CS 3100, Models of Computation, Spring 20, Lec 7

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

bit.ly/3100fs20Syllabus

Lecture 3, covering Ch 5 and 6


5 Designing DFA **57**

5.1 Understanding the Language to Be Realized 57

5.1.1 The Language of Equal Changes 57

5.1.2 Best Practices to Correct DFA Design and Verification 58

5.2 Examples of Designing DFA 59

5.2.1 The Language of Blocks of 3 59

5.2.2 DFA for "Ends with 0101" 60

5.2.3 DFA for "MSB/LSB-first Binary Number is Divisible by 3" 60

5.2.4 DFA for "Third-last bit is a 1" 62

5.3 Automd: A Markdown Language for All Machines 64

5.3.1 Markdown for DFA 64

6 Operations on DFA **67**

6.1 Complementation of DFA 67

6.2 Union and Intersection of DFA 67

6.3 Language Equivalence and Isomorphism 71

6.4 DFA Minimization and Myhill-Nerode Theorem 72

6.4.1 Fully Worked-out Example of DFA Minimization . . 72

6.4.2 Salient Code Excerpts 75

6.5 Examples of Language Design and Manipulation 76

6.5.1 Use of Union, Minimization, and Language Equivalence 77

6.5.2 Use of DeMorgan's Law 78

Today: Algorithms around DFA

- Algorithms that help build more complex DFA out of simpler DFA
- Algorithms that minimize DFA into a unique minimal form
  - We will be running this notebook alongside, and study the code
    - First_Jove_Tutorial/CH4-5-6/L7-f19-exercises.ipynb
  - Quiz3 is very similar and Asg-2 also helps you build-up your understanding

Remember, if you can code something, you truly understand it!

*** coding is teaching… er, teaching a computer!**

*** if you teach the most unforgiving beast (a computer), you have spelled out things to the last detail**

Recap: Formal structure of a DFA

- Q is a *finite nonempty* set of states,
- Σ is a *finite nonempty* alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is a *total* transition function,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is a *finite, possibly empty* set of final (or *accepting*) states.

The language of a DFA is the set of strings that lead the DFA from one of the I states to an F state

Example: Language of strings over Sigma={0,1} that **"Begin with a 0"**

- List five strings in numeric order
- Draw a DFA
- Express it as (Q,Sigma,delta,q0,F)

- Now design **"End with a 1"**

Intersect “Begin with 0” and “End with 1”

Intersection has these kinds of strings (express in English): ” …. “

- List five strings in numeric order
- It is as if we have the DFA side by side, and fed them the same input string
- When both DFA enter a final state, accept the string seen so far!
  - Algorithm: March them in tandem (lock-step) – i.e. tie the DFA together and let them march together!

- Q = Q1 x Q2
- q0 = (q01, q02)
- F = { (f1, f2) : }
- delta((q1, q2), a) = (…dfa1’s next state… , …dfa2’s next state…)

Union “Begin with 0” and “End with 1”

- Q = Q1 x Q2
- q0 = (q01, q02)
- F = { (f1, f2) : }
- delta((q1, q2), a) = (…dfa1’s next state… , …dfa2’s next state…)

Complement “Begin with 0”

Complementation : a string is in the complement IFF it is not in the original DFA
Which of these change, and which ones stay the same? Then draw the true and complement side by side

- Q =
- q0 =
- F =
- delta (q, a) =

Language equivalence and isomorphism

Two DFA are language equivalent if they accept the same set of strings

They are isomorphic if they are language equivalent and have the same number of states

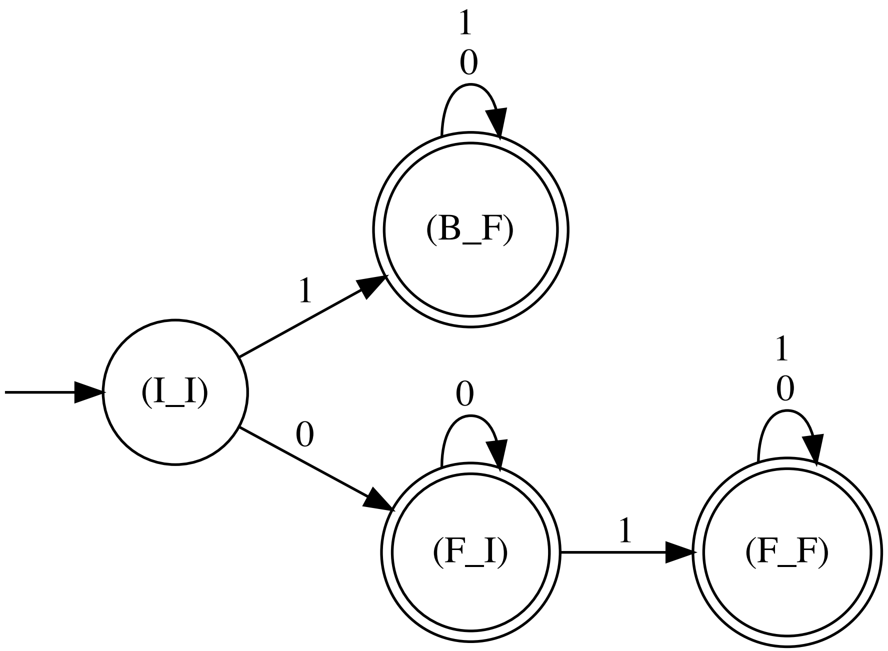
Then we can place one DFA on top of another, and their states and transitions will match

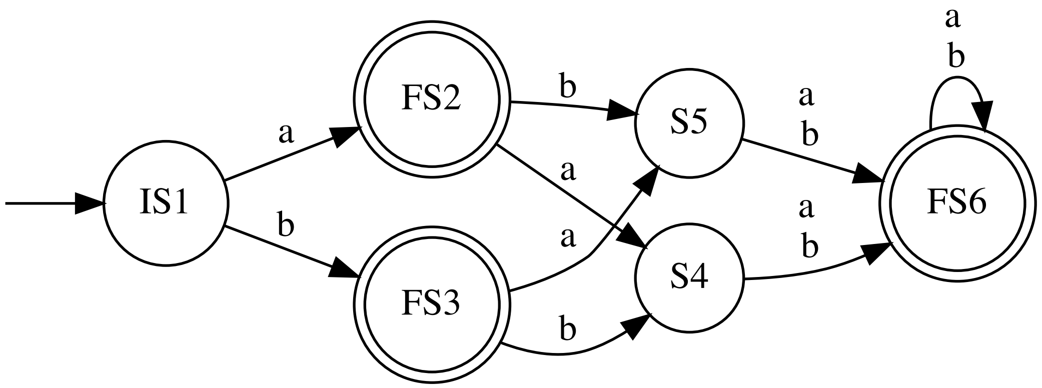
Print them, place one on top, “hold them to light”

- Express language equivalence of L1 and L2 in terms of two intersection-complement checks!
- Solution:
    - Start from: L1 = L2 iff L1 contained in L2 and vice-versa
        - Read L1 contained in L2 as “L1 fully inside L2”
        - Now read it as “L1 NOT OUTSIDE L2”
    - Break each containment into an intersection-complement check 

DFA minimization

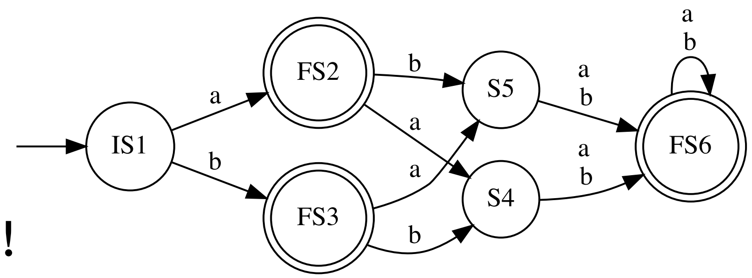
Let's experiment with two minimization problems.
First approach: "Eyeball algorithm"





DFA minimization: Fig 6.7 of book

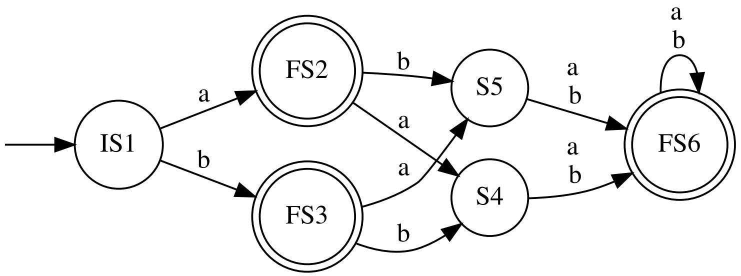
Full list of states : IS1, FS2, FS3, S4, S5, FS6
Row : IS1, FS2, FS3, S4, S5 (left to right)
Col : FS2, FS3, S4, S5, FS6 (top to bottom)
Purpose: to make ”N choose 2” state combos easily in a tabular display!
Then iterate in any order (recommend: left col, top to bottom, then second col,…)
6 choose 2 is 15



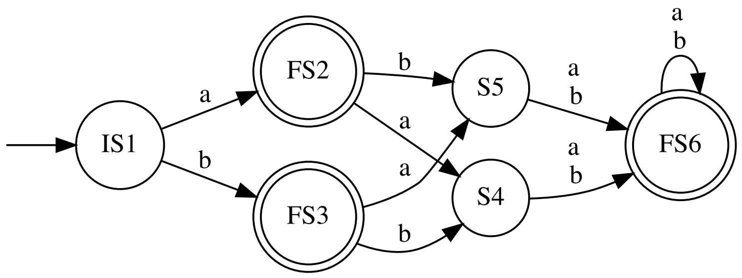
Two states S1 and S2 are k-distinguishable if

- (k=0) : S1 is a final and S2 is a non-final (or vice-versa)
- (k>1) : S1 and S2 go to states S1’ and S2’ on some a in Sigma and S1’ and S2’ are k-1 distinguishable

Final equivalence classes of size 2
{FS3, FS2}, {S5, S4}
These classes have no overlap
They don’t merge



DFA minimization: Fig 6.7 of book



Frame-0
(Initial)

	IS1	FS2	FS3	S4	S5
FS2	-1				
FS3	-1	-1			
S4	-1	-1	-1		
S5	-1	-1	-1	-1	
FS6	-1	-1	-1	-1	-1

Frame-1
(0-distinguishable)

	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	-1	-1	0	0

Frame-2
(1-distinguishable)

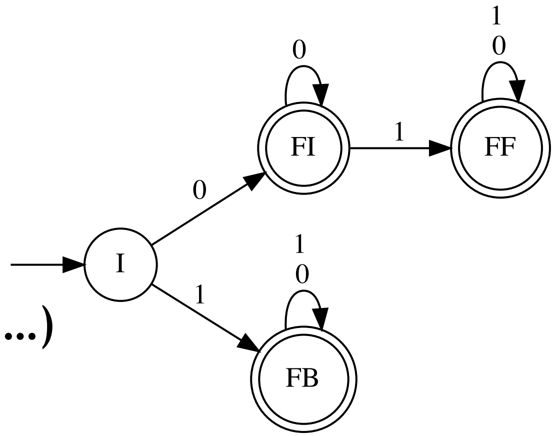
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

Frame-3 = Frame-4
(2-distinguishable)

	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	2	0	0		
S5	2	0	0	-1	
FS6	0	1	1	0	0

DFA minimization: Fig 6.7 of book

Full list of states : I, FI, FF, FB
Row : I, FI, FF (left to right)
Col : FI, FF, FB (top to bottom)
Purpose: to make ”N choose 2” state combos easily in a tabular display!
Then iterate in any order (recommend: left col, top to bottom, then second col,…)



DFA minimization: Fig 6.7 of book

Final equivalence classes of size 2 :
{FF, FI}, {FB, FI}, {FB, FF}
They have overlaps
Merge them into a super-equivalence class {FF, FI, FB}