Regular Expressions

Wednesday, February 9, 2022
Announcements

• HW 2 due Sunday 2/13 11:59pm EST
So Far: Regular Language Representations

1. State diagram (NFA/DFA)
   - 1. $Q = \{q_1, q_2, q_3\}$,
   - 2. $\Sigma = \{0, 1\}$,
   - 3. $\delta$ is described as

2. Formal description
   - 4. $q_1$ is the start state, and
   - 5. $F = \{q_2\}$.

3. $\Sigma^*001\Sigma^*$

A practical application: text search

These define a computer (program) that finds strings containing $001$

Need a more concise (textual) notation
Regular Expressions Are Widely Used

- Unix
- Perl
- Python
- Java
Last Time: Why Do We Care These Ops Are Closed?

- Union
- Concat
- Kleene star

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

A regular expression is a regular expression if it 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.
Recursive Definitions

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

```java
/* Linked list Node*/
class Node {
    int data;
    Node next;
}
```

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

A node followed by a list

Left sub-tree is a binary tree

Right sub-tree is a binary tree
Regular Expressions: Formal Definition

A regular expression $R$ 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, 3 Recursive Cases
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 Expression: Concrete Example

*Entire regular expr:* language whose strings come from these languages concatenated (implicit op) together

- the language \{“0”, “1”\}
- (0 ∪ 1)0*
- the language \{“”, “0”, “00”, …\}
- the language \{“0”\}
- the language \{“1”\}

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

\[ R \text{ is a regular expression if } R \text{ is} \]
\[ 1. a \text{ for some } a \text{ in the alphabet } \Sigma, \]
\[ 2. \varepsilon, \]
\[ 3. \emptyset, \]
\[ 4. (R_1 ∪ R_2), \text{ where } R_1 \text{ and } R_2 \text{ are regular expressions}, \]
\[ 5. (R_1 ∘ R_2), \text{ where } R_1 \text{ and } R_2 \text{ are regular expressions, or} \]
\[ 6. (R_1^*) \text{, where } R_1 \text{ is a regular expression.} \]
Regular Expressions = Regular Langs?

A regular expression is defined as follows:

1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\epsilon$,
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.

Three 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

⇒ If a language is regular, it is described by a reg expression

⇐ If a language is described by a reg expression, it is regular
  • Easier
  • For a given regular expression, convert to equiv NFA!
  • (Hint: we mostly did this already when discussing closed ops)
RegExpr → NFA

*R is a regular expression* if *R* is
1. *a* for some *a* in the alphabet Σ,
2. ε,
3. ∅,
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.
**Thm:** A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, it is described by a reg expression
  • Harder
  • Need to convert an DFA or NFA to an equivalent Regular Expression
  • To do so, we 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 to an equivalent NFA!
Generalized NFAs (GNFAs)

- GNFA = NFA with regular expression transitions

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

Goal: convert GNFAs to Regular Exprs
GNFA $\rightarrow$ RegExpr 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?
GNFA→RegExpr Preprocessing

• First modifies 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

  Does this change the language of the machine?
GNFA→RegExp function (recursive)

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

```
qi (R_1) (R_2)^* (R_3) \cup (R_4) qj
```

- 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 $\rightarrow$ RegExpr: “Rip/Repair” step

before

\[ q_i \xrightarrow{R_1} q_{rip} \xrightarrow{R_2} q_j \]

\[ R_3 \]

\[ R_4 \]

after

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

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

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

After: $(R_1)(R_2)^* (R_3) \cup (R_4)$
GDFA→RegExpr: “Rip/Repair” step

Before:
- $q_i \xrightarrow{R_1} q_{rip} \xrightarrow{R_2} q_i$
- $q_i \xrightarrow{R_3} q_{rip} \xrightarrow{R_2} q_i$
- $q_{rip} \xrightarrow{R_2} q_{rip}$

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

$$(R_1)(R_2)^* (R_3) \cup (R_4)$$

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

Before:
- path through $q_{rip}$ has 3 transitions
- One is self loop
GNFA→RegExp: “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→RegExp: Rip/Repair “Correctness”

before

\[ R_1, R_2, R_3, R_4 \]

\[ q_i \rightarrow q_j \]

after

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

Must show these are equivalent
GNFA→RegExpr “Correctness”

• Where “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
\[ R_1 \rightarrow R_2 \rightarrow q_{\text{rip}} \rightarrow R_3 \rightarrow R_4 \]

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

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\( \Rightarrow \text{RegExpr} \) to convert GNFA to equiv regular expression!

⇐ If a language is described by a regular expr, it is regular
✓ • Convert regular expression to 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
Kinds of Mathematical Proof

• Proof by construction

• Proof by induction
  • Use this when working with recursive definitions
Proof by Induction

To prove that a *Statement* is true for a recursively defined thing $x$:
1. Prove *Statement* for the base case of $x$ (usually easy)
2. Prove *Statement* for the inductive (recursive) case of $x$:
   - Assume the induction hypothesis (IH):
     - I.e., *Statement* is true for some “smaller” $x_{\text{smaller}}$
     - E.g., if $x$ is string, then “smaller” = length of string
   - Use IH (and other facts) to prove *Statement* for “larger” $x$
     - Usually involves a case analysis on how to go from $x_{\text{smaller}}$ to $x$

• Why can we assume IH is true???
  - Because we can always start at base case,
  - Then use it to prove for slightly larger case,
  - Then use that to prove for slightly larger case ...
Natural Numbers Are Recursively Defined

A Natural Number is:

• zero
• Or $n + 1$, where $n$ is a Natural Number

This definition is valid because recursive reference is “smaller”

So proving things about Natural Numbers requires induction!
Proof By Induction: Example (Sipser Ch 0)

Prove true: \( P_t = PM^t - Y \left( \frac{M^t - 1}{M - 1} \right) \)

- \( P_t \) = loan balance after \( t \) months
- \( t \) = # months
- \( P \) = principal = original amount of loan
- \( M \) = interest (multiplier)
- \( Y \) = monthly payment
Proof By Induction: Example (Sipser Ch 0)

Prove true: \( P_t = PM^t - Y \left( \frac{M^t - 1}{M - 1} \right) \)

Proof: by induction on natural number \( t \)

Base Case, \( t = 0 \):
- Goal: Show \( P_0 = P \)
- Proof of Goal:
  \[
  P_0 = PM^0 - Y \left( \frac{M^0 - 1}{M - 1} \right) = P
  \]

An inductive proof exactly follows the recursive definition (here, natural numbers) that the induction is “on”

A Natural Number is:
- zero
- Or \( n + 1 \), where \( n \) is a natural number

Plug in \( t = 0 \)

Simplify, to get to goal statement
Proof By Induction: Example (Sipser Ch 0)

Prove true: \( P_t = PM^t - Y \left( \frac{M^t - 1}{M - 1} \right) \)

**Inductive Case:** \( t > 0 \)
- Inductive Hypothesis (IH), assume statement true for some \( t = k \)
  \[ P_k = PM^k - Y \left( \frac{M^k - 1}{M - 1} \right) \]

  - Goal statement to prove, for \( t = k+1 \):
  \[ P_{k+1} = PM^{k+1} - Y \left( \frac{M^{k+1} - 1}{M - 1} \right) \]

  - Proof of Goal:
    \[ P_{k+1} = PM_{k+1} - Y = \left[ PM^k - Y \left( \frac{M^k - 1}{M - 1} \right) \right] M - Y = PM^{k+1} - Y \left( \frac{M^{k+1} - 1}{M - 1} \right) \]

"Connect together" known definitions and statements

Plug in IH

Simplify, to derive goal statement

**An inductive proof exactly follows the recursive definition (here, natural numbers) that the induction is “on”**

A Natural Number is:
- zero
- Or \( n + 1 \), where \( n \) is a natural number

Definition of \( P_{k+1} \)
Homomorphisms

A homomorphism is a function $f : \Sigma \rightarrow \Gamma$ from one alphabet to another.

- Assume $f$ can be used on characters, strings, and languages

- E.g., like a secret decoder!
  - $f(“x”) \rightarrow “c”$
  - $f(“y”) \rightarrow “a”$
  - $f(“z”) \rightarrow “t”$
  - $f(“xyz”) \rightarrow “cat”$
Homomorphisms Closed Under Regular Languages

**Thm:** If a language $A$ is regular, and $f$ is a homomorphism, ...

... then $f(A)$ is a regular language

A *homomorphism* is a function $f : \Sigma \to \Gamma$ from one alphabet to another.
How to Prove A Language Is Regular?

- Construct DFA
- Construct NFA
- Create Regular Expression

\text{Slightly different because of recursive definition}

\begin{itemize}
  \item $R$ is a regular expression if $R$ is
  \begin{enumerate}
    \item $a$ for some $a$ in the alphabet $\Sigma$, \\
    \item $\varepsilon$, \\
    \item $\emptyset$, \\
    \item $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions, \\
    \item $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions, or \\
    \item $(R_1^*)$, where $R_1$ is a regular expression.
  \end{enumerate}
\end{itemize}
Homomorphisms Closed Under Regular Languages

**Thm:** If a language \( A \) is regular, and \( f \) is a homomorphism, ...

- If \( A \) is regular then it has a regular expression \( R \)

... then \( f(A) \) is a regular language

- To show that \( f(A) \) is a regular language, we create a regular expression representing it (using \( R \))

A *homomorphism* is a function \( f : \Sigma \rightarrow \Gamma \) from one alphabet to another.
Homomorphisms Closed Under Regular Languages

**Thm:** If language $A$ is regular, and $f$ is a homomorphism, then $f(A)$ is regular.

**Proof:** By induction on $R$, the regular expression for $A$.

$R$ is a *regular expression* if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\epsilon$,
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.

An inductive proof exactly follows the recursive definition that the induction is “on”.

If $R = a$, then $f(A)$ has a regular expression $f(a)$ and is thus regular.

3 Base Cases

3 Recursive Cases
Homomorphisms Closed Under Regular Languages

**Thm:** If language $A$ is regular, and $f$ is a homomorphism, then $f(A)$ is regular

**Proof:** By induction on $R$, the regular expression for $A$

A regular expression 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.

An inductive proof exactly follows the recursive definition that the induction is “on”

Inductive case # 1: $R = R_1 \cup R_2$

3 Base Cases

3 Recursive Cases
Homomorphisms Closed Under Regular Languages

**Thm:** If language $A$ is regular, and $f$ is a homomorphism, then $f(A)$ is regular

**Proof:** By induction on $R$, the regular expression for $A$

**Inductive Case #1:** $R = R_1 \cup R_2$ where $R_1, R_2$ describe “smaller” reg langs $A_1, A_2$

IH (assume the theorem is true for “smaller” languages)
- If language $A_1$ is regular, then $f(A_1)$ is regular
- If language $A_2$ is regular, then $f(A_2)$ is regular

**Goal:** If language $A_1 \cup A_2$ is regular, then $f(A_1 \cup A_2)$ is regular

**Proof of Goal** (piece together known definitions and statements!)
- $f(A_1 \cup A_2) = f(A_1) \cup f(A_2)$ (because $f$ and $\cup$ don’t affect each other)
- $f(A_1)$ is regular (because of IH)
- $f(A_2)$ is regular (because of IH)
- $f(A_1) \cup f(A_2)$ is regular (because union is closed for regular languages)
In-Class quiz 2/9

See gradescope