geq 0\}$.

Derivations

S->ab | aSb | SS

Then, which of the following strings is **not** in the language defined by this CFG?

a) aababbab

b) aaabbabaabbb

c) aaabaabbab

d) abababab

Derivations

- Consider the following Grammar: S->aSb | ab | SS
- Derivation for aabbab

Example: Leftmost Derivations

- Leftmost derivation: forcing the leftmost variable
- Balanced-parentheses grammar:

  S -> SS | aSb | ab

- $S \Rightarrow_{lm} SS \Rightarrow_{lm} aSbS \Rightarrow_{lm} aabbS \Rightarrow_{lm} aabbab$
- Thus, $S \Rightarrow^*_{lm} aabbab$
- S => SS => Sab => aSbab => aabbab is a derivation, but not a leftmost derivation.

Example: Rightmost Derivations

- Balanced-parentheses grammar:
  $S \text{ -> } SS \mid aSb \mid ab$

- $S \Rightarrow_{rm} SS \Rightarrow_{rm} Sab \Rightarrow_{rm} aSbab \Rightarrow_{rm} aabbab$

- Thus, $S \Rightarrow^*_{rm} aabbab$

- $S \Rightarrow SS \Rightarrow SSS \Rightarrow SabS \Rightarrow ababS \Rightarrow ababab$ is neither a rightmost nor a leftmost derivation.

Which of the following is a rightmost derivation of the grammar S -> SS | aSb | ab?

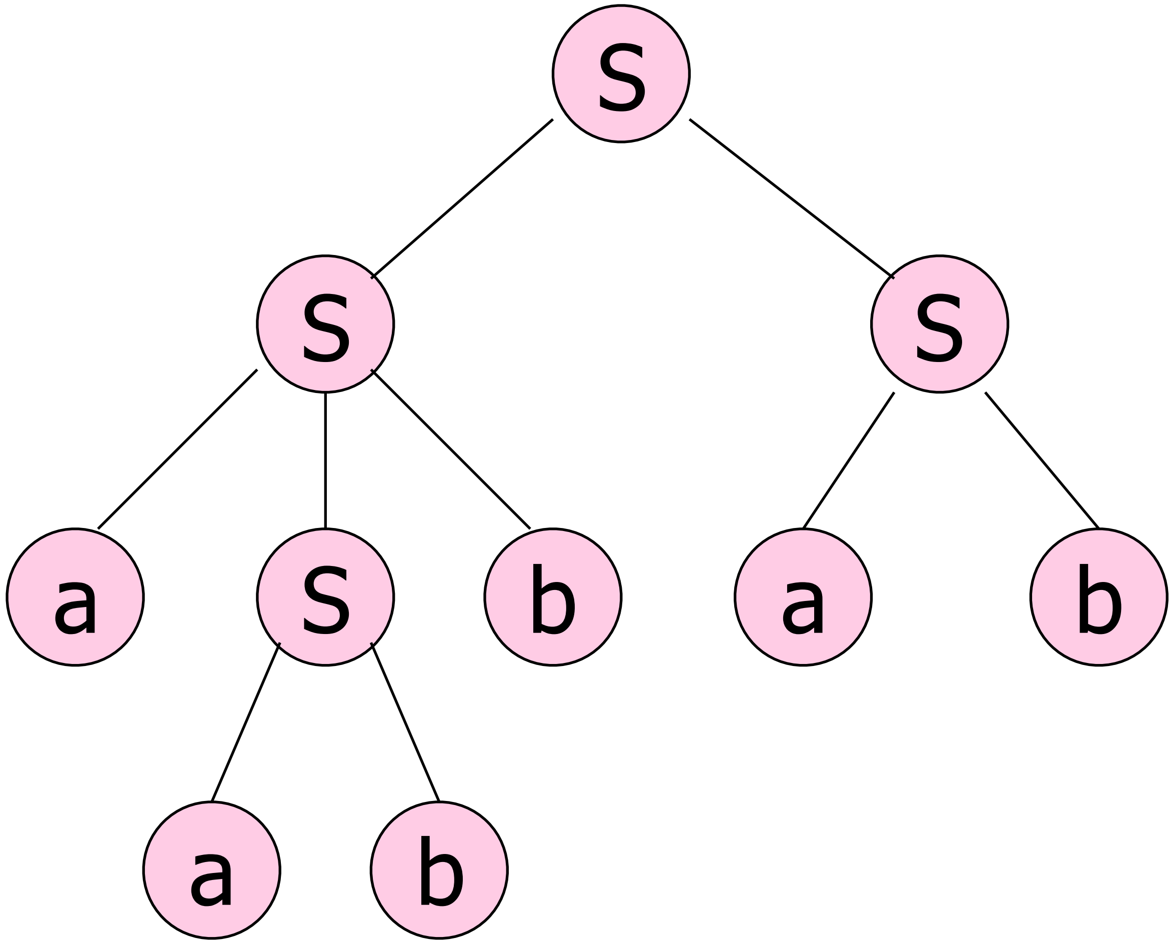
1. S=>SS=>SSS=>abSS=>abaSbS=>abaabbS=>abaabbab
2. S=>SS=>SSS=>SaSbS=>SaabbS=>Saabbab=>abaabbab
3. S=>SS=>SSS=>SSab=>SaSbab=>Saabbab=>abaabbab
4. S=>SS=>abS=>abSS=>abaSbS=>abaabbS=>abaabbab

Parse Trees

- Parse trees are trees labeled by symbols of a particular CFG.
- Leaves: labeled by a terminal or **ε**.
- Interior nodes: labeled by a variable.
  - Children are labeled by the body of a production for the parent.
- Root: must be labeled by the start symbol.

Example: Parse Tree

S -> SS | aSb | ab



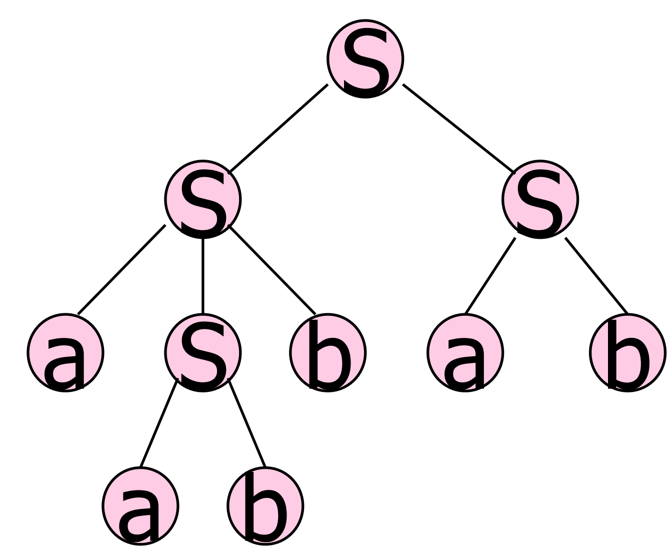
Parse Trees

Which of the following cannot appear as the label of a node a leaf or interior node b of a parse tree?

1. Variable S
2. Terminal a
3. Terminal b
4. Terminal String ab

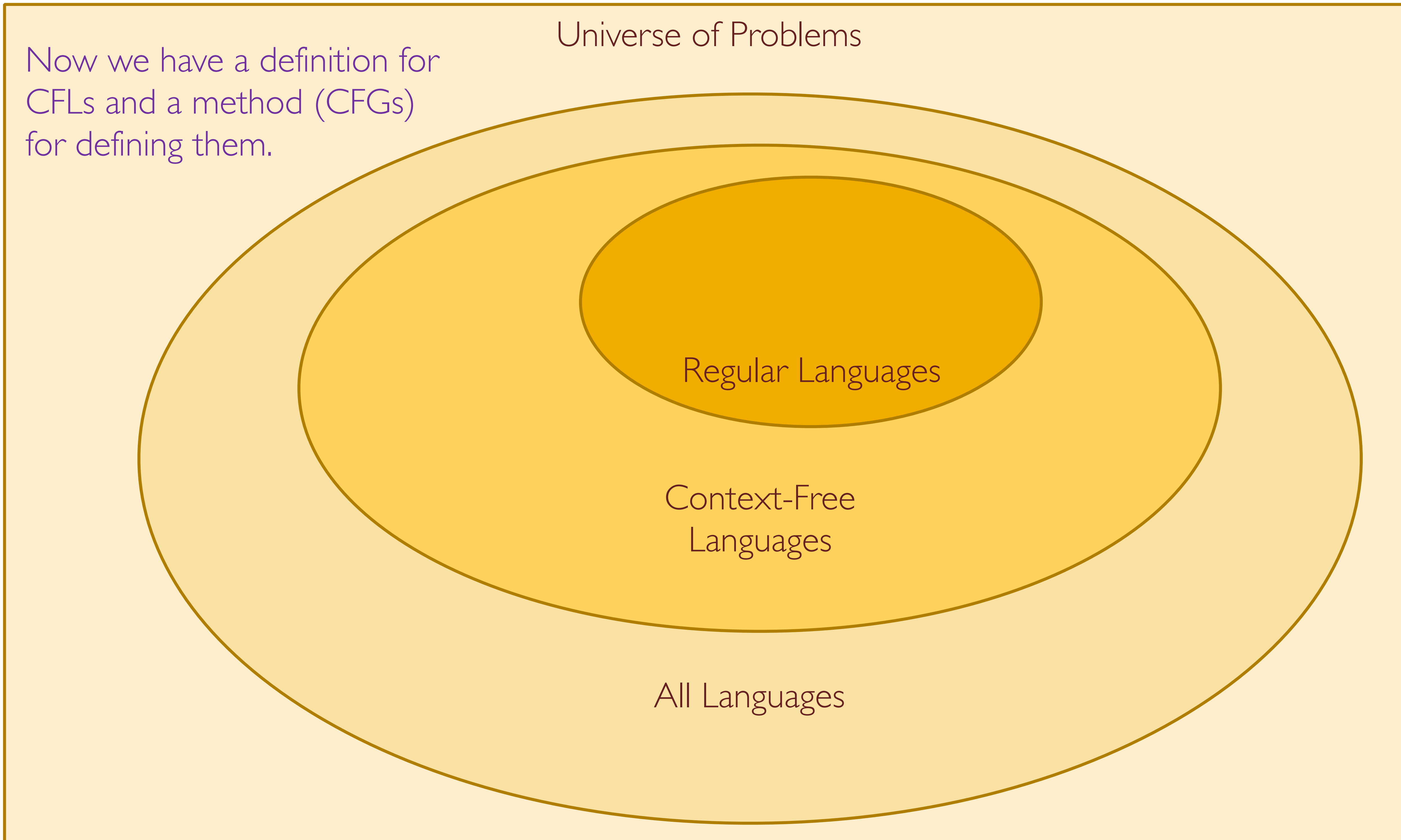
Yield of a Parse Tree

- The concatenation of the labels of the leaves in left-to-right order
  - That is, in the order of a preorder traversal.
- is called the yield of the parse tree.
- Example: yield of is aabbab



The Space of Problems

Now we have a definition for CFLs and a method (CFGs) for defining them.



CFG: Formal Definition

DEFINITION 2.2

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

Last time

- Consider the following Grammar: S -> 0S1| 01.
- Give derivations and Parse Trees for string 0011

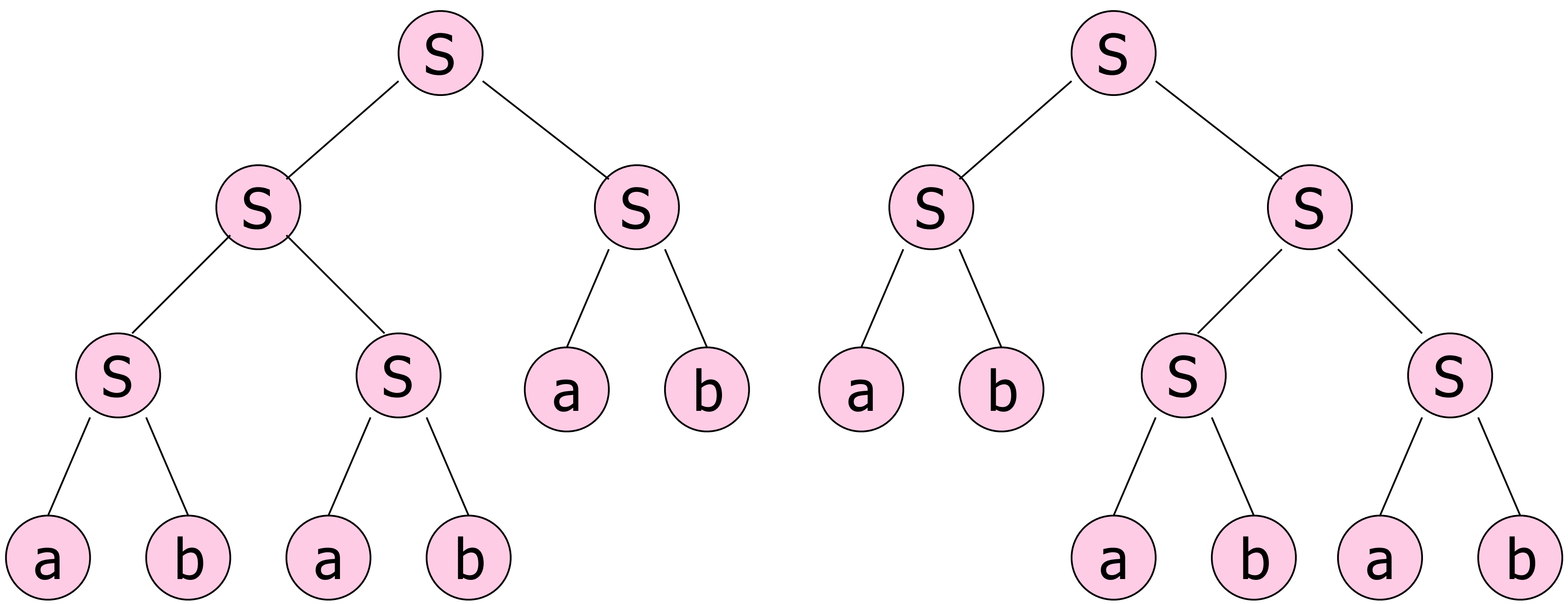
Ambiguous Grammars

- A CFG is ambiguous if there is a string in the language that is the yield of two or more parse trees (or two different leftmost (or rightmost) derivations)

- Example: S -> SS | aSb | ab

- Two parse trees for ababab on next slide.

Ambiguous Grammars



Example

- S -> S + S | S * S | 1 | 2 | 3 | 4
- Can you give 2 parse trees for 2 + 3 * 4?

Ambiguity is a Property of Grammars, not Languages

- For the balanced-parentheses language, here is another CFG, which is unambiguous.

- B -> aRB | ε

B, the start symbol, derives balanced strings.

- R -> b | aRR

R generates certain strings that have one more right paren than left.

Example: Unambiguous Grammar

- B -> aRB | **ε**   R -> b | aRR
- Construct a unique leftmost derivation for a given balanced string of parentheses by scanning the string from left to right.
  - If we need to expand B, then use B -> aRB if the next symbol is “a”; use **ε** if at the end.
  - If we need to expand R, use R -> b if the next symbol is “b” and aRR if it is “a”.

The Parsing Process

Remaining Input:

aabbab

Next symbol

Steps of leftmost derivation:

B

B -> aRB | ε

R -> b | aRR

The Parsing Process

Remaining Input:

abbab

Next symbol

Steps of leftmost derivation:

B

aRB

B -> aRB | ε

R -> b | aRR

The Parsing Process

Remaining Input:

bbab

Next symbol

Steps of leftmost derivation:

B

aRB

aaRRB

B -> aRB | ε

R -> b | aRR

The Parsing Process

Remaining Input:

bab

Next symbol

Steps of leftmost derivation:

B

aRB

aaRRB

aabRB

B -> aRB | ε

R -> b | aRR

The Parsing Process

Remaining Input:

ab

Next symbol

Steps of leftmost derivation:

B

aRB

aaRRB

aabRB

aabbB

B -> aRB | ε

R -> b | aRR

The Parsing Process

Remaining Input:

b

Next symbol

Steps of leftmost derivation:

B   aabbaRB

aRB

aaRRB

aabRB

aabbB

B -> aRB | ε   R -> b | aRR

The Parsing Process

Remaining Input:

Next symbol

Steps of leftmost derivation:

B

aRB

aaRRB

aabRB

aabbB

aabbaRB

aabbabB

B -> aRB | ε

R -> b | aRR

The Parsing Process

Remaining Input:

Next symbol

Steps of leftmost derivation:

B
aRB
aaRRB
aabRB
aabbB
aabbaRB
aabbabB
aabbab

B -> aRB | ε   R -> b | aRR

LL(1) Grammars

- As an aside, a grammar such
  B -> aRB | ε R -> b | aRR,

- where you can always figure out the production to use in a leftmost derivation by scanning the given string left-to-right and looking only at the next one symbol is called LL(1).
  - "Leftmost derivation, left-to-right scan, one symbol of lookahead."

LL(1) Grammars – (2)

- Most programming languages have LL(1) grammars.

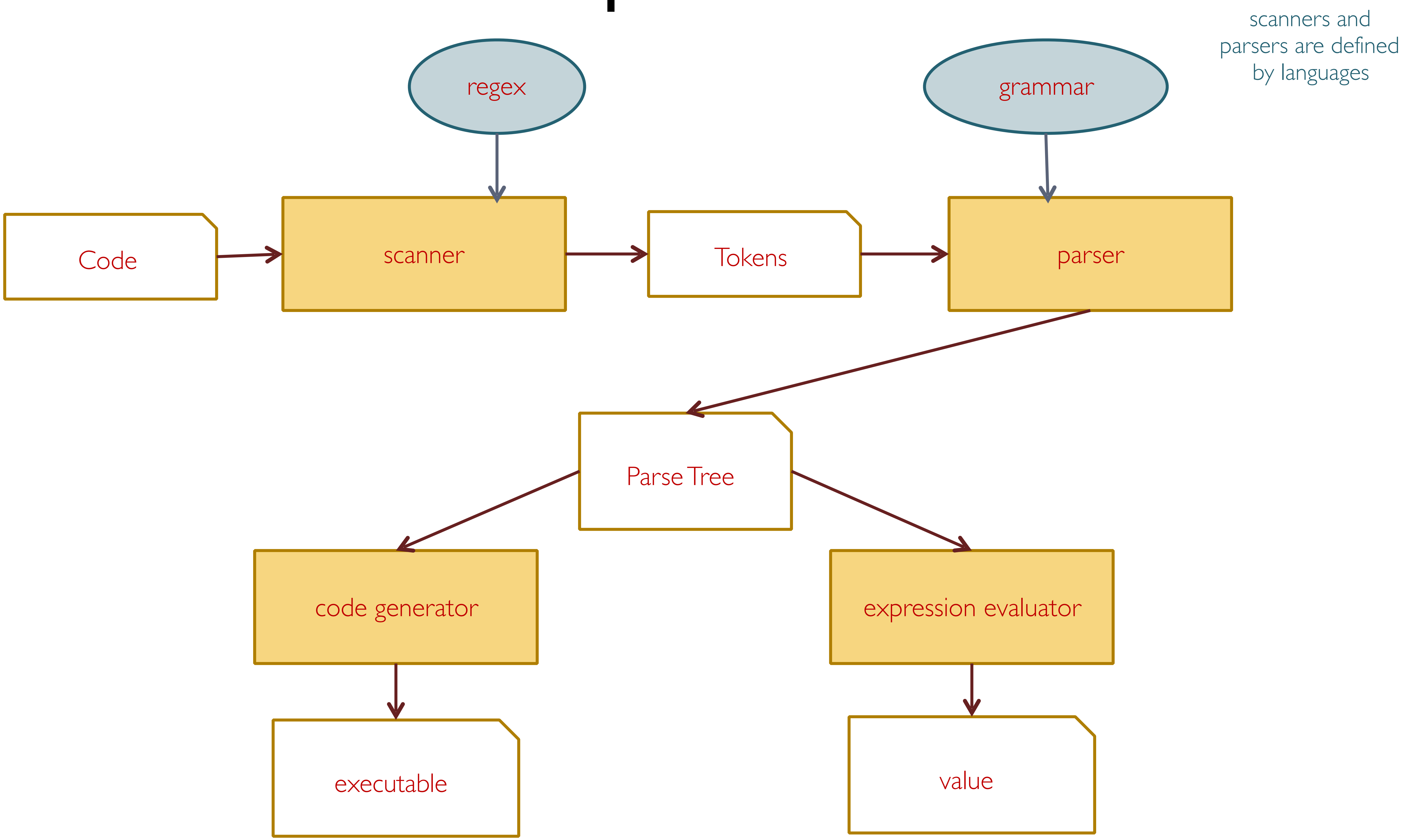
- LL(1) grammars are never ambiguous.

Inherent Ambiguity

- It would be nice if for every ambiguous grammar, there were some way to “fix” the ambiguity, as we did for the balanced-parentheses grammar.

- Unfortunately, certain CFL’s are inherently ambiguous, meaning that every grammar for the language is ambiguous.

Compilers


regex
grammar
scanners and
parsers are defined
by languages
Code
scanner
Tokens
parser
Parse Tree
code generator
expression evaluator
executable
value