NFAs, Regular Expressions, and Equivalence with DFAs

Nondeterministic Finite Automaton (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol - can have 0 or >1
  - Also can have edges labeled by empty string ε
- Definition: The language recognized by an NFA is the set of strings x that label some path from its start state to one of its final states

Three ways of thinking about NFAs

- Outside observer: Is there a path labeled by x from the start state to some final state?
- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)
- Parallel exploration: The NFA computation runs all possible computations on x step-by-step at the same time in parallel

Design an NFA to recognize the set of binary strings that contain 111 or have an even # of 1's

NFAs and Regular Expressions

Theorem: For any set of strings (language) A described by a regular expression, there is an NFA that recognizes A.

Proof idea: Structural induction based on the recursive definition of regular expressions...

Note: One can also find a regular expression to describe the language recognized by any NFA but we won’t prove that fact
Regular expressions over $\Sigma$

- **Basis:**
  - $\emptyset, \varepsilon$ are regular expressions
  - $a$ is a regular expression for any $a \in \Sigma$
- **Recursive step:**
  - If $A$ and $B$ are regular expressions then so are:
    - $(A \cup B)$
    - $(AB)$
    - $A^*$

**Basis**

- **Case $\emptyset$:**
- **Case $\varepsilon$:**
- **Case $a$:**

**Inductive Hypothesis**

- Suppose that for some regular expressions $A$ and $B$ there exist NFAs $N_A$ and $N_B$ such that $N_A$ recognizes the language given by $A$ and $N_B$ recognizes the language given by $B$.

**Inductive Step**

- **Case $(A \cup B)$:**

**Inductive Step**

- **Case $(A \cup B)$:**
Inductive Step

• Case (AB):

NFAs and DFAs

- Every DFA is an NFA
  - DFAs have requirements that NFAs don’t have

- Can NFAs recognize more languages? No!

- Theorem: For every NFA there is a DFA that recognizes exactly the same language

Conversion of NFAs to a DFAs

- Proof Idea:
  - The DFA keeps track of ALL the states that the part of the input string read so far can reach in the NFA
  - There will be one state in the DFA for each subset of states of the NFA that can be reached by some string
Conversion of NFAs to DFAs

- New start state for DFA
  - The set of all states reachable from the start state of the NFA using only edges labeled $\lambda$.

Example: NFA to DFA

- Example: NFA to DFA
  - Final states for the DFA
    - All states whose set contain some final state of the NFA.
Exponential blow-up in simulating nondeterminism

- In general the DFA might need a state for every subset of states of the NFA
  - Power set of the set of states of the NFA
  - An example where roughly $2^n$ is necessary
    - Is the $(n-1)$th character from the end a 1?

- The famous “P=NP?” question asks whether a similar blow-up is always necessary to get rid of nondeterminism for polynomial-time algorithms