CSC 3130: Automata theory and formal languages

Context-free languages

Context-free grammar

- This is an a different model for describing languages
- The language is specified by productions (substitution rules) that tell how strings can be obtained, e.g.

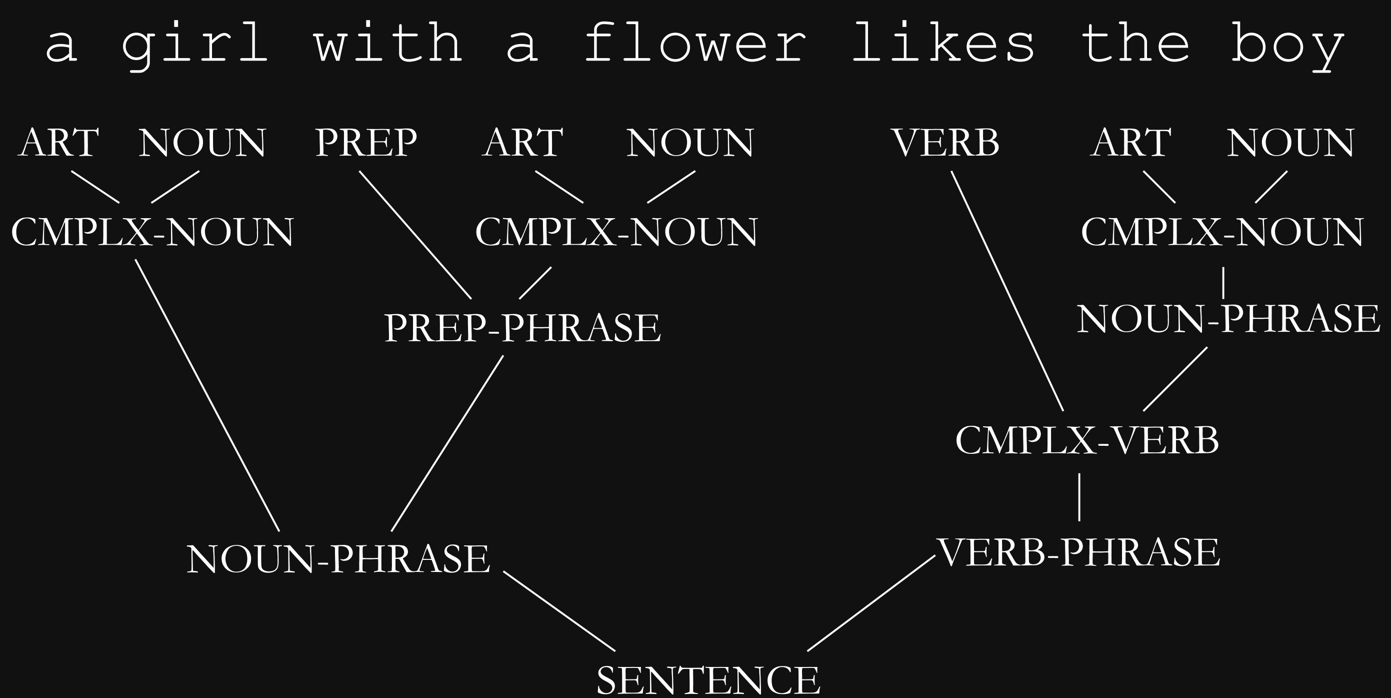
$$A \to 0A1 \qquad A, B \text{ are variables}$$
$$A \to B \qquad 0, 1, \# \text{ are terminals}$$
$$B \to \# \qquad A \text{ is the start variable}$$

- Using these rules, we can derive strings like this:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Some natural examples

- Context-free grammars were first used for natural languages



Natural languages

- We can describe (some fragments) of the English language by a context-free grammar:

SENTENCE → NOUN-PHRASE VERB-PHRASE
NOUN-PHRASE → CMPLX-NOUN
NOUN-PHRASE → CMPLX-NOUN PREP-PHRASE
VERB-PHRASE → CMPLX-VERB
VERB-PHRASE → CMPLX-VERB PREP-PHRASE
PREP-PHRASE → PREP CMPLX-NOUN
CMPLX-NOUN → ARTICLE NOUN
CMPLX-VERB → VERB NOUN-PHRASE
CMPLX-VERB → VERB

ARTICLE → a
ARTICLE → the
NOUN → boy
NOUN → girl
NOUN → flower
VERB → likes
VERB → touches
VERB → sees
PREP → with

variables: SENTENCE, NOUN-PHRASE, …

terminals: a, the, boy, girl, flower, likes, touches, sees, with

start variable: SENTENCE

Programming languages

- Context-free grammars are also used to describe (parts of) programming languages
- For instance, expressions like $(2 + 3) * 5$ or $3 + 8 + 2 * 7$ can be described by the CFG

<expr> → <expr> + <expr>

<expr> → <expr> * <expr>

<expr> → (<expr>)

<expr> → 0

<expr> → 1

…

<expr> → 9

Variables: <expr>

Terminals: +, *, (,), 0, 1, …, 9

Motivation for studying CFGs

- Context-free grammars are essential for understanding the meaning of computer programs

code: $(2 + 3) * 5$

meaning: "add 2 and 3, and then multiply by 5"

- They are used in compilers

Definition of context-free grammar

- A context-free grammar (CFG) is a 4-tuple (V, T, P, S) where
  - V is a finite set of variables or non-terminals
  - T is a finite set of terminals ($V \cap T = \varnothing$)
  - P is a set of productions or substitution rules of the form

    $$\mathrm{A} \longrightarrow \alpha$$

    where A is a symbol in V and α is a string over $V \cup T$
  - S is a variable in V called the start variable

Shorthand notation for productions

- When we have multiple productions with the same variable on the left like

$E \to E + E$
$E \to E * E$
$E \to (E)$
$E \to N$

$N \to 0N$
$N \to 1N$
$N \to 0$
$N \to 1$

Variables: E, N
Terminals: $+, *, (,), 0, 1$
Start variable: E

we can write this in shorthand as

$E \to E + E \mid E * E \mid (E) \mid 0 \mid 1$
$N \to 0N \mid 1N \mid 0 \mid 1$

Derivation

- A derivation is a sequential application of productions:

$$
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow (E) * E \\
&\Rightarrow (E) * N \\
&\Rightarrow (E + E) * N \\
&\Rightarrow (E + E) * 1 \\
&\Rightarrow (E + N) * 1 \\
&\Rightarrow (N + N) * 1 \\
&\Rightarrow (N + 1N) * 1 \\
&\Rightarrow (N + 10) * 1 \\
&\Rightarrow (1 + 10) * 1
\end{aligned}
$$

derivation

$$\alpha \Rightarrow \beta$$

means β can be obtained from α with one production

$$\alpha \overset{*}{\Rightarrow} \beta$$

means β can be obtained from α after zero or more productions

Language of a CFG

- The language of a CFG (V, T, P, S) is the set of all strings containing only terminals that can be derived from the start variable S

$$L = \{\omega \mid \omega \in T^* \text{ and } S \overset{*}{\Rightarrow} \omega \}$$

- This is a language over the alphabet T

- A language L is context-free if it is the language of some CFG

Example 1

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

variables: A, B
terminals: 0, 1, $\#$
start variable: A

- Is the string $00\#11$ in L?
- How about $00\#111$, $00\#0\#1\#11$?

- What is the language of this CFG?

$$L = \{0^n\#1^n : n \geq 0\}$$

Example 2

$$S \to SS \mid (S) \mid \varepsilon$$

convention: variables in uppercase, terminals in lowercase, start variable first

- Give derivations of (), (()())

S	$\Rightarrow (S)$	(rule 2)
	$\Rightarrow ()$	(rule 3)

S	$\Rightarrow (S)$	(rule 2)
	$\Rightarrow (SS)$	(rule 1)
	$\Rightarrow ((S)S)$	(rule 2)
	$\Rightarrow ((S)(S))$	(rule 2)
	$\Rightarrow (()(S))$	(rule 3)
	$\Rightarrow (()())$	(rule 3)

- How about ())?

Examples: Designing CFGs

- Write a CFG for the following languages
  - Linear equations over x, y, z, like:
    $\mathrm{x} + 5\mathrm{y} - \mathrm{z} = 9$
    $11\mathrm{x} - \mathrm{y} = 2$
  - Numbers without leading zeros, e.g., 109, 0 but not 019
  - The language $L = \{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^m\mathrm{d}^m \mid n \geq 0, m \geq 0\}$
  - The language $L = \{\mathrm{a}^n\mathrm{b}^m\mathrm{c}^m\mathrm{d}^n \mid n \geq 0, m \geq 0\}$

Context-free versus regular

- Write a CFG for the language $(0 + 1)^*111$

$$S \to A111$$
$$A \to \varepsilon \mid 0A \mid 1A$$

- Can you do so for every regular language?

Every regular language is context-free

- Proof:

From regular to context-free

regular expression	➡ CFG
$\varnothing$	grammar with no rules
ε	$S \to \varepsilon$
a (alphabet symbol)	$S \to a$
$E_1 + E_2$	$S \to S_1 \mid S_2$
E_1E_2	$S \to S_1S_2$
$E_1{*}$	$S \to SS_1 \mid \varepsilon$

In all cases, S becomes the new start symbol

Context-free versus regular

- Is every context-free language regular?
- No! We already saw some examples:

$$\begin{aligned} \mathrm{A} &\rightarrow 0\mathrm{A}1 \mid \mathrm{B} \\ \mathrm{B} &\rightarrow \# \end{aligned}$$

$$L = \{0^n\#1^n \colon n \geq 0\}$$

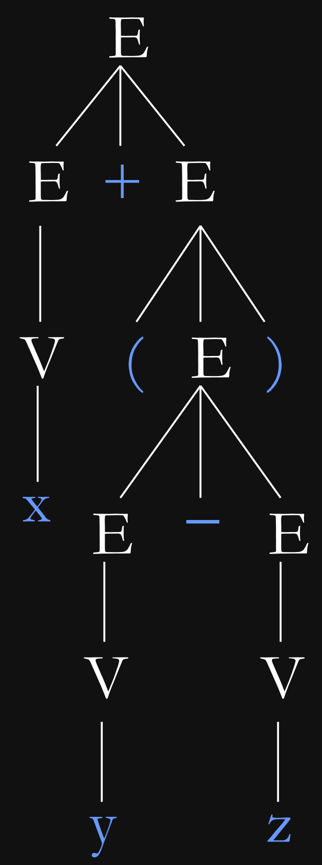
- This language is context-free but not regular

Parse tree

- Derivations can also be represented using parse trees

$$E \rightarrow E + E \mid E - E \mid (E) \mid V$$
$$V \rightarrow x \mid y \mid z$$

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow V + E \\ &\Rightarrow x + E \\ &\Rightarrow x + (E) \\ &\Rightarrow x + (E - E) \\ &\Rightarrow x + (V - E) \\ &\Rightarrow x + (y - E) \\ &\Rightarrow x + (y - V) \\ &\Rightarrow x + (y - z) \end{aligned}$$



Definition of parse tree

- A parse tree for a CFG G is an ordered tree with labels on the nodes such that
  - Every internal node is labeled by a variable
  - Every leaf is labeled by a terminal or ε
  - Leaves labeled by ε have no siblings
  - If a node is labeled A and has children $A_1, \ldots, A_k$ from left to right, then the rule

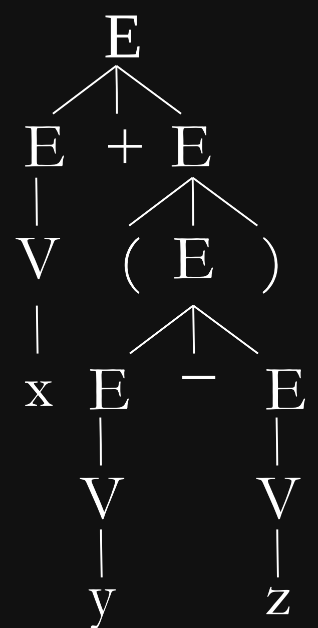
$$A \rightarrow A_1 \ldots A_k$$

    is a production in G.

Left derivation

- Always derive the leftmost variable first:

$$
\begin{aligned}
\mathrm{E} &\Rightarrow \mathrm{E} + \mathrm{E} \\
&\Rightarrow \mathrm{V} + \mathrm{E} \\
&\Rightarrow \mathrm{x} + \mathrm{E} \\
&\Rightarrow \mathrm{x} + (\mathrm{E}) \\
&\Rightarrow \mathrm{x} + (\mathrm{E} - \mathrm{E}) \\
&\Rightarrow \mathrm{x} + (\mathrm{V} - \mathrm{E}) \\
&\Rightarrow \mathrm{x} + (\mathrm{y} - \mathrm{E}) \\
&\Rightarrow \mathrm{x} + (\mathrm{y} - \mathrm{V}) \\
&\Rightarrow \mathrm{x} + (\mathrm{y} - \mathrm{z})
\end{aligned}
$$

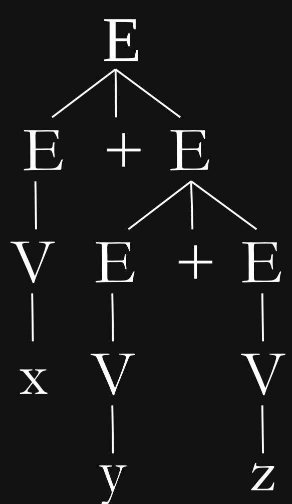


- Corresponds to a left-to-right traversal of parse tree

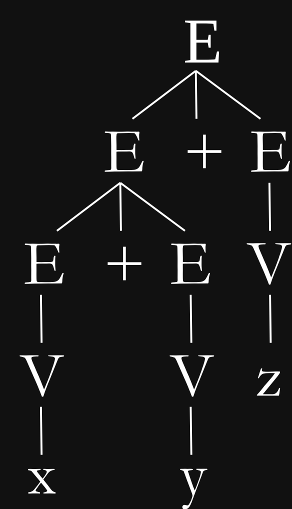
Ambiguity

- A grammar is ambiguous if some strings have more than one parse tree
- Example:

$$\boxed{\begin{array}{l} E \to E + E \mid E - E \mid (E) \mid V \\ V \to x \mid y \mid z \end{array}}$$

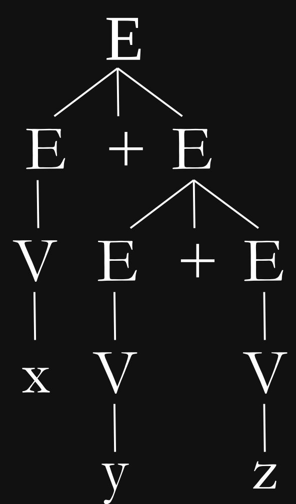


$$x + y + z$$

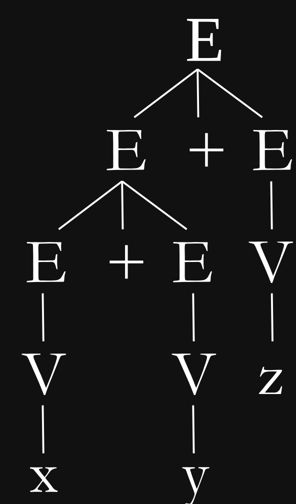


Why ambiguity matters

- The parse tree represents the intended meaning:



$x + y + z$



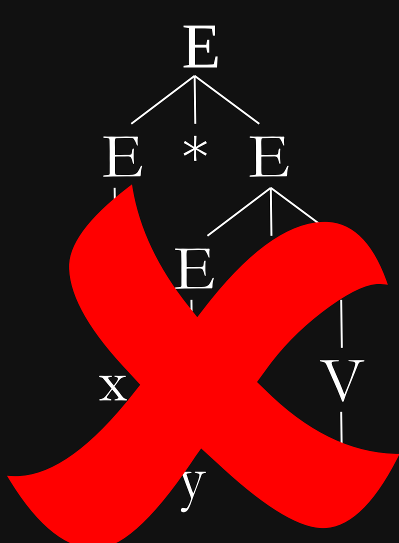
"first add y and z,
and then add this to x"

"first add x and y,
and then add z to this"

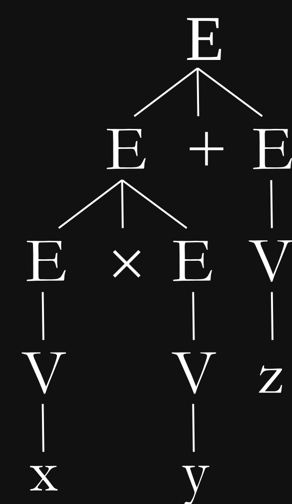
Why ambiguity matters

- Suppose we also had multiplication:

$$\begin{array}{l} \mathrm{E} \rightarrow \mathrm{E} + \mathrm{E} \mid \mathrm{E} - \mathrm{E} \mid \mathrm{E} \times \mathrm{E} \mid (\mathrm{E}) \mid \mathrm{V} \\ \mathrm{V} \rightarrow \mathrm{x} \mid \mathrm{y} \mid \mathrm{z} \end{array}$$



$\mathrm{x} \times \mathrm{y} + \mathrm{z}$



"first $\mathrm{y} + \mathrm{z}$, then $\mathrm{x} \times$"

"first $\mathrm{x} \times \mathrm{y}$, then $+ \mathrm{z}$"

Disambiguation

- Sometimes we can rewrite the grammar to remove the ambiguity

$$E \to E + E \mid E - E \mid E \times E \mid (E) \mid V$$
$$V \to x \mid y \mid z$$

- Rewrite grammar so $\times$ cannot be broken by $+$:

$$E \to T \mid E + T \mid E - T$$
$$T \to F \mid T \times F$$
$$F \to (E) \mid V$$
$$V \to x \mid y \mid z$$

T stands for term: $x * (y + z)$
F stands for factor: x, $(y + z)$

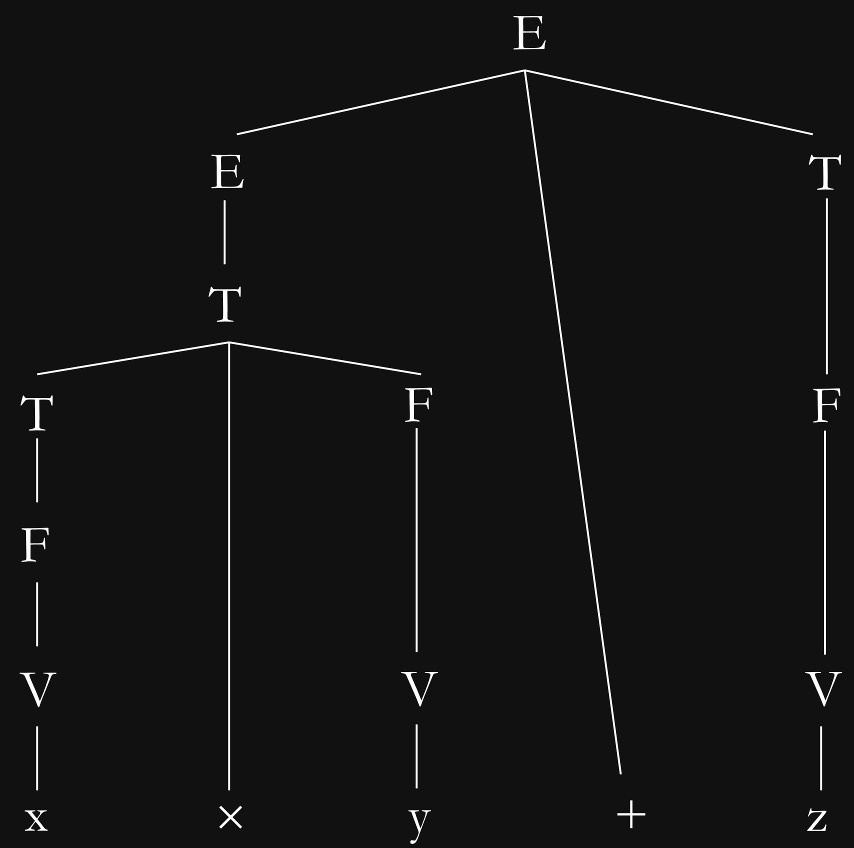
A term always splits into factors

A factor is either a variable or a parenthesized expression

Disambiguation

- Example

$$E \to T \mid E + T \mid E - T$$
$$T \to F \mid T \times F$$
$$F \to (E) \mid V$$
$$V \to x \mid y \mid z$$



Disambiguation

- Can we always disambiguate a grammar?

- No, for two reasons
  - There exists an inherently ambiguous context-free L: Every CFG for this language is ambiguous
  - There is no general procedure that can tell if a grammar is ambiguous

- However, grammars used in programming languages can typically be disambiguated

Another Example

$$S \to aB \mid bA$$
$$A \to a \mid aS \mid bAA$$
$$B \to b \mid bS \mid aBB$$

- Is ab, baba, abbbaa in L?
- How about a, bba?

- What is the language of this CFG?
- Is the CFG ambiguous?