Minimal FA's

Decision Properties of Regular Languages

Decision Properties

- Questions we can ask about regular languages and how we answer such questions.

Languages

- Recall.
  - What is a language?
  - What is a class of languages?

Regular Languages

- What we know about regular languages
  - Described using regular expressions
    - Set operations of union, concatenation, Kleene Star
  - Kleene Theorem
    - A language is regular iff there exists a finite automata that accepts the language

Closure Properties of Regular Languages

- Regular languages are closed under
  - Union
  - Concatenation
  - Kleene Star
  - Intersection
  - Difference
  - Complement

Finite Automata

- What we know about finite automata
  - Three equivalent variants
    - Finite Automata (Deterministic)
    - Non-deterministic Finite Automata
    - Non-deterministic Finite Automata with Λ transitions
  - Can convert from one to the other

Regular Languages

- Means of specifying a regular language
  1. Regular Expression
  2. Finite Automata (Deterministic)
  3. Non-deterministic Finite Automata
  4. Non-deterministic Finite Automata with Λ transitions

We can convert from one specification to another.

Decision Properties

- Given regular languages, specified in any one of the four means, can we develop algorithms that answer the following questions:
  1. Is the language empty?
  2. Is the language finite?
  3. Is a given string in the language?
  4. Given two languages, are there strings that are in both?
  5. Is the language a subset of another regular language?
  6. Is the language the same as another regular language?

Non-regular languages

- And then there's the question:
  - Is there a language L that is not regular?

Decision Properties

- Some of these questions we can answer already with what we know
  - Is a given string in the language?
  - Given two languages are there strings that are in both?
- Others require additional tools:
  - Is the language the same as another regular language?
  - Is there a language L that is not regular?

Additional tools

- Minimal finite automata
  - Create a finite automata with the minimum number of possible states
- Pumping Lemma
  - Defines necessary properties for strings in a regular language
  - Can be used to show that languages are not regular
  - We will consider this next week…

Minimal Finite Automata

- Motivation
  - Consider the question:
    - Do two finite automata accept the same language?
  - To answer, we introduce the Minimal Finite Automata (MFA)
    - Given an FA, create a new FA with the minimal number of states possible that accepts the same language.

Minimal Finite Automata

- Motivation
  - Given the question:
    - Do two finite automata accept the same language?
  - Answer:
    - We can generate the MFA for each FA, then compare the MFAs on a state by state basis.

Minimal Finite Automata

- Plan
  - Equivalent states of an FA
  - Devise an algorithm (based on equivalent states) that creates a minimal FA from an FA
  - Some examples

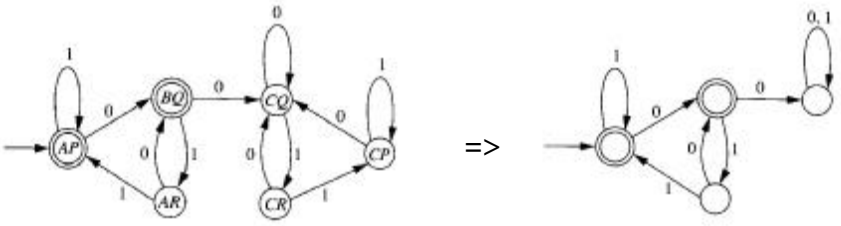
Minimal Finite Automata

- Equivalent States
  - $M = (Q, \Sigma, q_0, A, \delta)$
  - Two states, $p, q \in Q$ are said to be <u>equivalent</u> if
    - For all strings $x \in \Sigma^*$
    - $\delta^*(p, x)$ is in an accepting state iff $\delta^*(q, x)$ is in an accepting state
      - If $\delta^*(p, x)$ is an accepting state then $\delta^*(q, x)$ is an accepting state
      - If $\delta^*(p, x)$ is not an accepting state then $\delta^*(q, x)$ is not an accepting state
  - If two states are not equivalent, they are said to be <u>distinguishable</u>.

Minimal Finite Automata

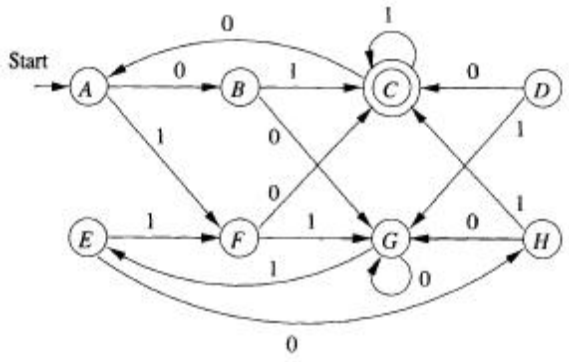
- Equivalent States
  - In building a MFA, equivalent states can be combined.

Minimal Finite Automata



Minimal Finite Automata

- Example



Minimal Finite Automata

- Example:
  - States C and G are distinguishable
    - One is accepting, one is not
  - States A and G are distinguishable
    - δ^* (A, 01) = C (accepting)
    - δ^* (G, 01) = E (not-accepting)

Minimal Finite Automata

- Example:
  - States B and H are equivalent
    - δ (B, 1) = δ (H, 1) = C
      - δ^* (B, 1x) = δ^* (H, 1x) for any x
    - δ (B, 0) = δ (H, 0) = G
      - δ^* (B, 0x) = δ^* (H, 0x) for any x
    - So for any x, δ^* (B, x) and δ^* (H, x) will either both be accepting or both not be accepted.

Minimal Finite Automata

- Example:
  - States A and E are equivalent
    - δ (A, 1) = δ (E, 1) = F
      - δ^* (A, 1x) = δ^* (E, 1x) for any x
    - δ (A, 0) = B, δ (E, 0) = H
      - B and H are equivalent
      - δ^* (A, 0x) and δ^* (E, 0x) will either both be accepting or both be non-accepting.

Minimal Finite Automata

- Recursive algorithm to find distinguishable states:
  - Consider pairs {p,q}
  - For each pair we will determine whether p is distinguishable from q
  - Said another way, for each pair {p,q} we will determine if p is not equivalent to q.

Minimal Finite Automata

- Recursive algorithm
  - Base case:
    - If p is accepting and q is non-accepting then {p,q} is distinguishable
  - Induction
    - For some pair {p,q} if
      - δ (p,a) = r and δ (q,a) = s and
      - {r,s} is distinguishable then
      - {p,q} is distinguishable

Minimal Finite Automata

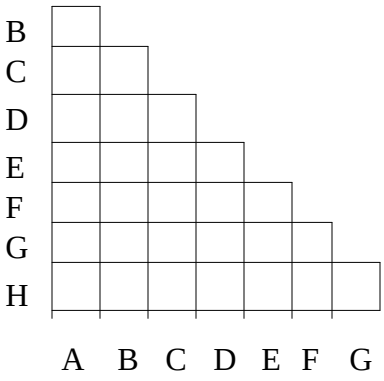
- Let's take a look at this induction step
  - If $r = \delta(p,a)$ and $s = \delta(q,a)$ are distinguishable, then there is a string x such that $\delta(r,x)$ is accepting and $\delta(s,x)$ is not, or visa-versa
  - Then for x, $\delta(p,ax)$ is accepting and $\delta(q,ax)$ is not, or visa-versa.
  - We found a string, ax such that $\delta(p,ax)$ is accepting and (q,ax) is not (or visa-versa), thus {p,q} are distinguishable

Minimal Finite Automata

- This algorithm is sometime best visualized by using a table with each table cell representing a pair of states. A mark in a table cell indicates that the two states of the pair are distinguishable.

Minimal Finite Automata

- Distinguishable table



Minimal Finite Automata

- Restatement of algorithm
  - For all pairs {p,q} such that p is accepting and q is not, mark the equivalent cell in the table.
  - Consider each pair {p,q} not yet marked.
    - Determine $r = \delta(p,a)$ and $s = \delta(q,a)$ for each a in Σ.
    - If {r,s} is marked, then mark {p,q}
  - Repeat until no further cells are marked during an iteration of the algorithm

Minimal Finite Automata

- Example

δ (A, 0) = B	δ (A, 1) = F
δ (B, 0) = G	δ (B, 1) = **C**
δ (C, 0) = A	δ (C, 1) = **C**
δ (D, 0) = **C**	δ (D, 1) = G
δ (E, 0) = H	δ (E, 1) = F
δ (F, 0) = **C**	δ(F, 1) = G
δ (G, 0) = G	δ(G, 1) = E
δ (H, 0) = G	δ(H, 1) = **C**

Minimal Finite Automata

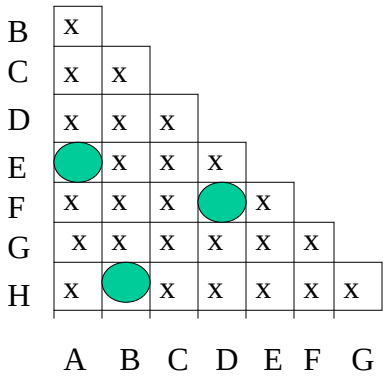
- Let's try on our example

	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

Minimal Finite Automata

- Once our table is complete
  - All unmarked cells correspond to state pairs that are not-distinguishable, i.e. they are equivalent
  - Combine equivalent states into one
  - Transitions from equivalent states should map to equivalent states

Minimal Finite Automata



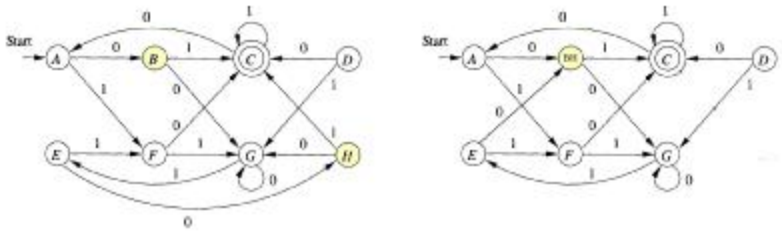
E and A are equivalent

H and B are equivalent

D and F are equivalent

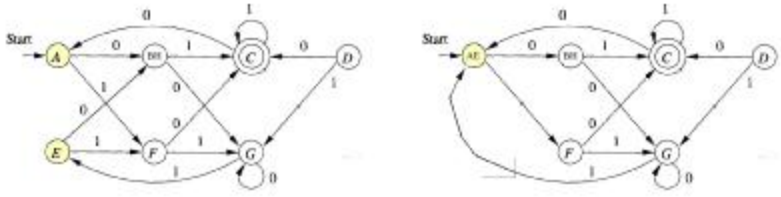
Minimal Finite Automata

- Combine H and B



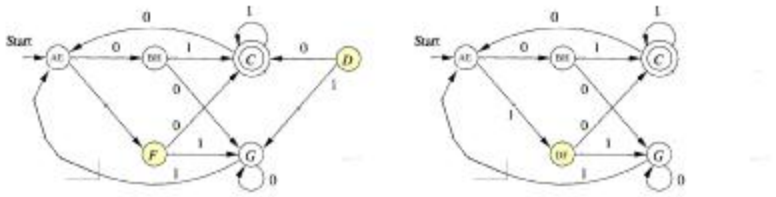
Minimal Finite Automata

- Combine E and A



Minimal Finite Automata

- Combine D and F



Minimal Finite Automata

- What have we done?
  - Defined the notion of equivalent states
  - Developed a recursive algorithm to determine which states in an FA are equivalent
  - Combine equivalent states to create FA with minimal number of states.

Minimal Finite Automata

- Let's revisit the question:
  - Given 2 specifications of regular languages, do the specifications describe the same language.
    - Create a MFA for each language
    - Compare the MFAs on a state by state basis.

For the mathematically minded

- Let's go back to our Discrete Math
  - Relation
    - Defines relationship between objects
    - Usually given as an ordered pair,
      - (x, y) where x,y ∈ some Set
  - Equivalence relation
    - Reflective: (a, a) - aRa
    - Symmetric: if (a,b) then (b,a) – aRb -> bRa
    - Transitive: if (a,b) and (b,c) then (a,c) – aRb and bRc -> aRc

For the mathematically minded

- Equivalence relations
  - The nice thing about equivalence relations
    - It partitions the elements of your set into a number of distinct and disjoint subsets.
    - Each subset is called an equivalence class

For the mathematically minded

- MFA and Equivalence Classes
  - Distinguishablitilty can be shown to be an equivalence relation on a language.
  - This relation partitions the strings of L into a number of equivalence classes.
  - Each equivalence class corresponds to a state in the MFA.

Decision Properties

- Recall:
  - One reason for all this theory is to create a model for computation
  - Develop algorithms for problems using theoretical machines
  - Decision problems for regular languages
    - "yes"/"no" problems

Decision Properties

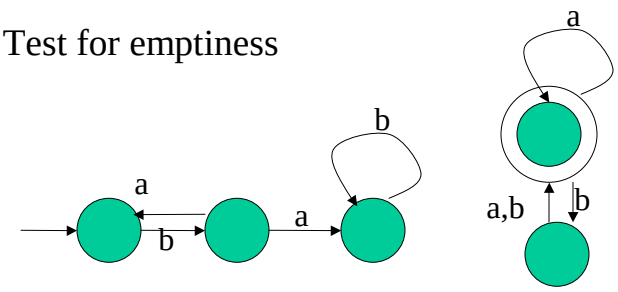
- Given regular languages, specified in any one of the four means, can we develop algorithms that answer the following questions:
  1. Is the language empty?
  2. Is the language finite? (save for next time)
  3. Is a given string in the language?
  4. Given two languages, are there strings that are in both?
  5. Is the language a subset of another regular language?
  6. Is the language the same as another regular language?

Decision properties

- Test for emptiness
  - Obtain the FA for the regular language
  - L(M) will be empty if either
    - M has no accepting states
    - There is no path from the start state to any of the accepting states.
      - Can determine this by systematically mapping all paths from the start state

Decision Problems

- Test for emptiness



- There is no path from the start to an accepting state

Decision properties

- Test for membership
  - Given a language L and a string x, is x in L?
  - Obtain the FA for the language
  - Run the machine on input x
    - If the machine accepts, x in L
    - Otherwise, x not in L

Decision properties

- Test for inclusion in 2 languages
  - Given two languages L_1 and L_2, is there a string x that is in both.
  - The same as asking $x \in (L_1 \cap L_2)$
  - We know how to build an FA for $L_1 \cap L_2$ from the FAs that accept these languages.
  - Run x through this machine
    - If the machine accepts, x in both L_1 and L_2
    - Otherwise, x not in L_1 and L_2

Decision properties

- Test for subset
  - Given tw regular languages L_1 and L_2
    - is $L_1 \subseteq L_2$
  - This is the same as asking
    - Is $L_1 - L_2 = \varnothing$
  - We know how to build a FA for L_1- L_2 from the FAs that accept these languages
  - Apply the empty test to the resultant FA

Decision properties

- Test for equivalence
  - Given specifications for two regular languages, do the specifications describe the same language?
    - Create the MFA for each
    - Compare on a state by state basis.

Summary

- Decision Problems for regular languages
- New tools
  - Minimal Finite Automata
  - Pumping Lemma (soon…)