Conventional Finite-State Automata

Dr. John McDonough

Spoken Language Systems
Saarland University

May 11, 2009
Introduction

- In this lecture, we introduce the conventional finite-state automata.
- We will also consider two important operations on finite-state automata:
  - The *power set construction* enables a deterministic finite-state automaton to be constructed from a nondeterministic automaton.
  - The $\epsilon$-removal algorithm enables an automaton without $\epsilon$-transitions to be constructed from an automaton with $\epsilon$-transitions.
- We will begin with a simple example.

**Coverage**: Hopcroft and Ullman (1974), Chapter 2
Problem: A man has a boat, a goat, a wolf, and a cabbage.
In the boat there is only enough room for the man and one of his possessions.
The man must cross a river with his entourage but:
  - For obvious reasons, the wolf cannot be left alone with the goat.
  - For obvious reasons, the goat cannot be left alone with the cabbage.
How can the man solve this problem?
Luckily, the man has a day job as a professor of computer science, and realizes that the problem can be solved with a finite-state model.

Let the current state of the system correspond to the objects still on the left bank of the river.

Initially, the state is MWGC-∅; i.e., the man, wolf, goat, and cabbage are on the left side of the river, nothing is on the right side, as indicated by ∅.

Every time the man crosses the river by himself, we label the transition in the transition diagram with $m$.

Every time the man crosses the river and carries one of his entourage, we label the transition with one of $w$, $g$, or $c$, for wolf, goat, or cabbage respectively.

This results in the transition diagram shown in Figure ??.
Transition Diagram
A solution to the man-wolf-goat-cabbage problem corresponds to a path through the transition diagram from the start state MWGC-$\emptyset$ to the end state $\emptyset$-MWGC.

It is clear from the transition diagram that there are two equally short solutions to the problem.

There are an infinite number of possible solutions, all but two of which involve useless cycles.

As with all finite-state automata, there is a unique start state.

This particular FSA also has a single valid end or accepting state, which is not generally the case.
Formal Definitions

Formally define a *finite-state automaton* (FSA) as the 5-tuple $(Q, \Sigma, \delta, i, F)$ where

- $Q$ is a finite set of *states*,
- $\Sigma$ is a finite *alphabet*,
- $i \in Q$ is the *initial state*,
- $F \subseteq Q$ is the set of *final states*,
- $\delta$ is the *transition function* mapping $Q \times \Sigma$ to $Q$, which implies $\delta(q, a)$ is a state for each state $q$ and input $a$ provided that $a$ is accepted when in state $q$. 
Extending $\delta$ to Strings

- To handle strings, we must extend $\delta$ from a function mapping $Q \times \Sigma$ to $Q$, to a function mapping $Q \times \Sigma^*$ to $Q$, where $\Sigma^*$ is the \textit{Kleene closure}.
- Let $\delta(q, w)$ be the state that the FSA is in after beginning from state $q$ and reading the input string $w$.
- Formally, we require:
  1. $\hat{\delta}(q, \epsilon) = q$,
  2. for all strings $w$ and symbols $a$, $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$.
- Condition (1) implies that the FSA cannot change state without receiving an input.
- Condition (2) tells us how to find the current state after reading a nonempty input string $wa$; find $p = \hat{\delta}(q, w)$, then find $\delta(p, a)$.
- As $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$ we shall use $\delta$ to represent both $\delta$ and $\hat{\delta}$ henceforth.
A string $x$ is *accepted* by a FSA $M = (Q, \Sigma, \delta, i, F)$ if and only if $\delta(i, x) = p$ for some $p \in F$.

The *language accepted by* $M$, which we denote as $L(M)$, is that set $\{ x \mid \delta(i, x) \in F \}$.

A language is a *regular set*, or simply *regular*, if it is the set accepted by some automaton.

$L(M)$ is the *complete* set of strings accepted by $M$. 

**Regular Languages**
Consider a modification to the original definition of the FSA, whereby zero, one, or more transitions from a state with the same symbol are allowed.

This new model is known as the **nondeterministic finite-state automaton** (NFSA).

Observe that there are two edges labeled 0 out of state \( i \), one each going back to state \( i \) and to state \( q_3 \).
Formal Definitions: NFSA

Formally define a *nondeterministic finite-state automaton* (NFSA) as the 5-tuple \((Q, \Sigma, \delta, i, F)\) where

- \(Q\) is a finite set of *states*,
- \(\Sigma\) is a finite *alphabet*,
- \(i \in Q\) is the *initial state*,
- \(F \subseteq Q\) is the set of *final states*,
- \(\delta\) is the *transition function* mapping \(Q \times \Sigma\) to \(2^Q\), the power set of \(Q\). This implies \(\delta(q, a)\) is the set of all states \(p\) such that there is a transition labeled \(a\) from \(q\) to \(p\).
Theorem (equivalence of DFSAs and NFSAs): Let $L$ be the set accepted by a nondeterministic finite-state automaton. Then there exists a deterministic finite-state automaton that accepts $L$. 
Let \( M_1 = (Q_1, \Sigma, \delta_1, i_1, F_1) \) denote the NFSA accepting \( L \).

Define a DFSA \( M_2 = (Q_2, \Sigma, \delta_2, i_2, F_2) \) as follows:

- The states of \( M_2 \) are all subsets of the states of \( M_1 \), that is \( Q_2 = 2^{Q_1} \).
- \( M_2 \) keeps track in its states the subset of states that \( M_1 \) could be in at any given time.
- \( F_2 \) is the subset of states in \( Q_2 \) which contain a state \( f \in F_1 \).
- An element of \( m \in Q_2 \) will be denoted as \( m = [m_1, m_2, \ldots, m_N] \), where each \( m_n \in Q_1 \).
- Finally, \( i_2 = [i_1] \).
Definition of $\delta_2([p_1, p_2, \ldots, p_N], a)$

- By definition,
  
  \[ \delta_2([m_1, m_2, \ldots, m_N], a) = [p_1, p_2, \ldots, p_N] \]

  if and only if
  
  \[ \delta_1(\{m_1, m_2, \ldots, m_N\}, a) = \{p_1, p_2, \ldots, p_N\} \].

- In other words, $\delta_2([m_1, m_2, \ldots, m_N], a)$ is computed for $[m_1, m_2, \ldots, m_N] \in Q_2$ by applying $\delta$ to each $m_n \in Q_1$. 
Proof by Induction

We wish to demonstrate through induction on the string length $|x|$ that

$$\delta_2(i_2, x) = [m_1, m_2, \ldots, m_N]$$

if and only if

$$\delta_1(i_1, x) = \{m_1, m_2, \ldots, m_N\}.$$ 

- **Basis:** The result follows trivially for $|x| = 0$, as $i_2 = [i_1]$ and $x = \epsilon$.

- **Inductive Hypothesis:** Assume that the hypothesis is true for strings of length $N$ or less, and demonstrate it is then necessarily true for strings of length $N + 1$. 

Let $xa$ be a string of length $N + 1$, where $a \in \Sigma$. Then,

$$\delta_2(i_2, xa) = \delta_2(\delta_2(i_2, x), a).$$

By the inductive hypothesis,

$$\delta_2(i_2, x) = [m_1, m_2, \ldots, m_N]$$

if and only if

$$\delta(i_1, x) = \{m_1, m_2, \ldots, m_N\}.$$
Proof (cont’d.)

- But by the definition of \( \delta_2 \),
  \[ \delta_2([m_1, m_2, \ldots, m_N], a) = [p_1, p_2, \ldots, p_N] \]
  if and only if
  \[ \delta_1(\{m_1, m_2, \ldots, m_N\}, a) = \{p_1, p_2, \ldots, p_N\}. \]

- Thus,
  \[ \delta_2(i_2, xa) = [p_1, p_2, \ldots, p_N] \]
  if and only if
  \[ \delta(i_1, xa) = \{p_1, p_2, \ldots, p_N\}, \]
  which establishes the inductive hypothesis.
The power set $2^Q$ of $Q$ contains $2^{|Q|}$ subsets.

This implies that the power set construction requires exponential running time in the worst case; i.e., it is *intractable*.

Fortunately, for the FSAs used for speech recognition and many other applications, the vast majority of subsets in the power set are never constructed.

The key to successfully implementing the power set construction is to *not* construct a priori all subsets in the power set.

Rather, only those subsets are constructed which are actually required.

This subset is comprised of those subsets which are *accessible* from the initial node.
Conventional Finite-State Automata

Pseudocode for Power Set Construction

```python
00 def powerSetConstruction(τ₁, τ₂):
   01   F₂ ← ∅
   02   i₂ ← i₁
   03   Q ← { i₂ }
   04   while |Q| > 0:
   05       pop q₂ from Q
   06       if ∃ q ∈ q₂ such that q ∈ F₁:
   07           F₂ ← F₂ ∪ {q₂}
   08           for a such that δ(q₂, a) ≠ ∅:
   09               if δ₂(q₂, a) ∉ Q₂:
   10                 Q₂ ← Q₂ ∪ { δ₂(q₂, a) }
   11                 push δ₂(q₂, a) on Q
```

Figure: Pseudocode for power set construction.
Conventional Finite-State Automata

Finite-State Automata with \( \epsilon \)-Transitions

- We can further extend the definition of finite-state automata to allow \( \epsilon \)-transitions, which by definition consume no input symbol.
- Formally, define a *nondeterministic finite-state automaton with \( \epsilon \)-transitions* as the quintuple \( M = (Q, \Sigma, \delta, i, F) \).
- All elements of \( M \) have the same meaning as before except that \( \delta \) maps \( Q \times (\Sigma \cup \{ \epsilon \}) \) to \( 2^Q \).
- This implies that \( \delta(q, a) \) will consist of all states \( m \in Q \) such that there is a transition labeled \( a \) from \( q \) to \( p \), where either \( a = \epsilon \) or \( a \in \Sigma \).
- As before, we let \( L(M) \) denote the *language accepted by \( M = (Q, \Sigma, \delta, i, F) \)* such that \( L(M) = \{ w | \hat{\delta}(i, w) \text{ contains a state } p \in F \} \).
We now extend the definition of $\delta$ to $\hat{\delta}$ that maps $Q \times (\Sigma \cup \{\epsilon\})^*$ to $2^Q$.

In the end, $\hat{\delta}(q, w)$ will include all states $p$ such that there is a path from $q$ to $p$ labeled with $w$, perhaps including edges labeled $\epsilon$.

In computing $\hat{\delta}$, it will be necessary to determine the set of states accessible from a given state $q$ using only $\epsilon$-transitions.
We use $\varepsilon$–closure$(q)$ to denote the set of states $p \in Q$ such that there is a path from $q$ to $p$ consisting solely of $\varepsilon$-transitions.

This definition can be extended naturally to a set $P \subseteq Q$ according to

$$\varepsilon$–closure(P) = \bigcup_{q \in P} \varepsilon$–closure(q).$$
Theorem: If \( L \) is accepted by a NFSA with \( \epsilon \)-transitions, then \( L \) is accepted by a DFSA without \( \epsilon \)-transitions.

- Let \( M_1 = (Q_1, \Sigma, \delta_1, i_1, F_1) \) denote a NFSA with \( \epsilon \)-transitions. Let us construct \( M_2 = (Q_2, \Sigma, \delta_2, i_2, F_2) \) where

\[
F_2 = \begin{cases} 
F_1 \cup \{i_1\}, & \text{if } \epsilon\text{-closure}(i_1) \text{ contains a state } p \in F_1, \\
F_1, & \text{otherwise}, 
\end{cases}
\]

and \( \delta_2(q, a) \) is \( \hat{\delta}_1(q, a) \) for \( q \in Q_1 \) and \( a \in \Sigma \).

- We wish to show by induction on \( |x| \) that

\( \delta_2(i_2, x) = \hat{\delta}_1(i_1, x) \).
Inductive Hypothesis

- This may be untrue for $x = \epsilon$, however, as $\delta'(i, \epsilon) = \{i\}$, while $\hat{\delta}(i, \epsilon) = \epsilon$–closure($i$).
- Hence, we begin the induction with $|x| = 1$:
  - **Basis:** For $|x| = 1$, let $x = a$, and $\delta'(i, a) = \hat{\delta}(i, a)$ by the definition of $\delta'$.
  - **Induction:** For $|x| > 1$, let $x = wa$ for $w \in \Sigma^*$ and $a \in \Sigma$. Then
    \[
    \delta'(i, wa) = \delta'(\delta'(i, w), a).
    \]
Proof of Inductive Hypothesis

- By the inductive hypothesis, \( \delta'(i, w) = \hat{\delta}(i, w) \).
- Let \( \hat{\delta}(i, w) = P \). We must demonstrate that \( \delta'(P, a) = \hat{\delta}(i, wa) \).
- But
  \[
  \delta'(P, a) = \bigcup_{q \in P} \delta'(q, a) = \bigcup_{q \in P} \hat{\delta}(q, a).
  \]
- Then as \( P = \hat{\delta}(i, w) \) we have
  \[
  \bigcup_{q \in P} \hat{\delta}(q, a) = \hat{\delta}(i, wa)
  \]
  by the definition of \( \hat{\delta} \).
- Therefore,
  \[
  \delta'(i, wa) = \hat{\delta}(i, wa).
  \]
Completing the proof requires demonstrating that $\delta'(i, x)$ contains a state $q' \in F'$ if and only if $\hat{\delta}(i, x)$ contains a state $q \in F$. 
Pseudocode for ε–Removal

```
00  def epsilonRemoval(τ):
01      for p ∈ Q₁:
02          p.edges ← { e ∈ p.edges : e.symbol ≠ ε }
03          for q ∈ p.ε.closure:
04              E[p] ← E[p] ∪ {(p, a, w ⊗ w₁, r) : (q, a, w₁, r) ∈ E[q], a ≠ ε}
05      if q ∈ F and p ∉ F:
06          F ← F ∪ { p }
07          ρ[p] ← ρ[p] ⊕ (w ⊗ ρ[q])
```

Figure: Algorithm for ε–removal.
In this lecture, we have defined conventional finite-state automaton.

We have considered both deterministic and nondeterministic finite-state automata.

We have also considered the power set construction, whereby a deterministic automaton can be constructed from a nondeterministic automaton.

In addition, we have generalized the definition of automata to include $\epsilon$–transitions.

Finally, we have seen how an automaton without $\epsilon$–transitions can be constructed from an automaton with $\epsilon$–transitions.

In the next lecture, we will consider the relation between automata and regular expressions.