COMPUTATION: DAY 2

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Wednesday, January 28, 2026.

1. Deterministic Finite Automata

1.1 I review our definitions for a finite automata. We will soon introduce non-determinism and will specify that this sort of finite automata is a *deterministic finite automata* (DFA).

Definition 1.1 (Deterministic Finite Automata)**.** Given a finite automata $M = \langle\, Q, \Sigma, \delta, q_0, F \,\rangle$ and string $\sigma = \sigma_1\sigma_2 \ldots \sigma_k$, where $\sigma_i \in \Sigma$, a *computation* $q_0 \xrightarrow{s} q_k$ is a sequence of states from Q,

$$\langle\, q_0, q_1, \ldots, q_k \,\rangle$$

such that $\delta(q_{i-1}, \sigma_i) = q_i$ for all appropriate i.

Definition 1.2. Given a finite automata M as above, the *language* of M, $\mathcal{L}(M)$ is the set of stings,

$$\mathcal{L} = \{\, \sigma \in \Sigma^* \mid q_0 \xrightarrow{s} q_f \text{ with } q_f \in F \,\}.$$

Definition 1.3 (Regular Language)**.** A set $S \subseteq \Sigma^*$ is a *regular language* if there exists a finite automata M such that $\mathcal{L}(M) = S$.

Lemma 1.1. The languages $\emptyset$ and $\{\varepsilon\}$ are regular languages.

Proof: See figure 1a and figure 1b. □

1.2 The class of regular languages is a Kleene Algebra. It in fact has further properties such as closed by complement and intersection, which are not required or defined for a Kleene Algebra.

Theorem 1.1. If $S \subseteq \Sigma^*$ is regular, so is its complement $\bar{S} = \Sigma^* \setminus S$.

Proof: Since S is regular, let $M_S = \langle\, Q, \Sigma, \delta, q_0, F \,\rangle$ be the finite automata such that $\mathcal{L}(M_S) = S$. Define $M_{\bar{S}} = \langle\, Q, \Sigma, \delta, q_0, Q \setminus F \,\rangle$. Then $\mathcal{L}(M_{\bar{S}}) = \bar{S}$. □

(A) (B)

FIGURE 1. The languages $\emptyset$ and $\{\varepsilon\}$ are regular.

Theorem 1.2. If $S, T \subseteq \Sigma^*$ are regular languages, so are $S \cup T$ and $S \cap T$.

1.3 To prove that the union or intersection of two regular languages is regular, we build a finite automaton that tracks the state transitions of both automata at once. This construction uses pairs of states, so the resulting machine may have many states — up to the product of the sizes of the original automata. While a smaller automaton may exist for particular languages, the product construction works for all of them, and therefore provides a general proof of closure.

Proof: This is a proof by example, and I owe this nice example and the graphic depiction to Wayne Goddard of Clemson University. The languages,

$$\begin{aligned} L_1 &= \{\, \sigma \in \{\,0,1\,\}^* \mid \text{ there are an even number of 1's in } \sigma \ \} \\ L_2 &= \{\, \sigma \in \{\,0,1\,\}^* \mid \text{ there are an even number of 0's in } \sigma \ \} \end{aligned}$$

are regular, with the machine M_1 in figure 2a such that $\mathcal{L}(M_1) = L_1$ and machine M_2 in figure 2b such that $\mathcal{L}(M_2) = L_2$.

1.4 We simulate the pair of machines with a single machine whose state set is the cartesian product of the state sets of the two machines,

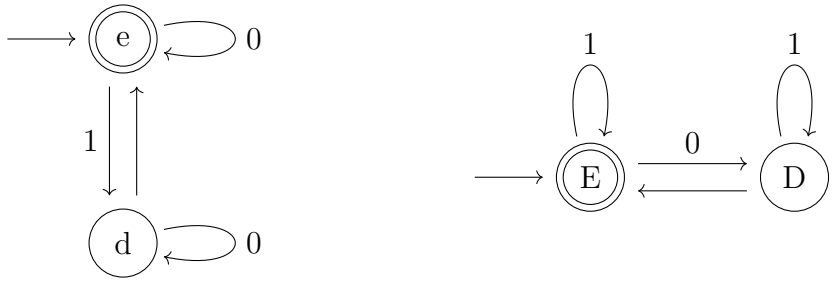
$$Q = Q_1 \times Q_2.$$

Then for any $q \in Q$, it is written as $q = (q_1, q_2)$, and means that in a parallel execution, M_1 would be in q_1 and M_2 would be in q_2.

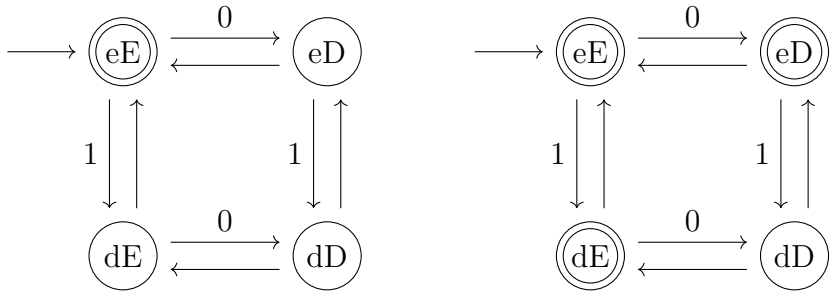
1.5 The start state is $q_0 = (q', q'')$, where q' is the start state for M_1 and q'' is the start state for M_2. The transition function folds into a single machine the transitions of the two component machines,

$$\delta(q) = \delta((q_1, q_2)) = (\delta_1(q_1), \delta_2(q_2)).$$

1.6 The accepting states are those derived from the accepting states of the two automata. If it is the union language, the for a final state $q = (q_1, q_2)$ it is sufficient



(A) Even ones (B) Even zeros



(C) Both even (D) Either even

FIGURE 2. Even and odd parity of 1's and 0's

that either $q_1 \in F_1$ or $q_2 \in F_2$ (see figure 2d),

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2).$$

If it is the intersection, then (see figure 2c),

$$F = F_1 \times F_2.$$

The general result follows from the uniformity of this construction. □

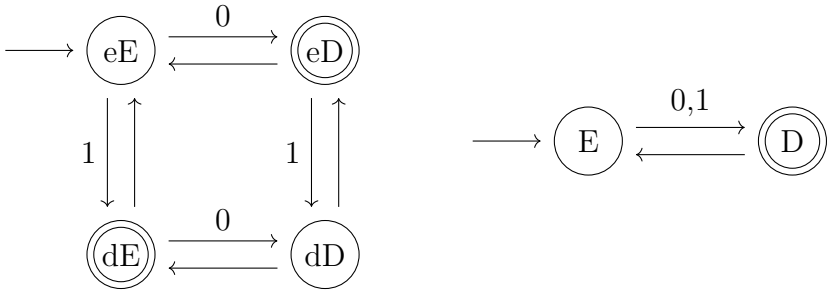
1.7 The product construction is automatic and applies to any pair of automata. It might define more states than is needed. For instance, this four state construction can recognize the language,

$$\{\, \sigma \in \{\, 0, 1 \,\} \,|\, \text{parity of 1's differs from parity of 0's}$$

by choosing final states as in figure 3a. However it can also recogiznized as in rigure 3b using only two states.

1.8 Proof: Let l by the length of σ, l_1 be the number of 1's in σ, and l_0 be the number of 0's in σ. Then from,

$$l = l_0 + l_1$$



(A) Using 4 states (B) Using 2 states

FIGURE 3. Parity of 1's and 0's differ

we conclude that if l is odd, then one of l_0 and l_1 is odd and the other even, so σ is in the language. However if l is even then l_0, l_1 are both even or both odd, so σ is not in the language. So the language is the same as the language of the parity of the length of σ. □

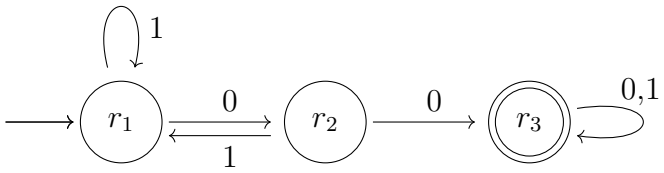
1.9 Another example is giving in figure 4. Let S_1 be the set of strings containing 00. It is recognized by the automata shown in figure4a. Let and S_2 be the set of strings ending in a 1. It is recognized by the automata show in figure 4b The product construction is given accepting the language $S_1 \cap S_2$ is shown in figure 4c.

2. Nondeterministic Finite Automata

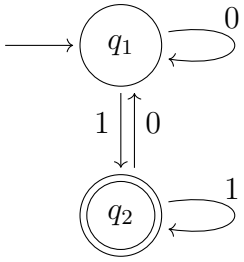
2.1 Up to this point, the finite automata as studied provide a simple and mechanistic computational model: each input induces exactly one computation, which either accepts or rejects. These are deterministic finite automata (DFA), and we used this determinism to prove closure under union and intersection by constructing a product automaton. However, constructions for concatenation and Kleene star are not nearly as direct in the deterministic setting.

2.2 A different approach was introduced by Rabin and Scott[1], who generalized finite automata to allow several possible next states for a given input. These nondeterministic finite automata (NFAs) may branch into many computations at once, accepting whenever at least one branch accepts. Although NFAs seem more powerful than DFAs, we will later prove that the two models recognize exactly the same class of

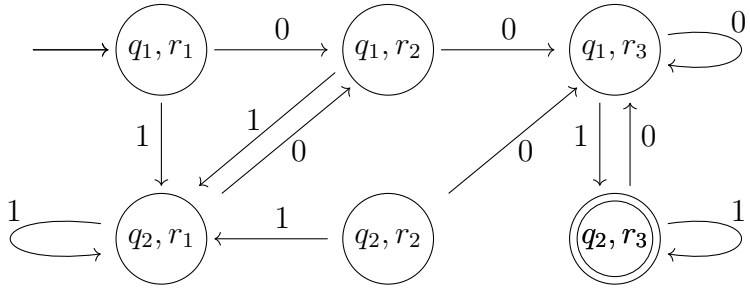
[1]M. O. Rabin and D. Scott, "Finite Automata and Their Decision Problems," IBM Journal of Research and Development, vol. 3, no. 2, pp. 114–125, 1959.



(A) Finite automata for stings containing 00.



(B) Finite Automata for strings ending in a 1.



(C) Result of the product construction

FIGURE 4. Strings both containing 00 and ending in a 1

regular languages. In the meantime, NFAs serve as a flexible specification mechanism that makes closure constructions — such as concatenation — considerably more transparent.

Definition 2.1 (Nondeterministic finite automata)**.** A *nondeterministic finite automata* is a five tuple $\langle Q, \Sigma, \delta, q, F \rangle$ where Q is a finite set of states, Σ is a finite alphabet, $q \in Q$ is the start state and $F \subseteq Q$ is the set of accepting or final states. The transition function,

$$\delta \ : \ Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$$

is defined on the augmented alphabet,

$$\Sigma_\varepsilon = \Sigma \cup \{ \dot{\varepsilon} \}$$

where the symbol $\dot{\varepsilon}$ is not in Σ and functions as the empty string in Σ^*. The set $\mathcal{P}(Q)$ is the set of subsets of Q.

Definition 2.2. A computation $q_1 \xrightarrow{\sigma} q_k$ of nondeterministic machine M on a string $\sigma = \sigma_1\sigma_2 \ldots \sigma_k$, $\sigma_i \in \Sigma_\varepsilon$, is a sequence of states from Q,

$$\langle\, q \,\rangle = q_0, q_1, \ldots, q_k$$

such that $q_i \in \delta(q_{i-1}, \sigma_i)$ for all appropriate i.

2.3 Unlike a deterministic automata, the nondeterministic automata has a choice from the set $\delta(q_{i-1}, \sigma_i)$. The notion demands that $\delta(q_{i-1}, \sigma_i) \neq \emptyset$.

2.4 When computing in $\sigma \in \Sigma^*$, a σ' can used which is the insertion of symbols $\dot{\varepsilon}$ anywhere in the string σ. We formalize this by saying $\sigma' = \sigma$ when, for $\sigma' \in \Sigma_\varepsilon^*$ and $\sigma \in \Sigma^*$, σ results by deleting all $\dot{\varepsilon}$ from σ'.

Definition 2.3. The language $\mathcal{L}(M)$ of a nondeterministic finite automata M with start state q_0 and final state set F is the,

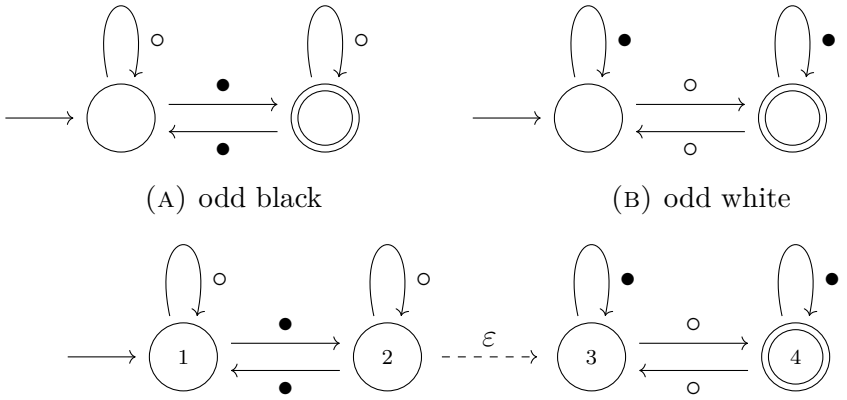
$$\mathcal{L}(M) = \{\, \sigma \in \Sigma^* \mid \exists\, \sigma' \in \Sigma_\varepsilon^*, q \in F, \text{ s.t. } \sigma' = \sigma \text{ and } q_0 \xrightarrow{\sigma'} q \,\}$$

2.5 The asymmetry of accepting and non-accepting. A deterministic machine has only one possible computation path for any given input. This path is uniquely determined by repeatedly applying the transition function, with no ambiguity. Since there are no alternative computations to consider, the outcome of that single computation is decisive: if it reaches an accepting state, the input is in the language; otherwise, the input is not.

2.6 A nondeterministic machine may have several possible computation paths for the same input. If at least one of these paths accepts, then the input is considered accepted and therefore belongs to the language. However, a rejecting path does not prove anything by itself, because some other path might accept. In the setting of finite automata, the input can only be rejected after all possible computation paths have been examined and all fail to accept.

3. Intuitive introduction to NFA's

3.1 We will introduce, one at a time, two features of NFAs. The first are transitions of the form $\delta(q, \dot{\varepsilon}) = q'$, called ε-edges, whice are drawn in diagrams as edges labelled ε. Intuitively, when the automaton in state q, it can "guess" that next state should be



(A) odd black

(B) odd white

(C) concatenation finite automata

FIGURE 5. Non-deterministic solution with an ε move

q' and take this transition without consuming an input symbol. The second feature is states for which $|\delta(q, \sigma)| \neq 1$, representing nondeterministic choices on the same input symbol. First we consider ε-edges and show how they can be used to construct an NFA that recognizes the concatenation of two languages, each recognized by an NFA.

3.2 The motivating example will be the concatenation of the two languages,

$$\begin{aligned} S &= \{\, \sigma \in \{\, \bullet, \circ \,\} \mid \text{ string } \sigma \text{ has an odd number of } \bullet \,\} \\ T &= \{\, \sigma \in \{\, \bullet, \circ \,\} \mid \text{ string } \sigma \text{ has an odd number of } \circ \,\} \\ R &= S \circ T \end{aligned}$$

Consider the string,

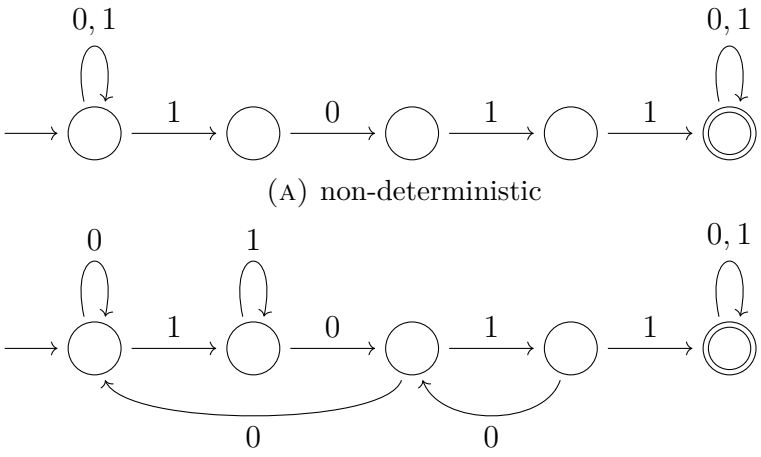
$$\rho = \bullet \circ \circ \bullet \circ \bullet \circ \bullet \circ \circ \bullet$$

It is in R because it can be parsed as a prefix in S and the remainder in R. It can be parsed as such in exactly two ways.

$$\rho = (\bullet\circ)(\circ \bullet \circ \bullet \circ \bullet \circ \circ \bullet) = (\bullet \circ \circ \bullet \circ\bullet)(\circ \bullet \circ \circ \bullet)$$

The question for a finite state automata is how is it to know where to split ρ between S and T?

3.3 We construct an NFA M_R for $R = S \circ T$ (see figure 5c) by connecting the two NFAs M_S for S (see figure 5a) and M_T for T (see figure 5b) by ε-edges. Specifically we connect with an ε-edge from each final state of M_S to the initial state of M_T.



(A) non-deterministic

(B) deterministic

FIGURE 6. Matching the pattern 1011, using non-determinism

3.4 The automaton can now nondeterministically choose where to split the input by following an ε-edge. If it chooses correctly, the portion of the string up to the jump is recognized by M_S, as indicated by reaching a final state, and the remaining portion is recognized by M_T as indicated by a successful path from its start state to one of its final states.

3.5 Conversely, any string accepted by this composite machine must be in the language, because it reached a final state of M_T and necessarily passed through a final state of M_S. In this way, the NFA accepts exactly the concatenation $S \circ T$.

Lemma 3.1. If $S, T \subseteq \Sigma^*$ are languages accepted by NFAs, so is the language $S \circ T$.

3.6 Nondeterminism: multiple edges. The non-determinist finite automata we are imagining can also have multiple outgoing edges from a state labeled with the same letter. In this case, the non-deterministic choice is made on which edge to take.

3.7 In figure 6a is a non-deterministic machine for matching the pattern 1011 anywhere in a string. The assuming the pattern is in the string, the machine nondeterministically guesses its location. It loops at the start state until reaching the start of the pattern and then proceeds with the four state transitions, ending in an absorbing final state.

3.8 If the pattern is not in the string, any attempt to pass from the start to the final state will get to a failure to pass to the next state, formally said, zero outgoing arrows for the letter, and that branch of the computation will fail.

3.9 A deterministic machine for the same language is given in figure 6b. It might come as a surprise how tricky the transition function is. However, it is true that for any language recognized by an NFA there is an DFA recognizing the same language.

Theorem 3.1. A language is accepted by some DFA if and only if it is accepted by some NFA.

3.10 Once this is proved we have,

Corollary 3.1. If S and T are regular languages so is $S \circ T$.