CS21
Decidability and Tractability

Lecture 3

January 10, 2014

Outline

- Regular Expressions
- FA and Regular Expressions
- Pumping Lemma

Next…

- Describe the set of languages that can be built up from:
  - unions
  - concatenations
  - star operations
- Called "patterns" or regular expressions
- **Theorem**: a language L is recognized by a FA if and only if L is described by a regular expression.

Regular expressions

- R is a regular expression if R is
  - a, for some $a \in \Sigma$
  - ε, the empty string
  - Ø, 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 "Σ" as shorthand for R
- example: $R = 0 \circ \Sigma^*$
  - shorthand: omit " ∘ " $R = 0\Sigma^*$
  - precedence: *, then ∘, then ∪, unless override by parentheses
  - in example $R = 0(\Sigma^*)$, not $R = (0\Sigma)^*$

Some examples

alphabet $\Sigma = \{0,1\}$

- {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 3rd position of w is 1}
  $= (1\Sigma\Sigma)^*(\varepsilon \cup 1 \cup 1\Sigma)$

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 +, x; $\emptyset$, ε behave like 0,1
- additional identities:
  - $R \cup R = R$ (here + and $\cup$ differ)
  - $(R_1{}^*R_2)^*R_1{}^* = (R_1 \cup R_2)^*$
  - $R_1(R_2R_1)^* = (R_1R_2)^*R_1$

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

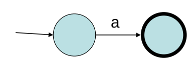
($\Leftarrow$) L is described by a regular expression implies L is recognized by a FA

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

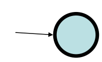
then NFA, FA equivalence implies a FA for L(R).

Regular expressions and FA

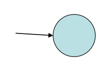
- R is a regular expression if R is
  - a, for some $a \in \Sigma$



  - ε, the empty string

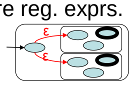


  - $\emptyset$, the empty set

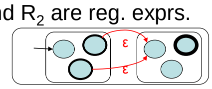


Regular expressions and FA

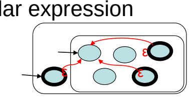
- $(R_1 \cup R_2)$, where R_1 and R_2 are reg. exprs.



- $(R_1 \circ R_2)$, 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 $w_i \in L(R)$

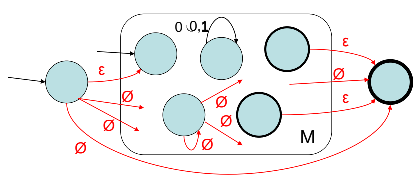
Regular expressions and FA

- 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_{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

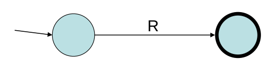
- converting our FA M into GNFA in normal form:



Regular expressions and FA

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

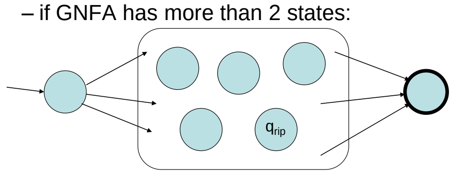
  – if normal-form GNFA has two states:



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

Regular expressions and FA

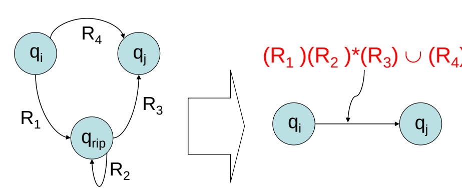
– if GNFA has more than 2 states:



– select one "q_{rip}"; 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 → k-state GNFA → (k-1)-state GNFA → (k-2)-state GNFA →…→ 2-state GNFA → R

– want to *prove* that this procedure is correct, i.e. L(R) = 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

Regular expressions and FA

– **Claim**: i-state GNFA G equivalent to (i-1)-state GNFA G' (obtained by removing q_{rip})

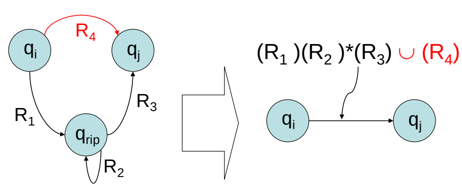
– **Proof**:

- if G accepts string w, then it does so by entering states: $q_0, q_1, q_2, q_3, \ldots, q_{accept}$
- if none are q_{rip} then G' accepts w (see slide)
- else, break state sequence into runs of q_{rip}:

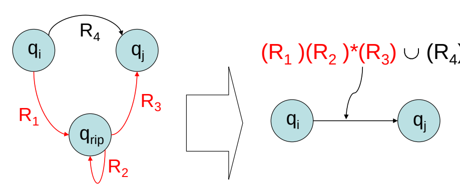
$$q_0q_1\ldots q_iq_{rip}q_{rip}\ldots q_{rip}q_j\ldots q_{accept}$$

- transition from q_i to q_j in G' allows all strings taking G from q_i to q_j using q_{rip} (see slide)
- thus G' accepts w

Regular expressions and FA



Regular expressions and FA



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.
- 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

- 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}

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

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?

Non-regular languages

Pumping Lemma: Let L be a regular language. There exists an integer p ("pumping length") for which every $w \in L$ with $|w| \geq p$ can be written as

$w = xyz$ such that

1. for every $i \geq 0$, $xy^iz \in L$, and
2. $|y| > 0$, and
3. $|xy| \leq p$.

Non-regular languages

- Using the Pumping Lemma to prove L is not regular:
  - assume L is regular
  - then there exists a pumping length p
  - select a string $w \in L$ of length at least p
  - argue that for every way of writing w = xyz that satisfies (2) and (3) of the Lemma, pumping on y yields a string not in L.
  - contradiction.

Pumping Lemma Examples

- Theorem: $L = \{0^n1^n : n \geq 0\}$ is not regular.
- Proof:
  - let p be the pumping length for L
  - choose $w = 0^p1^p$

$$w = \underbrace{000000000\ldots0}_{p}\underbrace{111111111\ldots1}_{p}$$

  - $w = xyz$, with $|y| > 0$ and $|xy| \leq p$.

Pumping Lemma Examples

– 3 possibilities:

$$w = \underbrace{00000}_{x}\underbrace{0000}_{y}\underbrace{0\ldots0111111111\ldots1}_{z}$$

$$w = \underbrace{000000000\ldots011}_{x}\underbrace{11111}_{y}\underbrace{11\ldots1}_{z}$$

$$w = \underbrace{0000000}_{x}\underbrace{00\ldots011111}_{y}\underbrace{1111\ldots1}_{z}$$

– in each case, pumping on y gives a string not in language L.

Pumping Lemma Examples

- Theorem: L = {w: w has an equal # of 0s and 1s} is not regular.
- Proof:
  - let p be the pumping length for L
  - choose $w = 0^p1^p$

$$w = \underbrace{000000000\ldots0}_{p}\underbrace{111111111\ldots1}_{p}$$

  - $w = xyz$, with $|y| > 0$ and $|xy| \le p$.

Pumping Lemma Examples

– 3 possibilities:

$$w = \underbrace{00000}_{x}\underbrace{0000}_{y}\underbrace{0\ldots0111111111\ldots1}_{z}$$

$$w = \underbrace{000000000\ldots011}_{x}\underbrace{11111}_{y}\underbrace{11\ldots1}_{z}$$

$$w = \underbrace{000000000\ldots}_{x}\underbrace{01}_{y}\underbrace{111111111\ldots1}_{z}$$

– first 2 cases, pumping on y gives a string not in language L; 3rd case a problem!

Pumping Lemma Examples

– recall condition 3: $|xy| \le p$

– since $w = 0^p1^p$ we know more about how it can be divided, and this case cannot arise:

$$w = \underbrace{000000000\ldots}_{x}\underbrace{01}_{y}\underbrace{111111111\ldots1}_{z}$$

– so we do get a contradiction.

– conclude that L is not regular.