Tianjin International Engineering Institute

Formal Languages and Automata

Lesson 9: Context-free languages

Marc Gaetano

Edition 2018

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.

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

A, B are variables
$0, 1, \#$ are terminals
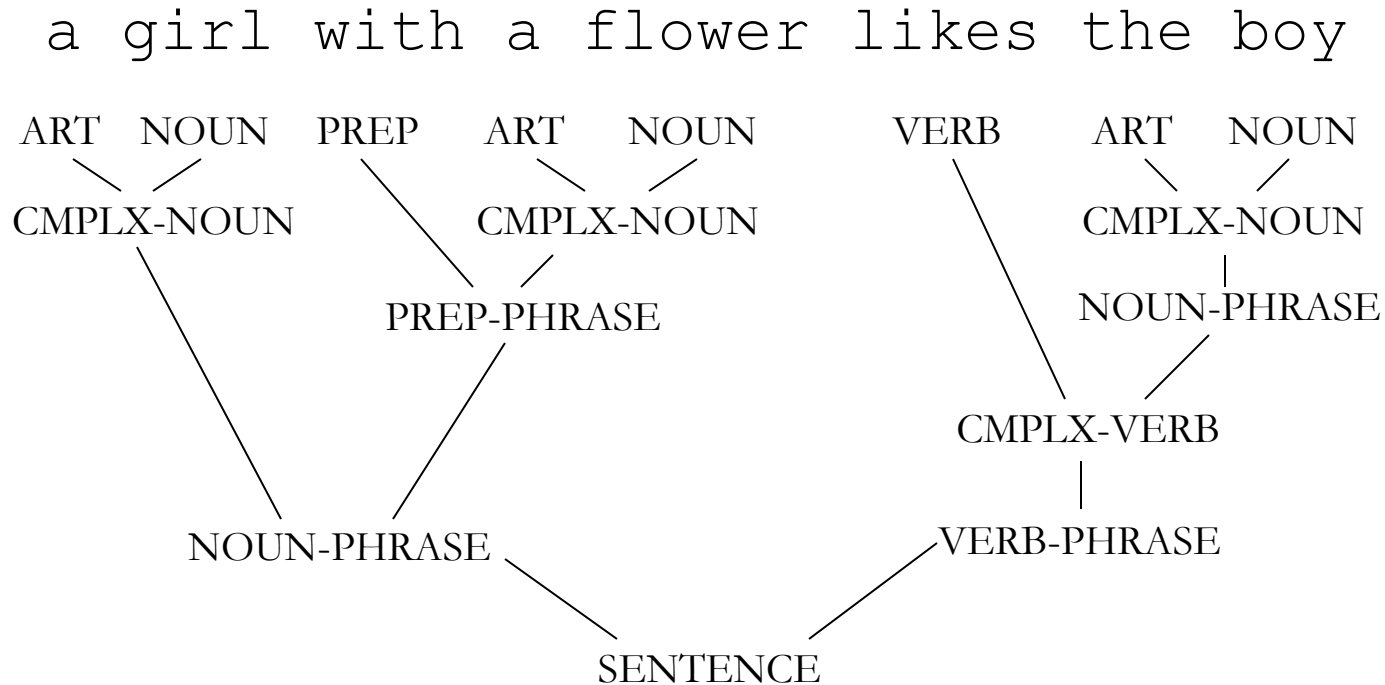A is the start variable

- 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

```
a girl with a flower likes the boy
```

ART   NOUN   PREP   ART   NOUN   VERB   ART   NOUN

CMPLX-NOUN   CMPLX-NOUN   CMPLX-NOUN

　　　　　　PREP-PHRASE   NOUN-PHRASE

　　　　　　　　　　　　　　　　　　　　　CMPLX-VERB

NOUN-PHRASE   VERB-PHRASE

　　　　　　　　SENTENCE

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

$<expr> \rightarrow <expr> + <expr>$

$<expr> \rightarrow <expr> * <expr>$

$<expr> \rightarrow (<expr>)$

$<expr> \rightarrow 0$

$<expr> \rightarrow 1$

$\ldots$

$<expr> \rightarrow 9$

Variables: $<expr>$

Terminals: $+, *, (,), 0, 1, \ldots, 9$

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
- We will study more on that in the *Compiler Design* course

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

$$A \rightarrow \alpha$$

    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

$E \rightarrow E + E$ $N \rightarrow 0N$ Variables: E, N

$E \rightarrow E * E$ $N \rightarrow 1N$ Terminals: $+$, $*$, $($, $)$, 0, 1

$E \rightarrow (E)$ $N \rightarrow 0$ Start variable: E

$E \rightarrow N$ $N \rightarrow 1$

we can write this in shorthand as

$E \rightarrow E + E \mid E * E \mid (E) \mid 0 \mid 1$

$N \rightarrow 0N \mid 1N \mid 0 \mid 1$

Derivation

- A derivation is a sequential application of productions:

$$E \Rightarrow E * E$$
$$\Rightarrow (E) * E$$
$$\Rightarrow (E) * N$$
$$\Rightarrow (E + E) * N$$
$$\Rightarrow (E + E) * 1$$
$$\Rightarrow (E + N) * 1$$
$$\Rightarrow (N + N) * 1$$
$$\Rightarrow (N + 1N) * 1$$
$$\Rightarrow (N + 10) * 1$$
$$\Rightarrow (1 + 10) * 1$$

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

$A \rightarrow 0A1 \mid B$ variables: A, B
$B \rightarrow \#$ 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

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

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:
    $x + 5y - z = 9$
    $11x - y = 2$

  - Numbers without leading zeros, e.g., 109, 0 but not 019

  - The language $L = \{a^n b^n c^m d^m \mid n \geq 0, m \geq 0\}$

  - The language $L = \{a^n b^m c^m d^n \mid n \geq 0, m \geq 0\}$

Context-free versus regular

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

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

- Can you do so for every regular language?

Every regular language is context-free

- Proof:

regular expression $\longleftrightarrow$ NFA $\longleftrightarrow$ DFA

From regular to context-free

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

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:

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

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

- This language is context-free but not regular

Parse tree

- Derivations can also be represented using parse trees

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

$$E \Rightarrow E + E$$
$$\Rightarrow V + E$$
$$\Rightarrow x + E$$
$$\Rightarrow x + (E)$$
$$\Rightarrow x + (E - E)$$
$$\Rightarrow x + (V - E)$$
$$\Rightarrow x + (y - E)$$
$$\Rightarrow x + (y - V)$$
$$\Rightarrow x + (y - z)$$

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:

$$E \Rightarrow E + E$$
$$\Rightarrow V + E$$
$$\Rightarrow x + E$$
$$\Rightarrow x + (E)$$
$$\Rightarrow x + (E - E)$$
$$\Rightarrow x + (V - E)$$
$$\Rightarrow x + (y - E)$$
$$\Rightarrow x + (y - V)$$
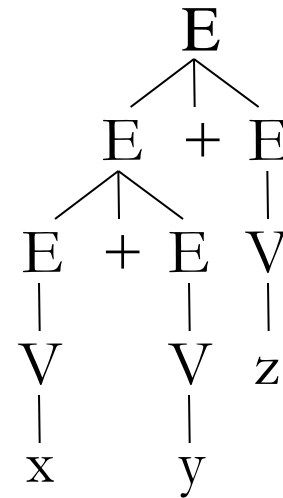$$\Rightarrow x + (y - z)$$
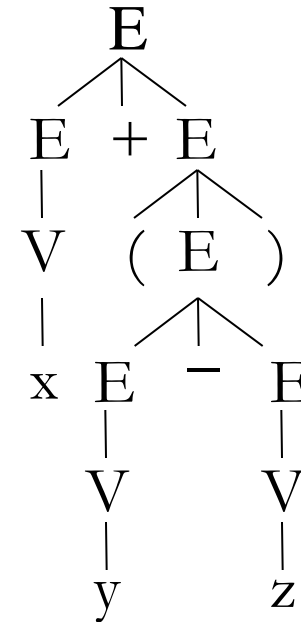
- Corresponds to a left-to-right traversal of parse tree

Ambiguity

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

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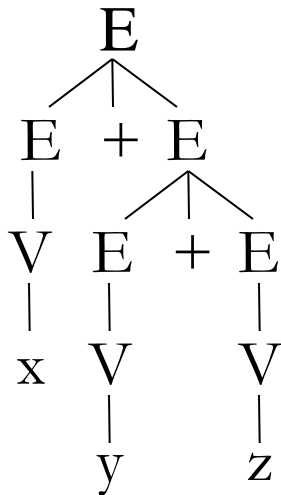
$$E \rightarrow E + E \mid E - E \mid (E) \mid V$$
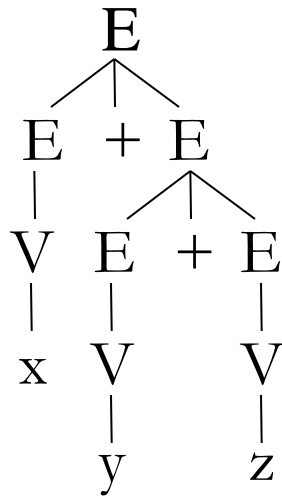$$V \rightarrow x \mid y \mid z$$

$$x + y + z$$

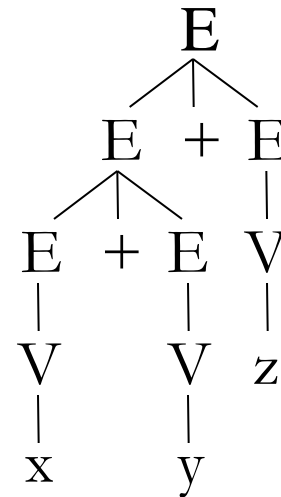# Why ambiguity matters

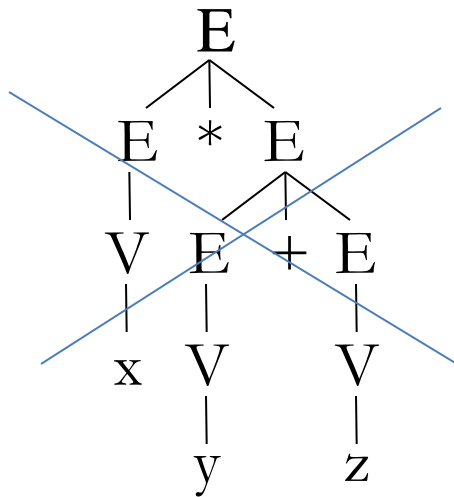- The parse tree represents the intended meaning:

$$x + y + z$$

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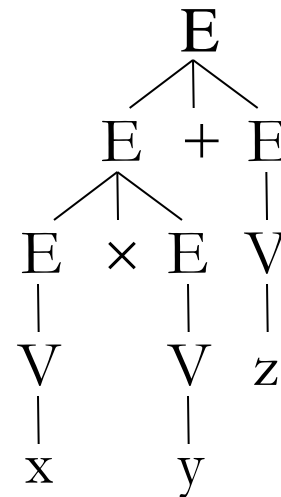
"first add x and y,
and then add z to this"

Why ambiguity matters

- Suppose we also had multiplication:

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

$$x \times y + z$$

"first $y + z$, then $x \times$"   "first $x \times y$, then $+ z$"

Disambiguation

- Sometimes we can rewrite the grammar to remove the ambiguity

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

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

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

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

A term always splits into factors

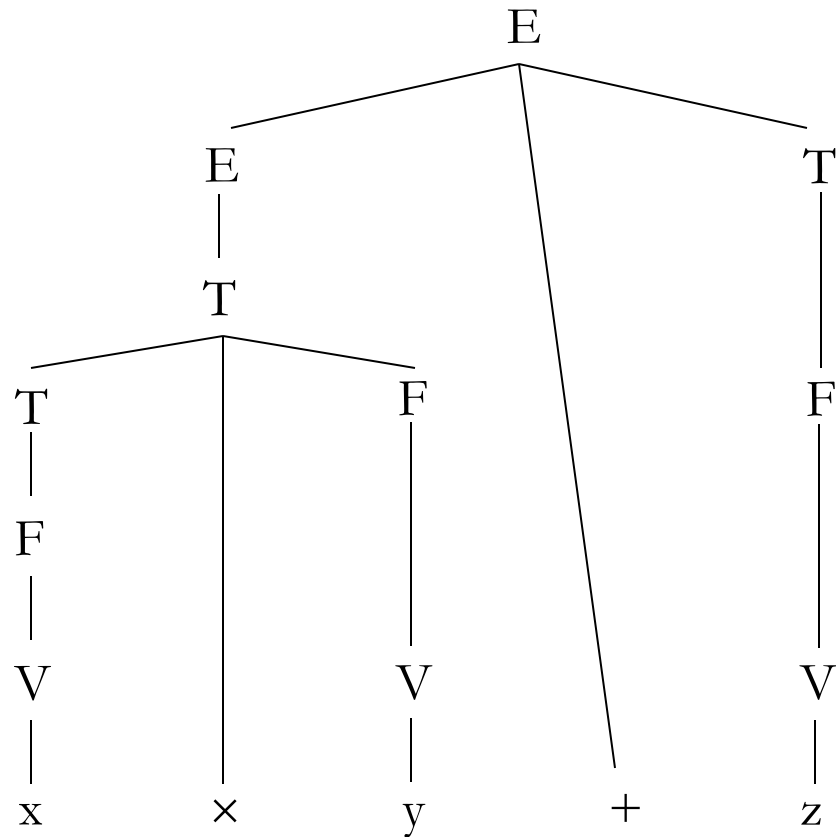A factor is either a variable or a parenthesized expression

Disambiguation

- Example

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

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

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

- Is ab, $baba$, $abbbaa$ in L?

- How about a, bba?

- What is the language of this CFG?

- Is the CFG ambiguous?