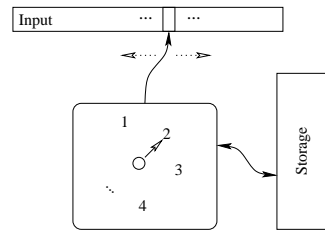


AUTOMATA (FINITE OR NOT)

- Generally any automaton
 - has a **finite-state control**
 - scans the input one symbol at a time
 - takes an action based on the currently scanned symbol and the current state
 - the action may yield a different current state
 - may make use of extra storage
- A **finite automaton** scans the input from left to right only and uses no additional storage
 - it cannot go back in the input
 - it can only remember (using the finite state control) a **finite amount of information** about the already seen input



DETERMINISTIC FINITE AUTOMATA

- A deterministic finite automaton is a tuple $M = (K, \Sigma, \delta, s, F)$
 - $K \Rightarrow$ finite set of states
 - $\Sigma \Rightarrow$ input alphabet
 - $F \subseteq K \Rightarrow$ set of final states
 - $s \in K \Rightarrow$ initial state
 - $\delta : K \times \Sigma \rightarrow K \Rightarrow$ transition function
- Configuration: $c \in K \times \Sigma^*$
- Yields in one step: $(q, aw) \vdash_M (q', w)$ iff $a \in \Sigma$ and $\delta(q, a) = q'$
 - $\vdash_M^* \Rightarrow$ reflexive and transitive closure
- w is accepted by M iff $\exists q \in F : (s, w) \vdash_M^* (q, \varepsilon)$
- The language accepted by M : $\mathcal{L}(M) = \{w \in \Sigma^* : \exists q \in F : (s, w) \vdash_M^* (q, \varepsilon)\}$

NONDETERMINISTIC FINITE AUTOMATA

- A **nondeterministic** finite automaton is a tuple $M = (K, \Sigma, \Delta, s, F)$
 - $K \Rightarrow$ finite set of states
 - $\Sigma \Rightarrow$ input alphabet
 - $F \subseteq K \Rightarrow$ set of final states
 - $s \in K \Rightarrow$ initial state
 - $\Delta \subseteq K \times (\Sigma \cup \{\varepsilon\}) \times K \Rightarrow$ transition relation
- Configuration: $c \in K \times \Sigma^*$
- Yields in one step: $(q, aw) \vdash_M (q', w)$ iff $a \in \Sigma \cup \{\varepsilon\}$ and $(q, a, q') \in \Delta$
 - $\vdash_M^* \Rightarrow$ reflexive and transitive closure
- w is accepted by M iff $\exists q \in F : (s, w) \vdash_M^* (q, \varepsilon)$
- The language accepted by M : $\mathcal{L}(M) = \{w \in \Sigma^* : \exists q \in F : (s, w) \vdash_M^* (q, \varepsilon)\}$

DETERMINISM VERSUS NONDETERMINISM

- Languages accepted by finite automata?
 - of finite strings for sure
- Deterministic FA \Rightarrow special case of nondeterministic FA
- In fact the two kind of finite automaton accept the same languages
- $M = (K, \Sigma, \Delta, s, F) \Rightarrow M' = (K', \Sigma, \delta', s', F')$ such that $\mathcal{L}(M) = \mathcal{L}(M')$
 - $K' = 2^K$
 - Let $E(q)$ be the closure of $\{q\}$ under $\{(p, r) : (p, \varepsilon, r) \in \Delta\}$
 - $s' = E(s)$
 - $F' = \{Q \subseteq K : Q \cap F \neq \emptyset\}$
 - $\delta'(Q, a) = \bigcup \{E(p) : p \in K, (q, a, p) \in \Delta \text{ for some } q \in Q\}$
- (proof on p. 71)
- DFA are more efficient, potentially difficult to understand, and often considerably larger (how much larger?)

CLOSURE PROPERTIES

- $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$. Can we construct $M = (K, \Sigma, \Delta, s, F)$ such that
 - $\mathcal{L}(M) = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$ (closure under union)?
 - $\mathcal{L}(M) = \mathcal{L}(M_1)\mathcal{L}(M_2)$ (closure under concatenation)?
 - $\mathcal{L}(M) = \mathcal{L}(M_1)^*$ (closure under Kleene star)?
 - $\mathcal{L}(M) = \overline{\mathcal{L}(M_1)}$ (closure under complement)?
 - $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ (closure under intersection)?

CLOSURE PROPERTIES

- $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$. Can we construct $M = (K, \Sigma, \Delta, s, F)$ such that
 - $\mathcal{L}(M) = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$ (closure under union):
 $K = K_1 \cup K_2$ $F = F_1 \cup F_2$ $\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \varepsilon, s_1), (s, \varepsilon, s_2)\}$
 - $\mathcal{L}(M) = \mathcal{L}(M_1)\mathcal{L}(M_2)$ (closure under concatenation):
 $s = s_1$ $F = F_2$ $\Delta = \Delta_1 \cup \Delta_2 \cup \{(f, \varepsilon, s_2) : f \in F_1\}$
 - $\mathcal{L}(M) = \mathcal{L}(M_1)^*$ (closure under Kleene star):
 $s = s_1$ $F = F_1$ $\Delta = \Delta_1 \cup \{(f, \varepsilon, s_1) : f \in F_1\}$
 - $\mathcal{L}(M) = \overline{\mathcal{L}(M_1)}$ (closure under complement):
 $s = s_1$ $\delta = \delta_1$ $F_1 = K \setminus F$
 - $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ (closure under intersection)?

CLOSURE PROPERTIES

- $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$. Can we construct $M = (K, \Sigma, \Delta, s, F)$ such that
 - $\mathcal{L}(M) = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$ (closure under union):
 $K = K_1 \cup K_2$ $F = F_1 \cup F_2$ $\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \varepsilon, s_1), (s, \varepsilon, s_2)\}$
 - $\mathcal{L}(M) = \mathcal{L}(M_1)\mathcal{L}(M_2)$ (closure under concatenation):
 $s = s_1$ $F = F_2$ $\Delta = \Delta_1 \cup \Delta_2 \cup \{(f, \varepsilon, s_2) : f \in F_1\}$
 - $\mathcal{L}(M) = \mathcal{L}(M_1)^*$ (closure under Kleene star):
 $s = s_1$ $F = F_1$ $\Delta = \Delta_1 \cup \{(f, \varepsilon, s_1) : f \in F_1\}$
 - $\mathcal{L}(M) = \overline{\mathcal{L}(M_1)}$ (closure under complement):
 $s = s_1$ $\delta = \delta_1$ $F_1 = K \setminus F$
 - $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ (closure under intersection):
 $\mathcal{L}(M_1) \cap \mathcal{L}(M_2) = \overline{\overline{\mathcal{L}(M_1)} \cup \overline{\mathcal{L}(M_2)}}$

CLOSURE UNDER INTERSECTION (CONSTRUCTIVE)

- $M_1 = (K_1, \Sigma, s_1, \delta_1, F_1)$, $M_2 = (K_2, \Sigma, s_2, \delta_2, F_2) \Rightarrow M = (K, \Sigma, s, \delta, F)$ such that $\mathcal{L}(M_1) \cap \mathcal{L}(M_2) = \mathcal{L}(M)$
- M must somehow run M_1 and M_2 "in parallel" to determine whether **both** accept the input
- It follows that at any given time we have to keep track of the current states of **both** M_1 and M_2 . We thus put $K = K_1 \times K_2$
- At the beginning of the computation both M_1 and M_2 are in their respective initial states, so $s = (s_1, s_2)$
- Similarly, in order for the input to be accepted, both M_1 and M_2 must be in one of their respective final states, so $F = F_1 \times F_2$
- Finally, δ should allow M to perform simultaneously exactly one transition of M_1 and exactly one transition of M_2 : $\delta((q_1, q_2), a) = (q'_1, q'_2)$ iff $\delta_1(q_1, a) = q'_1$ and $\delta_2(q_2, a) = q'_2$

- **Theorem:** Finite automata accept exactly all the languages in REG
- \supseteq :
 - REG = closure of $\{\{a\} : a \in \Sigma\} \cup \emptyset$ under union, concatenation, and Kleene star
 - Clearly FA accept $\{a\}$, \emptyset and are closed under the above operations
 - So FA accept all REG (closure is **minimal**)
- \subseteq :
 - Let $M = (\{q_1, q_2, \dots, q_n\}, \Sigma, \Delta, q_1, F)$
 - Let $\langle i, j, k \rangle$ be the path from q_i to q_j of **rank** k (i.e., q_α in the path implies $\alpha \leq k$)
 - Let $R(i, j, k)$ be the set of strings in Σ^* along all the paths $\langle i, j, k \rangle$
 - Obviously, $\mathcal{L}(M) = \bigcup \{R(1, j, n) : q_j \in F\}$
 - We prove that **all** $R(i, j, k)$ are regular

- \subseteq (cont'd):
 - Proof that $R(i, j, k)$ are regular is by induction over k
 - basis: All the $\langle i, j, 0 \rangle$ are transitions of M only, so $R(i, j, 0)$ are clearly regular
 - inductive hypothesis: all the $R(i, j, k-1)$ are regular
 - $R(i, j, k) = R(i, j, k-1) \cup R(i, k, k-1)R(k, k, k-1)^*R(k, j, k-1)$
 - then $R(i, j, k)$ are regular given the closure of regular expressions under union, concatenation, and Kleene star

ALGORITHMS FOR REGULAR LANGUAGES

REGULAR AND NON-REGULAR LANGUAGES

- nondeterministic to deterministic FA \Rightarrow exponential time
- nondeterministic FA to regular expression \Rightarrow exponential time
 - $O(|K|)$ computations of $R(i, j, k)$, but $R(i, j, k)$ **doubles** each time
- Whether two FA or regular expressions accept/generate the same language
 - polynomial time for DFA
 - likely exponential time for NFA, regular expressions
- Decide whether $w \in \mathcal{L}(M)$:
 - $O(|w|)$ if M is deterministic
 - $O(|K|^2|w|)$ if M is nondeterministic

- Typical application of regular languages: **pattern matching**

$$L_x = \{w \in \Sigma^* : x \text{ is a string of } w\}$$

- Regular languages can be described by regular expressions, finite automata (deterministic or not), and any combination of union, concatenation, intersection, complement, Kleene star of the above
- Languages that are not regular can be found using a **pumping theorem**:
 - Let L be a regular language. Then there exists $n \geq 1$ such that any $w \in L$ with $|w| \geq n$ can be written as $w = xyz$ with
 - $y \neq \epsilon$
 - $|xz| \leq n$
 - $xy^iz \in L$ for all $i \geq 0$

Trivial proof using the pigeonhole principle

- Typical examples of non-regular languages: $\{a^n b^n : n \geq 0\}$, $\{a^p : p \text{ is prime}\}$, $\{a^n b^n c^m : n, m \geq 0\}$