Chapter 6:
NFA Applications
Implementing NFAs

- The problem with implementing NFAs is that, being nondeterministic, they define a more complex computational procedure for testing language membership.
- To implement an NFA we must give a computational procedure that can look at a string and decide whether the NFA has at least one sequence of legal transitions on that string leading to an accepting state.
- This seems to require searching through all legal sequences for the given input string—but how?
Implementing NFAs

• One approach is to convert the NFA into a DFA and implement that instead.
• This NFA/DFA conversion is both useful and theoretically interesting: the fact that it is always possible shows that in spite of their extra flexibility, NFAs have exactly the same power as DFAs. They can define exactly the regular languages.
Outline

• 6.1 NFA Implemented With Backtracking Search
• 6.2 NFA Implemented With Bit-Mapped Parallel Search
• 6.3 The Subset Construction
• 6.4 NFAs Are Exactly As Powerful As DFAs
• 6.5 DFA Or NFA?
From NFA To DFA

- For any NFA, there is a DFA that recognizes the same language
- Proof is by construction: a DFA that keeps track of the set of states the NFA might be in
- This is called the \textit{subset construction}
- First, an example starting from this NFA:
Initially, the set of states the NFA could be in is just \( \{q_0\} \)

So our DFA will keep track of that using a start state labeled \( \{q_0\} \):
• Now suppose the set of states the NFA could be in is \( \{ q_0 \} \), and it reads a 0

• The set of possible states after reading the 0 is \( \{ q_0 \} \), so we can show that transition:
• Suppose the set of states the NFA could be in is \( \{q_0\} \), and it reads a 1
• The set of possible states after reading the 1 is \( \{q_0, q_1\} \), so we need another state:
• From \( \{q_0, q_1\} \) on a 0, the next set of possible states is \( \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_2\} \)

• From \( \{q_0, q_1\} \) on a 1, the next set of possible states is \( \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0, q_1, q_2\} \)

• Adding these transitions and states, we get…
The DFA construction continues.
Eventually, we find that no further states are generated.
That's because there are only finitely many possible sets of states: $P(Q)$.
In our example, we have already found all sets of states reachable from $\{q_0\}$...
Accepting States

- It only remains to choose the accepting states.
- An NFA accepts $x$ if its set of possible states after reading $x$ includes at least one accepting state.
- So our DFA should accept in all sets that contain at least one NFA accepting state.
Some Exercises

Convert the following NFAs into DFAs.

a)

b)

c)
Implementation Note

• The subset construction defined the DFA transition function by

\[ \delta_D(R, a) = \bigcup_{r \in R} \delta_N^*(r, a) \]

for some set of states R.
Start State Note

• In the subset construction, the start state for the new DFA is

\[ q_D = \delta^*_N(q_N, \varepsilon) \]

• Often this is the same as \( q_D = \{q_N\} \), as in our earlier example

• But the difference is important if there are \( \varepsilon \)-transitions from the NFA's start state
Empty-Set State Note

• The empty set is a subset of every set
• So the full subset construction always produces a DFA state for \( \{\} \)
• This is reachable from the start state if there is some string \( x \) for which the NFA has no legal sequence of moves: \( \delta_N^*(q_N, x) = \{\} \)
• For example, this NFA, with \( L(N) = \{ \varepsilon \} \)
• $P(\{q_0\}) = \{ \{\}, \{q_0\} \}$
• A 2-state DFA

\[
\begin{align*}
\delta_D(\{q_0\},0) &= \bigcup_{r \in \{q_0\}} \delta^*_N(r,0) = \{\}\ \\
\delta_D(\{q_0\},1) &= \bigcup_{r \in \{q_0\}} \delta^*_N(r,1) = \{\}\ \\
\delta_D(\{\},0) &= \bigcup_{r \in \{\}} \delta^*_N(r,0) = \{\}\ \\
\delta_D(\{\},1) &= \bigcup_{r \in \{\}} \delta^*_N(r,1) = \{\}\ \\
\end{align*}
\]
Trap State Provided

• The subset construction always provides a state for \{\}
• And it is always the case that

$$\delta_D(\{\}, a) = \bigcup_{r \in \{\}} \delta_N^*(r, a) = \{\}$$

so the \{\} state always has transitions back to itself for every symbol \(a\) in the alphabet

• It is a non-accepting trap state
Outline

- 6.1 NFA Implemented With Backtracking Search
- 6.2 NFA Implemented With Bit-Mapped Parallel Search
- 6.3 The Subset Construction
- 6.4 NFAs Are Exactly As Powerful As DFAs
- 6.5 DFA Or NFA?
NFAs Are Exactly As Powerful As DFAs

• We want to show that NFAs and DFAs are equivalent.
• This means we want to show that for any NFA there is a DFA and for any DFA there is an NFA.
Lemma 6.3

If $L(N)$ for some NFA $N$, then $L(N)$ is a regular language.

Proof: Every NFA $N$ gives rise to an equivalent DFA $D$ via the subset construction with $L(N) = L(D)$. Therefore $L(N)$ is regular.
Lemma 6.4

If \( L \) is any regular language, there is some NFA \( N \) for which \( L(N) = L \).

Proof:

- DFAs are just special NFAs that have never have a choice.
- To turn a DFA into an NFA all we have to do is modify the transition function from returning single states to sets of states:
  - Let \( L \) be any regular language
  - By definition there must be some DFA \( M = (Q, \Sigma, \delta, q_0, F) \) with \( L(M) = L \)
  - Define a new NFA \( N = (Q, \Sigma, \delta', q_0, F) \), where \( \delta'(q,a) = \{\delta(q,a)\} \) for all \( q \in Q \) and \( a \in \Sigma \), and \( \delta'(q,\varepsilon) = \{\} \) for all \( q \in Q \)
  - Now \( \delta'^*(q,x) = \{\delta^*(q,x)\} \), for all \( q \in Q \) and \( x \in \Sigma^* \)
  - Thus \( L(N) = L(M) = L \)
Theorem 6.4

A language $L$ is $L(N)$ for some NFA $N$ if and only if $L$ is a regular language.

Proof:

• Follows immediately from the previous lemmas
Assignment #3

• See website