Non-deterministic Finite Automata (NFAs)

Lecture 4
Thursday, September 7, 2017

Differences from DFA
- From state $q$ on same letter $a \in \Sigma$ multiple possible states
- No transitions from $q$ on some letters
- $\epsilon$-transitions!

Questions:
- Is this a “real” machine?
- What does it do?

NFA behavior

Machine on input string $w$ from state $q$ can lead to set of states (could be empty)
- From $q_\epsilon$ on 1
- From $q_\epsilon$ on 0
- From $q_0$ on $\epsilon$
- From $q_\epsilon$ on 01
- From $q_{00}$ on 00
NFA acceptance: informal

Informal definition: An NFA $N$ accepts a string $w$ iff some accepting state is reached by $N$ from the start state on input $w$.

The language accepted (or recognized) by a NFA $N$ is denote by $L(N)$ and defined as: $L(N) = \{ w \mid N$ accepts $w \}$.

NFA acceptance: example

Is $01$ accepted?
Is $001$ accepted?
Is $100$ accepted?
Are all strings in $1^*01$ accepted?
What is the language accepted by $N$?

Comment: Unlike DFAs, it is easier in NFAs to show that a string is accepted than to show that a string is not accepted.

Simulating NFA

Example the first

(N1) $\begin{array}{cccccc}
\text{A} & a, b \\
\text{B} & a \\
\text{C} & b \\
\text{D} & a \\
\text{E} & b \\
\end{array}$

Run it on input $ababa$.
Idea: Keep track of the states where the NFA might be at any given time.

$t = 0$: $\begin{array}{cccccc}
a, b \\
a \\
b \\
a \\
b \\
\end{array}$

Remaining input: $ababa$.

$t = 1$: $\begin{array}{cccccc}
a, b \\
a \\
b \\
a \\
b \\
\end{array}$

Remaining input: $baba$.

Formal Tuple Notation

Definition

A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- $Q$ is a finite set whose elements are called states,
- $\Sigma$ is a finite set called the input alphabet,
- $\delta: Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of $Q$),
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

$\delta(q, a)$ for $a \in \Sigma \cup \{\varepsilon\}$ is a subset of $Q$ — a set of states.
Reminder: Power set

For a set \( Q \) its power set is: \( \mathcal{P}(Q) = 2^Q = \{ X \mid X \subseteq Q \} \) is the set of all subsets of \( Q \).

Example

\( Q = \{1, 2, 3, 4\} \)

\[
\mathcal{P}(Q) = \left\{ \emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\} \right\}
\]

Example

Transition function in detail...

\[
\begin{align*}
\delta(q_\epsilon, \varepsilon) &= \{ q_\epsilon \} \\
\delta(q_\epsilon, 0) &= \{ q_\epsilon, q_0 \} \\
\delta(q_\epsilon, 1) &= \{ q_\epsilon \} \\
\delta(q_0, \varepsilon) &= \{ q_0, q_{00} \} \\
\delta(q_0, 0) &= \{ q_{00} \} \\
\delta(q_0, 1) &= \{ \} \\
\delta(q_{00}, \varepsilon) &= \{ q_{00} \} \\
\delta(q_{00}, 0) &= \{ \} \\
\delta(q_{00}, 1) &= \{ q_p \} \\
\delta(q_p, \varepsilon) &= \{ q_p \} \\
\delta(q_p, 0) &= \{ \} \\
\delta(q_p, 1) &= \{ q_p \}
\end{align*}
\]

Example

Extending the transition function to strings

\( \delta(q, a) \): set of states that \( N \) can go to from \( q \) on reading \( a \in \Sigma \cup \{ \varepsilon \} \).

Want transition function \( \delta^*: Q \times \Sigma^* \to \mathcal{P}(Q) \).

\( \delta^*(q, w) \): set of states reachable on input \( w \) starting in state \( q \).
Extending the transition function to strings

Definition

For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\varepsilon$-reach$(q)$ is the set of all states that $q$ can reach using only $\varepsilon$-transitions.

More formally, the transition function in an NFA with $\varepsilon$-transitions using large red arrows; we won’t normally do that.) This NFA deliberately has more $\varepsilon$-transitions than necessary. For example, consider the following NFA with $\varepsilon$-transitions

Example

What is:

- $\delta^*(s, \varepsilon)$
- $\delta^*(s, 0)$
- $\delta^*(c, 0)$
- $\delta^*(b, 00)$

Important: Formal definition of the language of NFA above uses $\delta^*$ and not $\delta$. As such, one does not need to include $\varepsilon$-transitions closure when specifying $\delta$, since $\delta^*$ takes care of that.
Another definition of computation

**Definition**

\[ q \xrightarrow{w}_N p : \text{State } p \text{ of NFA } N \text{ is reachable from } q \text{ on } w \iff \text{there exists a sequence of states } r_0, r_1, \ldots, r_k \text{ and a sequence } x_1, x_2, \ldots, x_k \text{ where } x_i \in \Sigma \cup \{\varepsilon\}, \text{ for each } i, \text{ such that:} \]

- \( r_0 = q, \)
- for each \( i, r_{i+1} \in \delta(r_i, x_{i+1}), \)
- \( r_k = p, \) and
- \( w = x_1x_2x_3\cdots x_k. \)

**Definition**

\[ \delta^* N(q, w) = \{ p \in Q \mid q \xrightarrow{w}_N p \}. \]

### Part II

**Constructing NFAs**

#### Why non-determinism?

- Non-determinism adds power to the model; richer programming language and hence (much) easier to “design” programs
- Fundamental in theory to prove many theorems
- Very important in practice directly and indirectly
- Many deep connections to various fields in Computer Science and Mathematics

Many interpretations of non-determinism. Hard to understand at the outset. Get used to it and then you will appreciate it slowly.

#### DFAs and NFAs

- Every DFA is a NFA so NFAs are at least as powerful as DFAs.
- NFAs prove ability to “guess and verify” which simplifies design and reduces number of states
- Easy proofs of some closure properties
Example
Strings that represent decimal numbers.

Example
\{strings that contain CS374 as a substring\}
\{strings that contain CS374 or CS473 as a substring\}
\{strings that contain CS374 and CS473 as substrings\}

Example
\( L_k = \{ \text{bitstrings that have a 1 } k \text{ positions from the end} \} \)

A simple transformation

Theorem
For every NFA \( N \) there is another NFA \( N' \) such that \( L(N) = L(N') \) and such that \( N' \) has the following two properties:
- \( N' \) has single final state \( f \) that has no outgoing transitions
- The start state \( s \) of \( N \) is different from \( f \)
Closure properties of NFAs

Are the class of languages accepted by NFAs closed under the following operations?
- union
- intersection
- concatenation
- Kleene star
- complement

Closure under union

**Theorem**

For any two NFAs $N_1$ and $N_2$ there is a NFA $N$ such that $L(N) = L(N_1) \cup L(N_2)$.

Closure under concatenation

**Theorem**

For any two NFAs $N_1$ and $N_2$ there is a NFA $N$ such that $L(N) = L(N_1) \cdot L(N_2)$. 
Closure under Kleene star

Theorem
For any NFA $N_1$ there is a NFA $N$ such that $L(N) = (L(N_1))^\ast$.

Does not work! Why?

Part IV
NFA\text{s} capture Regular Languages
Regular Languages Recap

Regular Languages

- $\emptyset$ regular
- $\{\epsilon\}$ regular
- $\{a\}$ regular for $a \in \Sigma$
- $R_1 \cup R_2$ regular if both are
- $R_1R_2$ regular if both are
- $R^*$ is regular if $R$ is

Regular Expressions

- $\emptyset$ denotes $\emptyset$
- $\epsilon$ denotes $\{\epsilon\}$
- $a$ denote $\{a\}$
- $r_1 + r_2$ denotes $R_1 \cup R_2$
- $r_1r_2$ denotes $R_1 R_2$
- $r^*$ denote $R^*$

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language.

NFA\’s and Regular Language

Theorem

For every regular language $L$ there is an NFA $N$ such that $L = L(N)$.

Proof strategy:

- For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$
- Induction on length of $r$

Base cases: $\emptyset$, $\{\epsilon\}$, $\{a\}$ for $a \in \Sigma$.

Inductive cases:

- $r_1$, $r_2$ regular expressions and $r = r_1 + r_2$. By induction there are NFAs $N_1$, $N_2$ s.t $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA $N$ s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$.
- $r = r_1^* r_2$. Use closure of NFA languages under concatenation.
- $r = (r_1)^*$. Use closure of NFA languages under Kleene star.
Example

$$(\epsilon+0)(1+10)^*$$

Final NFA simplified slightly to reduce states