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 **U** B = {x | xεA or xεB}
  - **Intersection**: A ∩ A B {x | xεA and xεB}
  - **Complementation of B wrt A**: A/B = {x| xεA and x not in B}
- (p.44) Possible operations on strings in languages
  - **Concatenation**: A ◦ B = {xy |xy a string where xεA and yεB}
  - **Star**: A^* = { x|x = ε or x= $x_1x_2 \ldots x_k$ where k≥1 and all x_kεA }
  - **Plus**: A^+ = { x|x= $x_1x_2 \ldots x_k$ where k≥1 and all x_kεA }
- 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, ∑, δ, q0, F)
    - Q = Q1 x Q2
      - i.e. states in M are "named" as tuples (r1, r2)
        - r1 in Q1, r2 in Q2
    - ∑ same for all 3 machines
    - δ((r1,r2), a) = (δ1(r1,a), δ2(r2,a))
    - q0 = (q1, q2)
    - F = { (r1, r2) | r1εF1 <u>or</u> r2εF2}
- New machine keeps track of states of <u>both</u> machines
  - If either ends up in their F, then accept
  - If neither accept, then reject
- Do example: A1 = set of even # of a's, A2 = odd # of b's
- (p.47) To show we've proven closure, must show:
  - If w is accepted by either M1 or M2, it is accepted by M
  - If w is accepted by M, it is accepted by either M1 or M2
- 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 ε Q1
      - = δ2(q, a) if q ε Q2
      - = {q1, q2} if q = q0 and a = ε
      - = ϕ 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, ∑, δ1, q1, F1)
  - A2 accepted by NFA N2, and N2 = (Q2, ∑, δ2, q2, F1)
  - Construct N = (Q, ∑, 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 ε Q1 and q not in F1
      - = δ1(q, a) if q ε F1 and a ≠ ε
      - = δ1(q, a) U {q2} if q ε F1 and a = ε
      - = δ2(q, a) if q ε 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 ε Q1 and q not in F1
      - = δ1(q, a) if q ε F1 and a ≠ ε
      - = δ1(q, a) U {q1} if q ε F1 and a = ε
      - = {q1}
      - = ϕ if q = Q0 and a ≠ ε

- (p63 – Section 1.3) Regular Expressions
- Example: describing arithmetic expressions:

  <op1> -> + | -

  <op2> -> * | /

  <factor> -> <number> | (<arith-expr>) | <factor>^<factor>

  <term> -> <factor> | <term> <op2> <factor>

  < arith-expr > -> <term> | < arith-expr > <op1> <term>

  - 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)

  <regex > -> ϕ | ε | … any member of ∑ …

  | (<regex > U <regex >)

  | (<regex > ◦ <regex >)

  | (<regex>*)

  - Note: this demands () all the time
  - No assumed precedence
  - Normal Precedence rules – drop unnecessary ()
    - Do inside () first
    - Do * first, then ◦, then U
- Examples p. 65 Example 1.53
- Redo of BNF to “build-in” precedence

  <basic-regex> -> ϕ | ε | … any member of ∑ …

  <regex-factor> -> <basic-regex> | (<regex>)

  | <regex-factor>*

  <regex-term> -> <regex-factor>

  | <regex-term> ◦ < regex-factor >

  <regex > -> <regex-term>

  | <regex> U <regex>

- Examples p. 65
- (p. 66) **Identities**: for all R
  - R U ϕ = R. Adding empty language to any other does not change it
  - R ◦ ε = R. Concatenating the empty string to any string in a language does not change R
- (p. 66) **Non-identities**
  - R U ε may be different from R.
    - E.g. R = 0 so L(R) = {0}, but L(R U ε) = {0, ε}
  - R ◦ ϕ may be different from R.
    - E.g. R = 0 so L(R) = {0}, but L(R ◦ ϕ) = ϕ
      - 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}
  - (+ U – U ε) (D^+ U D^+.D* U 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, Σ, δ, q_{start}, q_{accept})
  - δ: (Q-{q_{accept}}) x (Q-{q_{start}}) -> R, where R is all regex over Σ
  - GNFA accepts w if $w=w_1 \ldots w_k$ where each w_i string from Σ^*
  - and sequence of states q0,…qk such that
  - $q0 = q_{start}$, $qk = q_{final}$
  - w_i ε $L(R_i)$ 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 ε transition to old start
  - Add new final state with ε from all old final states
  - If edge has multiple labels
    - Replace by single edge with label = U of prior labels
  - Add edge with ϕ 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}}

- $\delta'(q_i,q_j) = (R1)(R2)^*(R3) \cup (R4)$ where
  - $R1 = \delta(q_i,q_{rip})$ (label on edge from q_i to q_{rip})
  - $R2 = \delta(q_{rip},q_{rip})$ (label on edge on self loop q_{rip})
  - $R3 = \delta(q_{rip},q_j)$ (label on edge from q_{rip} to q_j))
  - $R4 = \delta(q_i,q_j)$ (original label on edge from q_i to q_j)
- Eg. p. 75,76