CS 172: Computability and Complexity

Regular Expressions

Sanjit A. Seshia
EECS, UC Berkeley

Acknowledgments: L. von Ahn, L. Blum, M. Blum
The Picture So Far

DFA  ↔  NFA

Regular language
Today’s Lecture

DFA ↔ NFA

Regular language ↔ Regular expression
Regular Expressions

• What is a regular expression?
Regular Expressions

- Q. What is a regular expression?
- A. It’s a “textual”/ “algebraic” representation of a regular language
  - A DFA can be viewed as a “pictorial” / “explicit” representation

- We will prove that a regular expressions (regexps) indeed represent regular languages
Regular Expressions: Definition

\( \sigma \) is a regular expression representing \( \{ \sigma \} \)
\( ( \sigma \in \Sigma ) \)
\( \varepsilon \) is a regular expression representing \( \{ \varepsilon \} \)
\( \emptyset \) is a regular expression representing \( \emptyset \)

If \( R_1 \) and \( R_2 \) are regular expressions representing \( L_1 \) and \( L_2 \) then:

\( (R_1R_2) \) represents \( L_1 \cdot L_2 \)
\( (R_1 \cup R_2) \) represents \( L_1 \cup L_2 \)
\( (R_1)^* \) represents \( L_1^* \)
Operator Precedence

1. $\ast$

2. $\cdot$ (often left out; $a \cdot b \rightarrow ab$)

3. $\cup$
Example of Precedence

\[ R_1 \ast R_2 \cup R_3 = ((R_1 \ast) R_2) \cup R_3 \]
What’s the regexp?

\{ w \mid w \text{ has exactly a single 1} \} \\
0^*10^*
What language does $\emptyset^*$ represent?

$\{\varepsilon\}$
What’s the regexp?

\{ w \mid w \text{ has length } \geq 3 \text{ and its 3rd symbol is } 0 \}

\Sigma^2 \ 0 \ \Sigma^*

\Sigma = (0 \cup 1)
Some Identities

Let $R$, $S$, $T$ be regular expressions

- $R \cup \emptyset = ?$
- $R \cdot \emptyset = ?$
- **Prove:** $R ( S \cup T ) = R S \cup R T$
  
  *(what’s the proof idea?)*
Some Applications of Regular Expressions

• **String matching & searching**
  – Utilities like grep, awk, …
  – Search in editors: emacs, …

• **Programming Languages**
  – Perl
  – Compiler design: lex/yacc

• **Computer Security**
  – Virus signatures
Virus Signature as String

Chernobyl virus code fragment

```
... pop ecx jecxz SFModMark mov esi, ecx mov eax, 0d601h pop edx pop ecx ...
```

Sequence of words, one for each instruction:

- i0
- i1
- i2
- i3
- i4
- i0

virus!
Virus Signature as Regexp

Simple obfuscated Chernobyl virus code fragment

... 
nop
pop ecx
nop
jecxz SFModMark
mov esi, ecx
nop
nop
mov eax, 0d601h
pop edx
pop ecx
...
Equivalence Theorem

A language is regular
↑ if and only if ↓
some regular expression describes it
Part I ("if part")

Some regular expression $R$ describes a language

$\Rightarrow$

That language is regular

There exists NFA $N$ such that $R$ describes $L(N)$
Given regular expression $R$, we show there exists NFA $N$ such that $R$ represents $L(N)$

Proof idea?
Given regular expression $R$, we show there exists NFA $N$ such that $R$ represents $L(N)$

Proof Idea: Induction on the length of $R$:

Base Cases (R has length 1):

- $R = \sigma$
- $R = \varepsilon$
- $R = \emptyset$
Inductive Step:
Assume \( R \) has length \( k > 1 \) and that any regular expression of length < \( k \) represents a language that can be recognized by an NFA

What might \( R \) look like?

\[
R = R_1 \cup R_2
\]
\[
R = R_1 R_2
\]
\[
R = (R_1)^*
\]

(remember: we have NFAs for \( R_1 \) and \( R_2 \))
Part I ("if part")

Some regular expression $R$ describes a language

$\Rightarrow$

That language is regular

There exists NFA $N$ such that $R$ describes $L(N)$

DONE!
An Example

Transform \((1(0 \cup 1))^*\) to an NFA
Part II ("only if part")

A language is regular

⇒

Some regular expression $R$ describes it

Turn DFA into equivalent regular expression
Proof Sketch

1. DFA $\rightarrow$ Generalized NFA
   - NFA with edges labeled by regexps, 1 start state, and 1 accept state
2. GNFA with $k$ states $\rightarrow$ GNFA with 2 states
   - $k > 2$; delete states but maintain equivalence
3. 2-state GNFA $\rightarrow$ regular expression $R$

\[ \text{Diagram:} \quad \rightarrow \quad R \quad \rightarrow \quad \circ \]
A GNFA is a tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\)

- **Q** – set of states
- **\(\Sigma\)** – finite alphabet (not regexps)
- **\(q_{\text{start}}\)** – initial state (unique, no incoming edges)
  - \(\varepsilon\) transitions to old start state
- **\(q_{\text{accept}}\)** – accepting state (unique, no outgoing edges)
  - \(\varepsilon\) transitions from old accept states
- **\(\delta : (Q \setminus q_{\text{accept}}) \times (Q \setminus q_{\text{start}}) \rightarrow R\)**
  - \(R\) – set of all regexps over \(\Sigma\).

Example: Any string matching \(01^*0\) can cause the transition.
Step 1: DFA to GNFA

What’s the corresponding GNFA?
Step 1: DFA to GNFA

Add unique and distinct start and accept states

Edges with multiple labels → regexp labels

If internal states \((q_1, q_2)\) don’t have an edge between them, add one labeled with \(\emptyset\)
Step 2: Eliminate states from GNFA

While machine has more than 2 states:

Pick an internal state, rip it out and re-label the arrows with regular expressions to account for the missing state
Step 2: Eliminate states from GNFA

While machine has more than 2 states:

Pick an internal state, rip it out and re-label the arrows with regular expressions to account for the missing state.
\(q_0 \xrightarrow{(a^*b)(a \cup b)^*} q_3\)

\[\delta(q_0, q_3) = (a^*b)(a \cup b)^*\]
Formally: Add $q_{\text{start}}$ and $q_{\text{accept}}$ and create GNFA $G$
Run $\text{CONVERT}(G)$ to eliminate states & get regexp:

If $\#\text{states} = 2$

return the expression on the arrow going from $q_{\text{start}}$ to $q_{\text{accept}}$

If $\#\text{states} > 2$

?
Formally: Add $q_{\text{start}}$ and $q_{\text{accept}}$ to create $G$

Run \textsc{CONVERT}(G):

If $\#\text{states} > 2$

select $q_{\text{rip}} \in Q$ different from $q_{\text{start}}$ and $q_{\text{accept}}$

define $Q' = Q - \{q_{\text{rip}}\}$

define $\delta'$ as:

$\delta'(q_i, q_j) = \delta(q_i, q_{\text{rip}})\delta(q_{\text{rip}}, q_{\text{rip}})\delta(q_{\text{rip}}, q_j) \cup \delta(q_i, q_j)$

return \textsc{CONVERT}(G')  /* recursion */

(what does this look like, pictorially?)
Prove: CONVERT(G) is equivalent to G
Proof *by induction on k* (number of states in G)

**Base Case:**

✓ \( k = 2 \)

**Inductive Step:**
Assume claim is true for \( k-1 \) states

Prove that G and \( G' \) are equivalent

By the induction hypothesis, \( G' \) is equivalent to CONVERT(G')
The Complete Picture

DFA ↔ NFA

Regular language ↔ Regular expression
Which language is regular?

\[ C = \{ w \mid w \text{ has equal number of } 1\text{s and } 0\text{s}\} \]

NOT REGULAR

\[ D = \{ w \mid w \text{ has equal number of occurrences of } 01 \text{ and } 10\} \]

REGULAR!
Next Steps

• Read Sipser 1.4 in preparation for next lecture