CS420

Regular Languages

Wednesday, February 1, 2023
UMass Boston Computer Science

[Diagram showing hierarchy of automata starting with Turing Machines, then Linear bounded Automata, Push-down Automata, and ending with Finite State Automata = Regular Languages!]

Finite State Automata = Regular Languages!
Announcements

• HW 0 in
  • Due Tues 1/31 11:59pm EST

• HW 1 out
  • Due Tues 2/7 11:59pm EST

• Quiz preview:
  Why do we know that a language is a regular language if it has an FSM recognizing it?
Last Time: Computation and Languages

- The **language** of a machine is the **set of all strings that it accepts**

- A **computation model** is **equivalent to the set of machines** it defines
  - E.g., all possible Finite State Automata are a computation model

---

**Definition**

A **finite automaton** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the **states**,
2. \(\Sigma\) is a finite set called the **alphabet**,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**,
4. \(q_0 \in Q\) is the **start state**, and
5. \(F \subseteq Q\) is the **set of accept states**.

---

- Thus: a **computation model** is also **equivalent to a set of languages**

\[ M \text{ accepts } w \text{ if } \hat{\delta}(q_0, w) \in F \]
Last Time: Regular Languages: Definition

If a finite automaton (FSM) recognizes a language, then that language is called a regular language.

A language is a set of strings.

$M$ recognizes language $A$ if $A = \{w | M \text{ accepts } w\}$.
Last Time: A Language, Regular or Not?

• If given: a Finite Automaton $M$
  • We know: $L(M)$, the language recognized by $M$, is a regular language
  • Because:
    
    If a finite automaton (FSM) recognizes a language, then that language is called a regular language.

• If given: a Language $A$
  • Is $A$ a regular language?
    • Not necessarily!
  • How do we determine, i.e., prove, that $A$ is a regular language?
An Inference Rule: Modus Ponens

Premises
• If $P$ then $Q$
• $P$ is true

Conclusion
• $Q$ is true

Example Premises
• If there is an FSM recognizes language $A$, then $A$ is a regular language
• There is an FSM $M$ where $L(M) = A$

Conclusion
• $A$ is a regular language!

We know this (definition of regular language)

... then we need to show

If we want to prove this ...
Proving a Language is Regular: Example

Prove that the following language is regular:

\[ L = \{ w \mid w \text{ is a string with an odd # of } 1\text{s} \} \]

\[ \Sigma = \{ 0, 1 \} \]
Proving a Language is Regular: Example

**Statements**
1. If an FSM recognizes $L$, then $L$ is a regular language
2. $M = (Q, \Sigma, \delta, q_0, F)$ is an FSM (todo)
3. $M$ recognizes $L$
4. $L = \{ w \mid w \text{ is string with odd # of 1s} \}$ is a regular language

**Justifications**
1. Def. of a Regular Language
2. Definition of an FSM
3. This is hard problem! In this class, we use tests.
4. Stmt #1 & #3 (modus ponens)
Tips on Designing Finite Automata

Analogy

Finite Automata ~ “Programs” ::
Designing Finite Automata ~ “Programming”!

1. **Confirm understanding** of the problem
   - Create tests: examples and expected results (accept / reject)

In programming, to “understand” a problem, create examples!
FSM $M$ Examples: accept strs with odd # 1s

• On input 1:
  • Accept
• On input 0:
  • Reject
• On input 01:
  • Accept
• On input 11:
  • Reject
• On input 1101:
  • Accept
• On input $\varepsilon$
  • Reject
Tips on Designing Finite Automata

Analogy
Finite Automata ~ “Programs” ::
*Designing* Finite Automata ~ “Programming”!

1. **Confirm understanding** of the problem
   - Create tests: examples and expected results (accept / reject)

2. **Decide information to “remember”**
   - These are the machine states: some are accept states; one is start state

3. **Determine transitions** between states
Designing FSM $M$: accept strs with odd # 1s

- **States:**
  - 2 states:
    - seen even 1s so far
    - seen odds 1s so far

- **Alphabet:** 0 and 1

- **Transitions:**

- **Start / Accept states:**
Tips on Designing Finite Automata

Analogy
Finite Automata $\sim$ “Programs” ::
\textit{Designing} Finite Automata $\sim$ “Programming”!

1. **Confirm understanding of the problem**
   - Create tests: examples and expected results (accept / reject)

2. **Decide information to “remember”**
   - These are the machine states: some are accept states; one is start state

3. **Determine transitions** between states

4. **Test machine behaves as expected**
   - Use initial examples; and create additional tests if needed
Does the Machine Accept Expected Strings?

- On input 1:
  - Accept
- On input 0:
  - Reject
- On input 01:
  - Accept
- On input 11:
  - Reject
- On input 1101:
  - Accept
- On input ε
  - Reject
Proving a Language is Regular: Example

**Statements**

1. If an FSM recognizes $L$, then $L$ is a regular language

2. $M = (\text{even}, \text{odd})$ is an FSM

3. $M$ recognizes $L$

4. $L = \{ w \mid w \text{ is string with odd # of 1s} \}$

**Justifications**

1. Def. of a Regular Language

2. Definition of an FSM

3. See examples. This isn’t a proof, but good enough for programmers(?) and CS 420

4. Stmt #1 & #3 (modus ponens)
In-class exercise

• **Prove:** the following language is a regular language:
  • \( A = \{ w \mid w \text{ has exactly three 1's} \} \)
  • Key step: design a finite automata that recognizes it!

• Where \( \Sigma = \{ 0, 1 \} \)

• Remember:

---

**Definition**

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \( Q \) is a finite set called the **states**, 
2. \( \Sigma \) is a finite set called the **alphabet**, 
3. \( \delta : Q \times \Sigma \rightarrow Q \) is the **transition function**, 
4. \( q_0 \in Q \) is the **start state**, and 
5. \( F \subseteq Q \) is the **set of accept states**.
In-class exercise Solution

- Design finite automata recognizing:
  - \{w \mid w \text{ has exactly three 1's}\}

- States:
  - Need one state to represent how many 1's seen so far
  - \( Q = \{q_0, q_1, q_2, q_3, q_{4+}\}\)

- Alphabet: \( \Sigma = \{0, 1\}\)

- Transitions:

- Start state:
  - \( q_0 \)

- Accept states:
  - \( \{q_3\}\)

So finite automata are used to **recognize simple string patterns**?

**Yes!**

Do you know a “programming language” to recognize simple string patterns?

Make sure to test this with your examples!
So Far: Finite State Automaton, a.k.a. DFAs

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the **states**,
2. $\Sigma$ is a finite set called the **alphabet**,
3. $\delta: Q \times \Sigma \to Q$ is the transition function,$^1$
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the set of accept states.

- **Key characteristic:**
  - Has a **finite** number of states
  - i.e., a “program” with access to only a **single cell** of memory,
    - Where: states = the possible values that can be written to memory

- Often used for **text matching**
Combining DFAs?

Password Requirements

- Passwords must have a minimum length of ten (10) characters - but more is better!
- Passwords **must include at least 3** different types of characters:
  - upper-case letters (A-Z)
  - lower-case letters (a-z)
  - symbols or special characters (% , & , * , $, etc.)
  - numbers (0-9)
- Passwords cannot contain all or part of your email address
- Passwords cannot be re-used

To match all requirements, combine smaller DFAs into one big DFA?

https://www.umb.edu/it/password

(We do this with programs all the time)
Password Checker DFAs

What if this is not a DFA?

$M_5$: AND

$M_3$: OR

$M_1$: Check special chars

$M_2$: Check uppercase

$M_4$: Check length

Want to be able to easily combine DFAs, i.e., **composability**

We want these operations:

- OR: $\text{DFA} \times \text{DFA} \rightarrow \text{DFA}$
- AND: $\text{DFA} \times \text{DFA} \rightarrow \text{DFA}$

To combine more than once, operations must be **closed**!
“Closed” Operations

• Set of Natural numbers = \{0, 1, 2, \ldots\}
  • Closed under addition:
    • if \( x \) and \( y \) are Natural numbers,
    • then \( z = x + y \) is a Natural number
  • Closed under multiplication?
    • yes
  • Closed under subtraction?
    • no

• Integers = \{\ldots, -2, -1, 0, 1, 2, \ldots\}
  • Closed under addition and multiplication
  • Closed under subtraction?
    • yes
  • Closed under division?
    • no

• Rational numbers = \{x \mid x = y/z, y and z are Integers\}
  • Closed under division?
    • No?
    • Yes if \( z \neq 0 \)

A set is closed under an operation if: the result of applying the operation to members of the set is in the same set.
Why Care About Closed Ops on Reg Langs?

• Closed operations preserve “regularness”

• i.e., it preserves the same computation model!

• This way, a “combined” machine can be “combined” again!

We want:
OR, AND : DFA × DFA → DFA

• So this semester, we will look for operations that are closed!
Password Checker: “OR” = “Union”

$M_3$: OR

$M_1$: Check special chars

$M_2$: Check uppercase

(a)

???

(b)
Password Checker: “OR” = “Union”

$M_3$: OR

$M_1$: Check special chars

$M_2$: Check uppercase

(a)
Union of Languages

Let the alphabet $\Sigma$ be the standard 26 letters \{a, b, \ldots, z\}. If $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$, then

$$A \cup B = \{\text{good, bad, boy, girl}\}$$
Check-in Quiz 2/1

On gradescope