

CMSC 330: Organization of Programming Languages

Regular Expressions and Finite Automata

Introduction

- ▶ That's it for the basics of Ruby
 - If you need other material for your project, come to office hours or check out the documentation
- ▶ Next up: How do regular expressions (REs) really work?
 - Mixture of a very practical tool (string matching with REs) and some nice theory
 - A great computer science result

A Few Questions About REs

- ▶ What does a regular expression represent?
 - Just a set of strings
- ▶ What are the basic components of REs?
 - E.g., we saw that e^+ is the same as ee^*
- ▶ How are REs implemented?
 - We'll see how to build a structure to parse REs

Definition: Alphabet

- ▶ An alphabet is a **finite** set of symbols
 - Usually denoted Σ
- ▶ Example alphabets:
 - Binary: $\Sigma = \{0,1\}$
 - Decimal: $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$
 - Alphanumeric: $\Sigma = \{0-9,a-z,A-Z\}$

Definition: String

- ▶ A **string** is a finite sequence of symbols from Σ
 - ε is the empty string ("" in Ruby)
 - $|s|$ is the length of string s
 - ▶ $|Hello| = 5$, $|\varepsilon| = 0$
 - Note
 - ▶ $\emptyset$ is the empty set (with 0 elements)
 - ▶ $\emptyset \neq \{\varepsilon\} \neq \varepsilon$
- ▶ Example strings:
 - $0101 \in \Sigma = \{0,1\}$ (binary)
 - $0101 \in \Sigma = \text{decimal}$
 - $0101 \in \Sigma = \text{alphanumeric}$

Definition: String concatenation

- ▶ String **concatenation** is indicated by juxtaposition
 - If $s_1 = \text{super}$ and $s_2 = \text{hero}$, then $s_1s_2 = \text{superhero}$
 - Sometimes also written $s_1 \cdot s_2$
 - For any string s , we have $s\varepsilon = \varepsilon s = s$
 - You **can** concatenate strings from different alphabets; then the new alphabet is the union of the originals:
 - ▶ If $s_1 = \text{super} \in \Sigma_1 = \{s,u,p,e,r\}$ and $s_2 = \text{hero} \in \Sigma_2 = \{h,e,r,o\}$, then $s_1s_2 = \text{superhero} \in \Sigma_3 = \{e,h,o,p,r,s,u\}$



Definition: Language

- ▶ A **language** L is a set of strings over an alphabet
- ▶ Example: The set of phone numbers over the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 9, (,), -\}$
 - Give an example element of this language (123) 456-7890
 - Are all strings over the alphabet in the language? No
 - Is there a Ruby regular expression for this language?
`/\(\d{3,3}\) \d{3,3}-\d{4,4}/`
- ▶ Example: The set of all strings over Σ
 - Often written Σ^*

Definition: Language (cont.)

- ▶ Example: The set of strings of length 0 over the alphabet $\Sigma = \{a, b, c\}$
 - $L = \{s \mid s \in \Sigma^* \text{ and } |s| = 0\} = \{\epsilon\} \neq \emptyset$
- ▶ Example: The set of all valid Ruby programs
 - Is there a Ruby regular expression for this language?
No. Matching (an arbitrary number of) brackets so that they are balanced is impossible using REs `{ { { ... } } }`
- ▶ Can REs represent all possible languages?
 - The answer turns out to be no!
 - The languages represented by regular expressions are called, appropriately, the **regular languages**

Operations on Languages

- ▶ Let Σ be an alphabet and let L, L_1, L_2 be languages over Σ
- ▶ Concatenation $L_1 L_2$ is defined as
 - $L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$
- ▶ Union is defined as
 - $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$
- ▶ **Kleene closure** is defined as
 - $L^* = \{x \mid \varepsilon \text{ or } x \in L \text{ or } x \in LL \text{ or } x \in LLL \text{ or } \dots\}$

Definition: Regular Expressions

- ▶ Given an alphabet Σ , the **regular expressions** over Σ are defined inductively as

regular expression	denotes language
$\emptyset$	$\emptyset$
ε	$\{\varepsilon\}$
each element $\sigma \in \Sigma$	$\{\sigma\}$

Constants

Definition: Regular Expressions (cont.)

- ▶ Let A and B be regular expressions denoting languages L_A and L_B , respectively

regular expression	denotes language
AB	$L_A L_B$
$(A B)$	$L_A \cup L_B$
A^*	L_A^*

Operations

- ▶ There are no other regular expressions over Σ

Regular Expressions Denote Languages

- ▶ By applying operations on constants
 - Generates a set of strings (i.e., a language)
 - Examples
 - ▶ $a \rightarrow \{a\}$
 - ▶ $a|b \rightarrow \{a\} \cup \{b\} = \{a, b\}$
 - ▶ $a^* \rightarrow \{\epsilon\} \cup \{a\} \cup \{aa\} \cup \dots = \{\epsilon, a, aa, \dots\}$
- ▶ If $s \in$ language generated by a RE r , we say that r **accepts**, **describes**, or **recognizes** string s

Precedence

- Order in which operators are applied
 - In arithmetic
 - Multiplication $\times$ > addition $+$
 - $2 \times 3 + 4 = (2 \times 3) + 4 = 10$
 - In regular expressions
 - Kleene closure $*$ > concatenation > union $|$
 - $ab|c = (a b) | c = \{“ab”, “c”\}$
 - $ab^* = a (b^*) = \{“a”, “ab”, “abb” \dots\}$
 - $a|b^* = a | (b^*) = \{“a”, “”, “b”, “bb”, “bbb” \dots\}$
 - Can change order using parentheses $()$
 - E.g., $a(b|c)$, $(ab)^*$, $(a|b)^*$

Regular Languages

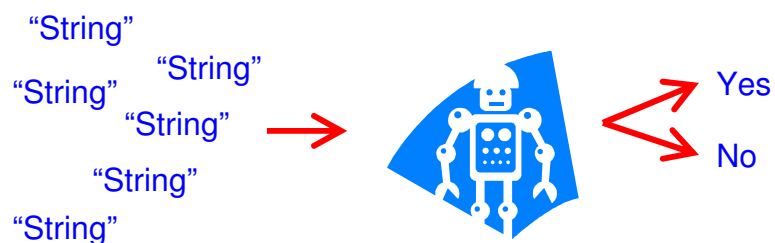
- The languages that can be described using regular expressions are the **regular languages** or **regular sets**
- Not all languages are regular
 - Examples (without proof):
 - The set of palindromes over Σ
 - reads the same backward or forward
 - $\{a^n b^n \mid n > 0\}$ (a^n = sequence of n a's)
- Almost all programming languages are not regular
 - But aspects of them sometimes are (e.g., identifiers)
 - Regular expressions are commonly used in parsing tools

Ruby Regular Expressions

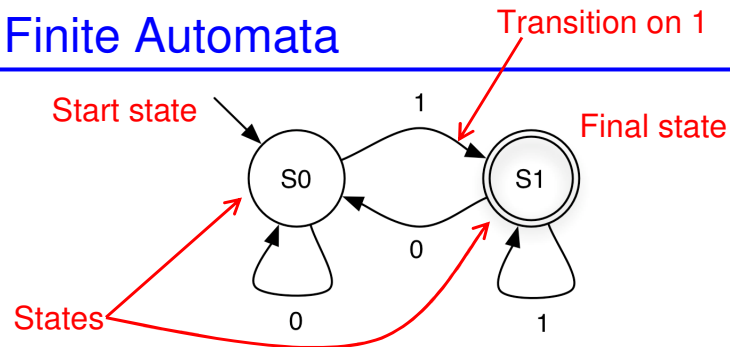
- ▶ Almost all of the features we've seen for Ruby REs can be reduced to this formal definition
 - `/Ruby/` – concatenation of single-character REs
 - `/(Ruby|Regular)/` – union
 - `/(Ruby)*/` – Kleene closure
 - `/(Ruby)+/` – same as `(Ruby)(Ruby)*`
 - `/(Ruby)?/` – same as `( $\epsilon$ |(Ruby))` (`//` is `$\epsilon$`)
 - `/[a-z]/` – same as `(a|b|c|...|z)`
 - `/[^0-9]/` – same as `(a|b|c|...)` for `a,b,c,...  $\in$   $\Sigma$  - {0..9}`
 - `^`, `$` – correspond to extra characters in alphabet

Implementing Regular Expressions

- ▶ We can implement a regular expression by turning it into a **finite automaton**
 - A “machine” for recognizing a regular language



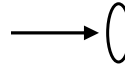
Finite Automata



- ▶ Machine starts in **start** or **initial** state
- ▶ Repeat until the end of the string is reached
 - Scan the next symbol **s** of the string
 - Take transition edge labeled with **s**
- ▶ String is **accepted** if automaton is in **final** state when end of string reached

Finite Automata: States

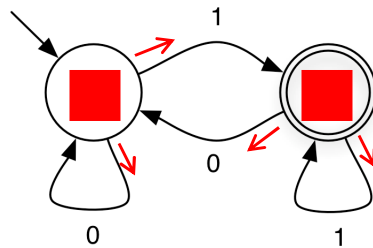
- ▶ Start state
 - State with incoming transition from no other state
 - Can have only 1 start state



- ▶ Final states
 - States with double circle
 - Can have 0 or more final states
 - Any state, including the start state, can be final



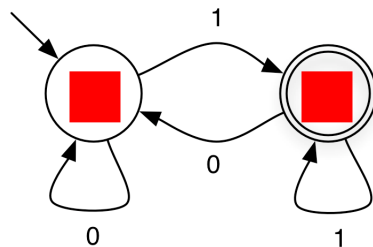
Finite Automaton: Example 1



0 0 1 0 1 1
⏟ ⏟ ⏟ ⏟ ⏟ ⏟

accepted

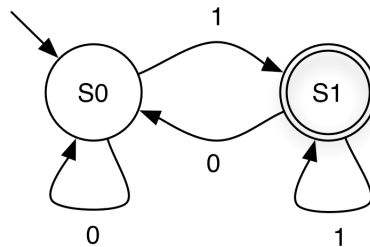
Finite Automaton: Example 2



0 0 1 0 1 0
⏟ ⏟ ⏟ ⏟ ⏟ ⏟

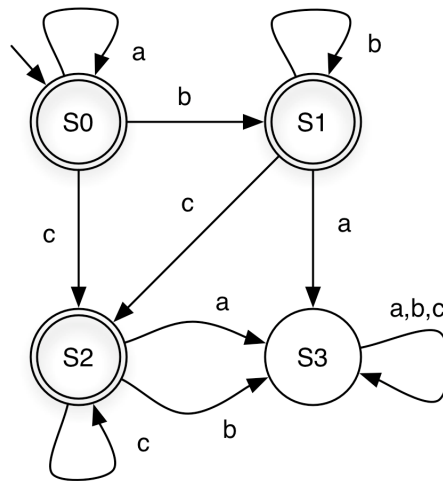
not accepted

What Language is This?



- ▶ All strings over $\{0, 1\}$ that end in 1
- ▶ What is a regular expression for this language?
 $(0|1)^*1$

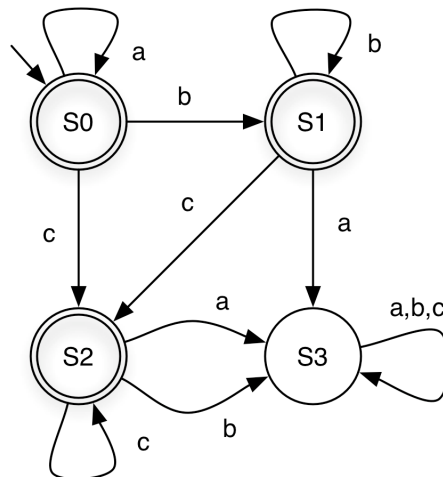
Finite Automaton: Example 3



string	state at end	accepts ?
aabcc	S2	Y
acc	S2	Y
bbc	S2	Y
aabbb	S1	Y
aa	S0	Y
ϵ	S0	Y
acba	S3	N

(a,b,c notation shorthand for three self loops)

Finite Automaton: Example 3 (cont.)



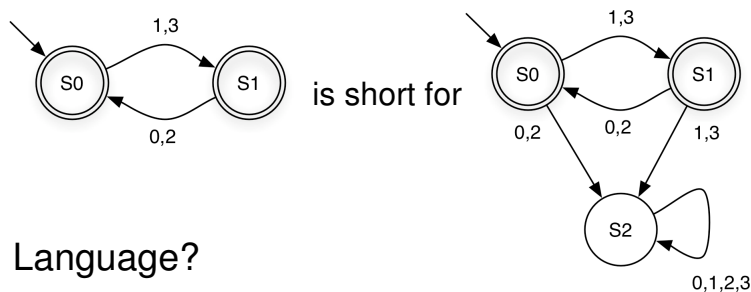
What language does this DFA accept?

$a^*b^*c^*$

S3 is a **dead state** – a nonfinal state with **no** transition to another state

Dead State: Shorthand Notation

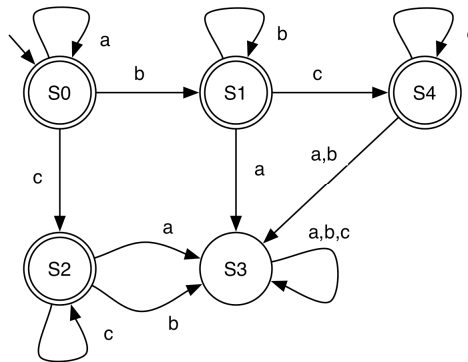
- If a transition is omitted, assume it goes to a dead state that is not shown



- Language?

- Strings over $\{0,1,2,3\}$ with alternating even and odd digits, beginning with odd digit

What Lang. Does This FA Accept?

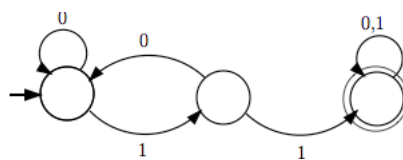


$a^*b^*c^*$ again, so DFAs are not unique

Practice

Give the English descriptions and the DFA or regular expression of the following languages:

- ▶ $((0|1)(0|1)(0|1)(0|1)(0|1))^*$
 - All strings with length a multiple of 5
- ▶ $(01)^*|(10)^*|(01)^*0|(10)^*1$
 - All alternating binary strings



All binary strings containing the substring "11"

Practice

- ▶ Give the regular expressions and finite automata for the following languages
 - You and your neighbors' names
 - All protein-coding DNA strings (including only ATCG and appearing in multiples of 3)
 - All binary strings containing an even length substring of all 1's
 - All binary strings containing exactly two 1's
 - All binary strings that start and end with the same number

Review

- ▶ Languages
 - Sets of strings
 - Operations on languages
- ▶ Regular expressions
 - Constants
 - Operators
 - Precedence
- ▶ Finite automata
 - States
 - Transitions
 - Accept strings