Model Checking with BDDs

Sets as Propositions

- Consider the universe "ABCD"

```
[('A',[~p2,~p1]),('B',[~p2,p1])
 ,('C',[p2,~p1]),('D',[p2,p1])]
```

- Or the universe [1,5,6,79,13]

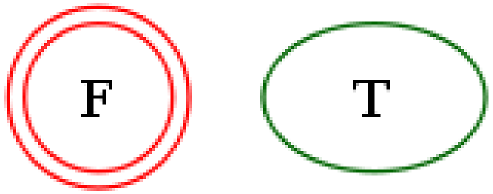
```
[(1,[~p3,~p2,~p1]),(5,[~p3,~p2,p1])
,(6,[~p3,p2,~p1]),(79,[~p3,p2,p1])
,(13,[p3,~p2,~p1])]
```

Consider some subsets

```
subset "ABCD" "A"
~p1 /\ ~p2

subset "ABCD" "AC"
(~p1 /\ ~p2) \/ (~p1 /\ p2)

subset "ABCD" "ACDB"
(~p1 /\ ~p2) \/ (p1 /\ ~p2) \/ (~p1 /\ p2)
  \/ (p1 /\ p2)

subset "ABCD" ""
Absurd
```

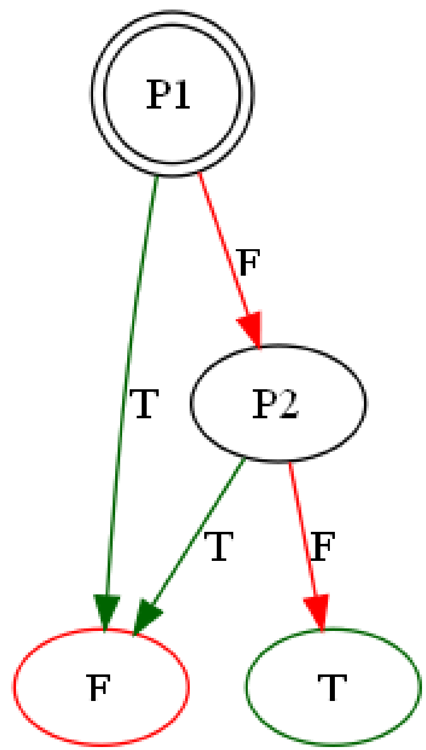
And their BDDs

- Set “ABCD” “”
- Absurd



```
[('a',[~p2,~p1]),('b',[~p2,p1]),
 ('c',[p2,~p1]),('d',[p2,p1])]
```

```
subset "abcd" "a“
```

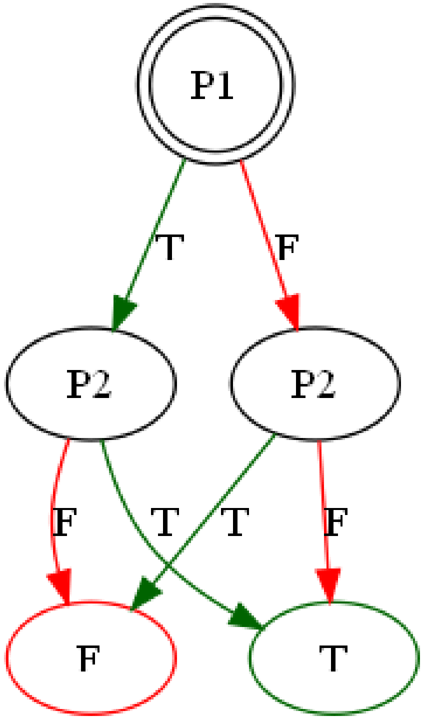



```
~p1 /\ ~p2
```

[('a',[~p2,~p1]),('b',[~p2,p1]),
('c',[p2,~p1]),('d',[p2,p1])]

subset “abcd” “ad”

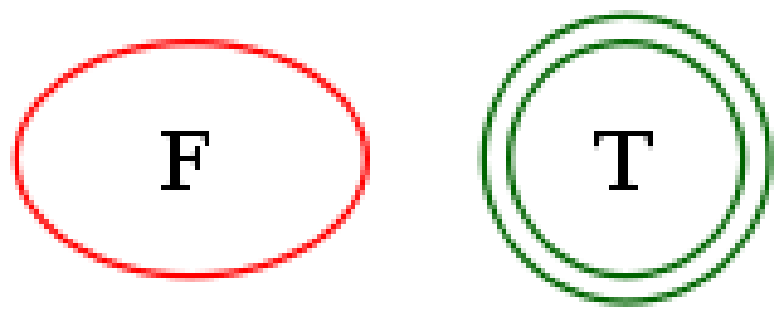
(~p1 /\ ~p2) \/
(p1 /\ p2)



```
[('a',[~p2,~p1]),('b',[~p2,p1]),
 ('c',[p2,~p1]),('d',[p2,p1])]
```

```
subset “abcd” “adbc”
```

```
(~p1 /\ ~p2) \/
(p1 /\ ~p2) \/
 (~p1 /\ p2) \/
(p1 /\ p2)
```



```
initial xs = zip xs (reverse (g 1))
  where n = numbits (length xs)
        g:: Int -> [[Prop Int]]
        g m | m > n = [[]]
        g n = map (LetterP n:) ys ++ map ((NotP
  (LetterP n)):) ys
           where ys =  (g (n+1))
subset univ set = foldr acc AbsurdP univ
  where acc x prop | elem x set = orOpt (get x) prop
        acc x prop = prop
        mapping = initial univ
        get n = case lookup n mapping of
                  Just literals -> andL literals
```

Membership test

- Represent an element of a set as the singleton subset
- item univ x = subset univ [x]
- Then membership uses the tautology
- {x} `elem` zs iff {x} == {x} ∩ zs

Lift to BDDs

```
subsetB x y = p2b (subset x y)
itemB x y = p2b(item x y)

mem univ x xs = same temp (conj
  temp (subsetB univ xs))
   where temp = (itemB univ x)
```

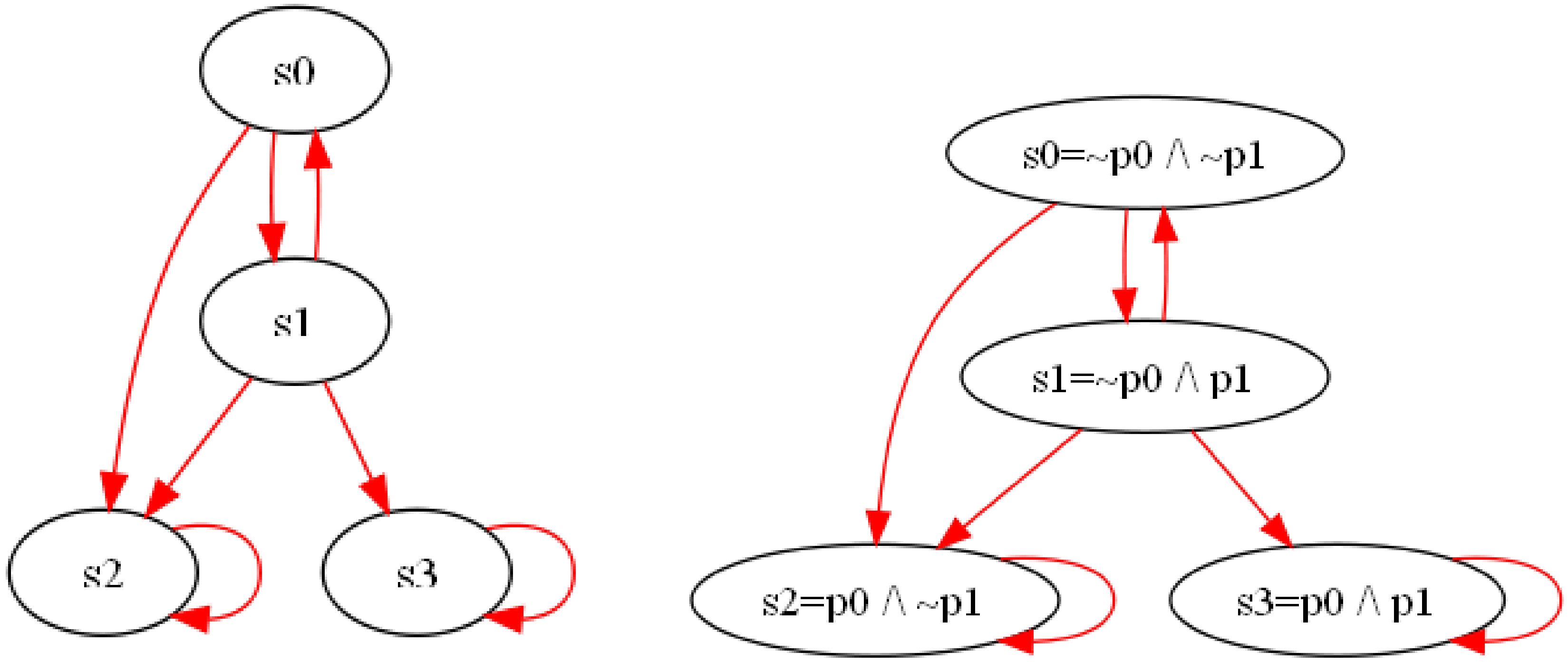
A relation between two items in a set

```
[('a',[~p2,~p1]),
 ('b',[~p2,p1]),
  ('c',[p2,~p1]),
   ('d',[p2,p1])]
```

- R(a,b) = True
- R(b,c) = True
- R(c,c) = True
- R(_,_) = False

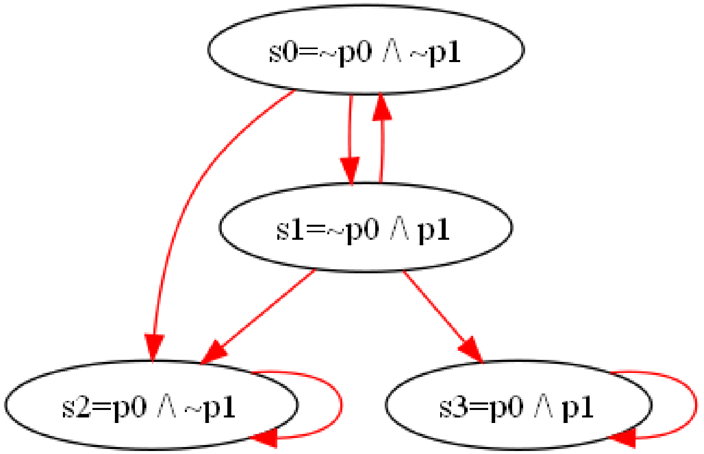
Graph transitions

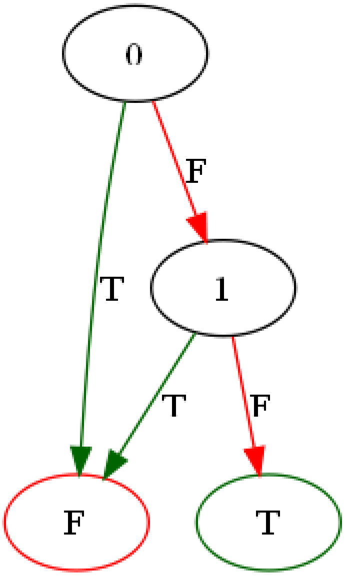
- Consider the graph and its assignment of states to boolean formula



Recall how we represent subsets

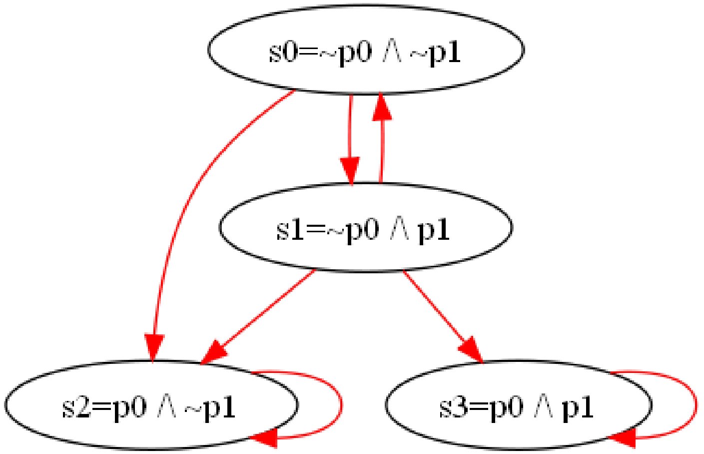
- u1 = [0,1,2,3]
- sub1 = subset u1 [0]
- ~p0 /\ ~p1

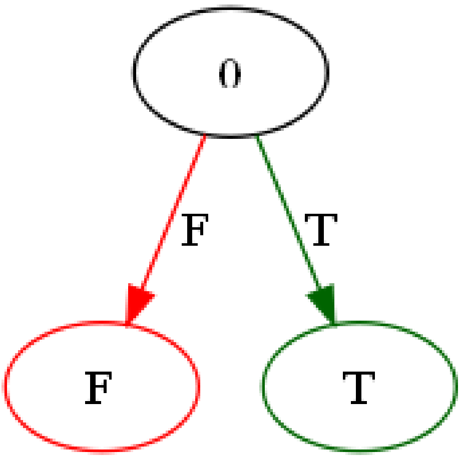




Subset {s2,s3}

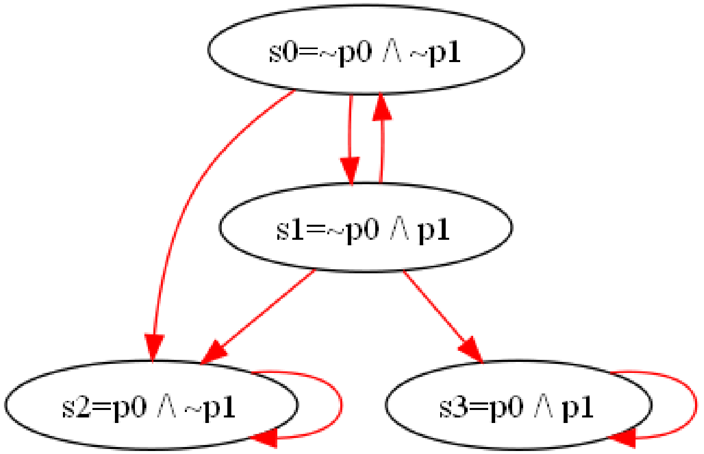
- u1 = [0,1,2,3]
- sub2 = subset u1 [2,3]
- (p0 /\ ~p1) \/ (p0 /\ p1)

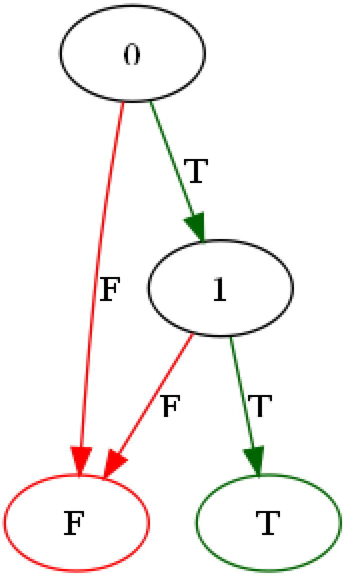




Subset {s3}

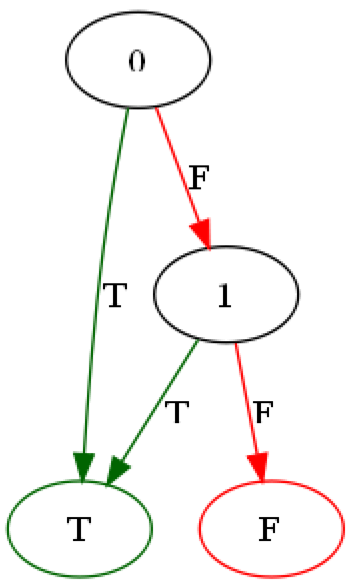
- u1 = [0,1,2,3]
- sub3 = subset u1 [3]
- p0 /\ p1

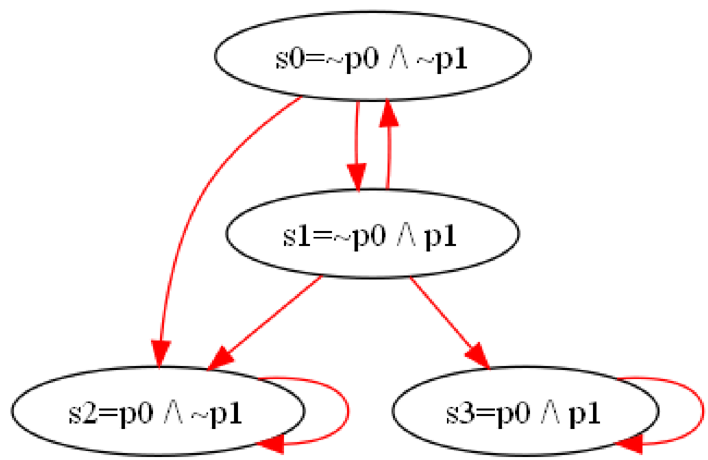




Subset {s1,s2,s3}

- u1 = [0,1,2,3]
- sub4 = subset u1 [1,2,3]
- (~p0 /\ p1) \/ (p0 /\ ~p1) \/ (p0 /\ p1)

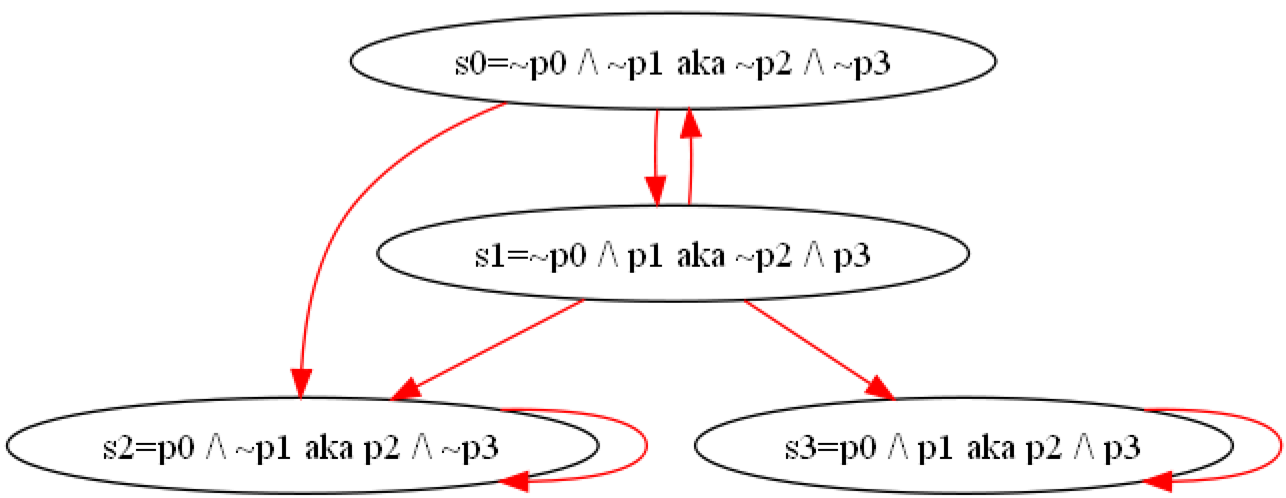


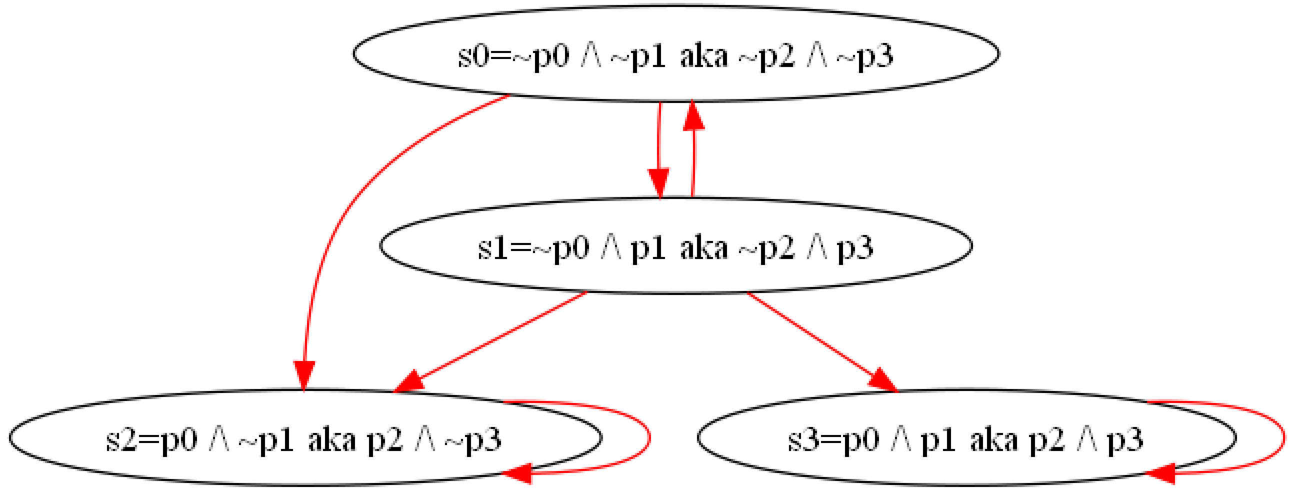


The transition relation

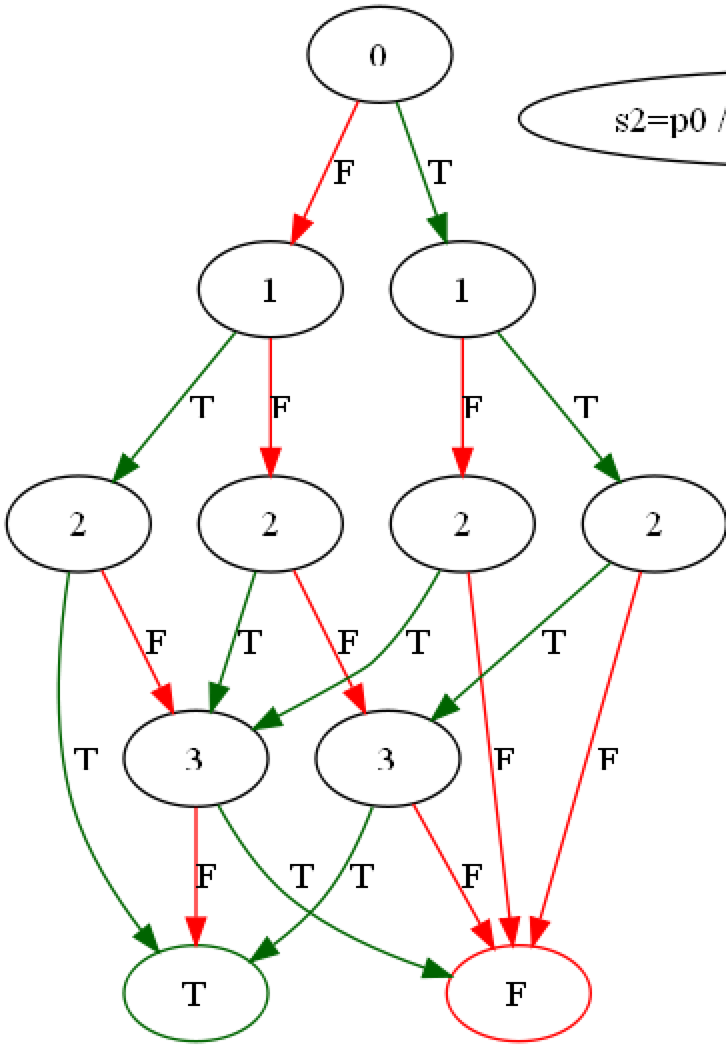
- Introduce new variables p2 and p3 that mirror p0 and p1

```
prop1 =
(~p0 /\ ~p1 /\ ~p2 /\ p3)
\/
(~p0 /\ ~p1 /\ p2 /\ ~p3)
\/
(~p0 /\ p1 /\ ~p2 /\ ~p3)
\/
(~p0 /\ p1 /\ p2 /\ ~p3)
\/
(p0 /\ ~p1 /\ p2
\/
(~p0 /\ p1 /\ p2
\/
(p0 /\ p1 /\ p2 /
```





s0=~p0 /\ ~p1 aka ~p2 /\ ~p3
s1=~p0 /\ p1 aka ~p2 /\ p3
s2=p0 /\ ~p1 aka p2 /\ ~p3
s3=p0 /\ p1 aka p2 /\ p3



0
F
T
1
1
T
F
F
T
2
2
2
2
F
T
F
T
T
3
3
T
F
F
T
T
F
T
F

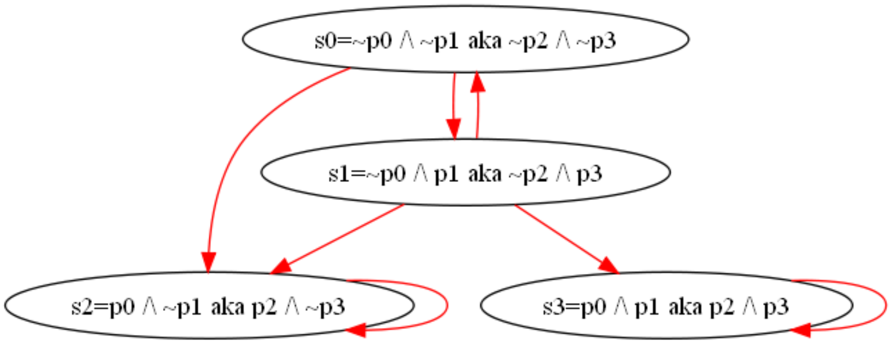
As a BDD

States reachable in one step

- Let sub be a set of states
- What is reachable in one step?

- step set = prop1 /\ set

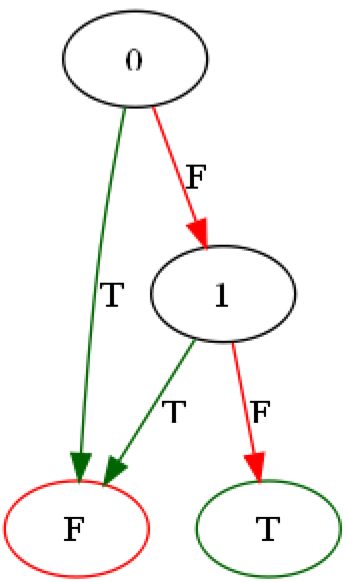
To take a step conjoin

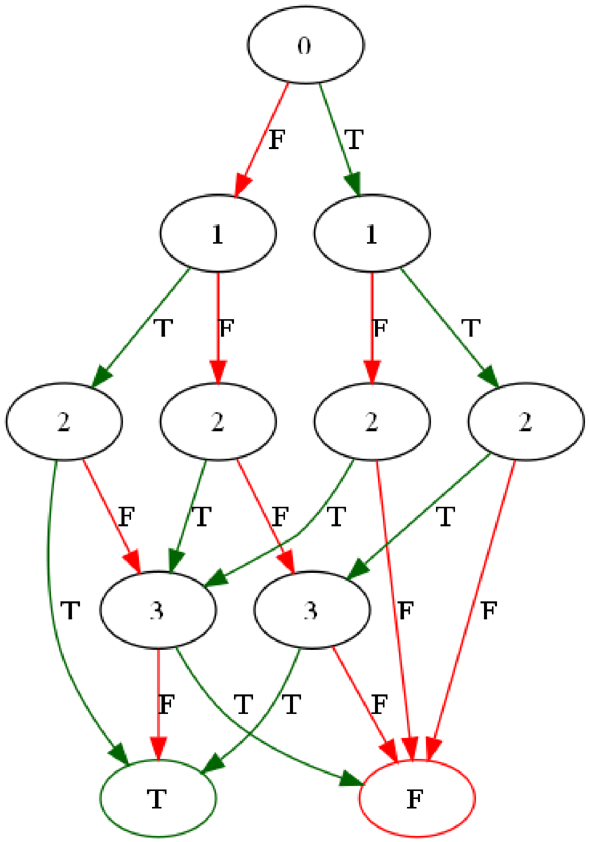


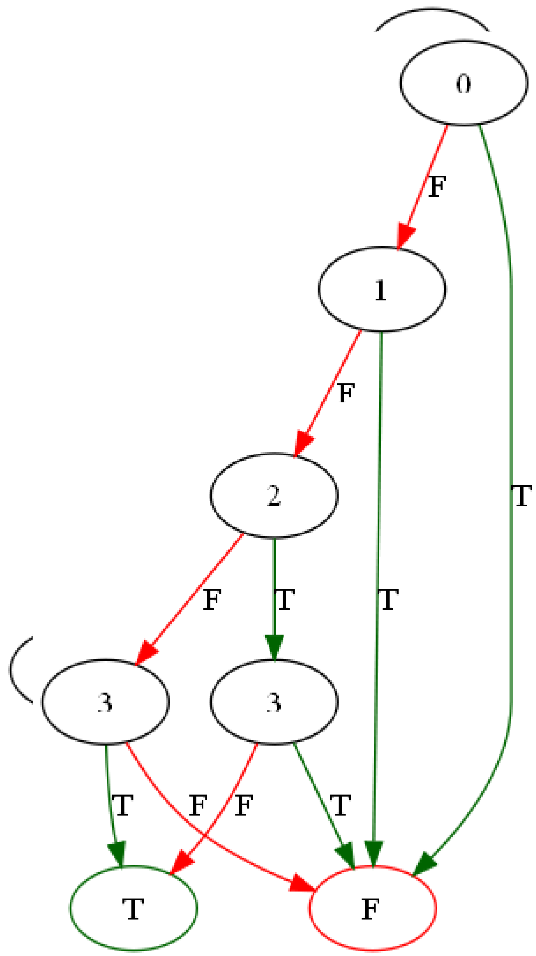
- sub1={s0}

prop1

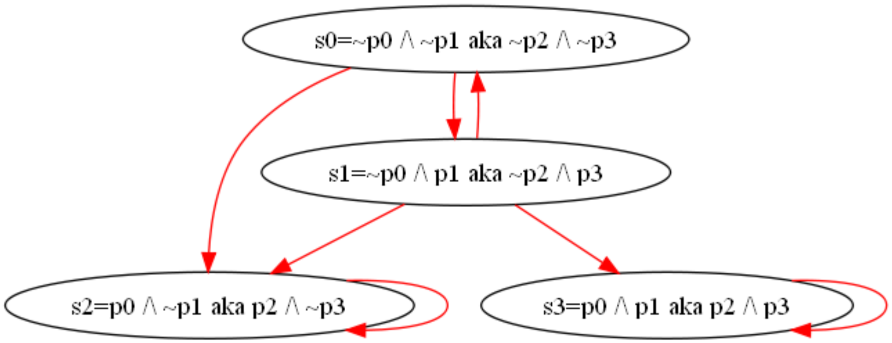
sub1 /\ prop1

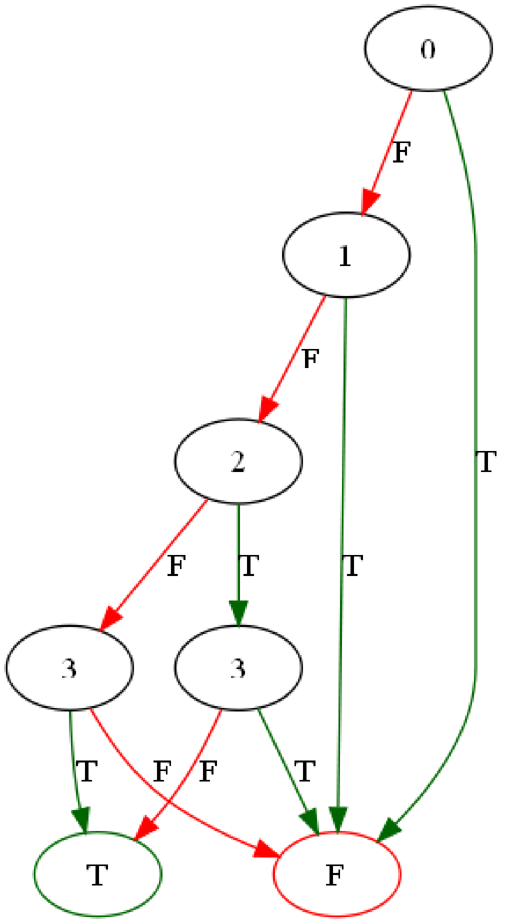






- Note the paths to True
- There are two of them
- Each corresponds to one next state
- The values of p2 and p3 tell what states {s1,s2}





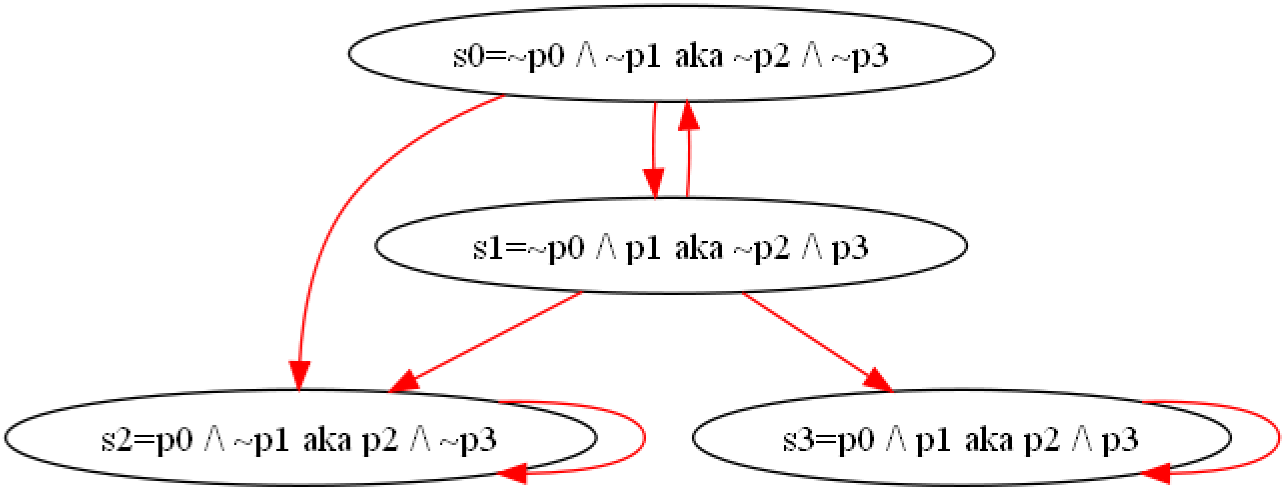
Consider the solutions

- [[(0,False),(1,False),(2,False),(3,True)],
- [(0,False),(1,False),(2,True),(3,False)]]

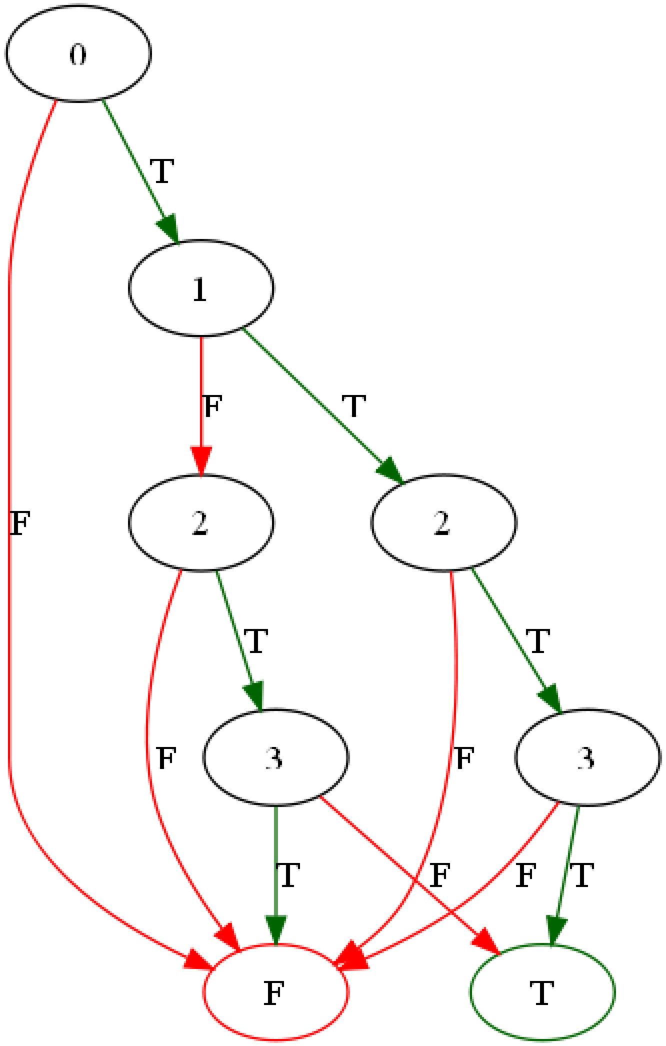
- By throwing away the assignments to p0 and p1, and be renaming p2 to p0, and p3 to p1, we get to solutions

- ~p0 /\ p1
- p0 /\ ~p1

- Cooresponding
- to the states
- {s1,s2}

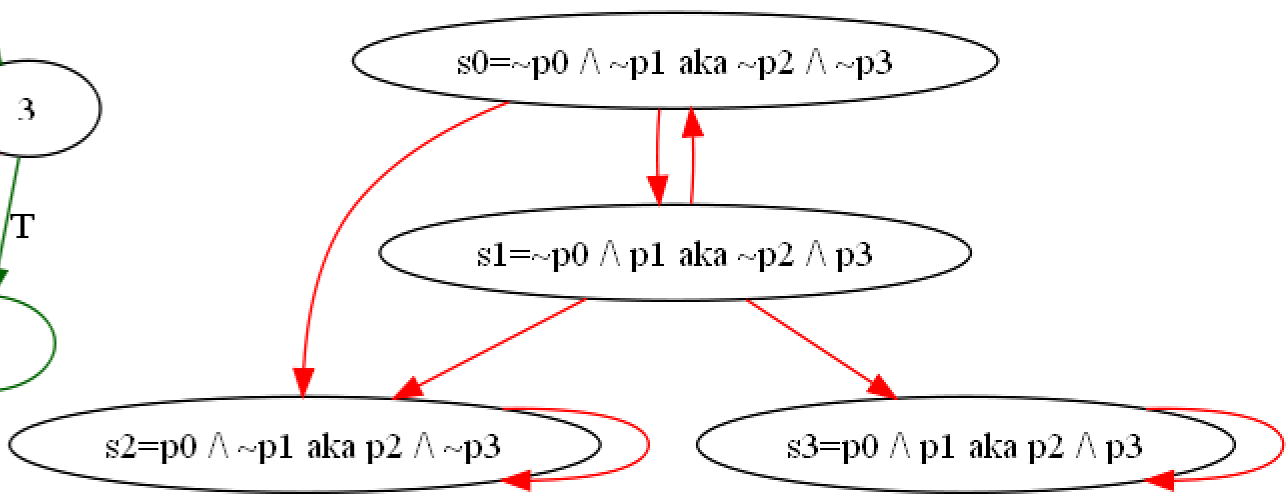


Start at the set {s2,s3}

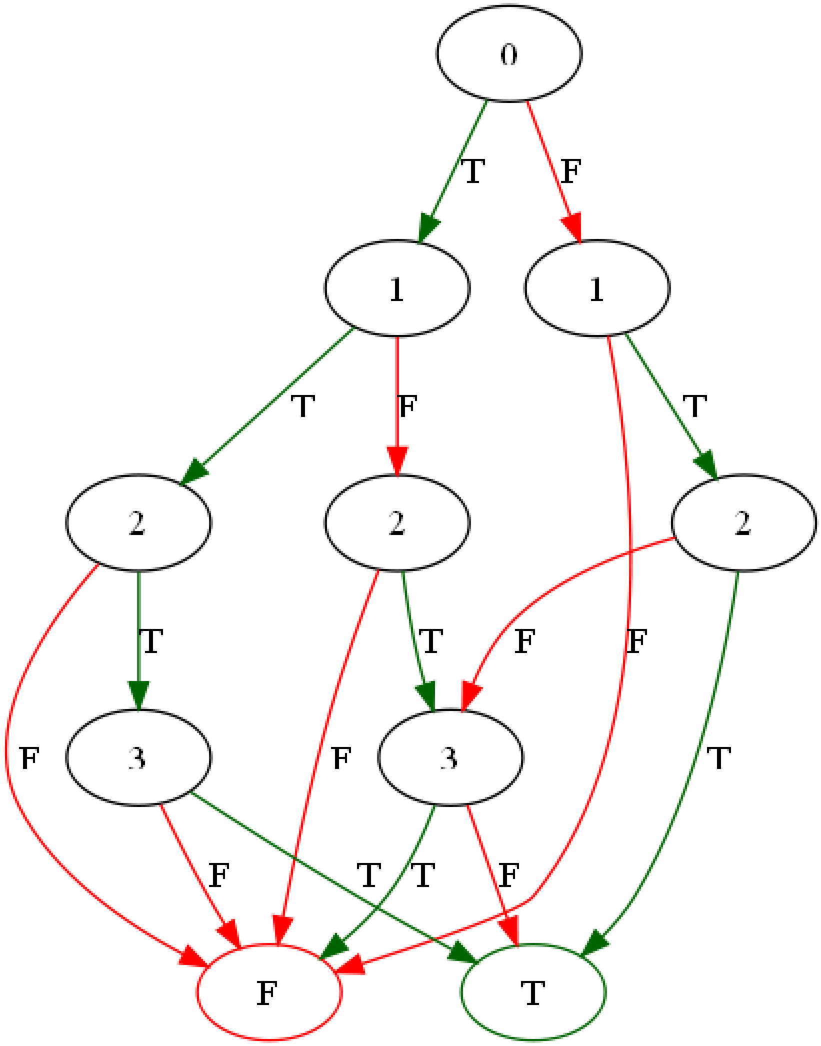


- [[(0,True),(1,False),(2,True),(3,False)] ,
- [(0,True),(1,True),(2,True),(3,True)]]

- p0 /\ ~p1
- p0 /\ p1



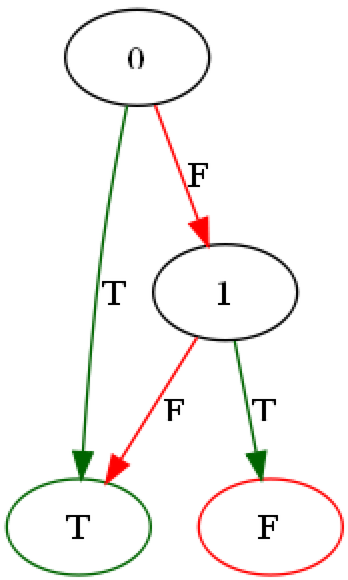
Start at the subset {s1,s2,s3}

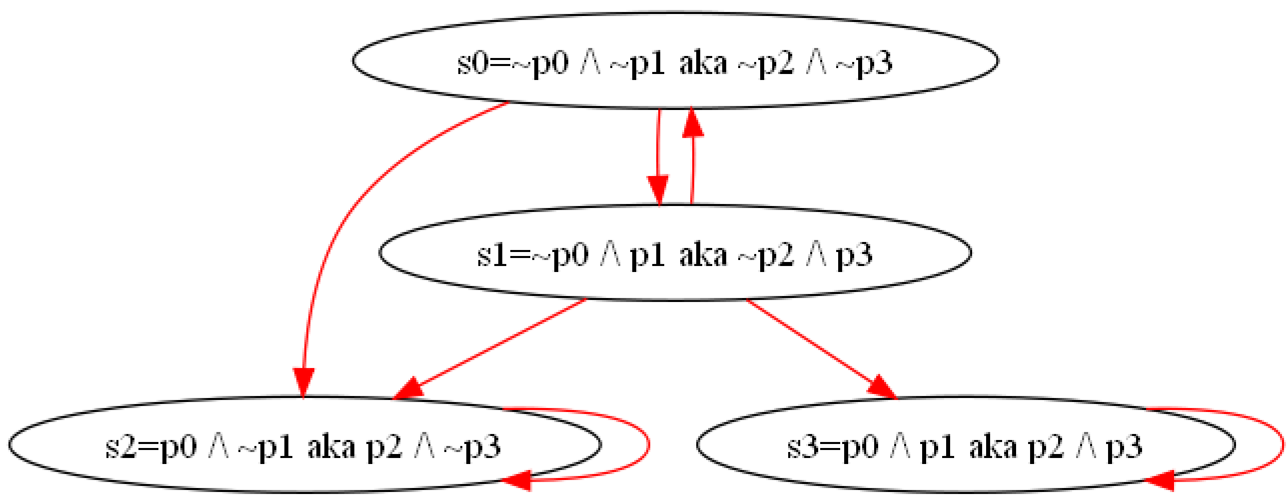


- ~p0 /\ ~p1
- p0
- p0 /\ ~p1
- p0 /\ p1

```
p0 \/ (p0 /\ p1) \/ (p0 /\ ~p1) \/ (~p0 /\ ~p1)
```

- This corresponds to the BDD
- Which is every state except s2, which is exactly what can be reached from {s1,s2,s3}

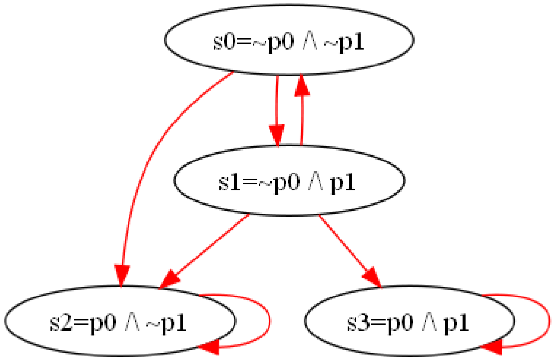


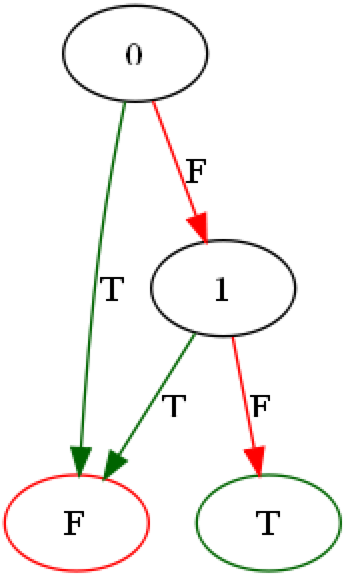


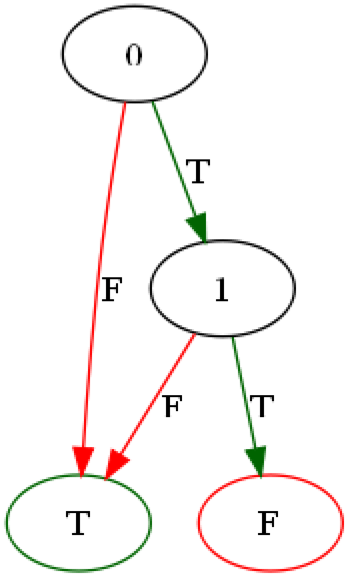
To take multiple steps

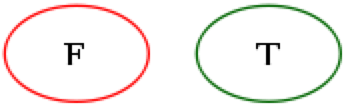
- Compute the states reachable in 1 step
- Union in the starting states
- And repeat

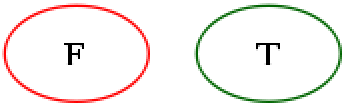
- sub1 = {s0}
- *LectureBDD> pnG (p2b sub1)
- *LectureBDD> pnG (p2b (step sub1))
- *LectureBDD> pnG (p2b (step (step sub1)))
- *LectureBDD> pnG (p2b (step (step (step sub1))))











- sub3 = {s3}
- *LectureBDD> pnG (p2b sub3)
- *LectureBDD> pnG (p2b (step sub3))