

SAT-Based Model Checking

Chapter 10

Outline

- Bounded Model Checking
- Verifying Reachability Properties with k -induction
- Model Checking with Inductive Invariants
- Model Checking with Craig Interpolants
- Property-Directed Reachability

Overview

- SAT solvers can solve propositional formulas. (See Chapter 9.)
- SAT solvers have become very fast

Techniques

- Bounded Model Checking: Is there a trace of length k that violates the property?
- k -induction: Can we prove the property inductively for *any* trace?
- Create an inductive invariant that is stronger than the property
 - Craig Interpolants
 - Property-Directed Reachability

In this chapter, we will only consider LTL formulas

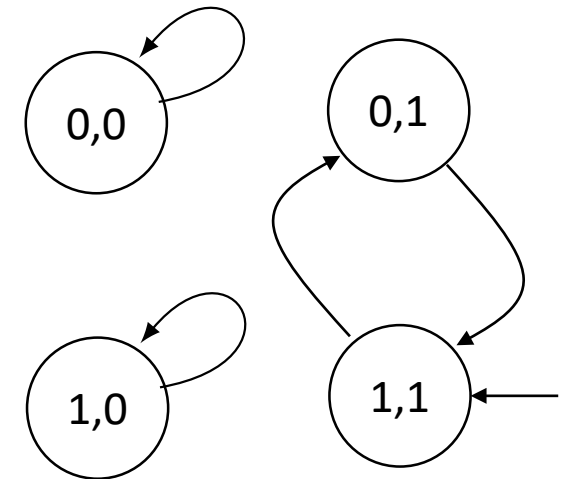
Preliminaries

Postimage

$\text{postimg}(S)$ = states reachable from S in one step

$\text{postimg}(\{ (0,1) \}) =$

$\text{postimg}(\{ (0,0), (1,1) \}) =$



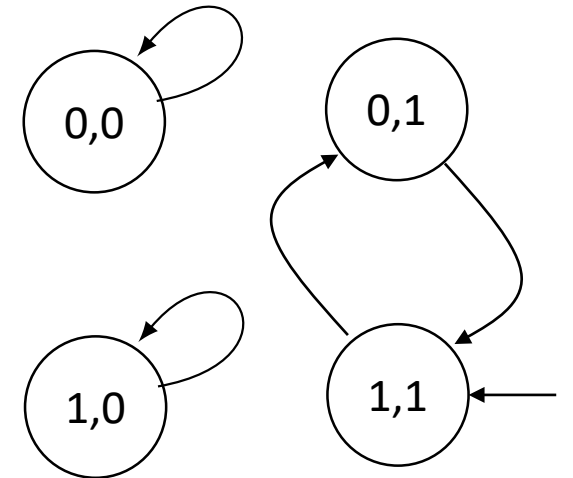
Kripke structure

Paths

Initial states: $\mathcal{S}_0(x, y) = (x = 1 \wedge y = 1)$

Transitions: $\mathcal{R}(x, y, x', y') = (x' = (x + y) \bmod 2) \wedge (y' = y)$

$$\begin{aligned} Path_1(V, V') &= S \wedge R(V, V') = \\ &x \wedge y \wedge (x' = (x + y) \bmod 2) \wedge (y' = y) = \\ &x \wedge y \wedge \neg x' \wedge y'. \end{aligned}$$



Kripke structure

Bounded Model Checking

Computer Aided Verification Award 2018

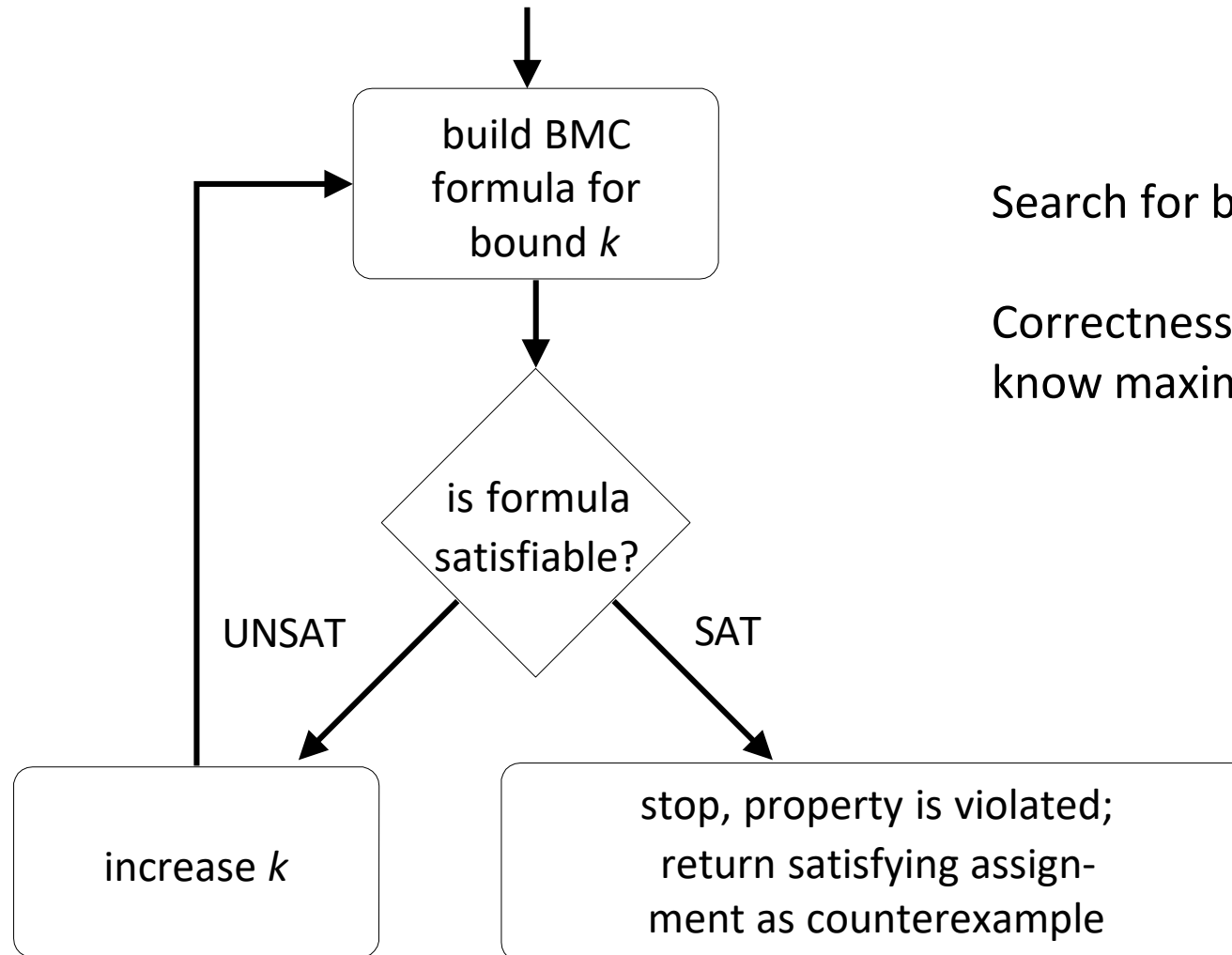
Bounded Model Checking has revolutionized the way model checking is used and perceived. It has increased the capabilities of model checkers by orders of magnitude, turning them into a standard tool for hardware verification and a very important component of the toolkit available for software verification...

(1999)



Flavio Lerda, Daniel Kroening, Armin Biere, Alessandro Cimatti,
Not pictured: Edmund M. Clarke, Yunshan Zhu

Bounded Model Checking



Search for bugs within k steps.

Correctness can only be proven if we know maximal value for k

Reachability Properties

Property

p always holds iff we cannot reach a state with $\neg p$

Kripke Structure

$M = (S, S_0, R, AP, L)$ – represented symbolically (See Chapter 3)

State variables $V = \{v_1, \dots, v_n\}$

Reachability – Paths

$$\mathit{path}_0(s_0) =$$

$$\mathit{path}_1(s_0, s_1) =$$

$$\mathit{path}_2(s_0, \dots, s_2) =$$

$$\mathit{path}_k(s_0, \dots, s_k) =$$

Reachability – Paths

$$path_0(s_0) = S_0(s_0)$$

$$path_1(s_0, s_1) = S_0(s_0) \wedge R(s_0, s_1)$$

$$path_2(s_0, \dots, s_2) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2)$$

$$path_3(s_0, \dots, s_3) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2) \wedge R(s_2, s_3)$$

$$path_k(s_0, \dots, s_k) = S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$$

Reachability – Building the Formula

Reachability – Building the Formula

$$path_k(s_0, \dots, s_k) = S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$$

Path starts in initial state Exists transition from s_i to s_{i+1}

System is **incorrect** within k steps if

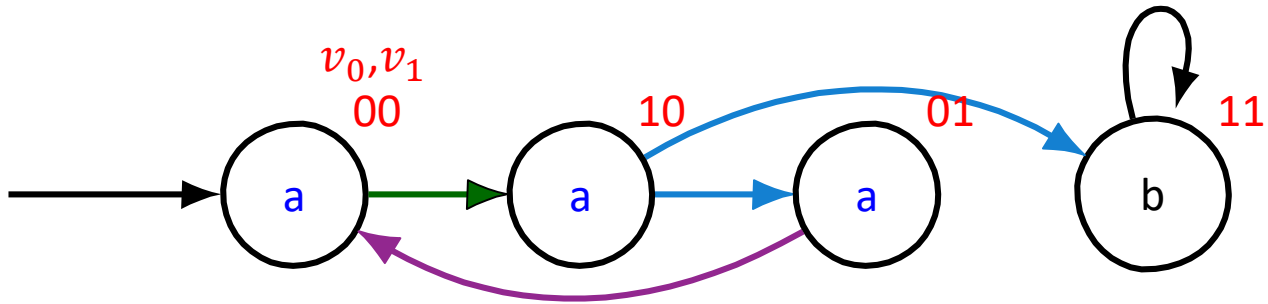
$$path_k(s_0, \dots, s_k) \wedge \bigvee_{i=0}^k \neg p(s_i)$$

There is a path to s_k One of the state violates p

Reachability – Correctness

Theorem 10.1. $path_k(s_0, \dots, s_k) \wedge \bigvee_{i=0}^k \neg p(s_i)$ is satisfiable iff there is a counterexample to $AG\ p$ of length $\leq k$.

Example



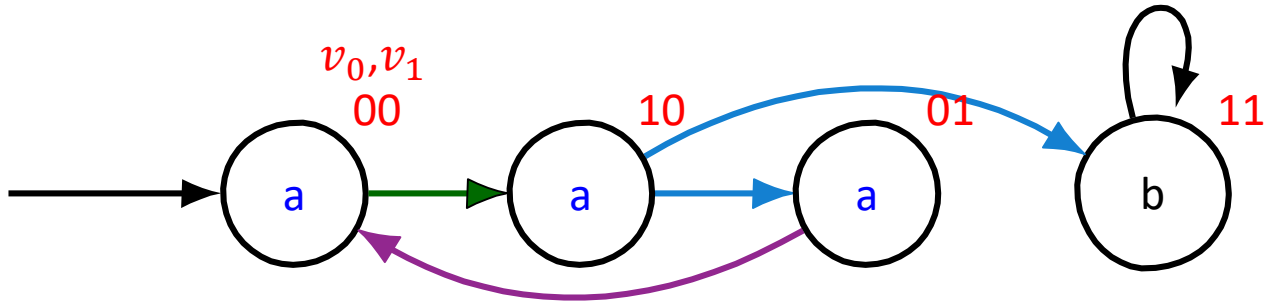
AG a

$S_0(v_0, v_1) =$.

$R(v_0, v_1, v'_0, v'_1) =$

$a(v_0, v_1) =$.

Example



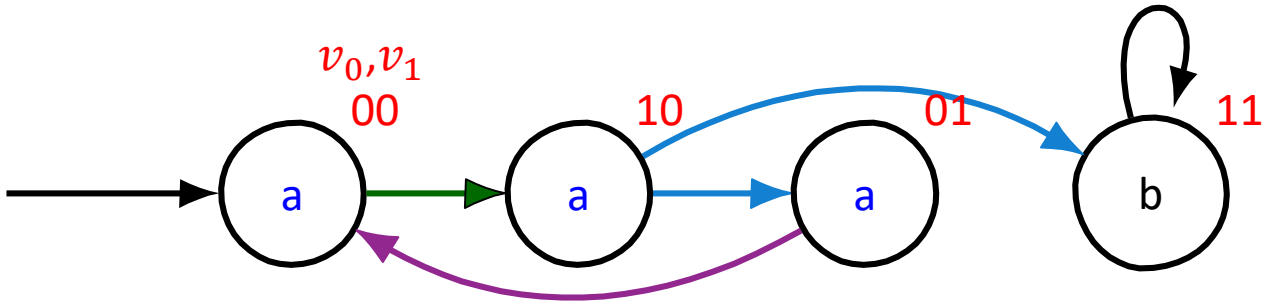
AG a

$$S_0(v_0, v_1) = \neg v_0 \wedge \neg v_1$$

$$\begin{aligned} R(v_0, v_1, v'_0, v'_1) \\ = & \neg v_0 \wedge \neg v_1 \wedge v'_0 \wedge \neg v'_1 \vee v_0 \wedge \neg v_1 \wedge v'_1 \\ \vee & \neg v_0 \wedge v_1 \wedge \neg v'_0 \wedge \neg v'_1 \vee v_0 \wedge v_1 \wedge v'_0 \wedge v'_1 \end{aligned}$$

$$a(v_0, v_1) = \neg v_0 \vee \neg v_1$$

Example: k=0



$$S_0(v_0, v_1) = \neg v_0 \wedge \neg v_1$$

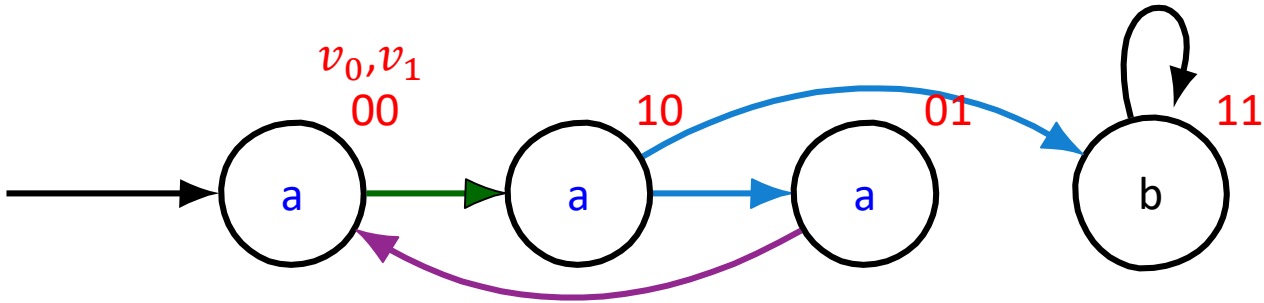
$$\begin{aligned} path_0(s_0, s_1, s_2) &= S_0(s_0) \\ &= \neg v_{0,0} \wedge \neg v_{1,0} \end{aligned}$$

$$\begin{aligned} R(v_0, v_1, v'_0, v'_1) &= \neg v_0 \wedge \neg v_1 \wedge v'_0 \wedge \neg v'_1 \vee v_0 \wedge \neg v_1 \wedge v'_1 \\ &\vee \neg v_0 \wedge v_1 \wedge \neg v'_0 \wedge \neg v'_1 \vee v_0 \wedge v_1 \wedge v'_0 \wedge v'_1 \end{aligned}$$

$$a(v_0, v_1) = \neg v_0 \vee \neg v_1$$

$$\bigvee_{i=0}^k \neg a(s_i) = \neg(\neg v_{00} \vee \neg v_{10})$$

Example: k=1



$$S_0(v_0, v_1) = \neg v_0 \wedge \neg v_1$$

$$R(v_0, v_1, v'_0, v'_1) = \neg v_0 \wedge \neg v_1 \wedge v'_0 \wedge \neg v'_1 \vee v_0 \wedge \neg v_1 \wedge v'_1 \vee \neg v_0 \wedge v_1 \wedge \neg v'_0 \wedge \neg v'_1 \vee v_0 \wedge v_1 \wedge v'_0 \wedge v'_1$$

$$a(v_0, v_1) = \neg v_0 \vee \neg v_1$$

$$path_1(s_0, s_1, s_2) = S_0(s_0) \wedge R(s_0, s_1)$$

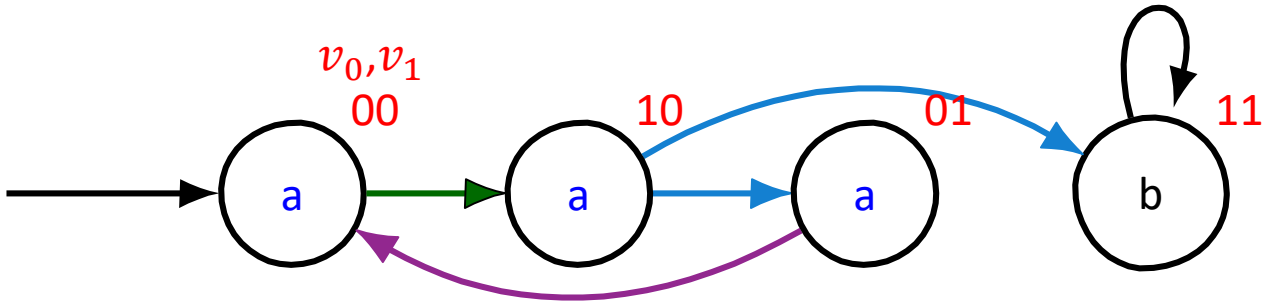
$$= \neg v_{0,0} \wedge \neg v_{1,0} \wedge$$

$$\left(\begin{array}{l} \neg v_{00} \wedge \neg v_{10} \wedge v_{01} \wedge \neg v_{11} \\ \vee v_{00} \wedge \neg v_{10} \wedge v_{11} \\ \vee \neg v_{00} \wedge v_{10} \wedge \neg v_{01} \wedge \neg v_{11} \\ \vee v_{00} \wedge v_{10} \wedge v_{01} \wedge v_{11} \end{array} \right)$$

v_{10} is variable v_1 in time 0

$$\bigvee_{i=0}^k \neg a(s_i) = \neg(\neg v_{00} \vee \neg v_{10}) \vee \neg(\neg v_{01} \vee \neg v_{11})$$

Example: k=2



$$S_0(v_0, v_1) = \neg v_0 \wedge \neg v_1$$

$$R(v_0, v_1, v'_0, v'_1) = \neg v_0 \wedge \neg v_1 \wedge v'_0 \wedge \neg v'_1 \vee v_0 \wedge \neg v_1 \wedge v'_1 \vee \neg v_0 \wedge v_1 \wedge \neg v'_0 \wedge \neg v'_1 \vee v_0 \wedge v_1 \wedge v'_0 \wedge v'_1$$

$$a(v_0, v_1) = \neg v_0 \vee \neg v_1$$

$$path_2(s_0, s_1, s_2) = S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2)$$

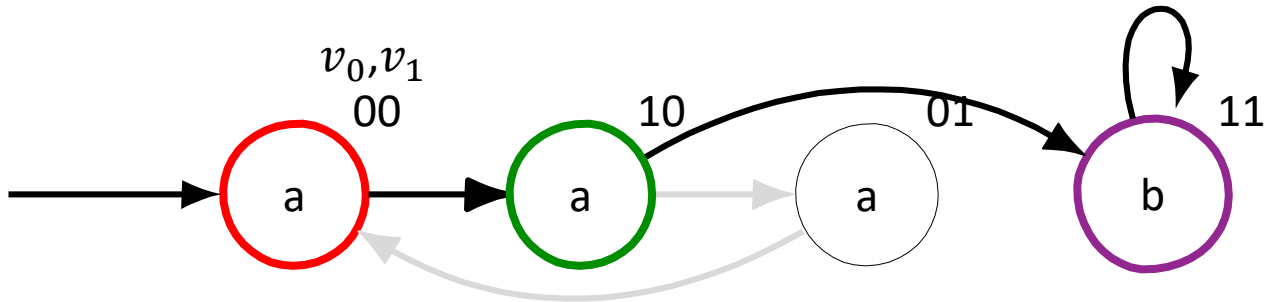
$$= \neg v_{0,0} \wedge \neg v_{1,0} \wedge$$

$$\left(\begin{array}{l} \neg v_{00} \wedge \neg v_{10} \wedge v_{01} \wedge \neg v_{11} \\ \vee v_{00} \wedge \neg v_{10} \wedge v_{11} \\ \vee \neg v_{00} \wedge v_{10} \wedge \neg v_{01} \wedge \neg v_{11} \\ \vee v_{00} \wedge v_{10} \wedge v_{01} \wedge v_{11} \end{array} \right) \wedge$$

$$\left(\begin{array}{l} \neg v_{01} \wedge \neg v_{11} \wedge v_{02} \wedge \neg v_{12} \\ \vee v_{01} \wedge \neg v_{11} \wedge v_{12} \\ \vee \neg v_{01} \wedge v_{11} \wedge \neg v_{02} \wedge \neg v_{12} \\ \vee v_{01} \wedge v_{11} \wedge v_{02} \wedge v_{12} \end{array} \right)$$

$$\bigvee_{i=0}^k \neg a(s_i) = \neg(\neg v_{00} \vee \neg v_{10}) \vee \neg(\neg v_{01} \vee \neg v_{11}) \vee \neg(\neg v_{02} \vee \neg v_{12})$$

Example



AG a

$$\begin{aligned}
 path_2(s_0, s_1, s_2) &= S_0(s_0) \wedge R(s_0, s_1) \wedge R(s_1, s_2) \\
 &= \neg v_{0,0} \wedge \neg v_{1,0} \wedge \\
 &\quad \left(\begin{array}{l} \neg v_{00} \wedge \neg v_{10} \wedge v_{01} \wedge \neg v_{11} \\ \vee v_{00} \wedge \neg v_{10} \wedge v_{11} \\ \vee \neg v_{00} \wedge v_{10} \wedge \neg v_{01} \wedge \neg v_{11} \\ \vee v_{00} \wedge v_{10} \wedge v_{01} \wedge v_{11} \end{array} \right) \wedge \\
 &\quad \left(\begin{array}{l} \neg v_{01} \wedge \neg v_{11} \wedge v_{02} \wedge \neg v_{12} \\ \vee v_{01} \wedge \neg v_{11} \wedge v_{12} \\ \vee \neg v_{01} \wedge v_{11} \wedge \neg v_{02} \wedge \neg v_{12} \\ \vee v_{01} \wedge v_{11} \wedge v_{02} \wedge v_{12} \end{array} \right)
 \end{aligned}$$

Satisfying assignment

i	0	1	2
v_{0i}	0	1	1
v_{1i}	0	0	1

$$\bigvee_{i=0}^k \neg a(s_i) = \neg(\neg v_{00} \vee \neg v_{10}) \vee \neg(\neg v_{01} \vee \neg v_{11}) \vee \neg(\neg v_{02} \vee \neg v_{12})$$

Try it with Z3!

```
(declare-const v00 Bool)
(declare-const v10 Bool)
(declare-const v01 Bool)
(declare-const v11 Bool)
(declare-const v02 Bool)
(declare-const v12 Bool)

(define-fun S0 ((v0 Bool) (v1 Bool)) Bool
  (and (not v0) (not v1)))

(define-fun R ((v0 Bool) (v1 Bool) (w0 Bool) (w1 Bool))
  Bool
  (or
   (and (not v0) (not v1) w0 (not w1))
   (and v0 (not v1) w1)
   (and (not v0) v1 (not w0) (not w1))
   (and v0 v1 w0 w1))
  )

(define-fun a ((v0 Bool) (v1 Bool)) Bool
  (or (not v0) (not v1))
  )

(define-fun path1 ((v00 Bool) (v10 Bool) (v01 Bool) (v11
  Bool)) Bool
  (and (S0 v00 v10)
   (R v00 v10 v01 v11))
  )
)
```

```
(define-fun path2 ((v00 Bool) (v10 Bool) (v01 Bool) (v11
  Bool) (v02 Bool) (v12 Bool)) Bool
  (and (S0 v00 v10)
   (R v00 v10 v01 v11)
   (R v01 v11 v02 v12))
  )
)

; k = 1
; (assert
;   (and (path1 v00 v10 v01 v11)
;        (or (not (a v00 v10))
;            (not (a v01 v11))
;            )
;   )
;)

; k = 2
; (assert
;   (and (path2 v00 v10 v01 v11 v02 v12)
;        (or (not (a v00 v10))
;            (not (a v01 v11))
;            (not (a v02 v12))
;            )
;   )
;)

(check-sat)
(get-model)
```

<https://jfmcc.github.io/z3-play/>

<https://rise4fun.com/Z3> or

<https://compsys-tools.ens-lyon.fr/z3/index.php>

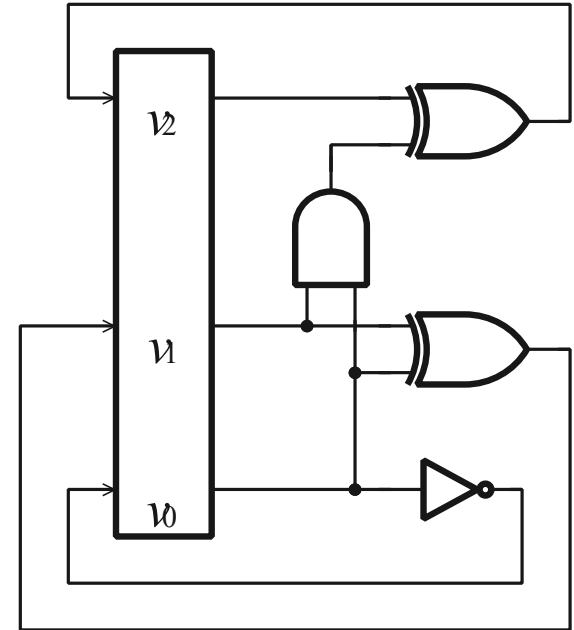
Another Example

Does modulo-8 counter of Chapter 3.5 ever reach 4?

$$\neg p = \neg v_0 \wedge \neg v_1 \wedge v_2.$$

$$S_0(V) = \neg v_0 \wedge \neg v_1 \wedge \neg v_2.$$

$$R(V, V') = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \oplus v_1) \wedge (v'_2 \leftrightarrow (v_0 \wedge v_1) \oplus v_2).$$



$k = 0$:

$$S_0(v) \quad \neg v_0 \wedge \neg v_1 \wedge \neg v_2 \wedge$$

$$\neg p(v) \quad \neg v_0 \wedge \neg v_1 \wedge v_2.$$

Another Example

Does module-8 counter of Chapter 3.5 ever reach 4?

$$\neg p = \neg v_0 \wedge \neg v_1 \wedge v_2.$$

$$S_0(V) = \neg v_0 \wedge \neg v_1 \wedge \neg v_2.$$

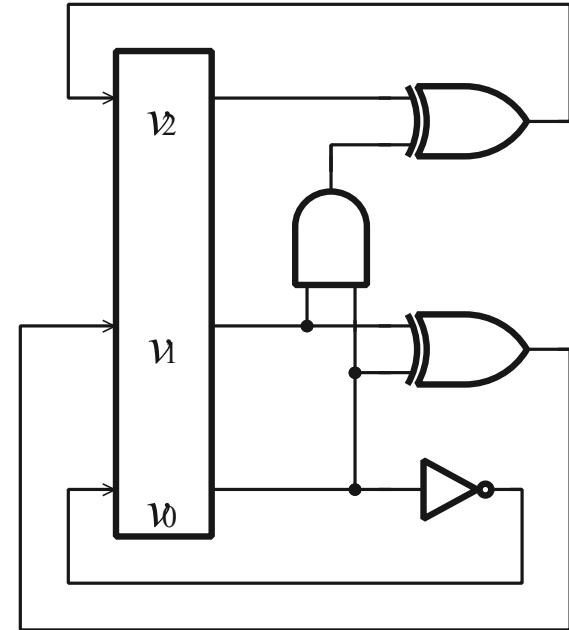
$$R(V, V') = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \oplus v_1) \wedge (v'_2 \leftrightarrow (v_0 \wedge v_1) \oplus v_2).$$

$k = 1$:

$$S_0(v) \quad \neg v_0 \wedge \neg v_1 \wedge \neg v_2 \wedge$$

$$R(v, v') \quad (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \oplus v_1) \wedge (v'_2 \leftrightarrow (v_0 \wedge v_1) \oplus v_2) \wedge$$

$$(\underbrace{\neg v_0 \wedge \neg v_1 \wedge v_2}_{\neg p(v)} \vee \underbrace{\neg v'_0 \wedge \neg v'_1 \wedge v'_2}_{\neg p(v')}).$$



Another Example

Does module-8 counter of Chapter 3.5 ever reach 4?

$$\neg p = \neg v_0 \wedge \neg v_1 \wedge v_2.$$

$$S_0(V) = \neg v_0 \wedge \neg v_1 \wedge \neg v_2.$$

$$R(V, V') = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \oplus v_1) \wedge (v'_2 \leftrightarrow (v_0 \wedge v_1) \oplus v_2).$$

$k = 2$:

$$S_0(v) \quad \neg v_0 \wedge \neg v_1 \wedge \neg v_2 \wedge$$

$$R(v, v') \quad (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \oplus v_1) \wedge (v'_2 \leftrightarrow (v_0 \wedge v_1) \oplus v_2) \wedge$$

$$R(v', v'') \quad (v''_0 \leftrightarrow \neg v'_0) \wedge (v''_1 \leftrightarrow v'_0 \oplus v'_1) \wedge (v''_2 \leftrightarrow (v'_0 \wedge v'_1) \oplus v'_2) \wedge$$

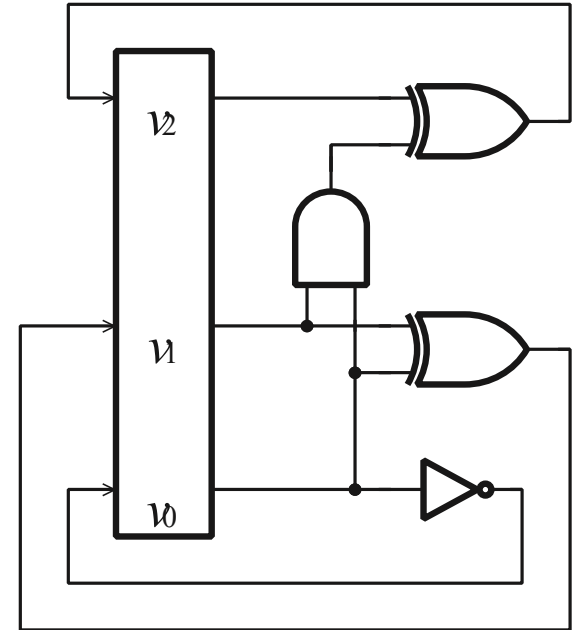
$$(\neg v_0 \wedge \neg v_1 \wedge v_2 \vee \neg v'_0 \wedge \neg v'_1 \wedge v'_2 \vee \neg v''_0 \wedge \neg v''_1 \wedge v''_2.)$$

$\neg p(v)$

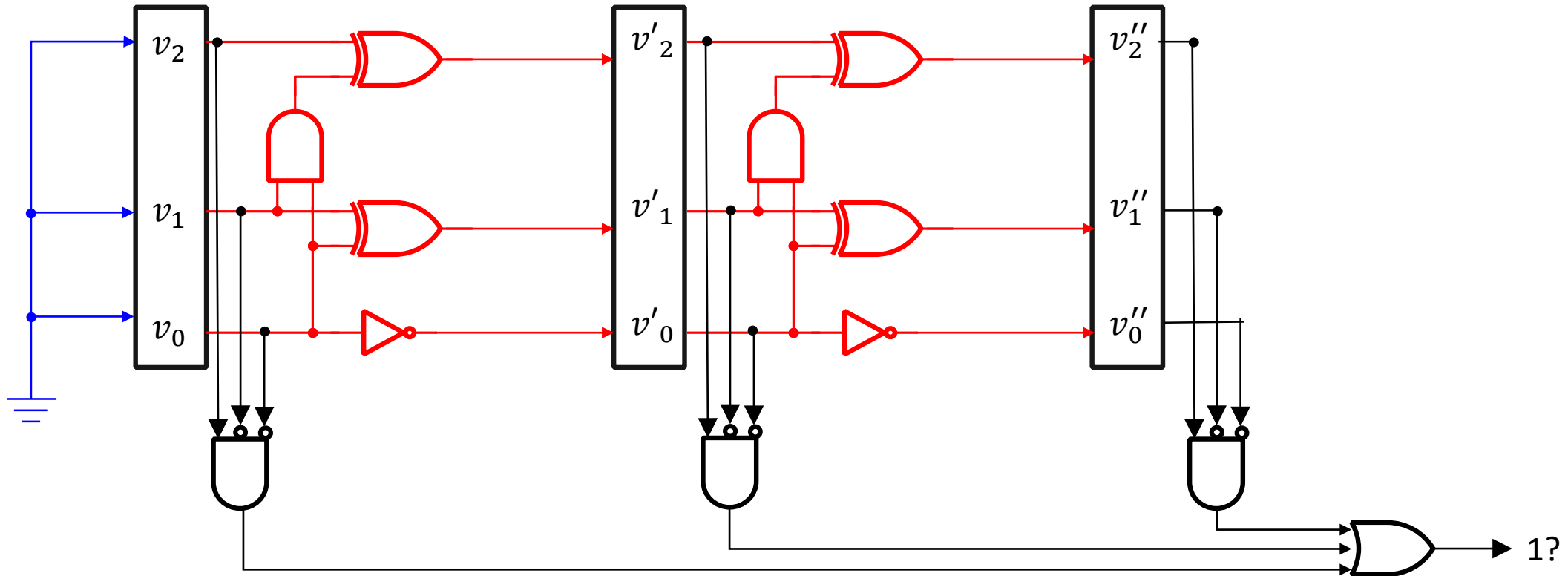
$\neg p(v')$

Model Checking

$\neg p(v'')$



The Electrical Engineer's Viewpoint



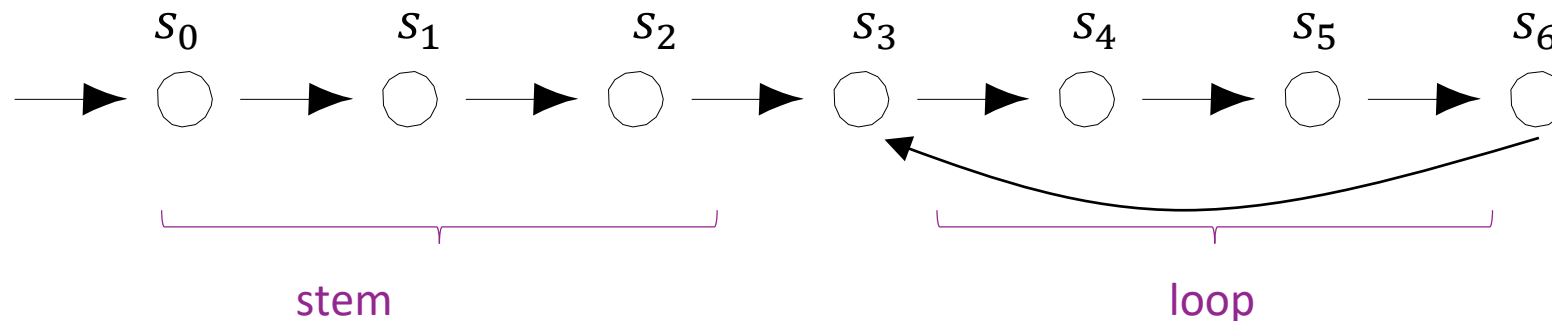
Eventuality Properties

Property

Suppose $\phi = \mathbf{AF} p$

Counterexamples fulfill $\mathbf{EG} \neg p$

Counterexamples have *Lasso Shape* (See Chapter 4.)



Completeness

BMC finds bugs of length $\leq k$.

Longer counterexamples may exist!

Is there a k big enough to exclude any counterexample?

Def. $M \models_k \phi$: all computations of length k satisfy ϕ .

Completeness threshold: Number CT such that $M \models_{CT} \phi \Rightarrow M \models \phi$

If completeness threshold known, stop BMC when $k = CT$

Ideas for finding CT ?

Completeness Threshold

Finding smallest CT is as difficult as model checking!

- Smallest CT is size of shortest counterexample, or 0 if the property is satisfied

$$New(V_0, \dots, V_k) = S_0(V_0) \wedge \bigwedge_{i=0}^{k-1} (R(V_i, V_{i+1})) \wedge \bigwedge_{j < i} V_i \neq V_j$$

Simple values for CT

- Number of state of M is bound on CT
- Diameter (longest simple path between two states) is bound on CT

These are typically really large numbers

HOMEWORK

Verifying Reachability Properties with k -induction



Mary Sheeran, Koen Claessen, Per Bjesse,
2000

Motivation

- Completeness thresholds usually very large
- Can we **prove** a property with fewer unrollings?
- **Idea: Use induction.**

Base: Prove $Q(0)$

Induction: Prove $Q(n - 1) \Rightarrow Q(n)$

Conclusion: $\forall n. Q(n)$

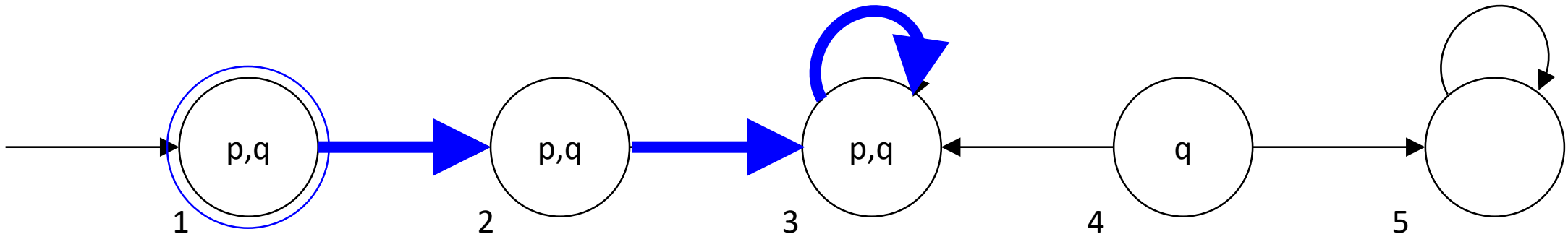
Caveat: Property may be true, but not inductive (see below)

Induction

Let's prove **AG** p on the following structure.

Take arbitrary path π

- **Base case:** $\pi(0) \models p$ true: $q_1 \models p$
- **Induction:** if $\pi(n - 1) \models p$ then $\pi(n) \models p$ true: any successor of a p -state is a p -state
- **Conclusion:** for any path π we have $\forall n. \pi(n) \models p$

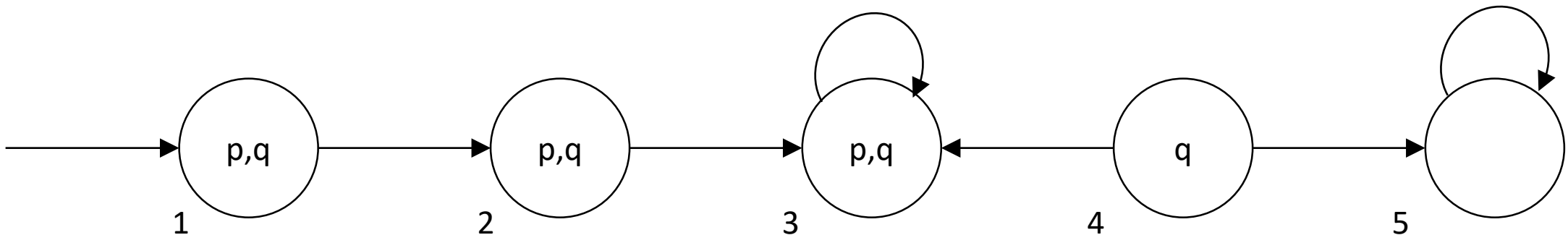


Satisfiability

Let's prove $AG\ p$ on the following structure. How can these properties be violated?

Take arbitrary path π

- **Base case:** $\pi(0) \models p$ $S_0(s) \wedge \neg p(s)$ Unsatisfiable
- **Induction:** if $\pi(n - 1) \models p$ then $\pi(n) \models p$ $p(s) \wedge R(s, s') \wedge \neg p(s')$ Unsatisfiable
- **Conclusion:** for any path π we have $\forall n. \pi(n) \models p$

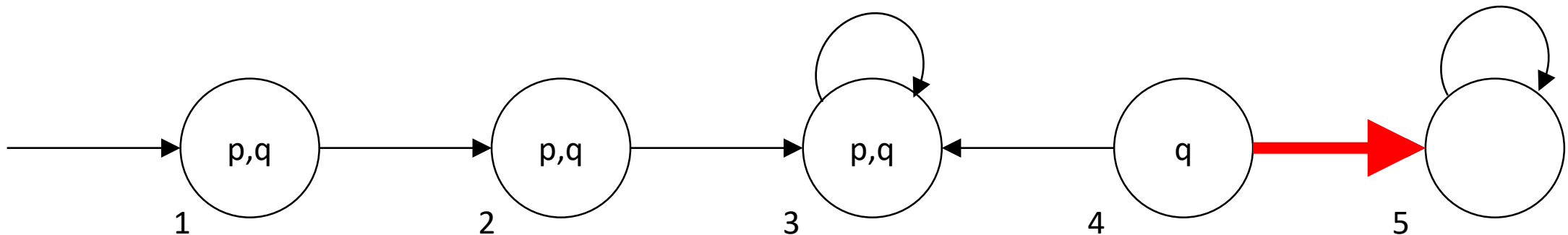


A Problem

Let's prove **AG** q on the following structure.

Take arbitrary path π

- **Base case:** $\pi(0) \models q$
- **Induction:** if $\pi(n - 1) \models q$ then $\pi(n) \models q$ **not true!**
- ~~**Conclusion:** for any path π we have $\forall n. \pi(n) \models q$~~ **not all true properties are inductive**



k -induction

Base:

Induction:

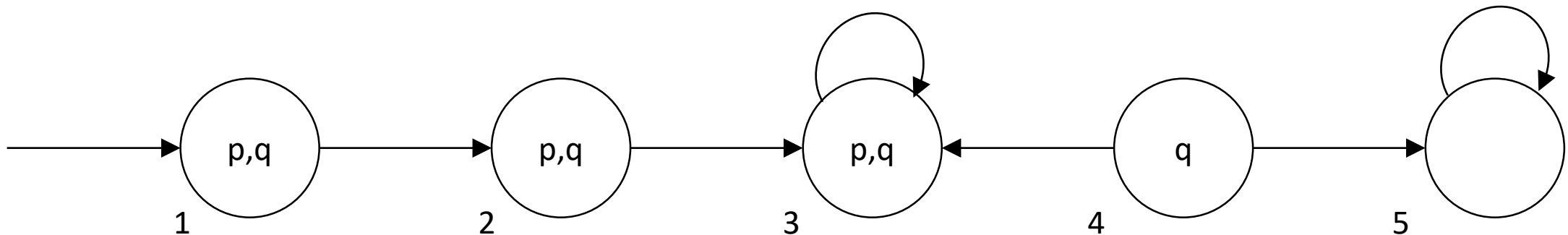
Conclusion: $\forall n. Q(n)$

In our setting:

Base. all paths of length k from S_0 are labeled q

Induction. all paths of length k labeled with all qs are followed by a q

Conclusion. All paths from S_0 are labeled q



k -induction

Base: Prove $Q(0) \wedge \dots \wedge Q(k - 1)$

Induction: Prove $Q(n - k) \wedge \dots \wedge Q(n - 1) \Rightarrow Q(n)$

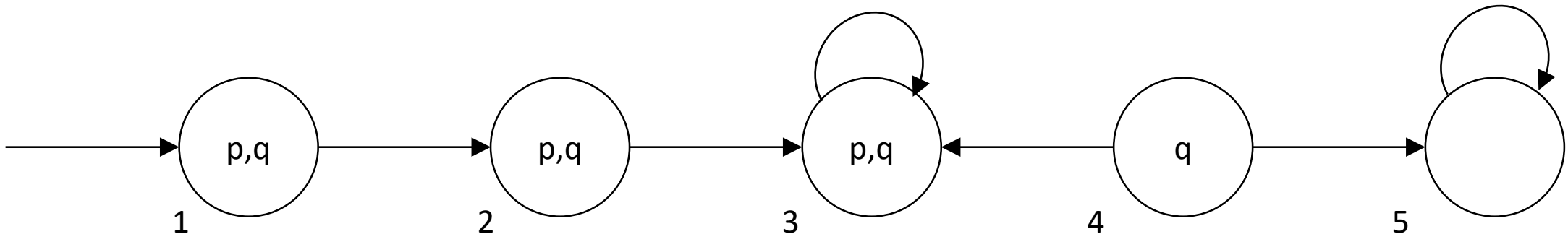
Conclusion: $\forall n. Q(n)$

In our setting:

Base. all paths of length k from S_0 are labeled q

Induction. all paths of length k labeled with all qs are followed by a q

Conclusion. All paths from S_0 are labeled q



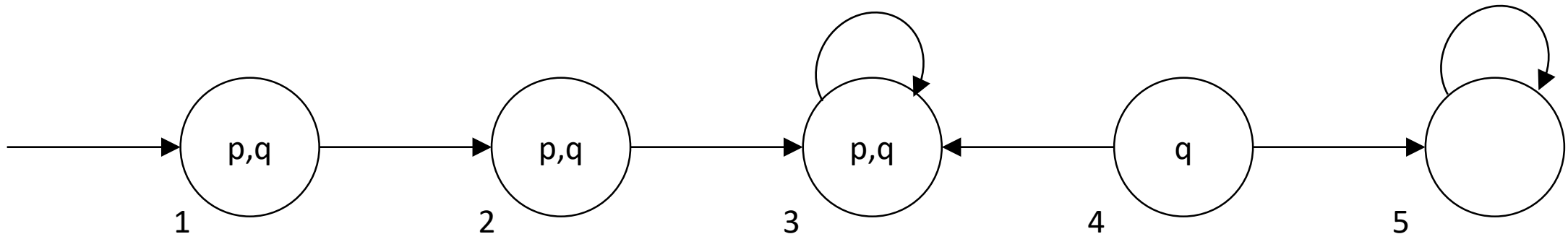
Prove $AG\ q$ using 2-induction

Base: Consider all paths of length 2 from q_1 : $q_1 \models q$ and $q_2 \models q$.

Induction: Do all successors of paths of length 2 labeled q, q fulfill q ?

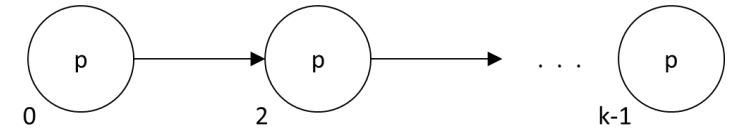
- (q_1, q_2)
- (q_2, q_3)
- (q_3, q_4)
- (q_4, q_4)

Conclusion: for any path π we have $\forall n. \pi(n) \models p$



k-induction as Satisfiability

Base. all paths of length k from S_0 are labeled p



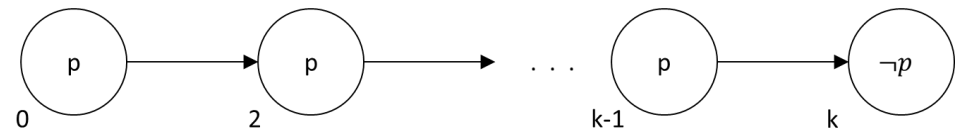
This is BMC!

$$S_0(s_0) \wedge \bigwedge_{i=0}^{k-2} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k-1} \neg p(s_i)$$

Induction. all paths of length k labeled with all p 's are followed by a p

$$\bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=0}^{k-1} p(s_i) \wedge \neg p(s_k)$$

Formula satisfiable iff there is a counterexample



k-induction

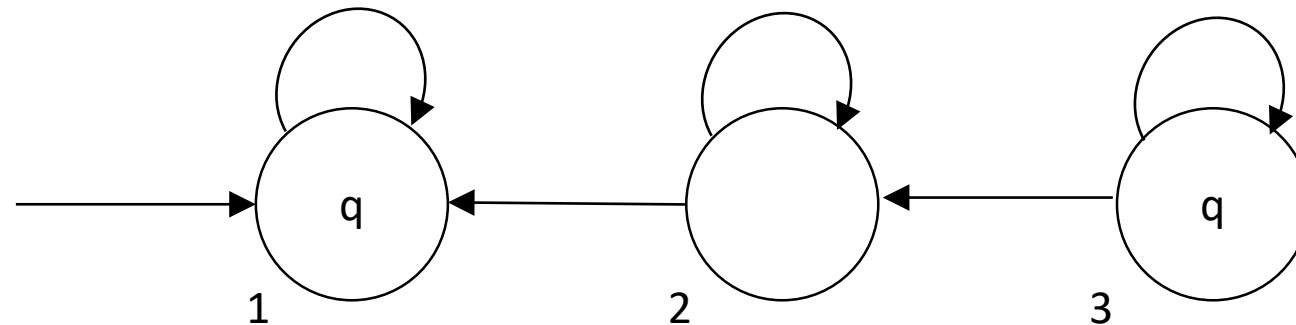
```
while(k=0; ; k++){  
    build BMC formula  $\phi$   
    if  $\phi$  SAT return “bug!”  
  
    build induction formula  $\psi$   
    if  $\phi$  UNSAT return “correct!”  
}
```


k-induction is not Complete

System satisfies **AG** q , but induction step fails for any k

Base. all paths of length k from S_0 are labeled q

Induction. all paths of length k labeled with all qs are followed by a q . **FALSE**

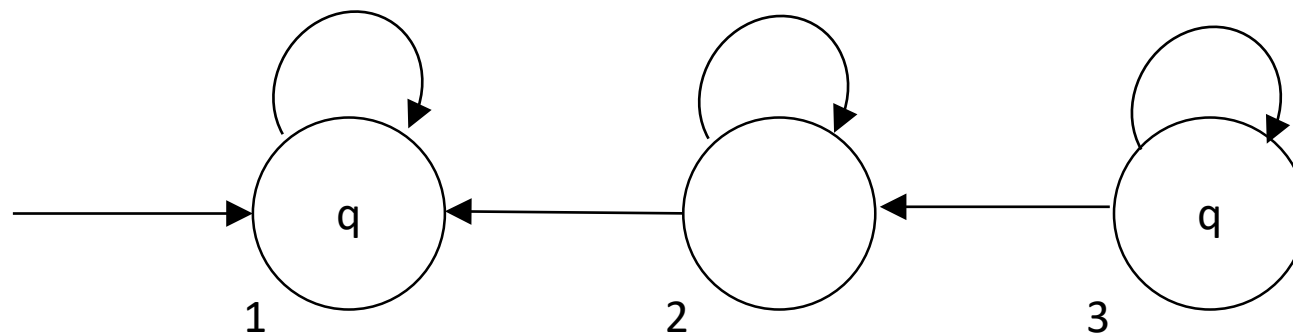


Making k-induction Complete

Induction. all **noncyclic** paths of length k labeled with all qs are followed by a q

$$\bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=0}^{k-1} p(s_i) \wedge \neg p(s_k) \wedge$$

$$\bigwedge_{i=0}^{k-1} \bigwedge_{j=i+1}^k s_i \neq s_j$$

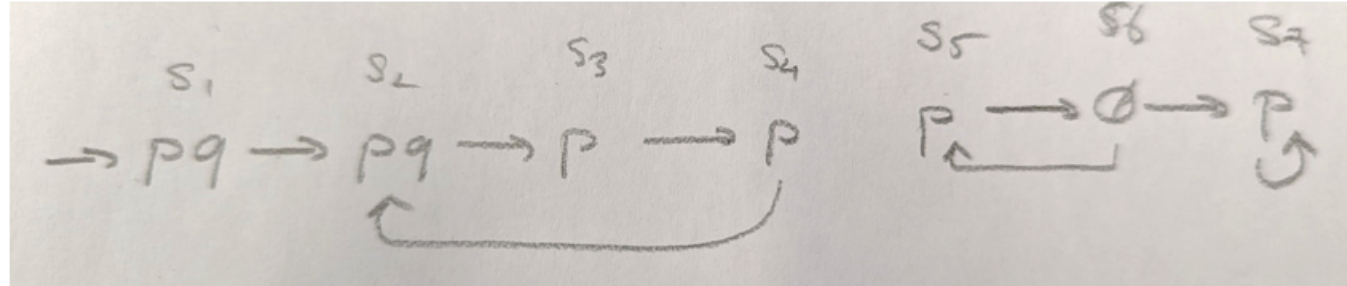


Model Checking Homework 2

Deadline: 24 March 2:00pm

Send solution to: modelchecking@iaik.tugraz.at

Consider the following Kripke structure K with states s_1 through s_7 and atomic propositions p and q .



Task 2a. [5 points]

We want to use BMC to prove whether q is always true. Suppose you are given formulas R , S_0 , and p for the transition relation, the initial states and the property, resp.

- Will BMC find a counterexample? If so, what is the smallest k such that BMC finds a counterexample?
- Show the BMC formula, using R , S_0 , and p . (You don't have to find R , S_0 , and p .)
- Is the formula satisfiable? Explain.

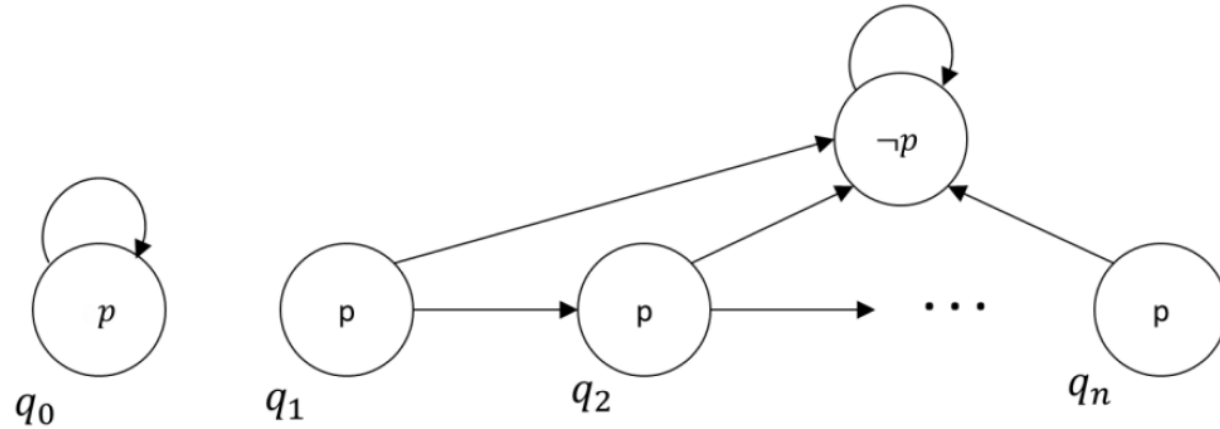
Task 2c. [5 points]

We want to use k -induction to prove that p is always true

- What is the smallest k such that k -induction can prove the property correct?
- Show the base and the induction formula for k -induction, using R , S'_0 , and p for the k you found.
- Is the formula satisfiable? Explain.

Consider the following synchronous Kripke structure K .

HW2



We wish to prove that p is always true.

Task 2a. [5 points]

Suppose that q_1 is the initial state. Suppose you are given formulas R , S_0 , and p for the transition relation, the initial states and the property, resp.

- What is the smallest k such that BMC finds a counterexample?
- Show the BMC formula, using R , S_0 , and p .
- Is the formula satisfiable? Explain.

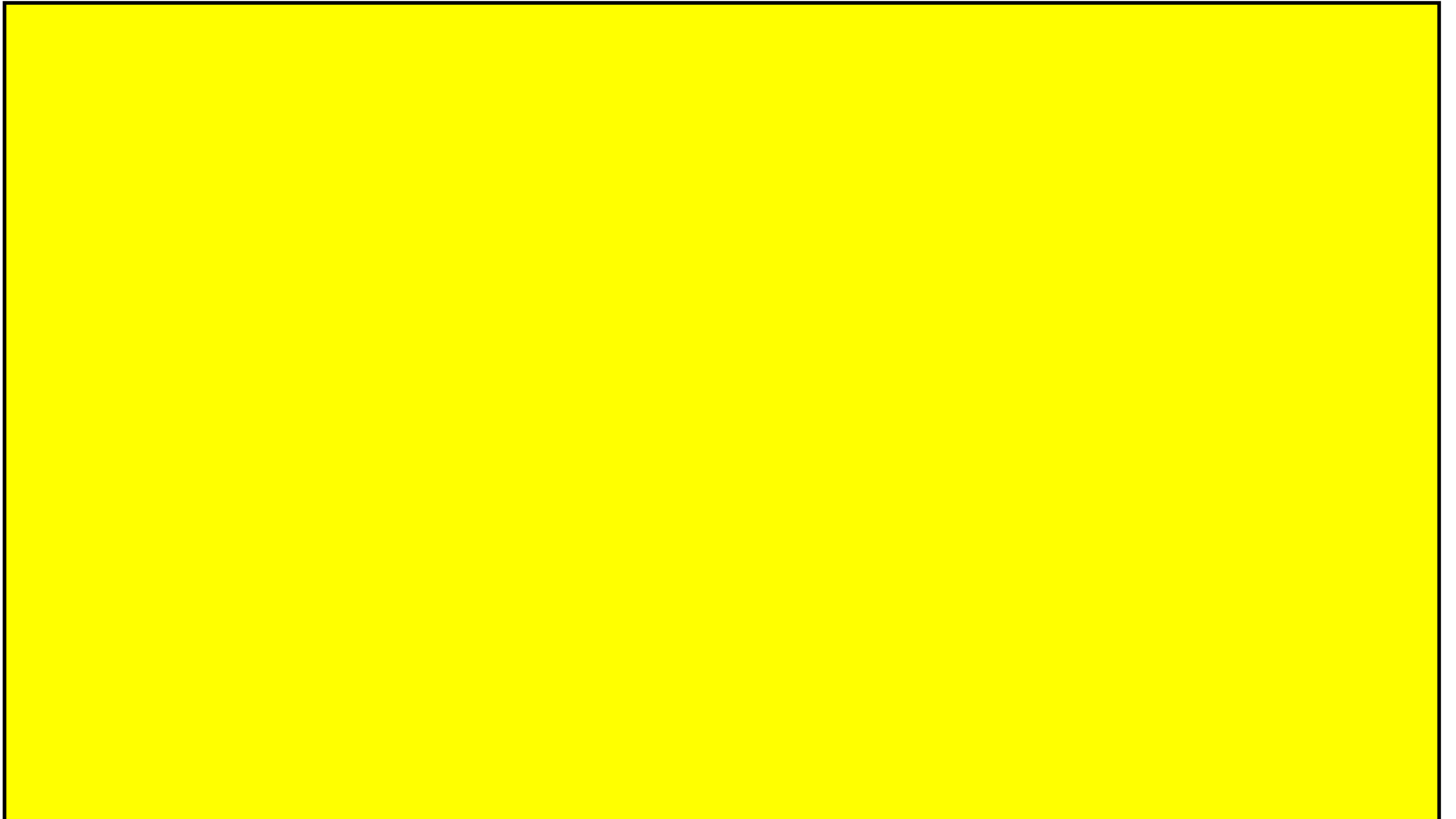
Task 2c. [5 points]

Suppose that q_0 is the initial state. The new formula for the initial states is S'_0 .

- What is the smallest k such that k -induction can prove the property correct?
- Suppose $n=2$. Choose an appropriate k and show the k -induction formula, using R , S'_0 , and p .
- Is the formula satisfiable? Explain.

Note to myself

- There is a BASE formula and an INDUCTION formula. You need both to do k-induction. That was unclear in the home work.h



Model Checking with Inductive Invariants

Homework

Find results here:

<https://cloud.tugraz.at/index.php/s/zeEgt8ptcRQCXEW>

Grades are pseudonymized, your code is the sha256 hash of your matriculation number

Questions? Ask in

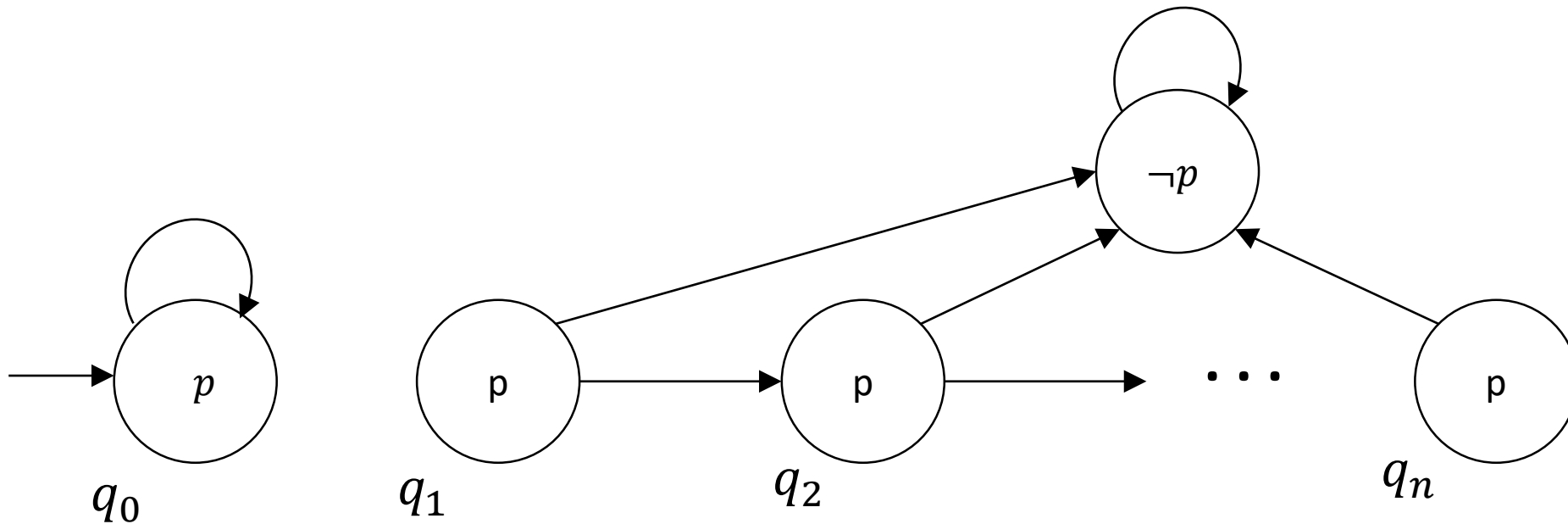
- discord #lecture,
- discord FilipCC, or
- write an email to modelchecking@iaik.

Problems with k -induction

Problem: Sometimes k is very large

In the following machine, you need $k = n + 1$ to prove $\mathbf{AG} p$.

Idea: Automatically find better inductive invariants.



Inductive Invariant

Remember $postimage(Q) = \{s' \mid \exists s. R(s, s')\}$ (see Chapter 5).

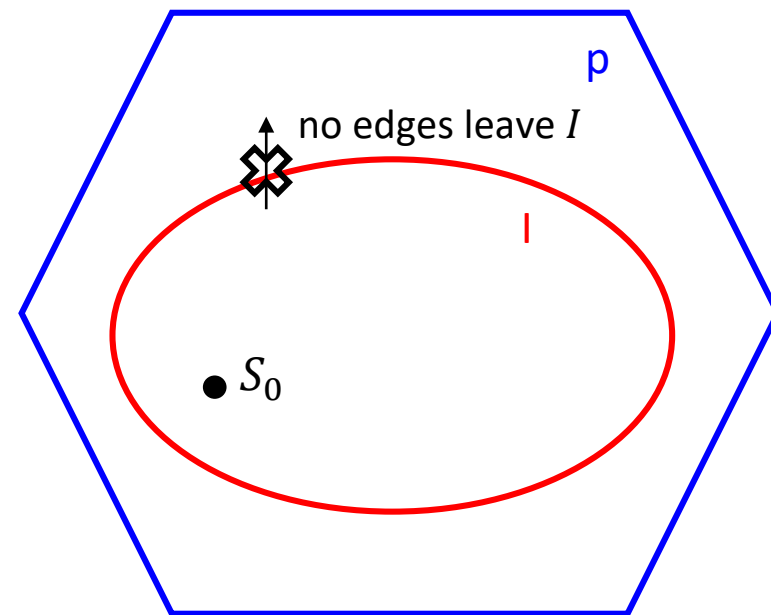
Definition. $I \subseteq S$ is an **inductive invariant** for $AG\ p$ if

1. $S_0 \subseteq I$
2. $postimage(I) \subseteq I$
3. $\forall s \in I. s \models p$

If there is an inductive invariant for $AG\ p$, then $AG\ p$ holds.

In formulas:

1. $S_0 \rightarrow I$
2. $I \wedge R \rightarrow I'$
3. $I \rightarrow p$



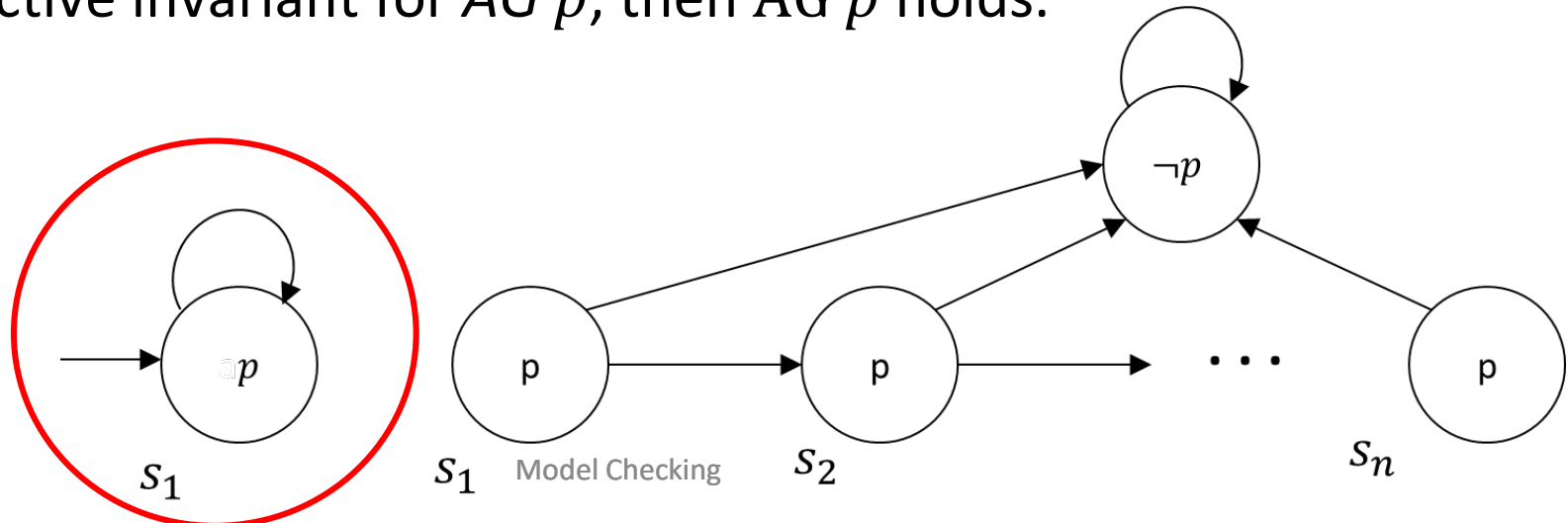
Inductive Invariant

Remember $postimage(Q) = \{s' \mid \exists s. R(s, s')\}$ (see Chapter 5).

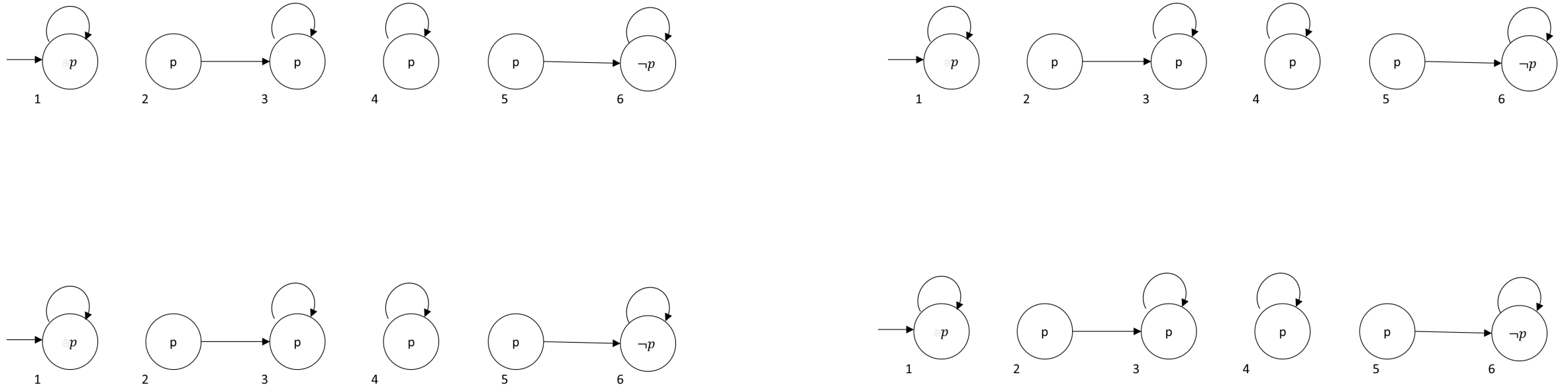
Definition. $I \subseteq S$ is an **inductive invariant** for $AG\ p$ if

1. $S_0 \subseteq I$
2. $postimage(I) \subseteq I$
3. $\forall s \in I. s \models p$

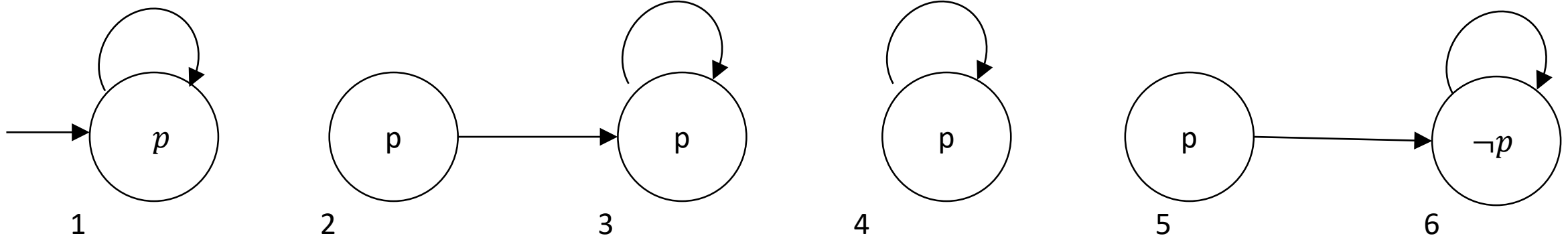
If there is an inductive invariant for $AG\ p$, then $AG\ p$ holds.



Multiple Invariants



Strongest & Weakest Invariant



Smallest (strongest) invariant is reachable state

Largest (weakest) invariant is states that cannot reach $\neg p$

Model Checking with Craig Interpolants

Ken McMillan, 2003

2010 CAV Award: “has significantly influenced both academic research and industrial practice, and has dramatically changed the scale of systems that can be analyzed by model checking.”



Kenneth McMillan

Interpolants as Inductive Invariants

- BMC finds bugs (and absence of bugs up to k steps)
- How to Show Correctness?
 - k -induction
 - Interpolants
- Find Interpolants I such that
 - States reachable in k steps are in I
 - no bad states are in I
- Interpolants are (good) overapproximation of post-image computation

Interpolant



William Craig, 1957

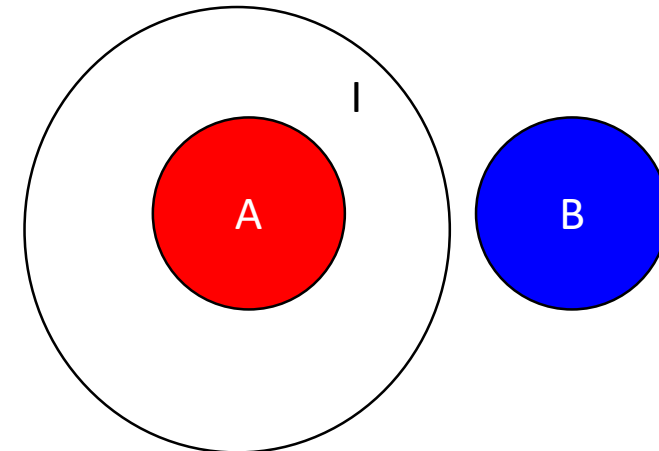
Definition. Given formulas A, B such that $A \wedge B = \perp$, an **interpolant** is a formula I such that

1. $A \rightarrow I$
2. $I \wedge B \equiv \perp$
3. I only uses symbols that occur both in A and in B

Example. Let

$$A = (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2,$$

$$B = (\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4.$$



Interpolant



William Craig, 1957

Definition. Given formulas A, B such that $A \wedge B = \perp$, an **interpolant** is a formula I such that

1. $A \rightarrow I$
2. $I \wedge B \equiv \perp$
3. I only uses symbols that occur both in A and in B

Example. Let

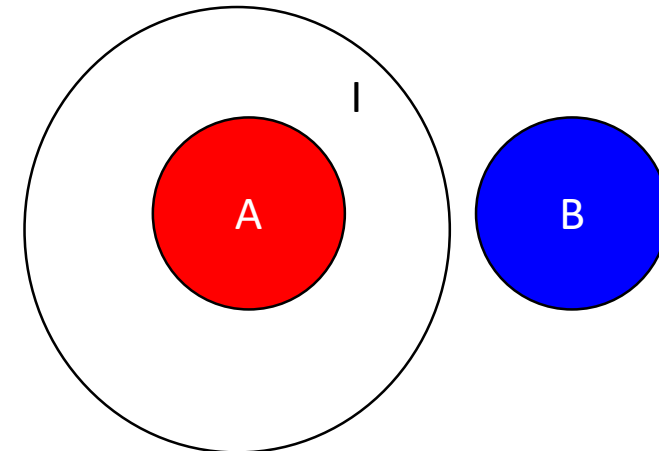
$$A = (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2,$$

$$B = (\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4.$$

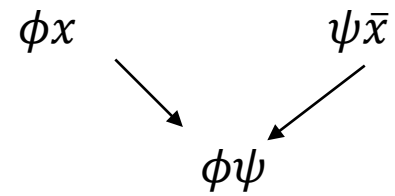
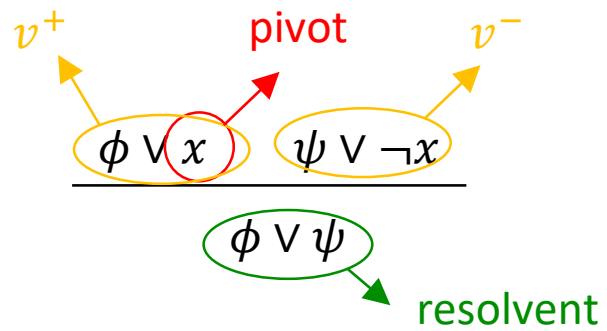
$A \wedge B$ is not satisfiable.

$\neg a_3 \wedge a_2$ is an interpolant:

1. $((a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2) \rightarrow (\neg a_3 \wedge a_2)$
2. $(\neg a_3 \wedge a_2) \wedge ((\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4) \equiv \perp$
3. a_2 and a_3 occur in A and in B



Resolution (Chap 9)



Interpolants from Resolution Proofs

For clause C , $C|B$ is obtained by removing literals not in B

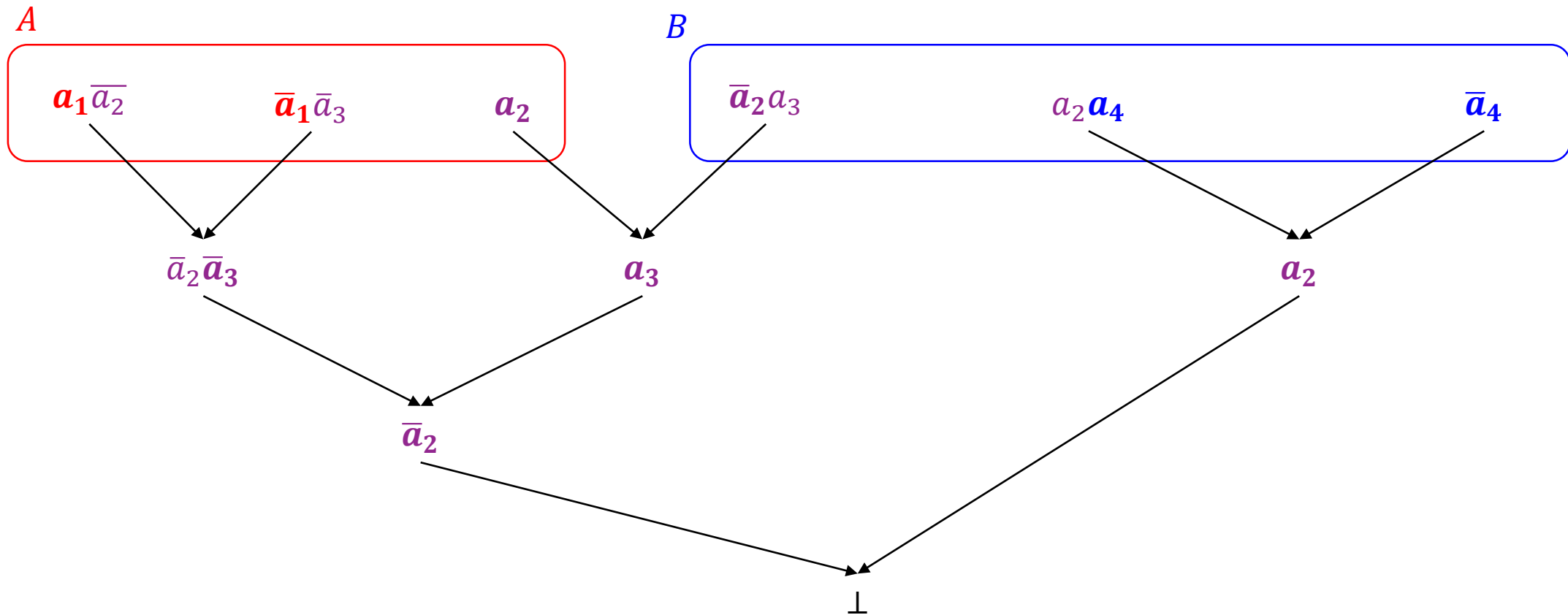
Algorithm. Go through resolution proof top-down.

1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$

Interpolation Example

Algorithm. Go through resolution proof top-down.

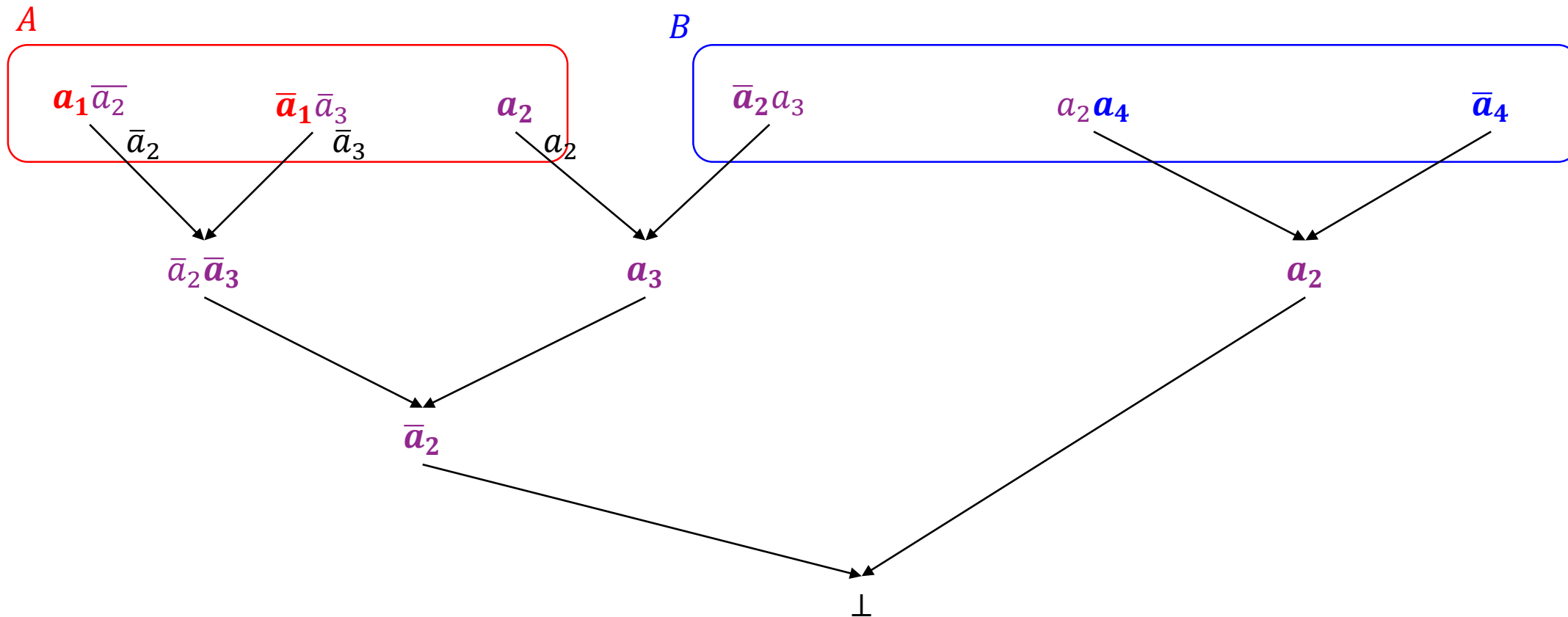
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

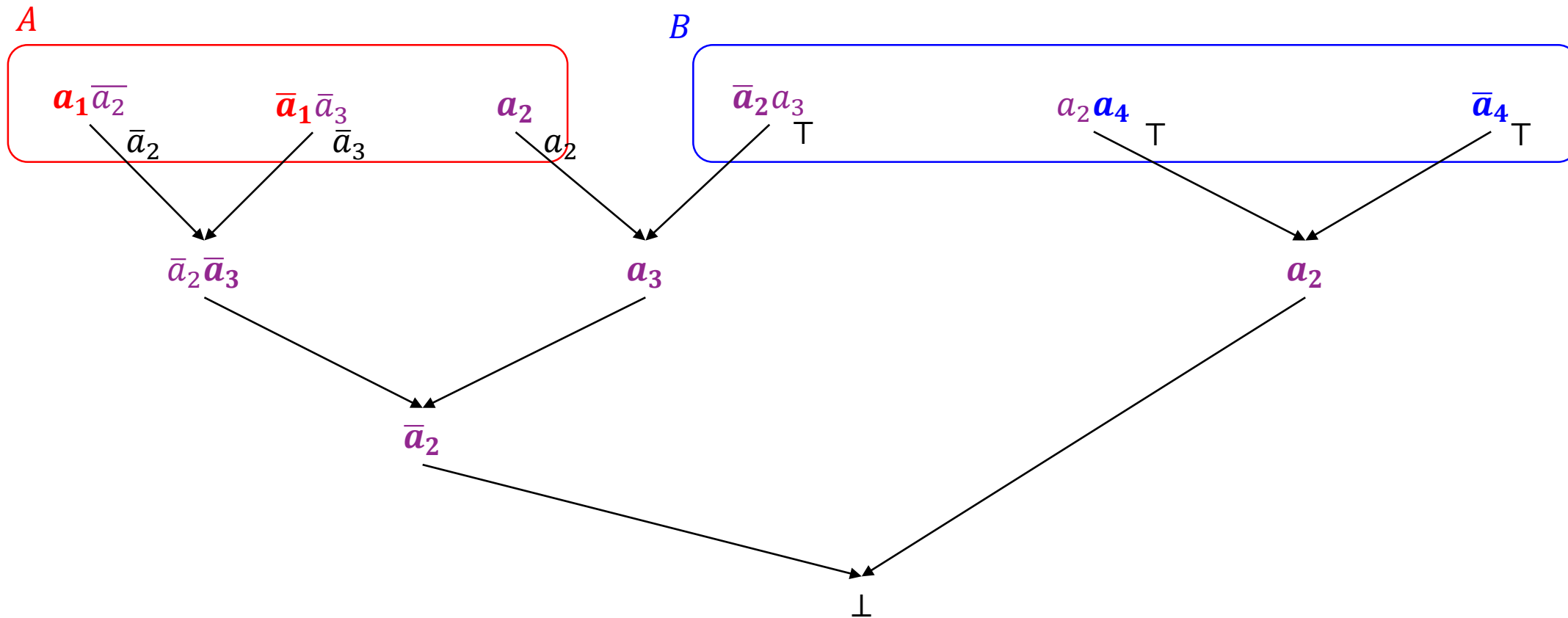
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

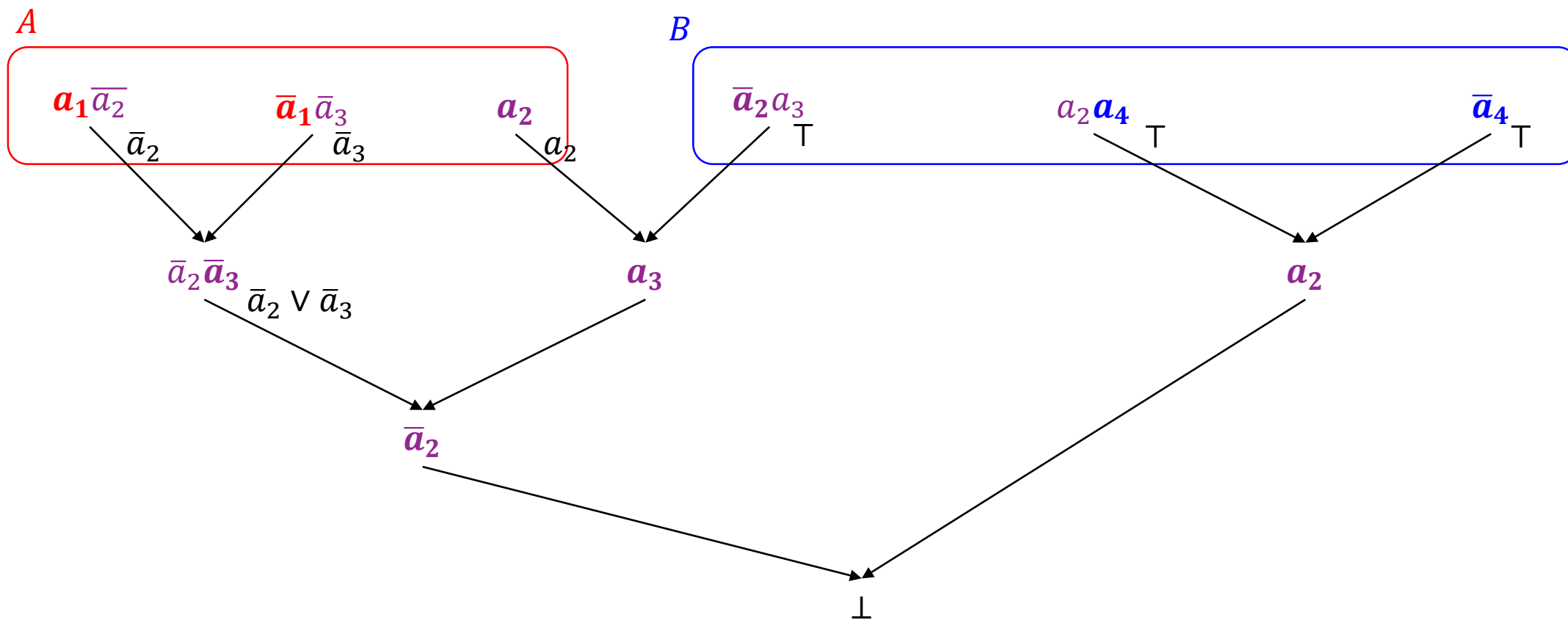
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

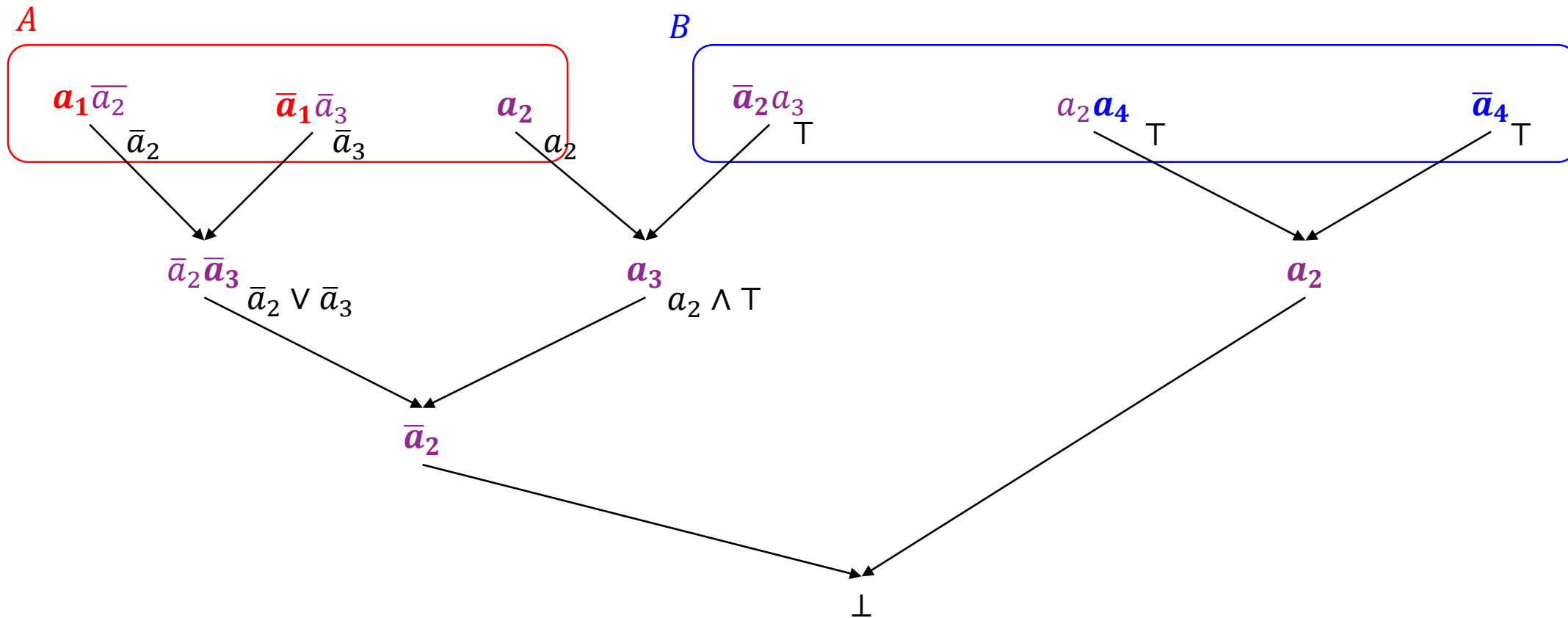
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

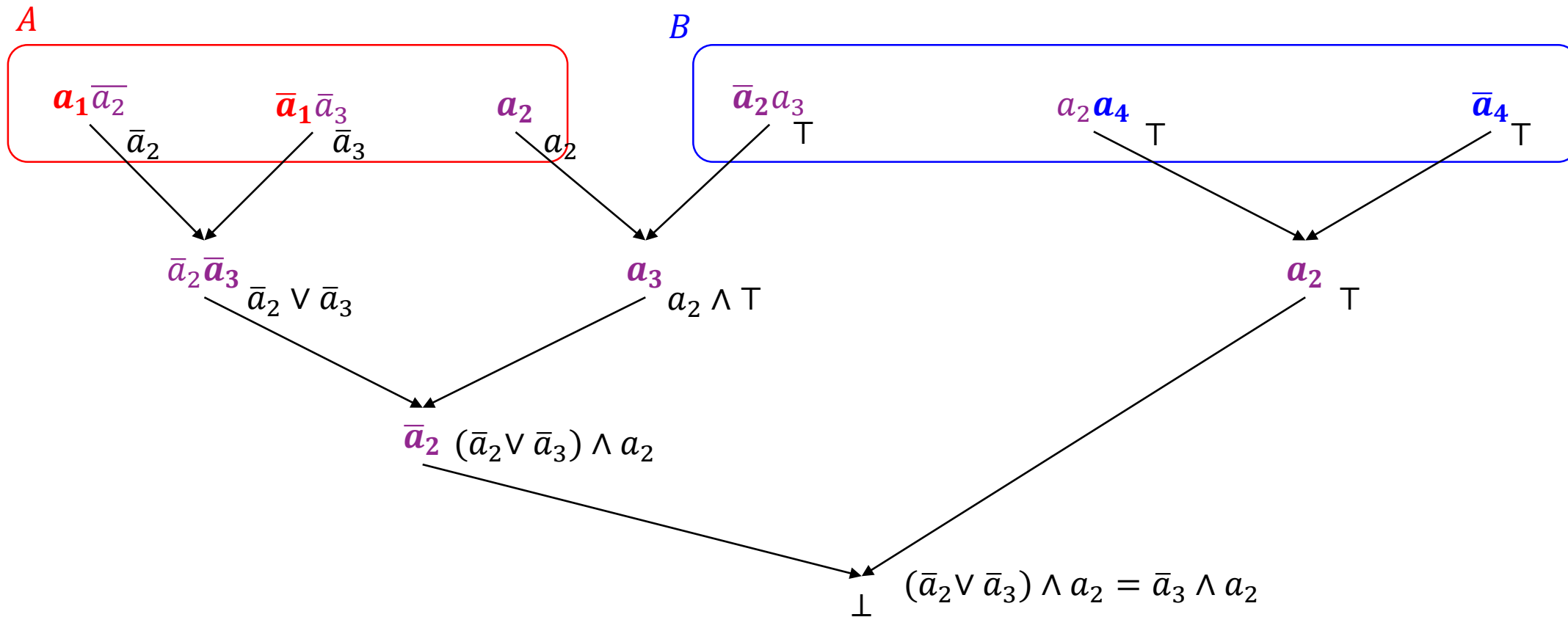
1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Interpolation Example

Algorithm. Go through resolution proof top-down.

1. If leaf v is labeled $C \in A$, then $Itp(v) = C|B$
2. If leaf v is labeled $C \in B$, then $Itp(v) = \top$
3. If node v has pivot variable $x \in B$ then $Itp(v) = Itp(v^+) \wedge Itp(v^-)$
4. If node v has pivot variable $x \notin B$ then $Itp(v) = Itp(v^+) \vee Itp(v^-)$



Model Checking with Interpolations

Computation with Overapproximations

BMC to prove p : $S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^k \neg p(s_i)$.

Suppose $R \rightarrow R'$

What does this do?

$$S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R'(s_i, s_{i+1}) \wedge \bigvee_{i=0}^k \neg p(s_i)$$

Idea 1:

if $S_0 \wedge \neg p$ is SAT return " $M \not\models AG p$ ";

$Q := S_0(s_0)$;

while true do

$A := Q(s_0) \wedge R(s_0, s_1)$;

$B =: \neg p$;

if $A \wedge B$ is SAT **then**

 ...

else

 compute interpolant I for A and B ;

 If $I == Q$ then return " $M \models AG p$ ";

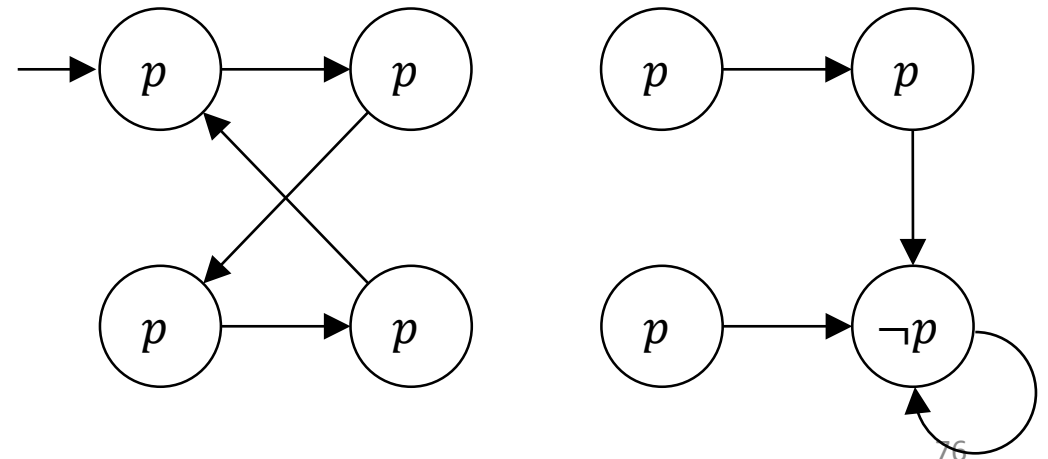
$Q := Q \vee I$;

end if

end while

What could possibly go wrong?

And what do we do against it?



Reachability Checking with Interpolation

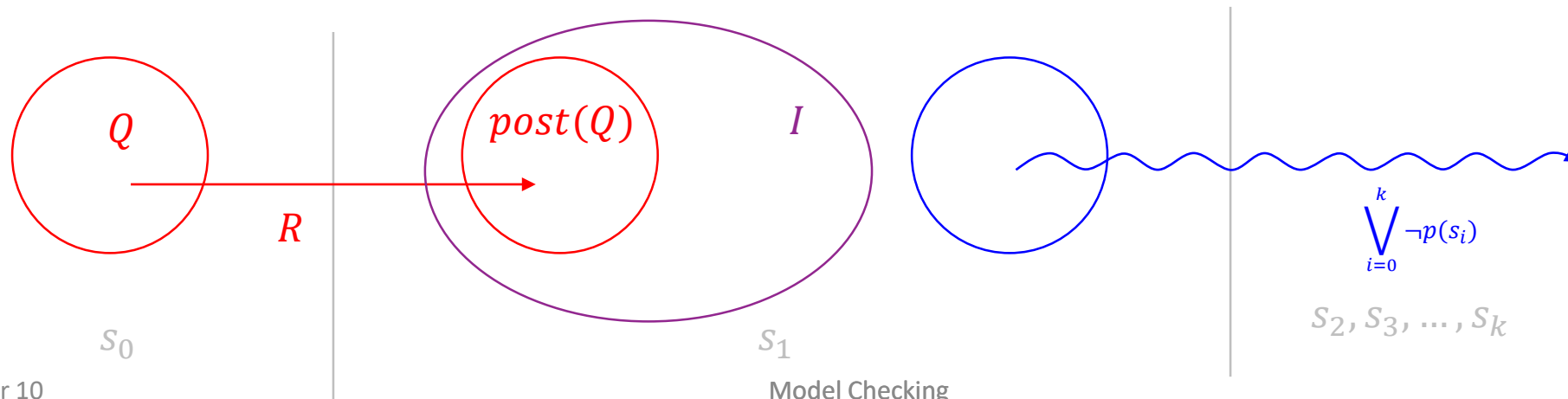
Recall BMC check for $\neg \mathbf{AG}p$:

$$S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^k \neg p(s_i).$$

Instead, start from Q such that $Q \models p$

$$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i).$$

Suppose ϕ unsatisfiable, $I(s_1)$ is an interpolant



Reachability Checking with Interpolation

Recall BMC check for $\neg \mathbf{AG} p$:

$$S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^k \neg p(s_i).$$

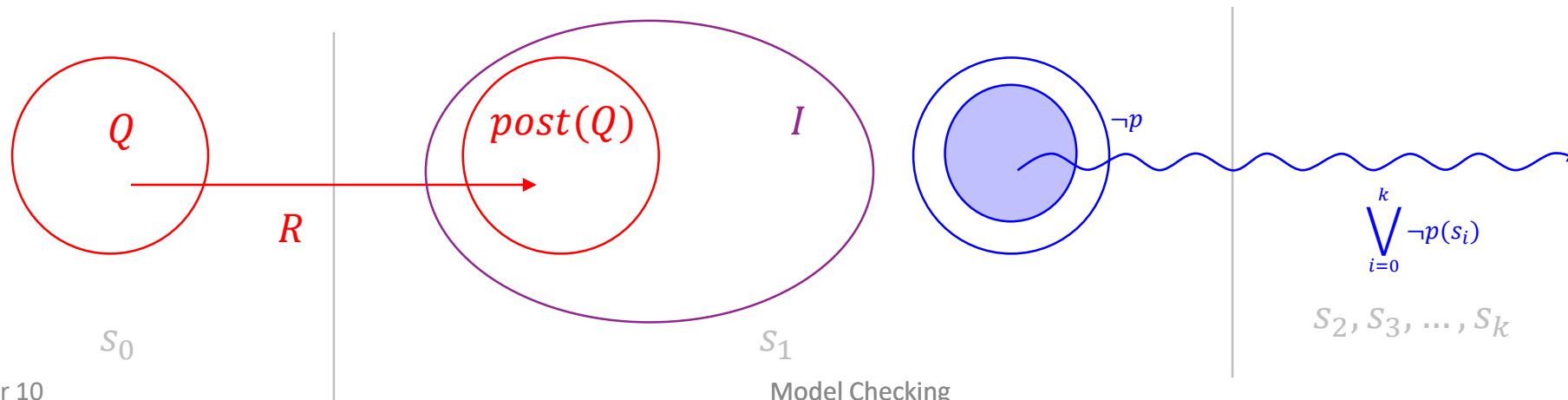
Instead, start from Q such that $Q \models p$

$$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i).$$

Suppose ϕ unsatisfiable, $I(s_1)$ is an interpolant

Note 1: $\neg p(s_1) \rightarrow B$
so $I(s_1) \wedge \neg p(s_1) = \perp$

Note 2: $I \supseteq \text{post}(Q)$



Interpolant Reachability Idea

$$\phi = Q(s_0) \wedge R(s_0, s_1) \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^k \neg p(s_i).$$

1. Start with $Q = S_0$
2. If ϕ not satisfiable, set Q to $Q \cup I$
3. If Q remains unchanged, p is **not reachable** (*Interpolants are approximation to post-image*), otherwise goto 2
4. If ϕ is satisfiable and $Q = S_0$, $\neg p$ is **reachable**
5. If ϕ is satisfiable and $Q \neq S_0$, increase k to increase precision of approximation, goto 1.

Procedure terminates when k is diameter of system (or earlier!)

Algorithm

procedure CraigReachability(model M , $p \in AP$)

if $S_0 \wedge \neg p$ is SAT return " $M \not\models AG p$ ";

$k := 1$;

$Q := S_0(s_0)$;

while true **do**

$A := Q(s_0) \wedge R(s_0, s_1)$;

$B := \bigvee_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i)$;

if $A \wedge B$ is SAT **then**

if $Q = S_0$ then return " $M \not\models AG p$ ";

 Increase k

$Q := S_0(s_0)$;

else

 compute interpolant I for A and B ;

if $I(s_0) == Q$ then return " $M \models AG p$ ";

$Q := Q \vee I(s_0)$;

end if

end while

end procedure

// $\neg p$ can be reached from Q

// $\neg p$ can be reached from S_0

// Not sure if path to $\neg p$ is real. Increase precision

// Reached the fixpoint of overapproximated reachability?

// Another step of overapproximated reachability?

10.4.4 Correctness

If CraigReachability returns “ $M \models AG p$ ” then $M \models AG p$

Let Q_i denote Q at iteration i . For all i , $Q_i \leftarrow postimage^i(Q_0)$. If $I \rightarrow Q_i$, we have reached a fixed point $Q^* = Q_i$ so $Q^* \leftarrow postimage^*(Q_0)$. Now because $Q_i \wedge \neg p = \perp$, we have $postimage^*(Q_0) \wedge \neg p = \perp$.

If CraigReachability returns “ $M \not\models AG p$ ” then $M \not\models AG p$

$A \wedge B$ encodes a path from Q_0 to $\neg p$.

CraigReachability terminates

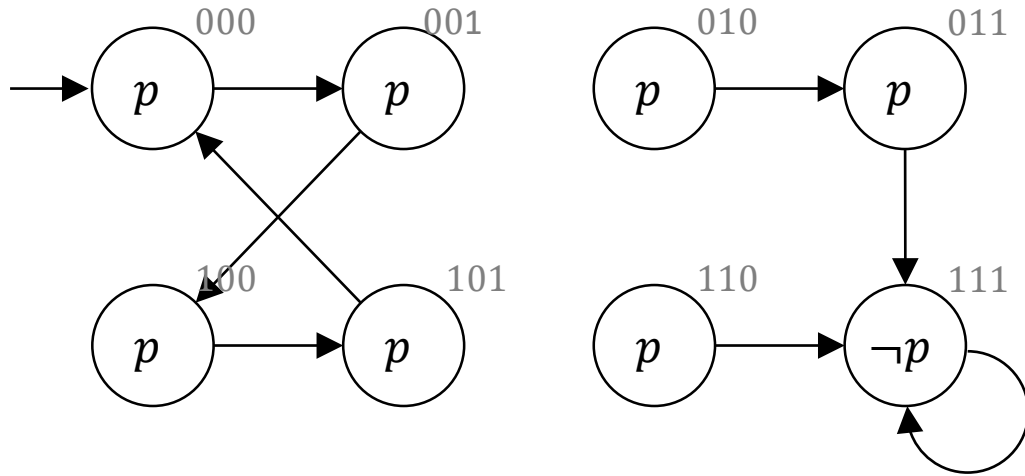
Note that k increases.

If $M \not\models AG p$, there is a path of length l to $\neg p$ and we will find it when $l = k$.

Suppose $M \models AG p$. If k is the diameter of the graph, no I and thus no Q_i can contain a state that reaches $\neg p$. Thus, $A \wedge B$ is never SAT and the algorithm terminates because the Q_i cannot grow forever.

$x_1x_2x_3$

Example $AG p$



$$\phi = Q(s_0) \wedge R(s_0, s_1) \\ \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i).$$

if $A \wedge B$ is SAT then

if $Q = S_0$ then return " $M \not\models AG p$ ";

increase k

$Q := S_0(s_0)$;

else

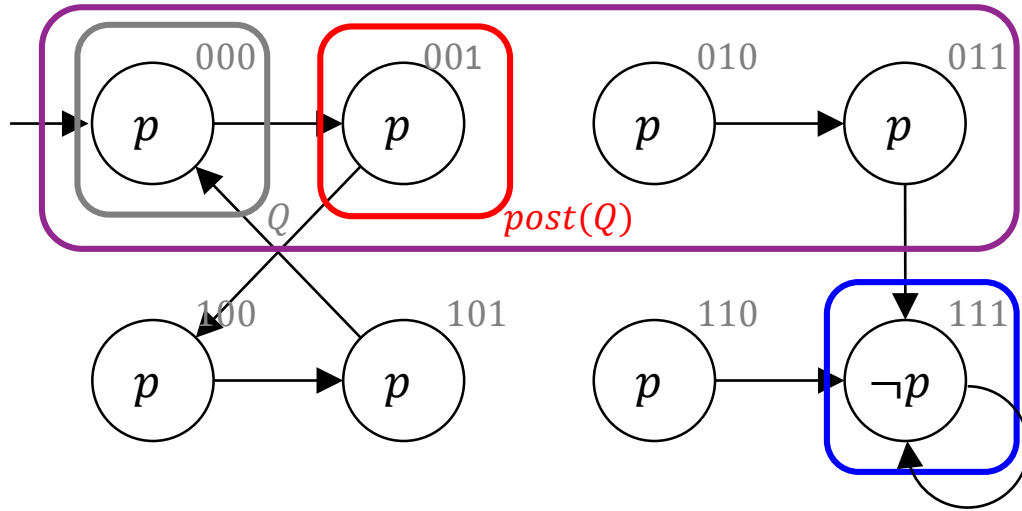
compute interpolant I for A and B ;

if $I(s_0) \rightarrow Q$ then return " $M \models AG p$ ";

$Q := Q \vee I(s_0)$;

$x_1x_2x_3$

Example $AG p$



$$\phi = Q(s_0) \wedge R(s_0, s_1) \\ \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i).$$

$$k = 1.$$

$$Q = \neg x_1 \wedge \neg x_2 \wedge \neg x_3 = \{000\}.$$

ϕ is UNSAT

Invariant checks first bit: $I = \neg x_1$

if $A \wedge B$ is SAT then

if $Q = S_0$ then return " $M \not\models AG p$ ";

increase k

$Q := S_0(s_0)$;

else

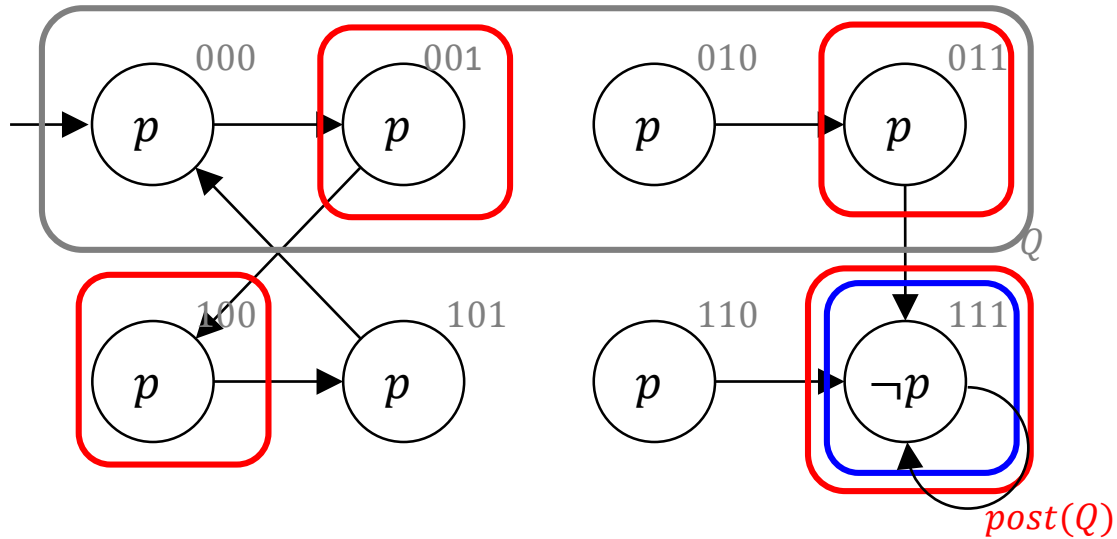
compute interpolant I for A and B ;

if $I(s_0) \rightarrow Q$ then return " $M \models AG p$ ";

$Q := Q \vee I(s_0)$;

$x_1x_2x_3$

Example $AG p$



$$\phi = Q(s_0) \wedge R(s_0, s_1) \\ \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i).$$

$$k = 1.$$

$$Q = \neg x_1 = \{000, 001, 010, 011\}.$$

ϕ is SAT

if $A \wedge B$ is SAT then

if $Q = S_0$ then return " $M \not\models AG p$ ";

increase k

$Q := S_0(s_0)$;

else

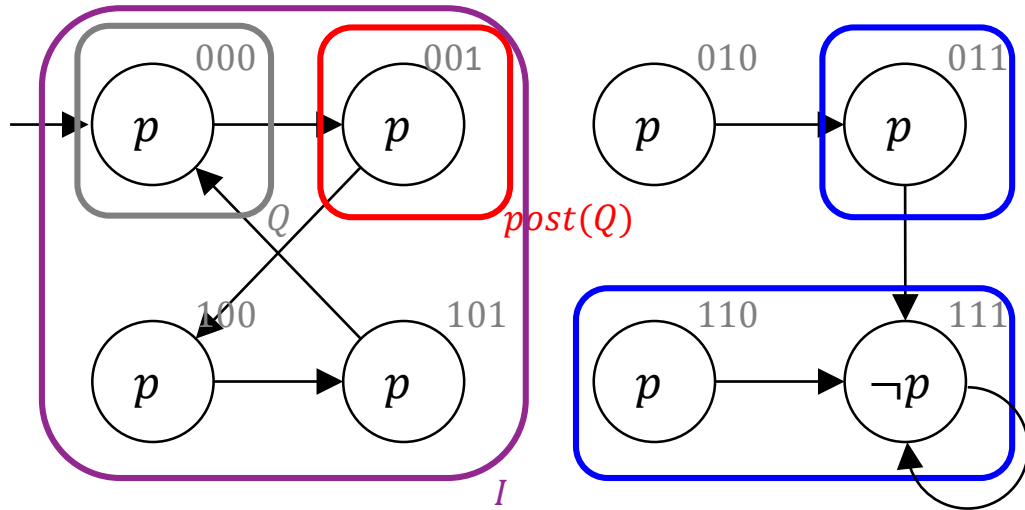
compute interpolant I for A and B ;

if $I(s_0) \rightarrow Q$ then return " $M \models AG p$ ";

$Q := Q \vee I(s_0)$;

$x_1x_2x_3$

Example $AG p$



$$\phi = Q(s_0) \wedge R(s_0, s_1) \\ \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i).$$

$$k = 2.$$

$$Q = \neg x_1 \wedge \neg x_2 \wedge \neg x_3 = \{000\}.$$

ϕ is UNSAT

Invariant checks 2nd bit: $I = \neg x_2$

if $A \wedge B$ is SAT then

if $Q = S_0$ then return " $M \not\models AG p$ ";

increase k

$Q := S_0(s_0)$;

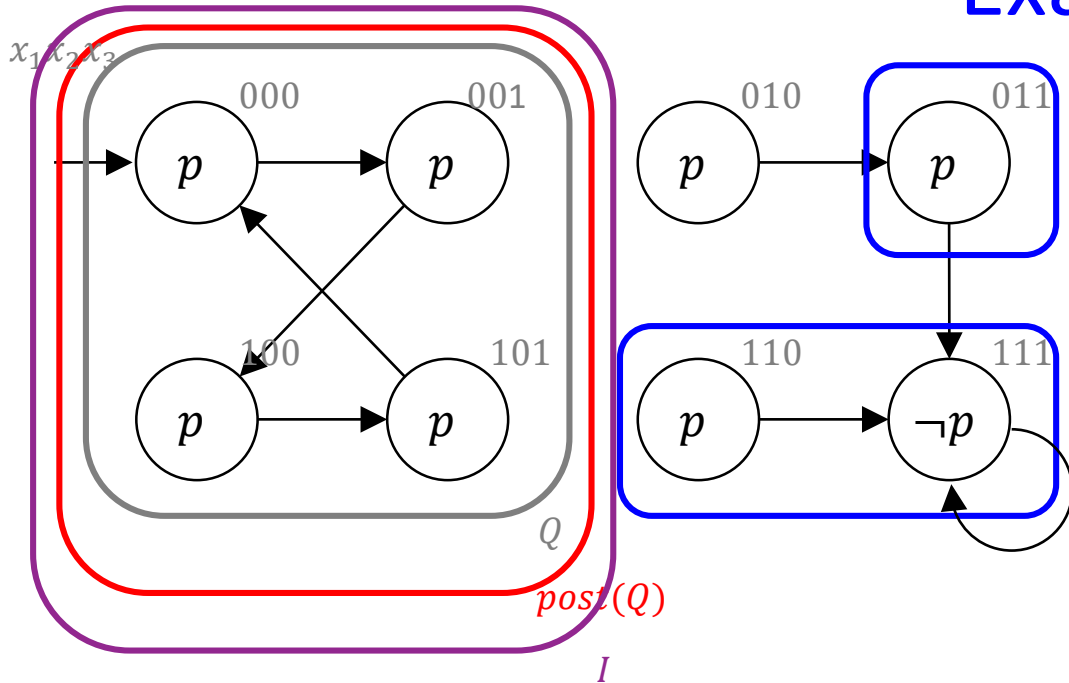
else

compute interpolant I for A and B ;

if $I(s_0) \rightarrow Q$ then return " $M \models AG p$ ";

$Q := Q \vee I(s_0)$;

Example $AG p$



if $A \wedge B$ is SAT then

if $Q = S_0$ then return " $M \not\models AG p$ ";

increase k

$Q := S_0(s_0)$;

else

compute interpolant I for A and B ;

if $I(s_0) \rightarrow Q$ then return " $M \models AG p$ ";

$Q := Q \vee I(s_0)$;

$$\phi = Q(s_0) \wedge R(s_0, s_1) \\ \wedge \bigwedge_{i=1}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=1}^k \neg p(s_i).$$

$k = 2.$

$$Q = \neg x_2 = \{000, 001, 100, 101\}$$

ϕ is UNSAT

$$I = \neg x_2 = Q.$$

Algorithm terminates.

How Did I Pick the Interpolants?

What I did

- Start with $A = \text{posting}(Q)$
- Perform each of the following steps
 1. Can I throw away x_3 ? (Is $(\exists x_3. A) \cap B = \emptyset$?) If yes, $A := \exists x_3. A$
 2. Can I throw away x_2 ? If yes, $A := \exists x_2. A$
 3. Can I throw away x_1 ? If yes, $A := \exists x_1. A$

This hack only works because the $\text{posting}(Q)$ is a state or a cube!

In the homework, you will get CNFs. Try getting rid of clauses.

Note that A is always a valid interpolant.

Next Week

2PM i13: Property-Directed Reachability

Homework!

Deadline: **April 13, 2023, 4:00 pm**

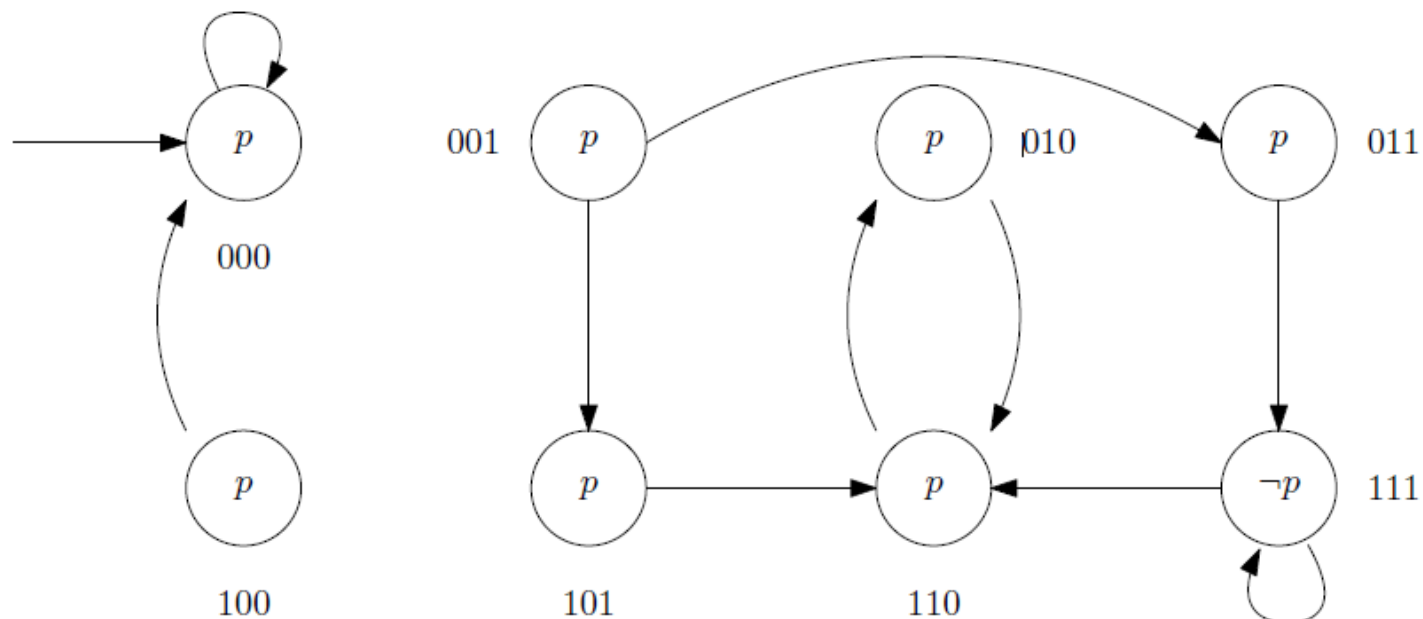
Send your solution to `modelchecking@iaik.tugraz.at`

Homework can be done in groups of 1 or 2 students.

The groups need not be the same for each homework.

Indicate clearly which students present the homework.

Consider the following Kripke structure K , with states $(x_1, x_2, x_3) \in \{0, 1\}^3$ and atomic proposition p .



Task 1. [10 points] Use Model Checking with Craig Interpolants to prove whether the property $AG p$ is true or false.

Clearly indicate the steps. I would like to see the interpolants as formulas, for anything else, you can use set notation. You can also draw the sets, but use enough copies of the Kripke structure to make sure we can understand your steps, at least one for every k .

Use the same heuristic shown in class to find the interpolants. The heuristic shown in class is a hack, but it works in this example.

Property-Directed Reachability *or IC3*



Aaron Bradley

PDR

Property-Directed Reachability or IC3

- Makes no copies of transition relation – memory efficient
- Overapproximate postimage (like interpolation)

PDR Notation

Formula $X(V) \wedge R(V, V') \wedge Y(V')$ is shortened to $\mathbf{X} \wedge \mathbf{R} \wedge \mathbf{Y}'$

Meaning: there is a state $s \in X$ and a state $s' \in Y$ such that $(s, s') \in R$
(there is an edge from s to s')

$\mathbf{s} := \mathbf{SAT}(F_i \wedge \mathbf{R} \wedge \neg \mathbf{P}')$:

- $s := FALSE$ if $\neg \mathbf{SAT}(F_i \wedge \mathbf{R} \wedge \neg \mathbf{P}')$
- $s :=$ a state satisfying in F_i with an edge to a state in $\neg P$ in otherwise

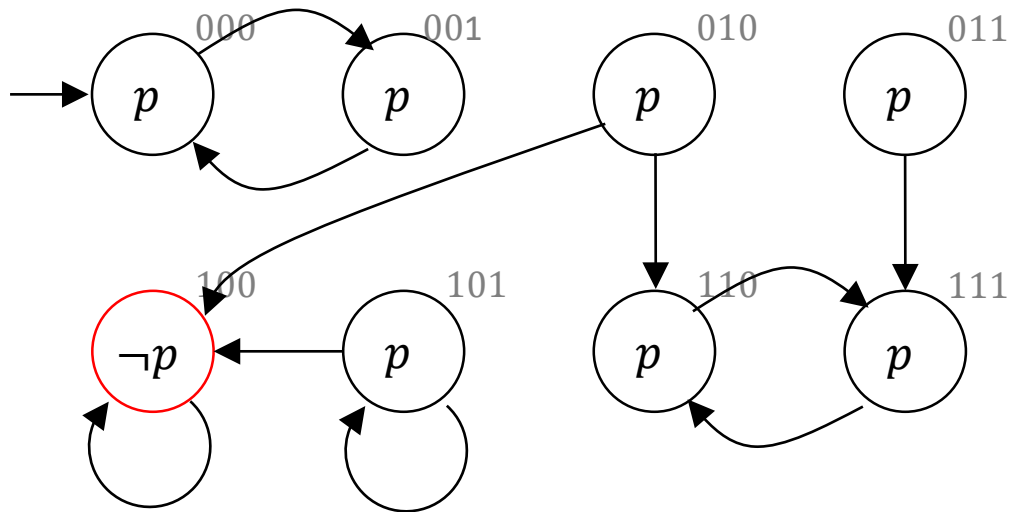
PDR: Notation

Definition

- $I \subseteq S$ is **inductive** if
 1. $S_0 \rightarrow I$ ($S_0 \subseteq I$)
 2. $I \wedge R \rightarrow I'$ ($\text{postimage}(I) \subseteq I$)
- $I \subseteq S$ is **inductive relative to F** if
 1. $S_0 \rightarrow I$ ($S_0 \subseteq I$)
 2. $I \wedge F \wedge R \rightarrow I'$ ($\text{postimage}(F \cap I) \subseteq I$)

Relative Inductiveness

$x_1x_2x_3$

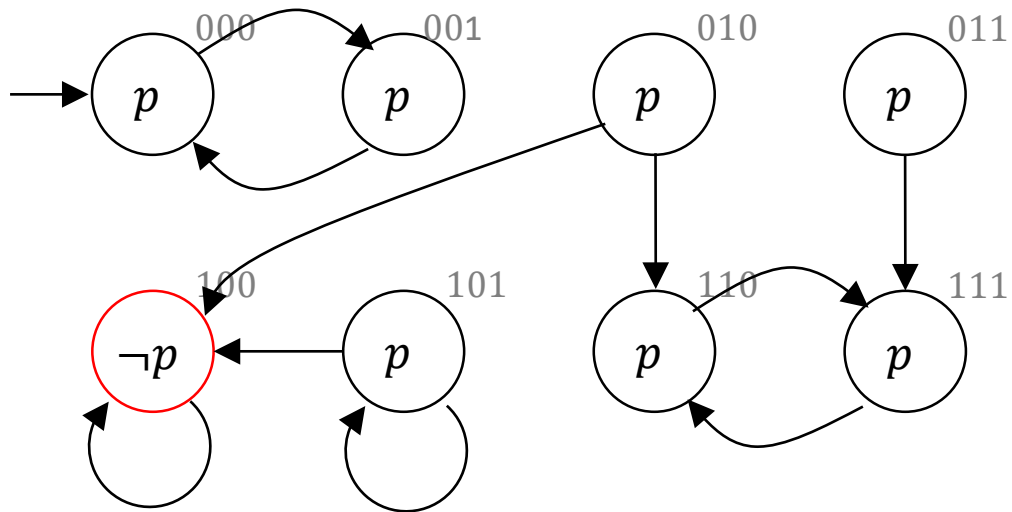


Inductive facts

- Is $\neg x_1$ inductive?
- Is $\neg x_2$ inductive?

Relative Inductiveness

$x_1x_2x_3$



Inductive facts

- Is $\neg x_1$ inductive?
 - $S_0 \rightarrow \neg x_1$
 - $\neg x_1 \wedge R \rightarrow \neg x'_1$ is **false**
 - **No!**
- Is $\neg x_2$ inductive?
 - $S_0 \rightarrow \neg x_2$
 - $\neg x_2 \wedge R \rightarrow \neg x'_2$
 - **Yes!**
- Is $\neg x_1$ inductive relative to x_2 ?
 - $S_0 \rightarrow \neg x_1$
 - $\neg x_2 \wedge \neg x_1 \wedge R \rightarrow \neg x'_1$
 - **Yes!**

Idea: Find (relatively) inductive facts.

PDR: Data Structures & Invariants

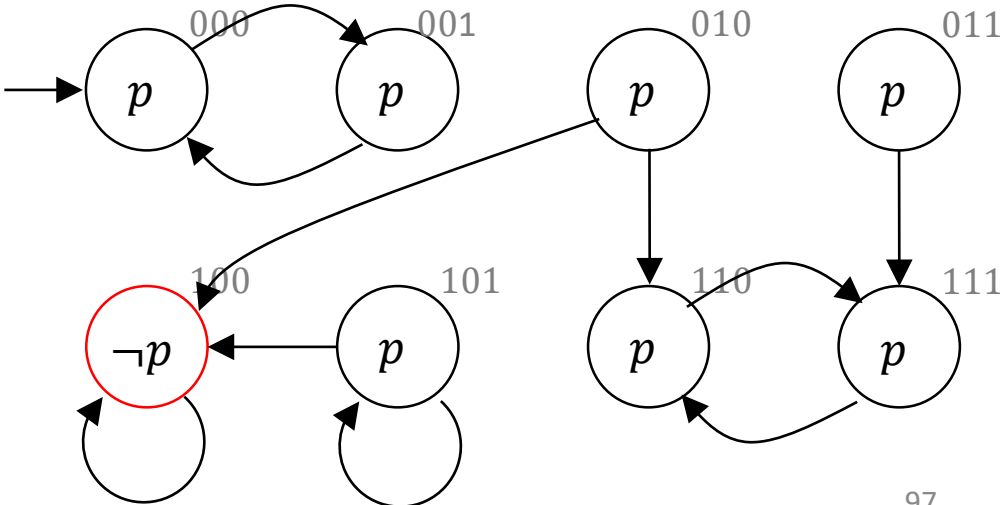
Data Structure

Clause: Disjunction of literals. **Cube:** Conjunction of literals.

Clause and cubes signify a set of states. Longer clauses – more states. Longer cubes – fewer states

Formulas F_0, \dots, F_k over V , stored as sets of Clauses (Sets $F_0, \dots, F_k \subseteq S$)

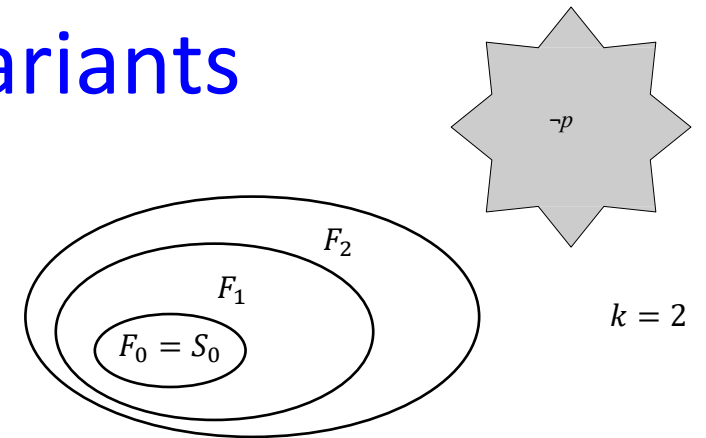
Example



PDR: Data Structures & Invariants

Invariants

- I1: $S_0 \rightarrow F_0$ ($S_0 \subseteq F_0$)
- I2: $F_i \rightarrow F_{i+1}$ ($F_i \subseteq F_{i+1}$)
 - I2': $F_i = F_{i+1} \wedge c_{i1} \wedge \dots \wedge c_{in}$
- I3: $F_i \rightarrow P = FALSE$ ($F_i \subseteq P$)
- I4: $F_i \wedge R \rightarrow F'_{i+1}$ ($postimg(F_i) \subseteq F_{i+1}$)



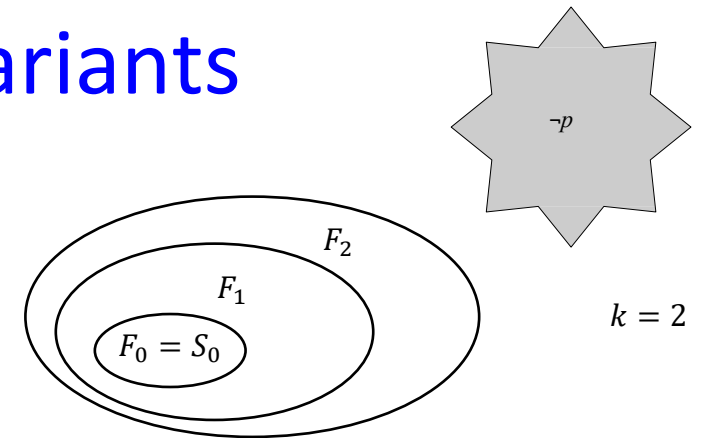
PDR: Data Structures & Invariants

Invariants

- I1: $S_0 = F_0$ ($S_0 = F_0$)
- I2: $F_i \rightarrow F_{i+1}$ ($F_i \subseteq F_{i+1}$)
 - I2': $F_i = F_{i+1} \wedge c_{i1} \wedge \dots \wedge c_{in}$
- I3: $F_i \rightarrow P = FALSE$ ($F_i \subseteq P$)
- I4: $F_i \wedge R \rightarrow F'_{i+1}$ ($postimg(F_i) \subseteq F_{i+1}$)

Facts

1. $\forall 0 < i \leq k$: There is no trace from F_i to $\neg p$ of $k - i$ edges or less (I3,I4)
2. Because of I4:
 1. $postimg^i(S_0) \subseteq F_i$
 2. There is no trace from S_0 to F_i of length $< i$
 3. There is no counterexample $k + 1$ edges or less (with I3)
- If $F_i = F_{i+1}$ then system is correct. (By I3, I4, F_i is an inductive invariant)



PDR, First Version

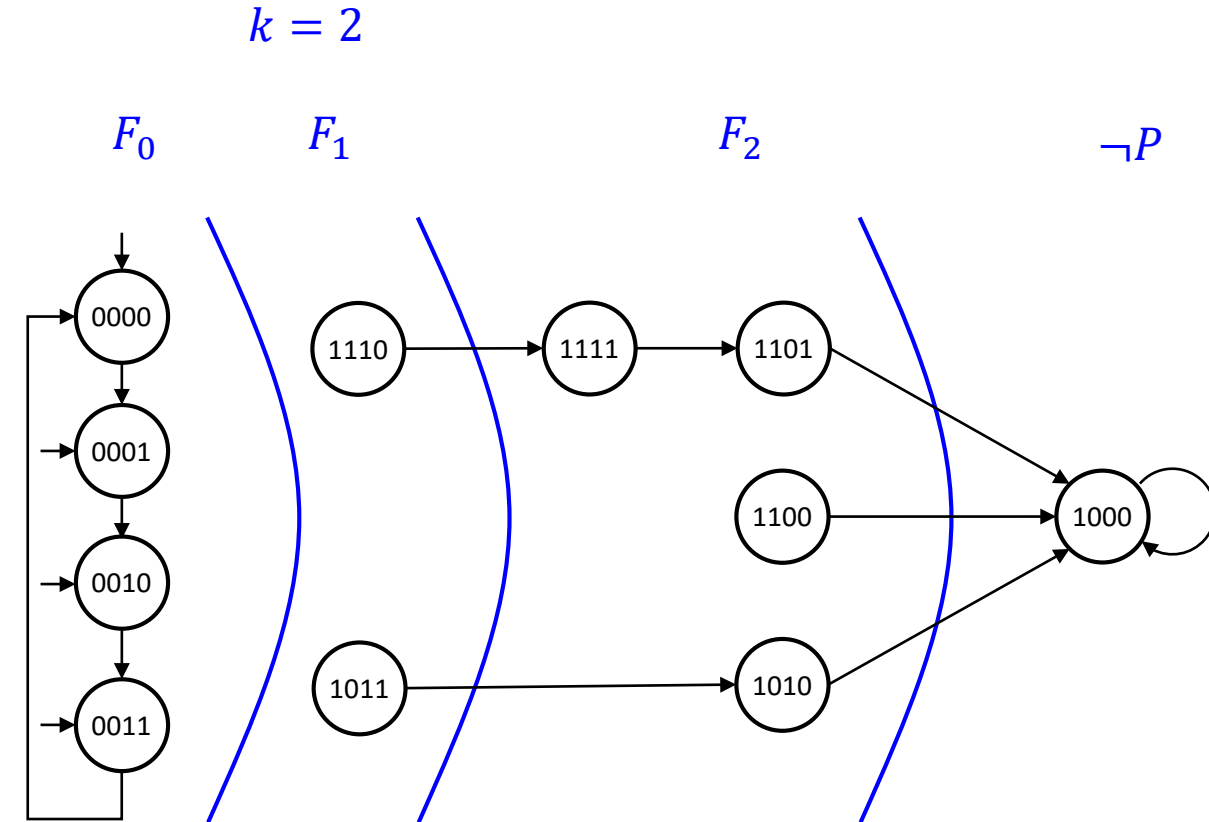
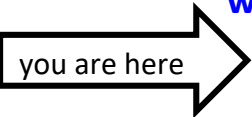
```

function PDR(Model  $M$ )
  if SAT( $S_0 \wedge \neg P$ ) or SAT( $S_0 \wedge R \wedge \neg P'$ ) then FAIL
   $F_0 := S_0; F_1 := P; k := 1;$ 
  while(true)
    while( $s := \text{SAT}(F_k \wedge R \wedge \neg P')$ )
      removeBad( $k, s$ )
       $k++; F_k := P$ 

    if  $\exists 0 \leq i < k - 1: F_i := F_{i+1}$  then SUCCEED

  // post:  $\neg \text{SAT}(F_i \wedge s)$ 
  function removeBad( $i \in N, \text{state } s$ )
    if SAT( $S_0 \wedge c$ ) then FAIL
    while( $t := \text{SAT}(F_{i-1} \wedge R \wedge s')$ )
      removeBad( $i - 1, t$ )

     $\forall 0 < j \leq i: F_j := F_j \wedge \neg s$ 
  
```



Goal: look for counterexamples of length 3.
 If we find one – done
 If we don't – We have an F_3

PDR, First Version

function PDR(Model M)

if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

$F_0 := S_0; F_1 := P; k := 1;$

while(true)

while($s := \text{SAT}(F_k \wedge R \wedge \neg P')$)

 removeBad(k, s)

$k++; F_k := P$

if $\exists 0 \leq i < k - 1: F_i = F_{i+1}$ **then SUCCEED**

// post: $\neg \text{SAT}(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge s$) **then FAIL**

while($t := \text{SAT}(F_{i-1} \wedge R \wedge s')$)

 removeBad($i - 1, t$)

$\forall 0 < j \leq i: F_j := F_j \wedge \neg s$

remove states in F_k
with edge to $\neg P$

remove states in F_i
with path to $\neg P$ of
length $k - i + 1$

PDR, First Version

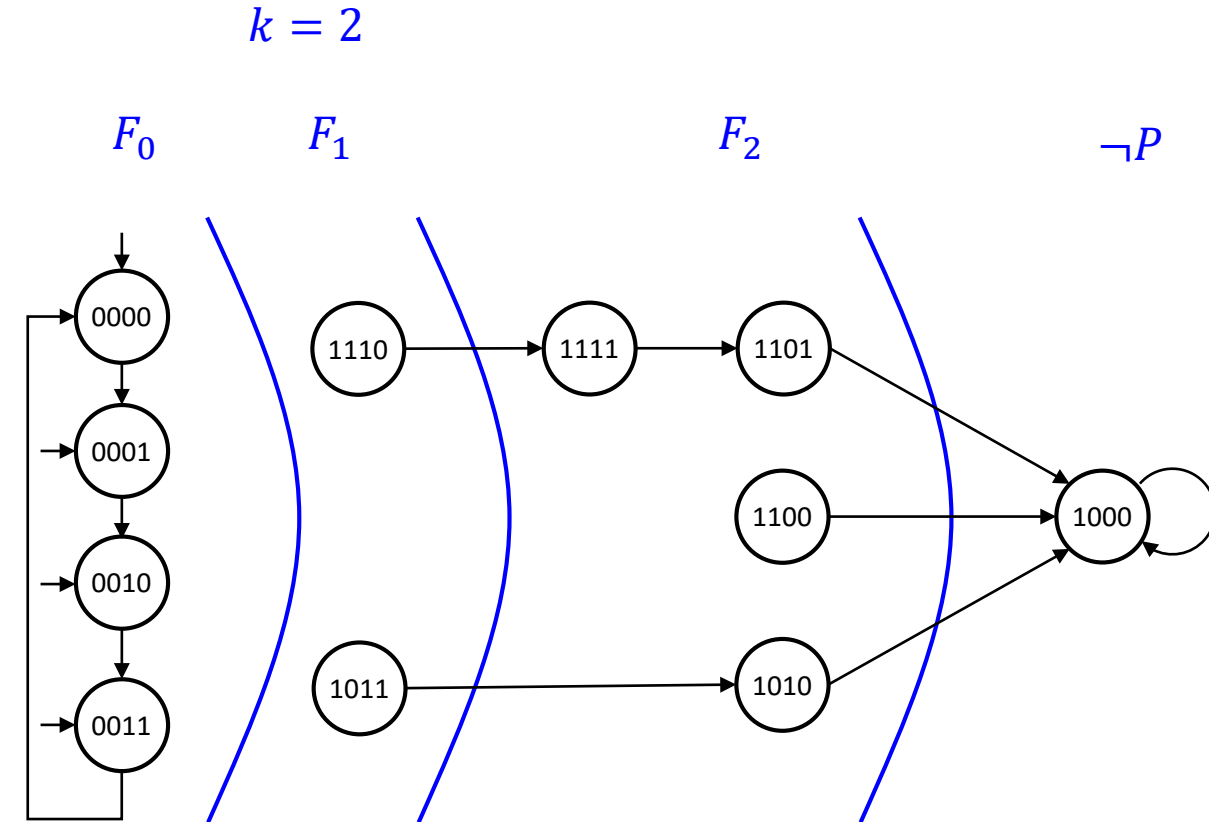
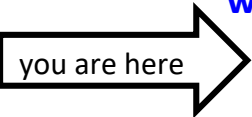
```

function PDR(Model  $M$ )
  if SAT( $S_0 \wedge \neg P$ ) or SAT( $S_0 \wedge R \wedge \neg P'$ ) then FAIL
   $F_0 := S_0; F_1 := P; k := 1;$ 
  while(true)
    while( $s := \text{SAT}(F_k \wedge R \wedge \neg P')$ )
      removeBad( $k, s$ )
       $k++; F_k := P$ 

    if  $\exists 0 \leq i < k - 1: F_i := F_{i+1}$  then SUCCEED

  // post:  $\neg \text{SAT}(F_i \wedge s)$ 
  function removeBad( $i \in N, \text{state } s$ )
    if SAT( $S_0 \wedge c$ ) then FAIL
    while( $t := \text{SAT}(F_{i-1} \wedge R \wedge s')$ )
      removeBad( $i - 1, t$ )

     $\forall 0 < j \leq i: F_j := F_j \wedge \neg s$ 
  
```



Do the invariants hold?

PDR, First Version

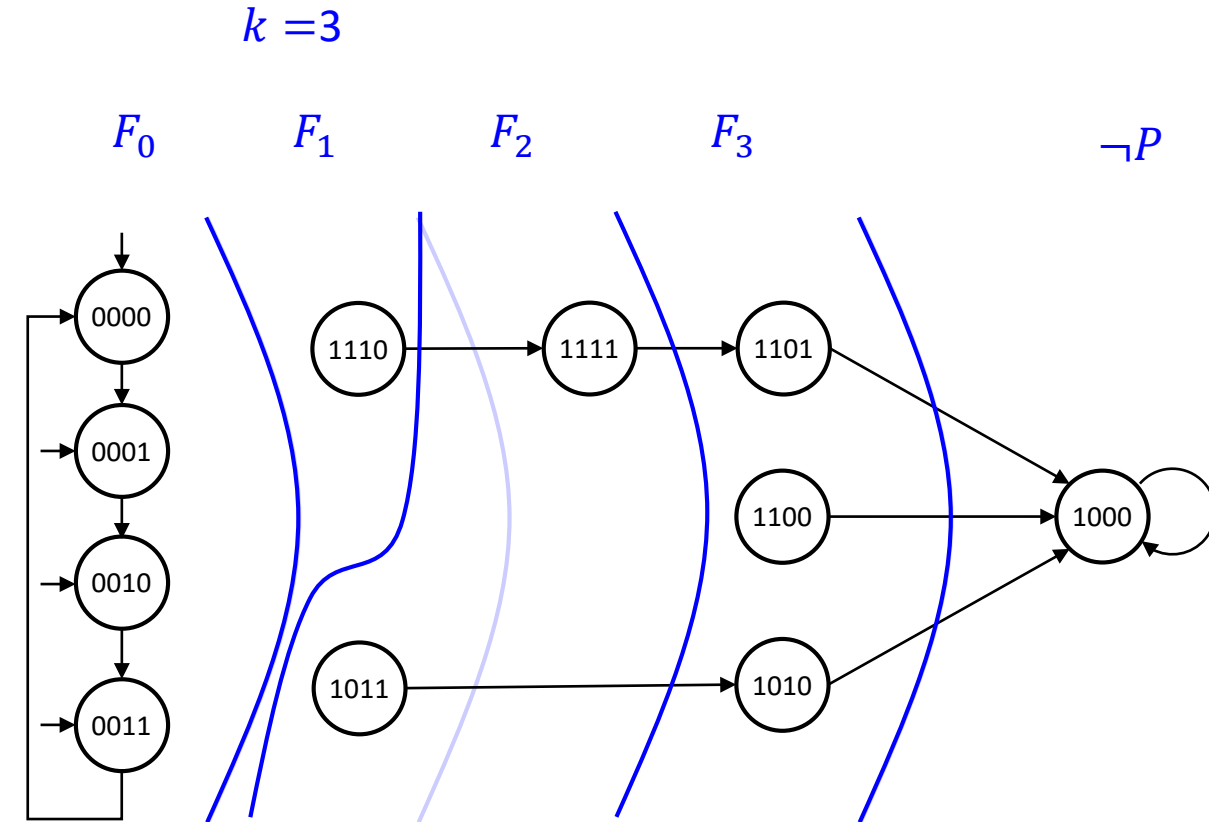
```

function PDR(Model  $M$ )
  if SAT( $S_0 \wedge \neg P$ ) or SAT( $S_0 \wedge R \wedge \neg P'$ ) then FAIL
   $F_0 := S_0; F_1 := P; k := 1;$ 
  while(true)
    while( $s := \text{SAT}(F_k \wedge R \wedge \neg P')$ )
      removeBad( $k, s$ )
       $k++; F_k := P$ 

    if  $\exists 0 \leq i < k - 1: F_i := F_{i+1}$  then SUCCEED

  // post:  $\neg \text{SAT}(F_i \wedge s)$ 
  function removeBad( $i \in N, \text{state } s$ )
    if SAT( $S_0 \wedge c$ ) then FAIL
    while( $t := \text{SAT}(F_{i-1} \wedge R \wedge s')$ )
      removeBad( $i - 1, t$ )

     $\forall 0 < j \leq i: F_j := F_j \wedge \neg s$ 
  
```



PDR, First Version

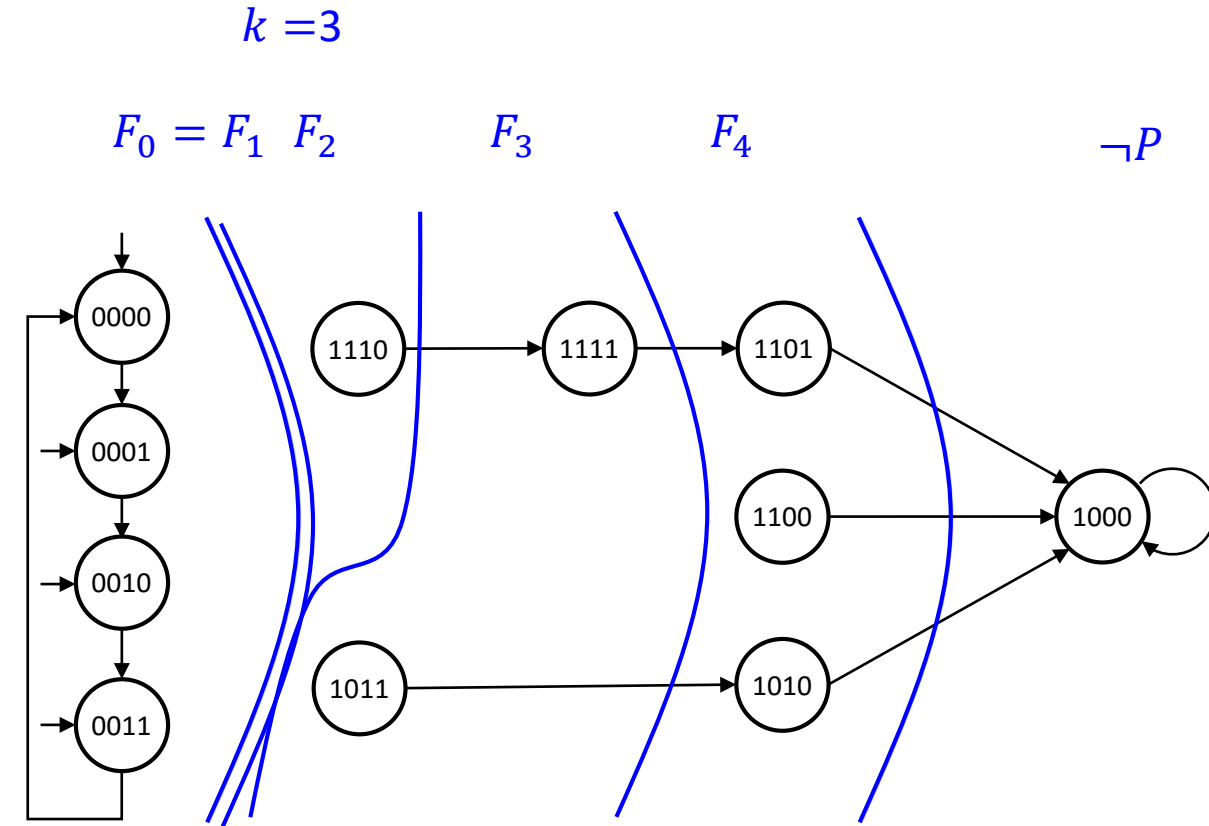
```

function PDR(Model  $M$ )
  if SAT( $S_0 \wedge \neg P$ ) or SAT( $S_0 \wedge R \wedge \neg P'$ ) then FAIL
   $F_0 := S_0; F_1 := P; k := 1;$ 
  while(true)
    while( $s := \text{SAT}(F_k \wedge R \wedge \neg P')$ )
      removeBad( $k, s$ )
       $k++; F_k := P$ 

    if  $\exists 0 \leq i < k - 1: F_i := F_{i+1}$  then SUCCEED

// post:  $\neg \text{SAT}(F_i \wedge s)$ 
function removeBad( $i \in N, \text{state } s$ )
  if SAT( $S_0 \wedge c$ ) then FAIL
  while( $t := \text{SAT}(F_{i-1} \wedge R \wedge s')$ )
    removeBad( $i - 1, t$ )

   $\forall 0 < j \leq i: F_j := F_j \wedge \neg s$ 
  
```

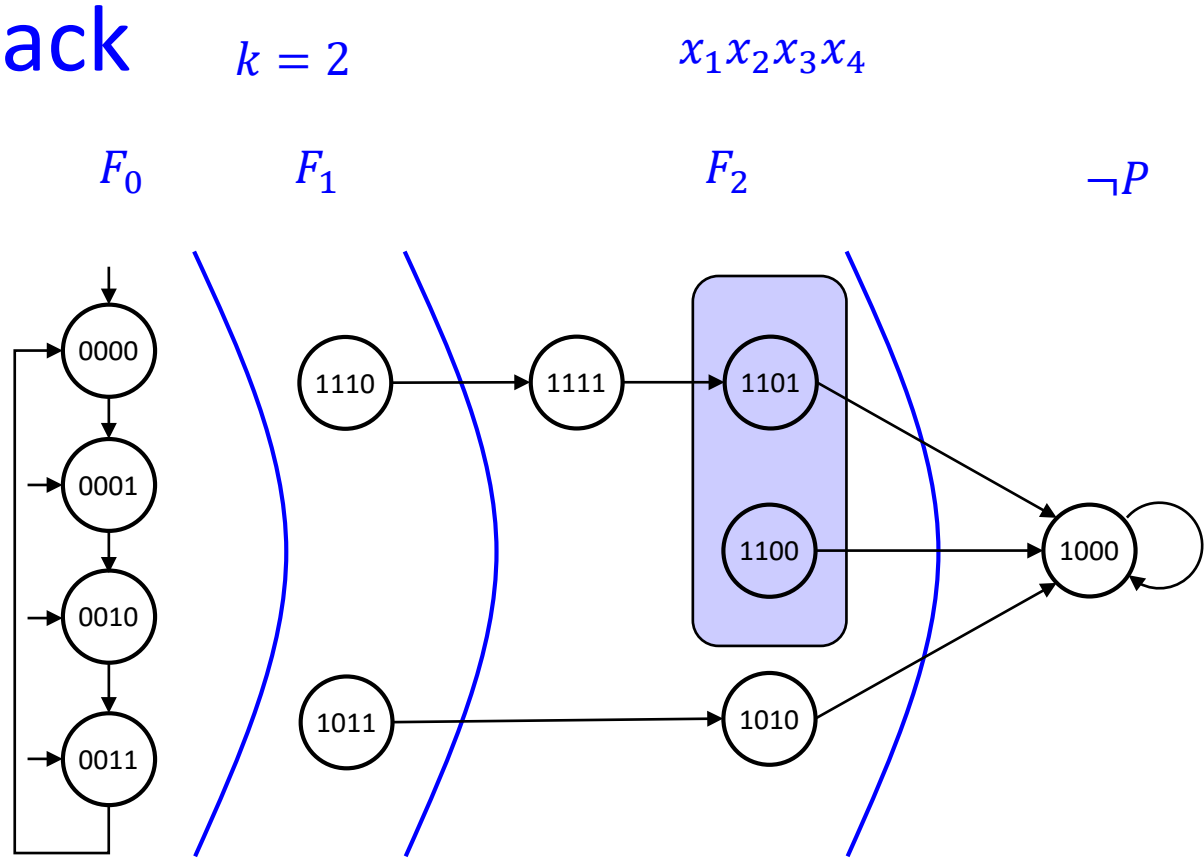


Drawback

- The first version considers every state individually
- But similar states behave similarly.
Example: 1100 and 1101.
Generalize to $110- = x_1 \wedge x_2 \wedge \neg x_3!$

Conditions

- $\text{UNSAT}(F_1 \wedge R \wedge x'_1 \wedge x'_2 \wedge \neg x'_3)$
- $\text{UNSAT}(S_0 \wedge x_1 \wedge x_2 \wedge \neg x_3)$



PDR: Naive Generalization

```

function PDR(Model  $M$ )
  if SAT( $S_0 \wedge \neg P$ ) or SAT( $S_0 \wedge R \wedge \neg P'$ ) then FAIL
   $F_0 := S_0; F_1 := P; k := 1;$ 
  while(true)
    while( $s := \text{SAT}(F_k \wedge R \wedge \neg P')$ )
      removeBad( $k, s$ )
     $k++;$   $F_k := P$ 

    if  $\exists 0 \leq i < k - 1: F_i := F_{i+1}$  then SUCCEED
  
```

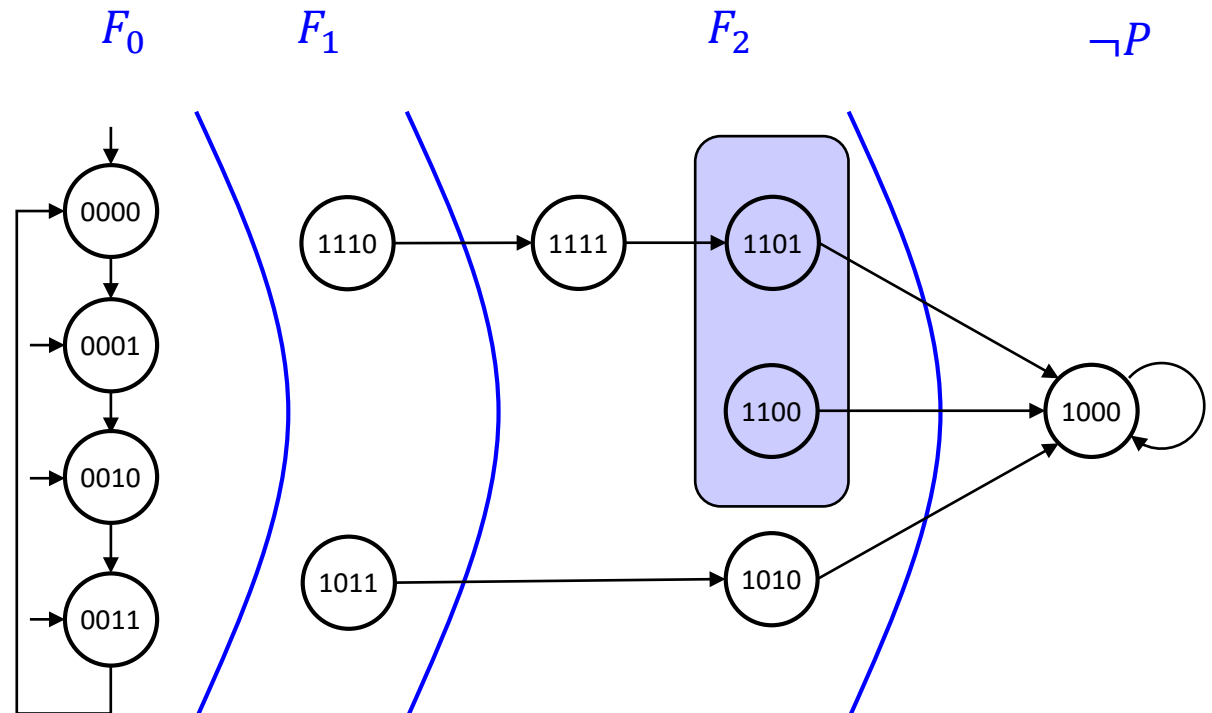
// post: $\neg \text{SAT}(F_i \wedge s)$

```

function removeBad( $i \in N, \text{state } s$ )
  if SAT( $S_0 \wedge c$ ) then FAIL
  while( $t := \text{SAT}(F_{i-1} \wedge R \wedge s')$ )
    removeBad( $i - 1, t$ )
   $g := \text{generalizeNaive}(i, s)$ 
   $\forall 0 < j \leq i: F_j := F_j \wedge \neg g$ 
  
```

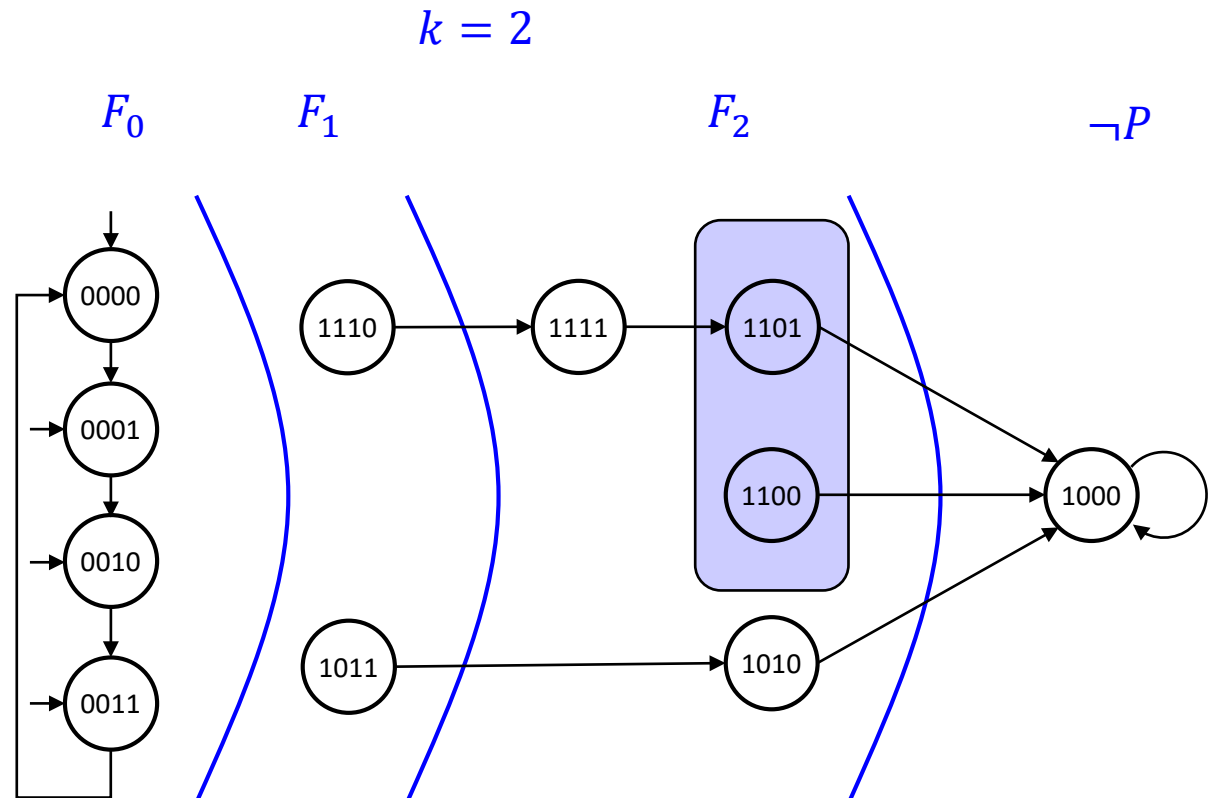
```

function generalizeNaive( $i, \text{state } s$ )
  return a shortest cube  $c$  such that
    -  $c \leftarrow s$ 
    -  $\neg \text{SAT}(F_{i-1} \wedge R \wedge c')$ 
    -  $\neg \text{SAT}(S_0 \wedge c)$ 
  
```



Generalize Further?

- We can generalize to 110-
- Can we generalize to 11--?
 - NO: $f = x_1 \wedge x_2$, we have $\text{SAT}(F_1 \wedge R \wedge f)$
- State 1100 is the problem



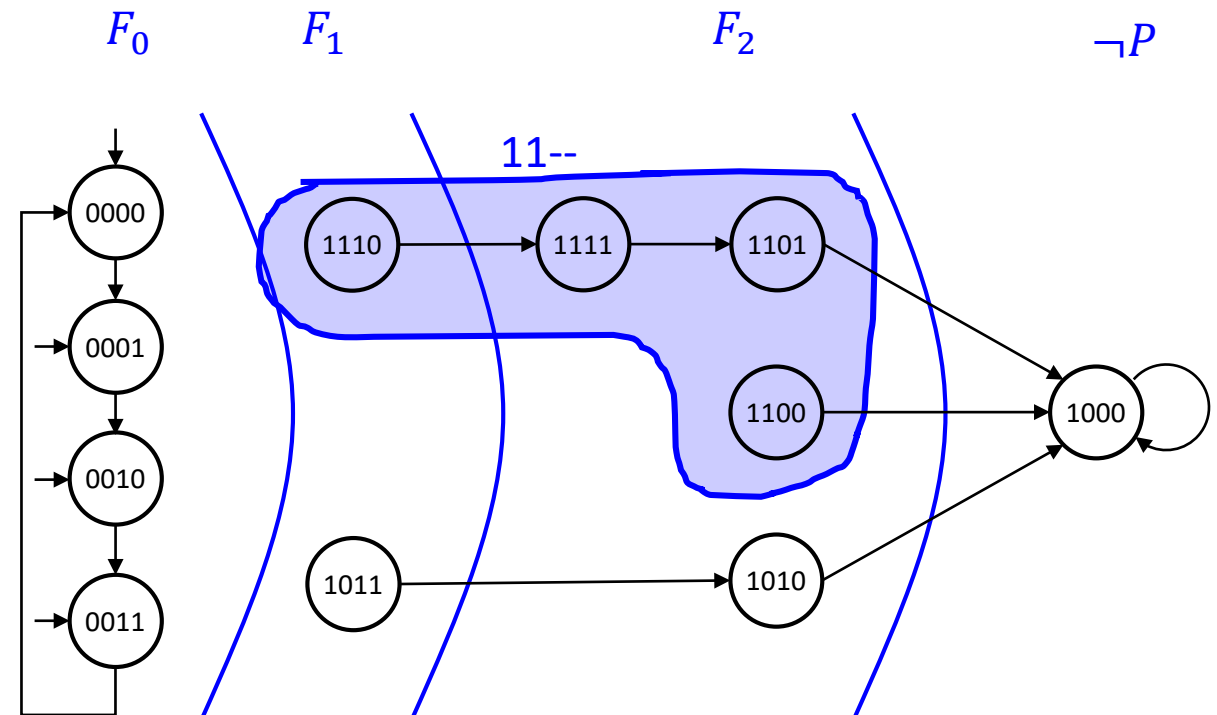
Relative Inductiveness

shortest cube c such that

- $c \leftarrow S$
- $\neg \text{SAT}(\neg c \wedge F_1 \wedge R \wedge c')$
- $\neg \text{SAT}(S_0 \wedge c)$

$\neg c$ is relative inductive wrt F_2

Why?



PDR: Relative Inductiveness

```

function PDR(Model  $M$ )
  if SAT( $S_0 \wedge \neg P$ ) or SAT( $S_0 \wedge R \wedge \neg P'$ ) then FAIL
   $F_0 := S_0; F_1 := P; k := 1;$ 
  while(true)
    while( $s := \text{SAT}(F_k \wedge R \wedge \neg P')$ )
      removeBad( $k, s$ )
     $k++; F_k := P$ 

    if  $\exists 0 \leq i < k - 1: F_i = F_{i+1}$  then SUCCEED
  
```

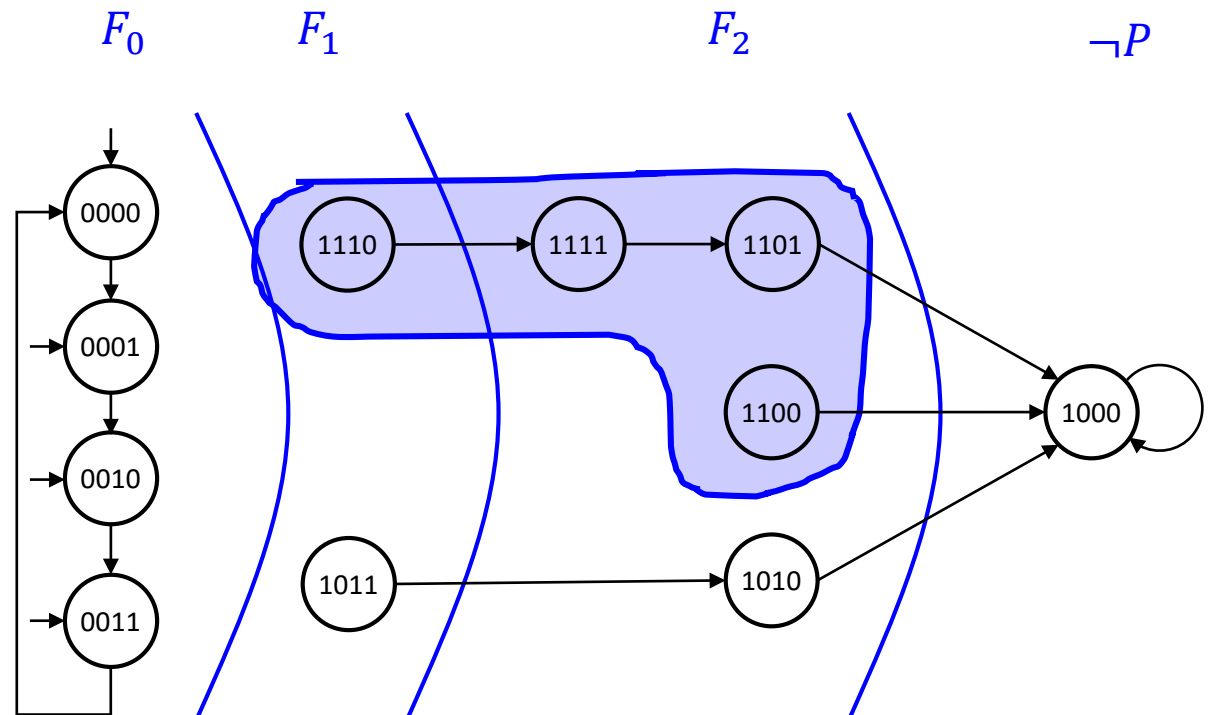
// post: $\neg \text{SAT}(F_i \wedge s)$

```

function removeBad( $i \in N, \text{state } s$ )
  if SAT( $S_0 \wedge c$ ) then FAIL
  while( $t := \text{SAT}(F_{i-1} \wedge R \wedge s')$ )
    removeBad( $i - 1, t$ )
   $g := \text{generalize}(i, s)$ 
   $\forall 0 < j \leq i: F_j := F_j \wedge \neg g$ 
  
```

```

function generalize( $i, \text{state } s$ )
  return a shortest cube  $c$  such that
    -  $c \leftarrow s$ 
    -  $\neg \text{SAT}(\neg c \wedge F_{i-1} \wedge R \wedge c')$ 
    -  $\neg \text{SAT}(S_0 \wedge c)$ 
  
```



Generalization

function generalize(i , state s)

$c := s$

while c changes

let l_1, \dots, l_n be the literals of c

for $i := 1$ **to** n

$c' = c$ with l_i removed

if relInd(c') **then** $c = c'$

return c

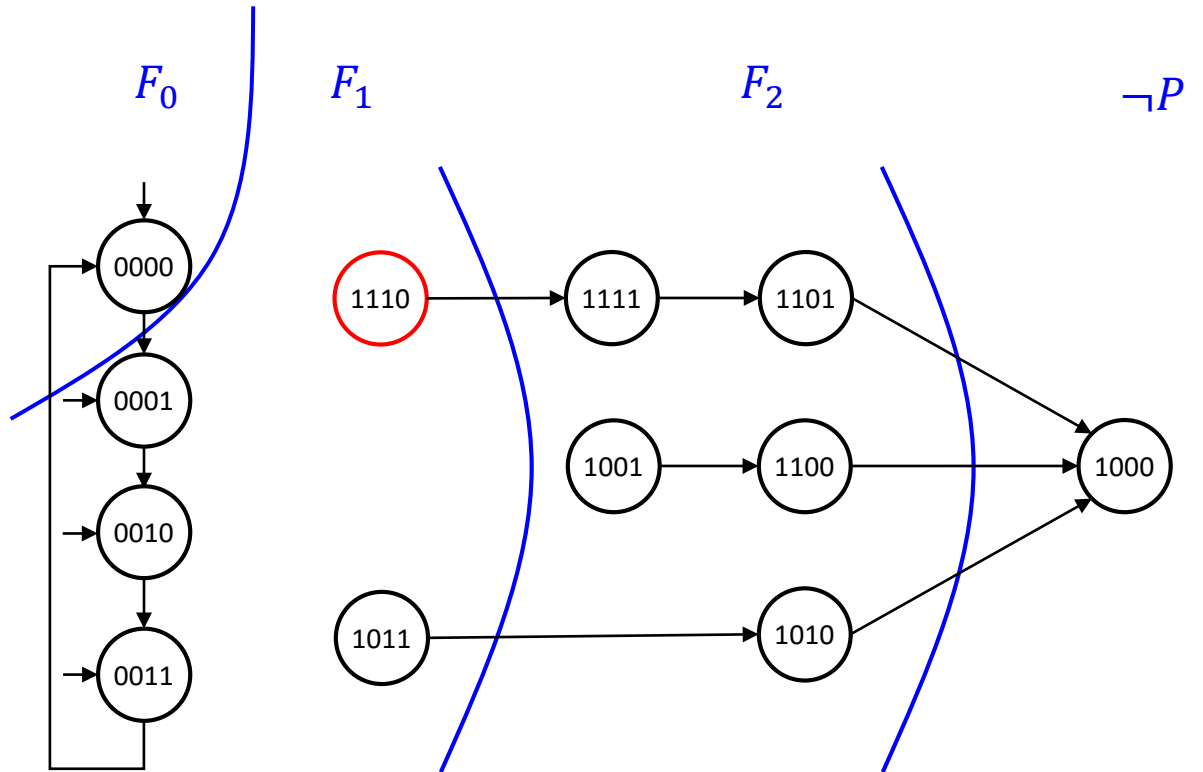
function relInd(cube c)

return $\neg\text{SAT}(\neg c \wedge F_{i-1} \wedge R \wedge c)$
 $\wedge \neg\text{SAT}(S_0 \wedge c)$

Propagate Clauses

Suppose you are removing 1110.
You can generalize to 1---

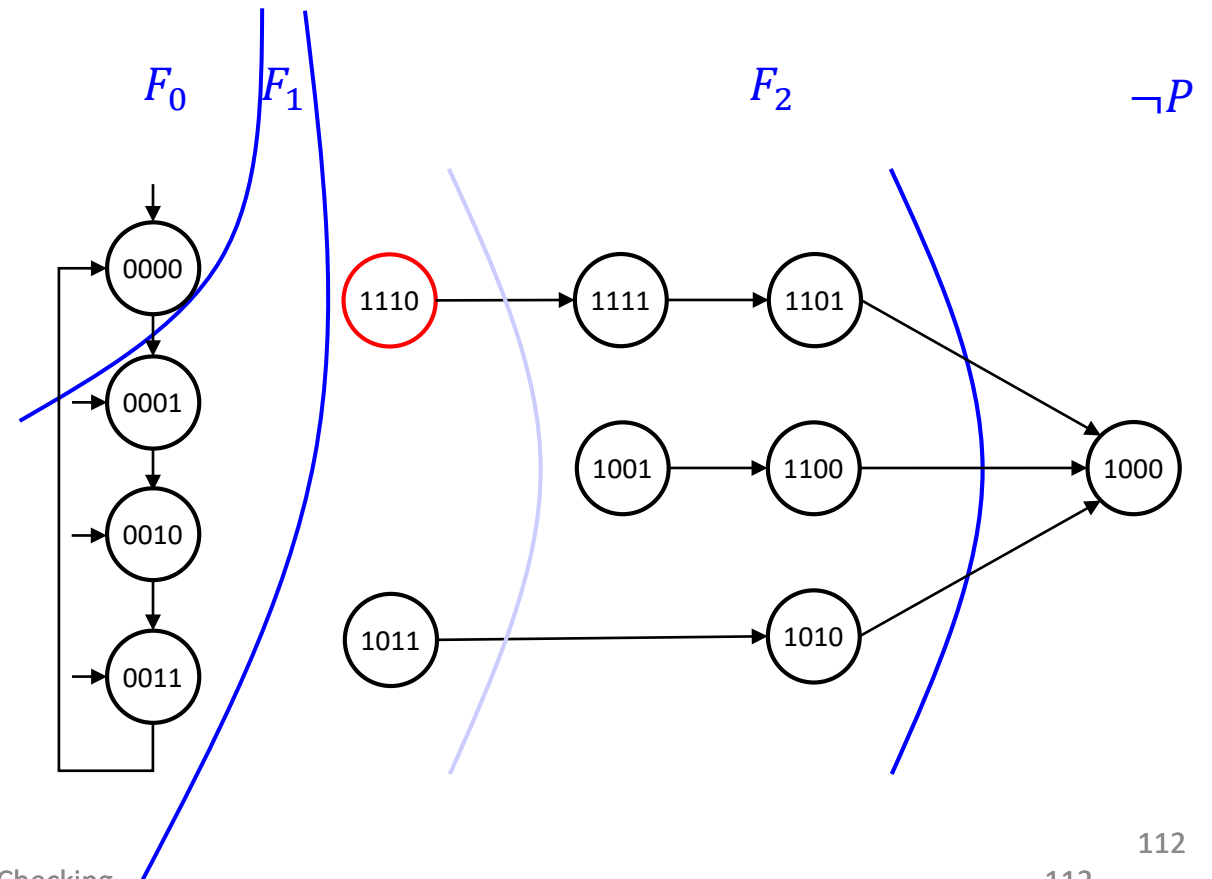
$$F_1 := F_1 \wedge \neg x_1$$



Propagate Clauses

$F_2 \wedge x_1 \notin \text{postimg}(F_1)$, so we can add $\neg x_1$ to F_2

$\text{UNSAT}(F_1 \wedge R \wedge F'_2 \wedge x'_1)$



PDR, Final: Propagate Clauses

```
function PDR(Model  $M$ )
  if SAT( $S_0 \wedge \neg P$ ) or SAT( $S_0 \wedge R \wedge \neg P'$ ) then FAIL
   $F_0 := S_0; F_1 := P; k := 1;$ 
  while(true)
    while( $s := \text{SAT}(F_k \wedge R \wedge \neg P')$ )
      removeBad( $k, s$ )
     $k++;$   $F_k := P$ 
    propagateClauses( $k$ )
    if  $\exists 0 \leq i < k - 1: F_i = F_{i+1}$  then SUCCEED
```

// post: $\neg \text{SAT}(F_i \wedge s)$

```
function removeBad( $i \in N, \text{state } s$ )
  if SAT( $S_0 \wedge s$ ) then FAIL
  while( $t := \text{SAT}(F_{i-1} \wedge R \wedge s')$ )
    removeBad( $i - 1, t$ )
   $g := \text{generalize}(i, s)$ 
   $\forall 0 < j \leq i: F_j := F_j \wedge \neg g$ 
```

```
function generalize( $i, \text{state } s$ )
  return a shortest cube  $c$  such that
    -  $c \leftarrow s$ 
    -  $\neg c$  inductive relative to  $F_{i-1}$ 
```

```
function propagateClauses( $k$ )
  for  $i := 1$  to  $k - 1$ 
    for every clause  $c \in F_i$ 
      if  $\neg \text{SAT}(F_i \wedge R \wedge \neg c')$ 
         $F_{i+1} := F_{i+1} \wedge c$ 
```


Further Ideas

- This version is somewhat simplified and doesn't find long counterexamples quickly
- Equivalence of frames = syntactic check
 - Use implication and subsumption to simplify clauses
 - Check Mischchenko paper to see if we add clauses when they are subsumed

Performance

Hardware Model Checking Competition 2020

1. AVR 11 variants of IC3+[abstraction], 2x BMC, 3x k-induction
2. AVY interpolation + PDR
3. nuXmv “portfolio”, including IC3
4. Pono: protfolio, including BMC, k-induction, interpolation, IC3

Literature

Literature

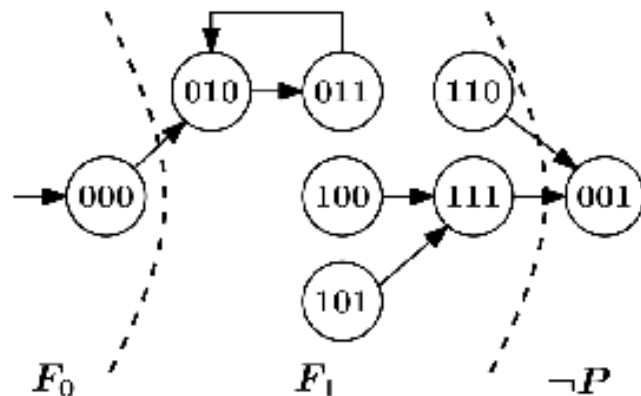
- A. R. Bradley, SAT-Based Model Checking without Unrolling, VMCAI 2011.
http://ecee.colorado.edu/~bradleya/ic3/ic3_bradley.pdf
- N. Een, A. Mishchenko, R. Brayton, Efficient Implementation of Property Directed Reachability, FMCAD 2011.
https://people.eecs.berkeley.edu/~alanmi/publications/2011/fmcad11_pdr.pdf
- F. Somenzi, Aaron R. Bradley: IC3: where monolithic and incremental meet. FMCAD 2011: 3-8. http://theory.stanford.edu/~arbrad/papers/ic3_tut.pdf
- A. R. Bradley: Understanding IC3. SAT 2012: 1-14.
https://theory.stanford.edu/~arbrad/papers/Understanding_IC3.pdf

Model Checking Homework 4

Deadline: 7 April 4:00pm

Send solution to: modelchecking@iaik.tugraz.at

Consider the following synchronous Kripke structure K , with states of the form $x_1x_2x_3$. The only initial state is 000, and we are given a property P that holds everywhere but in 001.



We wish to prove that P is always true using PDR. We began the algorithm, obtaining the frames F_0 and F_1 as shown in the figure.

Task 4a [4 points]. Starting from the figure, carry out two iterations of the first variant of the PDR (from $k=1$, until $k=3$) shown in class. Clearly indicate the steps and the frames at the end of each iteration. Is the property P verified at the end? Why/Why not?

Task 4b [3 points]. As before, perform two iterations of PDR starting from the figure. This time use “naive generalization” during the removal of bad states, as shown in class. Clearly indicate the steps and the frames at the end of each iteration. Is the property P verified at the end? Why/Why not?

Task 4c [3 points]. Which of the following statements are false? Justify your answer.

- The set $\neg x_1$ is inductive.
- The set $\neg x_3$ is inductive.
- The set $\neg x_2$ is inductive relative to $\neg x_1$.
- The set $\neg x_3$ is inductive relative to $\neg x_1$.