MA/CSSE 474

Theory of Computation

Finish NDFSM→DFSM

Minimize # states in a DFSM

Your Questions?

- Previous class days' material
- Reading Assignments
- HW2 solutions
- HW3 or HW4
- Tuesday's Exam
- Anything else

Another example of why we often need formal specifications instead of natural language

Nondeterministic and Deterministic FSMs

Clearly: {Languages accepted by some DFSM} $\subseteq$ {Languages accepted by some NDFSM}

More interesting:

***Theorem**:*

For each NDFSM, there is an equivalent DFSM.

"equivalent" means "accepts the same language"

Nondeterministic and Deterministic FSMs

***Theorem**:* For each NDFSM, there is an equivalent DFSM.

***Proof**:* By construction:

Given a NDFSM $M = (K, \Sigma, \Delta, s, A)$,
we construct $M' = (K', \Sigma, \delta', s', A')$, where

More precisely, each state in K' is the string that encodes a subset of K, using the standard set notation. **Example:** "{q0, q2}" We omit the quotation marks.

$K' \subseteq \mathcal{P}(K)$
$s' = eps(s)$
$A' = \{Q \subseteq K' : Q \cap A \neq \varnothing\}$

$\delta'(Q, a) = \bigcup\{eps(p): p \in K$ and $(q, a, p) \in \Delta$ for some $q \in Q\}$

Think of the simulator as the "interpreted version" and this as the "compiled version".

An Algorithm for Constructing the Deterministic FSM

1. Compute the *eps*(q)'s.

2. Compute s' = *eps*(s).

3. Compute δ'.

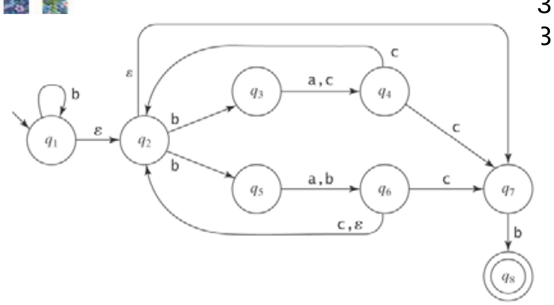
4. Compute K' = a subset of $\mathcal{P}(K)$.

5. Compute $A' = \{Q \in K' : Q \cap A \neq \varnothing\}$.

The Algorithm *ndfsmtodfsm*

ndfsmtodfsm(M: NDFSM) =
1. For each state q in K_M do:
    1.1 Compute *eps*(q).
2. s' = *eps*(s)
3. Compute δ':
    3.1 *active-states* = $\{s'\}$.
    3.2 $\delta' = \varnothing$.
    3.3 While there exists some element Q of *active-states* for which δ' has not yet been computed do:
        For each character c in Σ_M do:
            new-state = $\varnothing$.
            For each state q in Q do:
                For each state p such that $(q, c, p) \in \Delta$ do:
                    new-state = *new-state* $\cup$ *eps*(p).
            Add the transition (q, c, *new-state*) to δ'.
            If *new-state* $\notin$ *active-states* then insert it.
4. K' = *active-states*.
5. $A' = \{Q \in K' : Q \cap A \neq \varnothing\}$.

Draw part of the transition diagram for the DFSM constructed from the NDFSM that appeared a few slides earlier.



Later we may prove that this works for all NDFSMs M.

Finite State Machines

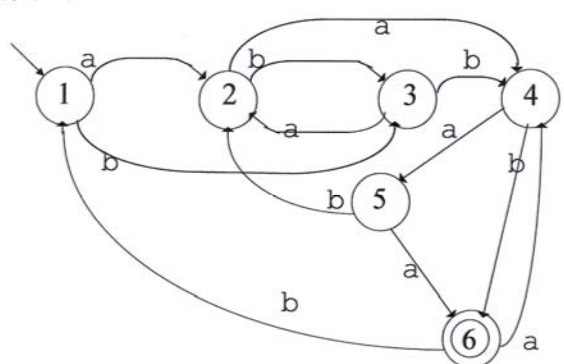
Intro to State Minimization

Among all DFSMs that are equivalent to a given DFSM, can we find one whose number of states is minimal?

Note that this is a different question from "Is there an equivalent machine with a minimal number of states?", which has an obvious answer.

State Minimization

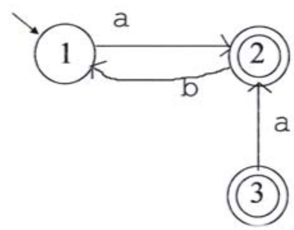
Consider:



Is this a minimal machine?
It's not immediately obvious!
We need tools!

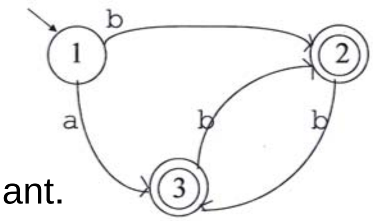
State Minimization

Step (1): Get rid of unreachable states.



State 3 is unreachable.

Step (2): Get rid of redundant states.

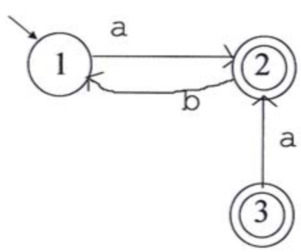


States 2 and 3 are redundant.

Getting Rid of Unreachable States

We can't easily find the unreachable states directly. But we can find the reachable ones and determine the unreachable ones from there.

An algorithm for finding the reachable states:

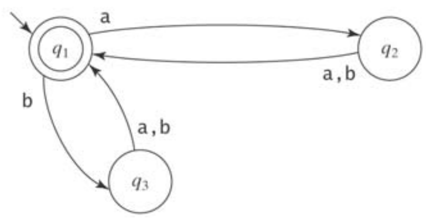


Like many algorithms from this course, the structure is "add things until nothing new can be added"

Getting Rid of Redundant States

Intuitively, two states are equivalent to each other (and thus one is redundant) if, starting in those states, all strings in Σ* have the same fate, regardless of which of the two states the machine is currently in.
But how can we tell this?

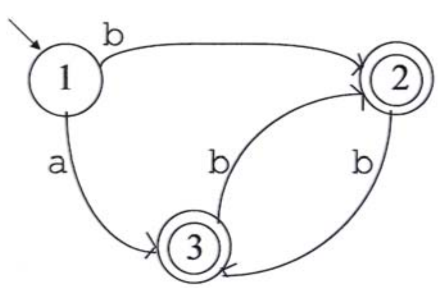
The simple case:



Two states have identical sets of transitions out.

Getting Rid of Redundant States

The harder case:



The outcomes in states 2 and 3 are the same, even though the states aren't.

An Algorithm for Minimization

Capture the notion of equivalence classes of strings with respect to a language.

Prove that we can always find a (unique up to state naming) a deterministic FSM with a number of states equal to the number of equivalence classes of strings.

Describe an algorithm for finding that deterministic FSM.

Equivalent Strings (w.r.t. L)

We say that two strings x and y are **equivalent** or ***indistinguishable*** with respect to a language *L* if,

no matter what string z is appended to both,
either both concatenated strings will be in *L* or neither will.

Write it in first-order logic:

$x \approx_L y$ iff

Example:

x: a **y:** bab

Suppose $L_1 = \{w \in \{a, b\}^* : |w| \text{ is even}\}$. Are **x** and **y** equivalent?
Suppose $L_2 = \{w \in \{a, b\}^* :$ every a is immediately followed by b$\}$.
Are **x** = a and **y** = aa equivalent with respect to L_2?

$\approx_L$ is an Equivalence Relation

$\approx_L$ is an equivalence relation because it is:

- Reflexive: $\forall x \in \Sigma^* (x \approx_L x)$, because:
  $\forall x, z \in \Sigma^* (xz \in L \leftrightarrow xz \in L)$.
- Symmetric: $\forall x, y \in \Sigma^* (x \approx_L y \rightarrow y \approx_L x)$, because:
  $\forall x, y, z \in \Sigma^* ((xz \in L \leftrightarrow yz \in L) \leftrightarrow$
  $(yz \in L \leftrightarrow xz \in L))$.
- Transitive: $\forall x, y, z \in \Sigma^* (((x \approx_L y) \wedge (y \approx_L w)) \rightarrow (x \approx_L w))$,
  because:
  $\forall x, y, z \in \Sigma^*$
  $(((xz \in L \leftrightarrow yz \in L) \wedge (yz \in L \leftrightarrow wz \in L)) \rightarrow$
  $(xz \in L \leftrightarrow wz \in L))$.

Because $\approx_L$ is an Equivalence Relation

An equivalence relation on a set partitions that set into equivalence classes

.

Thus:

- No equivalence class of $\approx_L$ is empty.
- Each string in Σ^* is in exactly one equivalence class of $\approx_L$.

An Example

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

What are the equivalence classes of $\approx_L$?

Hint: Try:

ε	aa	bbb
a	bb	baa
b	aba	
	aab	

Recall that x $\approx_L$ y iff $\forall z \in \Sigma^*$ ($xz \in L \leftrightarrow yz \in L$).

Another Example of $\approx_L$

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

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

The equivalence classes of $\approx_L$:

Recall that x $\approx_L$ y iff $\forall z \in \Sigma^*$ ($xz \in L \leftrightarrow yz \in L$).

Yet Another Example of $\approx_L$

$\Sigma = \{a, b\}$
L = aab*a

Do this one for practice later

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

The equivalence classes of $\approx_L$:

Recall that x $\approx_L$ y iff $\forall z \in \Sigma^* (xz \in L \leftrightarrow yz \in L)$.

More Than One Class Can Contain Strings in *L*

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

The equivalence classes of $\approx_L$:

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

Recall that x $\approx_L$ y iff $\forall z \in \Sigma^* (xz \in L \leftrightarrow yz \in L)$.

One More Example of $\approx_L$

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

ε	aa	aaaa
a	aba	aaaaa
b	aaa	

The equivalence classes of $\approx_L$:

Recall that x $\approx_L$ y iff $\forall z \in \Sigma^* (xz \in L \leftrightarrow yz \in L)$.