Regular Expressions

Thursday, September 29, 2022
Announcements

• HW 2
  • due Sunday 10/2 11:59pm EST
Last Time: A DFA’s Language

- Let DFA $M = (Q, \Sigma, \delta, q_0, F)$

- $M$ accepts $w$ if $\hat{\delta}(q_0, w) \in F$

- $M$ recognizes language $\{w \mid M \text{ accepts } w\}$

Definition: A DFA’s language is a regular language
Last Time: An NFA’s Language

• Let \( N = (Q, \Sigma, \delta, q_0, F) \)

• \( N \) accepts \( w \) if \( \hat{\delta}(q_0, w) \cap F \neq \emptyset \)
  • i.e., computation ends in at least one accept state

• \( N \) recognizes language \( \{ w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \} \)

An NFA’s language is a _____ language?
Last Time: Concatenation Closed for Reg Langs?

• Combining DFAs to recognize concatenation of languages ...
  ... produces an NFA

• So to prove concatenation is closed ...
  ... we must prove that NFAs also recognize regular languages.

Specifically, we must prove: NFAs $\Leftrightarrow$ regular languages
How to Prove a Statement: $X \Leftrightarrow Y$

$X \Leftrightarrow Y = \text{“}X \text{ if and only if } Y\text{”} = X \text{ iff } Y = X \iff Y$

Proof at minimum has 2 required parts:

1. $\Rightarrow$ if $X$, then $Y$
   - “forward” direction
   - assume $X$, then use it to prove $Y$

2. $\Leftarrow$ if $Y$, then $X$
   - “reverse” direction
   - assume $Y$, then use it to prove $X$
Proving NFAs Recognize Regular Langs

Theorem:
A language \( L \) is regular if and only if some NFA \( N \) recognizes \( L \).

Proof:
\[ \Rightarrow \text{If } L \text{ is regular, then some NFA } N \text{ recognizes it.} \]
(\text{Easier})
\begin{itemize}
  \item We know: if \( L \) is regular, then a DFA exists that recognizes it.
  \item So to prove this part: Convert that DFA \( \rightarrow \) an equivalent NFA! (see HW 2)
\end{itemize}
\[ \Leftarrow \text{If an NFA } N \text{ recognizes } L, \text{ then } L \text{ is regular.} \]
(\text{Harder})
\begin{itemize}
  \item We know: for \( L \) to be regular, there must be a DFA recognizing it
  \item Proof Idea for this part: Convert given NFA \( N \) \( \rightarrow \) an equivalent DFA
\end{itemize}
How to convert NFA \(\rightarrow\) DFA?

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.

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

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

Proof idea:
Let each “state” of the DFA = set of states in the NFA.
NFA computation can be in **multiple** states

DFA computation can only be in **one** state

So encode: a **set of NFA states** as one DFA state

This is similar to the proof strategy from **“Closure of union”** where: a state = a pair of states
Convert NFA$\rightarrow$DFA, Formally

• Let $NFA \; N = (Q, \Sigma, \delta, q_0, F)$

• An equivalent DFA $M$ has states $Q' = \mathcal{P}(Q)$ (power set of $Q$)
Example:

The NFA $N_4$

A DFA $D$ that is equivalent to the NFA $N_4$
NFA⇒DFA

Have: NFA $N = (Q, \Sigma, \delta, q_0, F)$
Want: DFA $M = (Q', \Sigma, \delta', q_0', F')$

1. $Q' = \mathcal{P}(Q)$  \hspace{1cm} A DFA state = a set of NFA states

2. For $R \in Q'$ and $a \in \Sigma$,
   $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$  \hspace{1cm} A DFA step = an NFA step for all states in the set

3. $q_0' = \{q_0\}$

4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$

No empty transitions
Flashback: Adding Empty Transitions

• Define the set $\varepsilon$-REACHABLE($q$)
  • ... to be all states reachable from $q$ via zero or more empty transitions

(Defined recursively)

• **Base case:** $q \in \varepsilon$-REACHABLE($q$)

• **Inductive case:**

$$\varepsilon$-REACHABLE($q$) = \{ r \mid p \in \varepsilon$-REACHABLE($q$) and $r \in \delta(p, \varepsilon) \}$$

A state is in the reachable set if...

... there is an empty transition to it from another state in the reachable set.
NFA→DFA

Have: NFA \( N = (Q, \Sigma, \delta, q_0, F) \)

Want: DFA \( M = (Q', \Sigma, \delta', q_0', F') \)

1. \( Q' = \mathcal{P}(Q) \)

2. For \( R \in Q' \) and \( a \in \Sigma \),
   \[
   \delta'(R, a) = \bigcup_{r \in R} \delta(r, a) \quad \text{ε-REACHABLE}(\delta(r, a))
   
   \]

3. \( q_0' = \{q_0\} \quad \text{ε-REACHABLE}(q_0) \)

4. \( F' = \{R \in Q' | R \text{ contains an accept state of } N\} \)

With empty transitions

Almost the same, except ...
Proving NFAs Recognize Regular Langs

**Theorem:**
A language $L$ is regular if and only if some NFA $N$ recognizes $L$.

**Proof:**

⇒ If $L$ is regular, then some NFA $N$ recognizes it.
  
  (Easier)
  - We know: if $L$ is regular, then a DFA exists that recognizes it.
  - So to prove this part: Convert that DFA → an equivalent NFA! (see HW 2)

⇐ If an NFA $N$ recognizes $L$, then $L$ is regular.
  
  (Harder)
  - We know: for $L$ to be regular, there must be a DFA recognizing it
  - Proof Idea for this part: Convert given NFA $N$ → an equivalent DFA ...
    … using our NFA to DFA algorithm!
Concatenation is Closed for Regular Langs

Proof

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$, and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$.

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

1. $Q = Q_1 \cup Q_2$
2. The state $q_1$ is the same as the start state of $N_1$
3. The accept states $F_2$ are the same as the accept states of $N_2$
4. Define $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$
\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \text{ and } q \not\in F_1 \\
\delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\
\delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\
\delta_2(q, a) & q \in Q_2.
\end{cases}
$$

If a language has an NFA recognizing it, then it is a regular language.

If language is regular, then it has an NFA recognizing it ...
Flashback: Union is Closed For Regular Langs

**Theorem**

The class of regular languages is closed under the union operation.

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.

**Proof:**

- How do we prove that a language is regular?
  - Create a DFA or NFA recognizing it!
- Combine the machines recognizing $A_1$ and $A_2$
  - Should we create a DFA or NFA?
Flashback: Union is Closed For Regular Langs

Proof
• Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize $A_1$, $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize $A_2$,

• Construct: a new machine $M = (Q, \Sigma, \delta, q_0, F)$ using $M_1$ and $M_2$

• states of $M$: $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$
  This set is the Cartesian product of sets $Q_1$ and $Q_2$

• $M$ transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

• $M$ start state: $(q_1, q_2)$

• $M$ accept states: $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

State in $M = M_1$ state + $M_2$ state

$M$ step = a step in $M_1$ + a step in $M_2$

Accept if either $M_1$ or $M_2$ accept
Union is Closed for Regular Languages

Add new start state, and $\epsilon$-transitions to old start states
Union is Closed for Regular Languages

**PROOF**

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$, and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
2. The state $q_0$ is the start state of $N$.
3. The set of accept states $F = F_1 \cup F_2$. 

 Alternate Proof, with NFAs
Union is Closed for Regular Languages

**PROOF**

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$, and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
2. The state $q_0$ is the start state of $N$.
3. The set of accept states $F = F_1 \cup F_2$.
4. Define $\delta$ so that for any $q \in Q$ and any $a \in \Sigma$, 

\[
\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \\
\delta_2(q, a) & q \in Q_2 \\
\{q_1, q_2\} & q = q_0 \text{ and } a = \epsilon \\
\emptyset & q = q_0 \text{ and } a \neq \epsilon 
\end{cases}
\]
List of Closed Ops for Reg Langs (so far)

- Union

- Concatentation
  - Kleene Star (repetition)
Kleene Star Example

Let the alphabet $\Sigma$ be the standard 26 letters $\{a, b, \ldots, z\}$.

If $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$, then

$$A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad,}
\text{goodgoodgood, goodgoodbad, goodbadgood, goodbadbadbad, \ldots}\}$$

Note: repeat zero or more times

(this is an infinite language!)
Kleene Star

New start (and accept) state, $\varepsilon$-transitions to old start state

Old accept states $\varepsilon$-transition to old start state
In-class exercise:

Kleene Star is Closed for Regular Langs

**THEOREM**

The class of regular languages is closed under the star operation.
Kleene Star is Closed for Regular Langs

**Proof** Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1^*$. 
Kleene Star is Closed for Regular Langs

**Proof** Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1^*$. 

1. $Q = \{q_0\} \cup Q_1$
2. The state $q_0$ is the new start state.
3. $F = \{q_0\} \cup F_1$

Kleene star of a language must accept the empty string!
Kleene Star is Closed for Regular Langs

**Proof** Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1^\ast$.

1. $Q = \{q_0\} \cup Q_1$
2. The state $q_0$ is the new start state.
3. $F = \{q_0\} \cup F_1$
4. Define $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_e$,

$$\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\
\delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\
\delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\
\{q_1\} & q = q_0 \text{ and } a = \varepsilon \\
\emptyset & q = q_0 \text{ and } a \neq \varepsilon.
\end{cases}$$
Many More Closed Operations on Regular Languages!

• Complement
• Intersection
• Difference
• Reversal
• Homomorphism
• (See HW2)
Why do we care about these ops?

- Union
- Concat
- Kleene star

- The are sufficient to represent all regular languages!
- I.e., they define regular expressions
So Far: Regular Language Representations

1. State diagram (NFA/DFA)

2. Formal description
   1. \( Q = \{ q_1, q_2, q_3 \} \),
   2. \( \Sigma = \{0,1\} \),
   3. \( \delta \) is described as

3. \[ \Sigma^* 001 \Sigma^* \]

4. \( q_1 \) is the start state, and
5. \( F = \{ q_2 \} \).

Analogy:
- All regular languages ~ a “programming language”
- One regular language ~ a “program” (e.g., find strings containing 001)

A practical application: text search

Need a more concise (textual) notation
Regular Expressions: A Widely Used Programming Language (inside other programming languages)

- Unix
- Perl
- Python
- Java
Why do we care about these ops?

- Union
- Concat
- Kleene star

- The are sufficient to represent all regular languages!
- I.e., they define regular expressions
Regular Expressions: Formal Definition

A regular expression \( R \) is a regular expression if \( R \) is

1. \( a \) for some \( a \) in the alphabet \( \Sigma \),
2. \( \varepsilon \),
3. \( \emptyset \),
4. \( (R_1 \cup R_2) \), where \( R_1 \) and \( R_2 \) are regular expressions,
5. \( (R_1 \circ R_2) \), where \( R_1 \) and \( R_2 \) are regular expressions, or
6. \( (R_1^*) \), where \( R_1 \) is a regular expression.

This is a recursive definition.
Recursive Definitions

Recursive definitions have:
- base case and
- recursive case (with a “smaller” object)

```c
/* Linked list Node*/

class Node {
  int data;
  Node next;
}
```

This is a recursive definition: Node used before it’s defined (but must be “smaller”)
Regular Expressions: Formal Definition

A regular expression \( R \) is a regular expression if \( R \) is:

1. \( a \) for some \( a \) in the alphabet \( \Sigma \), (A lang containing a) length-1 string
2. \( \varepsilon \), (A lang containing) the empty string
3. \( \emptyset \), The empty set (i.e., a lang containing no strings)
4. \( R_1 \cup R_2 \), where \( R_1 \) and \( R_2 \) are regular expressions,
5. \( R_1 \circ R_2 \), where \( R_1 \) and \( R_2 \) are regular expressions, or
6. \( R_1^* \), where \( R_1 \) is a regular expression.

3 Base Cases

union

concat

star

3 Recursive Cases
Regular Expression: Concrete Example

- **Operator Precedence:**
  - Parentheses
  - Kleene Star
  - **Concat** (sometimes \(\circ\), sometimes implicit)
  - Union

**Entire regular expr:** language whose strings come from these languages concat'ed (implicit) together

- the language \{"0","1"\}
- \((0 \cup 1)0^*\)
- the language \{"","0","00",...\}
- the language \{"0"\}
- the language \{"1"\}

---

*R is a regular expression* if *R* is:
1. \(a\) for some \(a\) in the alphabet \(\Sigma\),
2. \(\varepsilon\),
3. \(\emptyset\),
4. \((R_1 \cup R_2)\), where \(R_1\) and \(R_2\) are regular expressions,
5. \((R_1 \circ R_2)\), where \(R_1\) and \(R_2\) are regular expressions, or
6. \((R_1^*)\), where \(R_1\) is a regular expression.
Regular Expressions = Regular Langs?

A regular expression is a pattern used to check if a string is present in some text. Here are the rules:

1. a for some a in the alphabet Σ,
2. ε,
3. ∅,
4. (R₁ ∪ R₂), where R₁ and R₂ are regular expressions,
5. (R₁ ⊕ R₂), where R₁ and R₂ are regular expressions, or
6. (R₁*), where R₁ is a regular expression.

Base cases + union, concat, and Kleene star can express any regular language!

(But we have to prove it)
**Thm:** A Lang is Regular $\iff$ Some Reg Expr Describes It

$\Rightarrow$ If a language is regular, it is described by a reg expression

$\Leftarrow$ If a language is described by a reg expression, it is regular

(Easier)

- To prove this part: convert reg expr $\rightarrow$ equivalent NFA!
- (Hint: we mostly did this already when discussing closed ops)

How to show that a language is regular?

Construct a DFA or NFA!
$R$ is a regular expression if $R$ is
1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\varepsilon$,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions,
6. $(R_1^*)$, where $R_1$ is a regular expression.

$N$ is a non-deterministic finite automaton (NFA) for the regular expression $R$.
**Thm**: A Lang is Regular **iff** Some Reg Expr Describes It

⇒ If a language is regular, it is described by a reg expression
   (Harder)
   • To prove this part: Convert an DFA or NFA → equivalent Regular Expression
   • To do so, we first need another kind of finite automata: a **GNFA**

⇐ If a language is described by a reg expression, it is regular
   (Easier)
   • Convert the regular expression → an equivalent NFA!
Generalized NFAs (GNFAs)

A regular NFA is a GNFA with only single character regular expr transitions

Goal: convert GNFAs to Regular Exprs

• GNFA = NFA with regular expression transitions
GNFA→RegExp function

On GNFA input $G$:
- If $G$ has 2 states, return the regular expression transition, e.g.:

$$q_i \xrightarrow{(R_1) (R_2)^* (R_3) \cup (R_4)} q_j$$

Could there be less than 2 states?

Equivalent regular expression

 GNFA
GNFA→RegExp Preprocessing

• First, modify input machine to have:
  • New start state:
    • No incoming transitions
    • $\epsilon$ transition to old start state
  • New, single accept state:
    • With $\epsilon$ transitions from old accept states
**GNFA→RegExp function (recursive)**

On GNFA input $G$:

- **Base Case**: If $G$ has 2 states, return the regular expression transition, e.g.:

\[
q_i \rightarrow (R_1) (R_2)^* (R_3) \cup (R_4) \rightarrow q_j
\]

- **Recursive Case**: Else:
  - “Rip out” one state
  - “Repair” the machine to get an **equivalent** GNFA $G'$
  - **Recursively** call GNFA→RegExp($G'$)

Recursive definitions have:
- **base case** and
- **recursive case** (with a “smaller” object)
GNFA→RegExpr: “Rip/Repair” step

To convert a GNFA to a regular expression:
“rip out” state, then “repair”,
and repeat until only 2 states remain
**GNFA→RegExp: “Rip/Repair” step**

**Before:** two paths from $q_i$ to $q_j$:
1. Not through $q_{rip}$
2. Through $q_{rip}$

**After:**

$$q_i \xrightarrow{(R_1)(R_2)^* (R_3) \cup (R_4)} q_j$$
GNFA→RegExp: “Rip/Repair” step

After: still two “paths” from $q_i$ to $q_j$
1. Not through $q_{rip}$
2. Through $q_{rip}$

$$(R_1)(R_2)^* (R_3) \cup (R_4)$$
GNFA$\rightarrow$RegExpr: “Rip/Repair” step

Before:
- path through $q_{\text{rip}}$ has 3 transitions
- One is self loop
GNFA $\rightarrow$ RegExpr: "Rip/Repair" step

**Before:**
- path through $q_{rip}$ has 3 transitions
- One is self loop

**After:**
- Self loop becomes star operation
- Others are concat’ed together

\[ (R_1)(R_2)^* (R_3) \cup (R_4) \]
GNFA→\texttt{RegExpr}: Rip/Repair “Correctness”

before

Must show these are equivalent

after
GNFA→RegExpr “Correctness”

• “Correct” / “Equivalent” means:

\[ \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA→RegExpr}(G)) \]

• i.e., GNFA→RegExpr must not change the language!
  • Key step: the rip/repair step
**GNFA→RegExp: Rip/Repair “Correctness”**

Must show these are equivalent

before

\[ \begin{aligned} R_1 & \quad q_{\text{rip}} \\ \cdots & \quad q_i \quad R_4 \\ \cdots & \quad q_j \quad (R_1) (R_2)^* (R_3) \cup (R_4) \quad q_j \end{aligned} \]

after

**Must prove:**
- Every string accepted before, is accepted after
- 2 cases:
  1. Accepted string does not go through \( q_{\text{rip}} \)
     - Acceptance unchanged (both use \( R_4 \) transition part)
  2. String goes through \( q_{\text{rip}} \)
     - Acceptance unchanged?
     - Yes, via our previous reasoning
**Thm:** A Lang is Regular **iff** Some Reg Expr Describes It

⇒ If a language is regular, it is described by a regular expr
   Need to convert DFA or NFA to Regular Expression ...
   • Use GNFA→RegExpr to convert GNFA → equiv regular expression!

⇐ If a language is described by a regular expr, it is regular
   • Convert regular expression → equiv NFA!

Now we may use regular expressions to represent regular langs.

So we also have another way to prove things about regular languages!

So a regular language has these equivalent representations:
- DFA
- NFA
- Regular Expression
How to Prove A Language Is Regular?

• Construct DFA

• Construct NFA

• Create Regular Expression

Slightly different because of recursive definition

$R$ is a regular expression if $R$ is
1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\varepsilon$,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions, or
6. $(R_1^*)$, where $R_1$ is a regular expression.
Kinds of Mathematical Proof

• Proof by construction

• Proof by induction
  • Use this when working with recursive definitions
In-Class quiz 9/29

See gradescope