Section 1.3 Regular Expressions

- Sample syntax for numbers

  D -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

  <unsigned-number> -> D+

  <unsigned-fraction> -> <unsigned-number> |

  <unsigned-number>.{<unsigned-number>} |

  <number> -> {+ -}<unsigned-fraction>

- Sample syntax for describing arithmetic expressions:

  <op1> -> + | -

  <op2> -> * | /

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

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

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

  - Notice this defines a precedence for operators:
    - Do inside () first
    - Do * or / first before + or –
    - Do + or - last

- (p. 64) Describing regular expressions (**regex**) R

  <r-symbol> -> ϕ | ε | … any member of ∑ …

  Notes:

  ε is language with a <u>single</u> string – the empty string

  ϕ is the language with <u>no</u> strings

  ∑ stands for any character in the alphabet

  <r-factor> -> <r-symbol> | (<r-expr>) | <r-factor>*

  <r-term> -> <r-factor> | <r-term> ◦ <r-factor>

  <r-expr> -> <r-term> | <r-expr> U <r-factor>

  - Precedence rules
    - Do inside () first
    - Do * first, then ◦, then U
  - R^+ shorthand for RR*
  - ∑ is thus equivalent to (a_1 U a_2 U …) where a_i is a symbol in ∑
- L(R) stands for the language generated by the regular expression R

- Examples p. 65
- (p. 66) Identities: for all R
  - R U ϕ = R. Adding empty language to any other does not change it
  - $R \circ \varepsilon = R$. Concatenating the empty string 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}
  - $R \circ \phi$ may be different from R.
    - E.g. R = 0 so L(R) = {0}, but $L(R \circ \phi) = \phi$
      - 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.66-67) **Theorem 1.54**. A language is regular iff some regular expression describes it
  - (p. 67) If L is described by a regex, then it is regular
    - Proof by construction: given regex construct an NFA
    - See p. 67 for 6 cases and how to build their NFAs
    - Examples p. 68, 69
- (p70) To prove other way need **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, $\sum$, δ, q_{start}, q_{accept})
    - δ: (Q-{q_{accept}}) x (Q-{q_{start}}) -> R, where R is all regex over $\sum$
  - GNFA accepts w if $w=w_1 \ldots w_k$ where each w_i is string from $\sum^*$
    - 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