3515ICT: Theory of Computation

Regular languages

Notation and concepts concerning alphabets, strings and languages, and identification of languages with problems (H, 1.5).

Regular expressions (H, 3.1, 3.3–4; S, 1.3)

Definitions of regular expressions and regular languages: A regular expression over a finite alphabet Σ is composed from the expressions $\emptyset$, ε, a ($a \in \Sigma$), S ($S \subseteq \Sigma$) using union ($E_1 + E_2$ or $E_1 \cup E_2$), concatenation (E_1E_2), iteration (E^*, where $E^* = \{\varepsilon\} \cup E \cup E^2 \cup \cdots$), and parentheses. Precedence: Iteration > concatenation > union. Definition of the language $L(E)$ defined by a regular expression E. Abbreviations: $(a - z) = \{a, b, \ldots, z\}$, $E? = (E + \varepsilon)$, $E^+ = EE^*$.

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); Java identifiers; strings in $\{0,1\}^*$ denoting integers divisible by 3 (best postponed until finite automata have been introduced).

The same 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), or $(0+1)^*$ and $0^*(10^*)^*$.

Every finite language is regular. Informal proof and proof by induction (on size of language, and length of string in language).

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); $\{ a^nb^n \mid n \geq 0 \}$; legal arithmetic expressions using the identifier a, the operator $+$, and left and right parentheses.

Applications of regular expressions: compilers, editors, Web servers, information retrieval (H, 3.3).

Algebraic laws can be used to simplify regular expressions, treated lightly (H, 3.4.1–3.4.5): $\emptyset + L = L + \emptyset = L$, $\varepsilon L = L\varepsilon = L$, $\emptyset L = L\emptyset = \emptyset$, $L+M = M+L$, $(L+M)+N = L+(M+N)$, $(LM)N = L(MN)$, $L(M+N) = LM+LN$, $(L+M)N = LM+LN$,$L+L = L$, $(L^*)^* = L^*$, $(L^*M^*)^* = (L+M)^*$, $\emptyset^* = \varepsilon$, $\varepsilon^* = \varepsilon$, $L^+ = LL^* = L^*L$, $L^* = L^+ + \varepsilon$, $[L] = L + \varepsilon$.

Kozen's axiom system can be used to prove equivalence of regular expressions (omitted).

Deterministic finite automata (H, 1.1, 2.1–2; S, 1.1)

Example: Finite automaton for the regular expression $(1 + 01)^+$.

Definition of (deterministic) finite automata (DFA). A *DFA* has a set of states Q, an alphabet Σ, an initial state $q_0 \in Q$, a transition function $\delta : Q \times \Sigma \to Q$, and a set of accepting (or final) states $F \subseteq Q$. Often a DFA is drawn as a transition diagram, with each state having a transition for each element of Σ. The transition function can also be presented as a table, with rows indexed by states and columns by symbols. (In programs, the transition function δ of a DFA is represented by a state transition matrix.)

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

Definition of the extended transition function $\delta^* : Q \times \Sigma^* \to Q$: $\delta^*(q, \varepsilon) = q$, $\delta^*(q, as) = \delta^*(\delta(q, a), s)$.

A string $s \in \Sigma^*$ is *accepted* by a DFA if s leads from the initial state to an accepting state, *i.e.*, if $\delta^*(q_0, s) \in F$. The *language* $L(M)$ accepted, or recognised, by a DFA M is the set of strings in Σ^* 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 are very efficient.

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

Nondeterministic finite automata (H, 2.35; S, 1.2)

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 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 automaton (NFA). An *NFA* has a set of states Q, an alphabet Σ, an initial state $q_0 \in Q$, a transition function $\delta : Q \times \Sigma_\epsilon \to 2^Q$, and a set of accepting states $F \subseteq Q$. Here, $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$.

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

Example: NFAs for $L(01^* + 0^*1)$ and $L(01^*0^*1)$.

Example: NFA for decimal numbers (H, Ex. 2.16).

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

1. $\delta^*(q, \varepsilon) = E(\{q\})$, for $q \in Q$
2. $\delta^*(q, sa) = \bigcup_{p \in \delta^*(q,s)} E(\delta(p, a))$, for $q \in Q$, $s \in \Sigma^*$, $a \in \Sigma$

That is, $\delta^*(q, s)$ is the set of states that the machine *can* reach from state q with input string s, following ϵ-transitions without reading input at any time.

Here, $E(S)$ is the set of all states that can be reached from S using a sequence of zero or more ε-transitions. Formally, $E(S)$ is the least set S' such that $S \subseteq S'$ and, for all $q \in S'$, $\delta(q, \varepsilon) \in S'$. (This set can be computed by breadth-first graph traversal.)

Example (M, Ex.4.6): Consider NFA for $L((0^*101^* + 1^*000*)^*)$. Construct $\delta^*(q_0, 010)$.

A string $s \in \Sigma^*$ is *accepted* by an NFA if $\delta^*(q_0, s) \cap F \neq \emptyset$, *i.e.*, if some sequence of transitions leads from q_0 to an accepting state. The language $L(M)$ accepted, or recognised, by an NFA M is the set of strings in Σ^* that are accepted by M.

NFAs may have *many* fewer states than the corresponding DFA. Consider the above NFA for $\{\, s \in \{0,1\}^* \mid |s| \geq N$ and the Nth symbol from the right is $1 \,\}$. This has $N+1$ states; the corresponding DFA has 2^N states.

Equivalence of DFAs and NFAs by subset construction (H, 2.3.5–6; S, 1.2).

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. For each set S in Q_D and symbol a in Σ,

   $$\delta_D(S, a) = \bigcup_{p \in S} E(\delta_N(p, a)).$$

   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 Nth symbol from the right is 1 (H, Ex. 2.13).

Example: Construction of a DFA equivalent to the natural NFA for $L(0^*(01)^*0^*)$.

Example: Two-stage construction of a DFA equivalent to the four-state ε-NFA with $\delta = \{(q_0, \varepsilon, q_1), (q_0, \varepsilon, q_2), (q_1, 0, q_1), (q_1, 1, q_{3)}, q_2, 0, q_3), (q_2, 1, q_2)\}$ and $F = \{q_3\}$ (M, Ex. 5.6).

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

Kleene's theorem

A language is regular (has a regular expression) if and only if it is recognised by some DFA if and only if it is recongised by some NFA.

See Fig. 3.1 (H., p.91). See also Theorem 1.54 and Lemma 1.60 (S, 1.3). Our proof follows Sipser's proof, which seems to be the simplest.

1. Every DFA is equivalent to some regular expression, by the state elimination construction (H, 3.2.2; S, Lemma 1.60).
2. Every regular expression is equivalent to some NFA, by a simple inductive construction (H, 3.2.3; S, Lemma 1.55).

3. Every NFA is equivalent to some DFA, by the subset construction (H, 2.3.5; S, Theorem 1.39).

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):

	0	1
$\rightarrow$ p	q	p
*q	q	q

Example: Construct a regular expression from the following DFA that accepts all strings in $\{0, 1\}^*$ containing the substring 00:

	0	1
$\rightarrow$ p	q	p
q	r	p
*r	r	r

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

Example: Construct an NFA equivalent to the regular expression $(0 + 1)^*1(0 + 1)$ (H, Ex. 3.8). Construct a DFA equivalent to the resulting NFA.

Pumping lemma for regular languages (H, 4.1; S, 1.4)

This lemma allows us to prove that some languages are *not* regular.

The pumping lemma for regular languages (H, 4.1.1): Let L be a regular language. Then there exists a constant $p \geq 1$ such that, for every string w in L with $|w| \geq p$, there exist strings x, y, z such that $w = xyz$ and:

1. $|xy| \leq p$ (the initial section is not too long).

2. $|y| \geq 1$ (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 recongised, which allows any number of cycles through that state.

Applications (H, 4.1.2; S, 1.4). The following languages are not regular:

1. $\{ 0^n1^n \mid n \geq 1 \}$. See the next item. (S, Ex. 1.73)

2. All strings in $\{0, 1\}^*$ with an equal number of 0s and 1s. Suppose L is regular Let p be the pumping lemma for L. Choose $w = 0^p1^p$. Clearly $|w| \geq p$. Let $w = xyz$, where $|xy| \leq p$ and $|y| \geq 1$. Then $y = 0^m$ for some $m \geq 1$. By the pumping lemma, $xz = xy^0z \in L$. But xz has $n - m$ 0s and n 1s, with $m \geq 1$, so $xz \notin L$. This is a contradiction, so L is not regular. (S, Ex. 1.75; H, Ex. 4.2)

3. $\{ 0^n1^m \mid n \leq m \}$.

4. The set of balanced parenthesis strings.
5. The set of arithmetic expressions over a and b using operators + and * and parentheses. This set is proved non-regular in a similar way to the previous example. (Note that the set of arithmetic expressions *without* parentheses *is* regular.)
6. $\{\, ww \mid w \in \{0,1\}^* \,\}$ (S, Ex. 1.75)
7. $\{\, 1^{n^2} \mid n \geq 0 \,\}$ (S, Ex. 1.76)

Closure properties of regular languages (H, 4.2; S, 1.2)

The class of regular languages is closed under union, concatenation and iteration, complement, intersection, difference and reversal. The first three are immediate from regular expression construction (H, Thm 4.3). Complement ($\bar{L} = \Sigma^* - L$) follows from reversing the roles accepting and non-accepting states in a DFA for L (H, Thm 4.5). Intersection follows from union and complement using De Morgan's law, or by constructing a DFA for $L_1 \cap L_2$ from the DFAs for L_1 and L_2 using a product construction (H, Thm 4.8). This product construction can also be used to show that regular languages are closed under union and difference. Closure under difference also follows from $L_1 - L_2 = L_1 \cap \bar{L_2}$).

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.

Example: Construct the DFA that accepts the set of binary strings that are evenly divisble by 6 as the intersection of the sets of binary strings that are evenly divisible by 2 and by 3.

The class of regular languages is closed under symmetric difference ($L_1 \Delta L_2 = (L_1 - L_2) \cup (L_2 - L_1)$).

The class of regular languages is also closed under reversal — if L is regular, then $L^R = \{\, w^R \mid w \in L \,\}$ is regular — either by NFA construction (reverse the transitions and the roles of initial and accepting states), or by induction on their regular expression definition (H, Thm 4.11).

Example: The language $\{\, 0^m 1^n \mid m \neq n \geq 0 \,\}$ is not regular. This is not possible to prove using the pumping lemma alone. (Try.) However, suppose the language $\{\, 0^m 1^n \mid m \neq n \geq 0 \,\}$ were regular. Then

$$\begin{array}{rcl} \{\, 0^m 1^m \mid m \geq 0 \,\} & = & \{\, 0^m 1^n \mid m = n \geq 0 \,\} \\ & = & \{\, 0^m 1^n \mid m, n \geq 0 \,\} - \{\, 0^m 1^n \mid m \neq n \geq 0 \,\} \\ & = & L(0^* 1^*) - \{\, 0^m 1^n \mid m \neq n \geq 0 \,\} \end{array}$$

would be regular. But it's not, so neither is $\{\, 0^m 1^n \mid m \neq n \geq 0 \,\}$.

Example: Prove the set of strings in $\{0,1\}$ with *different* numbers of 0s and 1s is not regular. This is also not possible to prove using the pumping lemma alone. (Try.)

Decision problems for regular languages (H, 4.3)

What questions about regular languages can be answered and how long does it take to answer them? In answering such questions, we are free to convert the given description of the language(s) to any other representation of the language, though this conversion takes time.

The following problems are all decidable for regular languages.

1. (Emptiness) Is a given language empty, *i.e.*, does $L = \emptyset$?

   (a) Use structural induction on a regular expression for L: $\emptyset$ is empty; ε and $a \in \Sigma$ are not empty; $E_1 + E_2$ is empty iff both E_1 and E_2 are empty; $E_1 E_2$ is empty iff either E_1 or E_2 is empty; E^* is not empty.

   (b) Use reachability in a DFA for L: L is empty iff there is no path from the initial state to an accepting state. This can be checked by a breadth-first traversal of the DFA from q_0.

2. (Finiteness) Is the given language finite?

   (a) Use structural induction on a regular expression for L. (More complex than finiteness.)

   (b) Let n be the pumping-lemma constant for L. Then L is finite iff it contains no string whose length is between n and $2n - 1$. This requires testing $O(|\Sigma|^{2n})$ strings for membership in L.

      (If L has a string whose length is at least n, then, by the pumping lemma, L is infinite. If the shortest string w whose length is at least n is at least $2n$, then by the pumping lemma again, $w = xyz$, where $|xy| \leq n$ and $|y| \geq 1$, $xz \in L$. But $n \leq |xz| < |xyz| = |w|$, which is a contradiction.)

3. (Membership) Does a given string belong to the given language, *i.e.*, does $w \in L$?

   Convert the representation of L to a DFA M, if necessary. Then apply M to w, and see whether M accepts w, requiring time $O(|w|)$. But the time to convert a regular expression (or NFA) to a DFA can be *exponential* in the size of the regular expression (or NFA).

   We can also simulate an NFA directly, maintaining the current set of possible states, requiring time $O(|w| \times |Q|^2)$, which is what is done in practice.

4. (Equality) Do two language descriptions define the same language, *i.e.*, does $L_1 = L_2$?

   Clearly, $L_1 = L_2$ iff $L_1 \subseteq L_2$ and $L_2 \subseteq L_1$. But $L_1 \subseteq L_2$ iff $L_1 \cap \bar{L_2} = \emptyset$. We can compute the regular language $L_1 \cap \bar{L_2}$ and we can decide whether or not it is empty. And similarly for $L_2 \subseteq L_1$. So the problem is decidable.

   A more efficient decision procedure for equality of regular languages follows from the discussion below.

Myhill-Nerode theorem and minimisation of DFAs

Let L be any subset of Σ^*. For $u, v \in \Sigma^*$, define $u \equiv_L v$ if and only if $uw \in L$ iff $vw \in L$, for any w in Σ. The equivalence relation $\equiv_L$ induces a partition of Σ^* into disjoint components (or equivalence classes).

Examples: Construct the equivalence classes for the following languages:

1. $L = \{\, 0^n 1^n \mid n \geq 0 \,\}$

2. $P = \{\, w \in \{0,1\}^* \mid w = w^R \,\}$

3. $T = \{\, w \in \{0,1\}^* \mid w$ represents a binary integer divisible by 3 $\}$

Theorem (Myhill-Nerode): A language L is regular if and only if the number of equivalence classes of $\equiv_L$ is *finite*. If L is regular, the number of equivalence classes of $\equiv_L$ equals the number of states in the minimal DFA that recognises L. (S, Problems 1.51–1.52)

Corollary: Every regular language has a *unique* minimal DFA (up to state renaming). (H, 4.4)

Clearly, the Myhill-Nerode theorem can be used to prove languages are not regular.

Given a DFA for a regular language L, we can construct the minimal DFA for L either by first computing the set of distinguishable pairs (H, 4.4) or by computing the set of equivalence classes directly (Martin, 5.1–5.2). The second algorithm can be expressed as follows.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for a regular language L as above. (If M has any unreachable states, delete them.)

For every partition P of Q, define $\mathit{splits}(C_0, C_1, a) \equiv \exists\, p, q \in C_0$ s.t. $\delta(p, a) \in C_1$ and $\delta(q, a) \notin C_1$, for $C_0, C_1 \in P$ and $a \in \Sigma$.

Now, compute the set P of equivalence classes of states in Q.

```
$P := \{Q \setminus F,\ F\}$;
**while** $\exists\, C_0, C_1 \in P,\ a \in \Sigma$ s.t.$\mathit{splits}(C_0, C_1, a)$ **do**
  **let** $C_0, C_1 \in P,\ a \in \Sigma$ s.t $\mathit{splits}(C_0, C_1, a)$;
  $C_0' := \{\, p \in C_0 \mid \delta(p, a) \in C_1 \,\}$;
  $P := P \setminus \{C_0\} \cup \{C_0 \setminus C_0',\ C_0'\}$
**end while**
```

(Of course, in practice, we don't repeat the search for $C_0, C_1 \in P,\ a \in \Sigma$ s.t. $\mathit{splits}(C_0, C_1, a)$.)

Finally, construct the minimal DFA M_L for L from P.

Let $M_L = (Q', \Sigma, \delta', q_0', F')$, where $Q' = \{\, [q]_P \mid q \in Q \,\}$, $q_0' = [q_0]_P$, $F' = \{\, [q]_P \mid q \in F \,\}$, and $\delta'([p]_P, a) = [q]_P$, where $\delta(p, a) = q$, for each $p \in Q$ and $a \in \Sigma$.

Examples: Hopcroft *et al.*, Fig. 4.8; Martin, Fig. 5.3(a).

Because the minimal DFA for a regular language is unique, two regular languages are equal iff their minimal DFAs are identical (up to state renaming).

Slightly more efficiently, two regular languages are equal iff the initial states of their DFAs are equivalent after taking the union of the two DFAs (and choosing one initial state as the initial state of the union). This avoids computing the minimal DFA.

'H' or 'IALC' or 'Hopcroft *et al.*' refers to *Introduction to Automata Theory, Languages and Computation, Second Edition*, by J.E. Hopcroft,, R. Motwani and J.D. Ullman, Addison-Wesley, 2001.

'S' or 'Sipser' refers to *Introduction to the Theory of Computation, Second Edition*, by M. Sipser, Thomson, 2006.

'M' or 'Martin' refers to *Introduction to Languages and the Theory of Computation, Third Edition*, by J. Martin, McGraw-Hill, 2003.