Regular Expressions

Wednesday February 28, 2024
Announcements

• HW 3 out
  • Due Mon 3/4 12pm EST (noon)

• Reminder: Use Gradescope re-grade request for all grading questions / complaints!
List of Closed Ops for Reg Langs (so far)

- Union
  \[ A \cup B = \{ x \mid x \in A \text{ or } x \in B \} \]

- Concatentation
  \[ A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \]

- Kleene Star (repetition) ?
Kleene Star Example

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

If $A = \{\text{good, bad}\}$

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

Note: repeat zero or more times

(this is an infinite language!)
Kleene Star is Closed for Regular Langs?

Star: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0$ and each $x_i \in A\}$

Recognizes language $A$

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

Recognizes language $A^*$

Old accept states $\varepsilon$-transition to old start state
Kleene Star is Closed for Regular Langs

**Theorem**

The class of regular languages is closed under the star operation.
Why These (Closed) Operations?

• Union
• Concatenation
• Kleene star (repetition)

\[ A \cup B = \{ x \mid x \in A \text{ or } x \in B \} \]
\[ A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \]
\[ A^* = \{ x_1x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A \} \]

All regular languages can be constructed from:
- (language of) single-char strings (from some alphabet), and
- these three closed operations!
So Far: Regular Language Representations

1. State diagram (NFA/DFA)

   - $q_0$ transitions to $q_0$ on input 0 and to $q_00$ on input 1.
   - $q_00$ transitions to $q_0$ on input 0 and to $q_001$ on input 1.

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
   5. $F = \{q_2\}$

   (doesn’t fit)

3. Our Running Analogy:
   - Set of all regular languages ~ a “programming language”
   - One regular language ~ a “program”

4. $\sum^*001\sum^*$ Need a more concise (textual) notation??

Actually, it’s a real programming language, for text search / string matching computations.
Regular Expressions: A Widely Used Programming Language (in other tools / languages)

- Unix / Linux
- Java
- Python
- Web APIs

About regular expressions (regex)

Analytics supports regular expressions so you can create more flexible definitions for things like view filters, goals, segments, audiences, content groups, and channel groupings.

This article covers regular expressions in both Universal Analytics and Google Analytics 4.

In the context of Analytics, regular expressions are specific sequences of characters that broadly or narrowly match patterns in your Analytics data.

For example, if you wanted to create a view filter to exclude site data generated by your own employees, you could use a regular expression to exclude any data from the entire range of IP addresses that serve your employees. Let’s say those IP addresses range from 198.51.100.1 - 198.51.100.25. Rather than enter 25 different IP addresses, you could create a regular expression like 198\.51\.100\.\d\* that matches the entire range of addresses.

— Regular expression operations

The code: Lib/re.py

The module provides regular expression matching operations similar to those found in Perl.
Why These (Closed) Operations?

• Union

\[ A \cup B = \{ x \mid x \in A \text{ or } x \in B \} \]

• Concatenation

\[ A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \]

• Kleene star (repetition)

\[ A^* = \{ x_1x_2\ldots x_k \mid k \geq 0 \text{ and each } x_i \in A \} \]

All regular languages can be constructed from:
- (language of) single-char strings (from some alphabet), and
- these three closed operations!

They are used to 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.
Flashback: Recursive Definitions

Recursive definitions are definitions with a self-reference

A valid recursive definition must have:
- base case and
- recursive case (with a “smaller” self-reference)
Flashback: Recursive Definitions

```javascript
function factorial( n )
{
  if ( n == 0 )
    return 1;
  else
    return n * factorial( n - 1 );
}
```

- **Base case**: if \( n = 0 \), return 1.
- **Recursive case**: \( n \neq 0 \), return \( n \times \text{factorial}(n - 1) \).
- **Self-reference**: \( n \neq 0 \)
- **Recursive call with “smaller” argument**: \( n \neq 0 \)
Flashback: Recursive Definitions

A Natural Number is either:
- Zero, or
- the Successor of a Natural Number
Flashback: Recursive Definitions

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

Data structures are commonly defined recursively

Q: Where’s the base case??

I call it my billion-dollar mistake. It was the invention of the null reference in 1965.

— Tony Hoare —
Regular Expressions: Formal Definition

A regular expression is 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.

Note:
- A regular expression represents a language.
- The set of all regular expressions represents a set of languages.
Regular Expression: Concrete Example

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

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

- The language $\{0, 1\}$
- $\{(0 \cup 1)^*\}$
- The language $\{\varepsilon, 0, 00, \ldots\}$
- The language $\{0\}$
- The language $\{1\}$

**Definition:**

- A **regular expression** ($R$) is 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 Expression: More Examples

\[ 0^*10^* = \{ w \mid w \text{ contains a single 1} \} \]

\[ \Sigma^*1\Sigma^* = \{ w \mid w \text{ has at least one 1} \} \quad \Sigma \text{ in regular expression = “any char”} \]

\[ 1^*(01^+)^* = \{ w \mid \text{every 0 in } w \text{ is followed by at least one 1} \} \quad \text{let } R^* \text{ be shorthand for } RR^* \]

\[(0 \cup \varepsilon)(1 \cup \varepsilon) = \{ \varepsilon, 0, 1, 01 \} \quad 0 \cup \varepsilon \text{ describes the language } \{0, \varepsilon\}\]

\[1^*\emptyset = \emptyset \quad A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \]

nothing in B = nothing in A \circ B

\[\emptyset^* = \{ \varepsilon \} \quad \text{Star of any lang has } \varepsilon \]

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?

Prove: Any regular language can be constructed from:
- base cases +
- union, concat, Kleene star

3 Base Cases
1. a for some a in the alphabet Σ,
2. ε,
3. ⊘,

3 Recursive Cases
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.

We would like:
- A regular expression represents a regular language
- The set of all regular expressions represents the set of regular languages

(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)
- **Key step:** 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, or
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.
Thm: A Lang is Regular \textbf{iff} Some Reg Expr Describes It

\(\Rightarrow\) If a language is regular, it is described by a reg expression (Harder)
  - Key step: Convert an DFA or NFA \(\rightarrow\) equivalent Regular Expression
  - To do so, we first need another kind of finite automata: a \textbf{GNFA}

\(\Leftarrow\) If a language is described by a reg expression, it is regular (Easier)
  - Key step: Convert the regular expression \(\rightarrow\) an equivalent NFA!

(full proof requires writing Statements and Justifications, and creating an “Equivalence” Table)
Generalized NFAs (GNFAs)

- GNFA = NFA with regular expression transitions

Transition can read multiple chars

A plain NFA = a GNFA with single char regular expr transitions

Goal: convert GNFAs to equivalent Regular Exprs
GNFA→RegExp function

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

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

Could there be less than 2 states?
GNFA→RegExp Preprocessing

• First, modify input machine to have:

  • New start state:
    • No incoming transitions
    • \(\varepsilon\) transition to old start state

  • New, single accept state:
    • With \(\varepsilon\) transitions from old accept states

Modified machine always has 2+ states:
- New start state
- New accept state
GNFA→RegExp function (recursive)

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

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

- 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 “smaller” self-reference)
GNFA→RegExp: “Rip/Repair” step

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 $q_{rip}$
2. Through $q_{rip}$

**After:**

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

before
GNFA→RegExpr: “Rip/Repair” step

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

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

before

after
**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

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

This time, let’s really prove equivalence! (previously, we “proved” it)
GNFA→RegExpr Correctness

• Correct / Equivalent means:

$$\text{LANGOF}(G) = \text{LANGOF}(R)$$

• Where:
  • $G$ = a GNFA
  • $R$ = a Regular Expression
  • $R = \text{GNFA→RegExpr}(G)$

• i.e., $\text{GNFA→RegExpr}$ must not change the language!
  • Key step: the rip/repair step

This time, let’s really prove equivalence! (previously, we “proved” it)
GNFA→RegExp: Rip/Repair Correctness

Before:

- $q_i$ connected to $q_j$ by $R_4$
- $q_i$ connected to $q_{rip}$ by $R_1$
- $q_{rip}$ connected to $q_j$ by $R_3$
- $q_{rip}$ connected to itself by $R_2$

After:

- $q_i$ connected to $q_j$ by $(R_1)(R_2)^* (R_3) \cup (R_4)$

Must show these are equivalent

Equivalent = same language = accepts the same strings
**GNFA→RegExpr: Rip/Repair Correctness**

Must show these are equivalent

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

**Must prove:**
- Every string accepted before, is accepted after.
- 2 cases:
  1. Let \(w_1 = \text{str accepted before, doesn't go through } q_{\text{rip}}\)
     - after still accepts \(w_1\) bc: both use \(R_4\) transition
  2. Let \(w_2 = \text{str accepted before, goes through } q_{\text{rip}}\)
     - \(w_2\) accepted by after?
     - Yes, via our previous reasoning
GNFA→RegExpr “Correctness”

• “Correct” / “Equivalent” means:
  \( \text{LANGOF}(G) = \text{LANGOF}(R) \)

• Where:
  • \( G \) = a GNFA
  • \( R \) = a Regular Expression
  • \( R = \text{GNFA→RegExpr}(G) \)

• i.e., GNFA→RegExpr must not change the language!
  • Key step: the rip/repair step

How to really prove this part?
This time, let’s really prove equivalence! (previously, we “proved” it)
Inductive Proofs