CSC236 fall 2018

machines, expressions: equivalence

Danny Heap
heap@cs.toronto.edu / BA4270 (behind elevators)
http://www.teach.cs.toronto.edu/~heap/236/F18/
416-978-5899

Using Introduction to the Theory of Computation, Chapter 7

Outline

regular expressions, regular languages

notes

non-deterministic FSA (NFSA) example

FSA that accepts $L((010 + 01)^*)$

NFSAs are real

...you can always convert them to DFSA

Use **subset construction, notes page 219** if $\Sigma = \{0, 1\}$, the construction is, roughly

- start at the start state combined with any states reachable from start with ε-transitions
- if there are any 1-transitions from this new combined start state, combine them into a new state
- there are any 0-transitions from this new combined start, combine them into a new state
- repeat for every state reachable from the start...

NFSA that accepts $L((0 + 10)(0 + 10)*)$

construct the corresponding DFSA...

NFSA that accepts **Rev**$(L((0+10)(0+10)*))$

construct the corresponding DFSA...

FSAs, regexes are equivalent:

$L = L(M)$ for some DFSA $M \Leftrightarrow L = L(M')$ for some NFSA $M' \Leftrightarrow$
$L = L(R)$ for some regular expression R

step 1.0: convert $L(R)$ to $L(M')$

start with $\emptyset$, ε, $a \in \Sigma$

equivalence. . .

step 1.5: convert $L(R)$ to $L(M')$:
union, concatenation, stars

equivalence. . .

step 2: convert $L(M')$ to $L(M)$

use subset construction

there could be $2^{|Q|}$ subsets to consider, but often many are unreachable and may be ignored...

FSAs, regexes are equivalent:

$L = L(M)$ for some DFSA $M \Leftrightarrow L = L(M')$ for some NFSA $M' \Leftrightarrow$
$L = L(R)$ for some regular expression R

step 3: convert $L(M)$ to $L(R)$, eliminate states

equivalence...

state elimination recipe for state q

1. $s_1 \dots s_m$ are states with transitions **to** q, with labels $S_1 \dots S_m$
2. $t_1 \dots t_n$ are states with transitions **from** q, with labels $T_1 \dots T_n$
3. Q is any self-loop on q
4. Eliminate q, and add (union) transition label $S_i Q^* T_j$ from s_i to t_j.

regular languages closure

Regular languages are those that can be denoted by a regular expression or accept by an FSA. In addition:

- L regular $\Rightarrow \overline{L}$ regular
- L regular $\Rightarrow Rev(L)$ regular

pumping lemma (see course notes, page 234)

If $L \subseteq \Sigma^*$ is a regular language, then there is some $n_L \in \mathbb{N}$ (n_L depends on L) such that if $x \in L$ and $|x| \geq n_L$ then:

- $\exists u, v, w \in \Sigma^*, x = uvw$
- $|v| > 0$
- $|uv| \leq n_L$
- $\forall k \in \mathbb{N}, uv^k w \in L$

idea: if machine $M(L)$ has $|Q| = n_L$, $x \in L \wedge |x| \geq n_L$, denote $q_i = \delta^*(q_0, x[\,:i])$, so x "visits" $q_0, q_1, \ldots, q_L$ with the first $n_L + 1$ prefixes of x... so there is at least one state that x "visits" twice (pigeonhole principle)

consequences of regularity

How about $L = \{1^n 0^n | n \in \mathbb{N}\}$

How about $L = \{w \in \Sigma^* \mid |w| = p \wedge p \text{ is prime}\}$

notes