Regular expressions

- used for describing string patterns, e.g.

\[(0 \cup 1)0^*\]  - starting with 0 or 1, followed by any number of 0's

\[(0 \cup 1)^*\]  - any string over \( \{0, 1\} \)
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. $(R_1 \cdot R_2)$, 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:

- $\{ w \in \{0,1\}^* \mid w \text{ contains the substring 001 } \} = (0u1)^*\text{ 001 } (0u1)^*$

- $\{ w \in \{0,1\}^* \mid w \text{ does not contain two consecutive } 0\text{'s } \} = 1^*(011^*)^*(0u\varepsilon)$

- $\{ w \in \{0,1\}^* \mid |w| \text{ is divisible by 2 or 3 } \} = (0u1)^2)^* \cup (0u1)^3)^*$

- $\{ w \in \{0,1\}^* \mid |w| < 4 \} = (0u1u\varepsilon)^3$
Regular expressions

Examples: let $R$ be any regular expression

- $R \cdot \emptyset = ? \emptyset \quad L. \emptyset = \emptyset$

- $R \cdot \epsilon = ? \emptyset \quad R$

- $\emptyset^* = ? \epsilon \quad L^* \text{ always contains } \epsilon$

- $\epsilon^* = ? \epsilon$

The language defined by $R$ is denoted $L(R)$. We’ll often abuse notation and use $R$ to denote the language $L(R)$. 
Equivalence of reg. expr. and FA's

Kleene's Thm means: 3 a FA (det, or nondet b/c. we can then convert it to a DFA) for the language

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 $L(M) = L(R)$. ← easier, we'll do hint

Lemma 1.60: Given a FA $M$, there exists a regular expression $R$ such that $L(R) = L(M)$. 
Equivalence of reg. expr. and FA's

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

**PF:** we'll use induction on regular expressions.

**BASE CASE:**
1. $R = \emptyset$  
2. $R = \epsilon$  
3. $R = \sigma$ for some $\sigma \in \Sigma$

**INDUCTIVE CASE:** Let $R$ be a reg. expression obtained from rules 4, 5, or 6.

- if $R$ was obtained using rule 4: $R = R_1 \cup R_2$  
- if $R$ was obtained using rule 5: $R = R_1 \cdot R_2$  
- if $R$ was obtained using rule 6: $R = R_1^*$

Then we construct a finite automaton $M$ for $L(M)$, $UL(M_2)$, a language represented by $R$.

Note: this is called a structural induction.
Lemma 1.55: Given a regular expression $R$, there exists a FA $M$ such that $L(M) = L(R)$. 

Equivalence of reg. expr. and FA's

[Section 1.3]
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:
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?
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?

Given GNFA $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ we are constructing GNFA $G_1$ with one fewer state s.t. $L(G) = L(G_1)$

Let $G_1 = (Q_1, \Sigma, \delta_1, q_{\text{start}}, q_{\text{accept}1})$

Suppose $p \in Q - \{q_{\text{start}}, q_{\text{accept}}\}$ is the state we are eliminating

$Q_1 = Q - \{p\}$

$\delta_2(q, r) = \delta(q, r) \cup \delta(q, p) \delta(p, r)^*$

$\delta(q, r)$ is the regular expression for going from $q$ to $r$