CS 3100, Models of Computation, Spring 20, Lec 5

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

bit.ly/3100s20Syllabus

Where we are headed: the Pumping Lemma

Regular(L) => PumpingCondition(L)

PL effects be seen even in your cellphone!

E.g. the word-completion software of your phone is (essentially) a DFA

See my illustration online (class syllabus – video demo)

The fact that we can see such a nice theorem play out in real life is exciting !!

We want to express the PL as follows

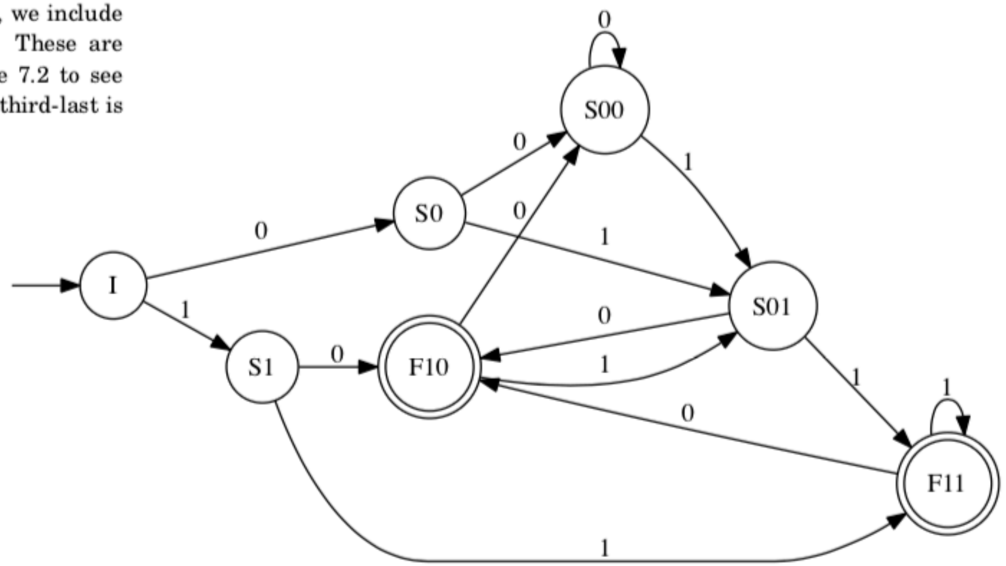
Regular(L) => PumpingCondition(L)

Then we can take the contrapositive

! Regular(L) <= !PumpingCondition(L)

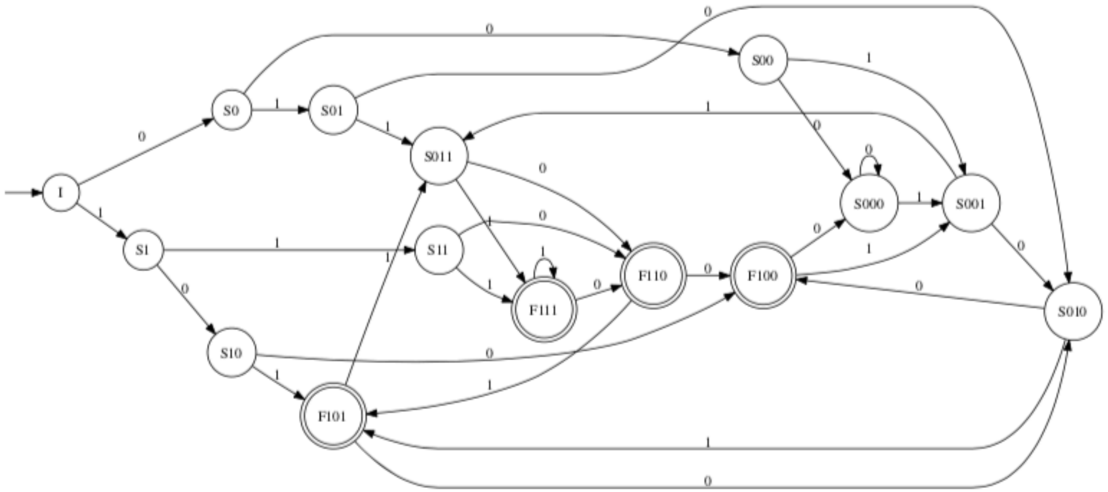
Fact: An N-state DFA has a FIRST loop in N steps

Figure 5.5: DFA drawing for 'second-last is 1' (above), and for comparison (to show the exponential growth), we include 'third-last is 1' also (below). These are *not* minimal DFA! See Figure 7.2 to see what the minimized DFA for 'third-last is 1' looks like.



- Means if "w is in L and w is of length at least N
  - w is a journey from the initial state to A final state
- Then one can find a loop form along w in N steps
- Then one call w = xyz
- where y is the loop

- And these alternative paths
  - that are also journeys from the initial state to the same final state



The previous slide said this, essentially

Regular(L) =>
Exists N :
For any string w = xyz in L of length >= N :
y is non-zero
AND xy of length <= N :
AND for all i >= 0 : x y^i z in L

For any string w in L, we can pump w in any direction, and you'll still find it in language L

Any direction means pump up or down… i.e. for any i … i.e.
i = 0 means pump down and i >= 2 means pump up
and still stay in the language

Pumping Lemma

! PumpingCondition(L) => ! Regular(L)

If we show that PumpingCondition is false of L, then L is not Regular

If we show that ! PumpingCondition is true of L, then ! (L is regular)

If we show that !PumpingCondition(L) is TRUE then !Regular(L)

What is !PumpingCondition(L) ?

Let’s make it true !!

Making !PumpingCondition True: Make this true

! Exists N :

For any string w = xyz in L of length >= N :

y is non-zero

AND xy of length <= N :

AND for all i >= 0 : x y^i z in L

Making !PumpingCondition True: Make this true

Forall N :

! For any string w = xyz in L of combined length >= N :

y is non-zero

AND xy of length <= N :

AND for all i >= 0 : x y^i z in L

Making !PumpingCondition True: Make this true

Forall N :

Exists a string w = xyz in L of combined length >= N :

! y is non-zero

AND xy of length <= N :

AND for all i >= 0 : x y^i z in L

Making !PumpingCondition True: Make this true

Forall N :

Exists a string w = xyz in L of combined length >= N :

! y is non-zero

AND xy of length <= N :

AND for all i >= 0 : x y^i z in L

Exists one string w in L that when you pump in a certain way, you can’t find it in language L … that is where we are going ; let’s do it slowly

Making !PumpingCondition True: Make this true

Forall N :

Exists a string w = xyz in L of combined length >= N :

! y is non-zero

AND xy of length <= N :

AND for all i >= 0 : x y^i z in L

Making !PumpingCondition True: Make this true

Forall N :

Exists a string w = xyz in L of combined length >= N :

y is non-zero

AND xy of length <= N

➔

exists i >= 0 : x y^i z is **not in L**

this is often called “pump out”

Proving !Regular(L)

- Pick a string w of length >= N
  - We must pick w with great care
  - **<u>It must be a parametric string</u>**
  - For our proof to work, it must harbor a loop within its first N steps

- Find an arbitrary split of w into xyz
- y is non-zero
- xy is confined to N
- and xy^i z not in L

The beauty of mathematical logic:

forms of arguments carry through – the contents notwithstanding

How you must present your proofs

1. For the language L .. E.g. {0^j 1^j : j >= 0 }, present parametric string w
   - E.g. let there be an N-state DFA
   - My parametric string w is 0^N 1^N
2. Describe x, y, and z generically
   - you have no control over x, y, z
   - Except that
     - y is non-empty
     - Say
       - "in my parametric string, y is all 0's
       - " x could be empty "
       - " z could be some left-over 0's and all the remaining 1's "
3. State which direction of pump you are choosing
   - " I chose i = 0 because " … explain your reason
   - " I chose i >= 2 because " … explain your reason
4. Argue that the pumped string is not in L
5. Hence ! PumpingCondition(L)
6. Hence ! Regular(L) -- QED

Prove these languages not regular

- { 0^i 1^i : i >= 0 }
- { (^i)^i : i >= 0 }
- { w w : w in Sigma* }
- { w : w is a Palindrome over Sigma }
- { 0^i 1^j : i != j }

Goals: Learn DFA design and DFA operations

- Regular languages are closed under
  - Union
  - Intersection
  - Complementation
- E.g. Design a DFA that accepts
  - Strings over {0,1}
  - The strings when viewed as "Big Endian" (MSB-first) must be multiples of 3
    AND
  - Must contain a 100
- Method:
  - Design a DFA for "multiples of 3"
  - Design a DFA for "Contain 100"
  - Intersect them
  - Minimize them
- We will now study intersection and minimization

Interactive design of DFA for “ w % 3 == 0”

DFA for “value(w) % 3 == 0”

DFA for “contains 100”

DFA intersection algorithm

- Given D1 = (Q1, Sigma, d1, q01, F1)
- and D2 = (Q2, Sigma, d2, q02, F2)
- The idea is to design a new DFA
- D =(Q, Sigma, d, q0, F) such that
  - D1 and D2 start at their respective start states q01 and q02
  - When a symbol a in Sigma comes in, both D1 and D2 must advance
  - Any string w accepted by D1 and D2 must be accepted by D

DFA intersection algorithm

- Given (Q1, Sigma, d1, q01, F1) and (Q2, Sigma, d2, q02, F2)
- Q =
- q0 =
- F =

DFA intersection algorithm

- Given (Q1, Sigma, d1, q01, F1) and (Q2, Sigma, d2, q02, F2)
- Q = Q1 x Q2
- Q0 = (q01, q02)
- F = F1 x F2

Jove sessions

Design a DFA for "multiples of 3"

In [2]:

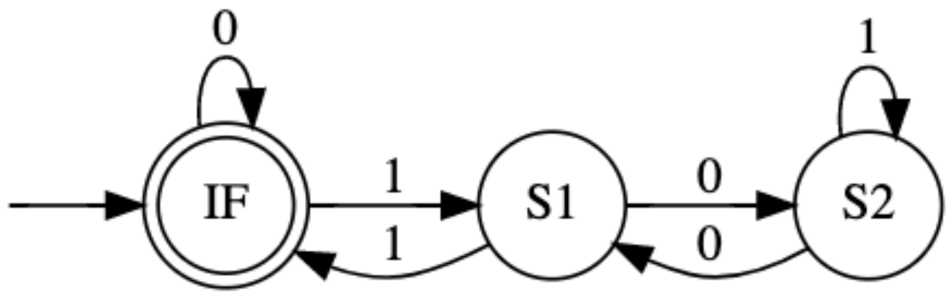
```
DFA3 = md2mc('''DFA

IF : 0 -> IF !! Initial and final, as "0" is divisible by 3
IF : 1 -> S1

S1 : 0 -> S2 !! A state with remainder 1, upon 0 shifts, becoming S2
S1 : 1 -> IF !! A state with remainder 1, upon 1, re-obtains value 3
             !! which is divisible by 3, hence we go to IF
S2 : 0 -> S1 !! A state with remainder 2, when fed 0 becomes 4
             !! which modulo 3 is 1
S2 : 1 -> S2 !! A state with value 2 will become S4, but adding 1
             !! gives S5, and mod of 3 gives S2

''')
```

```
Generating LALR tables
```

In [3]:

```
dotObj_dfa(DFA3)
```

Out[3]:



Jove sessions

DFA for "contains 100"

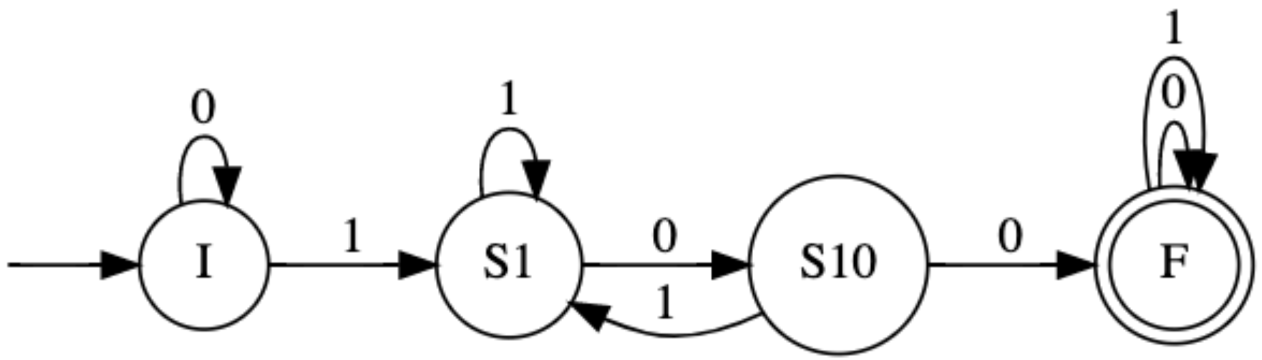
In [4]:

```
DFA100 = md2mc(''' DFA

I : 0 -> I
I : 1 -> S1

S1   : 0 -> S10
S1   : 1 -> S1

S10 : 0 -> F
S10 : 1 -> S1


F    : 0|1 -> F

''')
```

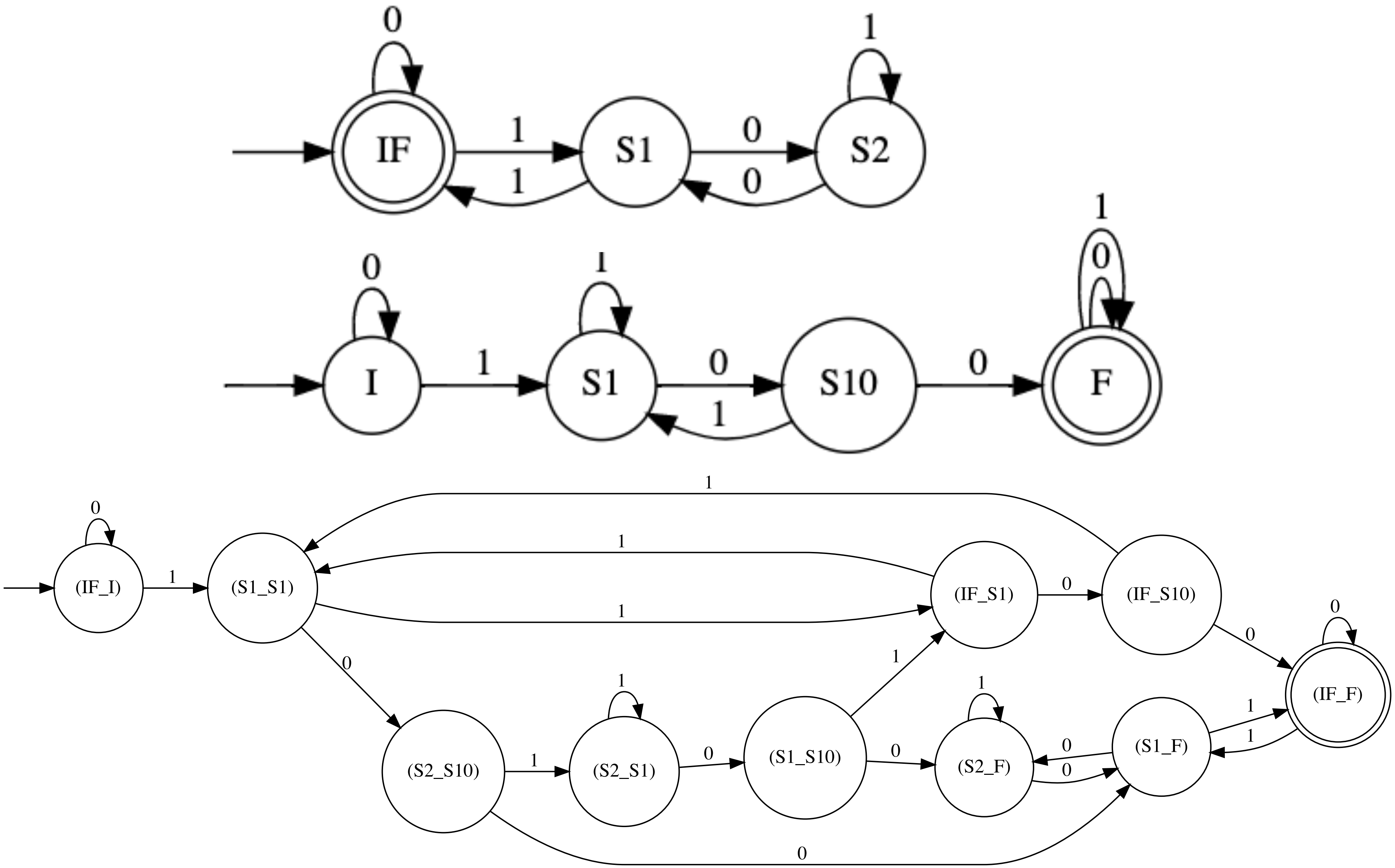
In [5]:

```
dotObj_dfa(DFA100)
```

Out[5]:



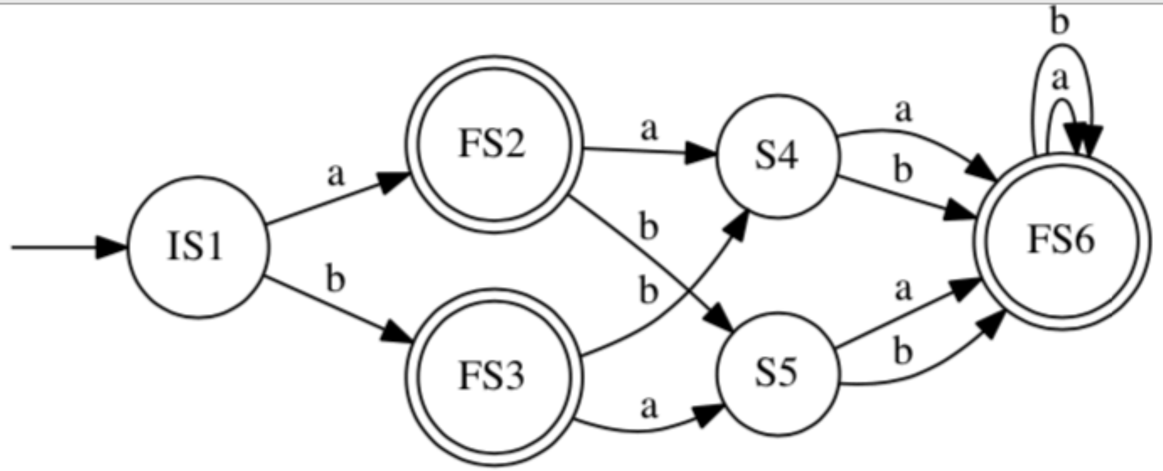
Intersection of the DFA at the top results in the DFA at the bottom

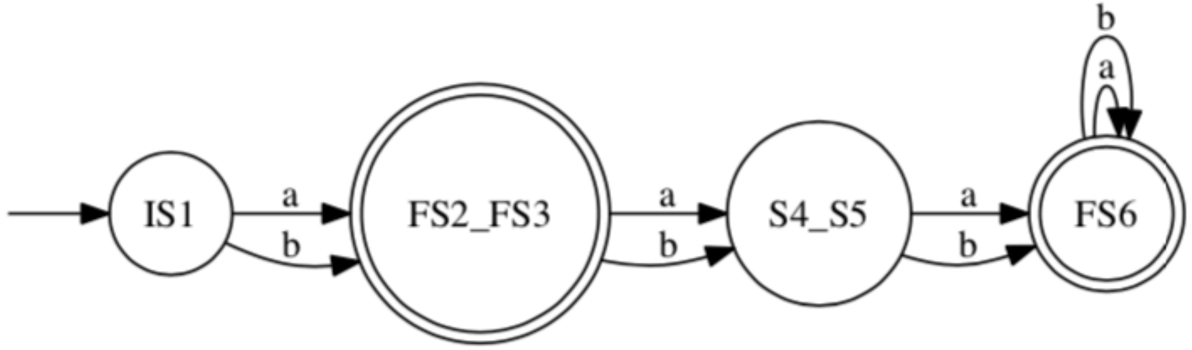


DFA minimization

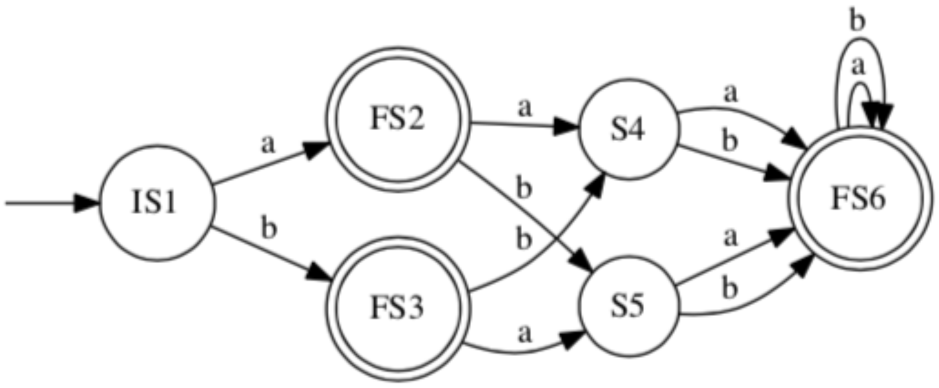
- Build a dynamic-programming table
  - Represent all combinations of two states
- Initially, distinguish combinations in which one state is accepting and the other is not
- For every pair of states not distinguished so far
  - If we march the states through a symbol such that
  - The next states have been distinguished
    - Then distinguish the starting states
- Do this systematically across all table entries

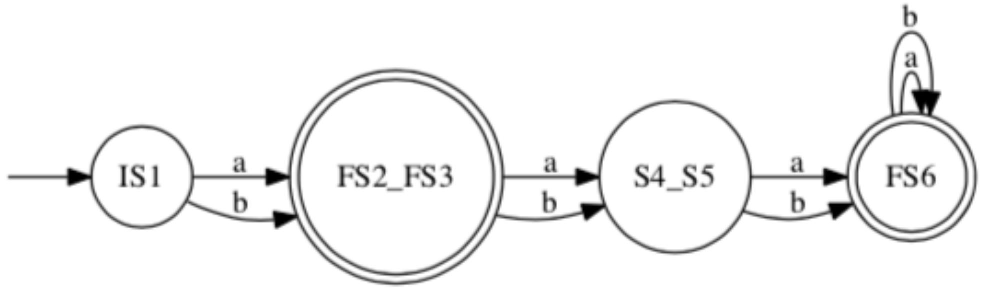
DFA minimization





DFA minimization





Frame-0
(Initial)

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

Frame-1
(0-distinguishable)

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

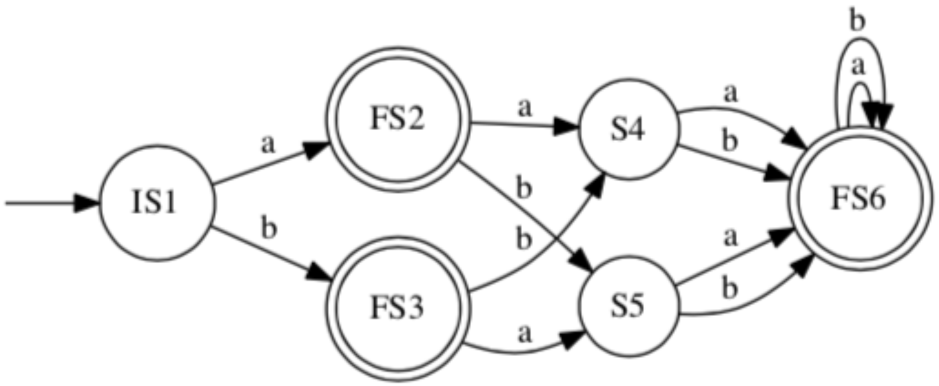
Frame-2
(1-distinguishable)

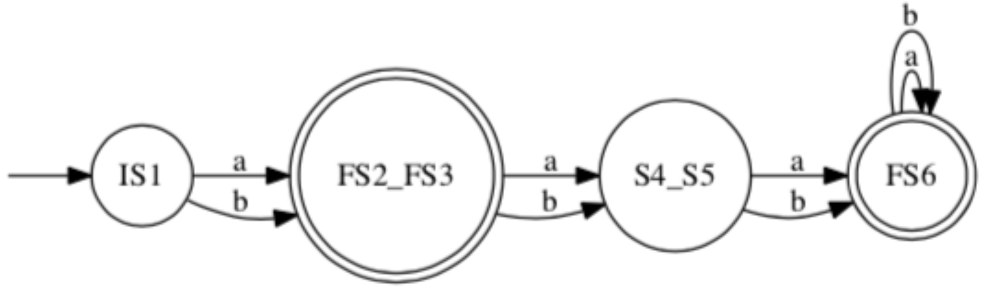
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

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

DFA minimization





Frame-0
(Initial)

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

Frame-1
(0-distinguishable)

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

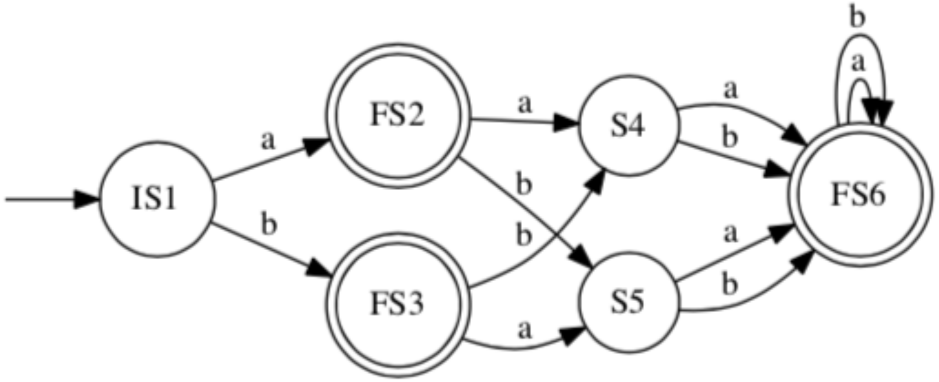
Frame-2
(1-distinguishable)

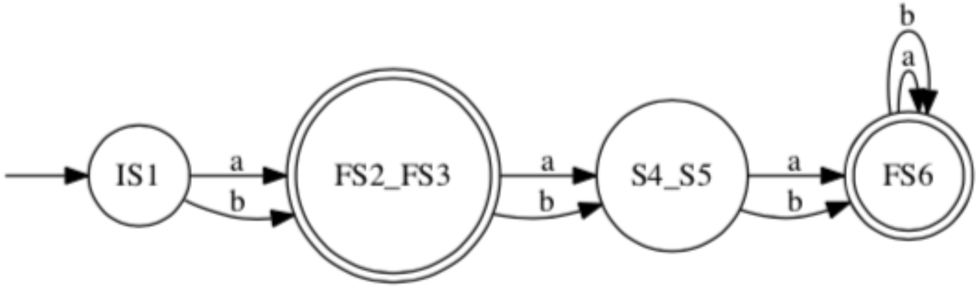
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

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

DFA minimization





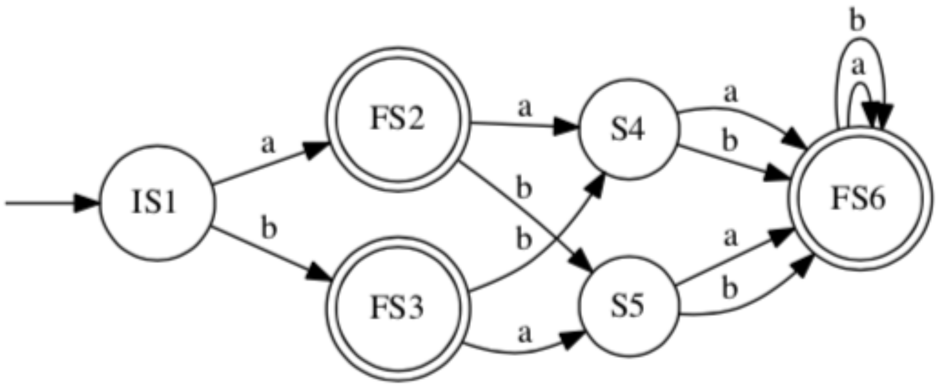

```
Frame-0
(Initial)
-------------------
FS2  -1
FS3  -1  -1
S4   -1  -1  -1
S5   -1  -1  -1  -1
FS6  -1  -1  -1  -1  -1
     IS1 FS2 FS3 S4  S5
```

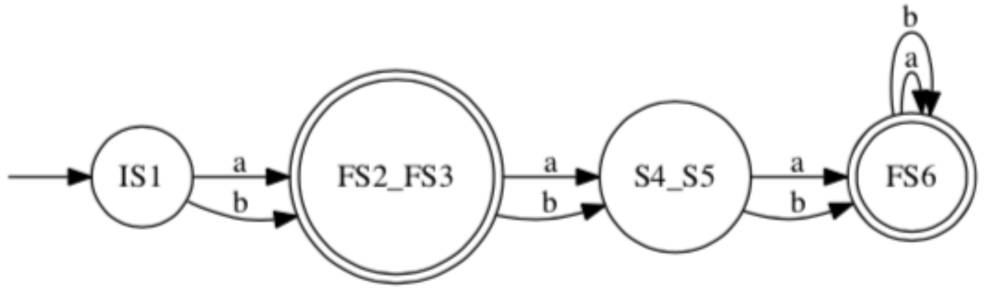
```
Frame-1
(0-distinguishable)
-------------------
FS2   0
FS3   0  -1
S4   -1   0   0
S5   -1   0   0  -1
FS6   0  -1  -1   0   0
     IS1 FS2 FS3 S4  S5
```

```
Frame-2
(1-distinguishable)
-------------------
FS2   0
FS3   0  -1
S4   -1   0   0
S5   -1   0   0  -1
FS6   0   1   1   0   0
     IS1 FS2 FS3 S4  S5
```

```
Frame-3 = Frame-4
(2-distinguishable)
-------------------
FS2   0
FS3   0  -1
S4    2   0   0
S5    2   0   0  -1
FS6   0   1   1   0   0
     IS1 FS2 FS3 S4  S5
```

DFA minimization





Frame-0
(Initial)

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

Frame-1
(0-distinguishable)

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

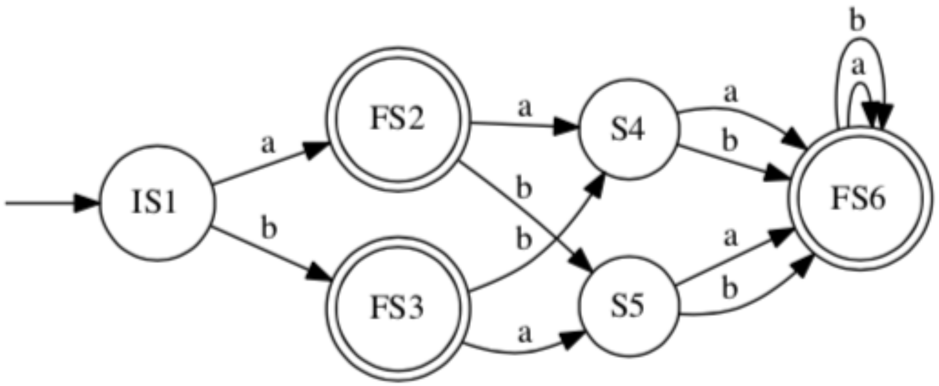
Frame-2
(1-distinguishable)

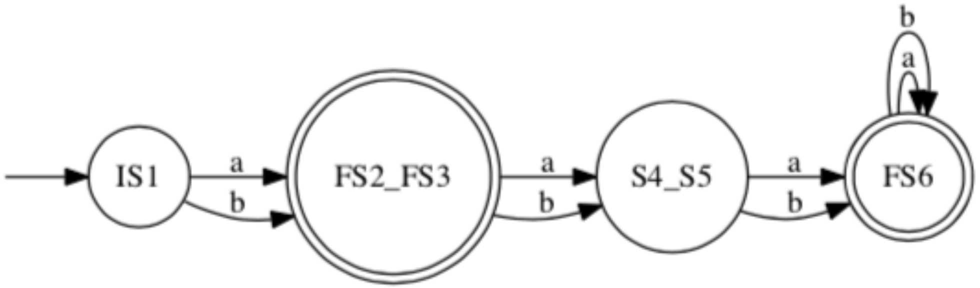
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

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

DFA minimization





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

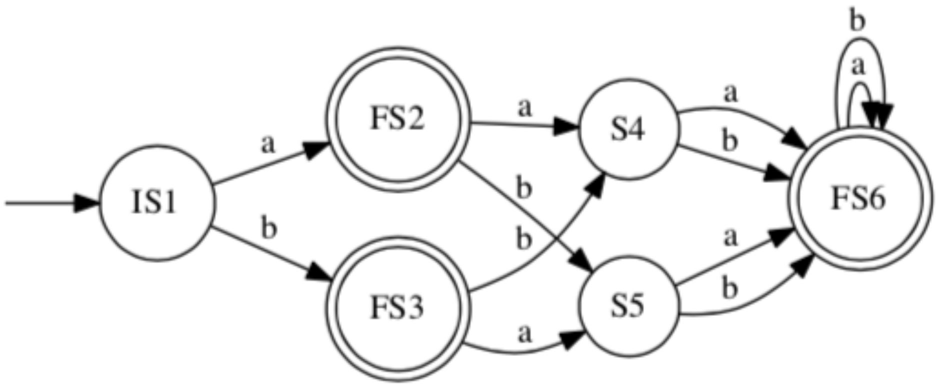
Frame-2
(1-distinguishable)

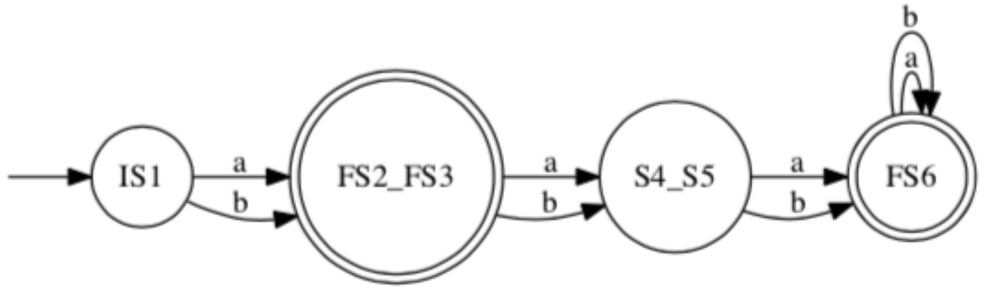
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

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





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

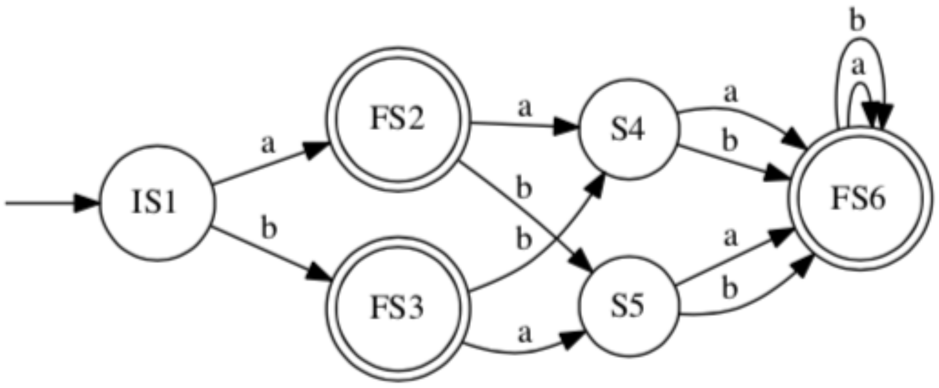
Frame-2
(1-distinguishable)

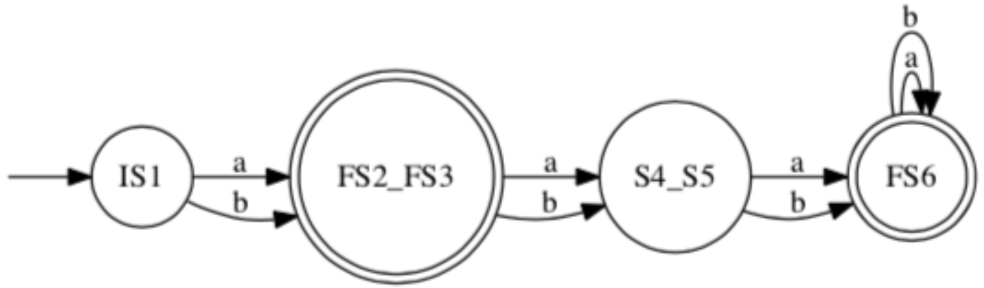
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

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





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

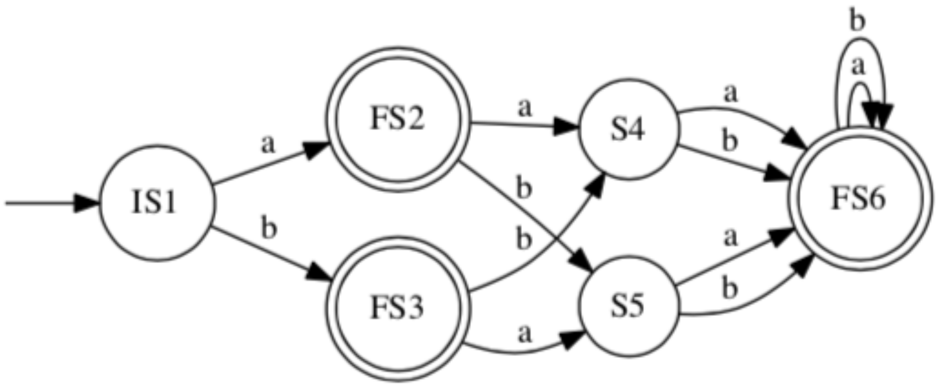
Frame-2
(1-distinguishable)

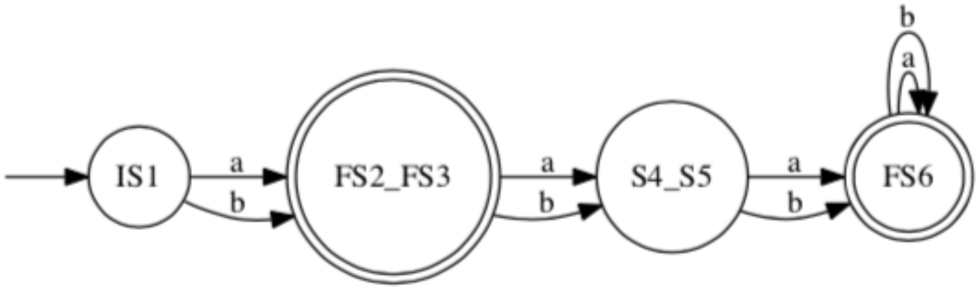
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

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





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

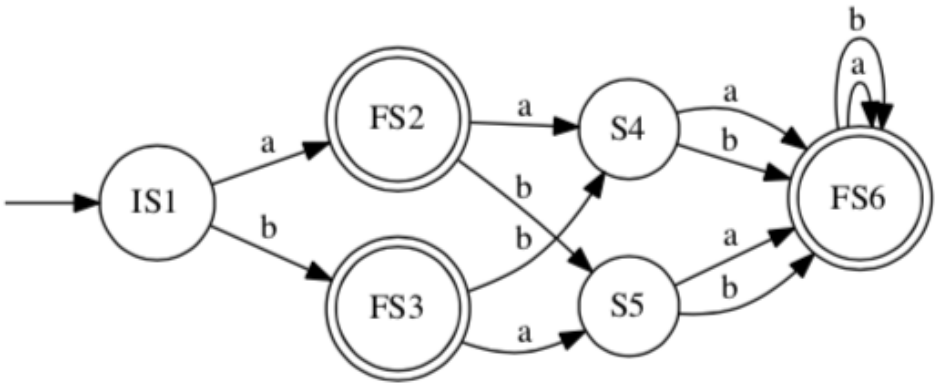
Frame-2
(1-distinguishable)

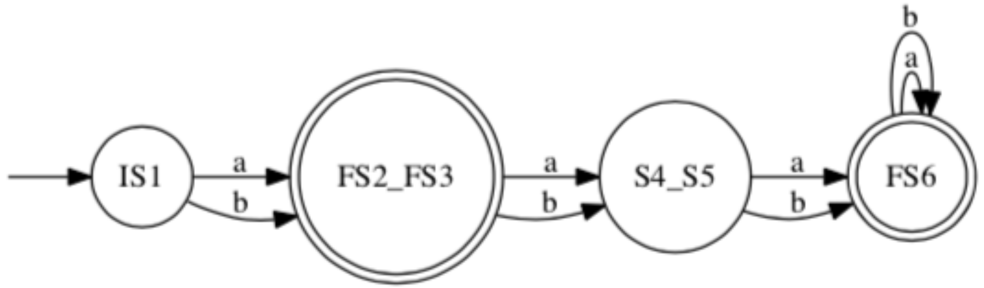
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

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





Frame-0
(Initial)

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

Frame-1
(0-distinguishable)

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

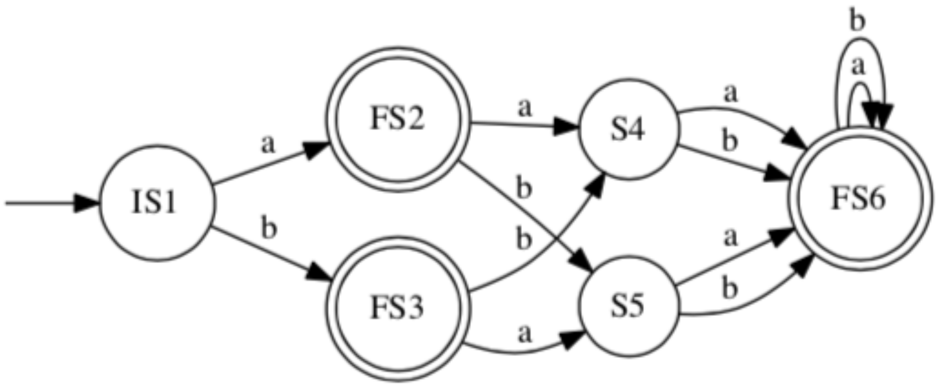
Frame-2
(1-distinguishable)

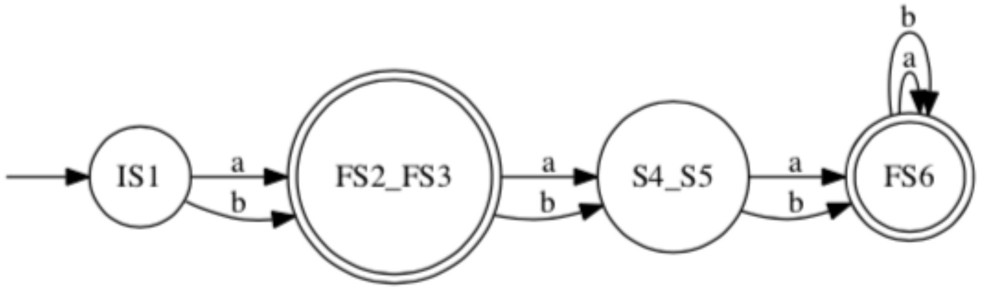
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

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

DFA minimization





Frame-0
(Initial)

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

Frame-1
(0-distinguishable)

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

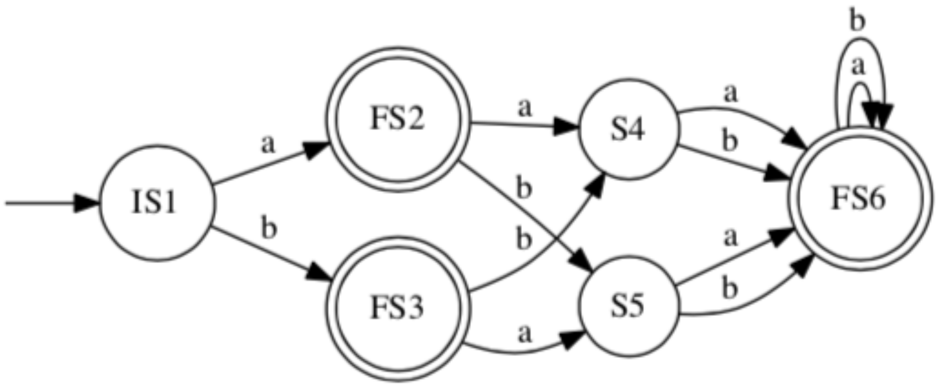
Frame-2
(1-distinguishable)

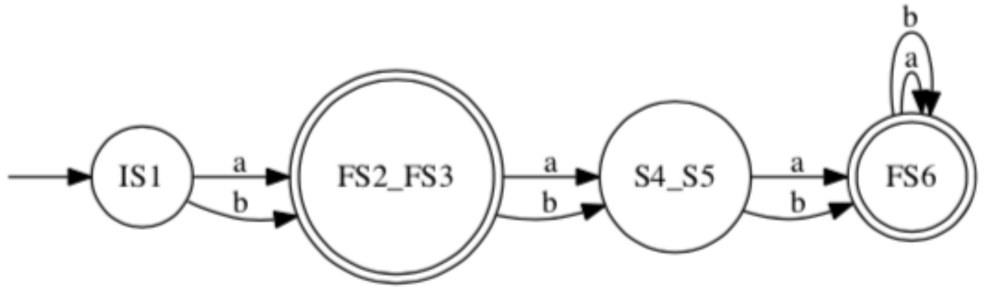
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

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

DFA minimization





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

Another DFA design through Boolean ops

- Doesn’t begin with 010
- AND
- Doesn’t end with 101

- Can use Demorgan’s laws
  - Design for Begins with 010
  - Design for End with 101
  - OR them
  - Complement them
- Compare with a direct design of the given problem!
  - This will be worked out in class interactively, by hand and by Jove