Nondeterministic Finite Automata

Nondeterminism is an abstraction which allows us to consider extensions of ordinary computation. We allow simulatneous execution paths, and accept if *at least one* path is an accepting path.

Ways to think about this:

- Concurrent (parallel) executions
- Execution tree
- "Guessing" an accpeting computation

Questions: Can nondeterminism be eliminated with no loss of expressive power? What is the cost?

NFA – Example

To model nondeterminism, transitions go from states to *sets of states*. Also, allow ϵ-*transitions* that do not consume any input.

q_1 is the start state and q_4 is the only accept state.

	0	1	ϵ
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

NFA Examples

- All strings over $\{0, 1\}$ containing a 1 in the next-to-last position.
- Using ϵ for OR.
- Another example:

	a	b	ϵ
q_1	$\emptyset$	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	$\emptyset$
q_3	$\{q_1\}$	$\emptyset$	$\emptyset$

where q_1 is the start state and the only accept state.

Formal Definition

A *nondeterministic finite automaton (NFA)* is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of *states*,
- Σ is a finite alphabet,
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is the *transition function*, i.e., $\delta(q, a) \subseteq Q$ where $q, q' \in Q$ and $a \in \Sigma \cup \{\epsilon\}$
- $q_0 \in Q$ is the *start state*,
- $F \subseteq Q$ is the set of *accept states* or *final states*.

Formal Definition of Computation

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 ... w_n$ be a string over Σ.

Then N *accepts* w if we can write $w = y_1 y_2 ... y_m$ where each $y_i \in \Sigma \cup \{\epsilon\}$ and there is a sequence of states $r_0, ..., r_m$ in Q s.t.

1. $r_0 = q_0$

2. $r_{i+1} \in \delta(r_i, y_{i+1})$

3. $r_m \in F$

Language Recognized by an NFA

The language of the machine M is the set L of all strings the language accepts. We write $L(M) = L$ and say M *accepts (or recognizes)* L. If the machine M accepts no strings then $L(M) = \emptyset$.

Converting NFA's to DFA's

Two DFA's or NFA's M_1 and M_2 are *equivalent* if $L(M_1) = L(M_2)$

Theorem For every NFA, there is an equivalent DFA

Proof Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we want to construct a DFA $D = (Q_D, \Sigma, \delta_D, q'_0, F_D)$ with $L(N) = L(D)$.

- *Basic idea*: states of DFA correspond to *sets* of states in NFA
- Current "set" of states = all states reachable from start state using input seen so far
- We start with the easier case of assuming there are no ϵ transitions in the NFA.

Defining D

- Alphabet is unchanged
- Q_D is defined to be $\mathcal{P}(Q)$, i.e., the set of all subsets of Q
- F_D is simply the set of all states in Q_D which contain (as sets) some state in F, i.e.

$$F_D = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$$

- The start state is $\{q_0\}$.
- What about δ_D?

Defining the Transition Function

- Idea: To determine $\delta_D(S, a)$, look at all the states $p \in S$ and see what states N takes p to, and then take their union.

- $\delta_D(S, a) = \bigcup_{p \in S} \delta(p, a)$

Example 1

Consider the following NFA.

$N = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
$\Sigma = \{0, 1\}$,
$F = \{q_3\}$, and
δ is specified by the following transition table:

q	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_4\}$
q_1	q_2	$\emptyset$
q_2	q_3	$\emptyset$
q_3	q_3	q_3
q_4	$\emptyset$	q_5
q_5	$\emptyset$	q_3

Generating the Corresponding DFA

- How many possible states?
- Do we really need to generate every possible state?
  - Only need to consider states that are reachable for some input.
- Use *lazy evaluation*
  - The start state is always reachable.
  - At a given step, add states reachable in one step from current states.
  - When can we stop? When we no longer discover new states.

Converting a DFA into an Equivalent NFA

- If $\delta_D(p, a) = q$ then set $\delta_N(p, a) = \{q\}$.
- Conclude: A language is accepted by a DFA iff it is accepted by an NFA without ϵ transitions.

Adding ϵ transitions

- For any state $S \in Q_D$, let $E(S)$ denote the set of states already in S or that can be reached from some state $r \in S$ by traveling along one or more ϵ arrows.
- Now, we modify δ_D:
  - $\delta_D(S, a) = \bigcup_{r \in S} E(\delta(r, a))$
- The new start state is $E(\{q_0\})$.

Correctness of the Construction

- Why does this work? At every step the state of the DFA constructed contains exactly all the states of the NFA that it could be in so far. In particular, if it could be in an accept state at the end of the computation, then it will be in an accept state at the end of the DFA computation.

- Note: this can be formalized by an induction argument.

- **Corollary:** A language is regular iff some NFA with ϵ-transitions recognizes it.

Example: Converting an NFA into a DFA

Another example:

	a	b	ϵ
q_1	$\emptyset$	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	$\emptyset$
q_3	$\{q_1\}$	$\emptyset$	$\emptyset$

where q_1 is the start state and the only accept state.

Applications of Nondeterminism: Closure under Union

1. Add new start state with ϵ arrows to both original start states.

2. Modify δ accordingly.

3. Accept states $=$ union of previous accept states.

Closed under Concatenation

Closed under Star