CS21
Decidability and Tractability
Lecture 4
January 10, 2022

Outline
• Regular Expressions
• FA and Regular Expressions
• Non-regular languages: Pumping Lemma
• Pushdown Automata
• Context-Free Grammars and Languages

Regular expressions
• R is a regular expression if R is
  – a, for some a ∈ Σ
  – Ø, the empty set
  – (R₁ ∪ R₂), where R₁ and R₂ are reg. exprs.
  – (R₁ * R₂), where R₁ and R₂ are reg. exprs.
  – (R₁ *), where R₁ is a regular expression
A reg. expression R describes the language L(R).

Some examples
alphabet Σ = \{0, 1\}
• \{w : w has at least one 1\}
  = Σ*1Σ*
• \{w : w starts and ends with same symbol\}
  = 0Σ*0 ∪ 1Σ*1 ∪ 0 ∪ 1
• \{w : |w| ≤ 5\}
  = (ε ∪ Σ)(ε ∪ Σ)(ε ∪ Σ)(ε ∪ Σ)(ε ∪ Σ)
• \{w : every 3rd position of w is 1\}
  = (1ΣΣ*)(ε ∪ 1 ∪ 1Σ)

Manipulating regular expressions
• The empty set and the empty string:
  – R ∪ Ø = R
  – Rε = εR = R
  – RØ = ØR = Ø
  – ∪ and * behave like +, ∗; Ø, ε behave like 0, 1
• additional identities:
  – R ∪ R = R
  – (R₁ ∗ R₂) ∗ R₃ ∗ = (R₁ ∪ R₂) ∗
  – R₁(R₂ ∗ R₃) ∗ = (R₁ ∪ R₂) ∗ R₃

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:
(⇒) L is recognized by a FA implies L is described by a regular expression
(⇐) L is described by a regular expression implies L is recognized by a FA.
Regular expressions and FA

($)\Rightarrow$ 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 ∈ Σ
  – ε, the empty string
  – Ø, the empty set

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 ∈ Σ* if can be written
    \[ w = w_1 w_2 w_3 \ldots w_k \]
  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:

\[
\begin{align*}
\Delta : &
\begin{cases}
\delta(q_0, a) = q_1 & 	ext{for } a = 0, 1 \\
\delta(q_0, \epsilon) = q_0 & 	ext{for } a = \epsilon
\end{cases}
\end{align*}
\]

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

– if normal-form GNFA has two states:

\[
\begin{align*}
&\epsilon \\
&\epsilon \\
&\epsilon \\
&\epsilon
\end{align*}
\]

the regular expression \( R \) labeling the single transition describes the language recognized by the GNFA

– if GNFA has more than 2 states:

– select one \( q_{\text{rip}} \); delete it; repair transitions so that machine still recognizes same language.
– repeat until only 2 states.

– how to repair the transitions:

– for every pair of states \( q_i \) and \( q_j \) do

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

– 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. \( L(R) = \text{language recognized by } M \)

• FA \( M \) equivalent to \( k \)-state GNFA

• i-state GNFA equivalent to \( (i-1) \)-state GNFA
(we will prove…)

• 2-state GFNA equivalent to \( R \)

– 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: \( q_0, q_1, q_2, \ldots, q_\text{accept} \)
• if none are \( q_{\text{rip}} \), then \( G' \) accepts \( w \) (see slide)
• else, break state sequence into runs of \( q_{\text{rip}} \): \( q_0q_1\ldots q_{\text{rip}}q_2\ldots q_{\text{rip}}q_3\ldots q_{\text{accept}} \)
• 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

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

\[ R_1 \]
\[ R_2 \]
\[ R_3 \]
\[ R_4 \]

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.

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.

Rephrasing 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

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