Section 1.3 Regular Expressions

- Sample syntax for numbers
  
  \[D \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9\]
  
  \[<\text{unsigned-number}> \rightarrow D^+\]
  
  \[<\text{unsigned-fraction}> \rightarrow <\text{unsigned-number}> |\]
  
  \[<\text{unsigned-number}>.|{<\text{unsigned-number}>} |\]
  
  \[<\text{number}> \rightarrow \{+ -\}<\text{unsigned-fraction}>\]

- Sample syntax for describing arithmetic expressions:
  
  \[<\text{op1}> \rightarrow + | -\]
  
  \[<\text{op2}> \rightarrow * | /\]
  
  \[<\text{factor}> \rightarrow <\text{number}> | (<\text{arith-expr}>)\]
  
  \[<\text{term}> \rightarrow <\text{factor}> | <\text{term}> <\text{op2}> <\text{factor}>\]
  
  \[<\text{arith-expr}> \rightarrow <\text{term}> | <\text{arith-expr}> <\text{op1}> <\text{term}>\]

- Notice this defines a precedence for operators:
  
  - Do inside () first
  
  - Do * or / first before + or –
  
  - Do + or - last
• (p. 64) Describing regular expressions (regex) $R$

\[
<r\text{-symbol}> \rightarrow \phi \mid \varepsilon \mid \ldots \text{any member of } \Sigma \ldots
\]

Notes:

- $\varepsilon$ is language with a single string – the empty string
- $\phi$ is the language with no strings
- $\Sigma$ stands for any character in the alphabet

\[
<r\text{-factor}> \rightarrow <r\text{-symbol}> \mid ( <r\text{-expr}> ) \mid <r\text{-factor}>^*
\]

\[
<r\text{-term}> \rightarrow <r\text{-factor}> \mid <r\text{-term}> ⊕ <r\text{-factor}>
\]

\[
<r\text{-expr}> \rightarrow <r\text{-term}> \mid <r\text{-expr}> \cup <r\text{-factor}>
\]

• Precedence rules
  - Do inside () first
  - Do $\ast$ first, then $\circ$, then $\cup$

• $R^+$ shorthand for $RR^*$

• $\Sigma$ is thus equivalent to $(a_1 \cup a_2 \cup \ldots)$ where $a_i$ is a symbol in $\Sigma$

• $L(R)$ stands for the language generated by the regular expression $R$
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 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.66-67) **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 \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\)
  • \( \delta : (Q-\{q_{\text{accept}}\}) \times (Q-\{q_{\text{start}}\}) \to R \), where \( R \) is all regex over \( \Sigma \)
  • GNFA accepts \( w \) if \( w=w_1...w_k \) where each \( w_i \) is string from \( \Sigma^* \)
    • and sequence of states \( q_0,...q_k \) such that
    • \( q_0 = q_{\text{start}}, q_k = q_{\text{final}} \)
    • \( w_i \in L(R_i) \) 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 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 $q_{\text{start}}$ and $q_{\text{accept}}$ and edge between them is desired regex
      • If k>2, repeat until k=2: convert G into G’
        • Select any start $q_{\text{rip}}$ other than $q_{\text{start}}$ and $q_{\text{accept}}$
        • Define G’ be GNFA where $Q’ = Q – \{q_{\text{rip}}\}$
        • For each $q_i$ in Q’ - $q_{\text{start}}$ and $q_j$ in Q’ – $\{q_{\text{accept}}\}$
          • $\delta'(q_i,q_j) = (R1)(R2)*(R3) U (R4)$ where
            • $R1 = \delta(q_i,q_{\text{rip}})$ (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