Adding Nondeterminism

Deterministic machine: at every step, there is only one choice.

Non-deterministic machine: at some steps, there may be more than one choice.

Nondeterminism in Practice

Omnipotent: Machine splits into a new machine for each choice; if any machine accepts, accept.

Omniscient: Whenever it has to make a choice, machine always guesses right.

Example NFA

Defining DFAs

A deterministic finite automaton is a 5-tuple:

\[ Q \] finite set ("states")
\[ \Sigma \] finite set ("alphabet")
\[ \delta : Q \times \Sigma \rightarrow Q \] transition function
\[ q_0 \in Q \] start state
\[ F \subseteq Q \] set of accepting states

How do we need to change this to support nondeterminism?
Defining NFAs

A nondeterministic finite automaton is a 5-tuple:

- \( Q \) finite set (“states”)
- \( \Sigma \) finite set (“alphabet”)
- \( \delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q) \) transition function
- \( q_0 \in Q \) start state
- \( F \subseteq Q \) set of accepting states

Output of transition function is a set of states, not just one state.

Computation Model of NFA

A string \( w \) is in the language of the NFA

\[ A = (Q, \Sigma, \delta, q_0, F) \]

if and only if

\[ \delta^*(q_0, w) \in F \]

where \( \delta^* \) is defined by:

\[ \delta^*(q, \epsilon) = q \]
\[ \delta^*(q, w = \alpha s) = \delta^*(\delta(q, \alpha), s) \]

Computation Model of NFA

A string \( w \) is in the language of the DFA

\[ A = (Q, \Sigma, \delta, q_0, F) \]

if and only if

\[ \delta^*(q_0, w) \cap F \neq \emptyset \]

where \( \delta^* \) is defined by:

\[ \delta^*(q, \epsilon) = \{q\} \]
\[ \delta^*(q, w = \alpha s) = \bigcup_{q_i \in \delta(q, \alpha)} \delta^*(q_i, s) \]

Is an NFA more powerful than a DFA?

Power of Machines

Languages that can be recognized by an A

Languages that can be recognized by a B

A and B are non-comparable.

Power of Machines

B is less powerful than A:
(1) Some A can recognize every language a B can recognize.
(2) There is some language that can be recognized by an A but not by any B.
Power of Machines

Languages that can be recognized by an $A$ can be recognized by some $B$:
1. every language that can be recognized by an $A$ can be recognized by some $B$.
2. every language that can be recognized by a $B$ can be recognized by some $A$.

Power of NFA/DFA

Easy part: is there any language a DFA can recognize that cannot be recognized by an NFA?

Hard part: is there any language an NFA can recognize that cannot be recognized by a DFA?

$L(NFA) \supseteq L(DFA)$

A nondeterministic finite automaton is a 5-tuple:
- $Q$: finite set ("states")
- $\Sigma$: finite set ("alphabet")
- $\delta$: $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$
- $q_0 \in Q$: start state
- $F \subseteq Q$: set of accepting states

Proof by construction:
Construct the NFA $N$ corresponding to any DFA $A = (Q, \Sigma, \delta, q_0, F)$:
$N = (Q, \Sigma, \delta', q_0', F)$ where
- $\forall q \in Q$, $\delta'(q, \epsilon) = \text{reject}$
- $\forall a \in \Sigma$, $\delta'(q, a) = \{\delta(q, a)\}$

$L(DFA) \supseteq L(NFA)$

Proof by Construction. Given $N = (Q, \Sigma, \delta, q_0, F)$, an NFA recognizing some language $A$, we construct a DFA $N' = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language:
- $Q' = \mathcal{P}(Q)$
- $\delta': Q' \times \Sigma \rightarrow Q'$ is defined to capture all possible states resulting from $\delta$ transitioning from the input state:
  - $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$
  - $q_0' = \delta(q_0)$
  - $F' = \{q \in Q' \mid q \cap F \neq \emptyset\}$
where $E: Q' \rightarrow Q'$: the epsilon-transition function defined by:
$$E(q) = q \cup \bigcup_{r \in \delta(q)} E(r)$$
Example Conversion

Converting NFAs to DFAs

How many states may be needed for the DFA corresponding to an NFA with \( N \) states?

\[ |Q| = 2^N \]

How much bigger is the transition function for the DFA corresponding to an NFA with \( N \) states?

\[
\text{DFA: } \delta: \mathcal{Q} \times \mathcal{E} \to \mathcal{Q} \\
\text{NFA: } \delta: \mathcal{Q} \times \mathcal{E} \to \mathcal{P}(\mathcal{Q})
\]

Regular Expressions

“something formatted like an [email] address”

[address@domain] *(A-Za-z0-9)*\* \( \) *(A-Za-z0-9)*

This matches most email addresses. Matching the full spec of all possible email addresses is left as exercise.
Regular Expression

Base:
- $a \in \Sigma$: \{ $a$ \}
- $\varepsilon$: \{ $\varepsilon$ \}
- $\emptyset$: \{ \}

Induction: $R_1, R_2$ are regular expressions
- $R_1 \cup R_2$: $L(R_1) \cup L(R_2)$
- $R_1 \cdot R_2$: \{ $xy$ | $x \in L(R_1)$ and $y \in L(R_2)$ \}
- $R_1^*$: \{ $x^n$ | $x \in L(R_1)$ and $n \geq 0$ \}

How powerful are Regular Expressions?

Is there a Regular Expression that describes the same language as any NFA?

Is there some NFA that describes the same language as any Regular Expression?

$L(\text{RE}) \subseteq L(\text{NFA})$

Base:
- $a \in \Sigma$: \{ $a$ \}
- $\varepsilon$: \{ $\varepsilon$ \}
- $\emptyset$: \{ \}

Induction: $R_1, R_2$ are regular expressions
- $R_1 \cup R_2$: $L(R_1) \cup L(R_2)$
- $R_1 \cdot R_2$: \{ $xy$ | $x \in L(R_1)$ and $y \in L(R_2)$ \}
- $R_1^*$: \{ $x^n$ | $x \in L(R_1)$ and $n \geq 0$ \}

Proof by construction.

Trivial to draw NFAs for each of these languages.

Proving: $R_1 \cdot R_2$

Assume there are NFAs $A_1$ and $A_2$ that recognize the languages $L(R_1)$ and $L(R_2)$. Show how to construct an NFA that recognizes the produced language for each part.
Charge

- PS2 will be posted by tomorrow
- Finish reading Chapter 1
- Thursday: how to prove a language is non-regular