UMB CS622
Regular Expressions

Wednesday September 22, 2021

Expressions

- Small Expression: $4.23
- Regular Expression: $6.23
- Large Expression: $6.23
Announcements

• HW1 graded
  • Use gradescope for grade questions / disputes

• HW2 due Sun 9/26 11:59pm EST
HW1 Review: Inductive Proofs

**Must state:**
- Induction on what
  - Often, “length of input string”
  - But not always!
- Base Case
- Inductive Case
  - with inductive hypothesis

Every statement and logical step **must have justification**

Usually taken from:
- Other theorems
- Definitions
- Given assumptions
HW1 Review: Problem 4

Q:
Prove that if some DFA $M = (Q, \Sigma, \delta, q_0, F)$ has a state $q$ such that
$\delta(q, a) = q$, for all $a \in \Sigma$, then $\hat{\delta}(q, w) = q$ for all possible strings $w \in \Sigma^*$.
Use induction on the length of $w$.

A:
Claim. If a DFA has a state $q$ such that $\forall a \in \Sigma \delta(q, a) = q$, then $\forall w \in \Sigma^* \hat{\delta}(q, w) = q$.

Proof. By induction on $w$.

Basis: Trivially, $\hat{\delta}(q, \epsilon) = q$ by the definition of $\hat{\delta}$.

Induction step: Let $w = w'x$ where $x \in \Sigma$, assume the inductive hypothesis $\hat{\delta}(q, w') = q$. The objective is to show $\hat{\delta}(q, w) = q$ using the claim’s precondition $\forall a \delta(q, a) = q$.

$\hat{\delta}(q, w) = \hat{\delta}(q, w'x)$ by substitution of $w = w'x$

$= \delta(\hat{\delta}(q, w'), x)$ by the definition of $\hat{\delta}$

$= \delta(q, x)$ by the inductive hypothesis

$= q$ by the precondition

$\square$
HW1 Review: Problem 3 (part 2)

Prove that the following language is regular:

\[
\{ w \mid w \text{ has exactly two 1s} \}
\]

In other words:

1. Design a DFA that recognizes the language; and
2. give an inductive proof that the DFA does indeed recognize the language.

Assume the language contains strings from alphabet \( \Sigma = \{0, 1\} \)

Q:

A:

Claim. \( \forall w \in \Sigma^* P(w) \), where \( P(w) = w \in L(M) \iff w \in A \).

Proof. By induction on \( w \).

- Basis: \( P(\epsilon) \) holds true as \( \epsilon \notin L(M) \) (the start state \( q_0 \) is not accepting) and \( \epsilon \notin A \) (\( \epsilon \) does not have two 1s).

- Induction step: Let \( w = w'a \) where \( a \in \Sigma \). Assume \( P(w') \), and consider \( P(w) \) throughout the following case analysis.

- If \( w' \) has zero 1s, then \( M \) is in state \( q_0 \):
  - Let \( a = 0 \): \( M \) stays in \( q_0 \) and rejects with zero 1s.
  - Let \( a = 1 \): \( M \) enters \( q_1 \) and rejects with one 1.

- If \( w' \) has one 1, then \( M \) is in state \( q_1 \):
  - Let \( a = 0 \): \( M \) stays in \( q_1 \) and rejects with one 1.
  - Let \( a = 1 \): \( M \) enters \( q_2 \) and accepts with two 1s.

- If \( w' \) has two 1s, then \( M \) is in state \( q_2 \):
  - Let \( a = 0 \): \( M \) stays in \( q_2 \) and accepts with two 1s.
  - Let \( a = 1 \): \( M \) enters \( q_3 \) and rejects with three 1s.

Not strong enough! (needs to say what each state represents)

These need justification (should come from IH)
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
   4. $q_1$ is the start state, and
   5. $F = \{ q_2 \}$.

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

A practical application: text search ... it doesn’t fit!

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

Need a more concise notation
Regular Expressions Are Widely Used

- Perl
- Python
- Java
- Every lang!
Regular Expressions: Formal Definition

A regular expression is a pattern that describes a set of strings.

- **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, union
  5. \((R_1 \circ R_2)\), where \(R_1\) and \(R_2\) are regular expressions, or concat
  6. \((R_1^*)\), where \(R_1\) is a regular expression, star

Base cases plus union, concat, and Kleene star can express any regular language! (But we have to prove it)
Regular Expression: Concrete Example

**Entire reg expr:** represents lang whose strings are strings from these langs concat’ed together (implicit concat op)

- the lang \{“0”, “1”\}
- \((0 \cup 1)0^*\)
- the lang \{“”, “0”, “00”, …\}
- the lang \{“0”\}
- the lang \{“1”\}

**Operator Precedence:**
- Parens
- Star
- Concat (sometimes implicit)
- Union
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
   • Easy!
   • For a given regexp, construct the equiv NFA!
   • (we mostly did it already when discussing closed ops)

How to show that a lang is regular?
Construct DFA or NFA!
A regular expression $R$ is defined as:

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.

The diagrams illustrate the construction of an NFA to recognize $A_1 \circ A_2$. The first diagram shows the transition from the initial state to an accept state upon reading the symbol $a$. The second diagram depicts the construction of $N$ to recognize $A_1 \circ A_2$. The transition arrows and states represent the transitions and states of the NFA, respectively.
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 DFA or NFA to Regular Expression
  • To do so, need new kind of machine: a GNFA

⇐ If a language is described by a reg expression, it is regular
  • Easy!
  • Construct the NFA! (Done)
Generalized NFAs (GNFAs)

- 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.:

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

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

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

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

before

after

$(R_1) (R_2)^* (R_3) \cup (R_4)$
GNFA\!\!→\!RegExpr: \textit{“Rip/Repair”} step

\begin{align*}
\text{Before:} \quad &R_1 \quad R_3 \\
& \downarrow \hspace{0.5cm} \downarrow \\
& q_{\text{rip}}
\end{align*}

\begin{align*}
\text{After: still two “paths” from } q_i \text{ to } q_j \\
1. \text{ Not through } q_{\text{rip}} \\
2. \text{ Through } q_{\text{rip}} \\
\end{align*}

\begin{align*}
& (R_1) (R_2)^* (R_3) \cup (R_4) \\
& \downarrow \\
& q_j
\end{align*}
GNFA $\rightarrow$ RegExpr: “Rip/Repair” step

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

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

\[ q_i \rightarrow q_j \]
GNFA→RegExpr: “Rip/Repair” step

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

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

This “informal” reasoning helps our intuition

Now lets formally prove correctness of GNFA→RegExpr
GNFA→RegExpr “Correctness”

• Where “Correct” means:

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

Use Proof by induction ... on size of G

This is the property we want to prove

• i.e., GNFA→RegExpr must not change the language!
Previously: Recursive (Inductive) Definitions

• Have (at least) two parts:
  • Base case
  • Inductive case
    • Self-reference must be “smaller”

• Example:

**Def: GNFA⇒RegExpr:** input G is a GNFA with n states:
  If \( n = 2 \): return the regular expression on the transition
  Else (G has \( n > 2 \) states):
    • “Rip” out one state and “Repair” to get \( G' \)
    • Recursively Call GNFA⇒RegExpr(\( G' \))

This is exactly the structure of an inductive proof!
GNFA→RegExpr is correct

Def: GNFA→RegExpr: input $G$ is a GNFA with $n$ states:
  If $n = 2$: return the regular expression on the transition
  Else ($G$ has $n > 2$ states):
  “Rip” out one state and “Repair” to get $G'$
  Recursively Call GNFA→RegExpr($G'$)

➢ Proof (by induction on size of $G$):
GNFA→RegExpr is correct

**Def:** GNFA→RegExpr: input $G$ is a GNFA with $n$ states:
- If $n = 2$: return the regular expression on the transition
- Else ($G$ has $n > 2$ states):
  - “Rip” out one state and “Repair” to get $G'$
  - Recursively Call GNFA→RegExpr($G'$)

**Proof** (by induction on size of $G$):
- **Base case:** $G$ has 2 states
  - $\text{LANGOF}(G) = \text{LANGOF}(\text{GNFA→RegExpr}(G))$ is true, by def of GNFA!
GNFA→RegExpr is correct

**Def:** GNFA→RegExpr: input $G$ is a GNFA with $n$ states:
- If $n = 2$: return the regular expression on the transition
- Else ($G$ has $n > 2$ states):
  - “Rip” out one state and “Repair” to get $G'$
  - Recursively Call GNFA→RegExpr($G'$)

**Proof** (by induction on size of $G$):
- **Base case:** $G$ has 2 states
  - $\text{LANGOF}(G) = \text{LANGOF}(\text{GNFA→RegExpr}(G))$ is true!
- **IH:** Assume $\text{LANGOF}(G') = \text{LANGOF}(\text{GNFA→RegExpr}(G'))$ is true!
  - For some $G'$ with $n-1$ states
**GNFA→RegExpr is correct**

**Def:** \( \text{GNFA→RegExpr} \): input \( G \) is a GNFA with \( n \) states:
- If \( n = 2 \): return the regular expression on the transition
- Else (\( G \) has \( n > 2 \) states):
  - “Rip” out one state and “Repair” to get \( G' \)
  - Recursively Call \( \text{GNFA→RegExpr}(G') \)

**Proof** (by induction on size of \( G \)):
- **Base case:** \( G \) has 2 states
  - \( \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA→RegExpr}(G)) \) is true!
- **IH:** Assume \( \text{LANGOF}(G') = \text{LANGOF}(\text{GNFA→RegExpr}(G')) \)
  - For some \( G' \) with \( n-1 \) states
    - **Induction Step:** Prove it’s true for \( G \) with \( n \) states
**GNFA→RegExpr is correct**

**Def:** GNFA→RegExpr: input $G$ is a GNFA with $n$ states:
- If $n = 2$: return the regular expression on the transition
- Else ($G$ has $n > 2$ states):
  - “Rip” out one state and “Repair” to get $G’$
  - Recursively Call GNFA→RegExpr($G’$)

**Proof** (by induction on size of $G$):

- **Base case:** $G$ has 2 states
  - $\text{LANGOF}(G) = \text{LANGOF}(\text{GNFA→RegExpr}(G))$ is true!

- **IH:** Assume $\text{LANGOF}(G’) = \text{LANGOF}(\text{GNFA→RegExpr}(G’))$
  - For some $G’$ with $n-1$ states

- **Induction Step:** Prove it’s true for $G$ with $n$ states
  - After “rip/repair” step, we have exactly a GNFA $G’$ with $n-1$ states
  - And we know $\text{LANGOF}(G’) = \text{LANGOF}(\text{GNFA→RegExpr}(G’))$ from the IH!
GNFA→RegExpr is correct

**Def:** GNFA→RegExpr: input $G$ is a GNFA with $n$ states:
- If $n = 2$: return the regular expression on the transition
- Else ($G$ has $n > 2$ states):
  - “Rip” out one state and “Repair” to get $G'$
  - Recursively Call GNFA→RegExpr($G'$)

**Proof** (by induction on size of $G$):
- **Base case:** $G$ has 2 states
  - $\text{LANGOF}(G) = \text{LANGOF}(\text{GNFA→RegExpr}(G))$ is true!
- **IH:** Assume $\text{LANGOF}(G') = \text{LANGOF}(\text{GNFA→RegExpr}(G'))$
- For some $G'$ with $n-1$ states

**Induction Step:** Prove it’s true for $G$ with $n$ states
- After “rip/repair” step, we have exactly a GNFA $G'$ with $n-1$ states
- And we know $\text{LANGOF}(G') = \text{LANGOF}(\text{GNFA→RegExpr}(G'))$ from the IH!
  - To go from $G$ to $G'$: just need to prove correctness of “rip/repair” step
GNFA $\Rightarrow$ RegExpr: “rip/repair” correctness

Must prove:
- Every string accepted before, is accepted after
- 2 cases:
  - Accepted string does not go through $q_{rip}$
    - Acceptance unchanged (both use $R_4$ transition part)
  - String goes through $q_{rip}$
    - Acceptance unchanged?

Mostly done this already! Just need to state more formally
Thm: A language is regular iff some regular expression describes it.

⇒ If a language is regular, it is described by a regular expression
   • Harder!
   • Need to convert DFA or NFA to regular expression
   • Use GNFA→RegExpr to convert GNFA to regular expression! (Done!)

⇐ If a language is described by a regular expression, it is regular
   • Construct the NFA! (Done)

Now we may use regular expressions to represent regular languages.

I.e., we have another way to prove things about regular languages!

So a regular language has these equivalent representations:
- DFA
- NFA
- Regular Expression
Thm: Reverse is Closed for Regular Langs

• For any string $w = w_1 w_2 \cdots w_n$, the reverse of $w$, written $w^R$, is the string $w$ in reverse order, $w_n \cdots w_2 w_1$. For any language $A$, let $A^R = \{ w^R | w \in A \}$

• Theorem: if $A$ is regular, so is $A^R$

• Proof (by induction on regular expressions):
Thm: Reverse is Closed for Regular Langs

if \( A \) is regular, so is \( A^R \)

Case Analysis, assume some regular language \( A \) is represented with the regular expression ...

1. \( a \) for some \( a \) in the alphabet \( \Sigma \), same reg. expr. represents \( A^R \) so it is regular
2. \( \varepsilon \), same reg. expr. represents \( A^R \) so it is regular
3. \( \emptyset \), same reg. expr. represents \( A^R \) so it is regular
4. \( (R_1 \cup R_2) \), where \( R_1 \) and \( R_2 \) are regular expressions, inductive
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.

Need to show: if \( A_1 \cup A_2 \) is a regular language, then \( (A_1 \cup A_2)^R \) is regular

IH: if \( A_1 \) and \( A_2 \) are the regular languages represented by \( R_1 \) and \( R_2 \), then \( A_1^R \) and \( A_2^R \) are regular too

Proof: \( (A_1 \cup A_2)^R = A_1^R \cup A_2^R \), because reversal and union don’t affect each other and are interchangeable

\( A_1^R \) and \( A_2^R \) are regular (from IH) and union is closed for regular langs (class thm), so \( A_1^R \cup A_2^R \) is regular
In-Class quiz 9/22

See gradescope