SE 3310b

Theoretical Foundations of Software Engineering

Context-Free Grammars

Aleksander Essex

Western Engineering

Context-free Languages

Non-regular languages

Recall our language:

$$L = \{0^n 1^n : n \geqslant 0\}$$

We used the pumping lemma to prove L is not regular. That means there's no DFA NFA or RE that can describe it. Ok then, so how *do* you describe it?

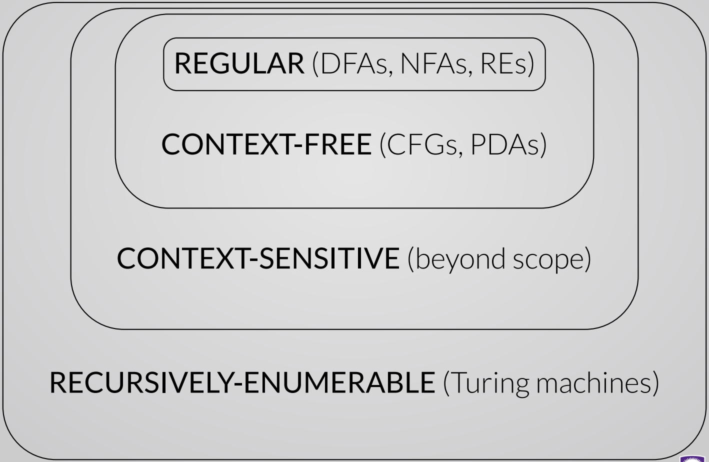
Western Engineering

Context-free Languages

We're ready to "graduate" to a broader class of languages: context-free languages (CFLs). The set of regular languages is a subset of the context-free languages, i.e., REG $\subset$ CFL.

That means if $L \in$ REG then $L \in$ CFL. Though as we will see, there exist languages $M \in$ CFL for which $M \notin$ REG.

Chomsky Hierarcy



Representing Languages

Language	Description Method: Automaton-based	Description Method: Expression-based
Regular	DFA/NFA	Regular expression
Context-free	Pushdown automaton	Context-free grammar
Context-sensitive	Linear bounded (not studied)	Context-sensitive grammar (not studied)
Recursively enumerable	Turing machine	Unrestricted grammar (not studied)

- **Previous lectures**: DFAs/NFAs/REs
- **This lecture**: Context-free grammar (CFGs)
- **Next lecture**: Pushdown automata (PDAs)
- **Later lectures**: Turing machines (TMs)

Western Engineering

Grammar

A grammar is a set of rules governing the structure of a string. In natural language it describes how sentences are constructed.

sentence	$\rightarrow$	noun-phrase verb-phrase
noun-phrase	$\rightarrow$	cmplx-noun | cmplx-noun prep-phrase
verb-phrase	$\rightarrow$	cmplx-verb | cmplx-verb prep-phrase
prep-phrase	$\rightarrow$	prep. cmplx noun
cmplx-noun	$\rightarrow$	article noun
cmplx-verb	$\rightarrow$	verb | verb noun-phrase
article	$\rightarrow$	*a* | *the*
noun	$\rightarrow$	*boy* | *girl* | *flower*
verb	$\rightarrow$	*touches* | *likes* | *flower*
prep.	$\rightarrow$	*with*

Grammar

Let's build a sentence with this grammar:

sentence	→	noun-phrase verb-phrase
	→	cmplx-noun verb-phrase
	→	article noun verb-phrase
	→	*a* noun verb-phrase
	→	*a boy* verb-phrase
	→	*a boy* cmplx-verb
	→	*a boy* verb
	→	*a boy sees*

Notice how we start out with a set of variables, and then we replace those variables with words, or other variables. We keep going until all that's left is words. We've used the grammar to *produce* a valid sentence.

Context-free Grammar

A context-free grammar (CFG) is formal grammar that describes the set of context-free languages.

Definition 1 (Context-free Grammar (CFG)).

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

1. V: A finite set of *variables*
2. Σ: A finite set of *terminals*
3. R: A finite set of *rules* in which a variable may be replaced by some concatenation of variables and terminals.
4. $S \in V$: A start variable

Context-free Language

Definition 2 (Context-free language)**.**

A language is *context-free* if and only if there exists some context-free grammar that describes it.

Context-free Grammar: The Idea

A context-free grammar is similar to a regular-expression. It's like recipe for producing strings. The set of all strings that can be *produced* or *derived* by this recipe constitute the language described by that grammar.

The idea is you start at the first variable. From there you can substitute variables for variables/terminals as per the *production rules*. When there's no more variables, you stop.

Western Engineering

Context-free Grammar: Example 1

Recall our language $L = \{0^n 1^n : n \geqslant 0\}$. Recall there's *no* DFA, NFA, or RE that describe it. But watch how easy it is to describe with a CFG:

$$\text{A} \quad \rightarrow \quad \text{0A1} \mid \epsilon$$

The interpretation of this production rule is:

- Start with A, then,
- Any instance of A can be replaced either by 0A1, or ϵ

Context-free Grammar: Example 1

So let's see this grammar in action. Suppose we use this grammar to derive the string 000111:

$$A$$
$$\downarrow$$
$$0A1$$
$$\downarrow$$
$$00A11$$
$$\downarrow$$
$$000A111$$
$$\downarrow$$
$$000111 \quad \leftarrow \text{No more variables, so we stop}$$

Context-free Grammar: Example 2

One important application of CFGs is to compiler theory in application of well-formed parenthesis. For example (()()) is in the language of well-formed parenthesis, but (() is not, nor is ())((().

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

Again, the interpretation of the production rule is:

- Start with S, then,
- Replace any instance of S with:
  - (S): S in a balanced pair of brackets
  - SS: S split in two components
  - ϵ: an empty string

Western Engineering

Context-free Grammar: Example 2

So let's see this grammar in action. Suppose we use this grammar to derive the string (()()):

$$S \\ \downarrow \\ (S) \\ \downarrow \\ (SS) \\ \downarrow \\ ((S)(S)) \\ \downarrow \\ (()())$$

(()()) ← No more variables, so we stop

Western Engineering

Context-free Grammar: Example 3

Let $L = \{w : w \text{ contains more 1s than 0s }\}$. Let's begin by considering the string that contains no 0's. We'll still need at least one 1 (so it contains more 1's than 0's), so let's start off with that rule, and then move on to other variables:

$$S \rightarrow T1T$$

Now, what should T be? Well we let T be replaced by ϵ, then we can terminate any T at any time, which seems useful.

Western Engineering

Context-free Grammar: Example 3

Next we need a rule that guarantees that any time we add a 0, we also add at least one 1:

$$T \rightarrow 0T1 \mid 1T0$$

Next, we should have the ability to add arbitrarily many additional 1's:

$$T \rightarrow 1T$$

Finally, we need the ability to "spawn" more variables in order to make arbitrary combinations of the above rules:

$$T \rightarrow TT$$

Context-free Grammar: Example 3

Our grammar looks like this:

$$\begin{array}{lcl} S & \rightarrow & T1T \\ T & \rightarrow & TT \mid 1T \mid 0T1 \mid 1T0 \mid \epsilon \end{array}$$

Try a few different derivations to convince yourself:

1. There is no way for this grammar to produce a string with equal or more 0's than 1's
2. It can produce any other string

Western Engineering

Regular languages are Context-free Languages

Theorem 3.

Every regular language is a context-free language.

Recall a language is regular iff there exists some DFA that recognizes it, and a language is context-free if and only if there exists some CFG that generates it. Therefore to show every regular language is context-free, it is sufficient to show every DFA has an equivalent CFG.

Western Engineering

Regular languages are Context-free Languages

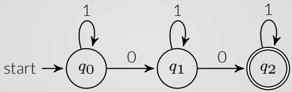
To convert a DFA to a CFG:

- All states $q_i \in Q$ become variables Q_i
- All symbols $s \in \Sigma$ become terminals
- All transitions $q_i \times s \to q_j$ becomes rules $Q_i \to sQ_j$
- Start state $q_0 \in Q$ becomes first rule $Q_0 \to \ldots$
- All transitions in to an accept state $q_f \in F$ become rules $Q_f \to \epsilon$

Western Engineering

Regular languages are Context-free Languages

Convert the following DFA to a CFG:



Following our conversion rules, this yields:

$$
\begin{array}{lcl}
Q_0 & \rightarrow & 1Q_0 \mid 0Q_1 \\
Q_1 & \rightarrow & 1Q_1 \mid 0Q_2 \\
Q_2 & \rightarrow & 1Q_2 \mid \epsilon
\end{array}
$$