EECS 2001A : Introduction to the Theory of Computation

Suprakash Datta

Course page: `http://www.eecs.yorku.ca/course/2001`
Also on Moodle

Another Characterization of Regular Languages

Regular Expressions

- Unix ‘grep’ command: Global Regular Expression and Print
- Lexical Analyzer Generators (part of compilers)
- Other practical uses in software design
- Will see some examples and then formulate a precise definition
- Finally will obtain another characterization of regular languages!

Examples of Regular Expressions

- $e_1 = a \cup b,\ L(e_1) = \{a, b\}$
- $e_2 = ab \cup ba,\ L(e_2) = \{ab, ba\}$
- $e_3 = a^*,\ L(e_3) = \{a\}^*$
- $e_4 = (a \cup b)^*,\ L(e_4) = \{a, b\}^*$
- $e_5 = (e_m \cdot e_n),\ L(e_5) = L(e_m) \cdot L(e_n)$
- $e_6 = a^* b \cup a^* bb,\ L(e_6) = \{w | w \in \{a, b\}^*$ and w has 0 or more a's followed by 1 or 2 b's$\}$

Regular Expressions: Recursive Definition

Regular Expressions (RE)

- $R = a$, with $a \in \Sigma$: $R \in \mathrm{RE}$
- $R = \epsilon$ (empty expression): $R \in \mathrm{RE}$
- $R = \emptyset$: $R \in \mathrm{RE}$
- $R = (R_1 \cup R_2)$, where $R_1, R_2 \in \mathrm{RE}$: $R \in \mathrm{RE}$
- $R = (R_1 \cdot R_2)$, where $R_1, R_2 \in \mathrm{RE}$: $R \in \mathrm{RE}$
- $R = (R_1^*)$, where $R_1 \in \mathrm{RE}$: $R \in \mathrm{RE}$

Precedence order: $*, \cdot, \cup$

Regular Expressions: Identities

- $R_1\emptyset = \emptyset R_1 = \emptyset$
- $R_1\epsilon = \epsilon R_1 = R_1$
- $R_1 \cup \emptyset = \emptyset \cup R_1 = R_1$
- $R_1 \cup R_1 = R_1$
- $R_1 \cup R_2 = R_2 \cup R_1$
- $R_1(R_2 \cup R_3) = R_1R_2 \cup R_1R_3$
- $(R_1 \cup R_2)R_3 = R_1R_3 \cup R_2R_3$
- $R_1(R_2R_3) = (R_1R_2)R_3$
- $\emptyset^* = \epsilon$
- $\epsilon^* = \epsilon$
- $(\epsilon \cup R_1)^* = R_1^*$

Regular Expressions: The Big Result

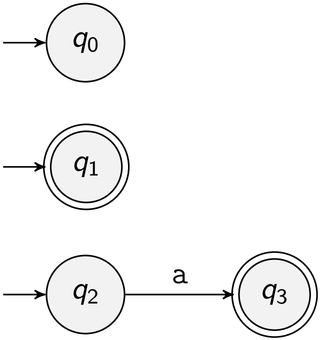
Regular expressions (RE) and Regular Languages are the same set

- Theorem 1.54 Let L be a language. Then L is regular if and only if there exists a regular expression that describes L
- Part 1: If a language is described by a regular expression, then it is regular (We will show how to convert a regular expression R into an NFA M such that $L(R) = L(M)$)
- Part 2: If a language is regular, then it can be described by a regular expression

Part 1: RE to RL (NFA construction)

Construction: Use recursive definition

- $R = \emptyset$, $R = \epsilon$
- $R = a$, with $a \in \Sigma$
- $R = (R_1 \cup R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1 \cdot R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1^*)$, with R_1 a regular expression



Last 3 are similar to closure of RL under union, concatenation, star

RE to NFA: Examples

- $R = ab \cup ba$ $(L = \{ab, ba\})$

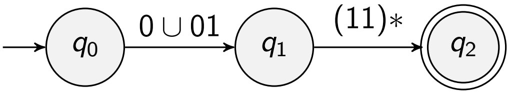
- $R = ab(ab)^*$ $(L = \{ab, abab, ababab, \ldots\})$

Part 2: RL to RE

If a language is regular, then it can be described by a regular expression.

- Why is this useful?
  – one use in answering "what language does this NFA accept?"
- We prove this by constructing a RL equivalent to a given NFA.
- Proof strategy:
  - regular implies equivalent DFA
  - convert DFA to generalized NFA (GNFA): NFA that have regular expressions as transition labels
  - convert GNFA to a form where the RE can be "read off"

Example GNFA



GNFA Definition

Generalized non-deterministic finite automaton
$M = (Q, \Sigma, \delta, q_{start}, q_{accept})$ with

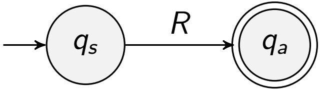
- Q: finite set of states
- Σ: the input alphabet
- q_{start}: the start state
- q_{accept}: the (unique) accept state
- $\delta : (Q \backslash \{q_{accept}\}) \times (Q \backslash \{q_{start}\}) \to \mathcal{R}$ is the transition function ($\mathcal{R}$ is the set of regular expressions over Σ)

(NOTE THE NEW DEFN OF δ)

GNFA δ function

- The interior $Q \setminus \{q_{accept}, q_{start}\}$ is “fully connected” by δ
- From q_{start} only ‘outgoing transitions’
- To q_{accept} only ‘incoming transitions’
- Missing $q_i \to q_j$ transitions are labeled $\delta(q_i, q_j) = \emptyset$

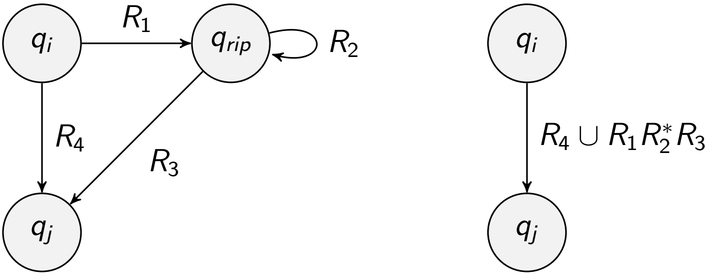
NFA to RE conversion

- given a DFA M, construct an equivalent GNFA M' with $k \geq 2$ states by adding a start and an accept state and connecting them to the old start and accept states with ϵ-transitions
- Add missing $q_i \rightarrow q_j$ transitions with the label $\emptyset$
- Merge transitions $q_i \xrightarrow{a} q_j$, $q_i \xrightarrow{b} q_j$ by replacing them with $q_i \xrightarrow{a \cup b} q_j$
- Reduce one-by-one the internal states until $k = 2$
- This GNFA will be of the form 

- This regular expression R will be such that $L(R) = L(M)$

NFA to RE: Removing States

- identify internal state q_{rip} to be removed
- For **every** $q_i \in Q \setminus \{q_{accept}\}$, $q_j \in Q \setminus \{q_{start}\}$, do the following



- Each such state removal preserves equivalence between the old and the converted GNFA

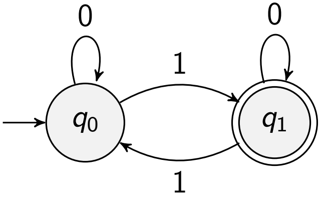
NFA to RE Conversion: Proof of Correctness

- Fairly complicated construction
- Can be programmed, so can be automatically computed
- The formal proof is by induction on the number of states k of the GNFA we started from, and is omitted

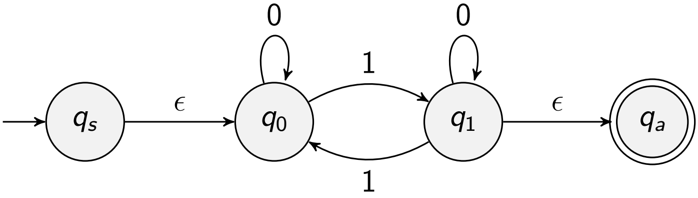
NFA to RE: Example

$L = \{w|$ the sum of the bits of w is odd$\}$

- The DFA for this language is:

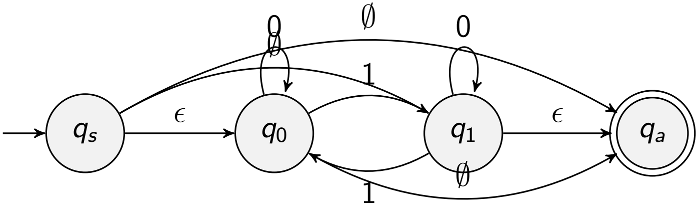


- Step 1: Add the new start and accept states:



NFA to RE: Example

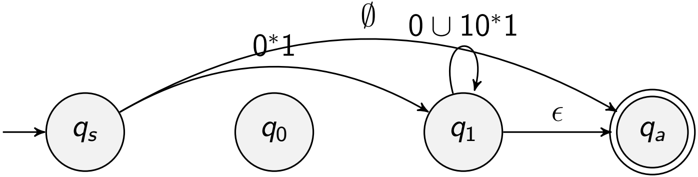
Step 2: Add in all the missing edges



Note: the start state will only have outgoing edges and the accept state will only have incoming edges

NFA to RE: Example

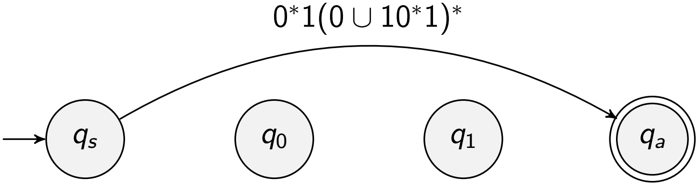
Step 3: Eliminate q_0



Note: the start state will only have outgoing edges and the accept state will only have incoming edges

NFA to RE: Example

Step 4: Eliminate q_1



Result: RL $L = \{w|$ the sum of the bits of w is odd$\}$
is equivalent to RE $0^*1(0 \cup 10^*1)^*$