**CIT3130: Theory of Computation**

### Regular languages

(“M” refers to the first edition of Martin and “H” to IALC by Hopcroft et al.)

Definitions of regular expressions and regular languages: A regular expression of an alphabet Σ is composed from the expressions \( \emptyset, \epsilon, a \ (a \in \Sigma), S \ (S \subseteq \Sigma) \) using union \( (E_1 + E_2) \), concatenation \( (E_1 E_2) \) and iteration \( (E^* \), where \( E^* = \{ \epsilon \} \cup E \cup E^2 \cup \cdots \). A language is regular if it is the language defined by some regular expression.

Examples of regular languages: strings of even length (M, Ex. 3.1); strings of odd length; strings in \{0, 1\}\(^*\) containing at least one 1 (M, Ex. 3.2); strings in \{0, 1\}\(^*\) of length at most 6 (M, Ex. 3.4); strings in \{0, 1\}\(^*\) that end in 1 and do not contain the substring 00 (M, Ex. 3.5); Java floating point constants (M, Ex. 3.6); strings in \{0, 1\}\(^*\) denoting integers divisible by 3 (best postponed until finite automata have been introduced).

A regular language may be defined by different regular expressions, e.g., \( 0^*1(0 + 1)^*, (0 + 1)^*1(0 + 1)^* \) and \( (0 + 1)^*10^* \) (M, Ex. 3.3).

Regular languages are closed under union, concatenation and iteration (trivial). Regular languages are also closed under complement \( (I' = \Sigma^* - I) \), intersection, difference, symmetric difference \( (I_1 \Delta I_2 = (I_1 - I_2) \cup (I_2 - I_1)) \). If \( I \) is regular, then \( rev(I) \) is regular, where \( rev(I) = \{ a_n a_{n-1} \ldots a_1 \mid a_1 a_2 \ldots a_n \in I \} \).

Every finite language is regular. Informal proof and proof by induction.

Examples of nonregular languages: \{ss\(^R\) \mid s \in \Sigma^*\}, where \( s^R = rev(s) \); the set of all palindromes in \{0, 1\}\(^*\) (M, Thm 4.3); \{a^n b^n \mid n \geq 0\}; legal arithmetic expressions using the identifier \( a \), the operator \(+\), and left and right parentheses.

Applications of regular expressions (H, 3.3).

Algebraic laws can be used to simplify regular expressions (H, 3.4): \( \emptyset + L = L + \emptyset = L \), \( \epsilon L = L \epsilon = L \), \( \emptyset L = L \emptyset = \emptyset \), \( L + M = M + L \), \( (L + M) + N = L + (M + N) \), \( (LM)N = L(MN) \), \( L(M + N) = LM + LN \), \( (L + M)N = LM + LN \), \( L + L = L \), \( L + \epsilon = L + \epsilon \), \( \epsilon + \epsilon = \epsilon \), \( L^+ = LL^* = L^*L \), \( L^* = L^+ + \epsilon \), \( [L] = L + \epsilon \).

*******

Finite-state automaton for the regular expression \((1 + 01)^+\), where \( E^+ = EE^* \).

Definition of (deterministic) finite-state automata (DFA). A DFA has a set of states \( Q \), an alphabet \( \Sigma \), an initial state \( q_0 \in Q \), a transition function \( \delta : Q \times \Sigma \rightarrow Q \), and a set of accepting states \( A \subseteq Q \). Often a DFA is drawn as a transition diagram, with each state having a transition for each element of \( \Sigma \). (In programs, the transition function \( \delta \) of a DFA is represented by a state transition matrix.)

Examples of DFAs: Initial and simplified DFA for the regular expression \{0, 1\}\(^*\)10 (M, Ex. 4.4); DFA for the binary numbers divisible by 3.

Definition of the extended transition function \( \delta^* : Q \times \Sigma^* \rightarrow Q \).
A string $s \in \Sigma^*$ is accepted by a DFA if $\delta^*(q_0, s) \in A$. The language $L(M)$ accepted, or recognised, by a DFA $M$ is the set of strings in $\Sigma^*$ that are accepted by $M$.

Consider the DFA with $Q = \{A, 0, D, 1, B\}$, $\Sigma = \{0, 1\}$, $q_0 = A$, $\delta = \{(A, 0, 0), (A, 1, 1), (0, 0, A), (0, 1, D), (D, 0, D), (D, 1, D), (1, 0, D), (1, 1, B), (B, 0, D), (B, 1, 1)\}$, $A = \{A, B\}$ (M, Ex. 4.5). What languages does this DFA recognise?

Another example of a DFA: Strings in $\{0, 1\}^*$ with an even number of 0s and an even number of 1s (H, Ex. 2.4).

Programs to recognise languages accepted by DFAs.

Kleene’s theorem: A language is regular (has a regular expression) if and only if it is recognised by some DFA. (See below.)

DFAs may have “unnecessary” states and transitions. Consider the DFA for $(1 + 01)^+$ (M, Ex. 5.1). We could omit one state, and say that if the machine has no transition it can quit immediately and not accept the string.

DFAs may be unnecessarily complex. Consider the DFA for $(0 + 1)^*(000 + 111)(0 + 1)^*$ and the nondeterministic finite-state automaton that corresponds directly to the regular expression (M, Ex. 5.2). We say the machine accepts a string if there is some sequence of transitions that leads to an accepting state.

Definition of a nondeterministic finite-state automaton (NFA). An NFA has a set of states $Q$, an alphabet $\Sigma$, an initial state $q_0 \in Q$, a transition function $\delta : Q \times \Sigma \rightarrow 2^Q$, and a set of accepting states $A \subseteq Q$.

Example of an NFA for $\{s \in \{0, 1\}^* \mid |s| \geq N \text{ and the Nth symbol from the right is 1} \}$, for $N \geq 1$ (M, Ex. 5.3; N, Ex. 2.13).

Definition of the extended transition function $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ for NFAs:

1. $\delta^*(q, \epsilon) = \{q\}$
2. $\delta^*(q, sa) = \bigcup_{p \in \delta^*(q, s)} \delta(p, a)$

That is, $\delta^*(q, s)$ is the set of states that the machine can reach from the initial state $q$ with input string $s$.

A string $s \in \Sigma^*$ is accepted by an NFA if $\delta^*(q_0, s) \cap A \neq \emptyset$. The language $L(M)$ accepted, or recognised, by an NFA $M$ is the set of strings in $\Sigma^*$ that are accepted by $M$.

NFAs may have many fewer states than the corresponding DFA. Consider the above NFA for $\{s \in \{0, 1\}^* \mid |s| \geq N \text{ and the Nth symbol from the right is 1} \}$.

Equivalence of DFAs and NFAs by subset construction (H, 2.3.5–6).

To construct a DFA $D$ equivalent to an NFA $N$, let $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$, where

1. $Q_D = 2^Q_N$, i.e., $Q_D$ is the set of subsets of $Q_N$. 

2
2. For each set $S$ in $Q_D$ and symbol $a$ in $\Sigma$, 
\[ \delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a). \]

I.e., $\delta_D(S, a)$ is the union of all the states that $N$ can go to from states in $S$ with input $a$.

3. $F_D$ is the set of states $S$ in $Q_D$ (equivalently, subsets $S$ of states in $Q_N$) such that $S \cap F_N \neq \emptyset$.

Example of subset construction (H, Exx. 2.6 and 2.9): Construction of DFA equivalent to 3-state NFA (H, Fig. 2.9) accepting all strings in $\{0, 1\}^*$ that end with 01.

Example of subset construction for $(N + 1)$-state NFA (H, Fig. 2.15) accepting all strings in $\{0, 1\}^*$ such that the $N$th symbol from the right is 1 (H, Ex. 2.13).

Application of DFAs and NFAs to text search (H, 2.4).

*****

The use of $\varepsilon$-transitions — moves that change state without reading input symbols — further simplify NFAs (M, Ex. 5.4) and the proof of Kleene’s theorem. NFAs with $\varepsilon$-transitions are called $\varepsilon$-NFAs.

Example: $\varepsilon$-NFA for decimal numbers (H, Ex. 2.16).

Definition of the extended transition function $\delta^*$ to incorporate $\varepsilon$-transitions by using $\varepsilon$-closures, (H, 2.5.4).

Equivalence of $\varepsilon$-NFAs and DFAs by a modified subset construction using $\varepsilon$-closures (H, 2.5.5).

Example: Construction of a DFA equivalent to the $\varepsilon$-NFA above for decimal numbers (H, Ex. 2.21).

Example: Construction of a DFA equivalent to the four-state $\varepsilon$-NFA with $\delta = \{(q_0, \varepsilon, q_1), (q_1, \varepsilon, q_2), (q_1, 0, q_3), (q_2, 0, q_3), (q_2, 1, q_2)\}$ and $A = \{q_3\}$ (M, Ex. 5.6).

*****

Kleene’s theorem: A language is regular (has a regular expression) if and only if it is recognised by some DFA.

See Fig. 3.1 (H., p.91).

Proof: Every NFA is equivalent to a DFA by the subset construction (H, 2.3.5). Every DFA is equivalent to a regular expression by the state elimination construction (H, 3.2.2). Every regular expression is equivalent to an $\varepsilon$-NFA by a simple inductive construction (H, 3.2.3). Finally, every $\varepsilon$-NFA is equivalent to an NFA by a modified subset construction (H, 2.5.5). (This last modification involves use of $\varepsilon$-closures.)

Example: Construct a regular expression from the following DFA that accepts all strings in $\{0, 1\}^*$ containing at least one 0 (H, Ex. 3.5, but using a different construction):
Example: Construct a regular expression from the following DFA that accepts all strings in \( \{0, 1\}^* \) containing the substring 00:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>q</td>
<td>p</td>
</tr>
<tr>
<td>*q</td>
<td>q</td>
<td>q</td>
</tr>
</tbody>
</table>

Example: Construct a regular expression from the 3-state DFA from Tutorial 1 that accepts all strings in \( \{0, 1\}^* \) representing (binary) numbers evenly divisible by 3.

Example: Construct an \( \epsilon \)-NFA equivalent to the language \((0 + 1)^*1(0 + 1)\) (H, Ex. 3.8).

Example: Construct a DFA (by the modified subset construction) equivalent to the resulting \( \epsilon \)-NFA.

******

Recognising that languages are not regular (H, 4.1).

The pumping lemma for regular languages (H, 4.1.1): Let \( L \) be a regular language. Then there exists a constant \( n \geq 1 \) such that, for every string \( w \) in \( L \) with \( |w| \geq n \), we can write \( w = xyz \) in such a way that:

1. \( |xy| \leq n \) (the initial section is not too long).
2. \( y \neq \epsilon \) (the string to pump is not empty).
3. For all \( k \geq 0 \), the string \( xy^kz \) is in \( L \) (the string \( y \) may be pumped any number of times, including 0, and the resulting string is still in \( L \)).

Proof outline: Every string in \( L \) with more symbols than states in the DFA for \( L \) must cause some state to repeat in being recognised, which allows any number of cycles through that state.

Applications. The following languages are not regular:

1. All strings in \( \{0, 1\}^* \) with an equal number of 0s and 1s (H, Ex. 4.2). If this language \( L \) is regular, there exists \( n \) satisfying the above condition. Let \( w = 0^n1^n \). Clearly \( |w| \geq n \).
   We can then write \( w \) as \( xyz \) with \( |xy| \leq n \), where \( y = 0^m \) for \( m \geq 1 \). By the lemma, \( xz \in L \) (\( k = 0 \)). But \( xz \) has \( n - m \) 0s and \( n \) 1s, which is a contradiction.
2. \( \{0^n1^n \mid n \geq 1 \} \).
3. The set of balanced parenthesis strings.
4. \( \{0^n1^m \mid n \leq m \} \).
5. \( \{0^n \mid n \text{ is a perfect square} \} \)
Closure properties of regular languages. Regular languages are closed under union, concatenation and iteration, complement, intersection, and difference. The first three are immediate from regular expression construction (H, Thm 4.3). Complement follows from reversing final and nonfinal states in a DFA (H, Thm 4.5). Intersection (and union) follow from a product construction on DFAs (H, Thm 4.8).

Example: Construct the DFA that accepts the language with an even number of 0s and an even number of 1s as the intersection of two simpler languages.

Regular languages are also closed under reversal — if $L$ is regular, then $L^R = \{ w^R \mid w \in L \}$ is regular — either by DFA construction or induction on regular expression definition (H, Thm 4.11).

Example: The set of strings in \{0, 1\}∗ with different numbers of 0s and 1s is not regular. This is harder (not possible?) to prove using the pumping lemma.

Decision problems for regular languages (H, 4.3). The following problems are decidable:

1. (Emptiness) Is the language empty? (Is there a path from the initial state to a final state? Use induction on the regular expression.)

2. (Finiteness) Is the language finite? (Let $n$ be the pumping-lemma constant. Test all strings of length between $n$ and $2n - 1$ for membership in $L$. If we find even one such string, then $L$ is infinite. The reason is that the pumping lemma applies to such a string, and it can be “pumped” to show an infinite sequence of strings are in $L$.)

3. (Membership) Does a given string belong to the language? (Apply the DFA for the language to the string.)

4. (Equality) Do two language descriptions define the same language? (A more complex problem, see H, 4.4)

Each DFA has a unique (up-to-renaming) equivalent minimal-state DFA (H, 4.4).