Lemma: The language recognized by a DFA is the set of strings $x$ that label some path from its start state to one of its final states.

Definition: $x$ is in the language recognized by an NFA if and only if $x$ labels a path from the start state to some final state.
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

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:
  - $\emptyset$, $\lambda$ 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 $\lambda$:
- Case $a$:
basis

• Case $\emptyset$: 

• Case $\lambda$: 

• 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 $(A \cup B)$:

Case $(A \cup B)$:
inductive step

Case (AB):

\[ \begin{align*} N_A & \quad N_B \end{align*} \]

inductive step

Case (AB):

\[ \begin{align*} N_A & \quad N_B \end{align*} \]

inductive step

Case A*:

\[ N_A \]

inductive step

Case A*:

\[ \begin{align*} N_A & \quad N_A \end{align*} \]
build an NFA for \((01 \cup 1)^*0\)

solution

\((01 \cup 1)^*0\)

NFAs and DFAs

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

Can NFAs recognize more languages?

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.

**New start state for DFA**

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

**Final states for the DFA**

- All states whose set contain some final state of the NFA.

---

**Conversion of NFAs to DFAs**

For each state of the DFA corresponding to a set $S$ of states of the NFA and each symbol $s$:

- Add an edge labeled $s$ to state corresponding to $T$, the set of states of the NFA reached by starting from some state in $S$, then following one edge labeled by $s$, and then following some number of edges labeled by $\lambda$.
- $T$ will be $\emptyset$ if no edges from $S$ labeled $s$ exist.
example: NFA to DFA

NFA

\[ \begin{array}{c}
\text{a} \\
\text{c} \quad \text{b}
\end{array} \]

DFA

\[ \begin{array}{c}
\text{a} \\
\text{c} \quad \text{b}
\end{array} \]

example: NFA to DFA

NFA

\[ \begin{array}{c}
\text{a} \\
\text{c} \quad \text{b}
\end{array} \]

DFA

\[ \begin{array}{c}
\text{a} \\
\text{c} \quad \text{b}
\end{array} \]
example: NFA to DFA

NFA

DFA

example: NFA to DFA

NFA

DFA

example: NFA to DFA

NFA

DFA

example: NFA to DFA

NFA

DFA
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
  – n-state NFA yields DFA with at most \(2^n\) states
  – We saw an example where roughly \(2^n\) is necessary
    Is the n\(^{th}\) char 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