Regular expressions

- used for describing string patterns, e.g.

\[(0 \cup 1)0^* \rightarrow \text{start with one zero or one one, followed by any number of zeros}\]

\[(0 \cup 1)^* \rightarrow \text{all strings}\]

\[(\{0\} \cup \{1\})^* \{01\} (\{0\} \cup \{1\})^* (01)^* 01 (01)^*\]
Regular expressions

Formal definition:

R is a regular expression if R is one of the following:

1. a for some \( a \in \Sigma \),
2. \( \varepsilon \)
3. \( \emptyset \)
4. \( (R_1 \cup R_2) \), where \( R_1, R_2 \) are regular expressions
5. \( (R_1.R_2) \), where \( R_1, R_2 \) are regular expressions
6. \( (R_1^*) \), where \( R_1 \) is a regular expression.

Note: this type of definition is called a recursive/inductive definition (i.e. the definition is a recursive algorithm).
Regular expressions

For convenience: \( R^+ = RR^* \)

Examples: give regular expressions for the following languages:

- \{ w \in \{0,1\}^* \mid w \text{ contains the substring 001} \} \quad \text{should be: } (0u1)^*001(0u1)^* \quad \text{will simplify by using preference of operations (like nfa)} \quad (5+(7\cdot2^8))

- \{ w \in \{0,1\}^* \mid w \text{ does not contain two consecutive 0's} \} \quad 1^*(011^*)^*(0u\varepsilon) \quad \text{will simplify by using preference of operations (like nfa)} \quad (5+(7\cdot2^8))

- \{ w \in \{0,1\}^* \mid |w| \text{ is divisible by 2 or 3} \} \quad (0u1^2)^* \cup (0u1^3)^* \quad \text{math preference: } 1 \text{ exponentiation, } 2 \text{ multiplication, } 3 \text{ addition, } 4 \text{ star, } 5 \text{ concat.}

- \{ w \in \{0,1\}^* \mid |w| < 4 \} \quad (0u1u\varepsilon)(0u1u\varepsilon)(0u1u\varepsilon) \quad \text{will shorten to: } (0u1u\varepsilon)^3 \quad \text{or } (0u1)^* \cup u (0u1)(0u1) \cup (0u1)(0u1)(0u1)
Examples: let \( R \) be any regular expression

- \( R \cdot \emptyset = \emptyset \)

- \( R \cdot \{\epsilon\} = R \)

- \( \emptyset^* = \emptyset \)

- \( \epsilon^* = \epsilon \)

- \( \{1\} \{T\}^* = \{1\} \epsilon \)

The language defined by \( R \) is denoted \( L(R) \). We’ll often abuse notation and use \( R \) to denote the language \( L(R) \).
Thm 1.54: A language is regular iff some regular expression describes it.

Lemma 1.55: Given a regular expression $R$, there exists a FA $M$ such that $L(M) = L(R)$.

Lemma 1.60: Given a FA $M$, there exists a regular expression $R$ such that $L(R) = L(M)$. 
Lemma 1.55: Given a regular expression $R$, there exists a FA $M$ such that $L(M) = L(R)$.

**Proof:** by structural induction

**Base Case:**
1. if $R = a$ for some $a \in \Sigma$, then NFA:

   ![Simple NFA for a single symbol]

2. if $R = \varepsilon$, then NFA:

   ![NFA for epsilon]

3. if $R = \emptyset$, then NFA:

   ![NFA for empty set]

**Inductive Case:**
4. if $R = (R_1 \cup R_2)$ then by the inductive hypothesis (IH) we have NFAs $N_1, N_2$ for $R_1, R_2$. Then construct NFA $N$ for $(R_1 \cup R_2)$ by applying the construction from last week.

Example:

![Example NFA for expression $0 \cup 1 (0 \cup 1)^*$]

More complicated than the straightforward NFA, but this process was completely automated (no need to think).
Lemma 1.60: Given a FA $M$, there exists a regular expression $R$ such that $L(R) = L(M)$.

Proof idea:

Generalized NFA (GNFA)

- transitions may be marked by reg.expr. (not just $\Sigma \cup \{\varepsilon\}$)
- single accept state that a) has arrows coming in from every other state, b) does not have any outgoing arrows ✓
- start state that a) has arrows to every other state, b) does not have any incoming arrows ✓
- all other states have arrows to all other states ✓
Lemma 1.60: Given a FA $M$, there exists a regular expression $R$ such that $L(R) = L(M)$.

Proof idea:

Generalized NFA (GNFA) $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ where all as usual except $\delta: (Q-\{q_{\text{accept}}\}) \times (Q-\{q_{\text{start}}\}) \to R$ where $R$ is the set of all regular expressions over $\Sigma$.

Idea: start with a GNFA, remove states one by one and redraw arrows as necessary.

How to get a GNFA:

Example: 

Suppose we remove the top state (q2), then we need to account for e.g. q1,q2,...q2,q_accept path: 01*.E union this with the current expression on q,q_accept

- new accepting state
- no outgoing
- need everybody incoming
- new starting state
- connect inner states with labels $\emptyset$
Lemma 1.60: Given a FA $M$, there exists a regular expression $R$ such that $L(R) = L(M)$.

Proof idea:

How to construct an equivalent GNFA with one fewer state?

**Algorithm/construction:**

1. For every internal state $q_r$, we'll remove $q_r$:
2. For every pair of state $r, s$ from the remaining states, $r \neq q_{accept}, s \neq q_{start}$: (update transition from $r$ to $s$, i.e. $\delta(r,s)$)
3. Return (when we have only 2 states): $\delta(q_{start}, q_{accept})$