CSL705: Theory of Computation

Ragesh Jaiswal, CSE, IIT Delhi
1. DFA Equivalence
2. DFA Minimisation
3. Myhill-Nerode Theorem
There are multiple questions that one may like to ask regarding regular languages:

- Is the language empty?
- Is a particular string $w$ in the language?
- Are two languages “equivalent”?
Design an algorithm that given a DFA,
There are multiple questions that one may like to ask regarding regular languages:

- Is the language empty?
- Is a particular string $w$ in the language?
- Are two languages “equivalent”?

Since we can easily go from one representation to another, the above questions may be asked just in terms of DFAs.

- Is the language empty? **Does there exist an accepting path?**
- Is a particular string $w$ in the language? **Does $w$ take the DFA from the start state to an accepting state?**
- Are two languages “equivalent”? **Are two DFA’s “equivalent”?**
Are two DFAs equivalent? That is, do they define the same language?

1. We will design an algorithm for the DFA equivalence problem.
2. Moreover, we will show that given any DFA, there is a way to produce an equivalent DFA with minimum number of states. We will also argue that such a DFA is unique.
Are two DFAs equivalent? That is, do they define the same language?

1. We will design an algorithm for the DFA equivalence problem.
2. Moreover, we will show that given any DFA, there is a way to produce an equivalent DFA with minimum number of states. We will also argue that such a DFA is unique.

**Definition (Equivalent states)**

Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.
Definition (Equivalent states)

Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.

- Are states $C$ and $G$ equivalent?
- Are states $A$ and $G$ equivalent?
- Are states $A$ and $E$ equivalent?
DFA Equivalence

**Definition (Equivalent states)**

Two states \( p, q \) of a given DFA are called equivalent if for all input strings \( w \), \( \hat{\delta}(p, w) \) is an accepting state iff \( \hat{\delta}(q, w) \) is an accepting state. Two states are called distinguishable if they are not equivalent.

- Are states \( C \) and \( G \) equivalent? **No**
- Are states \( A \) and \( G \) equivalent? **No**
- Are states \( A \) and \( E \) equivalent? **Yes**
Definition (Equivalent states)

Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.

Problem: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
Definition (Equivalent states)

Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.

Problem: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
DFA Equivalence

Definition (Equivalent states)
Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.

- **Problem**: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
DFA Equivalence

Definition (Equivalent states)

Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.

- **Problem**: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
Definition (Equivalent states)

Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.

Problem: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
DFA Equivalence

Definition (Equivalent states)
Two states \( p, q \) of a given DFA are called equivalent if for all input strings \( w \), \( \delta(p, w) \) is an accepting state iff \( \delta(q, w) \) is an accepting state. Two states are called distinguishable if they are not equivalent.

Problem: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
Definition (Equivalent states)

Two states \( p, q \) of a given DFA are called equivalent if for all input strings \( w \), \( \hat{\delta}(p, w) \) is an accepting state iff \( \hat{\delta}(q, w) \) is an accepting state. Two states are called distinguishable if they are not equivalent.

Problem: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
Definition (Equivalent states)

Two states $p, q$ of a given DFA are called equivalent if for all input strings $w$, $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state. Two states are called distinguishable if they are not equivalent.

- **Problem:** Design an algorithm that given a DFA, determines if any pair of states are equivalent.
DFA Equivalence

Definition (Equivalent states)
Two states \( p, q \) of a given DFA are called equivalent if for all input strings \( w \), \( \hat{\delta}(p, w) \) is an accepting state iff \( \hat{\delta}(q, w) \) is an accepting state. Two states are called distinguishable if they are not equivalent.

- **Problem:** Design an algorithm that given a DFA, determines if any pair of states are equivalent.
- What is the running time of this “table-filling” algorithm?
DFA Equivalence

**Definition (Equivalent states)**

Two states \( p, q \) of a given DFA are called equivalent if for all input strings \( w \), \( \hat{\delta}(p, w) \) is an accepting state iff \( \hat{\delta}(q, w) \) is an accepting state.

Two states are called distinguishable if they are not equivalent.

- **Problem**: Design an algorithm that given a DFA, determines if any pair of states are equivalent.
- What is the running time of this “table-filling” algorithm? \( O(n^4) \)
- Is it possible to make it better?
Problem 1: Design an algorithm that given a DFA, determines if any pair of states are equivalent.

Problem 2: Design an algorithm that given two DFAs determines if these DFAs are equivalent.
Problem 1: Design an algorithm that given a DFA, determines if any pair of states are equivalent.

Problem 2: Design an algorithm that given two DFAs determines if these DFAs are equivalent.

Solution: Consider a combined DFA and check if the start states are equivalent using the table filling algorithm.
DFA Minimisation
Problem 3: Design an algorithm that given a regular language $L$, outputs a DFA $A$ with minimum number of states such that $L(A) = L$.

Main idea: Partition the state space into equivalent “blocks” of states and define a DFA over these blocks.
Problem 3: Design an algorithm that given a regular language $L$, outputs a DFA $A$ with minimum number of states such that $L(A) = L$.

Main idea: *Partition the state space into equivalent “blocks” of states and define a DFA over these blocks.*

Claim 1: State equivalence is transitive.
Problem 3: Design an algorithm that given a regular language $L$, outputs a DFA $A$ with minimum number of states such that $L(A) = L$.

Main idea: Partition the state space into equivalent “blocks” of states and define a DFA over these blocks.

Claim 1: State equivalence is transitive.

Claim 2: Let $S$ denote any block (equivalence class) and let $a$ be any alphabet. All transitions from states within $S$ on $a$ is to states within the same block $S'$ (this may be different than $S$).
Problem 3: Design an algorithm that given a regular language \( L \), outputs a DFA \( A \) with minimum number of states such that \( L(A) = L \).

Main idea: Partition the state space into equivalent “blocks” of states and define a DFA over these blocks.

Claim 1: State equivalence is transitive.
Claim 2: Let \( S \) denote any block (equivalence class) and let \( a \) be any alphabet. All transitions from states within \( S \) on \( a \) is to states within the same block \( S' \) (this may be different than \( S \)).

Equivalent DFA \( B \):
- States are equivalence classes.
- Transitions are as per Claim 2.
- Start state is the block containing the start state of \( A \).
- A block is an accepting state iff all the states within the block are accepting.
DFA Minimization

Ragesh Jaiswal, CSE, IIT Delhi

CSL705: Theory of Computation
DFA Minimization

Ragesh Jaiswal, CSE, IIT Delhi

CSL705: Theory of Computation
Problem 3: Design an algorithm that given a regular language \( L \), outputs a DFA \( A \) with minimum number of states such that \( L(A) = L \).

Main idea: Partition the state space into equivalent “blocks” of states and define a DFA over these blocks.

Claim 1: State equivalence is transitive.

Claim 2: Let \( S \) denote any block (equivalence class) and let \( a \) be any alphabet. All transitions from states within \( S \) on \( a \) is to states within the same block \( S' \) (this may be different than \( S \)).

Equivalent DFA \( B \):
- States are equivalence classes.
- Transitions are as per Claim 2.
- Start state is the block containing the start state of \( A \).
- A block is an accepting state iff all the states within the block are accepting.

Claim 3: There cannot exist another DFA \( N \) with smaller number of states such that \( L(N) = L(A) \).
Problem 3: Design an algorithm that given a regular language \( L \), outputs a DFA \( A \) with minimum number of states such that \( L(A) = L \).

Main idea: *Partition the state space into equivalent “blocks” of states and define a DFA over these blocks.*

Claim 1: State equivalence is transitive.

Claim 2: Let \( S \) denote any block (equivalence class) and let \( a \) be any alphabet. All transitions from states within \( S \) on \( a \) is to states within the same block \( S' \) (this may be different than \( S \)).

Equivalent DFA \( B \):
- States are equivalence classes.
- Transitions are as per Claim 2.
- Start state is the block containing the start state of \( A \).
- A block is an accepting state iff all the states within the block are accepting.

Claim 3: There cannot exist another DFA \( N \) with smaller number of states such that \( L(N) = L(A) \).

Claim 4: \( B \) is the unique DFA with minimum number of states that is equivalent to \( A \).
Myhill-Nerode Theorem
Myhill-Nerode Theorem

- Let $L$ be any language over $\Sigma^*$. We say that strings $x$ and $y$ in $\Sigma^*$ are indistinguishable by $A$ iff for every string $z \in \Sigma^*$ either both $xz$ and $yz$ are in $L$ or both $xz$ and $yz$ are not in $L$. We write $x \equiv_L y$ in this case.

- **Claim 1**: $\equiv_L$ is an equivalence relation.

- Given DFA $M = (Q, \Sigma, \delta, s, F)$ we say that two strings $x, y \in \Sigma^*$ are indistinguishable by $M$ iff $\hat{\delta}(s, x) = \hat{\delta}(s, y)$. We write $x \equiv_M y$ in this case.

- **Claim 2**: $\equiv_M$ is an equivalence relation and there are finite number of equivalence classes.

- **Claim 3**: If $L = L(M)$ for a DFA $M$, then for any $x, y \in \Sigma^*$, if $x \equiv_M y$, then $x \equiv_L y$.

- **Claim 4**: If $L$ is a regular language, then $\equiv_L$ has a finite number of equivalence classes.

---

**Theorem (Myhill-Nerode Theorem)**

$L$ is regular if and only if $\equiv_L$ has a finite number of equivalence classes. Furthermore, there is a DFA $M$ with $L(M) = L$ having precisely one state for each equivalence class of $\equiv_L$. 

---

Ragesh Jaiswal, CSE, IIT Delhi  
CSL705: Theory of Computation
End