Week 8

CSE 390Z
May 16, 2023

Welcome in!

Grab your nametag and the workshop problems from the front

Announcements

Upcoming Assignments

● Quick Check 8	**Due:** Monday, May 22
● Imposter Syndrome Reflection	**Due:** Monday, May 22
● Homework Corrections 2	**Due:** Monday, May 29

Next Week

- Imposter Syndrome Panel

Conceptual Review

Regular Expressions (RegEx)

- Reminder: A set of strings is called a language.
- What Regular Expressions do is define a language for us, namely, the set of all strings matching some particular pattern.
- Regular Expressions are recursively defined, meaning they have a basis rule, a recursive rule, and obey the exclusion rule.
- Basis: empty string, no string, and a single character.
- The recursive rule is where the magic happens! There are three recursive rules for RegEx:

RegEx Recursive Rules (Operations)

- Union: If A and B are regular expressions then (A U B) is a regular expression that matches either A or B or both.
- Concatenation: If A and B are regular expressions then AB is a regular expression that matches the contents of A + the contents of B
- Wildcard (*): If A is a regular expression, A* is a regular expression which matches any string which can be subdivided into 0 (remember this) or more strings that match A

- Examples:
- (a U bc)
- 0(0 U 1)1
- 0*

Notes about RegEx

- While RegEx is powerful, they can't define every possible language. For example, the set of all palindromes can't be defined by regular expressions.
- The CSE 311, simple machine version of Regular Expressions are not interchangeable with the regular expressions some of you may be familiar with in some programming languages. These languages extend our definition of regular expressions with features that make these regular expressions more powerful.
- As a result, our definition of RegEx may have some limitations that don't exist in other versions of RegEx, just FYI!

Context-Free Grammars (CFGs): The Basics

- We can think of CFGs as being String “generators”
- Consist of an alphabet of “terminal symbols,” meaning symbols that don’t create another substitution into the resulting string, and a finite set V of nonterminal symbols which do create further substitutions.
- Each CFG has a start symbol which is one of the nonterminal symbols in V and is usually denoted S.
- Each of the nonterminal symbols has a production rule in the form of A -> w1|w2|w3..
- To generate a string using a CFG simply start with the start symbol S, then choose a nonterminal in the string and a production rule, then substitute in an element w(i) from the production rule into the nonterminal and rinse and repeat until you’re out of nonterminal characters!
- Example:

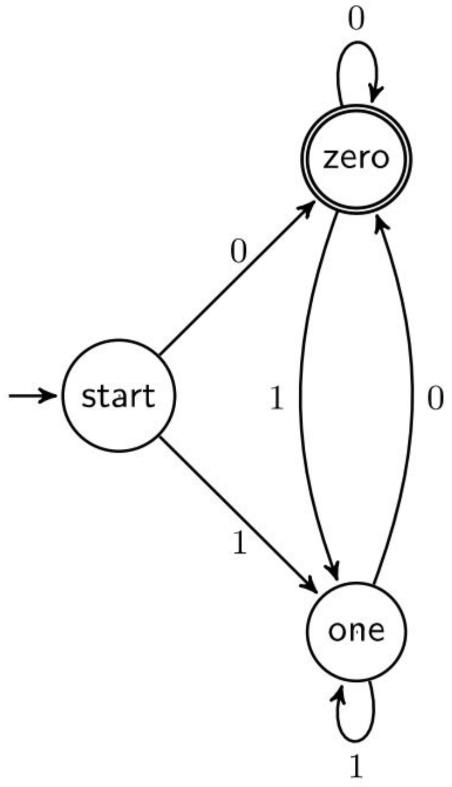
$$S \to 0S0|1S1|0|1|\varepsilon$$

Notes about CFGs

- CFGs can define more complex languages than Regular Expressions can. For example, they can be used to define the set of all palindromes, expressions with matched parentheses, arithmetic expressions, and even the syntax of the Java programming language!
- In fact, CFGs define a superset of Regular Expressions which will be proved in lecture!
- The main takeaway for RegEx and CFGs is that they give us ways of succinctly representing sets of strings.

Deterministic Finite Automata (DFAs)

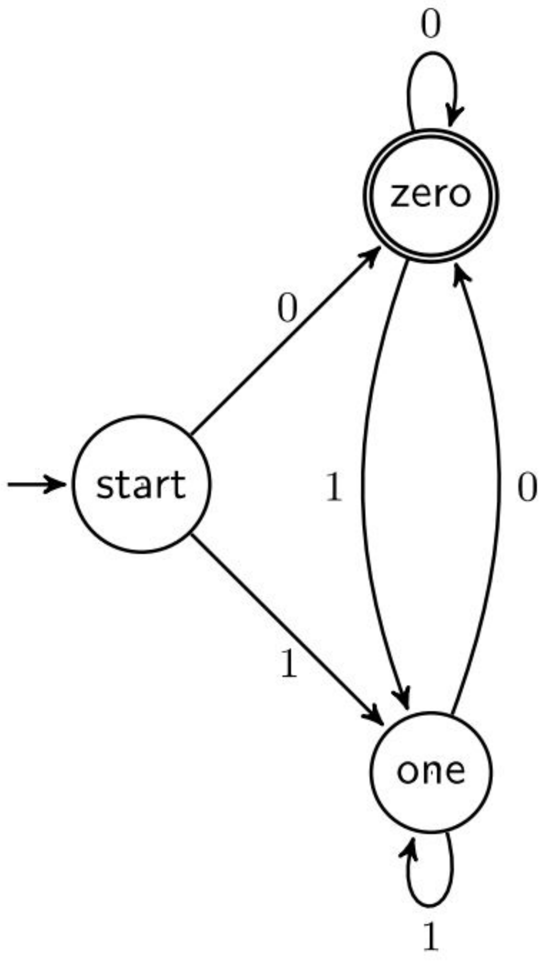
- DFA's allow us to check whether an input string matches a particular pattern
- Our machine will get an input string, and read through it one character at a time, updating its state at each step
- When it reads a character, it follows the arrow labeled with that character to its next state
- Once you have read your last character, accept the string if you are in a "final state" (double circle), otherwise reject the string



DFA Example

- Are the following strings accepted or rejected?
  - 011
  - 1010

- What set of strings does this DFA describe?



Regex, CFGs, and DFAs

Let $\Sigma = \{0, 1, 2\}$. Consider the language "all strings with an even number of 2's." We can design a regular expression, CFG, and DFA to construct this language.

Regex, CFGs, and DFAs

Let $\Sigma = \{0, 1, 2\}$. Consider the language “all strings with an even number of 2’s.” We can design a regular expression, CFG, and DFA to construct this language.

Regular Expression : **(0 U 1 U (2 (0U1)* 2))***

Regex, CFGs, and DFAs

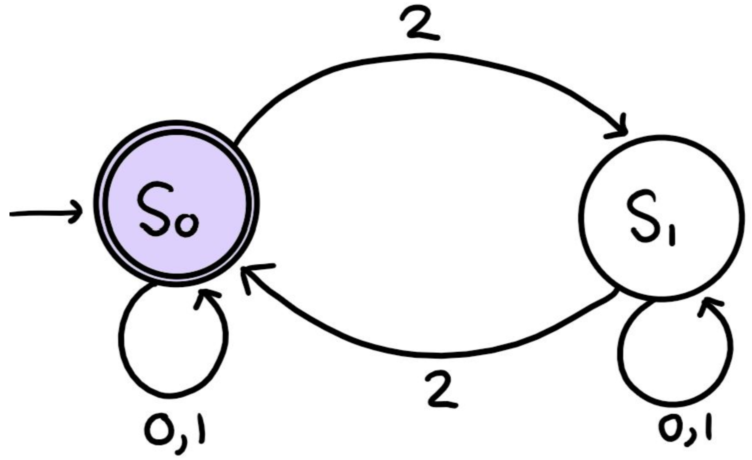
Let $\Sigma = \{0, 1, 2\}$. Consider the language “all strings with an even number of 2’s.” We can design a regular expression, CFG, and DFA to construct this language.

CFG : **S → Ɛ | 0S | 1S | S2S2S**

Regex, CFGs, and DFAs

Let $\Sigma = \{0, 1, 2\}$. Consider the language "all strings with an even number of 2's." We can design a regular expression, CFG, and DFA to construct this language.

DFA:



Workshop Problems