A linear algorithm for testing
Equivalence of Finite Automata
(Hopcroft and Karp, 1971)

Presented by
Chintan Rao
Vijeth D

November 24, 2009
The Problem

A Simple Algorithm

Hopcroft Karp Algorithm

Correctness

Conclusion

Outline

1. The Problem
2. A Simple Algorithm
   - Algorithm
   - Example
   - Analysis
   - Scope for Improvement
3. Hopcroft Karp Algorithm
   - Definitions
   - The Idea
   - Algorithm
   - Examples
4. Correctness
5. Conclusion
The Problem

Given two DFAs on same alphabet, we intend to determine if they accept the same language.

\[ M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1) \]
\[ M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2) \]

Is \( L(M_1) = L(M_2) \) ?
A Simple Algorithm

1. Minimize $M_1$ and $M_2$ to get canonical DFAs, $M'_1$ and $M'_2$.
   - Time complexity $= \mathcal{O}(|\Sigma|n \log n)$

2. Check if $M'_1$ and $M'_2$ are equivalent.
   - Time complexity $= \mathcal{O}(|\Sigma|m)$
     
     where $m$ is the number of states in $M'_1$ (or $M'_2$).

Overall complexity $= \mathcal{O}(|\Sigma|n \log n)$
Example

An Algorithm for testing equivalence of finite automata
Example

Automata Theory & Computability, Seminar

An Algorithm for testing equivalence of finite automata
Equivalence Classes

Automata Theory & Computability, Seminar

An Algorithm for testing equivalence of finite automata
Equivalence Classes

- \( C_0 \): (aaaa)*
- \( C_1 \): a(aaaa)*
- \( C_2 \): \( a^2(aaaa)* \)
- \( C_3 \): \( a^3(aaaa)* \)
- \( C_4 \): \( a^4(aaaa)* \)
- \( C_5 \): \( a^5(aaaa)* \)
- \( C_6 \): \( a^6(aaaa)* \)
- \( C_7 \): \( a^7(aaaa)* \)
Equivalence Classes

An Algorithm for testing equivalence of finite automata
Scope for improvement

Automata Theory & Computability, Seminar

An Algorithm for testing equivalence of finite automata
Scope for improvement
Scope for improvement

An Algorithm for testing equivalence of finite automata
Equivalent States

\[ p \approx q \overset{\text{def}}{\iff} \forall w \in \Sigma^* \left( \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F \right) \]

- Properties of \( \approx \)
  - Reflexive \( (p \approx p, \forall p) \)
  - Symmetric \( (p \approx q \Rightarrow q \approx p) \)
  - Transitive \( (p \approx q \text{ and } q \approx r \Rightarrow p \approx r) \)
  - Right Invariance
    \[ \forall a \in \Sigma \ (p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a)) \]
    \[ \forall w \in \Sigma^* \ (p \approx q \Rightarrow \hat{\delta}(p, w) \approx \hat{\delta}(q, w)) \]
The Idea

- Consider \( M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1) \) and \( M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2) \) as a single DFA.

\[
M = (Q_1 \cup Q_2, \Sigma, \delta, s_1, F_1 \cup F_2)
\]

where

\[
\delta(q, a) = \begin{cases} 
\delta_1(q, a) & \text{if } q \in Q_1 \\
\delta_2(q, a) & \text{if } q \in Q_2
\end{cases}
\]

- \( L(M_1) = L(M_2) \iff s_1 \approx s_2 \) in \( M \)
The Idea

Assume that $s_1 \approx s_2$ and see if there is any contradiction to

$$\forall w \in \Sigma^* \left( p \approx q \Rightarrow \hat{\delta}(p, w) \approx \hat{\delta}(q, w) \right)$$
Assume that $s_1 \approx s_2$ and see if there is any contradiction to

$$\forall w \in \Sigma^* \left(p \approx q \Rightarrow \hat{\delta}(p, w) \approx \hat{\delta}(q, w)\right)$$
Assume that $s_1 \approx s_2$ and see if there is any contradiction to

$$\forall w \in \Sigma^* \left( p \approx q \Rightarrow \hat{\delta}(p, w) \approx \hat{\delta}(q, w) \right)$$
Assume that $s_1 \approx s_2$ and see if there is any contradiction to

$$\forall w \in \Sigma^* \left( p \approx q \Rightarrow \hat{\delta}(p, w) \approx \hat{\delta}(q, w) \right)$$
The Idea

Assume that \( s_1 \approx s_2 \) and see if there is any contradiction to

\[
\forall w \in \Sigma^* \left( p \approx q \Rightarrow \hat{\delta}(p, w) \approx \hat{\delta}(q, w) \right)
\]
The Idea

Assume that $s_1 \approx s_2$ and see if there is any contradiction to

$$\forall w \in \Sigma^* \left( p \approx q \Rightarrow \hat{\delta}(p, w) \approx \hat{\delta}(q, w) \right)$$
The Idea

- A contradiction occurs if a final state and a non-final state end up being equivalent.
- Disjoint set (Union-Find) data structure is used for merging the states.
Hopcroft-Karp Algorithm

1. For every state \( q \in Q_1 \cup Q_2 \), make \( \text{MakeSet}(q) \)
2. \( \text{Union}(s_1, s_2) \)
3. Push \((s_1, s_2)\) on to a stack, \( S \)
4. While \( S \) is non-empty
   1. Pop pair \((q_1, q_2)\) from \( S \)
   2. For each \( a \in \Sigma \)
      1. \( r_1 = \text{Find}(\delta(q_1, a)) \)
      2. \( r_2 = \text{Find}(\delta(q_2, a)) \)
      3. If \( r_1 \neq r_2 \)
         \( \text{Union}(r_1, r_2) \)
         Push \((r_1, r_2)\) on to \( S \)

5. \( L(M_1) = L(M_2) \) if and only if no set contains both a final and a non-final state.
Complexity Analysis

1. Complexity = $O(n)$ where $n = |Q_1 \cup Q_2|$  
2. Complexity = $O(1)$  
3. Complexity = $O(1)$  
4. Complexity $\approx O(n|\Sigma|)$

1. There are at most $n - 1$ calls to `Union`  
2. For every union operation, we push exactly one pair to the stack  
3. For every pair in stack we have atmost $2|\Sigma|$ calls to `FindSet`  
4. If the number of Union-Find operations is $O(n)$ then overall time complexity of all instructions $\approx O(n)$ [Hopcroft, Ullman]  
5. Complexity = $O(n)$

**Overall Complexity** $\approx O(n|\Sigma|)$
An Example

Disjoint sets: \{0\} \{1\} \{2\} \{3\} \{4\} \{5\} \{6\} \{7\} \{8\} \{9\} \{10\} \{11\}
Disjoint sets: \{0, 4\} \{1\} \{2\} \{3\} \{5\} \{6\} \{7\} \{8\} \{9\} \{10\} \{11\}
An Example

Disjoint sets: \( \{0, 4\} \cup \{1, 5\} \cup \{2\} \cup \{3\} \cup \{6\} \cup \{7\} \cup \{8\} \cup \{9\} \cup \{10\} \cup \{11\} \)
An Example

Disjoint sets: \{0, 4\} \{1, 5\} \{2, 6\} \{3\} \{7\} \{8\} \{9\} \{10\} \{11\}
Disjoint sets: \{0, 4\} \{1, 5\} \{2, 6\} \{3, 7\} \{8\} \{9\} \{10\} \{11\}
Disjoint sets: \{0, 4, 8\} \{1, 5\} \{2, 6\} \{3, 7\} \{9\} \{10\} \{11\}
Disjoint sets: \{0, 4, 8\} \{1, 5, 9\} \{2, 6\} \{3, 7\} \{10\} \{11\}
An Example

Disjoint sets: \{0, 4, 8\} \{1, 5, 9\} \{2, 6, 10\} \{3, 7\} \{11\}
An Example

Disjoint sets: \{0, 4, 8\} \{1, 5, 9\} \{2, 6, 10\} \{3, 7, 11\}
Disjoint sets: \{0, 4, 8\} \{1, 5, 9\} \{2, 6, 10\} \{3, 7, 11\}
An Example

The DFAs accept the same language
Another Example

Disjoint sets: \{0\} \{1\} \{2\} \{3\} \{4\} \{5\} \{6\} \{7\}
Another Example

Disjoint sets: \{0, 4\} \{1\} \{2\} \{3\} \{5\} \{6\} \{7\}
Another Example

Disjoint sets: \{0, 4\} \{1, 5\} \{2\} \{3\} \{6\} \{7\}
Another Example

Disjoint sets: \{0, 4\} \{1, 5\} \{2, 6\} \{3\} \{7\}
Another Example

Disjoint sets: \{0, 4\} \{1, 5\} \{2, 6\} \{3, 7\}
Another Example

Disjoint sets: \{0, 4\} \{1, 5\} \{2, 6\} \{3, 7\}
Another Example

Disjoint sets: \( \{0, 4\} \ \{1, 3, 5, 7\} \ \{2, 6\} \)
Another Example

Disjoint sets: \{0, 4\} \{1, 3, 5, 7\} \{2, 6\}
Languages accepted by the DFAs are not same.
Correctness

The algorithm terminates.

- There are at most \( n - 1 \) union operations where \( n = |Q_1| + |Q_2| \)
- For every union operation, we push exactly one pair to the stack
Define equivalence relation $\equiv_E$ on $q \in Q_1 \cup Q_2$ as follows

$$q_1 \equiv_E q_2 \overset{\text{def}}{\iff} \text{FindSet}(q_1) = \text{FindSet}(q_2)$$

when the algorithm terminates.

**Claim:** $E$ is right invariant

$$q_1 \equiv_E q_2 \Rightarrow \delta(q_1, a) \equiv_E \delta(q_2, a) \forall a \in \Sigma$$
Connecting Sequence

At any point of time during the execution of the algorithm, a sequence of states \((q_1, q_2, \ldots, q_k)\) is said to be a connecting sequence if \(\forall i \in \{1, 2, \ldots, k - 1\}\), any of the following holds.

1. the pair \((q_i, q_{i+1})\) is currently present in stack or has been in stack previously.
2. \(\forall a \in \Sigma, \text{FindSet}(\delta(q_i, a)) = \text{FindSet}(\delta(q_{i+1}, a))\)

If \((q_1, q_2, \ldots, q_k)\) is currently a connecting sequence, then it will remain to be so in future.
Connecting Sequence

- If $q_1, q_2 \in$ a connecting sequence
  \[ q_1 \equiv_E q_2 \]
  \[ \delta(q_1, a) \equiv_E \delta(q_2, a) \forall a \in \Sigma \]
- If $q_1 \equiv_E q_2$, then there exists connecting sequence joining $q_1$ and $q_2$
  - Proof by induction
    After $i^{th}$ iteration, there exists a connecting sequence joining every pair of states in same set.
- Therefore, equivalence relation $\equiv_E$ is right invariant.
Correctness

Claim: If the algorithm outputs 'YES', then DFAs accept the same language.

Proof:
When the algorithm terminates, $s_1 \equiv_E s_2$

$\Rightarrow \delta(s_1, a) \equiv_E \delta(s_2, a) \ \forall a \in \Sigma$

$\Rightarrow \hat{\delta}(s_1, w) \equiv_E \hat{\delta}(s_2, w) \ \forall w \in \Sigma^*$

Since the algorithm outputs YES,

$\Rightarrow \hat{\delta}(s_1, w) \in F \iff \hat{\delta}(s_2, w) \in F \ \forall w \in \Sigma^*$

$\Rightarrow \hat{\delta}_1(s_1, w) \in F_1 \iff \hat{\delta}_2(s_2, w) \in F_2 \ \forall w \in \Sigma^*$

$\Rightarrow L(M_1) = L(M_2)$
Correctness

**Claim:** If DFAs accept the same language, then the algorithm outputs YES

**Proof:** Recall

\[ p \approx q \overset{\text{def}}{\iff} \forall w \in \Sigma^* \left( \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F \right) \]

- \( s_1 \approx s_2 \) in \( M \)
- \( q_1 \equiv_E q_2 \implies q_1 \approx q_2 \)
  
  Proved by induction. After \( i^{th} \) iteration, \( \text{FindSet}(q_1) = \text{FindSet}(q_2) \implies q_1 \approx q_2 \)

- For \( f \in F \) and \( q \not\in F \), we know that \( f \not\approx q \)
  
  \( \implies f \not\equiv_E q \)
  
  \( \implies \) The algorithm outputs YES.
This algorithm runs in $O(n|\Sigma|)$ which is best possible.

This algorithm can be easily extended to produce a 'witness' string if the input DFAs are not equivalent.

THANK YOU