Lecture 9

Non-Deterministic Finite Automata

CS 241: Foundations of Sequential Programs
Spring 2017
DFA Review

- a.k.a. finite state machines
- start state
- final/accepting states
- implicit error state
- accepted and rejected words
- $L(M)$ – the language recognized by DFA $M$
- Notice that $L(M) = L(M')$ even though $M \not= M'$
About that last point, $L(M) = L(M')$ even though $M \neq M'$
Can we determine if this is true for all pairs of machines $(M, M')$?

- Yes
- In $O(n^2)$ time, even ($n$ is the number of states in $M$ and $M'$ combined)
- How? The languages are infinite!
We start with a simpler problem: Are two states indistinguishable? States $p, q$ are distinguishable iff there exists a word $W$ that distinguishes them.

$W$ distinguishes $p$ from $q$ iff starting at one of $p, q$ with input $W$ would accept, while starting at the other would reject.

- **Base Case** - if only one of $p, q$ is accepting then $\epsilon$ distinguishes them.
- **Recursive Case** - If there is a symbol $s \in \Sigma$ s.t.
  \[
  \delta(p, s) = p', \quad \delta(q, s) = q' \quad \text{and} \quad (p', q') \text{ are distinguished by } W,
  \]
  then $p, q$ are distinguished by $sW$. 


Table Filling Example - A Graph View
Table Filling Example - Equivalence

A → B → C

A ← B ← C
NFAs

- \( L = \{bba, baa, bbba, bbbbaa, bbbbaaa, \ldots\} \) which is either 2 b’s followed by an a, or 1 or more b’s followed by 2 a’s
- try to derive this using a DFA
NFAs

- \( L = \{ bba, baa, bbbaa, bbbbaa, \ldots \} \) which is either 2 b’s followed by an a, or 1 or more b’s followed by 2 a’s

- try to derive using a nice NFA
NFA definition

Same as a DFA with the following change:

\[ T : Q \times \Sigma \rightarrow 2^Q \]

That is, we can be in a set of states, and thus \( T \) is a relation instead of a function.
NFA Interpreter Algorithm

Input: A word $W = w_1 w_2 ... w_n$, where each $w_i \in \Sigma$
Output: true if accepted, false if rejected

$S \leftarrow \{ q_0 \}$
for $i = 1$ to $n$ {
    $S \leftarrow \bigcup_{q \in S} T(q, w_i)$
}
return $S \cap A \neq \emptyset$
Implementing an NFA interpreter

Instead of keeping track of current state, keep track of a set of current states

The dictionary for $T$ is the same as for $\delta$ but the values are sets of states

Inefficient. Each transition requires (worst case) $n$ unions. Let this problem be called “Bird 1”
Differences between NFAs and DFAs

- One starts with N and the other with D
- NFAs seem more powerful
- They aren’t
  - \( \forall \text{ NFA } M, \exists \text{ DFA } M' \text{ s.t. } L(M) = L(M') \)
  - We can build \( M' \), we have the technology
  - Let this problem be called “Bird 2”
Representing sets as one label: Ordered sequence.

\{A, B, C\} \Rightarrow ABC

How does this change \( T \)?

How does it change \( S \)?

How does it change \( \bigcup_{q \in S} T(q, w_i) \)?

Doesn’t this make a huge DFA?

- It can
- It usually doesn’t
A few words about the subset construction

Apply the subset construction on the example language
$L = \{bba, baa, bbaa, bbbbaa, bbbbaa, \ldots \}$ which is either 2 b’s followed by an a, or 1 or more b’s followed by 2 a’s
Review

- An example comparing DFAs and NFAs: create an NFA (then a DFA) for all words over \( \Sigma = \{a, b\} \) with the subword \( aba \) in them.

- Can convert between an NFA and DFA using the subset construction
  - define each set of states that can be occupied at the same step
  - one state in the DFA for each unique set of states in the NFA
\( \epsilon \)-NFAs

- allows transition between states on “no input”
- can be used as “glue” for joining machines together
- example: \( L = \{ \text{card, cab, calf} \} \)
Converting $\epsilon$-NFAs to NFAs

It should not be surprising that $\epsilon$-NFAs can be converted to NFAs

1. take $\epsilon$ shortcuts
2. pull back final states
3. remove $\epsilon$ transitions
4. remove dead states
Killer app for Finite Automata/Transducers

Scanner: see asm.*