MA/CSSE 474

Theory of Computation

DFSM Minimization

Finite State Machines

State Minimization

Among all DSFMs that are equivalent to a given DFSM, find one whose number of states is minimal

Defining Equivalence for Strings

We want to capture the notion that two strings are equivalent or ***indistinguishable*** with respect to a language L if, no matter what string w tacked on to them on the right, either both concatenated strings will be in L or neither will. **Why is this the right notion?** Because it corresponds naturally to what the states of a recognizing FSM have to remember.

Example:

(1) a b a b a b

(2) b a a b a b

Suppose $L = \{w \in \{\texttt{a}, \texttt{b}\}^* : |w| \text{ is even}\}$. Are (1) and (2) equivalent?

Suppose $L = \{w \in \{\texttt{a}, \texttt{b}\}^*$: every a is immediately followed by b$\}$. Are (1) and (2) equivalent?

Q1a

Defining Equivalence for Strings

If two strings are indistinguishable with respect to L, we write:

$$x \approx_L y$$

Formally, $x \approx_L y$ iff $\forall z \in \Sigma^* \ (xz \in L \text{ iff } yz \in L)$.

Show that $\approx_L$ is an equivalence relation

Q1b,2

$\approx_L$ is an Equivalence Relation

An equivalence relation on a set partitions the set.
Thus:

- No equivalence class of $\approx_L$ is empty.
- Each string in Σ^* is in exactly one equivalence class of $\approx_L$.
- Notation: For any x in Σ^*, [x] means the equivalence class of $\approx_L$ that contains x.

An Example

$\Sigma = \{a, b\}$
$L = \{w \in \Sigma^*$: every a is immediately followed by b$\}$

The equivalence classes of $\approx_L$: Try:

ε	aa	bbb
a	bb	baa
b	aba	
	aab	

An Example

$\Sigma = \{$a, b$\}$
$L = \{w \in \Sigma^*$: every a is immediately followed by b$\}$

The equivalence classes of $\approx_L$:

[1]	[ε, b, abb, …]	[all strings in L].
[2]	[a, abbba, …]	[all strings that end in a and have no prior a that is not followed by a b].
[3]	[aa, abaa, …]	[all strings that contain at least one instance of aa].

Another Example of $\approx_L$

$\Sigma = \{$a, b$\}$
$L = \{w \in \Sigma^* : |w| \text{ is even}\}$

ε	bb	aabb
a	aba	bbaa
b	aab	aabaa
aa	bbb	
	baa	

The equivalence classes of $\approx_L$:

Yet Another Example of $\approx_L$

Σ = {a, b}
L = aab*a

ε	bb	aabaa
a	aba	aabbba
b	aab	aabbaa
aa	baa	
	aabb	

The equivalence classes of $\approx_L$:

A good example for practice later

When More Than One Class Contains Strings in *L*

Σ = {a, b}
L = {$w \in \Sigma^*$: no two adjacent characters are the same}

ε	aa	aabb
a	bb	aabaa
b	aba	aabbba
	aab	aabbaa
	baa	

The equivalence classes of $\approx_L$:

When More Than One Class Contains Strings in *L*

$\Sigma = \{a, b\}$
$L = \{w \in \Sigma^* :$ no two adjacent characters are the same$\}$

The equivalence classes of $\approx_L$:

[1] [ε]
[2] [a, aba, ababa, …]
[3] [b, ab, bab, abab, …]
[4] [aa, abaa, ababb…]

Does $\approx_L$ Always Have a Finite Number of Equivalence Classes?

$\Sigma = \{a, b\}$
$L = \{a^n b^n, n \geq 0\}$

ε	aa	aaaa
a	aba	aaaaa
b	aaa	

The equivalence classes of $\approx_L$:

The Best We Can Do

Theorem: Let L be a regular language and let M be a DFSM that accepts L. The number of states in M is greater than or equal to the number of equivalence classes of $\approx_L$.

Proof: Suppose that the number of states in M were less than the number of equivalence classes of $\approx_L$. Then, by the pigeonhole principle, there must be at least one state q that contains strings from at least two equivalence classes of $\approx_L$. But then M's future behavior on those strings will be identical, which is not consistent with the fact that they are in different equivalence classes of $\approx_L$.

The Best Is Unique

Theorem: Let L be a regular language over some alphabet Σ. Then there is a DFSM M that accepts L and that has precisely n states where n is the number of equivalence classes of $\approx_L$. Any other FSM that accepts L must either have more states than M or it must be equivalent to M except for state names.

Proof: (by construction)

$M = (K, \Sigma, \delta, s, A)$, where:

- K contains n states, one for each equivalence class of $\approx_L$.
- $s = [\varepsilon]$, the equivalence class of ε under $\approx_L$.
- $A = \{[x] : x \in L\}$.
- $\delta([x], a) = [xa]$. In other words, if M is in the state that contains some string x, then, after reading the next symbol, a, it will be in the state that contains xa.

Q3

Proof, Continued

We must show that:

- K is finite. Since L is regular, it is accepted by some DFSM M'. M' has some finite number of states m. By Theorem 5.4, $n \leq m$. So K is finite.

- δ is a function. In other words, it is defined for all (state, input) pairs and it produces, for each of them, a unique value. The construction defines a value of δ for all (state, input) pairs. The fact that the construction guarantees a unique such value follows from the definition of $\approx_L$.

Proof, Continued

- $L = L(M)$. To prove this, we must first show that $\forall s, t$ (([ε], st) $|\!-_M{}^*$ ([s], t)). We do this by induction on $|s|$.

  If $|s| = 0$ then we have ([ε], t) $|\!-_M{}^*$ ([ε], t), which is true since M simply makes zero moves.

On to the induction step:

Q4, 5, 6

Proof, Continued

- There exists no smaller machine *M#* that also accepts *L*. This follows directly from Theorem 5.4, which says that the number of equivalence classes of $\approx_L$ imposes a lower bound on the number of states in any DFSM that accepts *L*.

- There is no different machine *M#* that also has *n* states and that accepts *L*.

Constructing the Minimal DFA from $\approx_L$

$\Sigma = \{$a, b$\}$
$L = \{w \in \Sigma^*$: no two adjacent characters are the same$\}$

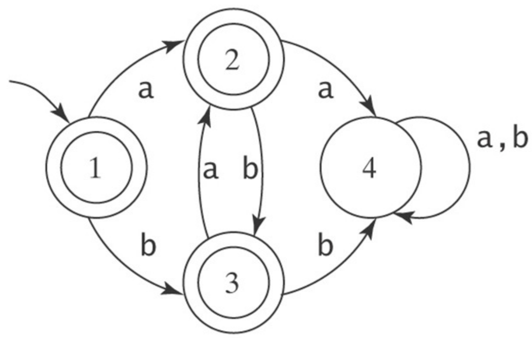
The equivalence classes of $\approx_L$:

1: [ε]	ε
2: [a, ba, aba, baba, ababa, ...]	(b∪ε)(ab)*a
3: [b, ab, bab, abab, ...]	(a∪ε)(ba)*b
4: [bb, aa, bba, bbb, ...]	the rest

- Equivalence classes become states
- Start state is [ε]
- Accepting states are all equivalence classes in *L*
- $\delta([x], a) = [xa]$

Constructing the Minimal DFA from $\approx_L$

$\Sigma = \{a, b\}$
$L = \{w \in \Sigma^* :$ no two adjacent characters are the same$\}$



The Myhill-Nerode Theorem

Theorem: A language is regular iff the number of equivalence classes of $\approx_L$ is finite.

Proof: *S*how the two directions of the implication:

L regular → the number of equivalence classes of $\approx_L$ is finite: If L is regular, then there exists some FSM M that accepts L. M has some finite number of states m. The cardinality of $\approx_L \leq m$. So the cardinality of $\approx_L$ is finite.

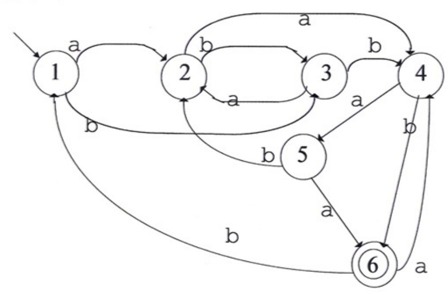
The number of equivalence classes of $\approx_L$ is finite → L regular: If the cardinality of $\approx_L$ is finite, then the construction that was described in the proof of the previous theorem will build an FSM that accepts L. So L must be regular.

So Where Do We Stand?

1. We know that for any regular language L there exists a minimal accepting machine M_L.
2. We know that $|K|$ of M_L equals the number of equivalence classes of $\approx_L$.
3. We know how to construct M_L from $\approx_L$.
4. We know that M_L is unique up to the naming of its states.

But is this good enough?

Consider:



Minimizing an Existing DFSM (Without Knowing $\approx L$)

Two approaches:

- Begin with M and collapse redundant states, getting rid of one at a time until the resulting machine is minimal.
- Begin by overclustering the states of L into just two groups, accepting and nonaccepting. Then iteratively split those groups apart until all the distinctions that L requires have been made.

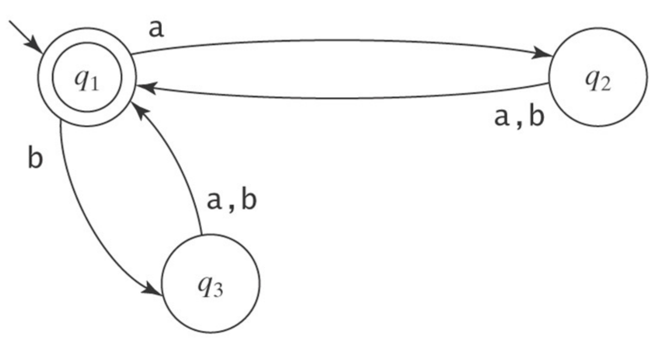
The Overclustering Approach

We need a definition for "equivalent", i.e., mergeable states.

Define $q \equiv p$ iff for all strings $w \in \Sigma^*$, either w drives M to an accepting state from both q and p or it drives M to a rejecting state from both q and p.

An Example

$\Sigma = \{\texttt{a}, \texttt{b}\}$ $\quad$ $L = \{w \in \Sigma^* : |w| \text{ is even}\}$



$q_2 \equiv q_3$

Constructing $\equiv$ as the Limit of a Sequence of Approximating Equivalence Relations $\equiv^n$

(Where *n* is the length of the input strings that have been considered so far)

Consider input strings, starting with ε, and increasing in length by 1 at each iteration. Start by way overgrouping states. Then split them apart as it becomes apparent (with longer and longer strings) that their behavior is not identical.

Constructing $\equiv_n$

- $p \equiv^0 q$ iff they behave equivalently when they read ε. In other words, if they are both accepting or both rejecting states.
- $p \equiv^1 q$ iff they behave equivalently when they read any string of length 1, i.e., if any single character sends both of them to an accepting state or both of them to a rejecting state. Note that this is equivalent to saying that any single character sends them to states that are $\equiv^0$ to each other.
- $p \equiv^2 q$ iff they behave equivalently when they read any string of length 2, which they will do if, when they read the first character they land in states that are $\equiv^1$ to each other. By the definition of $\equiv^1$, they will then yield the same outcome when they read the single remaining character.
- And so forth.

Constructing ≡, Continued

More precisely, $\forall p,q \in K$ and any $n \geq 1$, $q \equiv^n p$ iff:

1. $q \equiv^{n-1} p$, and
2. $\forall a \in \Sigma\ (\delta(p, a) \equiv^{n-1} \delta(q, a))$

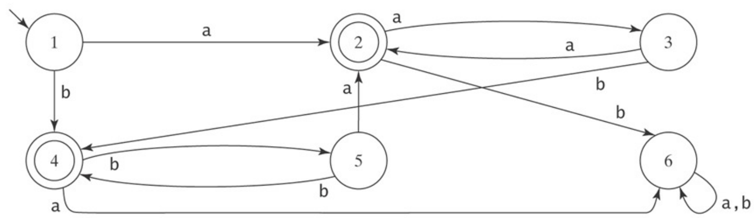
MinDFSM

MinDFSM(*M*: DFSM) =

1. *classes* := {*A*, *K*-*A*};
2. Repeat until no changes are made
   - 2.1. *newclasses* := ∅;
   - 2.2. For each equivalence class *e* in *classes*, if *e* contains more than one state do
     - For each state *q* in *e* do
       - For each character *c* in Σ do
         - Determine which element of *classes* *q* goes to if *c* is read
     - If there are any two states *p* and *q* that need to be split, split them. Create as many new equivalence classes as are necessary. Insert those classes into *newclasses*.
     - If there are no states whose behavior differs, no splitting is necessary. Insert *e* into *newclasses*.
   - 2.3. *classes* := *newclasses*;
3. Return $M^* = (classes, \Sigma, \delta, [s_M], \{[q]: \text{the elements of } q \text{ are in } A_M\})$, where δ_{M^*} is constructed as follows:
   - if $\delta_M(q, c) = p$, then $\delta_{M^*}([q], c) = [p]$

An Example

$\Sigma = \{a, b\}$

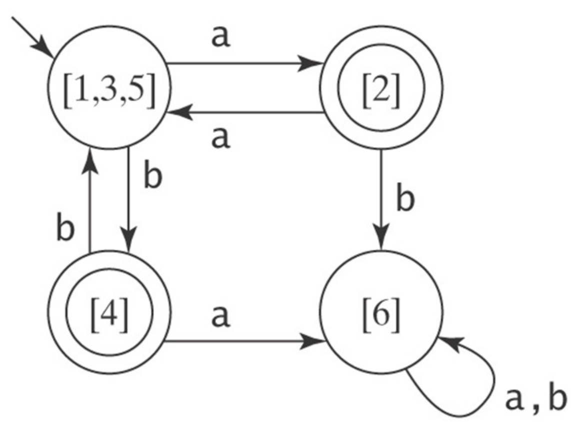


$\equiv^0$ =

$\equiv^1$ =

$\equiv^2$ =

The Result



Q7

Summary

- Given any regular language *L*, there exists a minimal DFSM *M* that accepts *L*.
- *M* is unique up to the naming of its states.
- Given any DFSM *M*, there exists an algorithm *minDFSM* that constructs a minimal DFSM that also accepts *L*(*M*).

Canonical Forms

A ***canonical form*** for some set of objects *C* assigns exactly one representation to each class of “equivalent” objects in *C*.

Further, each such representation is distinct, so two objects in *C* share the same representation iff they are “equivalent” in the sense for which we define the form.

A Canonical Form for FSMs

buildFSMcanonicalform(M: FSM) =

1. M' = *ndfsmtodfsm*(M).
2. M^* = *minDFSM*(M').
3. Create a unique assignment of names to the states of M^*.
4. Return M^*.

Given two FSMs M_1 and M_2:

buildFSMcanonicalform(M_1)

=

buildFSMcanonicalform(M_2)

iff $L(M_1) = L(M_2)$.