CS21
Decidability and Tractability

Lecture 4
January 11, 2021
Outline

• Regular Expressions
• FA and Regular Expressions
• Non-regular languages: Pumping Lemma
Regular expressions

• $R$ is a regular expression if $R$ is
  – $a$, for some $a \in \Sigma$
  – $\varepsilon$, the empty string
  – $\emptyset$, the empty set
  – $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are reg. exprs.
  – $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are reg. exprs.
  – $(R_1^*)$, where $R_1$ is a regular expression

A reg. expression $R$ describes the language $L(R)$. 
Regular expressions

• example: $R = (0 \cup 1)$
  – if $\Sigma = \{0,1\}$ then use “$\Sigma$” as shorthand for $R$

• example: $R = 0 \circ \Sigma^*$
  – shorthand: omit “$\circ$”  $R = 0\Sigma^*$
  – precedence: $\ast$, then $\circ$ then $\cup$, unless override by parentheses
  – in example $R = 0(\Sigma^*)$, not $R = (0\Sigma)^*$
Some examples

• \{w : w has at least one 1\}
  \[= \Sigma^*1\Sigma^*\]

• \{w : w starts and ends with same symbol\}
  \[= 0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1\]

• \{w : |w| \leq 5\}
  \[= (\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)(\varepsilon \cup \Sigma)\]

• \{w : every 3^{rd} position of w is 1\}
  \[= (1\Sigma\Sigma)^*(\varepsilon \cup 1 \cup 1\Sigma)\]

alphabet
\[\Sigma = \{0, 1\}\]
Manipulating regular expressions

• The empty set and the empty string:
  – $R \cup \emptyset = R$
  – $R\varepsilon = \varepsilon R = R$
  – $R\emptyset = \emptyset R = \emptyset$
  – $\cup$ and $\circ$ behave like $+$, $\times$; $\emptyset$, $\varepsilon$ behave like 0, 1

• additional identities:
  – $R \cup R = R$  (here $+$ and $\cup$ differ)
  – $(R_1 \ast R_2) \ast R_1 \ast = (R_1 \cup R_2) \ast$
  – $R_1 (R_2 R_1) \ast = (R_1 R_2) \ast R_1$
Regular expressions and FA

- **Theorem**: a language $L$ is recognized by a FA if and only if $L$ is described by a regular expression.

Must prove *two* directions:

$(\Rightarrow)$ $L$ is recognized by a FA implies $L$ is described by a regular expression

$(\Leftarrow)$ $L$ is described by a regular expression implies $L$ is recognized by a FA.
Regular expressions and FA

$(\Leftarrow)$ L is described by a regular expression implies L is recognized by a FA

**Proof**: given regular expression $R$ we will build a NFA that recognizes $L(R)$.

then NFA, FA equivalence implies a FA for $L(R)$. 

Regular expressions and FA

• $R$ is a regular expression if $R$ is
  
  – $a$, for some $a \in \Sigma$
  
  – $\varepsilon$, the empty string
  
  – $\emptyset$, the empty set
Regular expressions and FA

- \((R_1 \cup R_2)\), where \(R_1\) and \(R_2\) are reg. exprs.

- \((R_1 \circ R_2)\), where \(R_1\) and \(R_2\) are reg. exprs.

- \((R_1^*)\), where \(R_1\) is a regular expression
Regular expressions and FA

$(\Rightarrow)$ L is recognized by a FA implies L is described by a regular expression

**Proof**: given FA $M$ that recognizes L, we will

1. build an equivalent machine “Generalized Nondeterministic Finite Automaton” (GNFA)
2. convert the GNFA into a regular expression
Regular expressions and FA

• GNFA definition:
  – it is a NFA, but may have regular expressions labeling its transitions
  – GNFA accepts string \( w \in \Sigma^* \) if can be written
    \[ w = w_1w_2w_3\ldots w_k \]
    where each \( w_i \in \Sigma^* \), and there is a path from the start state to an accept state in which the \( i^{th} \)
    transition traversed is labeled with \( R \) for which \( w_i \in L(R) \)
Regular expressions and FA

• Recall step 1: build an equivalent GNFA

• Our FA $M$ is a GNFA.

• We will require “normal form” for GNFA to make the proof easier:
  – *single* accept state $q_{\text{accept}}$ that has all possible incoming arrows
  – every state has all possible outgoing arrows; exception: start state $q_0$ has no self-loop
Regular expressions and FA

- converting our FA $M$ into GNFA in normal form:
Regular expressions and FA

• On to step 2: convert the GNFA into a regular expression

  – if normal-form GNFA has two states:

    the regular expression $R$ labeling the single transition describes the language recognized by the GNFA
Regular expressions and FA

– if GNFA has more than 2 states:

– select one “q<sub>rip</sub>”; delete it; repair transitions so that machine still recognizes same language.

– repeat until only 2 states.
Regular expressions and FA

– how to repair the transitions:
– for every pair of states \(q_i\) and \(q_j\) do
Regular expressions and FA

– summary:
FA $M \rightarrow k$-state GNFA $\rightarrow (k-1)$-state GNFA $\rightarrow (k-2)$-state GNFA $\rightarrow \ldots \rightarrow 2$-state GNFA $\rightarrow R$

– want to prove that this procedure is correct, i.e. $L(R) = \text{language recognized by } M$

- $\text{FA } M \text{ equivalent to } k$-state GNFA
- $i$-state GNFA equivalent to $(i-1)$-state GNFA (we will prove…)
- $2$-state GFNA equivalent to $R$
Regular expressions and FA

- **Claim**: i-state GNFA G equivalent to (i-1)-state GNFA G’ (obtained by removing $q_{\text{rip}}$)

- **Proof**:
  - if G accepts string w, then it does so by entering states: $q_0, q_1, q_2, q_3, \ldots, q_{\text{accept}}$
  - if none are $q_{\text{rip}}$ then G’ accepts w (see slide)
  - else, break state sequence into runs of $q_{\text{rip}}$: $q_0q_1\ldots q_i q_{\text{rip}} q_{\text{rip}} \ldots q_{\text{rip}} q_j \ldots q_{\text{accept}}$
  - transition from $q_i$ to $q_j$ in G’ allows all strings taking G from $q_i$ to $q_j$ using $q_{\text{rip}}$ (see slide)
  - thus G’ accepts w
Regular expressions and FA

\[ (R_1)(R_2)^*(R_3) \cup (R_4) \]
Regular expressions and FA

\[(R_1)(R_2)^*(R_3) \cup (R_4)\]
Regular expressions and FA

– **Proof** (continued):

  • if $G'$ accepts string $w$, then every transition from $q_i$ to $q_j$ traversed in $G'$ corresponds to
    
    either
    
    a transition from $q_i$ to $q_j$ in $G$
    or
    
    transitions from $q_i$ to $q_j$ via $q_{rip}$ in $G$

  • In both cases $G$ accepts $w$.

  • Conclude: $G$ and $G'$ recognize the same language.
Regular expressions and FA

• **Theorem**: a language \( L \) is recognized by a FA iff \( L \) is described by a regular expr.

• Languages recognized by a FA are called **regular languages**.

• Rephrasings what we know so far:
  – regular languages closed under 3 operations
  – NFA recognize exactly the **regular languages**
  – regular expressions describe exactly the **regular languages**
Limits on the power of FA

• Is every language describable by a sufficiently complex regular expression?
• If someone asks you to design a FA for a language that seems hard, how do you know when to give up?

• Is this language regular?
  \{w : w has an equal # of “01” and “10” substrings\}
Limits on the power of FA

• Intuition:
  – FA can only remember finite amount of information. They cannot count
  – languages that “entail counting” should be non-regular…

• Intuition not enough:

\{w : w has an equal # of “01” and “10” substrings\}

= \Sigma^*0 \cup \Sigma^*1

but \{w : w has an equal # of “0” and “1” substrings\} is not regular!
Limits on the power of FA

How do you *prove* that there is *no* Finite Automaton recognizing a given language?