VTSA summer school 2015

Exploiting SMT for Verification of Infinite-State Systems

3. SMT-based Verification with IC3

Alberto Griggio
Fondazione Bruno Kessler – Trento, Italy

Outline

Introduction

IC3 for finite-state systems

SMT-based IC3 for infinite-state systems

IC3 for LTL verification

Introduction

- IC3 very successful SAT-based model checking algorithm
  - **I**ncremental **C**onstruction
  - of **I**nductive **C**lauses
  - for **I**ndubitable **C**orrectness

- Key principles:
  - Verification by induction
  - Inductive invariant built incrementally
    - by discovering (relatively-)inductive clauses
  - Exploiting efficient SAT solvers

Introduction

- IC3 has been further generalized to SMT in various ways

- We will look in some detail at one such generalization, called

  IC3 with Implicit Predicate Abstraction (IC3-IA)

  - Exploits several features of modern SMT solvers that we have discussed so far
    - Incremental solving
    - Assumptions and unsatisfiable cores
    - Interpolation

- A “hands-down” approach
  - We will build a (simple) real implementation on top of MathSAT

Proofs by Induction

- Given transition system $\langle I(X), T(X, X') \rangle$ and property $P(X)$
  - Base case (initiation):

$$I(X) \models P(X)$$

  - Inductive step (consectution):

$$P(X) \wedge T(X, X') \models P(X')$$

- Typically however, P is not inductive
  - Find an inductive invariant $Inv(X)$, stronger than P
    - $I(X) \models Inv(X)$
    - $Inv(X) \wedge T(X, X') \models Inv(X')$
    - $Inv(X) \models P(X)$

Outline

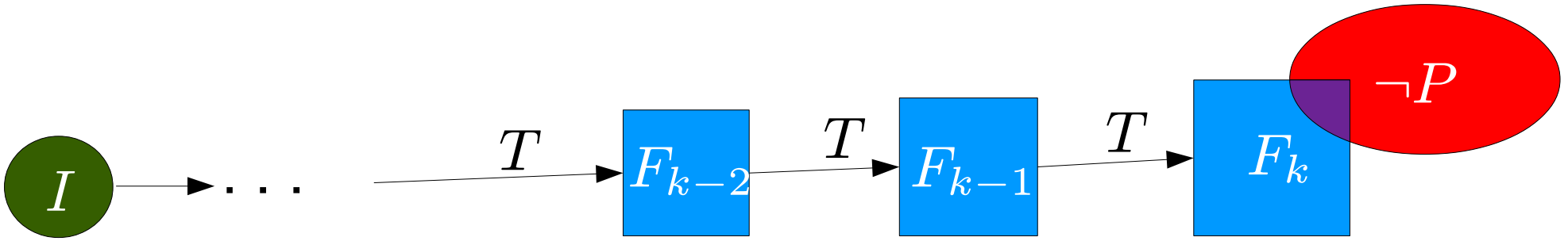
Introduction

IC3 for finite-state systems

SMT-based IC3 for infinite-state systems

IC3 for LTL verification

A (very) high level view of IC3



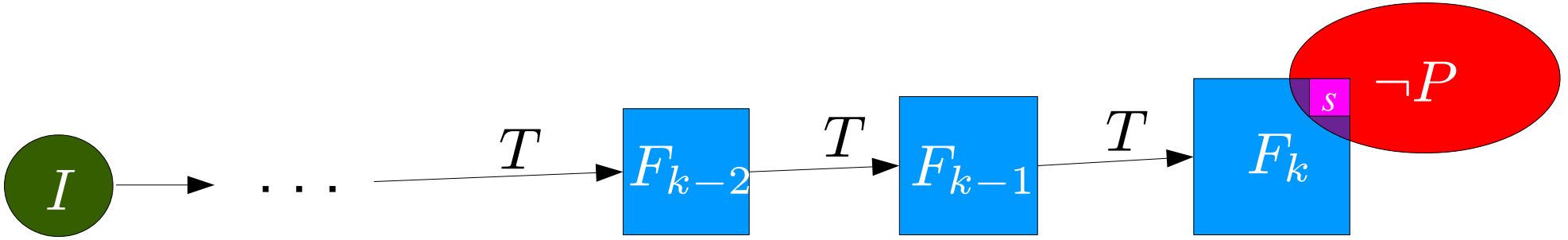
- Given a symbolic transition system and invariant property P, build an inductive invariant F s.t. $F \models P$
- Trace of formulae $F_0(X) \equiv I, \ldots, F_k(X)$ s.t:
  - for $i > 0$, F_i is a set of clauses overapproximation of states reachable in up to i steps

$$F_{i+1} \subseteq F_i \ (\text{so } F_i \models F_{i+1})$$

$$F_i \wedge T \models F'_{i+1}$$

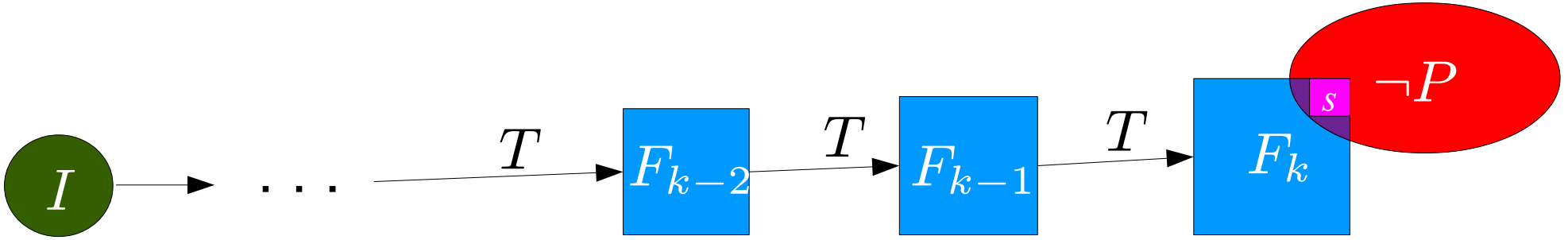
$$\text{for all } i < k, F_i \models P$$

A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \wedge \neg s \wedge T \wedge s'$
    (i.e., check if $F_{k-1} \wedge \neg s \wedge T \models \neg s'$)

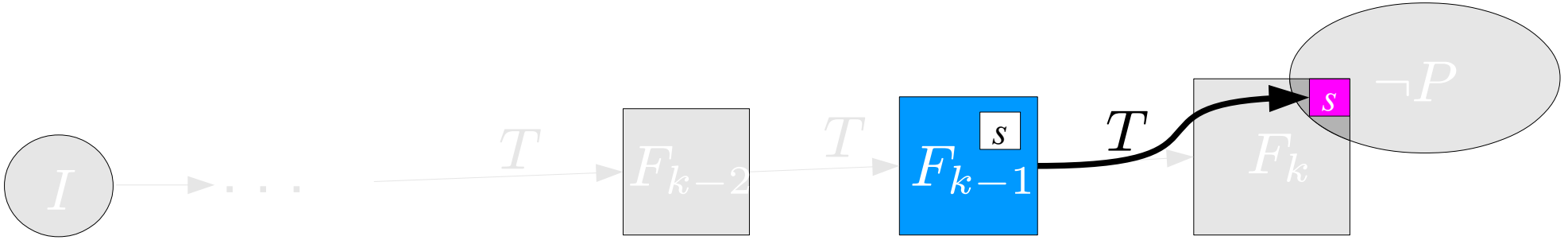
A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \wedge \neg s \wedge T \wedge s'$
    (i.e., check if $F_{k-1} \wedge \neg s \wedge T \models \neg s'$)

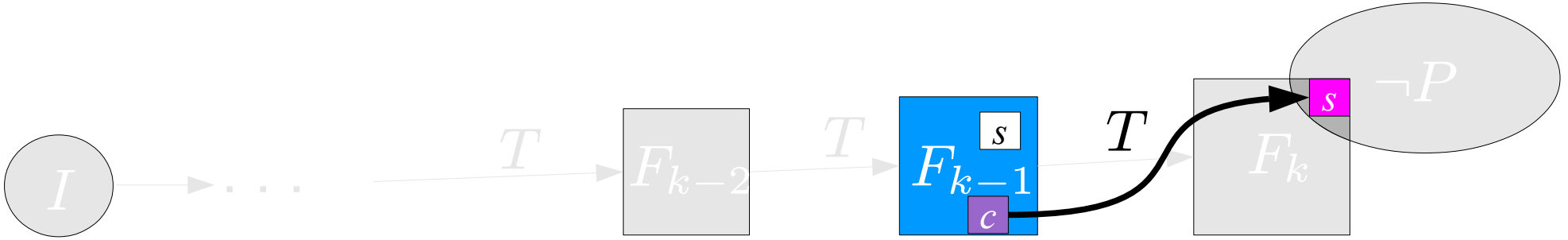
Check if s is inductive relative to F_{k-1}

A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \wedge \neg s \wedge T \wedge s'$
    (i.e., check if $F_{k-1} \wedge \neg s \wedge T \models \neg s'$)

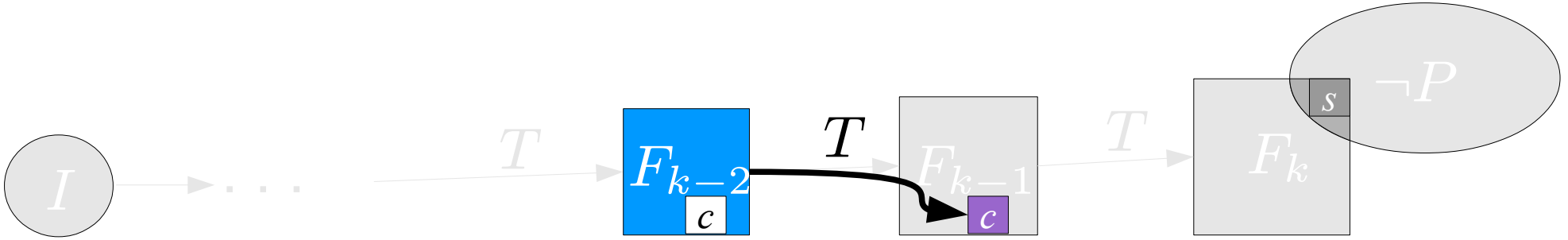
A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \wedge \neg s \wedge T \wedge s'$
    - SAT: s is reachable from $F_{k-1} \wedge \neg s$ in 1 step
    - Get a cube c in the preimage of s and try (recursively) to prove it unreachable from F_{k-2}, …
      - c is a counterexample to induction (CTI)

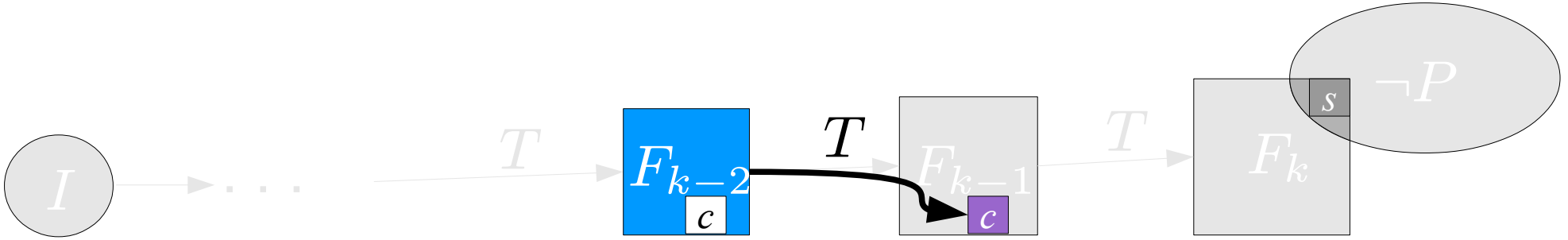
If I is reached, counterexample found

A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-2} \wedge \neg s \wedge T \wedge s'$

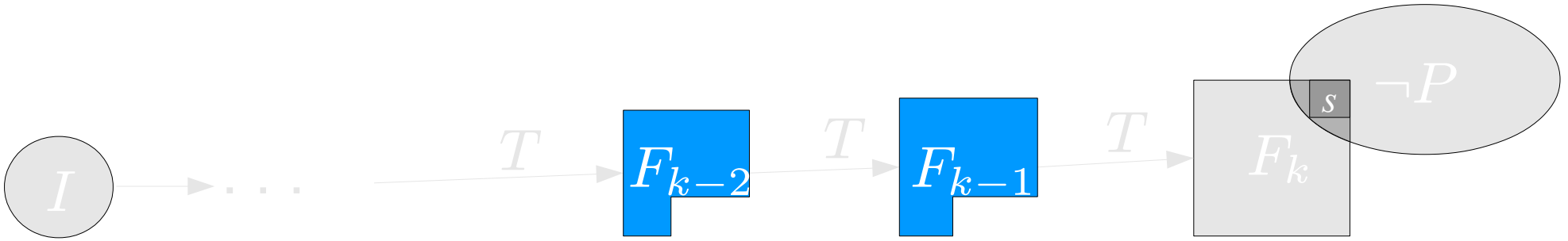
A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-2} \wedge \neg s \wedge T \wedge s'$
    - UNSAT: $\neg c$ is inductive relative to F_{k-2}
    - Generalize c to g and block by adding $\neg g$ to $F_{k-1}, F_{k-2}, \ldots, F_1$

$$F_{k-2} \wedge \neg c \wedge T \models \neg c'$$

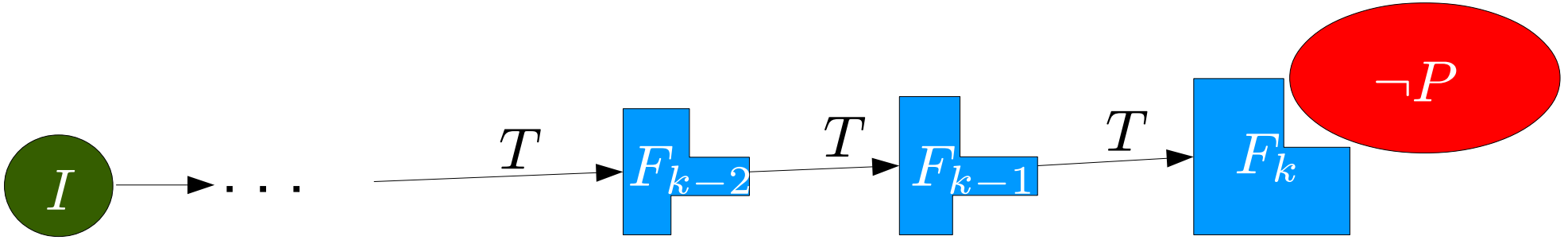
A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-2} \wedge \neg s \wedge T \wedge s'$
    - UNSAT: $\neg c$ is inductive relative to F_{k-2}
    - Generalize c to g and block by adding $\neg g$ to $F_{k-1}, F_{k-2}, \ldots, F_1$

$$F_{k-2} \wedge \neg c \wedge T \models \neg c'$$

A (very) high level view of IC3



Propagation: extend trace to F_{k+1} and push forward clauses

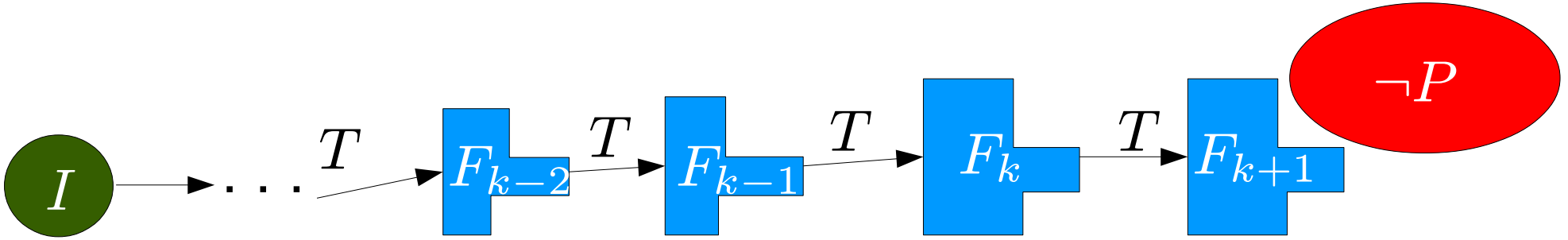
For each i and each clause $c \in F_i$:

Call SAT solver on $F_i \wedge T \wedge \neg c'$

If UNSAT, add c to F_{i+1}

$$F_i \wedge T \models c'$$

A (very) high level view of IC3



Propagation: extend trace to F_{k+1} and push forward clauses

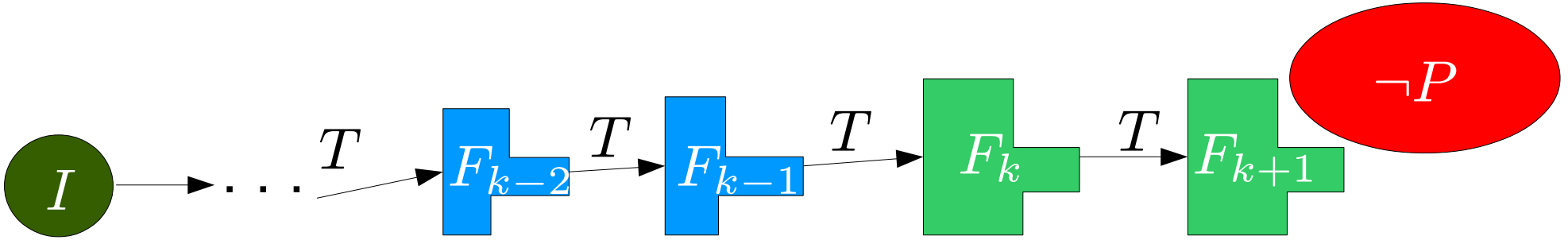
For each i and each clause $c \in F_i$:

Call SAT solver on $F_i \wedge T \wedge \neg c'$

If UNSAT, add c to F_{i+1}

$$F_i \wedge T \models c'$$

A (very) high level view of IC3



Propagation: extend trace to F_{k+1} and push forward clauses

For each i and each clause $c \in F_i$:

Call SAT solver on $F_i \wedge T \wedge \neg c'$

If UNSAT, add c to F_{i+1}

$$F_i \wedge T \models c'$$

If $\boxed{F_i \equiv F_{i+1}}$, P is proved,
otherwise start another round of blocking and propagation

IC3 pseudo-code

```
bool IC3(I, T, P):
    trace = [I]   # first elem of trace is init formula
    trace.push()  # add a new frame
    while True:
        # blocking phase
        while is_sat(trace.last() & ~P):
            c = extract_cube() # c |= trace.last() & ~P
            if not rec_block(c, trace.size()-1):
                return False # counterexample found

        # propagation phase
        trace.push()
        for i=1 to trace.size()-1:
            for each cube c in trace[i]:
                if not is_sat(trace[i] & ~c & T & c'):
                    trace[i+1].append(c)
            if trace[i] == trace[i+1]:
                return True # property proved
```

IC3 pseudo-code

```
bool rec_block(s, i):
    if i == 0:
        return False  # reached initial states
    while is_sat(trace[i-1] & ~s & T & s'):
        c = get_predecessor(i-1, T, s')
        if not rec_block(c, i-1):
            return False
    g = generalize(~s, i)
    trace[i].append(g)
    return True
```

Correctness (sketch)

- Consider the formula $F_{k-1} \wedge T \wedge s'$ where *s* is a bad cube
  - If UNSAT, then F_{k-1} is strong enough to block *s*
  - Since $F_i \wedge T \models F'_{i+1}$, then *s* is unreachable in *k* steps or less
  - Since $F_i \models F_{i+1}$, then we can add *s* to all $F_j, j \leq k$

Correctness (sketch)

- Consider the formula $F_{k-1} \wedge T \wedge s'$ where *s* is a bad cube
  - If UNSAT, then F_{k-1} is strong enough to block *s*
  - Since $F_i \wedge T \models F'_{i+1}$, then *s* is unreachable in *k* steps or less
  - Since $F_i \models F_{i+1}$, then we can add *s* to all $F_j, j \leq k$
- Consider now the relative induction check $F_{k-1} \wedge \neg s \wedge T \wedge s'$
  - We know that $I \equiv F_0 \not\models s$ because $I \models P$ (base case)
  - Since $F_i \models F_{i+1}$, then we know that $\neg s$ holds up to *k*

Correctness (sketch)

- Consider the formula $F_{k-1} \wedge T \wedge s'$ where *s* is a bad cube
  - If UNSAT, then F_{k-1} is strong enough to block *s*
  - Since $F_i \wedge T \models F'_{i+1}$, then *s* is unreachable in *k* steps or less
  - Since $F_i \models F_{i+1}$, then we can add *s* to all $F_j, j \leq k$
- Consider now the relative induction check $F_{k-1} \wedge \neg s \wedge T \wedge s'$
  - We know that $I \equiv F_0 \not\models s$ because $I \models P$ (base case)
  - Since $F_i \models F_{i+1}$, then we know that $\neg s$ holds up to *k*
- Propagation: for each $c \in F_i$, check $F_i \wedge T \wedge \neg c'$
  - we know that *c* holds up to *i*, if UNSAT then it holds up to *i+1*
  - since $F_i \models F_{i+1}$, $F_i \wedge T \models F'_{i+1}$ and $F_i \models P$, if $F_i \equiv F_{i+1}$ then the fixpoint is an inductive invariant

Inductive Clause Generalization

- Crucial step of IC3
- Given a relatively inductive clause $c \stackrel{\text{def}}{=} \{l_1, \ldots, l_n\}$ compute a generalization $g \subseteq c$ that is still inductive

$$F_{i-1} \wedge T \wedge g \models g' \qquad (1)$$

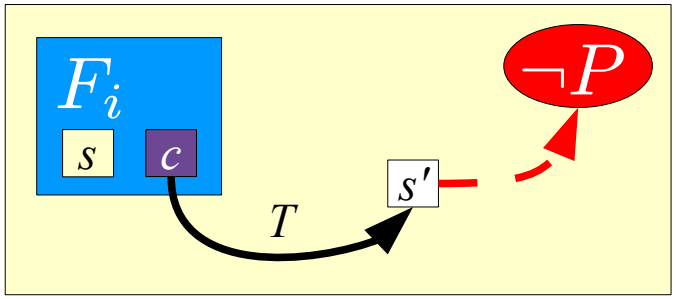
- Drop literals from c and check that (1) still holds
  - Accelerate with unsat cores returned by the SAT solver
    - Using SAT under assumptions
- However, make sure the base case still holds
  - If $I \not\models c \setminus \{l_j\}$, then l_j cannot be dropped

Simple iterative generalization

```
void indgen(c, i):
    done = False
    for iter = 1 to max_iters:
        if done:
            break
        done = True
        for each l in c:
            cand = c \ {l}
            if not is_sat(I & cand) and
               not is_sat(trace[i] & ~cand & T & cand'):
                c = get_unsat_core(cand)
                rest = cand \ c
                while is_sat(I & c):
                   l1 = rest.pop()
                   c.add(l1)
                done = False
                break
```

CTI computation

- When $F_i \wedge \neg s \wedge T \wedge s'$ is satisfiable:
  - s reaches $\neg P$ in k-i steps
  - s can be reached from F_i in 1 step
    - strengthen F_i by blocking cubes c in the preimage of s



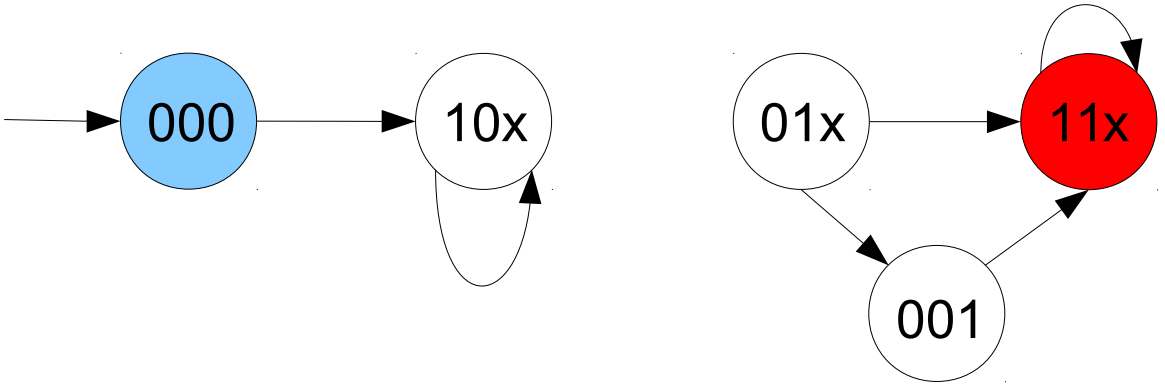
- Extract CTI c from the SAT assignment
  - And generalize to represent multiple bad predecessors
  - Use unsat cores, exploiting a functional encoding of the transition relation
    - If T is functional, then $c \wedge \text{inputs} \wedge T \models s'$
    - check $\text{inputs} \wedge T \wedge \neg s'$ under assumptions c

SAT-based CTI generalization

```
void generalize_cti(cti, inputs, next):
    for i = 1 to max_iters:
        b = is_sat(cti & inputs & T & ~next')
        assert not b # assume T to be functional
        c = get_unsat_core(cti)
        if should_stop(c, cti):
            break
        cti = c
```

Example

No counterexamples of length 0



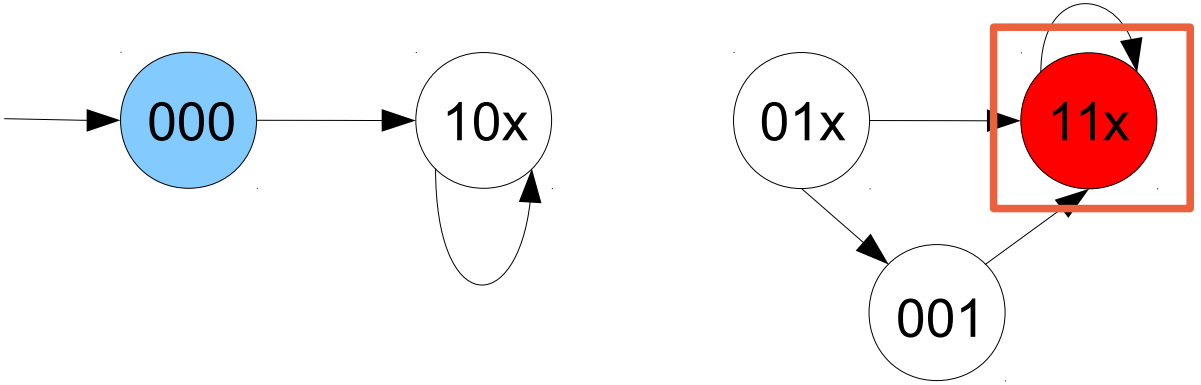
$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \top$$

[borrowed and adapted from F. Somenzi]

Example

Get bad cube $c = x_1 \wedge x_2$ in $F_1 \wedge \neg P$

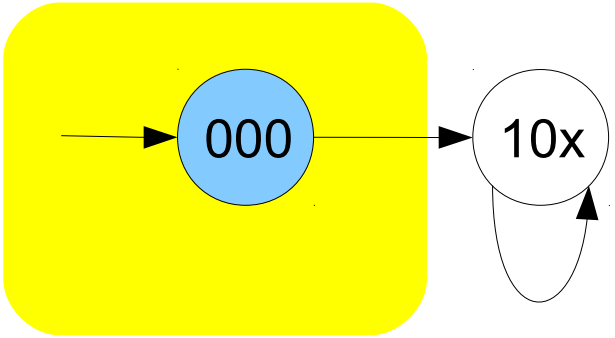


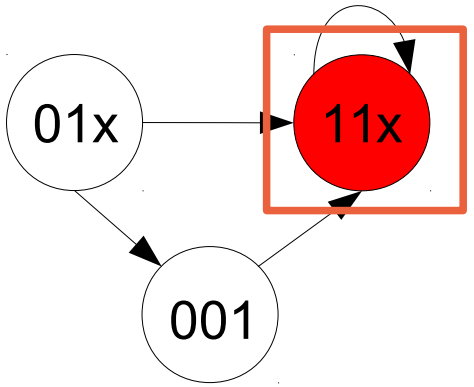
$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \top$$

Example

Is $\neg c$ inductive relative to F_0 ? $F_0 \wedge T \wedge \neg c \models \neg c'$



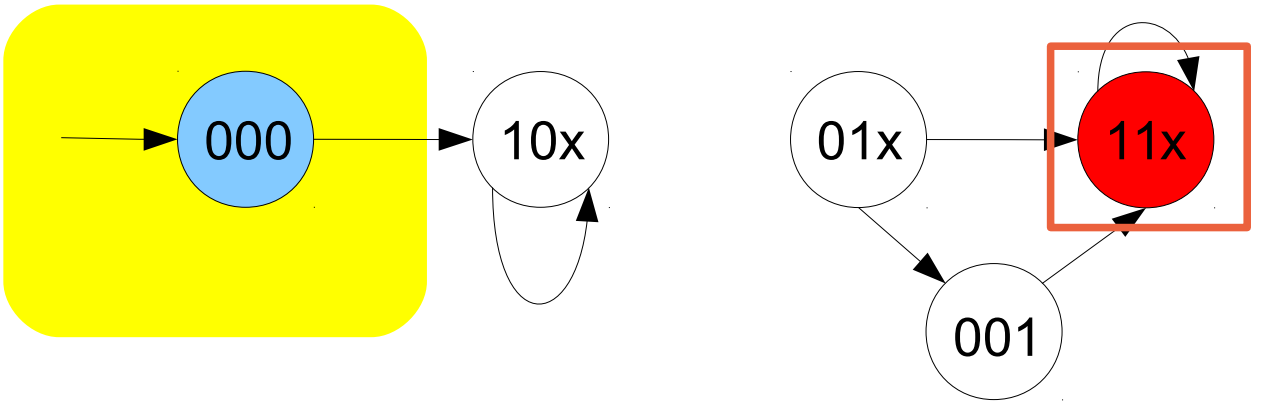


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \top$$

Example

Yes, generalize $\neg c = \neg x_1 \vee \neg x_2$

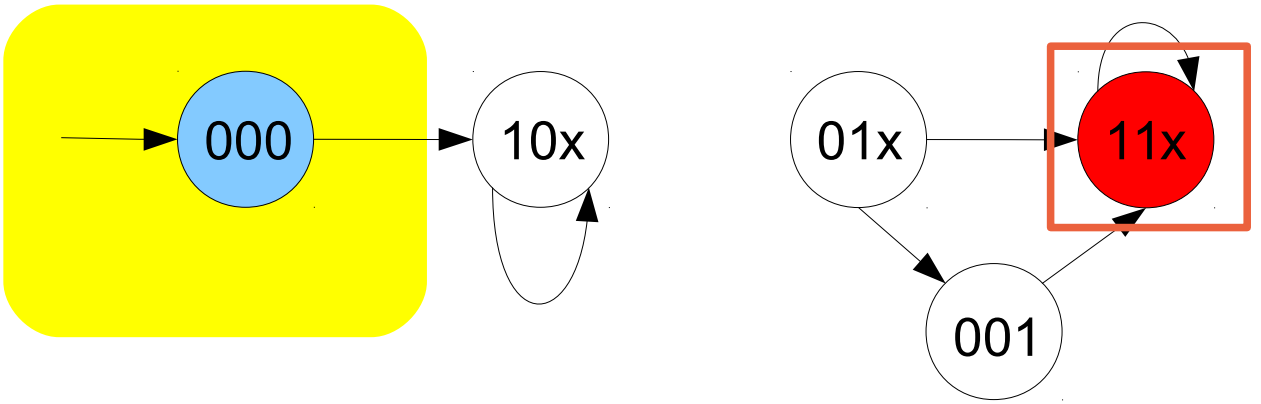


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \top$$

Example

Yes, generalize $\neg c = \neg x_1 \vee \neg x_2$



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

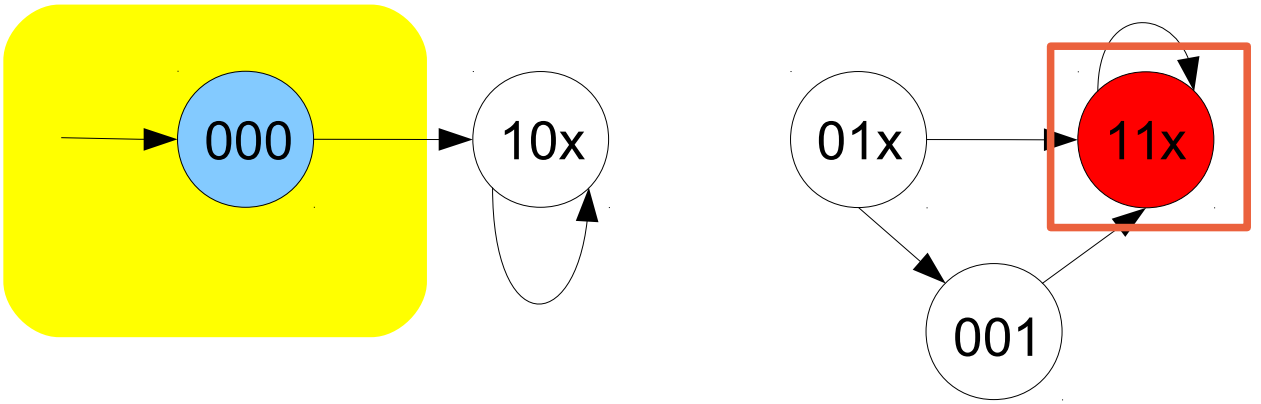
$$F_0 = I$$
$$F_1 = \top$$

Try dropping $\neg x_2$

$$F_0 \wedge T \wedge \neg x_1 \not\models \neg x_1'$$

Example

Yes, generalize $\neg c = \neg x_1 \vee \neg x_2$



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

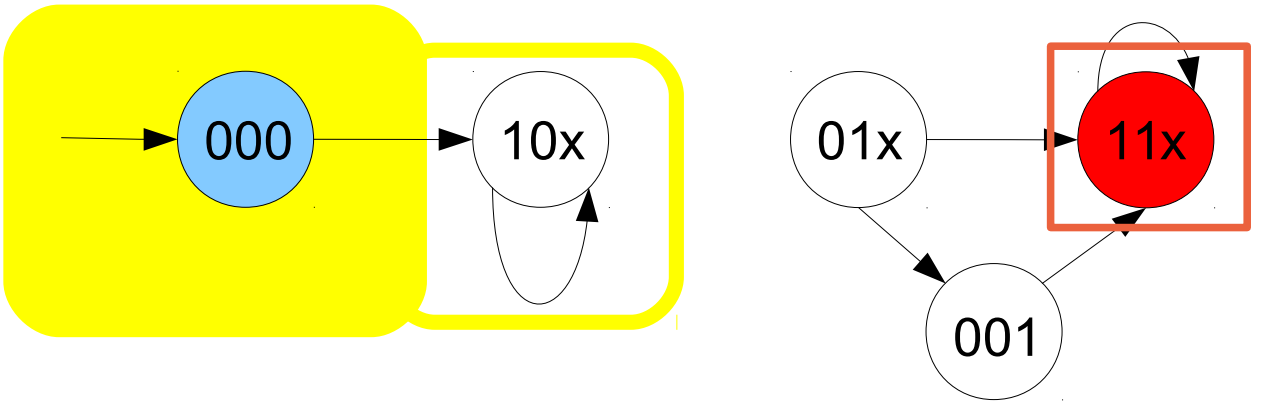
$$F_0 = I$$
$$F_1 = \top$$

Try dropping $\neg x_1$

$$F_0 \wedge T \wedge \neg x_2 \models \neg x_2'$$

Example

Yes, generalize $\neg c = \neg x_1 \vee \neg x_2$



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

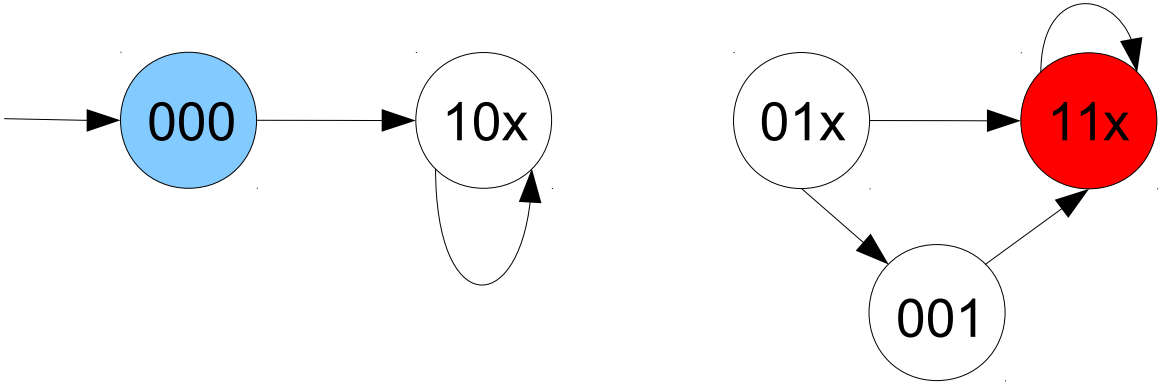
$$F_0 = I$$
$$F_1 = \top$$

Try dropping $\neg x_1$

$$F_0 \wedge T \wedge \neg x_2 \models \neg x_2'$$

Example

Update F_1

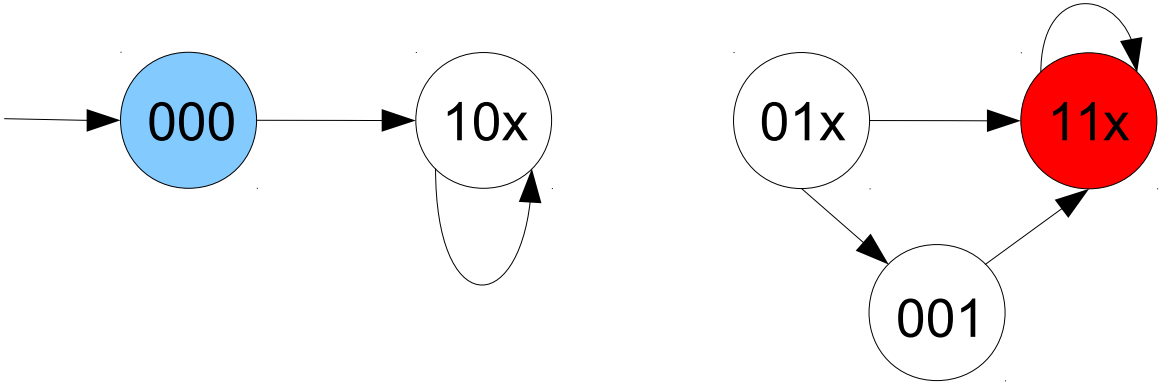


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2$$

Example

Blocking done for F_1. Add F_2 and propagate forward

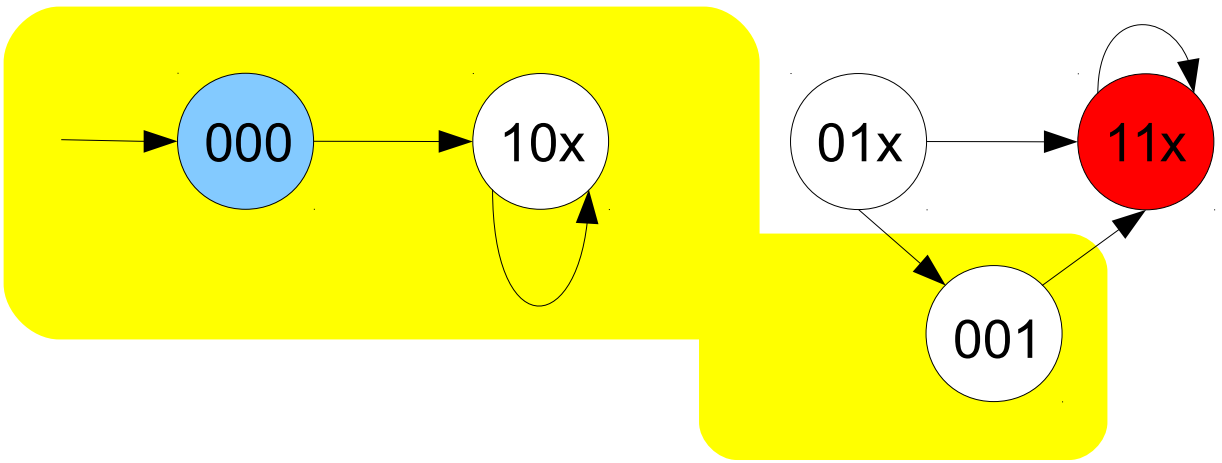


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Example

No clause propagates from F_1 to F_2

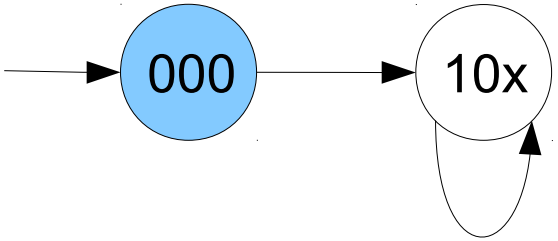


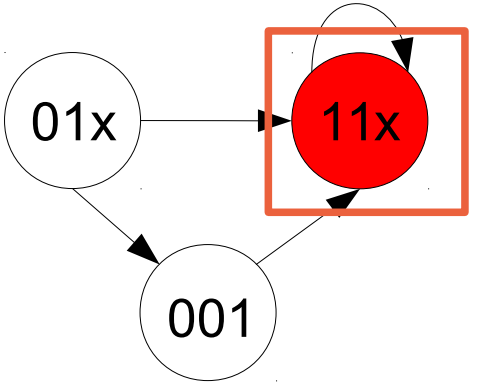
$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Example

Get bad cube $c = x_1 \wedge x_2$ in $F_2 \wedge \neg P$



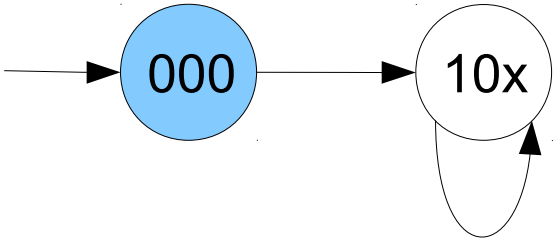


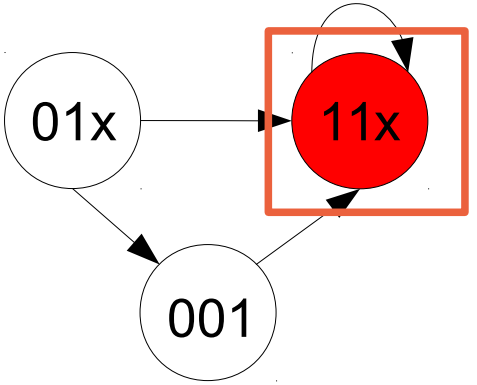
$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Example

Is $\neg c$ inductive relative to F_1 ? $F_1 \wedge T \wedge \neg c \models \neg c'$



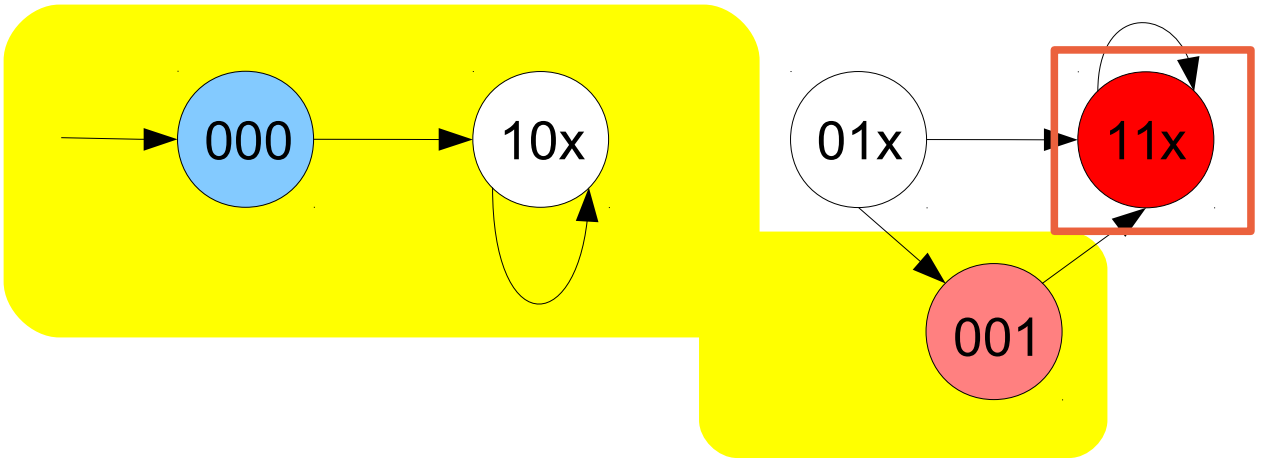


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Example

No, found CTI $s = \neg x_1 \wedge \neg x_2 \wedge x_3$

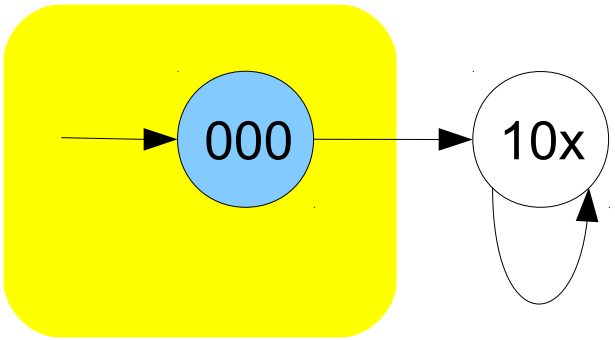


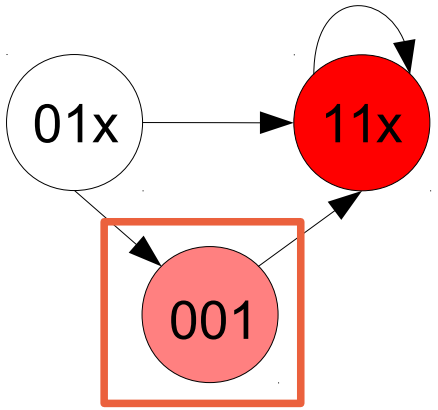
$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Example

Try blocking $\neg s$ at level 0: $F_0 \wedge T \wedge \neg s \models \neg s'$



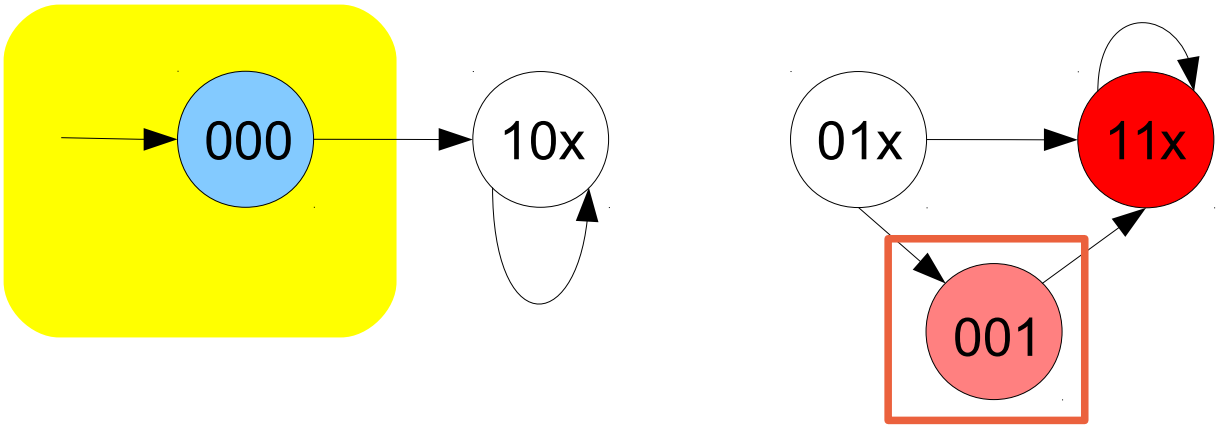


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Example

Yes, generalize $\neg s = x_1 \vee x_2 \vee \neg x_3$



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

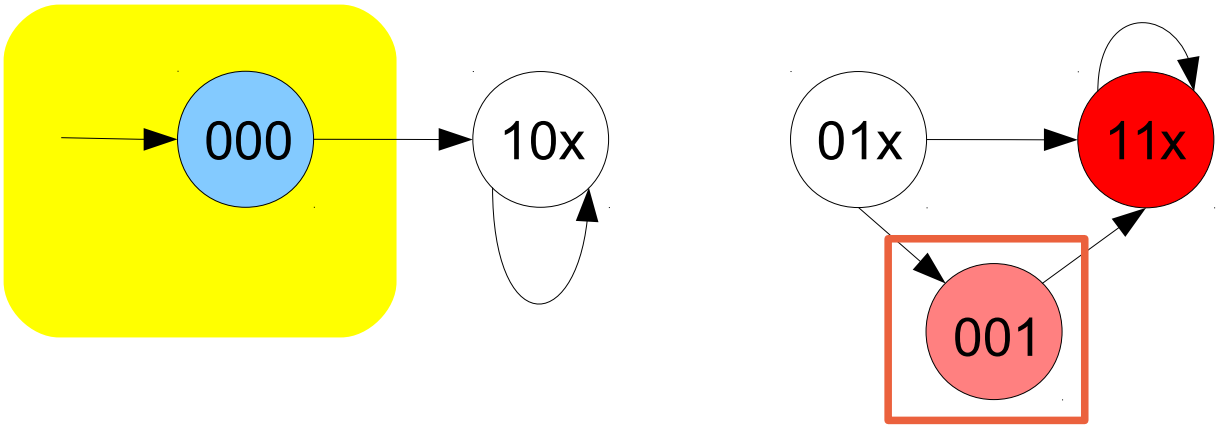
Try dropping x_1

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

$$F_0 \wedge T \wedge x_2 \vee \neg x_3 \not\models x_2' \vee \neg x_3'$$

Example

Yes, generalize $\neg s = x_1 \vee x_2 \vee \neg x_3$



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

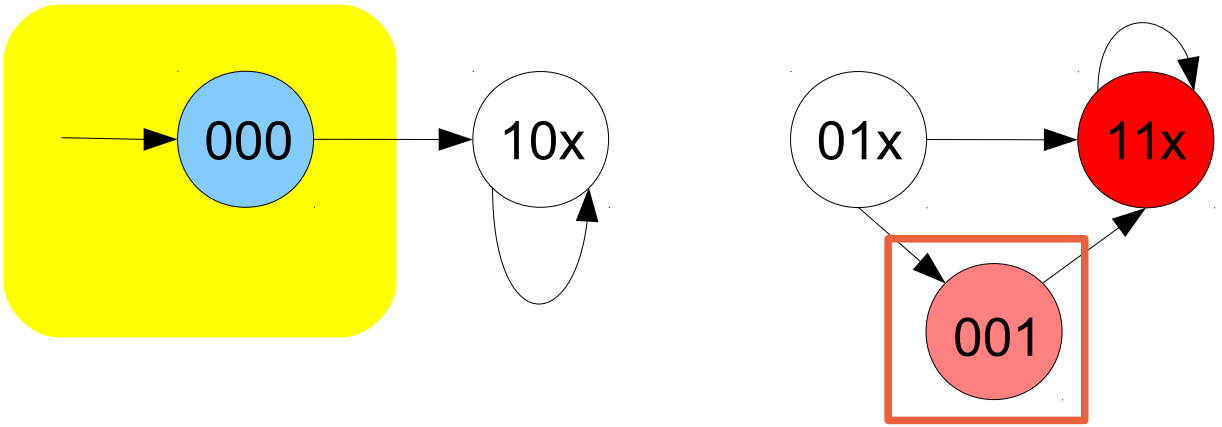
$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Try dropping x_2

$$F_0 \wedge T \wedge x_1 \vee \neg x_3 \models x'_1 \vee \neg x'_3$$

Example

Yes, generalize $\neg s = x_1 \vee x_2 \vee \neg x_3$



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

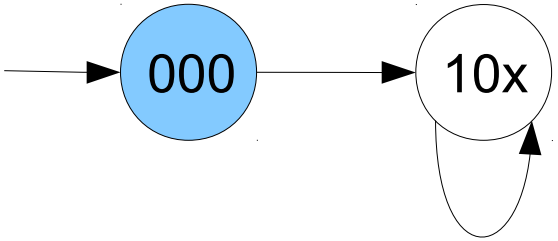
Try dropping x_3

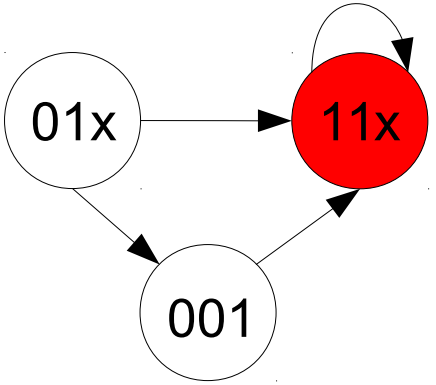
$$I \not\models x_1$$

$$F_0 = I$$
$$F_1 = \neg x_2$$
$$F_2 = \top$$

Example

Update F_1



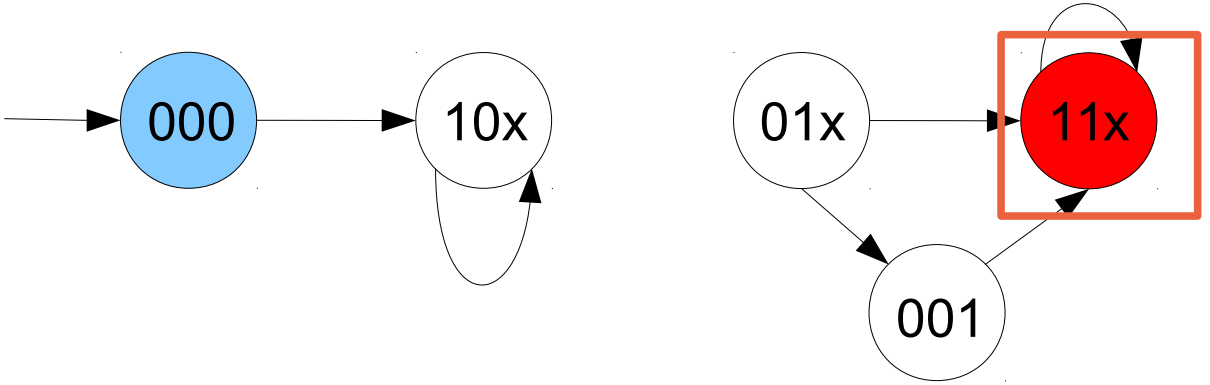


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \top$$

Example

Return to the original bad cube c

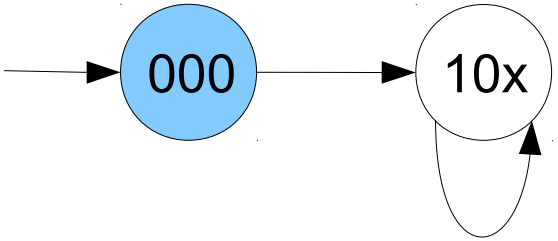


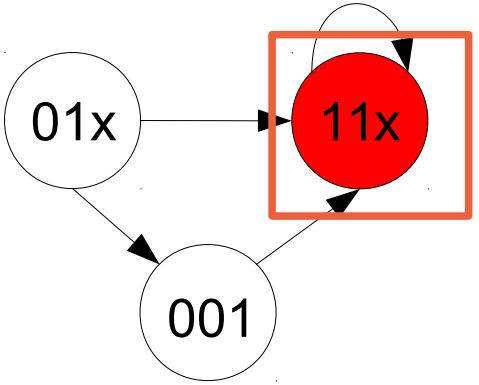
$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \top$$

Example

Is $\neg c$ inductive relative to F_1 ? $F_1 \wedge T \wedge \neg c \models \neg c'$



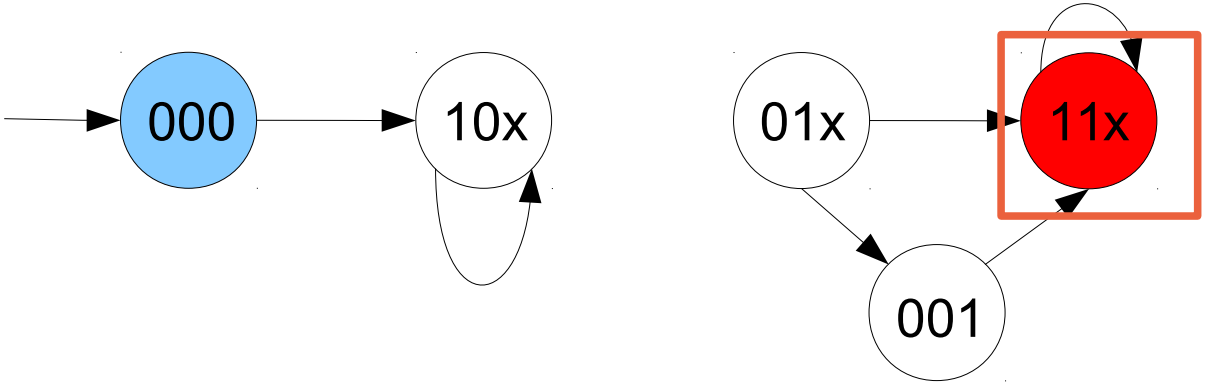


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \top$$

Example

Yes, generalize $\neg c = \neg x_1 \vee \neg x_2$



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

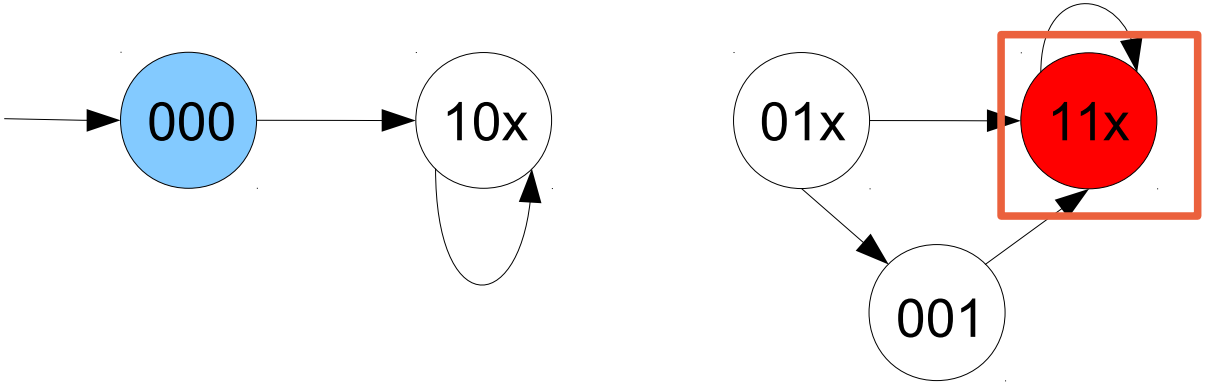
Try dropping $\neg x_1$

$$F_1 \wedge T \wedge \neg x_2 \models \neg x_2'$$

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \top$$

Example

Update F_2 and add new frame F_3

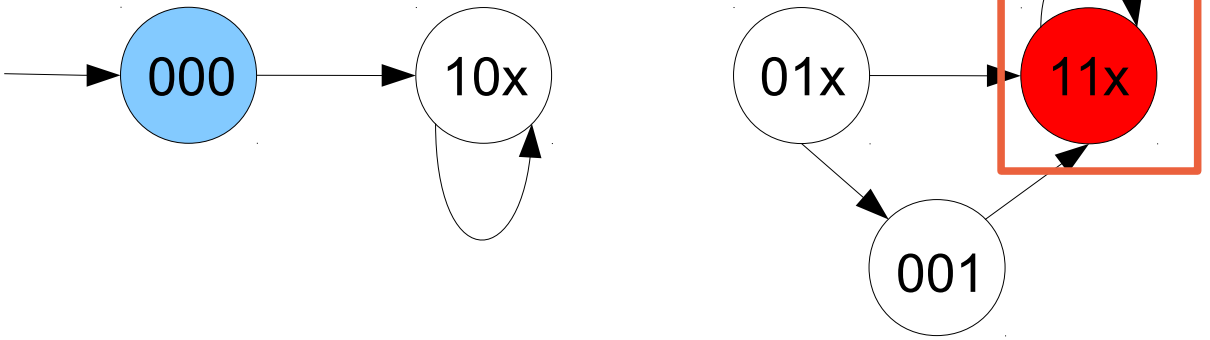


$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \neg x_2$$
$$F_3 = \top$$

Example

Perform forward propagation



$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

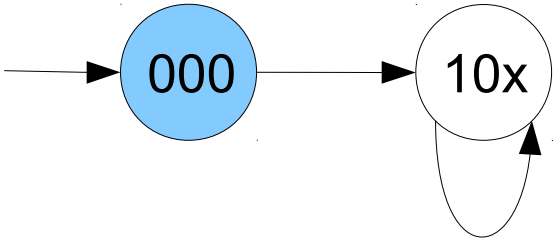
From F_1 to F_2 :

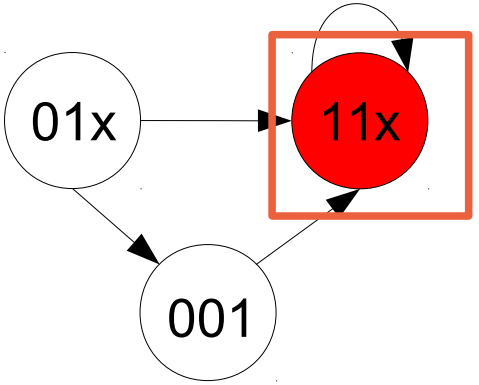
$$F_1 \wedge T \wedge (x_1 \vee \neg x_3) \models (x_1' \vee \neg x_3')$$

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \neg x_2$$
$$F_3 = \top$$

Example

Perform forward propagation





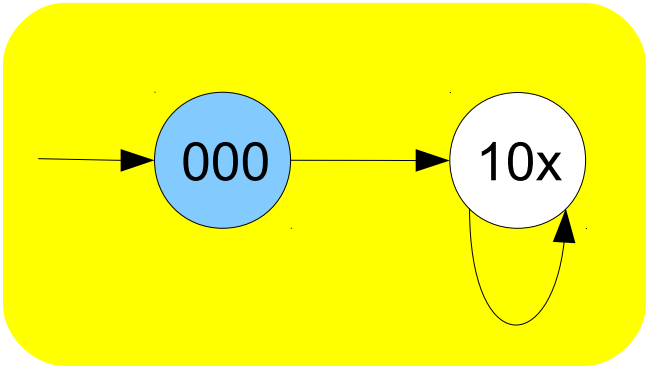
$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

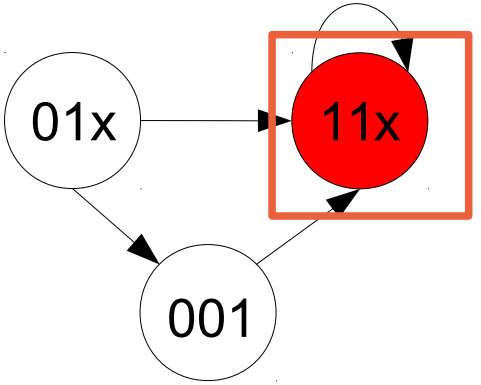
Found fixpoint!

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_3 = \top$$

Example

Perform forward propagation





$$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2$$

Inductive invariant:

$$F_1 \equiv F_2 \equiv \neg x_2 \wedge (x_1 \vee \neg x_3)$$

$$F_0 = I$$
$$F_1 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_2 = \neg x_2 \wedge (x_1 \vee \neg x_3)$$
$$F_3 = \top$$

Outline

Introduction

IC3 for finite-state systems

SMT-based IC3 for infinite-state systems

IC3 for LTL verification

IC3 with SMT

- How to generalize from SAT to SMT?

IC3 with SMT

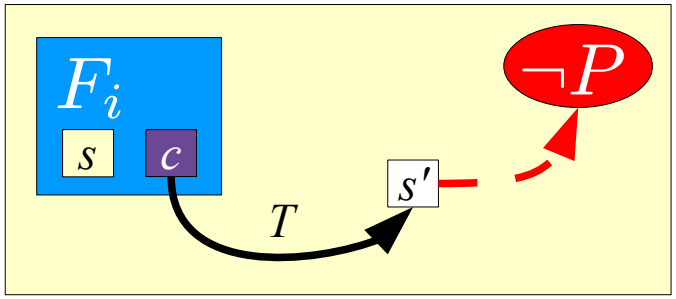
- How to generalize from SAT to SMT?
- Good news: replacing the SAT solver with an SMT solver is enough for partial correctness
- but what about:
  - termination?
  - efficiency?

IC3 with SMT

- How to generalize from SAT to SMT?
- Good news: replacing the SAT solver with an SMT solver is enough for partial correctness
- but what about:
  - termination?
    - Easy! (answer)
      - the problem is in general undecidable, so no hope here
  - efficiency?

$RelInd(F_{k-1}, T, s)$ with SMT

- When $F_i \wedge \neg s \wedge T \wedge s'$ is satisfiable:
  - *s* reaches $\neg P$ in *k-i* steps
  - *s* can be reached from F_i in 1 step
  - strengthen F_i by blocking cubes *c* in the preimage of *s*

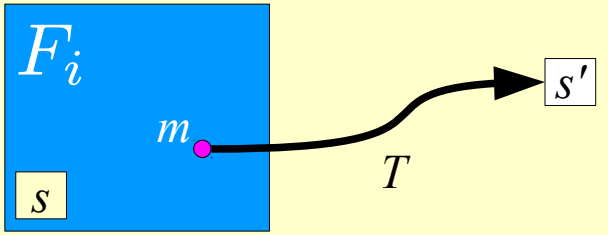


- In the Boolean case, get *c* from SAT assignment (and generalize)

- For SMT(LRA):
  - Would exclude a single point in an infinite space

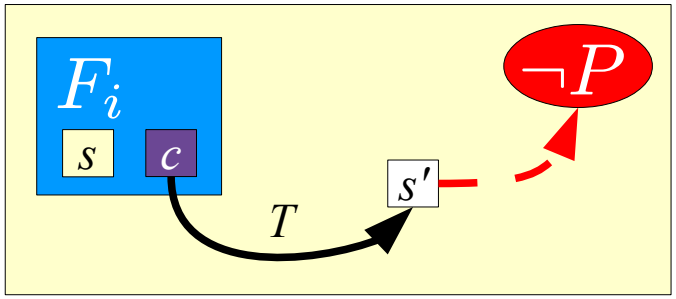
Single model m from SMT solver:

$$x = 3 \wedge y = 7$$



$RelInd(F_{k-1}, T, s)$ with SMT

- When $F_i \wedge \neg s \wedge T \wedge s'$ is satisfiable:
  - *s* reaches $\neg P$ in *k-i* steps
  - *s* can be reached from F_i in 1 step
  - strengthen F_i by blocking cubes *c* in the preimage of *s*

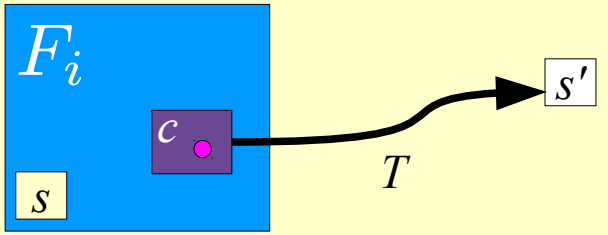


- In the Boolean case, get *c* from SAT assignment (and generalize)

- For SMT(LRA): underapproximated quantifier elimination
  - Encodes a set of predecessors
  - Cheaper than full quantifier elimination
    - But still potentially expensive
    - Not always available
      - E.g for UF+LRA

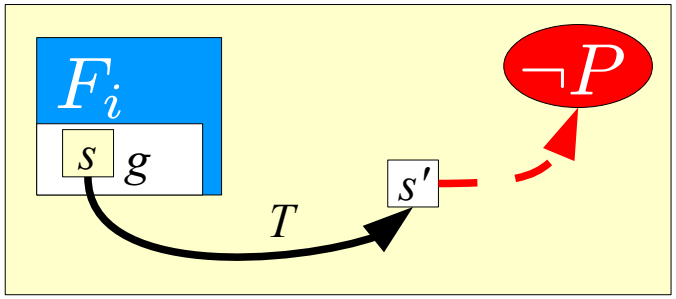
underapproximated preimage:

$(x \leq 3) \wedge (y \geq 7)$



$RelInd(F_{k-1}, T, s)$ with SMT

- When $F_i \wedge \neg s \wedge T \wedge s'$ is unsatisfiable:
  - Compute a generalization *g* of *s* to block
  - Block more than a single cube at a time



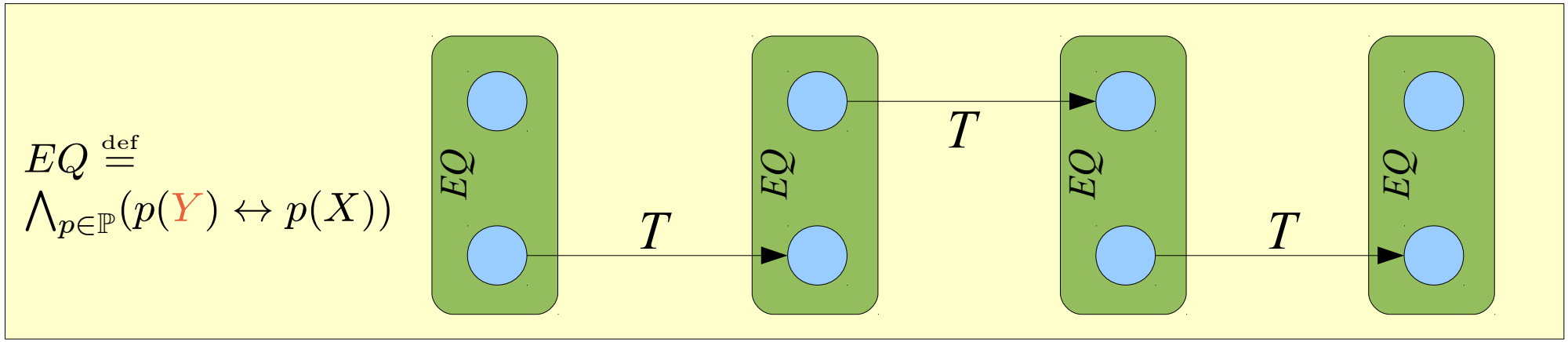
- In the Boolean case, use inductive generalization algorithms

- For SMT, Boolean algorithms plus theory-specific “ad hoc” techniques
  - Based on Farkas' lemma for LRA [HB SAT'12]
  - [WK DATE'13] for BV
  - [KJN FORMATS'12] for timed automata

Implicit Predicate Abstraction [Tonetta FM'09]

- Abstract version of k-induction, avoiding explicit computation of the abstract transition relation
  - By embedding the abstraction in the SMT encoding
- Given a set of predicates $\mathbb{P}$ and an unrolling depth k, the abstract path $\widehat{\mathrm{Path}}_{k,\mathbb{P}}$ is

$$\bigwedge_{1 \leq h < k} (T(Y^{h-1}, X^h) \wedge \bigwedge_{p \in \mathbb{P}} (p(X^h) \leftrightarrow p(Y^h)) \wedge T(Y^{k-1}, X^k)$$



IC3 with Implicit Abstraction

- Integrate the idea of Implicit Abstraction within IC3
- Use abstract transition relation $T(X, Y')$ instead of $T(X, X')$
- Learn clauses only over predicates $\mathbb{P}$
- Use abstract relative induction check:

$$\mathrm{AbsRelInd}(F, T, s, \mathbb{P}) := F(X) \wedge s(X) \wedge T(X, Y') \wedge \bigwedge_{p \in \mathbb{P}} (p(X') \leftrightarrow p(Y')) \wedge \neg s(X')$$

IC3 with Implicit Abstraction

- Integrate the idea of Implicit Abstraction within IC3
- Use abstract transition relation $T(X, Y')$ instead of $T(X, X')$
- Learn clauses only over predicates $\mathbb{P}$
- Use abstract relative induction check:

$$\mathrm{AbsRelInd}(F, T, s, \mathbb{P}) := F(X) \wedge s(X) \wedge T(X, Y') \wedge \bigwedge_{p \in \mathbb{P}} (p(X') \leftrightarrow p(Y')) \wedge \neg s(X')$$

- If UNSAT ⇨inductive strengthening as in the Boolean case
  - No theory-specific technique needed
  - Theory reasoning confined within the SMT solver

IC3 with Implicit Abstraction

- Integrate the idea of Implicit Abstraction within IC3
- Use abstract transition relation $T(X, Y')$ instead of $T(X, X')$
- Learn clauses only over predicates $\mathbb{P}$
- Use abstract relative induction check:

$$\mathrm{AbsRelInd}(F, T, s, \mathbb{P}) := F(X) \wedge s(X) \wedge T(X, Y') \wedge \bigwedge_{p \in \mathbb{P}} (p(X') \leftrightarrow p(Y')) \wedge \neg s(X')$$

- If SAT ⇨ abstract predecessor c from the SMT model μ
  - $c \stackrel{\text{def}}{=} \{p(X) \mid p \in \mathbb{P} \wedge \mu \models p(X)\} \cup \{\neg p(X) \mid \mu \not\models p(X)\}$
  - No quantifier elimination needed

Example

- $T \stackrel{\text{def}}{=} (2x_1' - 3x_1 \leq 4x_2' + 2x_2 + 3) \wedge (3x_1 - 2x_2' = 0)$
- $\mathbb{P} \stackrel{\text{def}}{=} \{(x_1 - x_2 \geq 4), (x_1 < 3)\}$
- $s \stackrel{\text{def}}{=} \neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3)$
- $RelInd(\emptyset, T, s)$ is SAT
- Compute a predecessor with $\exists_{\text{approx}} x_1', x_2'.(\neg s \wedge T \wedge s')$

  $(\frac{5}{2} \leq 3x_1 + x_2) \wedge \neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3) \wedge \neg(-\frac{2}{3} \leq x_1)$

Example

- $T \stackrel{\text{def}}{=} (2x_1' - 3x_1 \leq 4x_2' + 2x_2 + 3) \wedge (3x_1 - 2x_2' = 0)$
- $\mathbb{P} \stackrel{\text{def}}{=} \{(x_1 - x_2 \geq 4), (x_1 < 3)\}$
- $s \stackrel{\text{def}}{=} \neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3)$
- $RelInd(\emptyset, T, s)$ is SAT
- Compute a predecessor with $\exists_{\text{approx}} x_1', x_2'.(\neg s \wedge T \wedge s')$

  $(\frac{5}{2} \leq 3x_1 + x_2) \wedge \neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3) \wedge \neg(-\frac{2}{3} \leq x_1)$
- $\text{AbsRelInd}(\emptyset, T, s, \mathbb{P}) := T[X' \mapsto Y'] \wedge$

  $\neg s \wedge s' \wedge$

  $(x_1' - x_2' \geq 4) \leftrightarrow (y_1' - y_2' \geq 4) \wedge (x_1' < 3) \leftrightarrow (y_1' < 3)$
- Compute predecessor from SMT model $\mu \stackrel{\text{def}}{=} \{x_1 \mapsto 0, x_2 \mapsto 1\}$

  $\neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3)$

Example

- $T \stackrel{\text{def}}{=} (2x_1' - 3x_1 \leq 4x_2' + 2x_2 + 3) \wedge (3x_1 - 2x_2' = 0)$
- $\mathbb{P} \stackrel{\text{def}}{=} \{(x_1 - x_2 \geq 4), (x_1 < 3)\}$
- $s \stackrel{\text{def}}{=} \neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3)$
- $RelInd(\emptyset, T, s)$ is SAT
- Compute a predecessor with $\exists_{\text{approx}} x_1', x_2'.(\neg s \wedge T \wedge s')$

  $(\frac{5}{2} \leq 3x_1 + x_2) \wedge \neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3) \wedge \neg(-\frac{2}{3} \leq x_1)$
- $\text{AbsRelInd}(\emptyset, T, s, \mathbb{P}) := T[X' \mapsto Y'] \wedge$

  $\neg s \wedge s' \wedge$

  $(x_1' - x_2' \geq 4) \leftrightarrow (y_1' - y_2' \geq 4) \wedge (x_1' < 3) \leftrightarrow (y_1' < 3)$
- Compute predecessor from SMT model $\mu \stackrel{\text{def}}{=} \{x_1 \mapsto 0, x_2 \mapsto 1\}$

  $\neg(x_1 - x_2 \geq 4) \wedge (x_1 < 3)$

Abstraction Refinement

- Abstract predecessors are overapproximations
  - Spurious counterexamples can be generated
- We can apply standard abstraction refinement techniques
  - Use sequence interpolants to discover new predicates
  - Sequence of abstract states $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$
  - SMT check on $s_0^0 \wedge T_{\text{concrete}}^0 \wedge s_1^1 \wedge \ldots \wedge T_{\text{concrete}}^{k-1} \wedge s_k^k$
  - If unsat, compute sequence of interpolants for

    $[s_0^0 \wedge T_{\text{concrete}}^0 \wedge \ldots \wedge T_{\text{concrete}}^{i-1}], [s_i^i \wedge \ldots \wedge T_{\text{concrete}}^{k-1} \wedge s_k^k]$
  - Add all the predicates in the interpolants to $\mathbb{P}$

Incrementality

- Abstraction refinement is *fully incremental*
- No restart from scratch
- Can keep all the clauses of $F_1, \ldots, F_k$
  - Refinements monotonically strengthen T

    $$T_{\text{new}} \stackrel{\text{def}}{=} T_{\text{old}} \wedge \bigwedge_{p \in \mathbb{P}_{\text{new}}} (p(X) \leftrightarrow p(Y)) \wedge (p(X') \leftrightarrow p(Y'))$$

  - All IC3 invariants on $F_1, \ldots, F_k$ are preserved

    $F_{i+1} \subseteq F_i$ (so $F_i \models F_{i+1}$) ✔

    for all $i < k, F_i \models P$ ✔

    $\mathrm{F}_i \wedge T_{\text{new}} \models F'_{i+1}$ ✔

- Abstract counterexample check can use incremental SMT

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Check base case: $\mathrm{Init} \models \mathrm{Property}$ 
- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Predicates $\mathbb{P}$

  $(d = 1) \quad (c \geq d)$

  $(d > 2) \quad (c > d)$
- Trace: $F_0 := \text{Init}$

  $F_1 := \top$
- Get bad cube
  - SMT check $F_1 \wedge \neg \text{Prop}$
  - SAT with model $\mu := \{c = 0, d = 2\}$
  - Evaluate predicates wrt. μ
    - Return $c := \{\neg(d = 1), \neg(c \geq d), (d > 2), \neg(c > d)\}$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Rec. block *c*
  - Check

$$AbsRelInd(F_0, T, c, \mathbb{P}) := \text{Init} \wedge$$

$$\begin{array}{l} (y_c = c + d) \wedge (y_d = d + 1) \wedge \\ ((d' = 1) \leftrightarrow (y_d = 1)) \wedge ((c' \geq d') \leftrightarrow (y_c \geq y_d)) \wedge \\ ((d' > 2) \leftrightarrow (y_d > 2)) \wedge ((c' > d') \leftrightarrow (y_c > y_d)) \wedge \end{array}$$

$$\neg c \wedge c'$$

- Predicates $\mathbb{P}$

$$\begin{array}{ll} (d = 1) & (c \geq d) \\ (d > 2) & (c > d) \end{array}$$

- Trace: $F_0 := \text{Init}$, $F_1 := \top$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Rec. block *c*
  - Check

    $AbsRelInd(F_0, T, c, \mathbb{P})$ ✔
  - Unsat core: $\{(d' > 2)\}$
  - Update $F_1 := F_1 \wedge \neg(d > 2)$
- Predicates $\mathbb{P}$

  $(d = 1) \quad (c \geq d)$

  $(d > 2) \quad (c > d)$
- Trace: $F_0 := \text{Init}$

  $F_1 := \top$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Forward propagation

- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2)$$
$$F_2 := \top$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Get bad cube at 2
  - $c := \{\neg(d = 1), \neg(c \geq d), (d > 2), \neg(c > d)\}$
- Predicates $\mathbb{P}$

  $(d = 1)$ $(c \geq d)$
  $(d > 2)$ $(c > d)$
- Trace: $F_0 := \text{Init}$

  $F_1 := \neg(d > 2)$

  $F_2 := \top$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Rec. block *c*
  - ...
  - Update $F_1 := F_1 \wedge (c \geq d)$
  - ...
  - Update $F_2 := F_2 \wedge (c > d) \vee \neg(d > 2)$

- Predicates $\mathbb{P}$

  $(d = 1) \quad (c \geq d)$

  $(d > 2) \quad (c > d)$

- Trace: $F_0 := \text{Init}$

  $F_1 := \neg(d > 2)$

  $F_2 := \top$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Forward propagation

- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c > d) \vee \neg(d > 2)$$
$$F_3 := \top$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Get bad cube at 3
  - $c := \{\neg(d = 1), \neg(c \geq d), (d > 2), \neg(c > d)\}$
- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c > d) \vee \neg(d > 2)$$
$$F_3 := \top$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$

- Rec block *c*
  - Check

$$AbsRelInd(F_2, T, c, \mathbb{P})$$ 

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c > d) \vee \neg(d > 2)$$
$$F_3 := \top$$

  - SMT model

$$\mu := \{c = 0, d = 2, c' = 0, d' = 3, y_c = 2, y_d = 3\}$$

  - (Abstract) predecessor

$$s := \{\neg(d > 2), \neg(c > d), \neg(d = 1), \neg(c \geq d)\}$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Rec block *s* (at level 2)
  - . . .
  - Reached level 0, abstract cex:

$$q := \neg(d > 2), \neg(c > d), (d = 1), (c \geq d)$$
$$p := \neg(d > 2), \neg(c > d), \neg(d = 1), (c \geq d)$$
$$s := \neg(d > 2), \neg(c > d), \neg(d = 1), \neg(c \geq d)$$
$$c := \neg(d = 1), \neg(c \geq d), (d > 2), \neg(c > d)$$

- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c > d) \vee \neg(d > 2)$$
$$F_3 := \top$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Check abstract counterexample
  - SMT check

    $I_0 \wedge q_0 \wedge T_{0\mapsto 1} \wedge p_1 \wedge T_{1\mapsto 2} \wedge s_2 \wedge T_{2\mapsto 3} \wedge c_3$

    UNSAT

- Predicates $\mathbb{P}$

  $(d = 1)$ $(c \geq d)$

  $(d > 2)$ $(c > d)$

- Trace: $F_0 := \mathrm{Init}$

  F_1

  F_2

  F_3

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Predicates $\mathbb{P}$

  $(d = 1) \quad (c \geq d)$

  $(d > 2) \quad (c > d)$
- Trace: $F_0 := \text{Init}$

  F_1

  F_2

  F_3
- Check abstract counterexample
  - Compute sequence interpolant

$$\underbrace{I_0 \wedge q_0 \wedge T_{0\mapsto 1}}_{A_1} \underbrace{\wedge\, p_1 \wedge T_{1\mapsto 2} \wedge s_2 \wedge T_{2\mapsto 3} \wedge c_3}_{B_1}$$

$$\varphi_1 := (d_1 \geq 2)$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Predicates $\mathbb{P}$

  $(d = 1) \quad (c \geq d)$

  $(d > 2) \quad (c > d)$
- Trace: $F_0 := \text{Init}$

  F_1

  F_2

  F_3
- Check abstract counterexample
  - Compute sequence interpolant

$$\underbrace{I_0 \wedge q_0 \wedge T_{0\mapsto 1} \wedge p_1 \wedge T_{1\mapsto 2}}_{A_2} \wedge \underbrace{s_2 \wedge T_{2\mapsto 3} \wedge c_3}_{B_2}$$

$$\varphi_1 := (d_1 \geq 2)$$

$$\varphi_2 := (d_2 \geq 3)$$

Example

- System S with 2 state vars c and d
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$

- Predicates $\mathbb{P}$

$(d = 1) \quad (c \geq d)$

$(d > 2) \quad (c > d)$

$(d \geq 2) \quad (d \geq 3)$

- Trace: $F_0 := \text{Init}$

F_1

F_2

F_3

- Check abstract counterexample
  - Compute sequence interpolant

$$\underbrace{I_0 \wedge q_0 \wedge T_{0\mapsto 1} \wedge p_1 \wedge T_{1\mapsto 2} \wedge s_2 \wedge T_{2\mapsto 3}}_{A_3} \wedge \underbrace{c_3}_{B_3}$$

$\varphi_1 := (d_1 \geq 2)$

$\varphi_2 := (d_2 \geq 3)$

$\varphi_3 := \bot$

Update predicates $\mathbb{P}$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Update abstract trans
- Resume IC3 from level 3
  - . . .

- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$
$$(d \geq 2) \quad (d \geq 3)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c > d) \vee \neg(d > 2)$$
$$F_3 := \top$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Update abstract trans
- Resume IC3 from level 3
  - . . .

- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$
$$(d \geq 2) \quad (d \geq 3)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c \geq d) \vee \neg(d \geq 2) \wedge F_3$$
$$F_3 := (d = 1) \vee (d \geq 2) \wedge \neg(c \geq d) \wedge F_4$$
$$F_4 := (c > d) \vee \neg(d > 2)$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Update abstract trans
- Resume IC3 from level 3
  - . . .
  - Forward propagation

- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$
$$(d \geq 2) \quad (d \geq 3)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c \geq d) \vee \neg(d \geq 2) \wedge F_3$$
$$F_3 := (d = 1) \vee (d \geq 2) \wedge \neg(c \geq d) \wedge F_4$$
$$F_4 := (c > d) \vee \neg(d > 2)$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Update abstract trans
- Resume IC3 from level 3
  - . . .
  - Forward propagation

$$F_2 \wedge \widehat{T}_{\mathbb{P}} \models (c' \geq d') \vee \neg(d' \geq 2)$$

- Predicates $\mathbb{P}$

$$\begin{array}{ll} (d = 1) & (c \geq d) \\ (d > 2) & (c > d) \\ (d \geq 2) & (d \geq 3) \end{array}$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := (c \geq d) \vee \neg(d \geq 2) \wedge F_3$$
$$F_3 := (d = 1) \vee (d \geq 2) \wedge \neg(c \geq d) \wedge F_4$$
$$F_4 := (c > d) \vee \neg(d > 2)$$

Example

- System *S* with 2 state vars *c* and *d*
  - Init: $(d = 1) \wedge (c \geq d)$
  - Trans: $(c' = c + d) \wedge (d' = d + 1)$
  - Property: $(d > 2) \implies (c > d)$
- Update abstract trans
- Resume IC3 from level 3
  - . . .
  - Forward propagation

SAFE ✔

- Predicates $\mathbb{P}$

$$(d = 1) \quad (c \geq d)$$
$$(d > 2) \quad (c > d)$$
$$(d \geq 2) \quad (d \geq 3)$$

- Trace: $F_0 := \text{Init}$

$$F_1 := \neg(d > 2) \wedge (c \geq d) \wedge F_2$$
$$F_2 := F_3$$
$$F_3 := (c \geq d) \vee \neg(d \geq 2) \wedge$$
$$(d = 1) \vee (d \geq 2) \wedge$$
$$\neg(c \geq d) \wedge F_4$$
$$F_4 := (c > d) \vee \neg(d > 2)$$

Implementing IC3-IA

- Get the code at:
  http://es-static.fbk.eu/people/griggio/vtsa2015/
  - Open source (GPLv3) implementation on top of MathSAT
    http://mathsat.fbk.eu/
    - Incremental interface
    - Assumptions and unsat core
    - Interpolation
- Simple (~1700 lines of C++, including parser and statistics, according to David A. Wheeler's 'SLOCCount') yet competitive
  - Input in VMT format (a simple extension of SMT-LIB)

    https://nuxmv.fbk.eu/index.php?n=Languages.VMT
- Let's analyse it!

Outline

Introduction

IC3 for finite-state systems

SMT-based IC3 for infinite-state systems

IC3 for LTL verification

Linear Temporal Logic

- Syntax
  - A (quantifier-free) first-order formula φ
  - $\mathbf{X}\varphi$ (ne**X**t φ)
  - $\varphi \mathbf{U} \psi$ (φ **U**ntil ψ)
  - $\mathbf{F}\varphi$ (**F**inally φ)
  - $\mathbf{G}\varphi$ (**G**lobally φ)

- Semantics
  - Given an infinite path $\pi := s_0, s_1, \ldots, s_i, \ldots$
    - $\pi \models \mathbf{X}\varphi$ iff $s_1, \ldots \models \varphi$
    - $\pi \models \varphi \mathbf{U} \psi$ iff $\exists j > 0 . s_j, \ldots \models \psi$ and $\forall 0 \leq k < j . s_k, \ldots \models \varphi$
    - $\pi \models \mathbf{F}\varphi$ iff $\exists j . s_j, \ldots \models \varphi$
    - $\pi \models \mathbf{G}\varphi$ iff $\forall j . s_j, \ldots \models \varphi$
  - A system *S* satisfies an LTL formula φ ($S \models \varphi$) iff all inifinite paths of *S* satisfy φ

LTL verification

- Automata-based approach:
  - Given an LTL property φ, build a transition system $S_{\neg\varphi}$ with a fairness condition $f_{\neg\varphi}$, such that

$$S \models \varphi \quad \text{iff} \quad S \times S_{\neg\varphi} \models \mathbf{FG}\neg f_{\neg\varphi}$$

- Finite-state case:
  - **lasso-shaped counterexamples**, with $f_{\neg\varphi}$ at least once in the loop
  - **liveness to safety** transformation: absence of lasso-shaped counterexamples as an invariant property
    - Duplicate the state variables $X_{\text{copy}} = \{x_c | x \in X\}$
    - Non-deterministically save the current state
    - Remember when $f_{\neg\varphi}$ in extra state var triggered
    - Invariant: $\mathbf{G}\neg(X = X_{\text{copy}} \wedge \text{triggered})$

Liveness to Safety for Inifinite States

- Unsound for infinite-state systems
  - Not all counterexamples are lasso-shaped

$$I(S) \stackrel{\text{def}}{=} (x = 0) \qquad T(S) \stackrel{\text{def}}{=} (x' = x + 1) \qquad \varphi \stackrel{\text{def}}{=} \mathbf{FG}(x < 5)$$

- Liveness to safety with Implicit Abstraction
  - Apply the l2s transformation to the abstract system
    - Save the values of the predicates instead of the concrete state
  - Do it on-the-fly, **tightly integrating** l2s with IC3
  - Sound but incomplete
    - When abstract loop found, simulate in the concrete and refine
    - Might still diverge during refinement
      - Intrinsic limitation of **state** predicate abstraction

K-liveness

- Simple but effective technique for LTL verification of finite-state systems
- Key insight: $M \times M_{\neg\varphi} \models \mathbf{FG}\neg f_{\neg\varphi}$ iff exists *k* such that $f_{\neg\varphi}$ **is visited at most *k* times**
  - Again, a safety property
- K-liveness: increase k incrementally, within IC3
  - Liveness checking as a sequence of safety checks
  - Exploits the highly incremental nature of IC3
  - Sound also for infinite-state systems
    - What about completeness?

K-liveness for hybrid automata

- K-liveness is incomplete for infinite-state systems
  - Even if $M \times M_{\neg\varphi} \models \mathbf{FG}\neg f_{\neg\varphi}$, there might be **no concrete *k*** bound for the number of violations of $\neg f_{\neg\varphi}$

$$I(S) \stackrel{\text{def}}{=} (x = n) \quad T(S) \stackrel{\text{def}}{=} (x' = x + 1) \quad \varphi \stackrel{\text{def}}{=} \mathbf{FG}(x > n)$$

- K-zeno: extension of K-liveness for hybrid automata
  - Key idea: exploit progress of time to make k-liveness converge
  - By extending the original model with a **"symbolic fairness monitor"** Z^{φ}_{β} that forces time progress
  - Under certain conditions, restores completeness of k-liveness
    - If $M \times M_{\neg\varphi} \models \mathbf{FG}\neg f_{\neg\varphi}$, then exists k such that $M \times M_{\neg\varphi} \times Z^{\varphi}_{\beta}$ visits f_Z at most k times
    - (clearly, safety check can still diverge)

Selected bibliography

DISCLAIMER: again, this is definitely incomplete. Apologies to missing authors/works

- IC3 for finite-state systems
  - Bradley, Manna. **Checking Safety by Inductive Generalization of Counterexamples to Induction**. FMCAD 2007
  - Bradley. **SAT-based Model Checking Without Unrolling**. VMCAI 2011
  - Een, Mischenko, Brayton. **Efficient Implementation of Property-Directed Reachability**. FMCAD 2011
  - Hassan, Somenzi, Bradley. **Better Generalization in IC3**. FMCAD 2013
  - Vizel, Gurfinkel. **Interpolating Property-Directed Reachability**. CAV 2014

Selected bibliography

- IC3 for infinite-state systems
  - Hoder, Bjørner. **Generalized Property-Directed Reachability**. SAT 2012
  - Cimatti, Griggio, Mover, Tonetta. **IC3 Modulo Theories with Implicit Predicate Abstraction**. TACAS 2013
  - Komuravelli, Gurfinkel, Chaki. **SMT-Based Model Checking for Recursive Programs**. CAV 2014
  - Birgmeier, Bradley, Weissenbacher. **Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR)**. CAV 2014
  - Bjørner, Gurfinkel. **Property Directed Polyhedral Abstraction**. VMCAI 2015

Selected bibliography

- IC3 for LTL verification
  - Bradley, Somenzi, Hassan, Zhang. **An incremental approach to model checking progress properties**. FMCAD 2011
  - Claessen, Sörensson. **A liveness checking algorithm that counts**. FMCAD 2012
  - Cimatti, Griggio, Mover, Tonetta. **Verifying LTL Properties of Hybrid Systems with K-Liveness**. CAV 2014

Thank You