Discrete Event Systems

Exercise Session 4

Roland Schmid

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

14 October 2021

1 Context-Free Grammars

Give context-free grammars for the following languages over the alphabet $\Sigma = \{0, 1\}$:

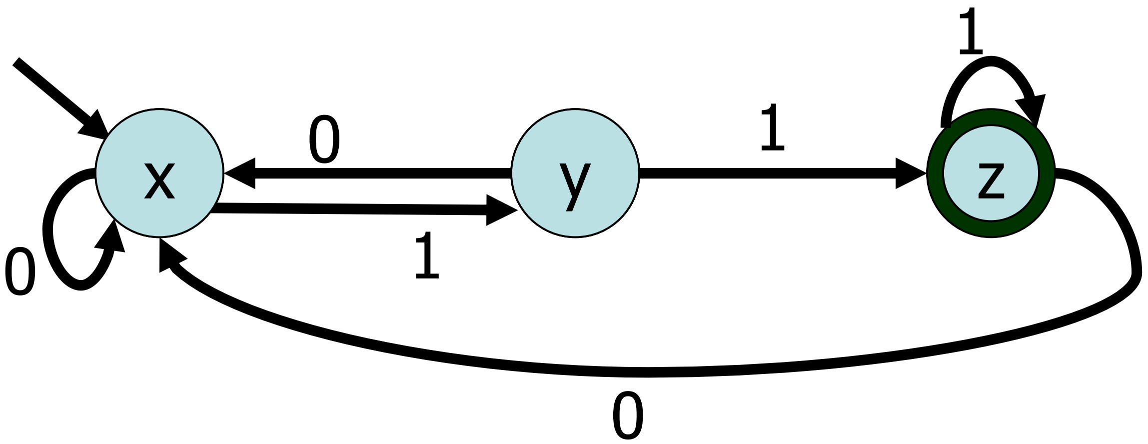
a) $L_1 = \{w \mid \text{the length of } w \text{ is odd}\}$

b) $L_2 = \{w \mid \text{contains more 1s than 0s}\}$

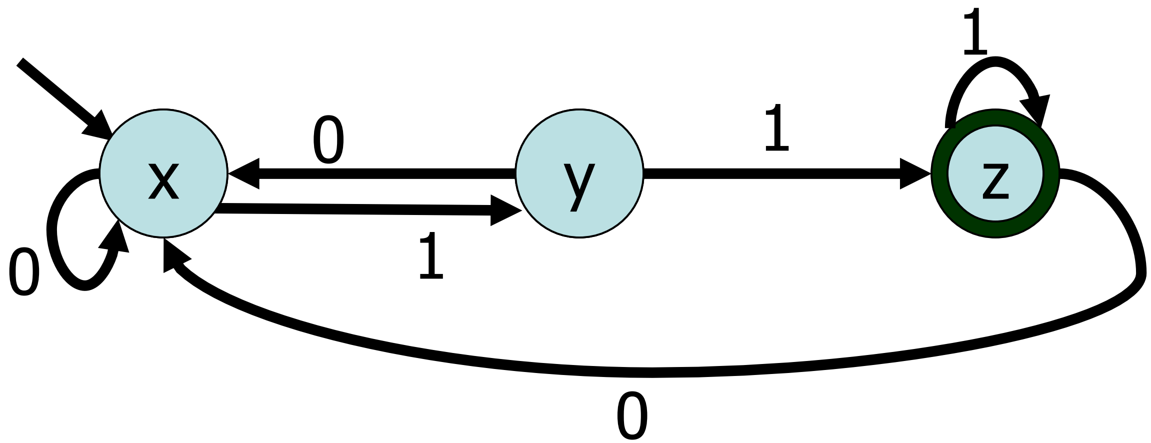
Model Robustness

- The class of regular languages was quite robust
  - Allows multiple ways for defining languages (automaton vs. regexp)
  - Slight perturbations of model do not change result (non-determinism)
- The class of context free languages is also robust:
  you can use either PDA's or CFG's to describe the languages in the class.
- However, it is less robust than regular languages when it comes to slight perturbations of the model:
  - Smaller classes
    - Right-linear grammars
    - Deterministic PDA's
  - Larger classes
    - Context Sensitive Grammars

Right Linear Grammars vs. Regular Languages



Right Linear Grammars vs. Regular Languages



- The DFA above can be simulated by the grammar
  - $x \rightarrow 0x \mid 1y$
  - $y \rightarrow 0x \mid 1z$
  - $z \rightarrow 0x \mid 1z \mid \varepsilon$

- Definition: A right-linear grammar is a CFG such that every production is of the form $A \rightarrow uB$, or $A \rightarrow u$ where u is a terminal string, and A,B are variables.

Right Linear Grammars vs. Regular Languages

- Theorem: If $M = (Q, \Sigma, \delta, q_0, F)$ is an NFA then there is a right-linear grammar $G(M)$ which generates the same language as M.

- *Proof*:
  - Variables are the states: $V = Q$
  - Start symbol is start state: $S = q_0$
  - Same alphabet of terminals Σ
  - A transition $q_1 \rightarrow a \rightarrow q_2$ becomes the production $q_1 \rightarrow aq_2$
  - For each transition, $q_1 \rightarrow aq_2$ where q_2 is an accept state, add $q_1 \rightarrow a$ to the grammar

- Homework: Show that the reverse holds. Right-linear grammar can be converted to a FSA. This implies that RL ≈ Right-linear CFL.

Right Linear Grammars vs. Regular Languages

- Theorem: If $M = (Q, \Sigma, \delta, q_0, F)$ is an NFA then there is a right-linear grammar $G(M)$ which generates the same language as M.

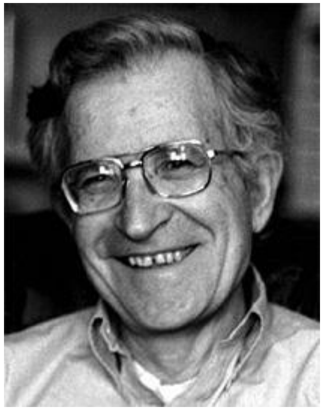
- *Proof*:
  - Variables are the states: $V = Q$
  - Start symbol is start state: $S = q_0$
  - Same alphabet of terminals Σ
  - A transition $q_1 \rightarrow a \rightarrow q_2$ becomes the production $q_1 \rightarrow aq_2$
  - For each transition, $q_1 \rightarrow aq_2$ where q_2 is an accept state, add $q_1 \rightarrow a$ to the grammar

- Homework: Show that the reverse holds. Right-linear grammar can be converted to a FSA. This implies that RL ≈ Right-linear CFL.

- Question: Can every CFG be converted into a right-linear grammar?

Chomsky Normal Form

- Chomsky came up with an especially simple type of context free grammars which is able to capture all context free languages, the Chomsky normal form (CNF).
- Chomsky's grammatical form is particularly useful when one wants to prove certain facts about context free languages. This is because assuming a much more restrictive kind of grammar can often make it easier to prove that the generated language has whatever property you are interested in.
- Noam Chomsky, linguist at MIT, creator of the Chomsky hierarchy, a classification of formal languages. Chomsky is also widely known for his left-wing political views and his criticism of the foreign policy of U.S. government.



Chomsky Normal Form

- Definition: A CFG is said to be in Chomsky Normal Form if every rule in the grammar has one of the following forms:

  - $S \rightarrow \varepsilon$ (ε for epsilon's sake only)
  - $A \rightarrow BC$ (dyadic variable productions)
  - $A \rightarrow a$ (unit terminal productions)

  where S is the start variable, A,B,C are variables and a is a terminal.

- Thus epsilons may only appear on the right hand side of the start symbol and other rights are either 2 variables or a single terminal.

CFG → CNF

- Converting a general grammar into Chomsky Normal Form works in four steps:

1. Ensure that the start variable doesn't appear on the right hand side of any rule.
2. Remove all epsilon productions, except from start variable.
3. Remove unit variable productions of the form $A \rightarrow B$ where A and B are variables.
4. Add variables and dyadic variable rules to replace any longer non-dyadic or non-variable productions

1 Context-Free Grammars

Give context-free grammars for the following languages over the alphabet $\Sigma = \{0, 1\}$:

a) $L_1 = \{w \mid \text{the length of } w \text{ is odd}\}$

b) $L_2 = \{w \mid \text{contains more 1s than 0s}\}$

2 Regular and Context-Free Languages

a) Consider the context-free grammar G with the production $S \to SS \mid 1S2 \mid 0$. Describe the language $L(G)$ in words, and prove that $L(G)$ is not regular.

b) The regular languages are a subset of the context-free languages. Give the context-free grammar for an arbitrary language L that is regular.

3 Context-Free or Not?

For the following languages, determine whether they are context free or not. Prove your claims!

a) $L = \{w\#x\#y\#z \mid w, x, y, z \in \{a, b\}^* \text{ and } |w| = |z|, |x| = |y|\}$

b) $L = \{w\#x\#y\#z \mid w, x, y, z \in \{a, b\}^* \text{ and } |w| = |y|, |x| = |z|\}$

4 Push Down Automata

For each of the following context free languages, draw a PDA that accepts L.

a) $L = \{u \mid u \in \{0,1\}^* \text{ and } u^{reverse} = u\} = \{u \mid \text{“}u \text{ is a palindrome”}\}$

b) $L = \{u \mid u \in \{0,1\}^* \text{ and } u^{reverse} \neq u\} = \{u \mid \text{“}u \text{ is no palindrome”}\}$

5 Ambiguity

Consider the following context-free grammar G with non-terminals S and A, start symbol S, and terminals "(", ")", and "0":

$$\begin{aligned} S &\rightarrow SA \mid \varepsilon \\ A &\rightarrow AA \mid (S) \mid 0 \end{aligned}$$

a) What are the eight shortest words produced by G?

b) Context-free grammars can be ambiguous. Prove or disprove that G is unambiguous.

c) Design a push-down automaton M that accepts the language $L(G)$. If possible, make M deterministic.

6 Counter Automaton

A push-down automaton is basically a finite automaton augmented by a stack. Consider a finite automaton that (instead of a stack) has an additional *counter* C, i.e., a register that can hold a single integer of arbitrary size. Initially, $C = 0$. We call such an automaton a *Counter Automaton* M. M can only increment or decrement the counter, and test it for 0. Since theoretically, all possible data can be coded into one single integer, a counter automaton has unbounded memory. Further, let $\mathcal{L}_{count}$ be the set of languages recognized by counter automata.

a) Let $\mathcal{L}_{reg}$ be the set of regular languages. Prove that $\mathcal{L}_{reg} \subseteq \mathcal{L}_{count}$.

b) Prove that the opposite is not true, that is, $\mathcal{L}_{count} \nsubseteq \mathcal{L}_{reg}$. Do so by giving a language which is in $\mathcal{L}_{count}$, but not in $\mathcal{L}_{reg}$. Characterize (with words) the kind of languages a counter automaton can recognize, but a finite automaton cannot.

c) Which automaton is stronger? A counter automaton or a push-down automaton? Explain your decision.