CS 3100, Models of Computation, Spring 20, Lec 6

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

bit.ly/3100s20Syllabus

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]:

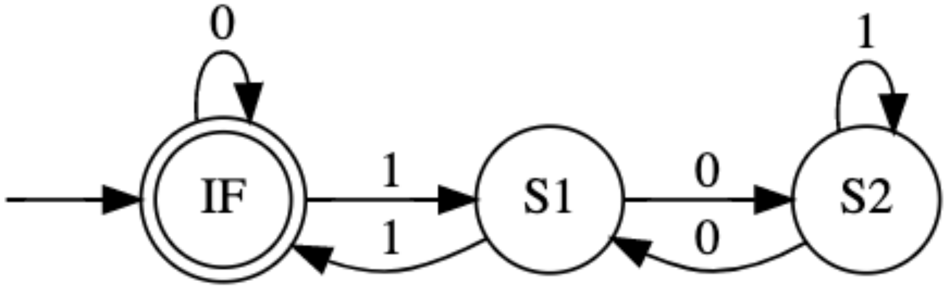
```
DFA3 = md2mc('''DFA

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"

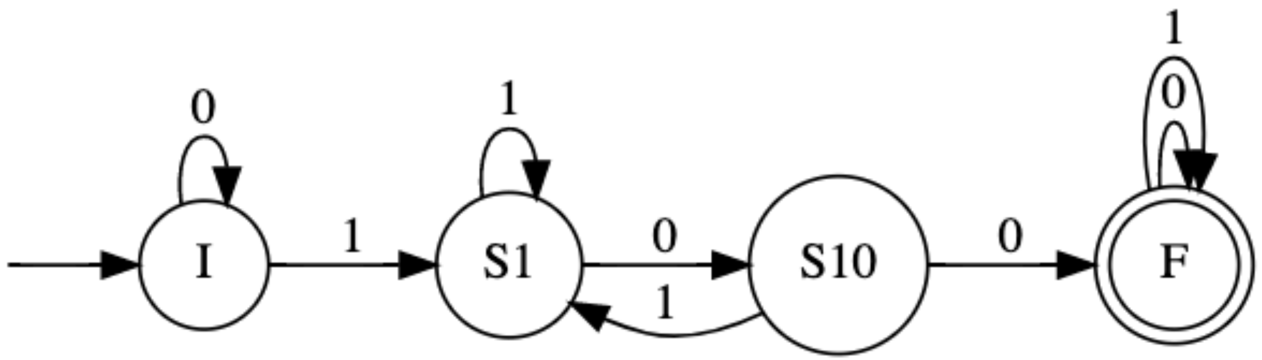
In [4]:

```
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

''')
```

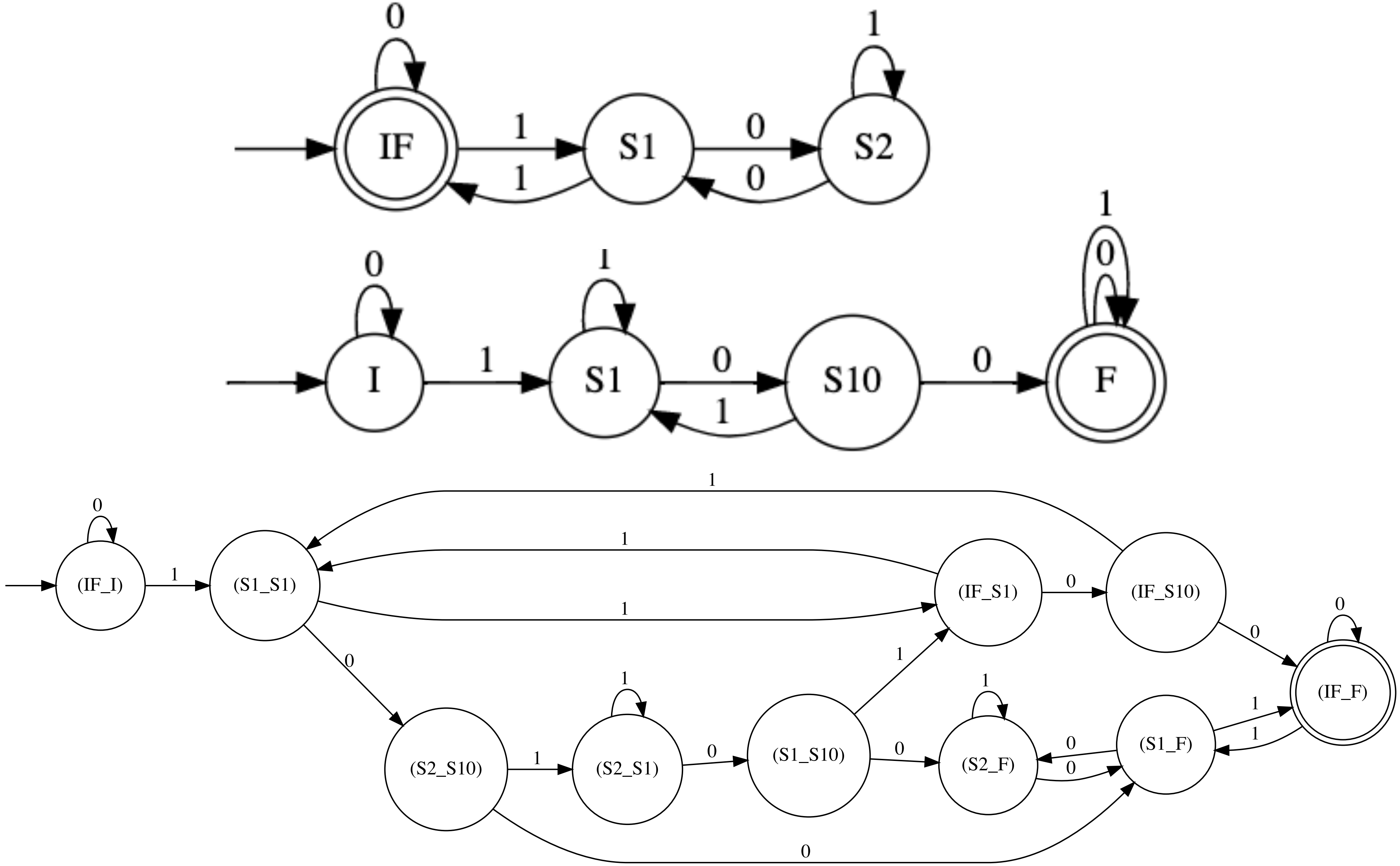
In [5]:

```
dotObj_dfa(DFA100)
```

Out[5]:



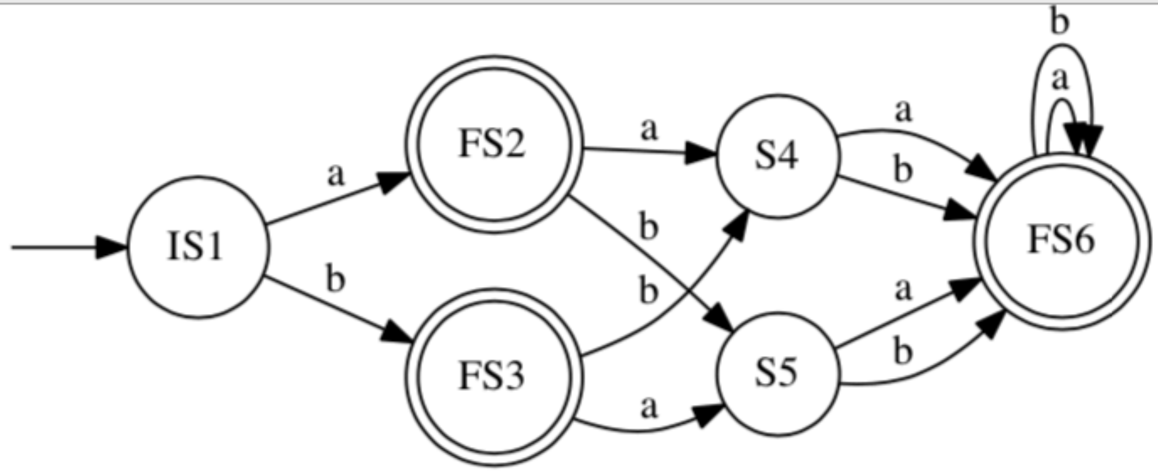
Intersection of the DFA at the top results in the DFA at the bottom

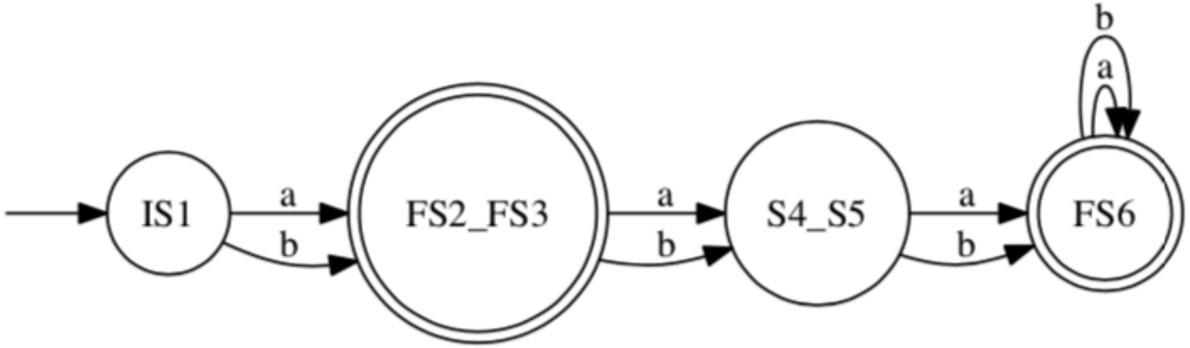


DFA minimization

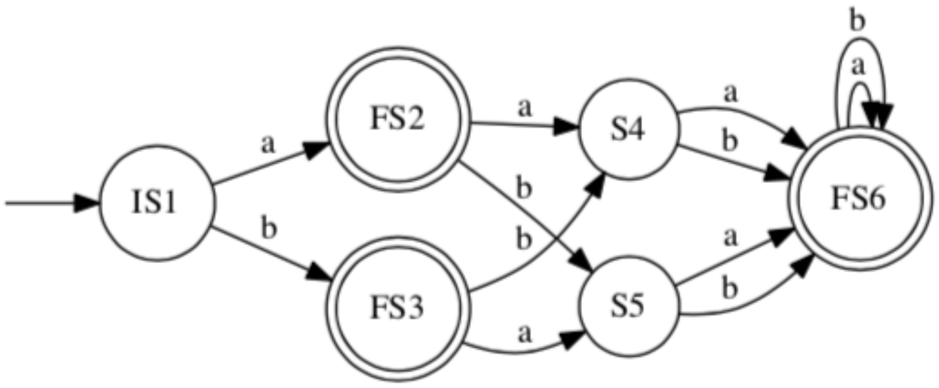
- Build a dynamic-programming table
  - Represent all combinations of two states
- Initially, distinguish combinations in which one state is accepting and the other is not
- For every pair of states not distinguished so far
  - If we march the states through a symbol such that
  - The next states have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

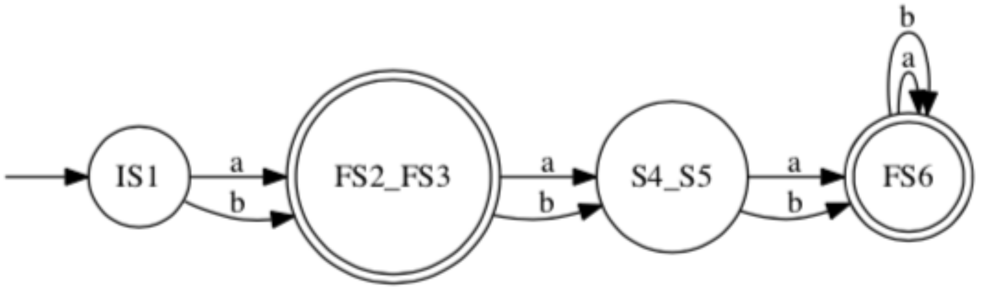
DFA minimization





DFA minimization





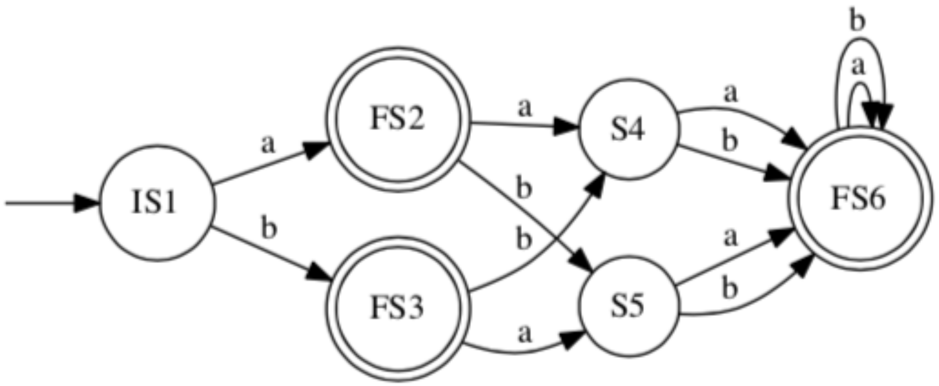

```
Frame-0
(Initial)
-------------------
FS2  -1

FS3  -1  -1

S4   -1  -1  -1

S5   -1  -1  -1   -1

FS6  -1  -1  -1   -1  -1

     IS1 FS2 FS3  S4  S5
```

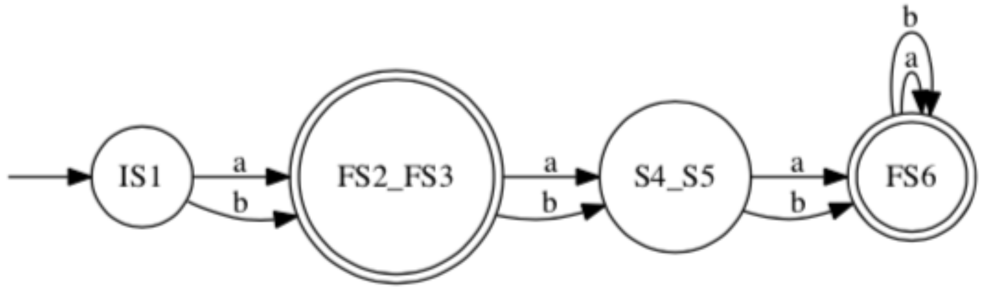
```
Frame-1
(0-distinguishable)
-------------------
FS2   0

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





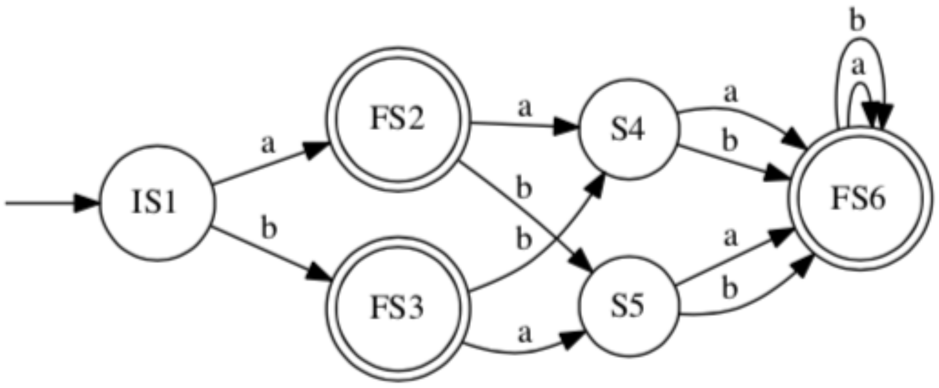

```
Frame-0
(Initial)
-------------------
FS2  -1
FS3  -1  -1
S4   -1  -1  -1
S5   -1  -1  -1  -1
FS6  -1  -1  -1  -1  -1
     IS1 FS2 FS3 S4  S5
```

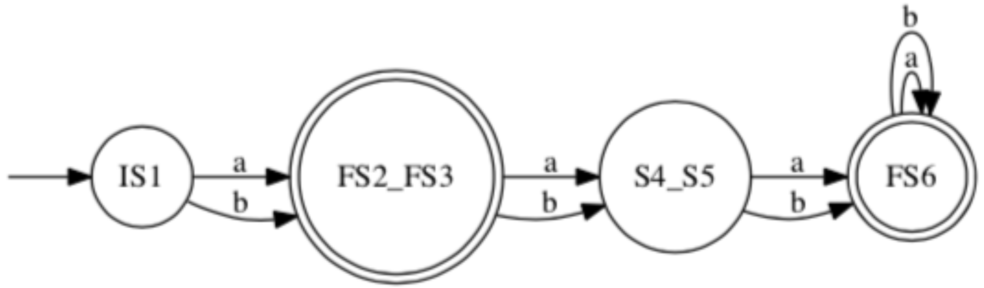
```
Frame-1
(0-distinguishable)
-------------------
FS2   0
FS3   0  -1
S4   -1   0   0
S5   -1   0   0  -1
FS6   0  -1  -1   0   0
     IS1 FS2 FS3 S4  S5
```

```
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)

	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

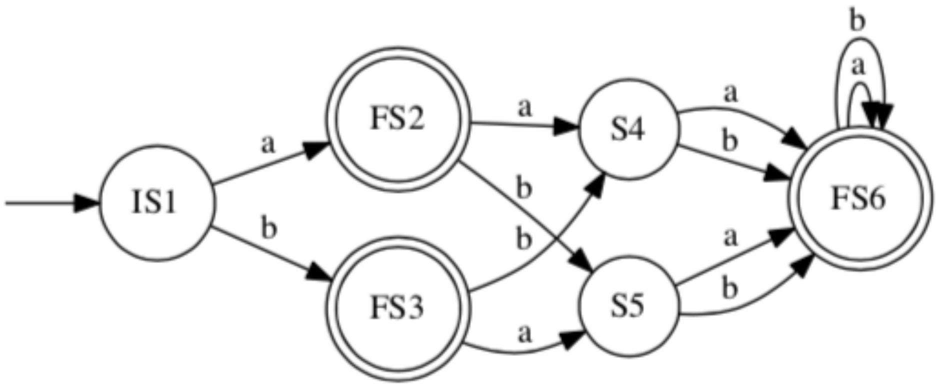
Frame-2
(1-distinguishable)

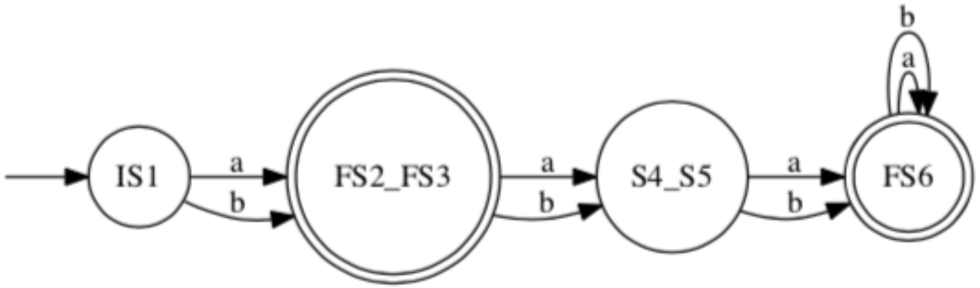
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

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





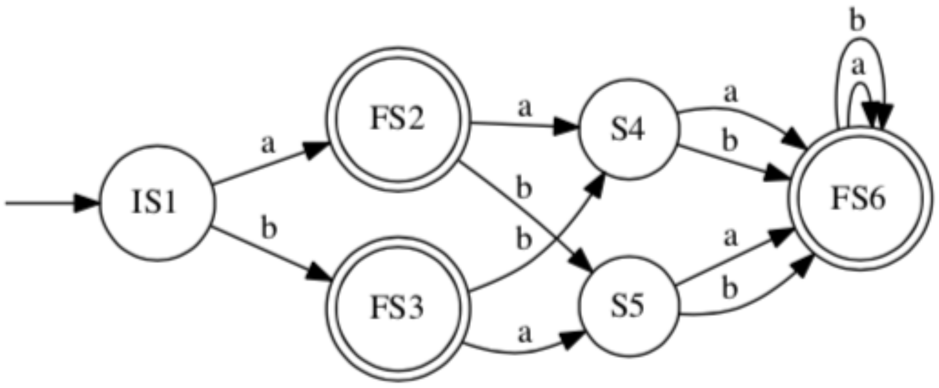

```
Frame-0
(Initial)
-------------------
FS2  -1
FS3  -1  -1
S4   -1  -1  -1
S5   -1  -1  -1  -1
FS6  -1  -1  -1  -1  -1
     IS1 FS2 FS3 S4  S5
```

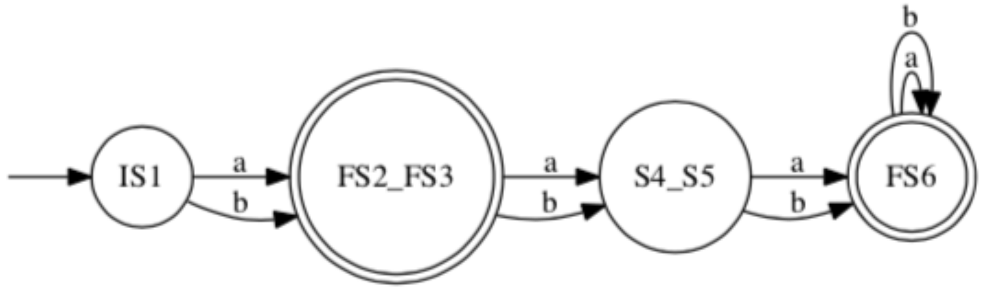
```
Frame-1
(0-distinguishable)
-------------------
FS2   0
FS3   0  -1
S4   -1   0   0
S5   -1   0   0  -1
FS6   0  -1  -1   0   0
     IS1 FS2 FS3 S4  S5
```

```
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

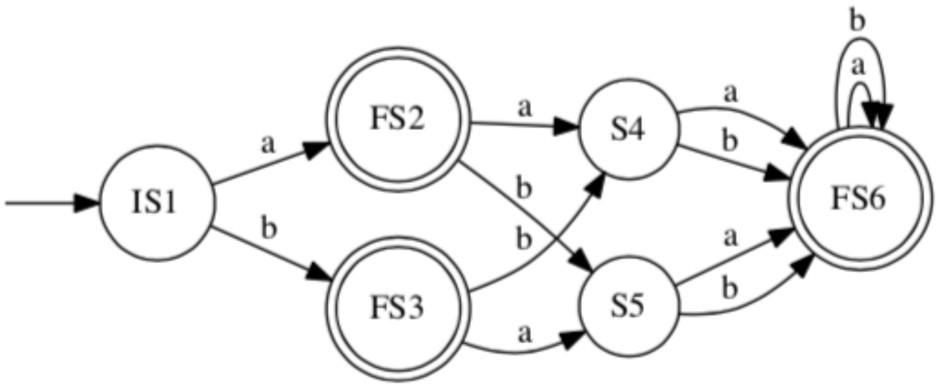
Frame-2
(1-distinguishable)

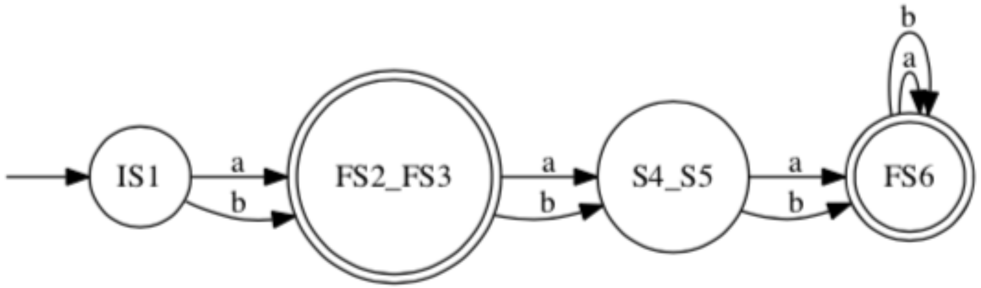
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

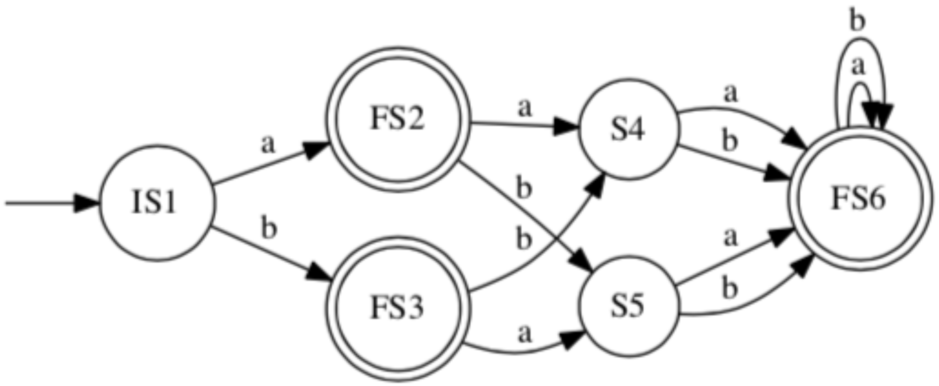
Frame-2
(1-distinguishable)

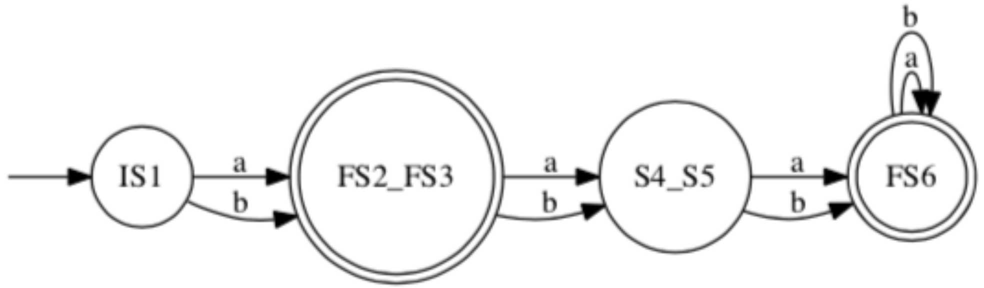
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

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

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

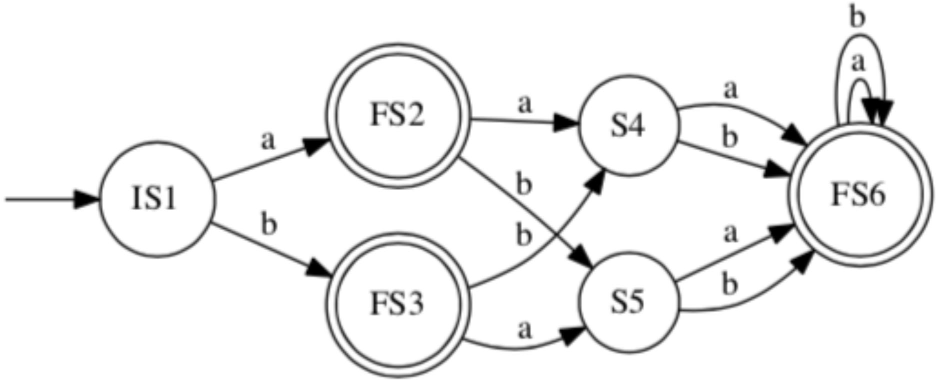
Frame-2
(1-distinguishable)

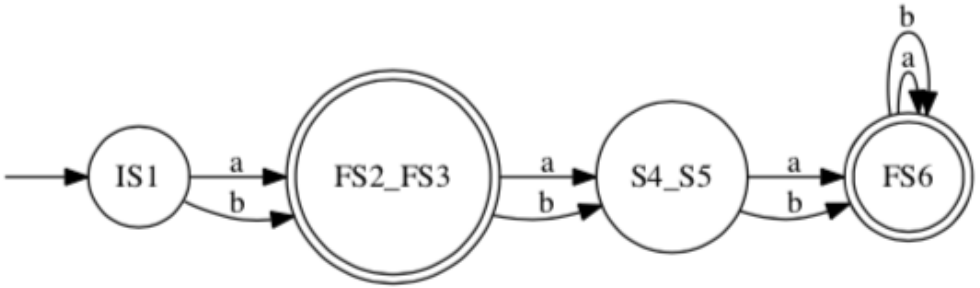
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

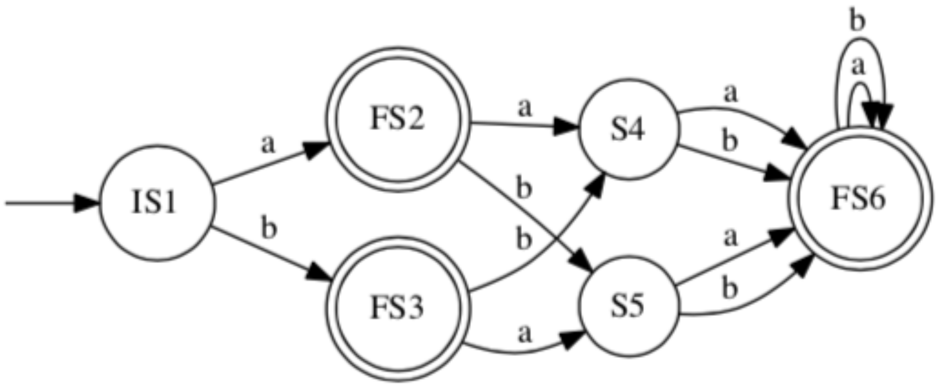
Frame-2
(1-distinguishable)

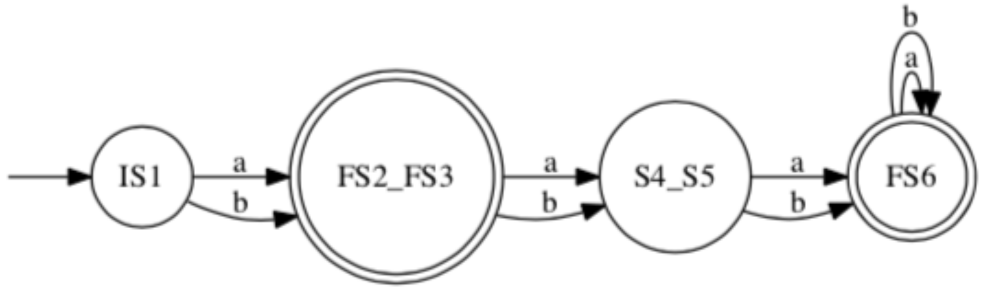
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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

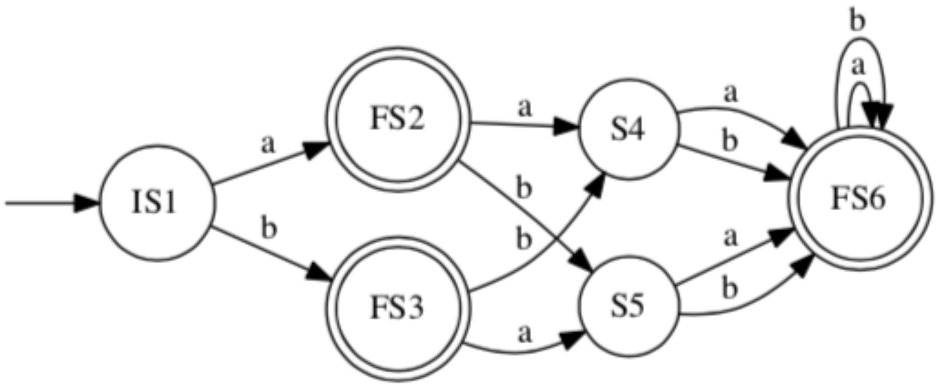
Frame-2
(1-distinguishable)

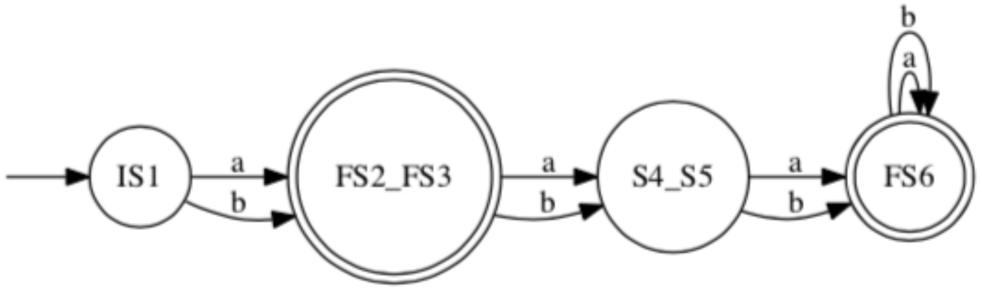
	IS1	FS2	FS3	S4	S5
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0

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)

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

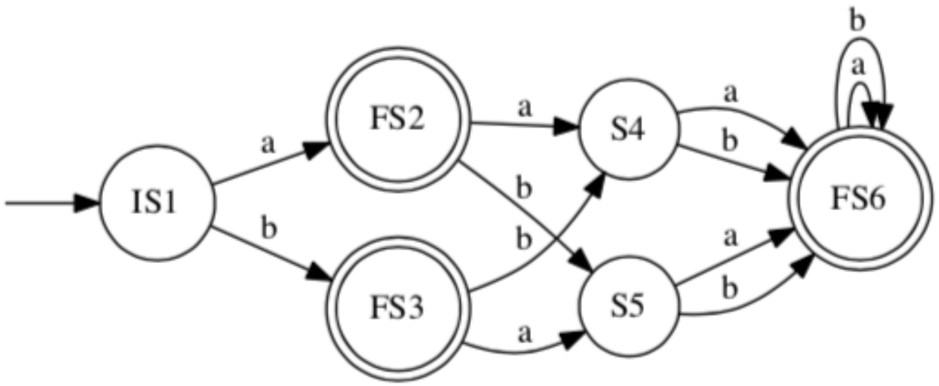
Frame-2
(1-distinguishable)

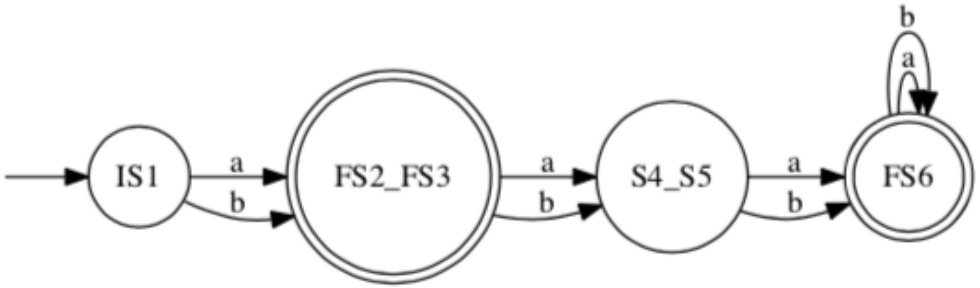
FS2	0				
FS3	0	-1			
S4	-1	0	0		
S5	-1	0	0	-1	
FS6	0	1	1	0	0
	IS1	FS2	FS3	S4	S5

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
```

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

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

Why NFA? Many answers!

1. Invented to overcome the limitations of a DFA
   - For some regular languages (that have a DFA), the DFA are **exponentially big**
   - **In many of those cases, an NFA will be linear / polynomial in size**
   - **Therefore reduces tedium (for humans) to specify**
   - **This use of NFA also turns into a syntactic approach called Regular Expressions**

2. **Nondeterminism is a fundamental idea in CS**
   - **Allows us to classify algorithms into "easy" (P) and "hard" (NP)**

Features of an NFA

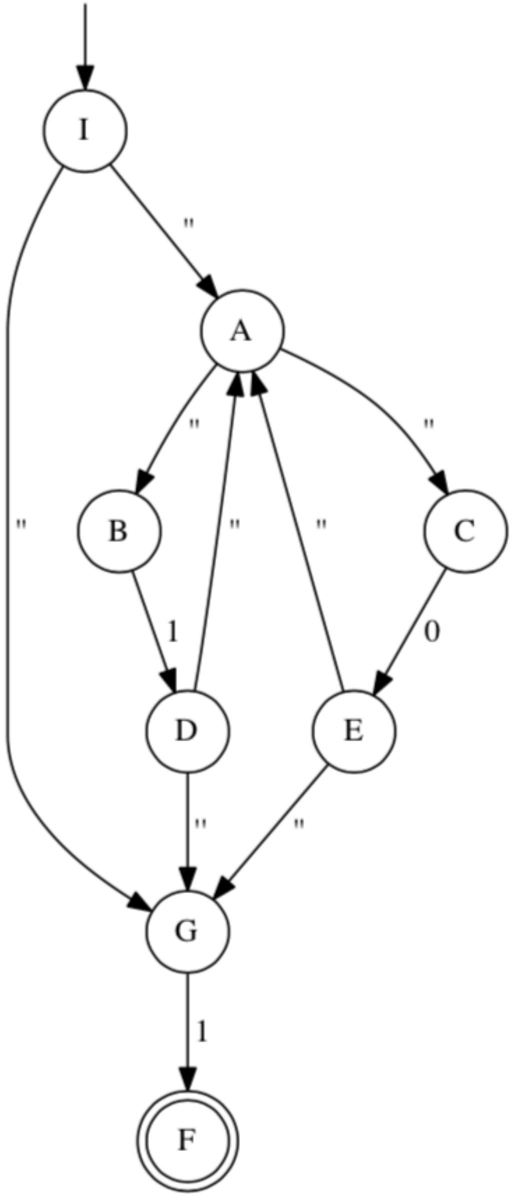
- Finite states
- Multiple initial states
  - Can begin in any initial state
- Transitions on Sigma
- Transitions also on Epsilon
  - Recall that Epsilon is not in Sigma
- Transitions lead to sets of next states
- Has final states (like before)
- Acceptance:
  - Begin at any initial state
  - A journey described by a string (laden perhaps with Epsilon)
  - Ends in a final state

Two NFA designs for “third last is a 1”

NFA have equivalent DFA - Subset Construction (function nfa2dfa)

- Bascially for each NFA that is in a set of states {S1,S2,S3}
  - For example we assume a set of 3 states an NFA is in.
- First E-close {S1,S2,S3}
  - Let S1 go to S11, S12 on ‘’
  - Let S2 go to S21 on ‘’
  - Let S3 go on S31, S32, S33 on ‘’
  - E-closure ({S1,S2,S3}) = {S11,S12,S21,S31,S32,S33}
- Fire a symbol, say a in Sigma from each of S11, S12,…S33
- Let the resulting SET OF STATES be
  - S11’, S12’, …. , S33’   (these are SETS of states)
- Take a set union of S11’, S12’, …, S33’
  - E-close that state.
- This is what {S1,S2,S3} transitions to, upon an “a” in Sigma
- Do this for Book77 NFA
- Do this for “third last is a 1” NFA

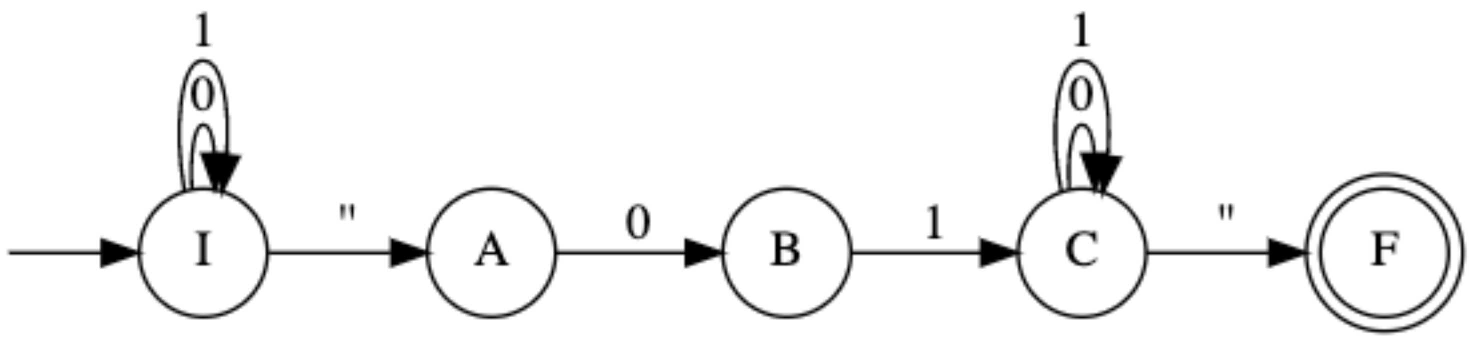
Book’s Figure 7.7 NFA : Write in Jove’s markdown in 2 ways, then convert to DFA via Subset Construction



Concepts around NFA, DFA, RE, and Applications

- NFA allow regular languages to be specified succinctly

E.g. NFA for “strings that contain 01” (one of many designs)



Concepts around NFA, DFA, RE, and Applications

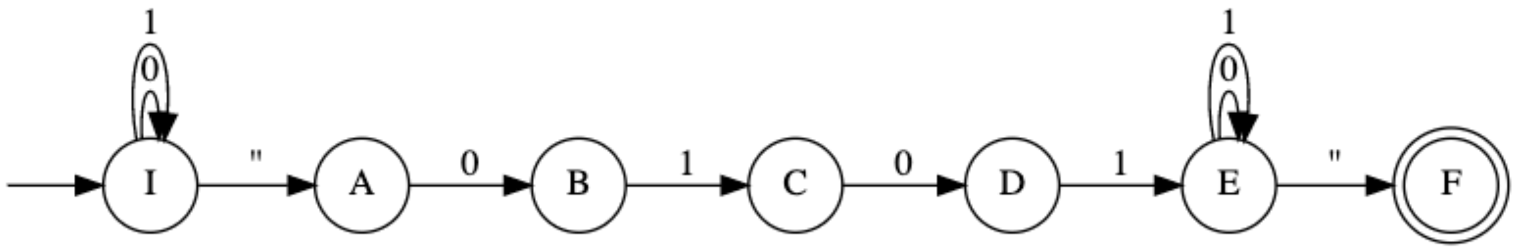
- NFA allow regular languages to be specified succinctly

E.g. NFA for “strings that contain 0101”

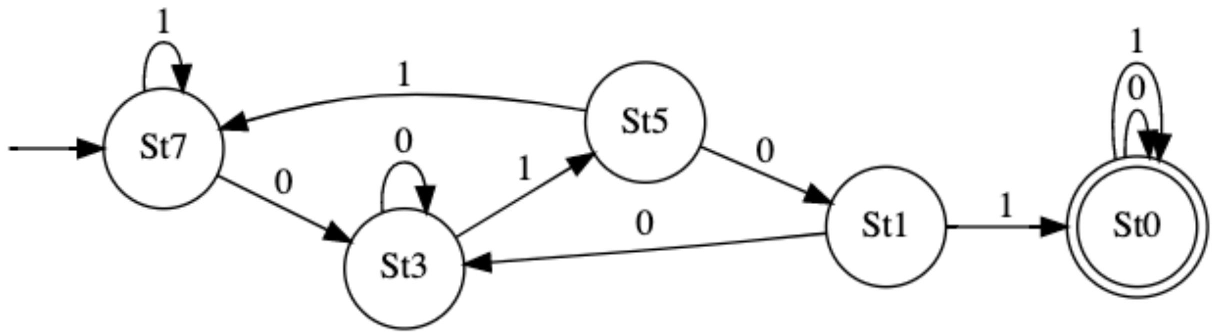
One NFA for “contains 0101”

```
nfahas0101 = md2mc('''
NFA
I : 0 | 1 -> I
I : '' -> A
A : 0  -> B
B : 1  -> C
C : 0  -> D
D : 1  -> E
E : 0 | 1 -> E
E : '' -> F
''')
```

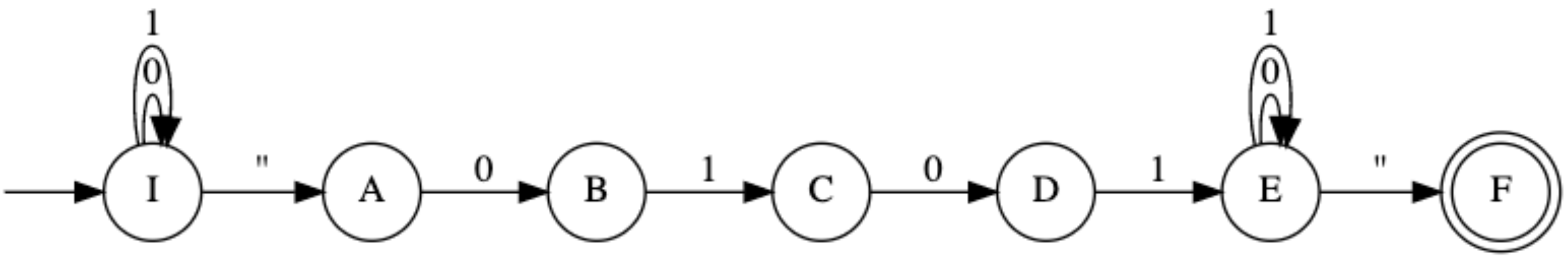
```
dotObj_nfa(nfahas0101)
```



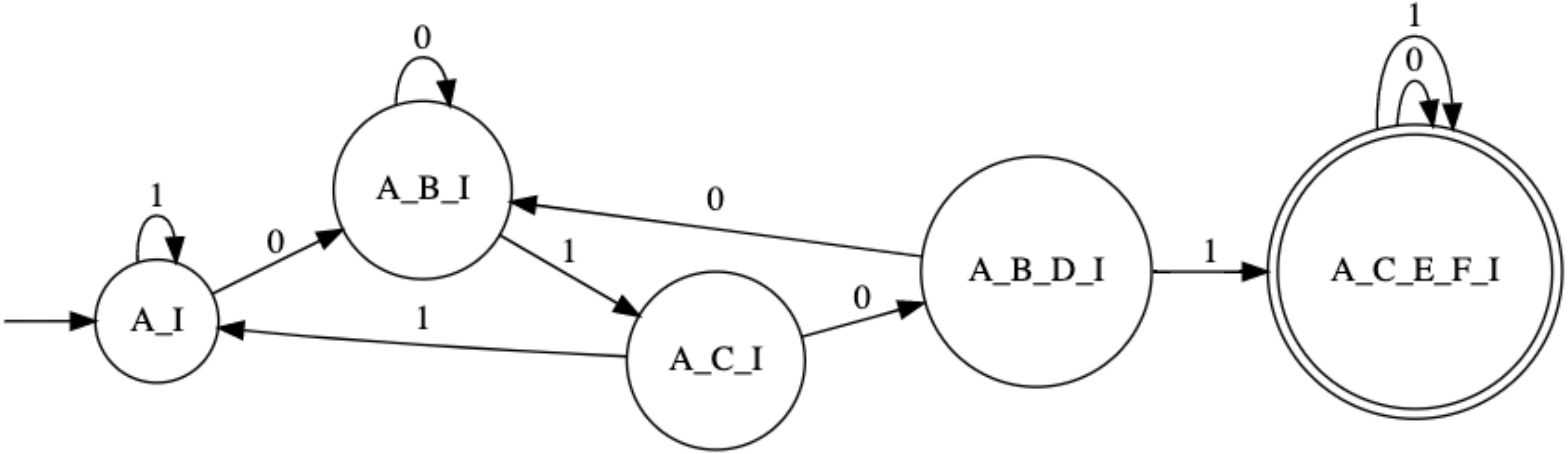

```
dotObj_dfa(min_dfa(nfa2dfa(nfahas0101)))
```



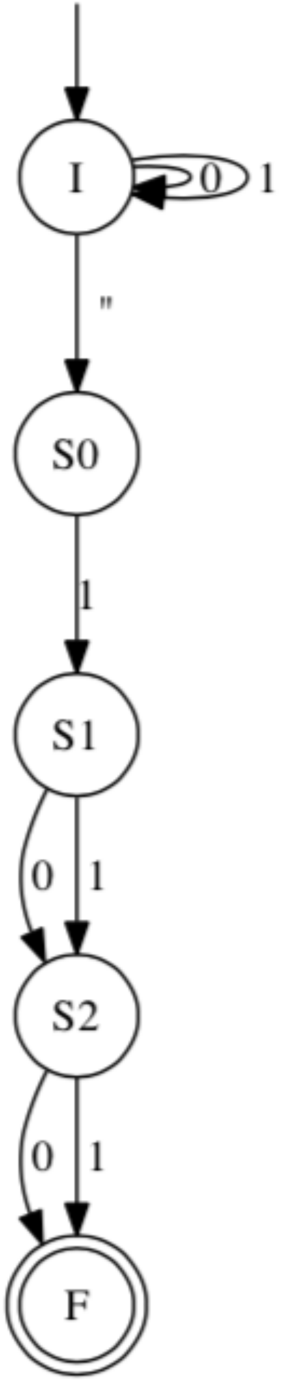
```
dotObj_nfa(nfahas0101)
```




```
dotObj_dfa(min_dfa(nfa2dfa(nfahas0101, STATENAME_MAXSIZE = 50)), STATENAME_MAXSIZE = 50)
```



What is an NFA formally?



Let Σ_ε stand for $(\Sigma \cup \{\varepsilon\})$. An NFA N is a structure $(Q, \Sigma, \delta, Q_0, F)$, where:

- Q is a *finite non-empty* set of states (as with DFA);
- Σ is a *finite non-empty* alphabet (as with DFA);
- $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathscr{P}(Q)$, is a transition function. An NFA's δ function takes a state in Q and a symbol or ε and returns a *set of states* (which is a member of $\mathscr{P}(Q)$, the *Powerset* of Q). See Figure 7.4 for the state transition table 'δ' for the example NFA.
- $Q_0 \subseteq Q$ is a *set of initial states*; and
- $F \subseteq Q$, is a *finite, possibly empty* set of final states.

State	Next state upon inputs		
	0	1	ε
I	$\{I\}$	$\{I\}$	$\{S0\}$
S0	{}	$\{S1\}$	{}
S1	$\{S2\}$	$\{S2\}$	{}
S2	$\{F\}$	$\{F\}$	{}
F	{}	{}	{}

```
{'Q'    : {'F', 'I',
           'S0','S1','S2'},
 'Sigma': {'0', '1'},
 'Delta':
 {('I', '0')  : {'I'},
  ('I', '1')  : {'I'},
  ('I', '')   : {'S0'},
  ('S0', '1') : {'S1'},
  ('S1', '0') : {'S2'},
  ('S1', '1') : {'S2'},
  ('S2', '0') : {'F'},
  ('S2', '1') : {'F'}},
  'q0': {'I'},
  'F' : {'F'}}
```

NFA to DFA Conversion

Algorithm for Subset Construction:

- Input: An NFA $N = (Q, \Sigma, \delta, Q_0, F)$
- Output: A language-equivalent DFA D
- Method: **Subset Construction**
  - Add the Eclosure of the initial state of the NFA as an unexpanded state of the DFA D being built. This would also be the **initial state of the DFA being built.**

    Repeat

    Choose a state S of D that has not been expanded

    Expand(S)

    Until there are no more unexpanded states in D
  - **Expand**(S):

    Mark S as expanded;

    If $S \cap F \neq \emptyset$, **record** S **to be a final state of the DFA**

    For each symbol c in Σ

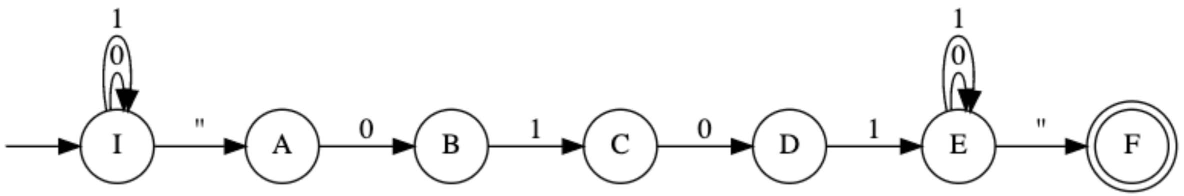
    For each state $s \in S$ do

    Let $s_c = \delta(s, c)$;

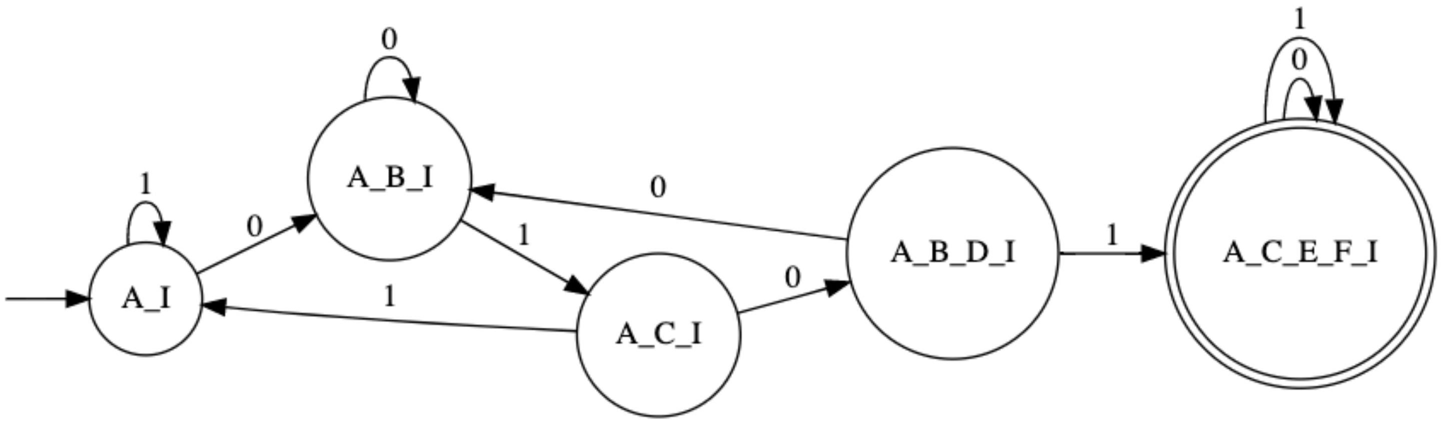
    Let $S_c = Eclosure((\cup_{s \in S} s_c))$;

Subset construction illustrated

```
dotObj_nfa(nfahas0101)
```




```
dotObj_dfa(min_dfa(nfa2dfa(nfahas0101, STATENAME_MAXSIZE = 50)), STATENAME_MAXSIZE = 50)
```

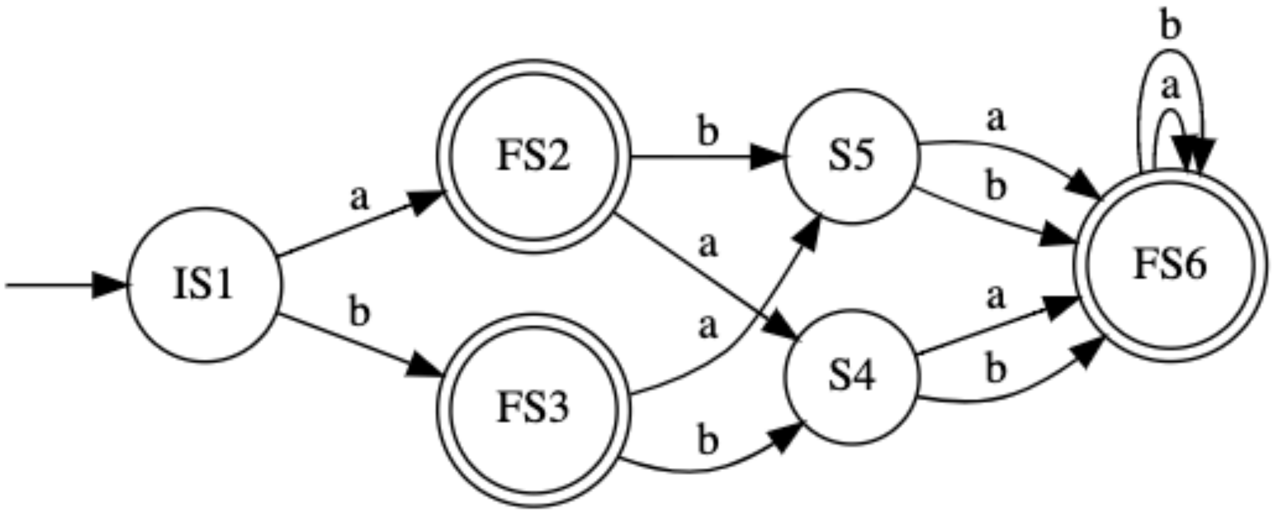


Review of concepts so far

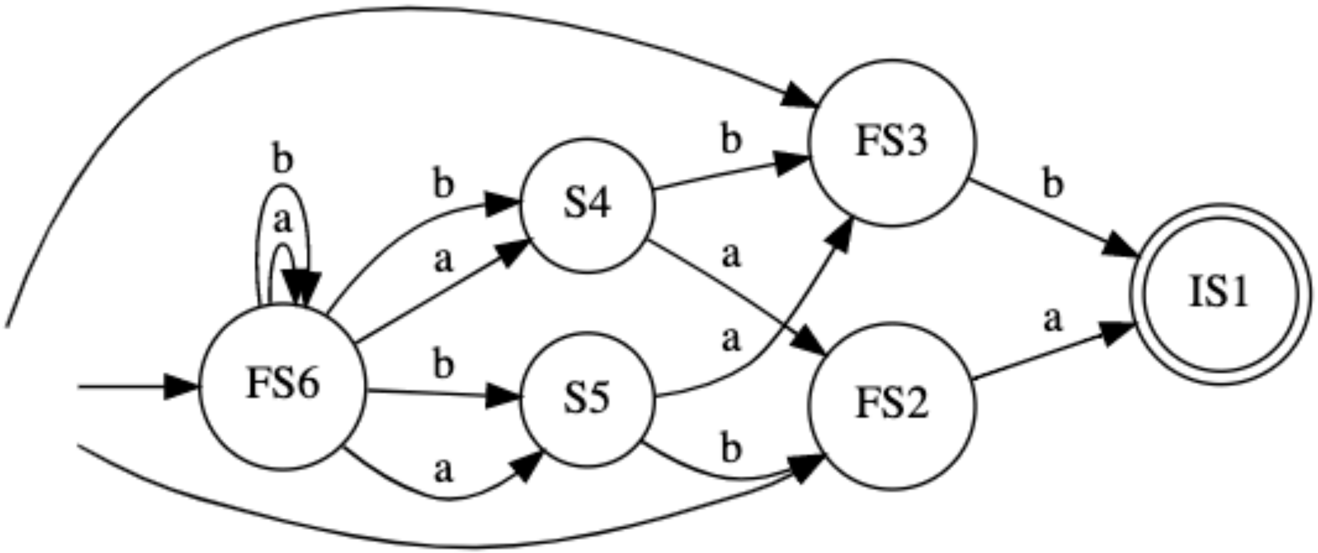
- NFA allow regular languages to be specified succinctly
  - No direct NFA minimization!
  - But they are often quite succinct
  - NFA can never be larger than DFA
    - DFA are essentially NFA
      - No epsilon moves
      - Next SET of states is to a singleton set
- NFA can be converted to a DFA with a potential exp blowup
  - Exp blowup is apparent when we convert the “Nth-last is a 1” NFA to a DFA
  - Algorithm is called subset construction

Reversal of DFA produce NFA

```
dotObj_dfa(FBloat)
```




```
dotObj_nfa(rev_dfa(FBloat))
```



What’s the language of FBloat and its reverse?

- Language of FBloat via language operations

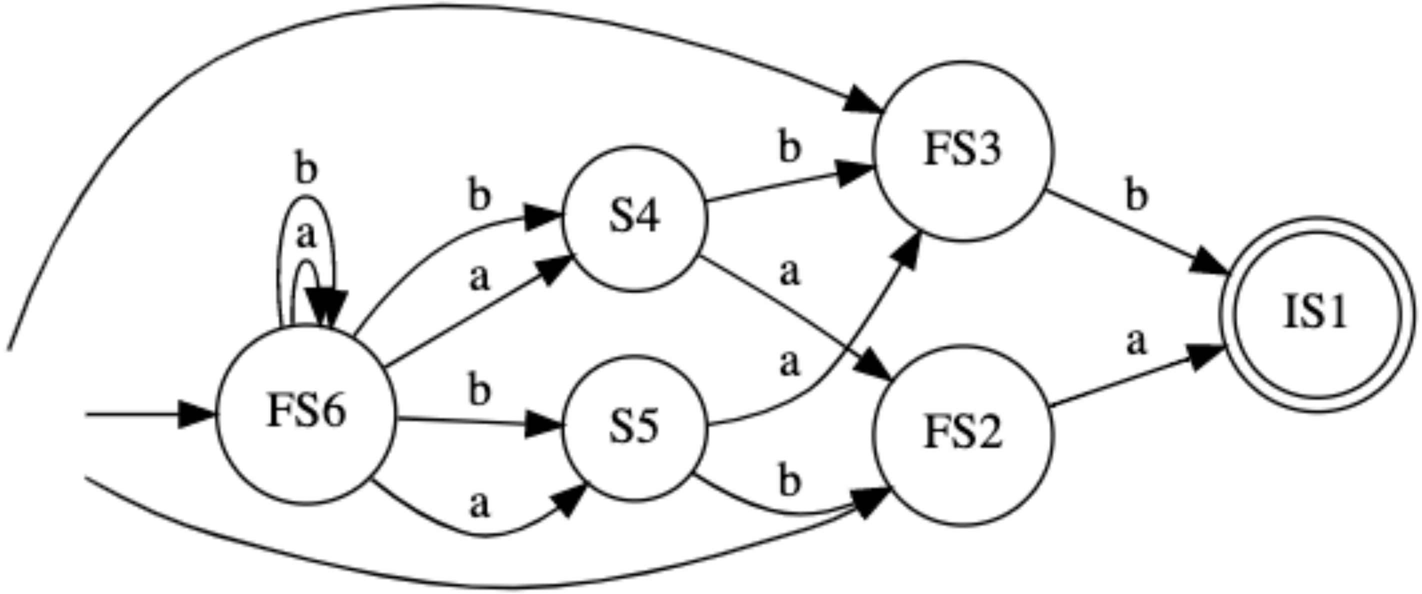
- Language of rev_dfa(Fbloat)

Reversal followed by nfa2dfa

i.e.

R ; D
so far

Do subset constrn.

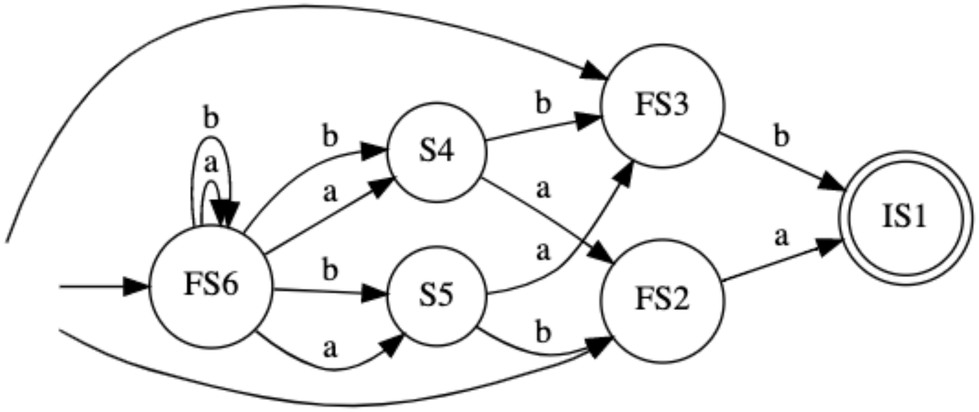


Reversal followed by nfa2dfa

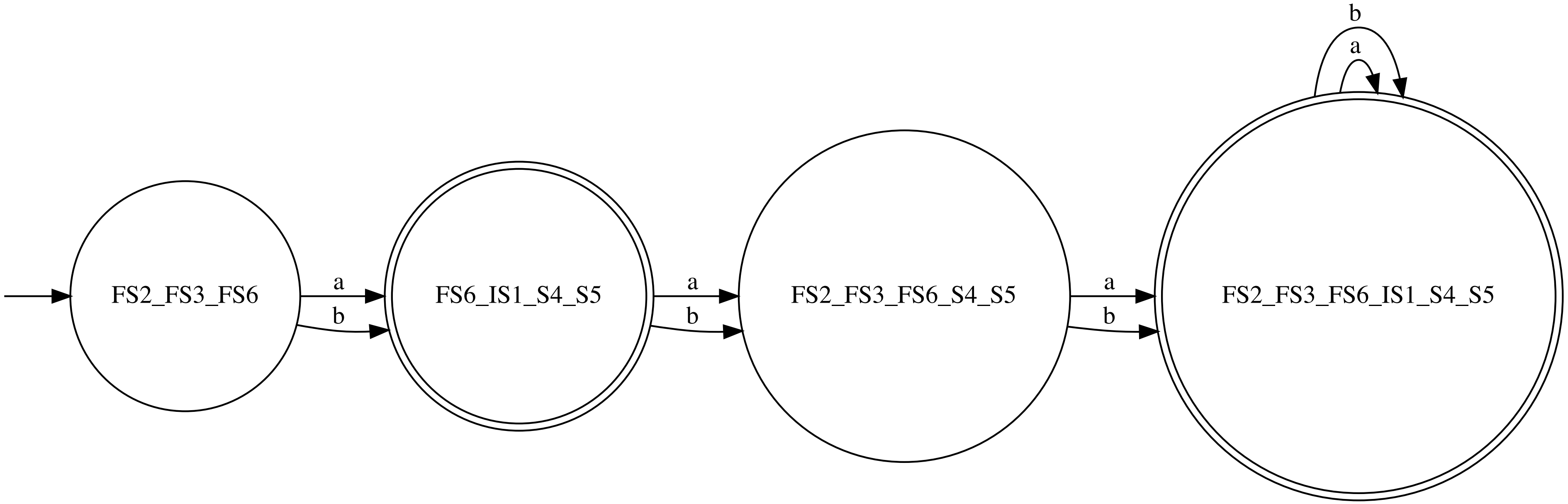
i.e.

R ; D
so far

Do subset constrn.



dotObj_dfa(nfa2dfa(rev_dfa(FBloat), STATENAME_MAXSIZE=50), STATENAME_MAXSIZE=50).render('/private/tmp/rdbloat')

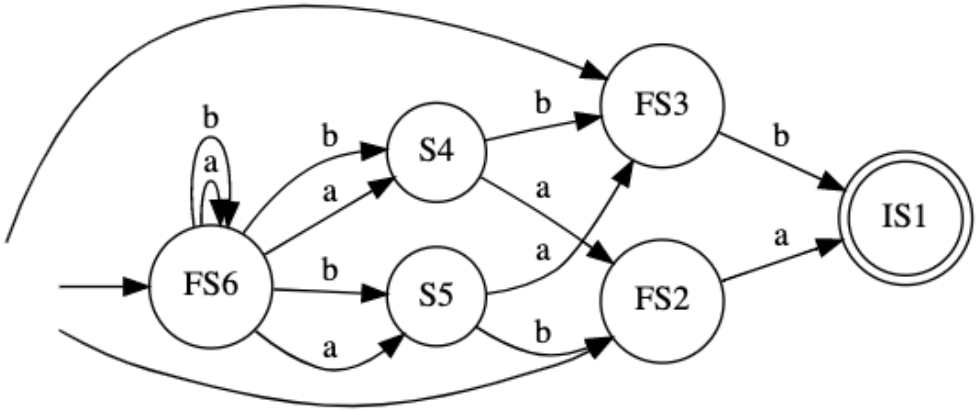


Reversal followed by nfa2dfa

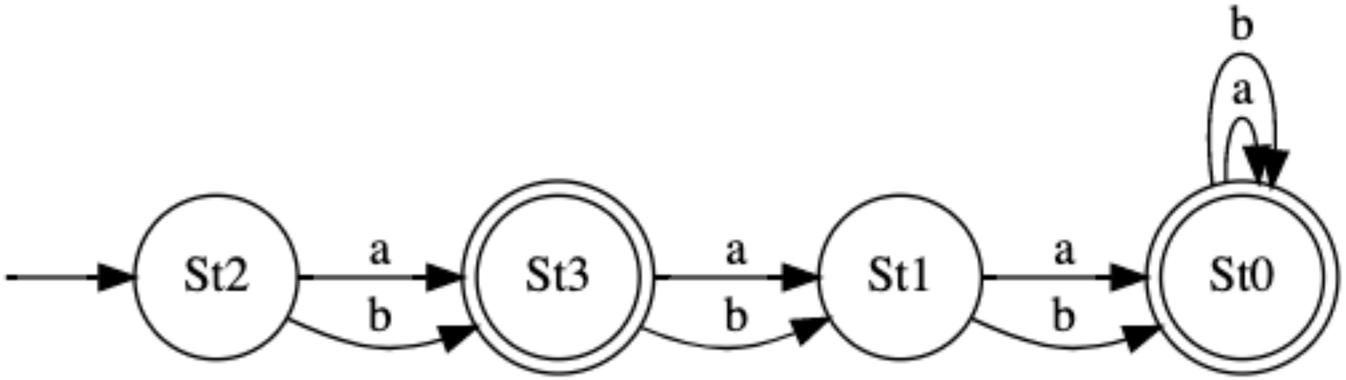
i.e.

R ; D so far

Do subset constrn.

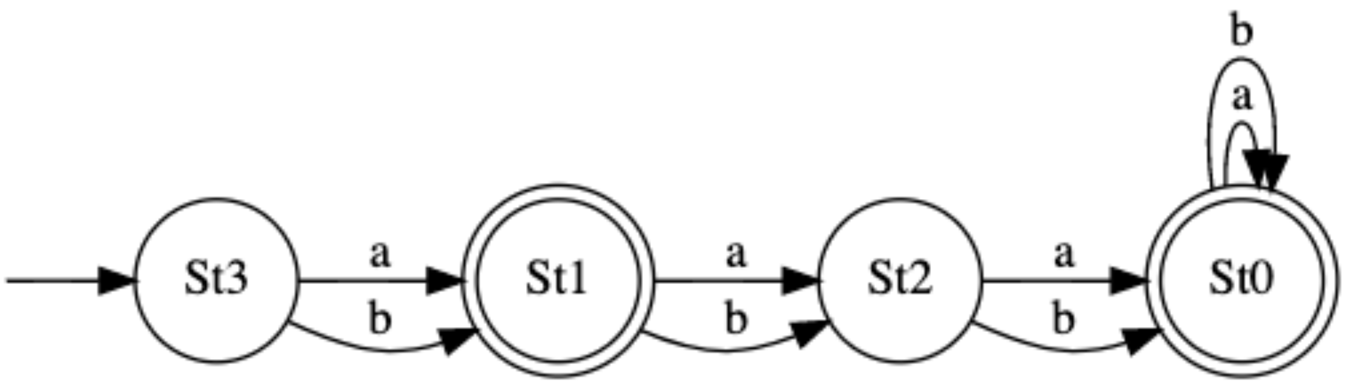



```
dotObj_dfa(nfa2dfa(rev_dfa(FBloat)))
```

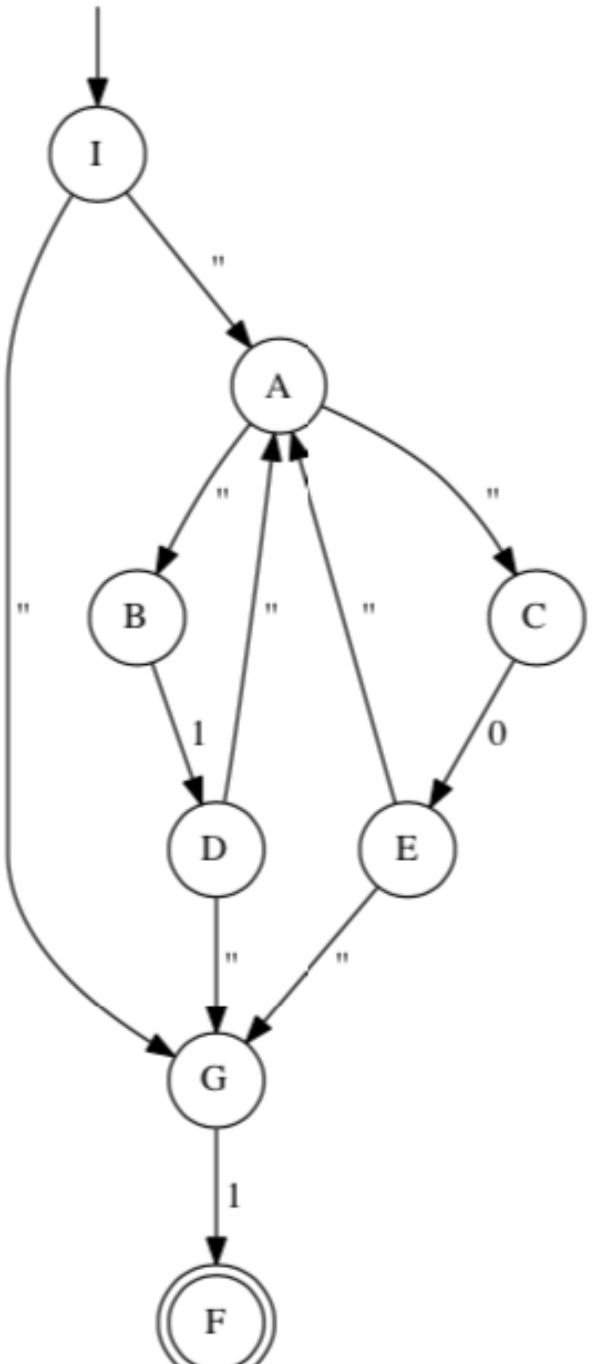


R ; D ; R ; D is Brzozowski's minimization!

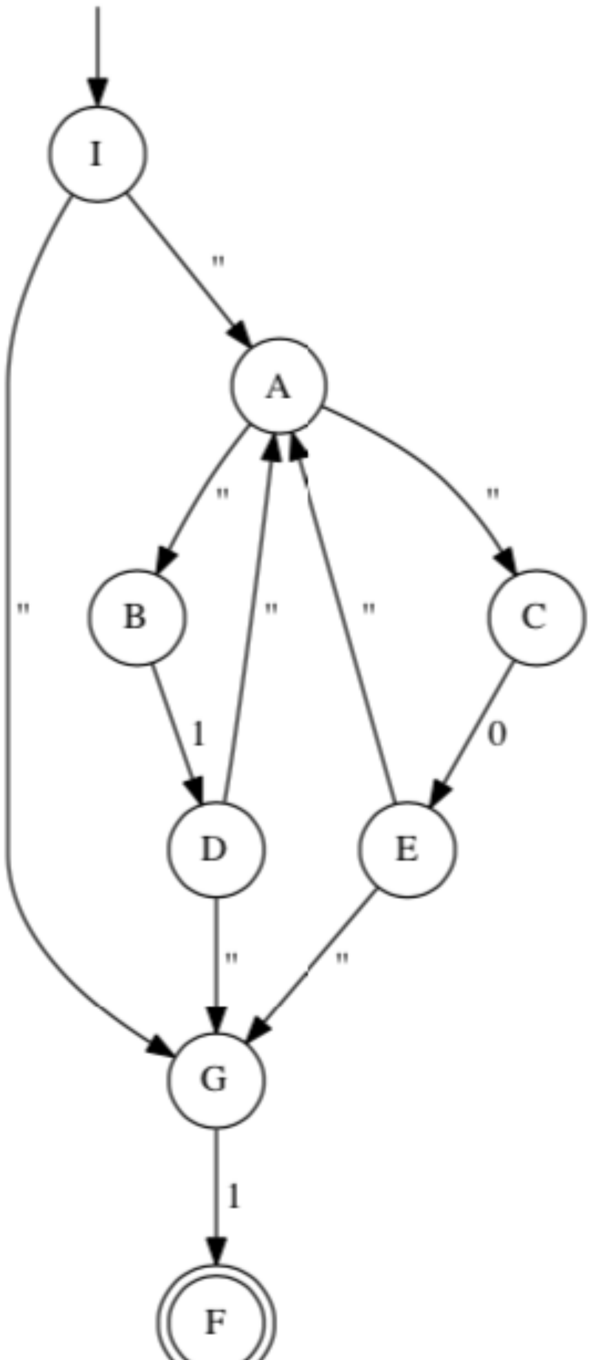
```
dotObj_dfa(nfa2dfa(rev_dfa(nfa2dfa(rev_dfa(FBloat)))))
```

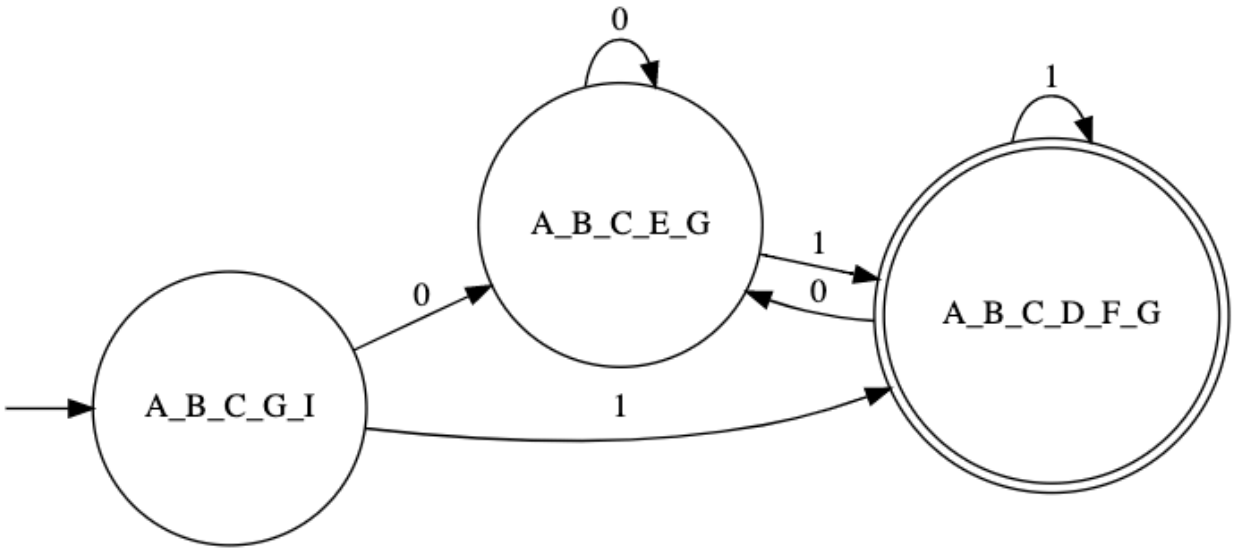


NFA2DFA for NFA with epsilons



NFA2DFA for NFA with epsilons





Summary

- DFA minimization can be done via Rev;Det;Rev;Det
  - This is Brzozowski's algorithm