Property-Directed Reachability *or IC3*

(a simplified version)

Aaron Bradley

PDR

Property-Directed Reachability or IC3

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

PDR Notation

Formula $X(V) \wedge R(V, V') \wedge Y(V')$ is shortened to $\boldsymbol{X} \wedge \boldsymbol{R} \wedge \boldsymbol{Y}'$
Meaning: there is an edge from s to s'

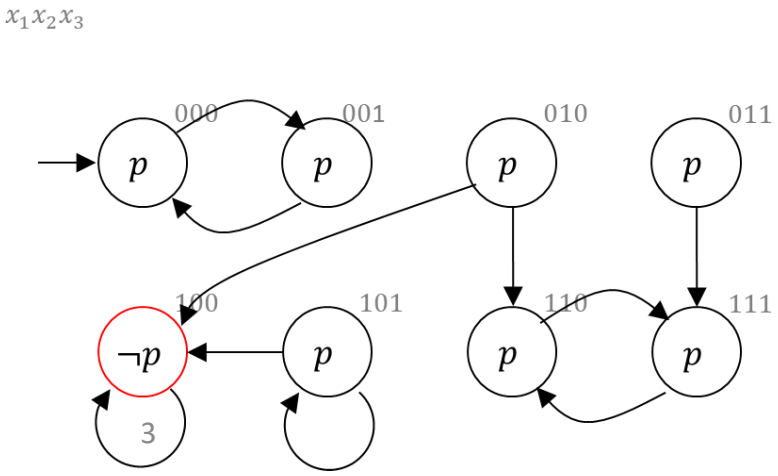
s := SAT($\boldsymbol{A} \wedge \boldsymbol{R} \wedge \boldsymbol{B}'$):

- $s := FALSE$ if ¬SAT(A ∧ R ∧ B')
- s := a state in A with an edge to a state in B, otherwise

Example

$SAT(\mathrm{x}_1 \wedge \mathrm{R} \wedge \neg x_1)$

$SAT(\neg \mathrm{x}_1 \wedge R \wedge x_1')$



PDR Notation

Formula $X(V) \wedge R(V, V') \wedge Y(V')$ is shortened to $\boldsymbol{X} \wedge \boldsymbol{R} \wedge \boldsymbol{Y}'$
Meaning: there is an edge from s to s'

s := SAT($\boldsymbol{A} \wedge \boldsymbol{R} \wedge \boldsymbol{B}'$):

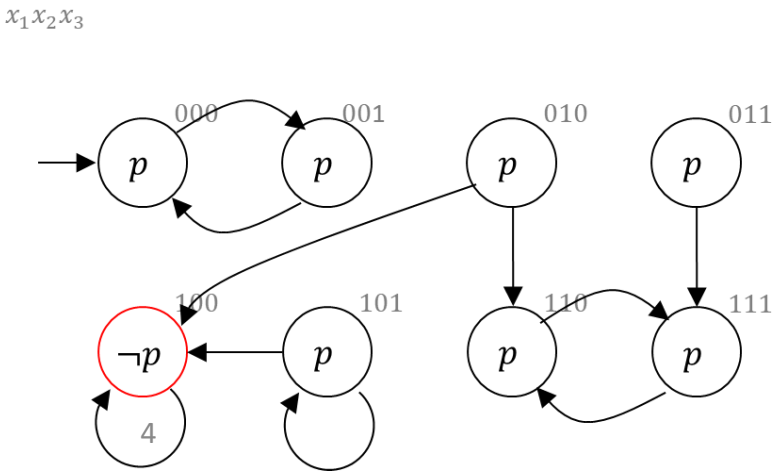
- $s := FALSE$ if ¬SAT(A ∧ R ∧ B')
- s := a state in A with an edge to a state in B, otherwise

Example

$SAT(\mathrm{x}_1 \wedge \mathrm{R} \wedge \neg x_1) = FALSE.$

$SAT(\neg \mathrm{x}_1 \wedge R \wedge x_1')$ returns

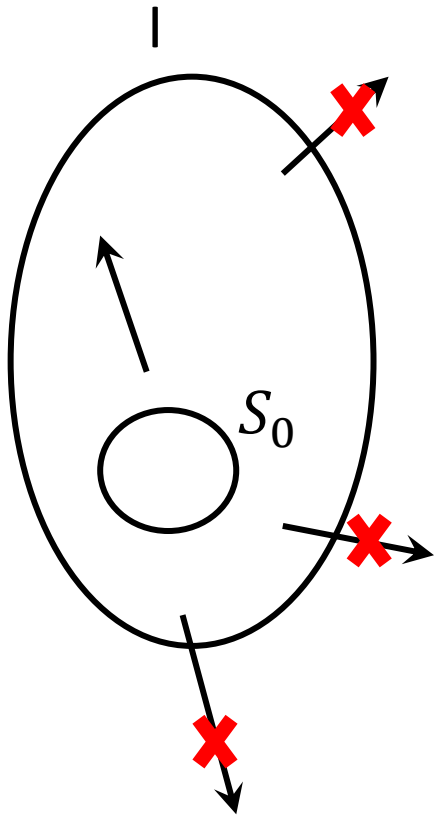
$\neg x_1 \wedge x_2 \wedge \neg x_2$ or $\neg x_1 \wedge x_2 \wedge x_2$



PDR: Notation

Definition

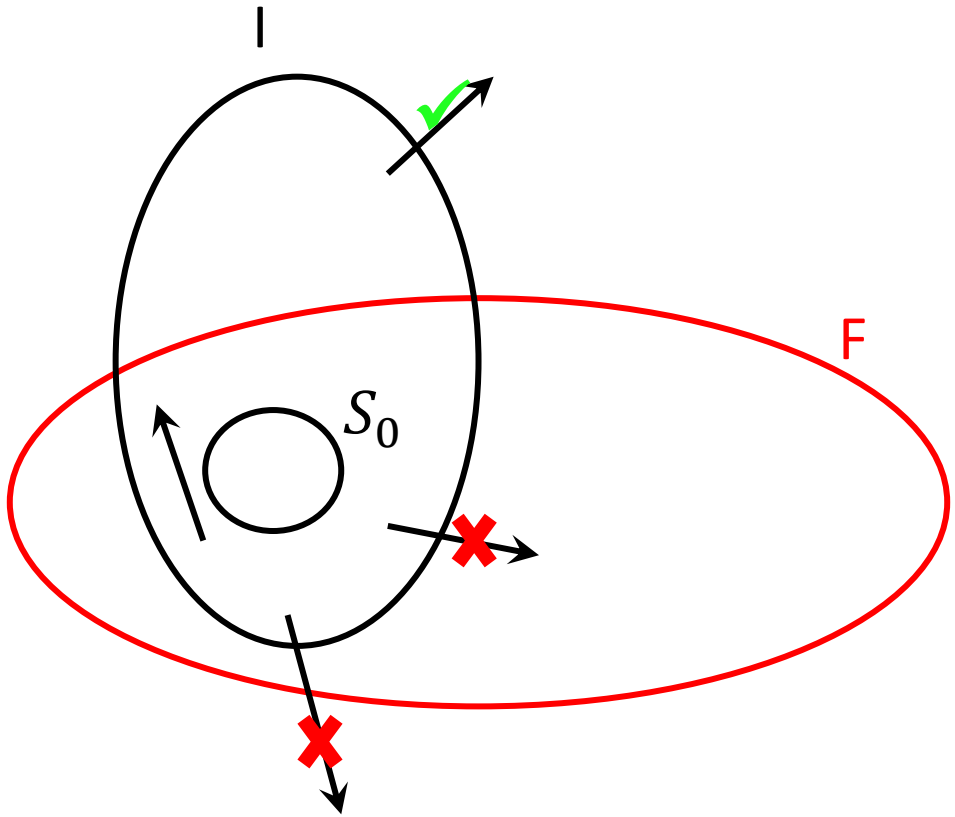
- $I \subseteq S$ is inductive if
  1. $S_0 \rightarrow I \;\; (S_0 \subseteq I)$
  2. $I \wedge R \;\rightarrow I' \; (img(I) \subseteq I)$



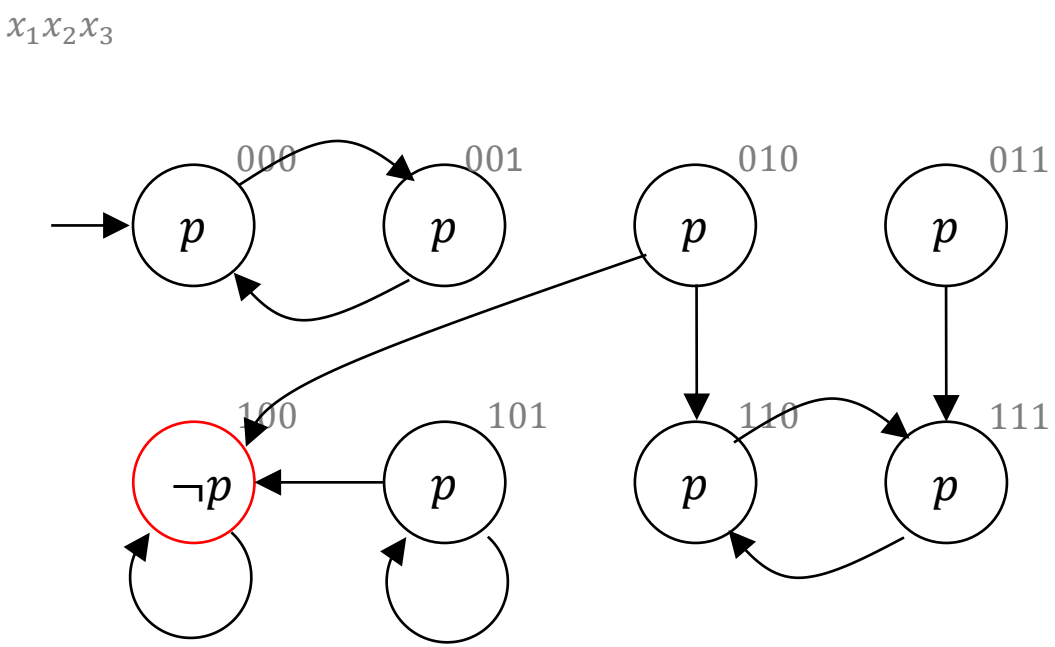
PDR: Notation

Definition

- $I \subseteq S$ is inductive if
  1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
  2. $I \wedge R \rightarrow I'$ $(img(I) \subseteq I)$
- $I \subseteq S$ is inductive relative to F if
  1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
  2. $I \wedge F \wedge R \rightarrow I'$ $(img(F \cap I) \subseteq I)$



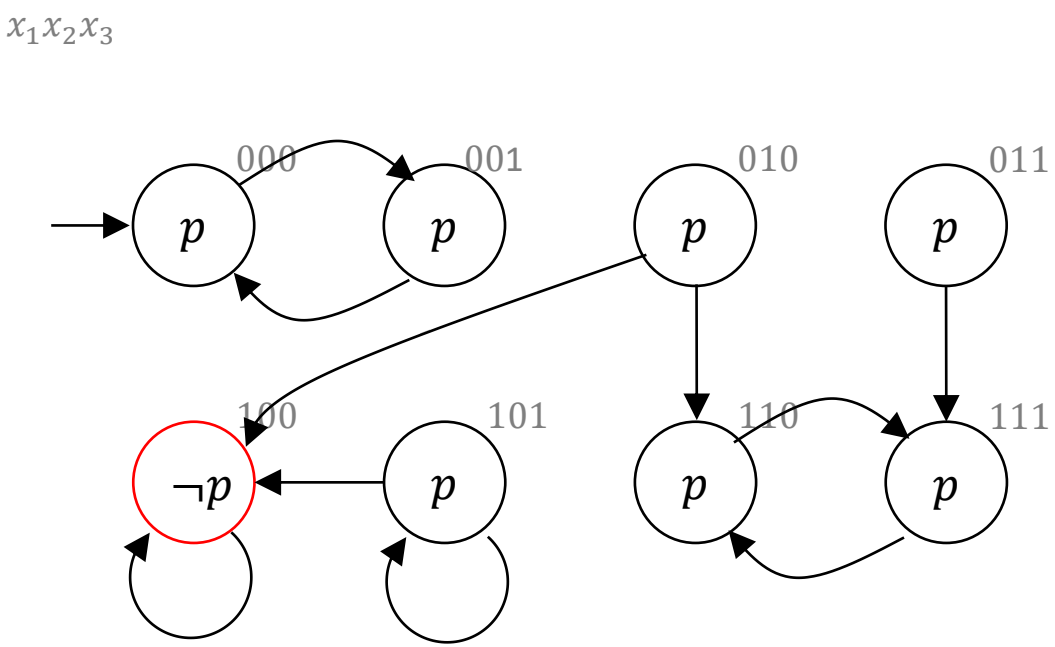
Relative Inductiveness



Inductive facts

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

Relative Inductiveness



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 $\neg x_2$?
  - $S_0 \rightarrow \neg x_1$
  - $\neg x_2 \wedge \neg x_1 \wedge R \rightarrow \neg x'_1$
  - **Yes**!

Idea: Find (relatively) inductive facts.

PDR: Data Structures & Invariants

Data Structures

Clause: Disjunction of literals. E.g. $\neg x_1 \vee x_2 \vee \neg x_3$

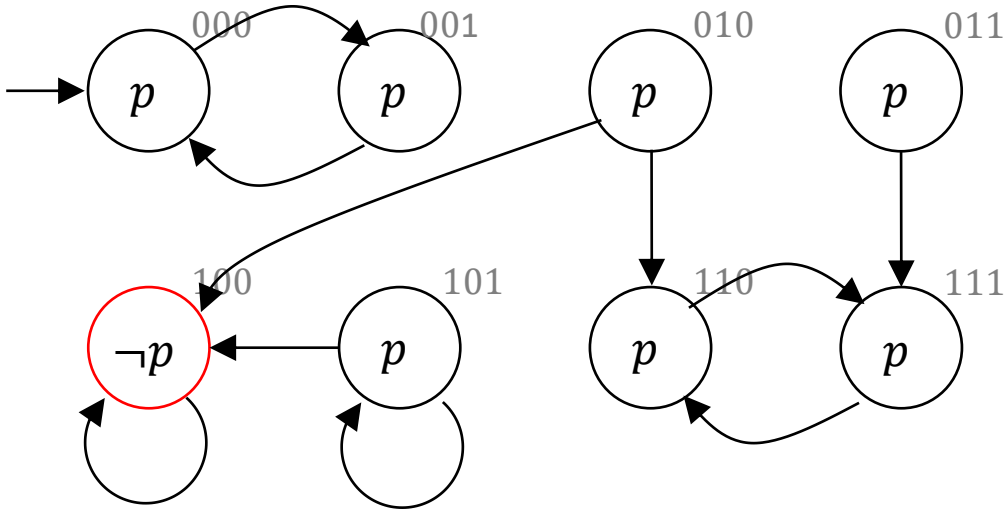
Cube: Conjunction of literals. E.g. $\neg x_1 \wedge x_2 \wedge \neg x_3$

- A state is a cube
- Clause and cubes are sets of states.

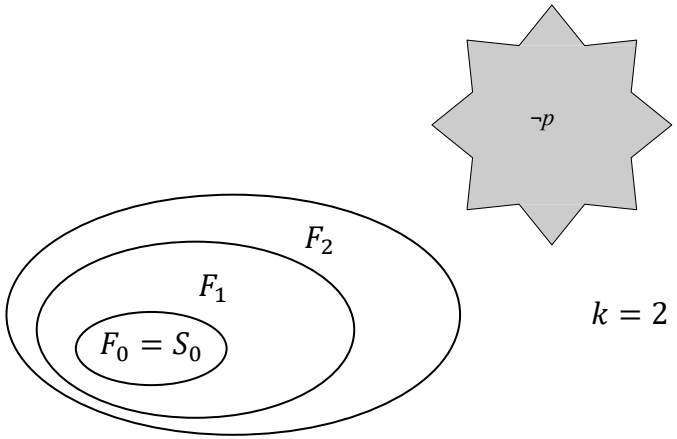
  Longer clauses – more states. Longer cubes – fewer states

Formulas $F_0, \ldots, F_k$ over V,

stored as sets of Clauses as CNF $(F_0, \ldots, F_k \subseteq S)$



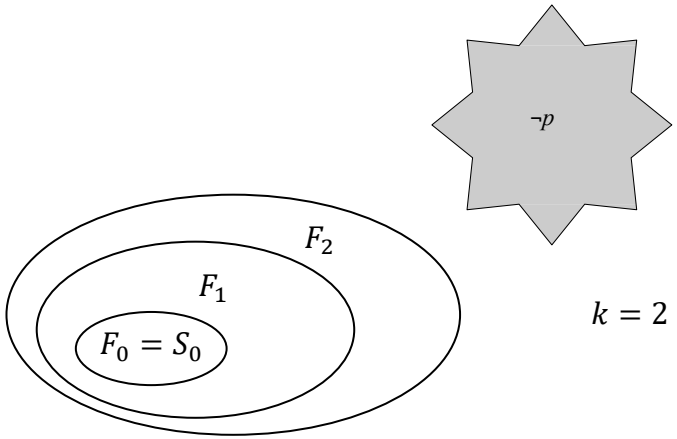
PDR: Data Structures & Invariants



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': $F_i = F_{i+1} \wedge c_{i1} \wedge \cdots \wedge c_{in}$
- I3: $F_i \rightarrow p . (F_i \subseteq p)$
- 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': $F_i = F_{i+1} \wedge c_{i1} \wedge \cdots \wedge c_{in}$
- I3: $F_i \rightarrow P$ $(F_i \subseteq P)$
- I4: $F_i \wedge R \rightarrow F'_{i+1}$ $(postimg(F_i) \subseteq F_{i+1})$

Facts. Suppose we have frames F_0 through F_k

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. There is no counterexample with k edges or less (with I1)
3. If $F_i = F_{i+1}$ then system is correct. (By I3, I4, F_i is an inductive invariant)

Storing Frames

I2': $F_i = F_{i+1} \wedge c_{i1} \wedge \cdots \wedge c_{in}$

For each frame F_i, store only Δ_i, the clauses that are new to that frame.

$F_i = F_{i+1}$ iff $\Delta_i = \emptyset$

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 := SAT($F_k \wedge R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

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

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

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

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

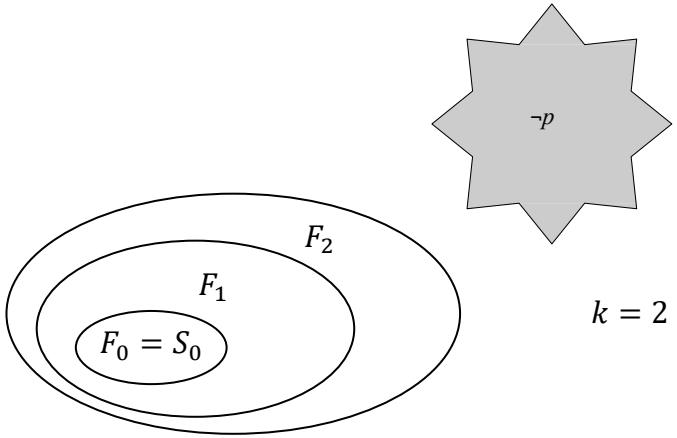
if SAT($S_0 \wedge s$) **then FAIL**

while(t := 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_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)

you are here

    while(s := 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: $\neg SAT(F_i \wedge s)$

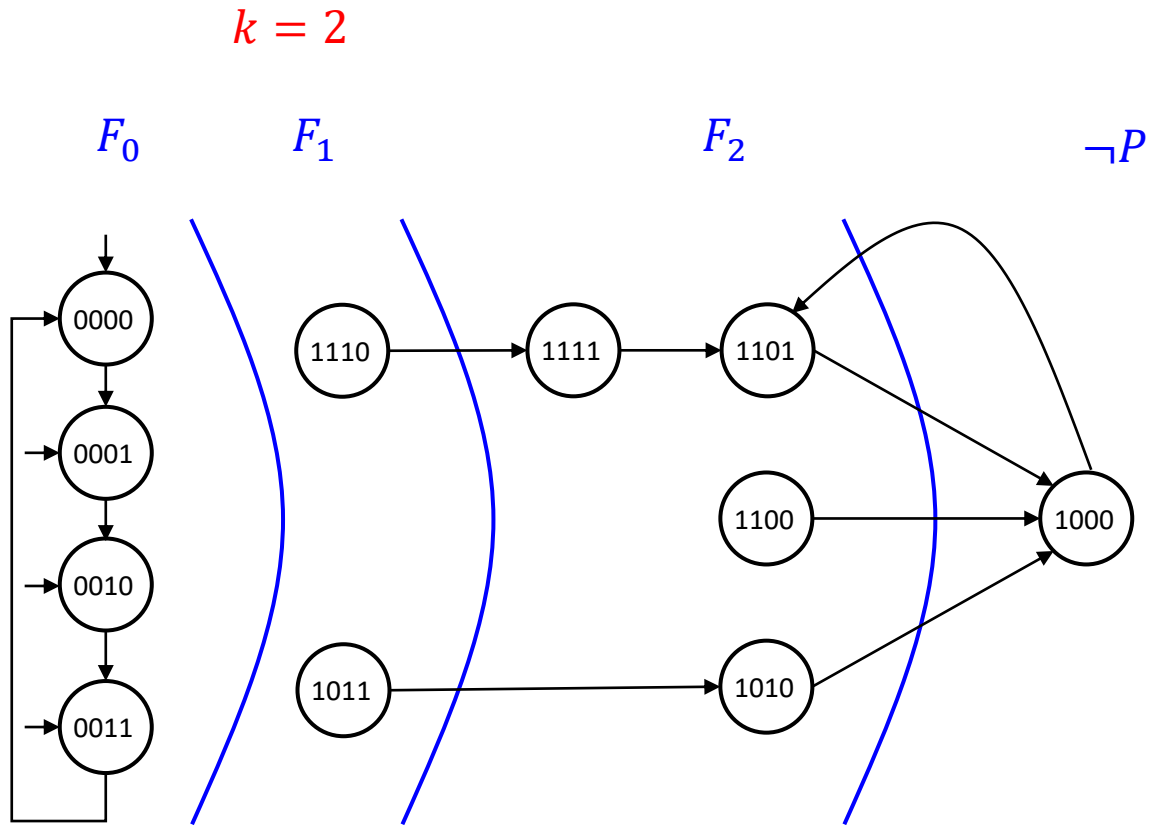
function removeBad($i \in N$, state s)

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

  while(t := 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)

you are here

    while(s := 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: $\neg SAT(F_i \wedge s)$

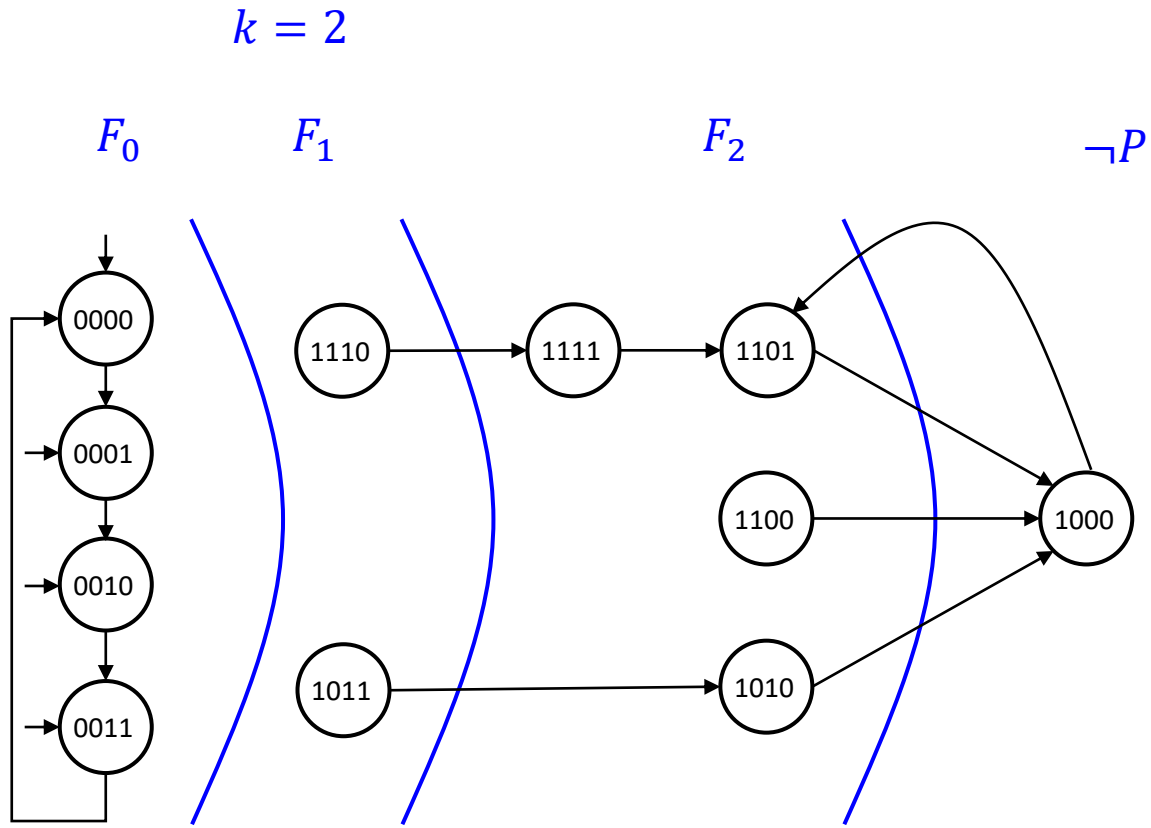
function removeBad($i \in N$, state s)

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

  while(t := 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)

you are here

    while(s := 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: $\neg SAT(F_i \wedge s)$

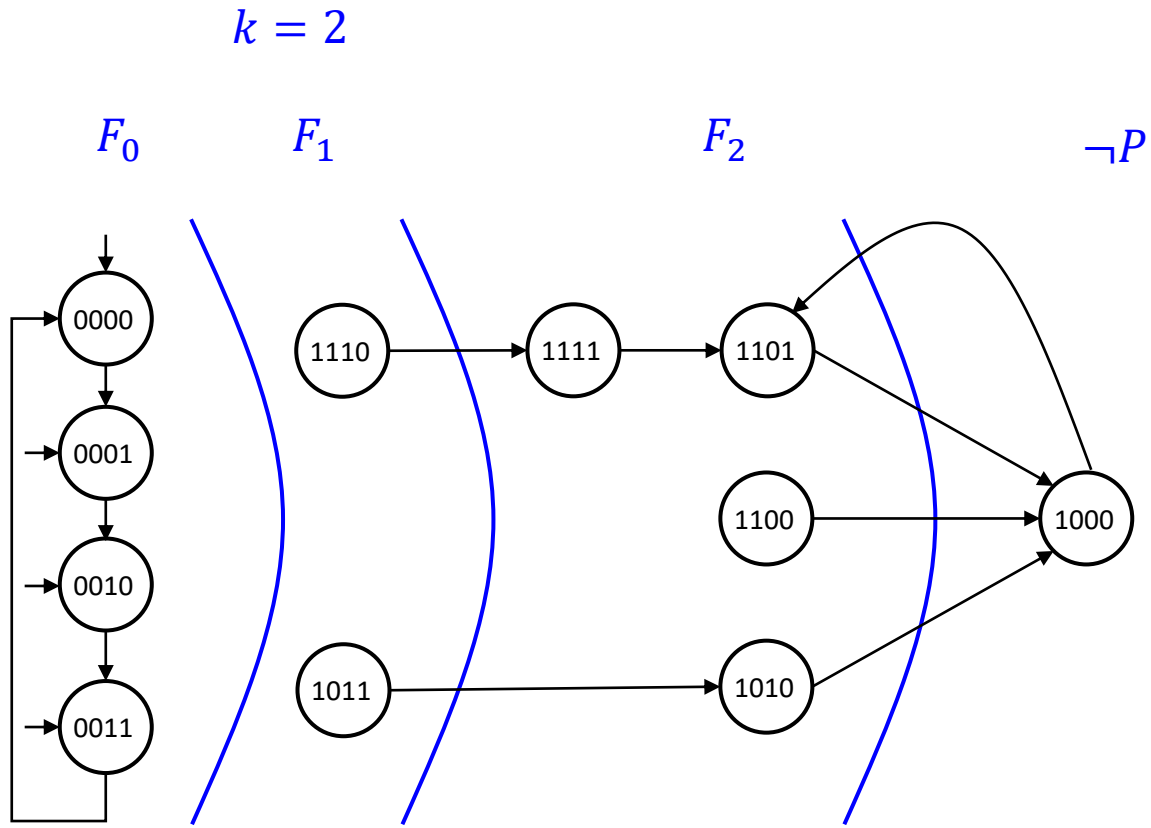
function removeBad($i \in N$, state s)

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

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

    removeBad($i-1, t$)

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



Suppose s := 1101

removeBad(2, 1101)

SAT($F_{i-1} \wedge R \wedge s'$) = FALSE

$F_2 := F_2 \wedge \neg(x_1 \wedge x_2, \neg x_3 \wedge x_4)$, same for F_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)

you are here

    while(s := 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: $\neg SAT(F_i \wedge s)$

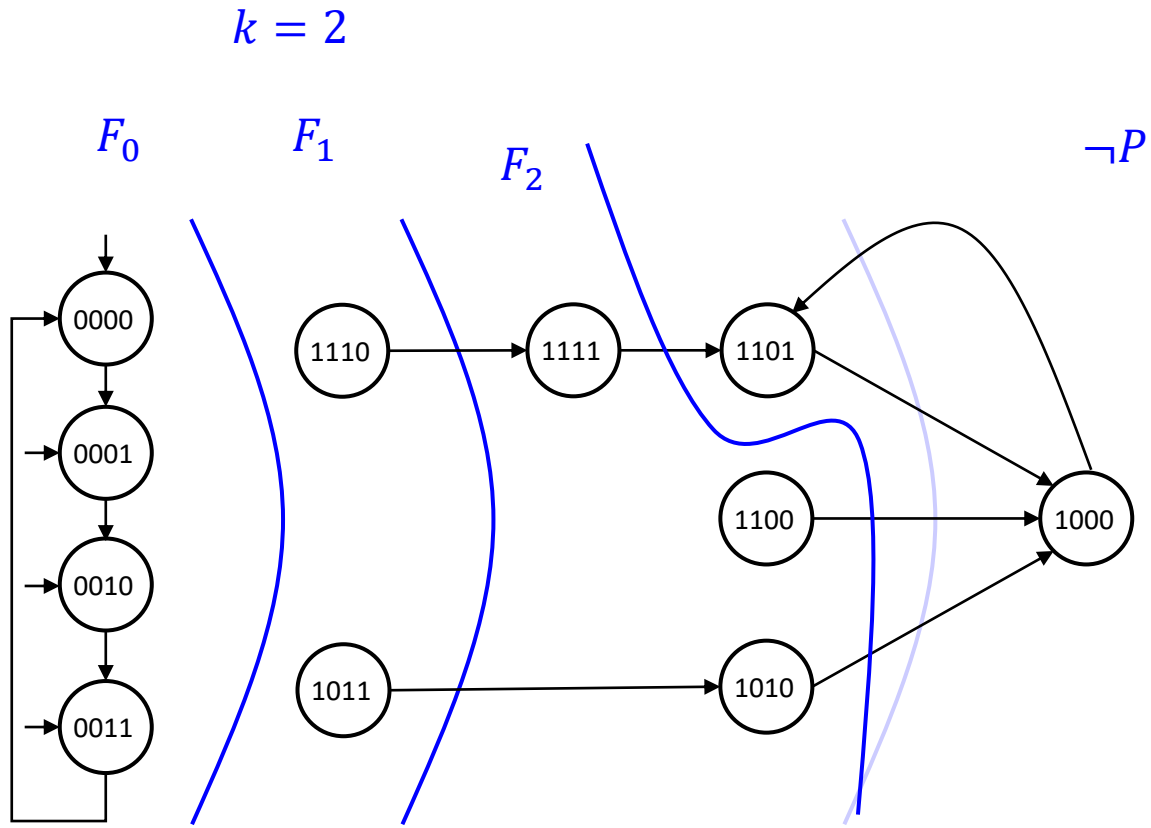
function removeBad($i \in N$, state s)

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

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

    removeBad($i-1, t$)

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



Suppose s := 1101

removeBad(2, 1101)

SAT($F_{i-1} \wedge R \wedge s'$) = FALSE

$F_2 := F_2 \wedge \neg(x_1 \wedge x_2, \neg x_3 \wedge x_4)$, same for F_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)

you are here

    while(s := 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: $\neg SAT(F_i \wedge s)$

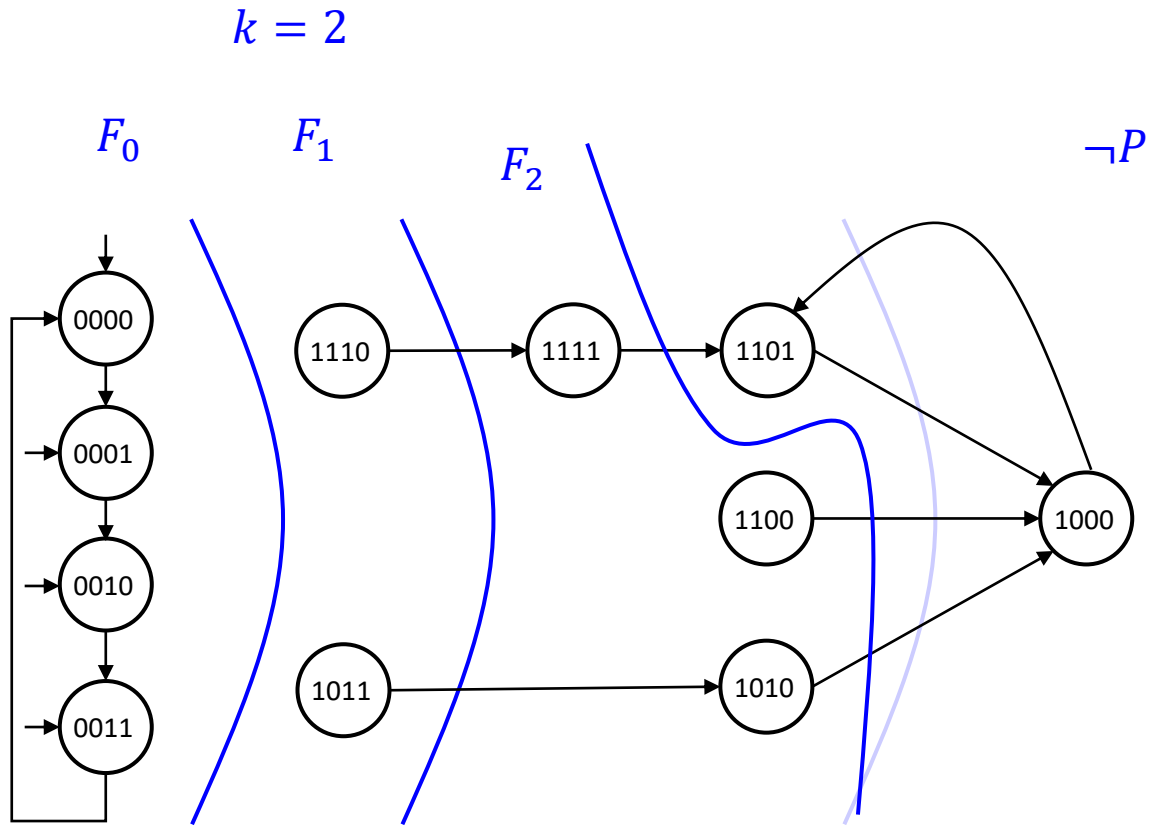
function removeBad($i \in N$, state s)

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

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

    removeBad($i-1, t$)

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



Suppose s= 1100, this state is removed from F_2

Suppose s = 1010

removeBad(2, 1010), leads to removal of 1011 from F_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 := 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: $\neg SAT(F_i \wedge s)$

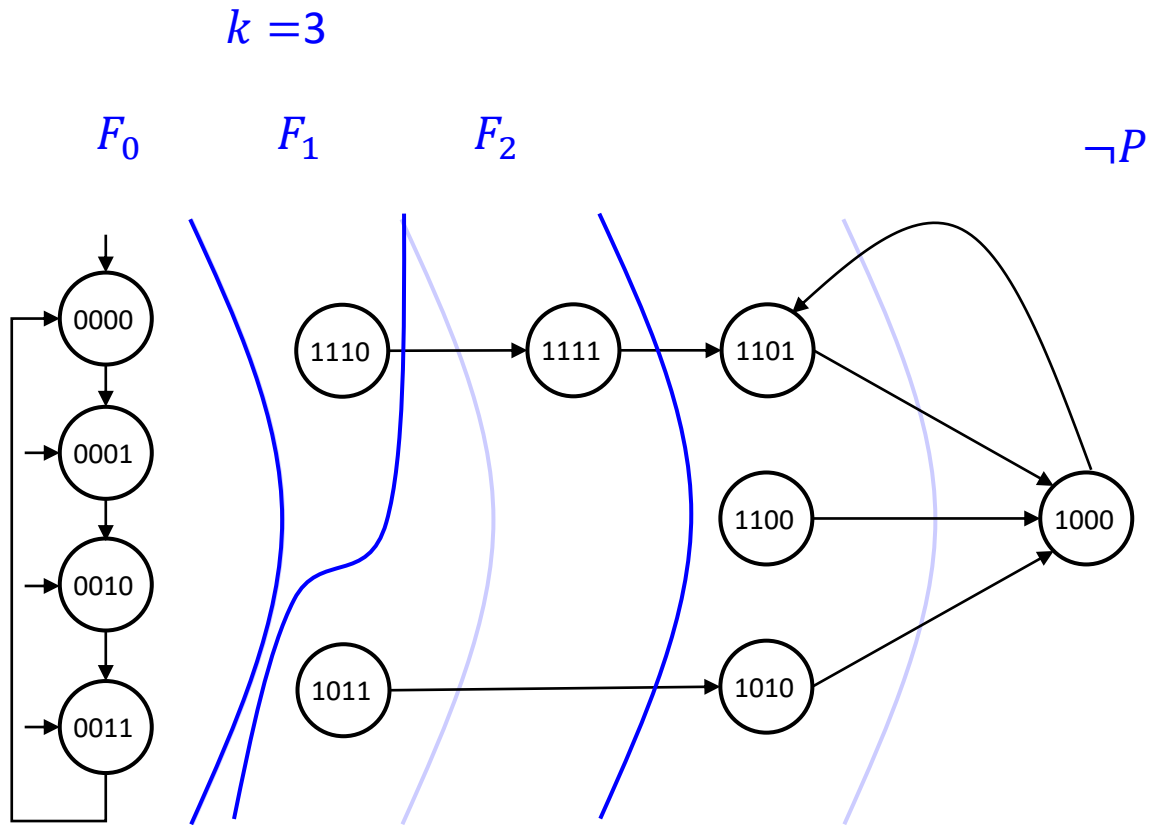
function removeBad($i \in N$, state s)

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

while(t := 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 := 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: $\neg SAT(F_i \wedge s)$

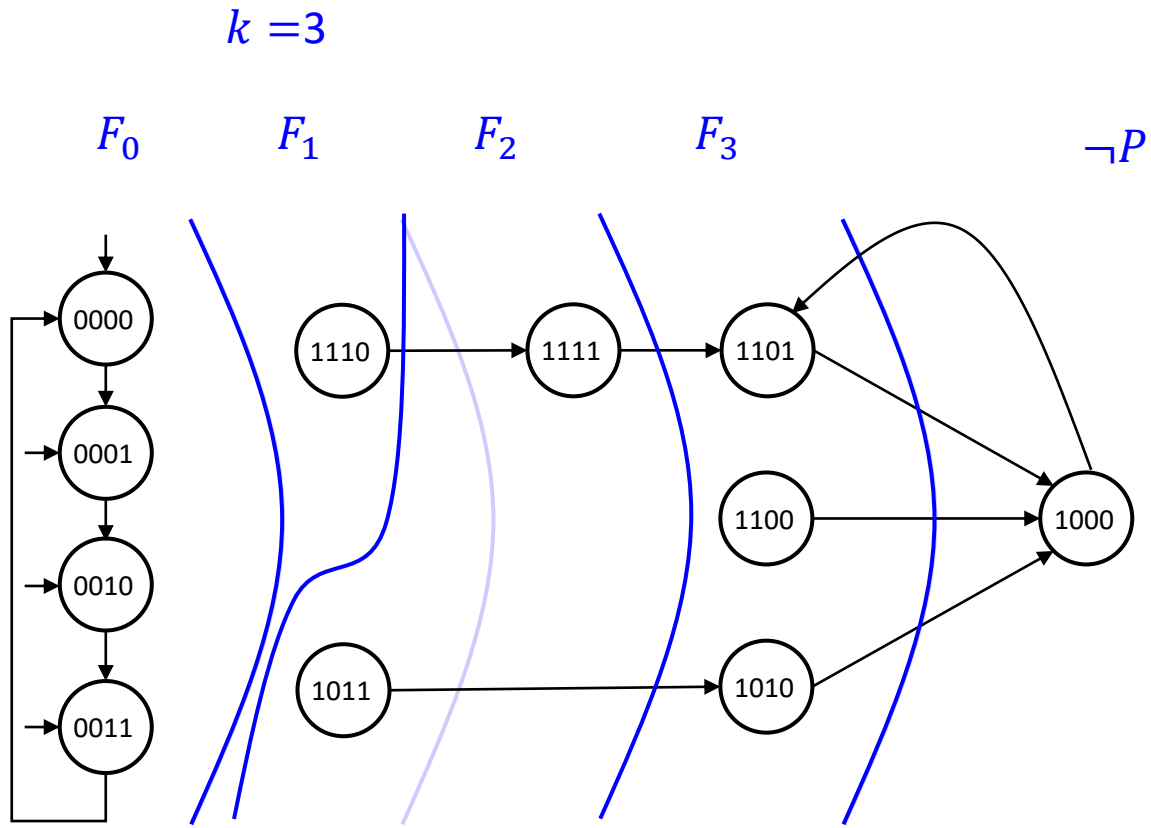
function removeBad($i \in N$, state s)

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

  while(t := 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 := 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: $\neg SAT(F_i \wedge s)$

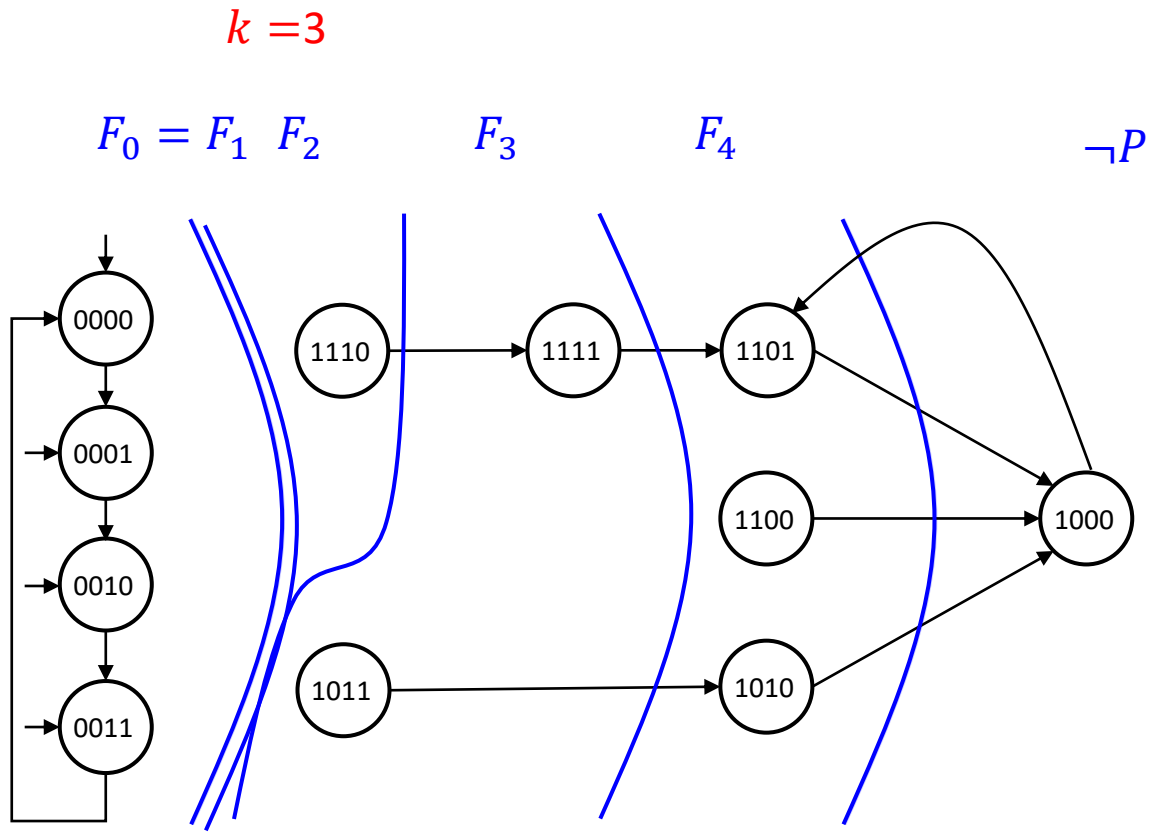
function removeBad($i \in N$, state s)

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

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

    removeBad($i-1, t$)

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

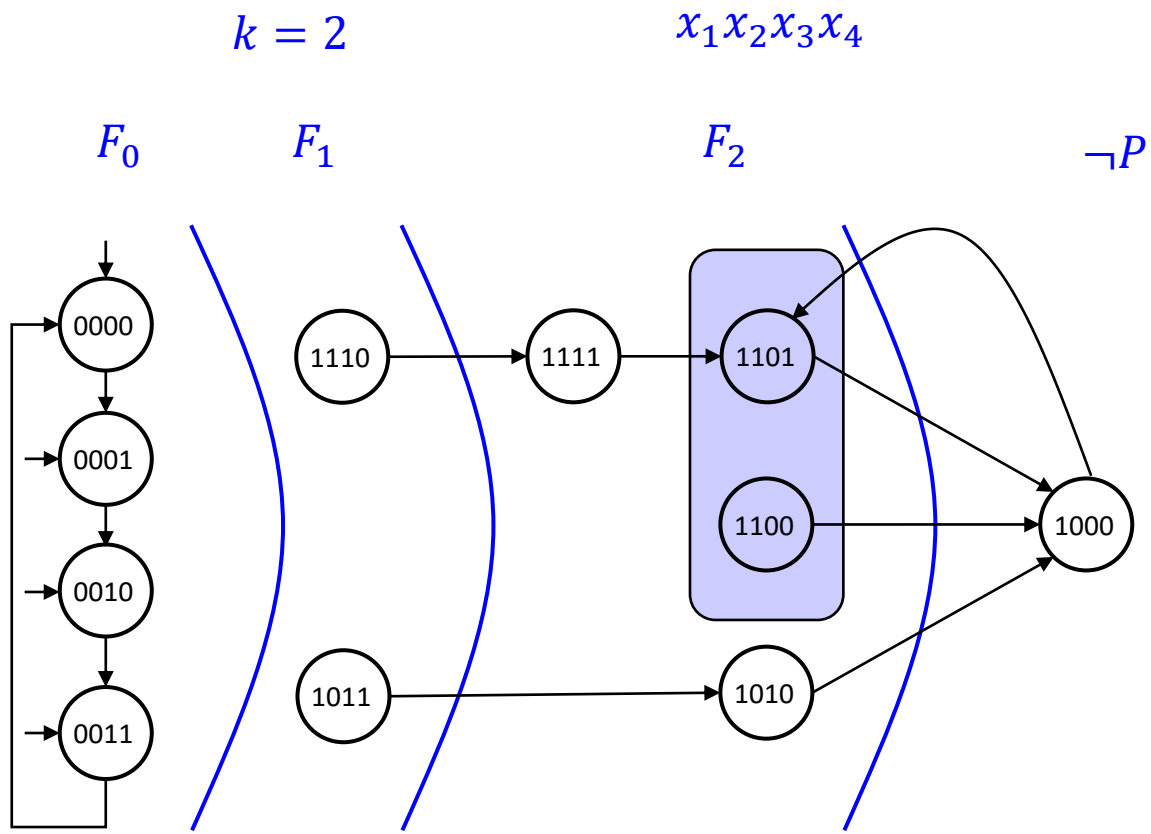


Drawback

- First version considers every state individually
- Similar states behave similarly! Example: 1100 and 1101. Generalize

  1101 to 110- = $x_1 \wedge x_2 \wedge \neg x_3$!

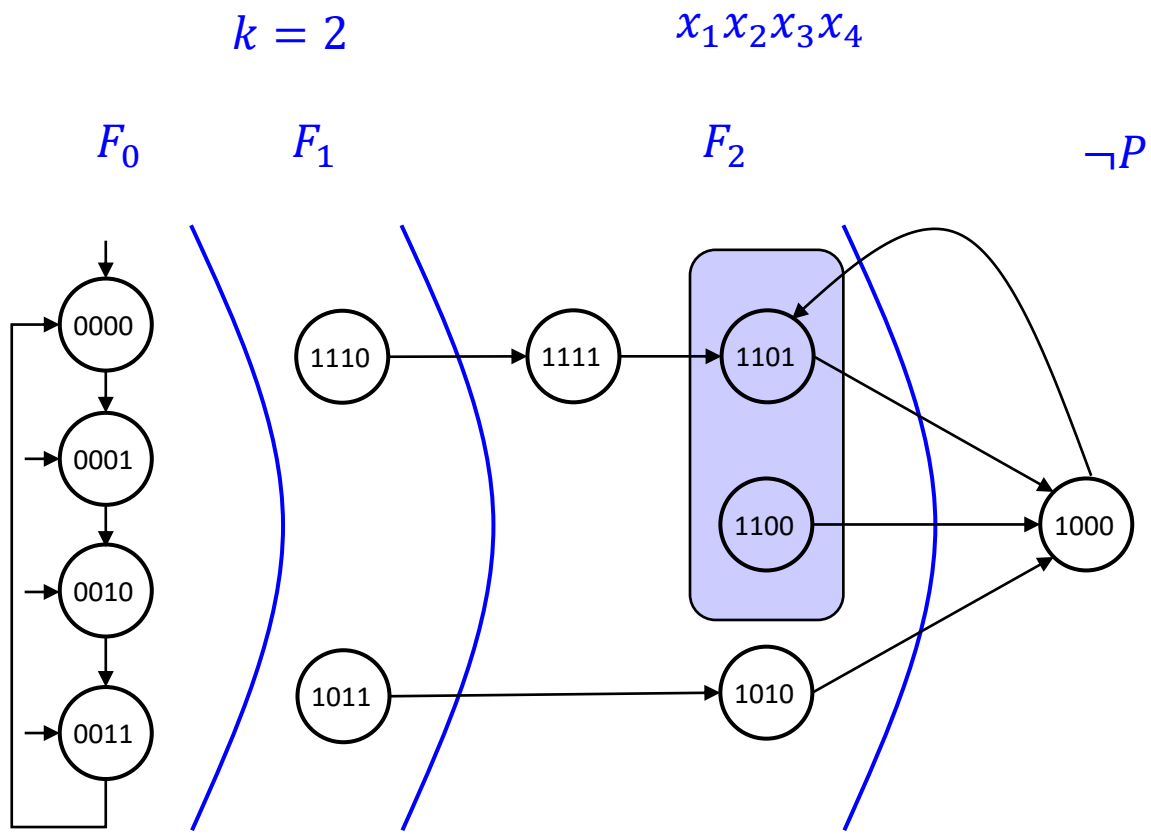
Conditions



Drawback

- First version considers every state individually
- Similar states behave similarly! Example: 1100 and 1101. Generalize

  1101 to 110- = $x_1 \wedge x_2 \wedge \neg x_3$!



Conditions

- UNSAT($F_1 \wedge R \wedge x'_1 \wedge x'_2 \wedge \neg x'_3$)
- 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 := 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: $\neg SAT(F_i \wedge s)$

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

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

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

removeBad($i-1, t$)

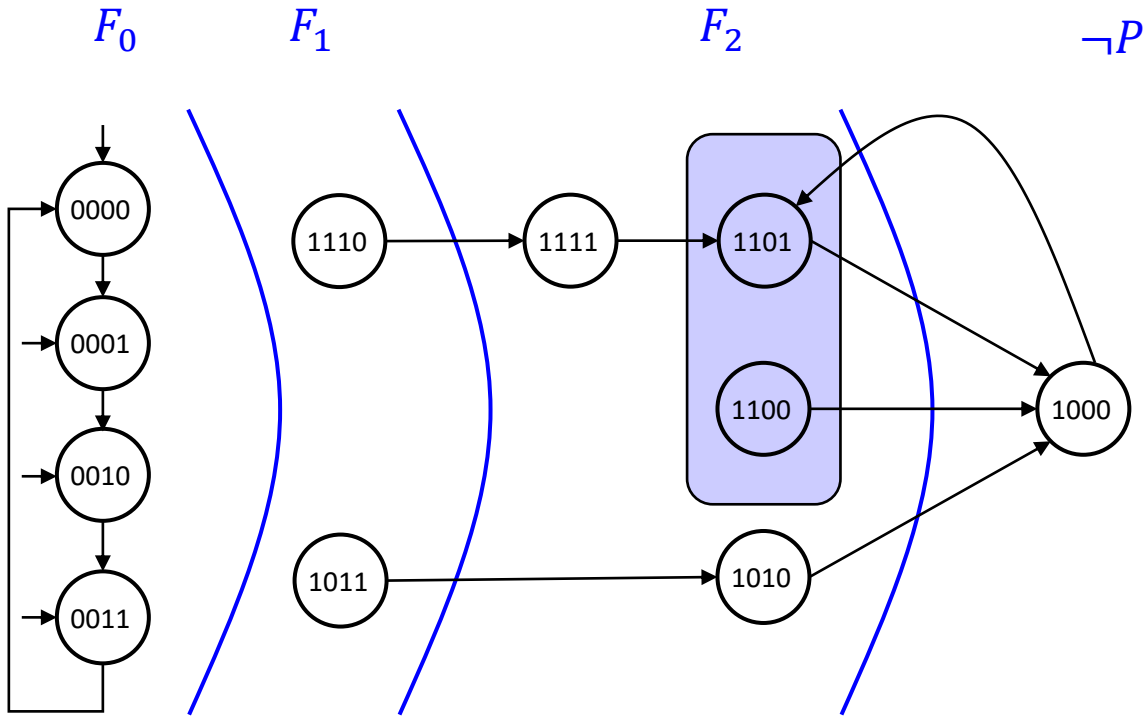
g := generalizeNaive(i, s)

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

function generalizeNaive(i, state s)

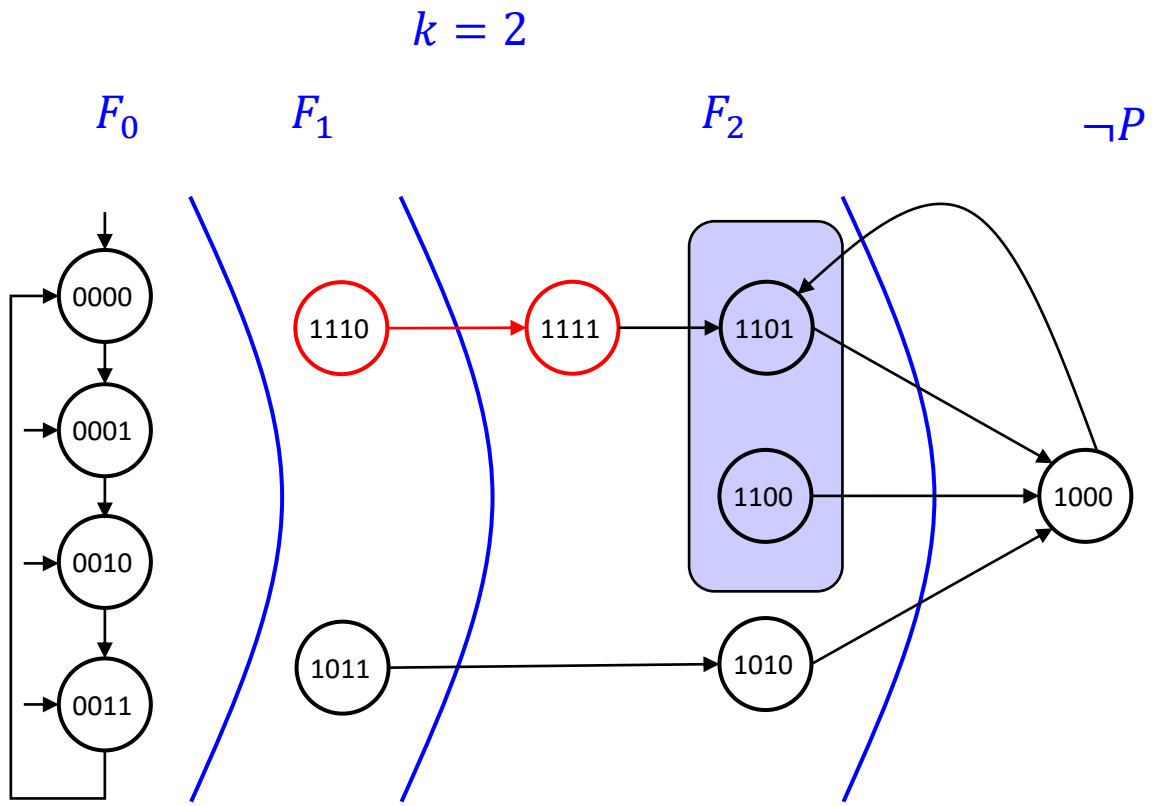
return a shortest cube c such that

- $c \leftarrow s$
- $\neg \mathrm{SAT}(\mathrm{F_{i-1}} \wedge R \wedge c')$
- $\neg \mathrm{SAT}(S_0 \wedge c)$



Generalize Further?

- We can generalize to 110-
- Can we generalize to 11--?
  - NO: for $c = x_1 \wedge x_2$, we have SAT($F_1 \wedge R \wedge c'$)
- Transition 1110 $\rightarrow$ 1111 is the problem



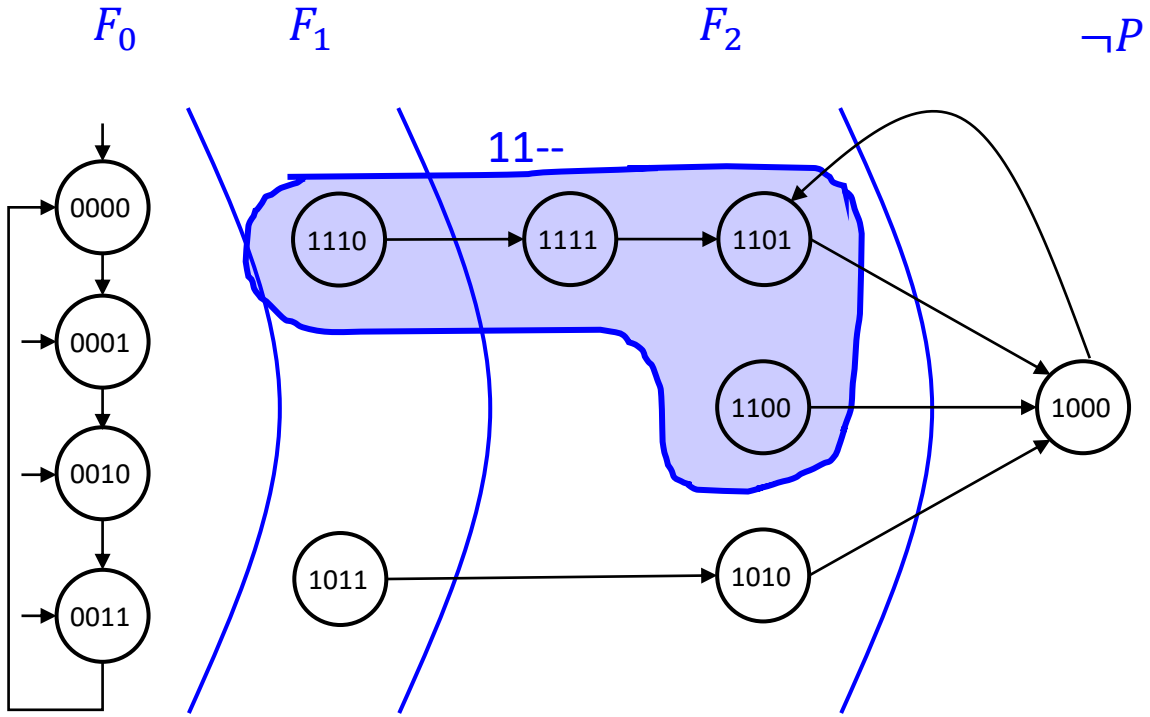
Relative Inductiveness

shortest cube c such that

- $c \leftarrow s$
- $\neg\mathrm{SAT}(\neg c \wedge \mathrm{F}_1 \wedge R \wedge c')$
- $\neg\mathrm{SAT}(S_0 \wedge c)$

$\neg c$ is relative inductive wrt F_1

Why?



Relative Inductiveness

shortest cube c such that

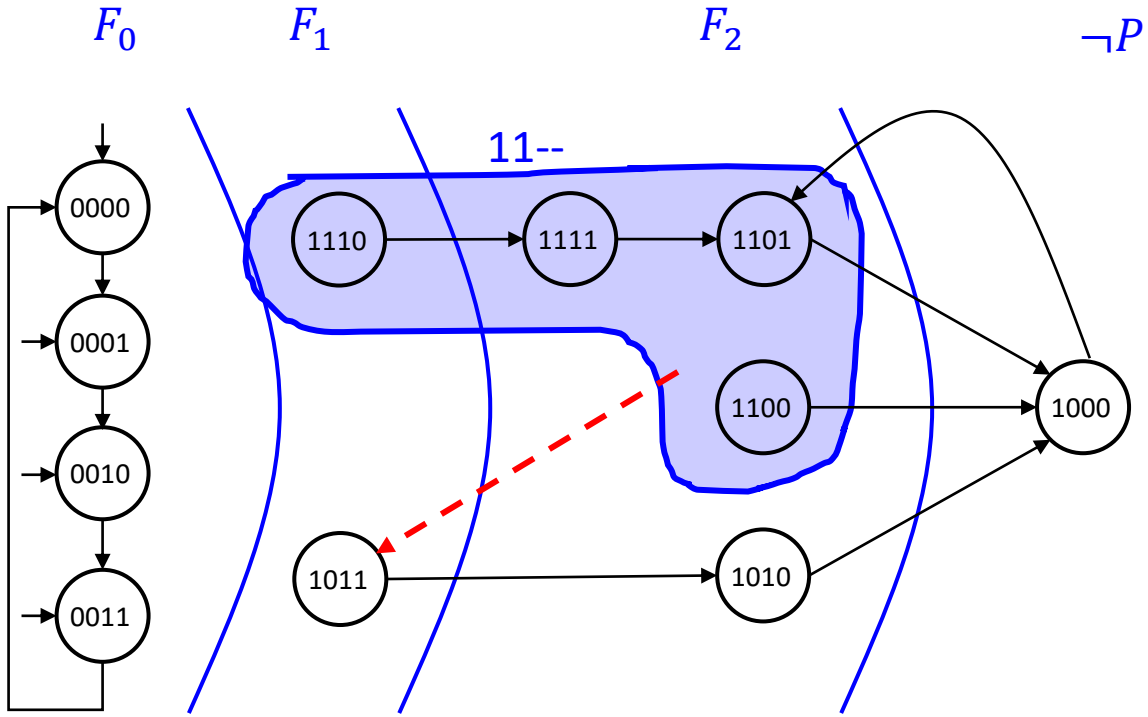
- $c \leftarrow s$
- $\neg\mathrm{SAT}(\neg c \wedge \mathrm{F}_1 \wedge R \wedge c')$
- $\neg\mathrm{SAT}(S_0 \wedge c)$

$\neg c$ is relative inductive wrt F_1

Why?

Because we need to maintain

I4: $F_i \wedge R \rightarrow F'_{i+1}$



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 := 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: $\neg SAT(F_i \wedge s)$

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

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

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

removeBad($i-1, t$)

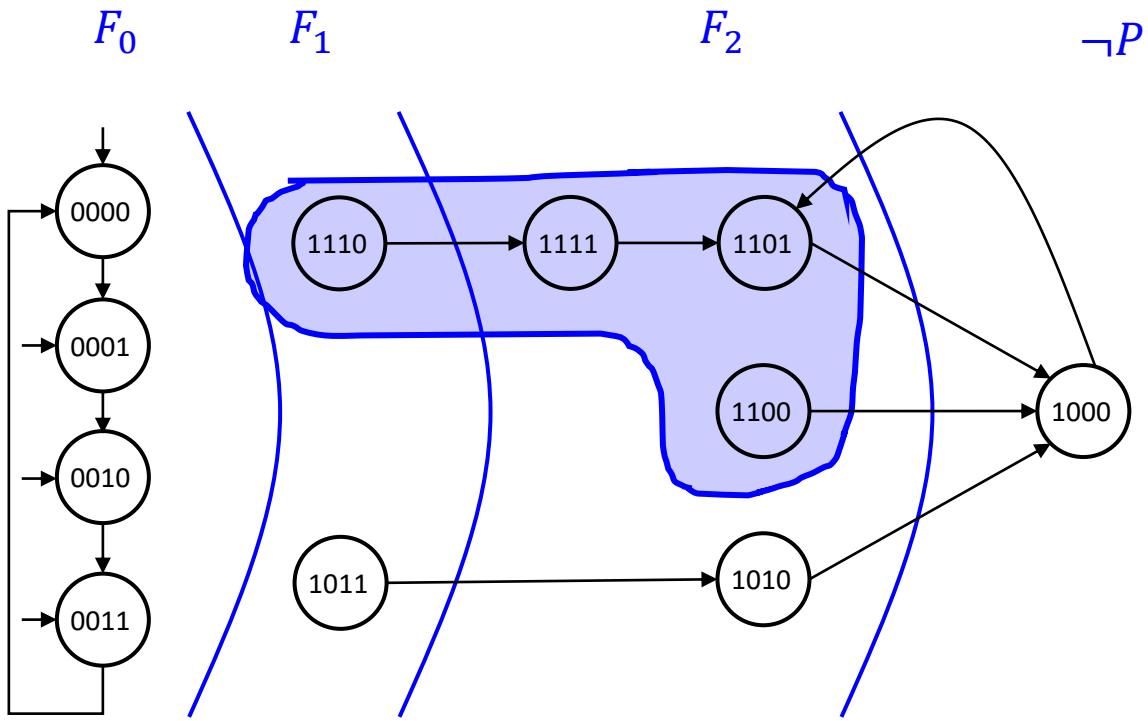
g := generalize(i,s)

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

function generalize(i, state s)

return a shortest cube c such that

- $c \leftarrow s$
- $\neg \mathrm{SAT}(\neg c \wedge \mathrm{F_{i-1}} \wedge R \wedge c')$
- $\neg \mathrm{SAT}(S_0 \wedge c)$



Generalization

function generalize(i, state s)

$c := s$

while c changes

let $l_1, \ldots, l_n$ 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 $\neg \mathrm{SAT}(\neg c \wedge \mathrm{F_{i-1}} \wedge R \wedge c)$
$\wedge \neg \mathrm{SAT}(S_0 \wedge c)$

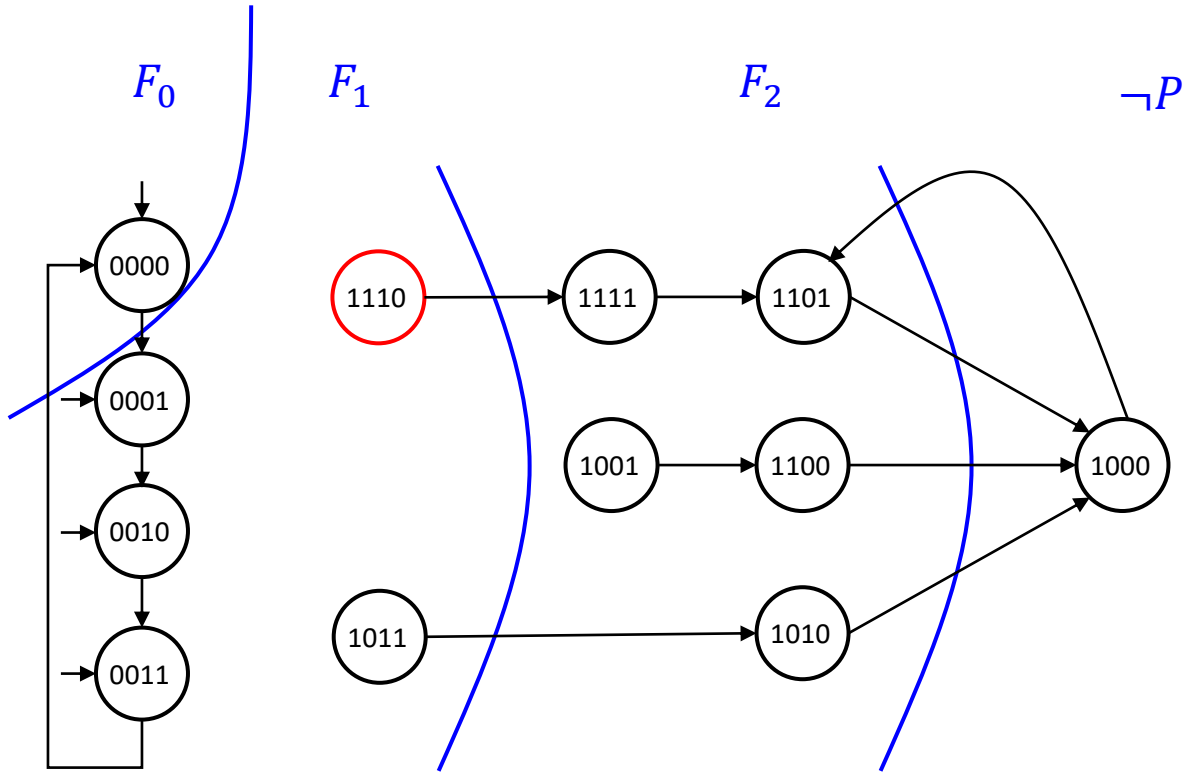
Propagate Clauses

Suppose you are removing 1110 from F_1

You can generalize 1110 to 1---

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

Can we do the same for F_2?



Propagate Clauses

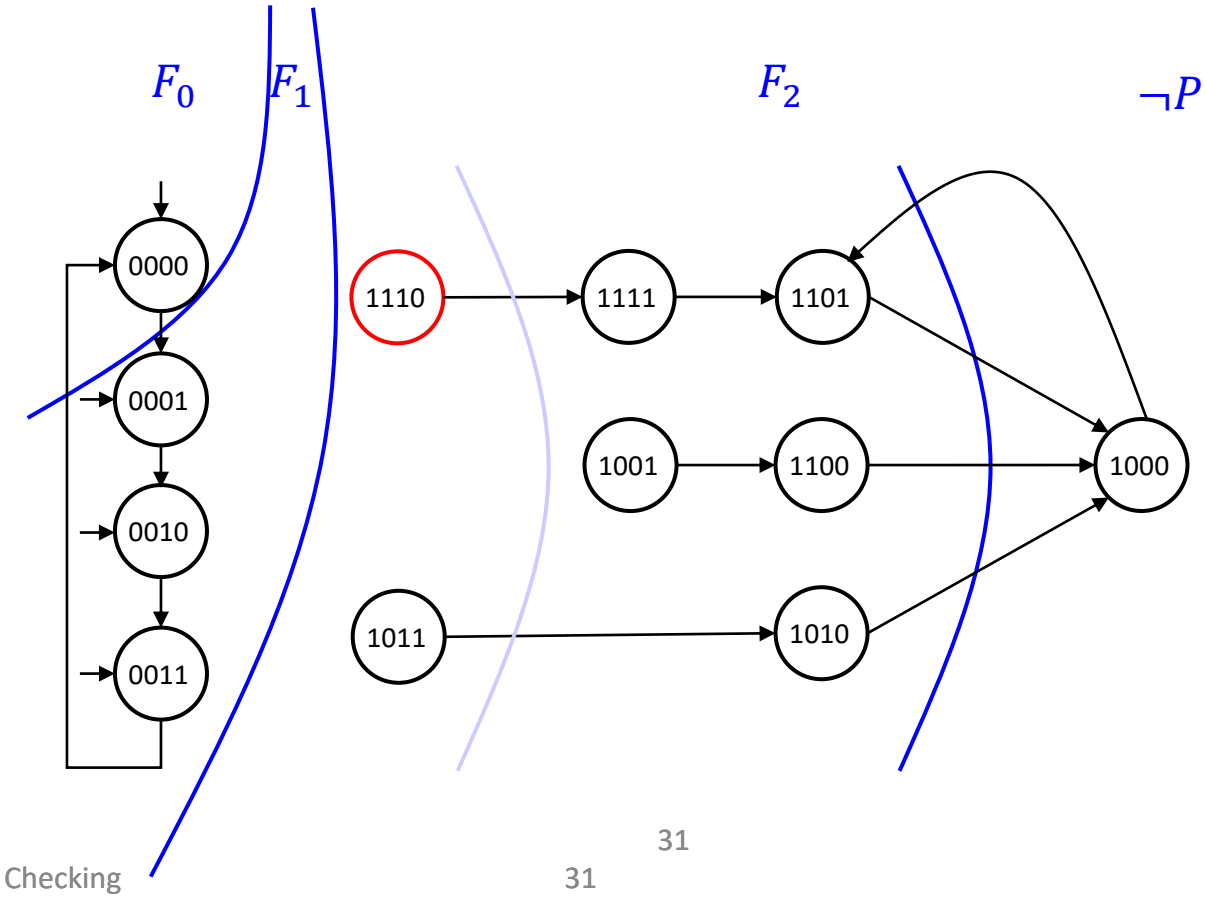
Suppose you are removing 1110 from F_1

You can generalize 1110 to 1---

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

$F_2 \wedge x_1 \notin img(F_1)$, so we can add $\neg x_1$ to F_2

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 := 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: $\neg SAT(F_i \wedge s)$

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

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

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

removeBad($i-1, t$)

g := generalize(i,s)

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

function gesneralize(i, 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 $\neg \mathrm{SAT}(F_i \wedge R \wedge \neg c')$

$F_{i+1} := F_{i+1} \wedge c$

Further Ideas

- This version is simplified, doesn’t find long counterexamples quickly
- Equivalence of frames = syntactic check
  - Use implication and subsumption to simplify clauses

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-Bassed 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