Chap. 1 Regular Languages

- **Language** = set of strings from some alphabet
- Language $L$ is accepted by FA $M$ if after last symbol both:
  - For any string in $L$, $M$ ends in accept state
  - For any string not in $L$, $M$ does not end in accept state
- **Regular Language (RL)**: any language accepted by a FA
  - Also called **Regular Expressions (regex)**
- Question: Is there a way of describing all, and only, languages accepted by a FA? I.e. is there a syntax for RLs?
  - Can we build “larger” languages from “smaller” ones?
  - Answer to all above: YES
- (p.44) Possible set operations on languages $A$ and $B$:
  - **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
  - **Intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
  - **Complementation of $B$ wrt $A$**: $A/B = \{x \mid x \in A \text{ and } x \not\in B\}$
- (p.44) Possible operations on strings in languages
  - **Concatenation**: $A \cdot B = \{xy \mid xy \text{ a string where } x \in A \text{ and } y \in B\}$
  - **Star**: $A^* = \{x \mid x = \epsilon \text{ or } x = x_1x_2...x_k \text{ where } k \geq 1 \text{ and all } x_k \in A\}$
  - **Plus**: $A^+ = \{x \mid x = x_1x_2...x_k \text{ where } k \geq 1 \text{ and all } x_k \in A\}$
- P. 45 Examples of above operations on some simple sets
• Fundamental Question: if we apply any of above operations to known RLs, are we guaranteed to get another RL?
  • Are we guaranteed we can build an FA that accepts result
  • Answer: YES if set of RLs is closed under the operation
  • Closure: A set is closed under some operation if applying it to any member(s) of the set returns another member of set
  • i.e. can we build a FA (DFA or NFA) that accepts any language created by applying specified operation
  • Typical proof process: by construction
    • Assume language A1 accepted by FA M1, A2 by M2:
    • Show how to build an M (typically using M1 and M2 as pieces) that accepts all strings from any combination of sets A1 and A2 using that operation
  • i.e Set of all RLs is closed under these operations
  • Assume following in closure proofs
    • A1 accepted by DFA M1, and M1 = (Q1, ∑, δ1, q1, F1)
    • A2 accepted by DFA M2, and M2 = (Q2, ∑, δ2, q2, F1),
    • Q1 ∩ Q2 = φ (i.e. no common states)
      • We can always “rename” states to prevent confusion
• (p.45,46) Prove **closure under U** by constructing new DFA $M$
  • Construct $M = (Q, \Sigma, \delta, q_0, F)$
    • $Q = Q_1 \times Q_2$
      • i.e. states in $M$ are “named” as tuples $(r_1, r_2)$
        • $r_1$ in $Q_1$, $r_2$ in $Q_2$
    • $\Sigma$ same for all 3 machines
    • $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
    • $q_0 = (q_1, q_2)$
    • $F = \{ (r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2 \}$
  • New machine keeps track of states of both machines
    • If either ends up in their $F$, then accept
    • If neither accept, then reject
  • Do example: $A_1 =$ set of even # of a’s, $A_2 =$ odd # of b’s
  • (p.47) To show we’ve proven closure, must show:
    • If $w$ is accepted by either $M_1$ or $M_2$, it is accepted by $M$
    • If $w$ is accepted by $M$, it is accepted by either $M_1$ or $M_2$
    • Both of above are fairly obvious by construction
  • Proof of **closure under intersection** is simple: change $F$!
• Since DFAs=NFAs, L is regular iff accepted by some NFA
• (p. 59-60) **Alternative construction proof of U** using NFAs
  • A1 accepted by NFA N1, and N1 = (Q1, ∑, δ1, q1, F1)
  • A2 accepted by NFA N2, and N2 = (Q2, ∑, δ2, q2, F1),
  • Construct N = (Q, ∑, δ, q0, F) to recognize A1 U A2
    • Q = {q0} U Q1 U Q2
    • F = F1 U F2
    • δ(q, a) =
      • = δ1(q, a) if q ∈ Q1
      • = δ2(q, a) if q ∈ Q2
      • = {q1, q2} if q = q0 and a = ε
      • = ∅ if q = q0 and a ≠ ε
  • New starting state q0 “guesses correctly” which other machine to start – without looking at any input
• Proving ∘ or * is “harder” – we don’t know when to stop string from one language and start other!
• Really need nondeterminism!
• (p. 60) Proof that RLs are **closed under concatenation**
  • See Fig. 1.48 on p. 61
    • ε edge from *each* final state of N1 to start state of N2
    • N “guesses” when to hop from N1 to N2
  • A1 accepted by NFA N1, and N1 = (Q1, Σ, δ1, q1, F1)
  • A2 accepted by NFA N2, and N2 = (Q2, Σ, δ2, q2, F1)
  • Construct N = (Q, Σ, δ, q0, F) to recognize A1 ◦ A2
    • Q = Q1 U Q2
    • q0 = q1 (from N1)
    • F = F2 (from N2)
    • δ(q, a) =
      • = δ1(q, a) if q ∈ Q1 and q not in F1
      • = δ1(q, a) if q ∈ F1 and a ≠ ε
      • = δ1(q, a) U {q2} if q ∈ F1 and a = ε
      • = δ2(q, a) if q ∈ Q2 and any a
• (p. 62) Proof that RLs are **closed under Kleene star**
  • See Fig. 1.50 on p. 62
    • Add ε edge from *each* final state back to start
    • Again guess correctly when to restart N1
  • A1 accepted by NFA N1, and N1 = (Q1, ∑, δ1, q1, F1)
  • Construct N = (Q, ∑, δ, q0, F) to recognize A1*
  • Q = {q0} U Q1
  • q0 = a new state
  • F = {q0} U F1
    • {q0} for empty set when 0 copies
  • δ(q,a) =
    • = δ1(q, a) if q ∈ Q1 and q not in F1
    • = δ1(q, a) if q ∈ F1 and a ≠ ε
    • = δ1(q, a) U {q1} if q ∈ F1 and a = ε
    • = {q1}
    • = φ if q = Q0 and a ≠ ε
• (p63 – Section 1.3) Regular Expressions

• Example: describing arithmetic expressions:

\[ \begin{align*}
\text{<op1>} & \rightarrow + \mid - \\
\text{<op2>} & \rightarrow * \mid / \\
\text{<factor>} & \rightarrow \text{<number>} \mid (\text{<arith-expr>}) \mid \text{<factor>}^\text{<factor>} \\
\text{<term>} & \rightarrow \text{<factor>} \mid \text{<term>} \text{<op2>} \text{<factor>} \\
\text{<arith-expr>} & \rightarrow \text{<term>} \mid \text{<arith-expr>} \text{<op1>} \text{<term>} \\
\end{align*} \]

• Notice this defines a precedence for operators:
  
  • Do inside () first
  • Do ^ next
  • Do * or / next before + or –
  • Do + or - last
• (p. 64) Describing regular expressions $R$ (no precedence)
  $<\text{regex} \; \rightarrow \; \phi \mid \varepsilon \mid \ldots \text{any member of } \Sigma \ldots$
  $\mid ( \langle \text{regex} \rangle \cup \langle \text{regex} \rangle )$
  $\mid (\langle \text{regex} \rangle \circ \langle \text{regex} \rangle )$
  $\mid (\langle \text{regex} \rangle^*)$

• Note: this demands () all the time
• No assumed precedence
• Normal Precedence rules – drop unnecessary ()
  • Do inside () first
  • Do * first, then $\circ$, then $\cup$

• Examples p. 65 Example 1.53
• Redo of BNF to “build-in” precedence
  $<\text{basic-regex} \; \rightarrow \; \phi \mid \varepsilon \mid \ldots \text{any member of } \Sigma \ldots$
  $<\text{regex-factor} \; \rightarrow \; <\text{basic-regex} \mid ( \langle \text{regex} \rangle )$
  $\mid <\text{regex-factor}^* \mid <\text{regex-term}> \circ <\text{regex-factor} >$
  $<\text{regex} \; \rightarrow \; <\text{regex-term} >$
  $\mid <\text{regex} \cup <\text{regex} >$
• Examples p. 65
• (p. 66) **Identities:** for all \( R \)
  • \( R \cup \emptyset = R \). Adding empty language to any other does not change it
  • \( R \circ \varepsilon = R \). Concatenating the empty string to any string in a language does not change \( R \)
• (p. 66) **Non-identities**
  • \( R \cup \varepsilon \) may be different from \( R \).
    • E.g. \( R = 0 \) so \( L(R) = \{0\} \), but \( L(R \cup \varepsilon) = \{0, \varepsilon\} \)
  • \( R \circ \emptyset \) may be different from \( R \).
    • E.g. \( R = 0 \) so \( L(R) = \{0\} \), but \( L(R \circ \emptyset) = \emptyset \)
      • There are no strings to concatenate on right
• (p. 66) Regex for `<number>` as defined above
  • \( D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \)
  • \((+ \cup - \cup \varepsilon) (D^+ \cup D^+.D^* \cup D^*.D^+)\)
• (p. 66) Theorem 1.54: A language is regular iff a regular expression describes it.
  • Remember all RLs eqvt to FA
• Lemma 1.55: If L described by a regex R, its regular
  • (p. 67) Proof by construction of an NFA: 6 cases
  • (p. 68, 69) Ex. 1.56, 1.57, 1.58, 1.59
• Lemma 1.60 (p. 69): If L is regular then it is described by a regex
  • Proof by construction from DFA to GNFA to regex
• Generalized NFAs (GNFA)
  • NFA where edges may have arbitrary regex on them
    • We know that any regex can be converted into an NFA
    • Thus could replace each such edge with a small NFA
  • Start state as transitions to every other state but no incoming
  • Only one accept state with transitions incoming from all others but no outgoing
  • Start and accept states must be different
  • Except for start and accept, transition from every state to every other state, including a self-loop
(p. 73) **Formal Definition of GNFA** \((Q, \Sigma, \delta, q_{start}, q_{accept})\)

- **\(\delta\):** \((Q-\{q_{accept}\}) \times (Q-\{q_{start}\}) \rightarrow R\), where \(R\) is all regex over \(\Sigma\)
- GNFA accepts \(w\) if \(w=w_1...w_k\) where each \(w_i\) string from \(\Sigma^*\)
- and sequence of states \(q_0,...q_k\) such that
- \(q_0 = q_{start}, q_k = q_{final}\)
- \(w_i \in L(R_i)\) where \(R_i = \delta(q_{i-1}, q_i)\) (i.e. the label on the edge)

(p. 71) Any DFA can be converted into GNFA

- Add new start state with \(\epsilon\) transition to old start
- Add new final state with \(\epsilon\) from all old final states
- If edge has multiple labels
  - Replace by single edge with label = \(U\) of prior labels
- Add edge with \(\emptyset\) between any states without an edge
- See Fig. 1-61: do conversion on paper to bigger NFA

(p. 69) **Lemma 1.60** If \(A\) is regular, then describable by regex

(p. 73) Proof by converting DFA \(M\) for \(A\) into GNFA \(G\)
- With \(k = \#\) states in \(G\)
- Then modify GNFA as follows
  - If \(k=2\) then GNFA must have \(q_{start}\) and \(q_{accept}\) and edge between them is desired regex
  - If \(k>2\), repeat until \(k=2\): convert \(G\) into \(G'\)
    - Select any start \(q_{rip}\) other than \(q_{start}\) and \(q_{accept}\)
    - Define \(G'\) be GNFA where \(Q' = Q - \{q_{rip}\}\)
    - For each \(q_i\) in \(Q' - q_{start}\) and \(q_j\) in \(Q' - \{q_{accept}\}\)
\[ \delta'(q_i, q_j) = (R1)(R2)^*(R3) \cup (R4) \]

- \( R1 = \delta(q_i, q_{\text{rip}}) \) (label on edge from \( q_i \) to \( q_{\text{rip}} \))
- \( R2 = \delta(q_{\text{rip}}, q_{\text{rip}}) \) (label on edge on self loop \( q_{\text{rip}} \))
- \( R3 = \delta(q_{\text{rip}}, q_j) \) (label on edge from \( q_{\text{rip}} \) to \( q_j \))
- \( R4 = \delta(q_i, q_j) \) (original label on edge from \( q_i \) to \( q_j \))

- Eg. p. 75, 76