Classical and Non-Classical Uses of SAT in Model-Checking

CP meets CAV
Master Class

Jean-François Raskin
Université Libre de Bruxelles

Objectives

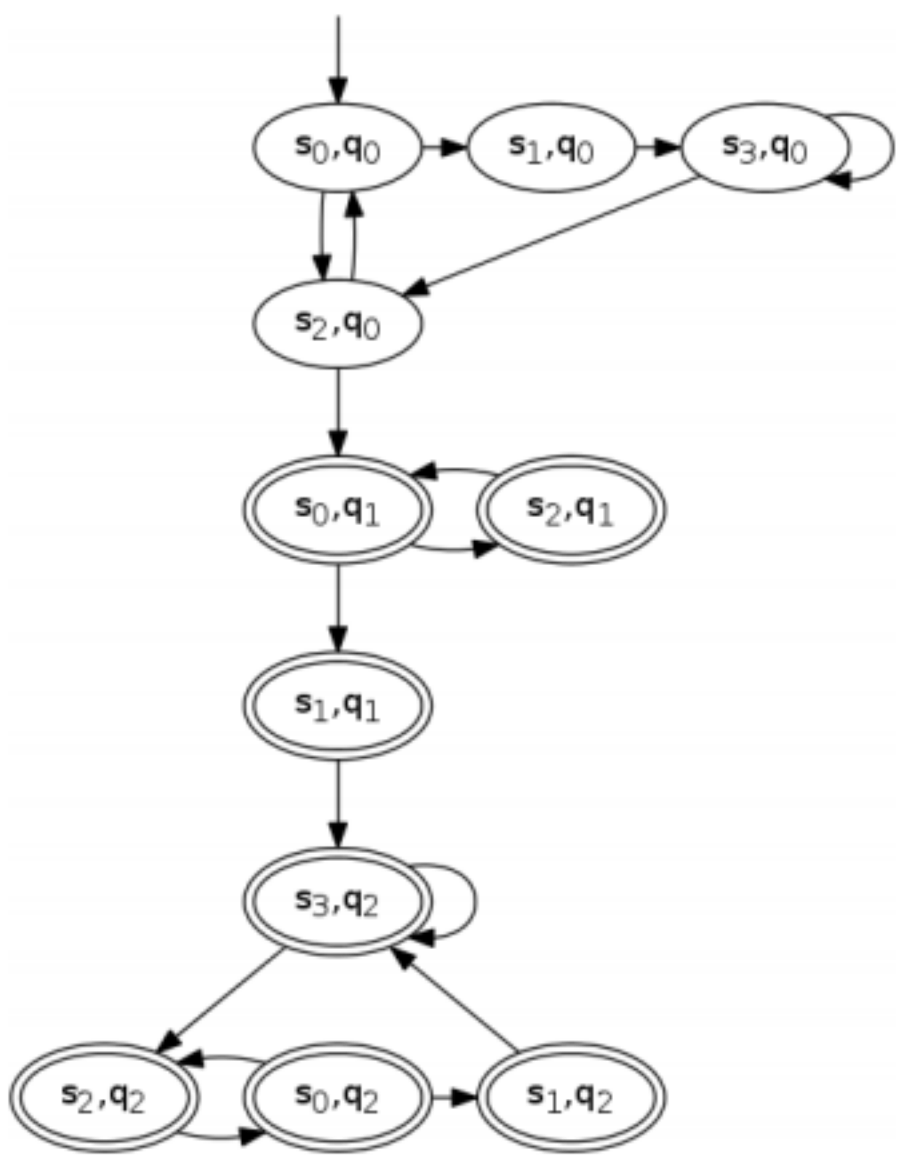
- Give representative **examples** of the use of SAT solvers in **verification algorithms** for finite state systems
- **Disclaimer I**: not my work
- **Disclaimer II**: by no means a full review of the literature (examples only)

Plan

- Bounded model-checking
- Unbounded model-checking
- Inductive invariant generation

Preliminaries

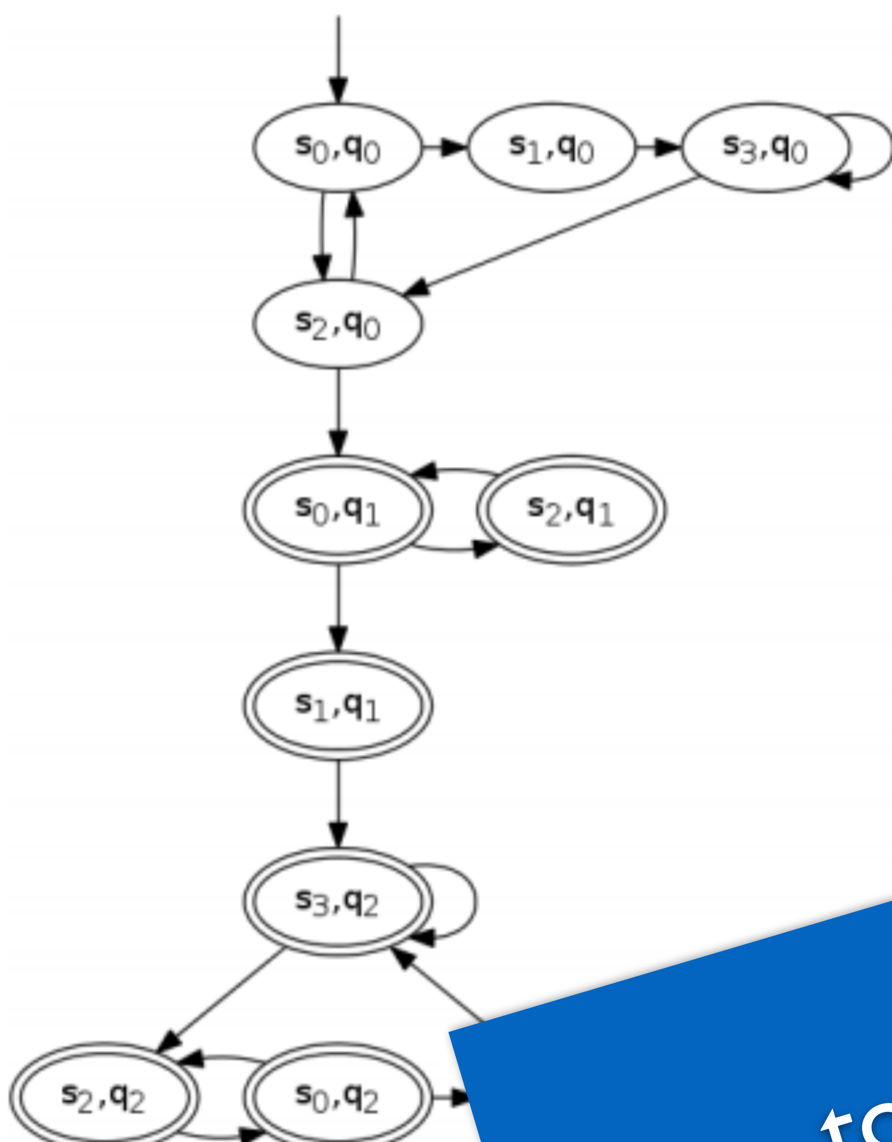
Transition Systems



The **basic** model
for CAV of reactive systems
=
Transition systems
=
Directed graphs with labels

Vertices = System/Prg states
Edges = transitions
from states to states
Labels = basic properties of states

Transition Systems



The **basic** model
for CAV of reactive systems
=
Transition systems
=
Directed graphs w

states
ransitions
rom states to states
Labels = basic properties of states

Usually far too large
to be represented explicitly

Symbolic transition systems

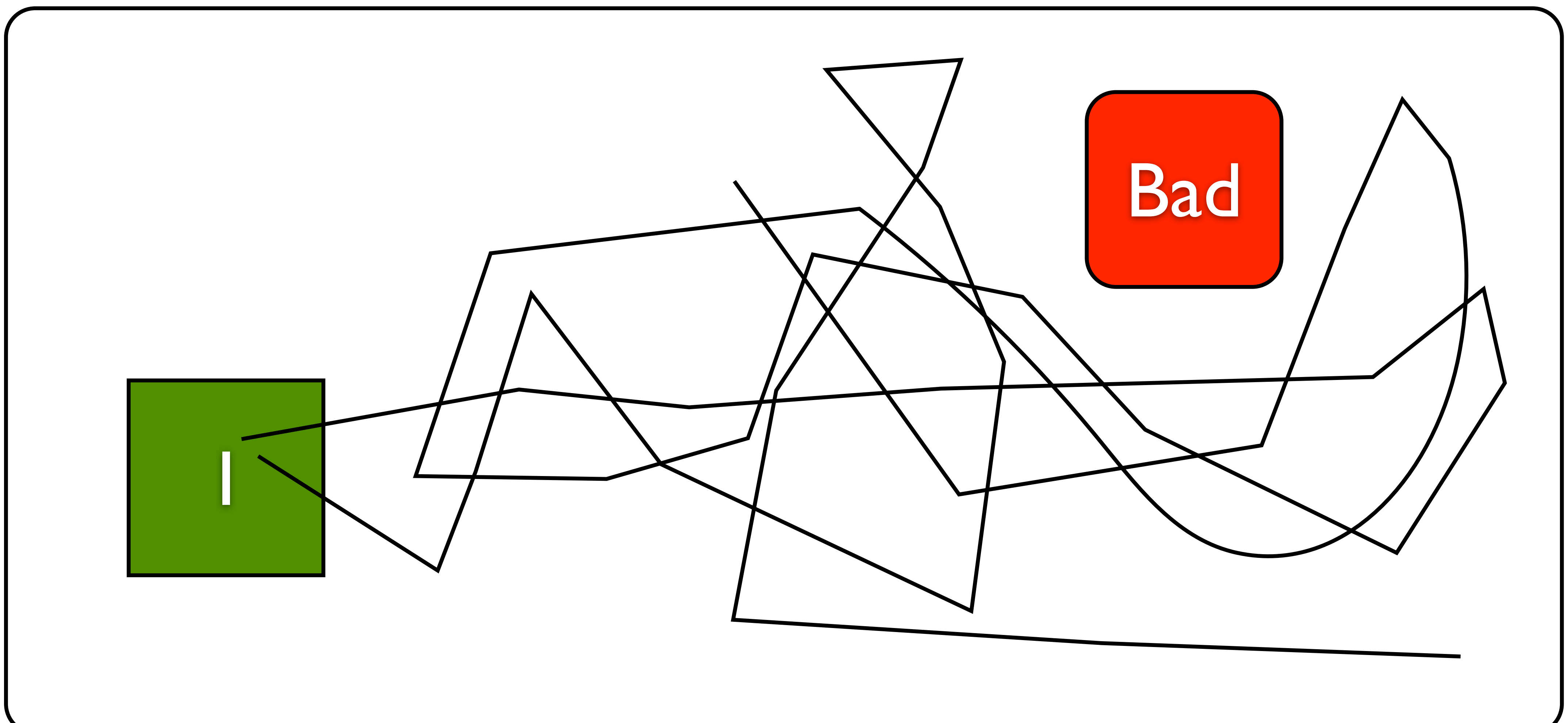
- Let $\mathfrak{B}(X)$ denotes the set of Boolean formulas over X, the variables of the system (or abstractions of them)
- For $F \in \mathfrak{B}(X)$, we note $[\![F]\!] = \{ v : X \rightarrow \{0,1\} \mid v \models F \}$
- A Symbolic Transition System (STS) S=(X,I,T) where:
  - X is a set of boolean variables
  - $I \in \mathfrak{B}(X)$ defines the initial states
  - $T \in \mathfrak{B}(X \cup X')$ defines the transition relation

Symbolic transition systems

- We associate to STS=(X,I,T) an explicit, so exponentially larger, transition system TS=(S,S_0,E):
  - $S = \{ v \mid v : X \to \{0,1\} \}$
  - $S_0 = \{ v \in S \mid v \models I \} = [\![S_0]\!]$
  - $E = \{ (v,v') \mid (v,v') \models T \} = [\![T]\!]$

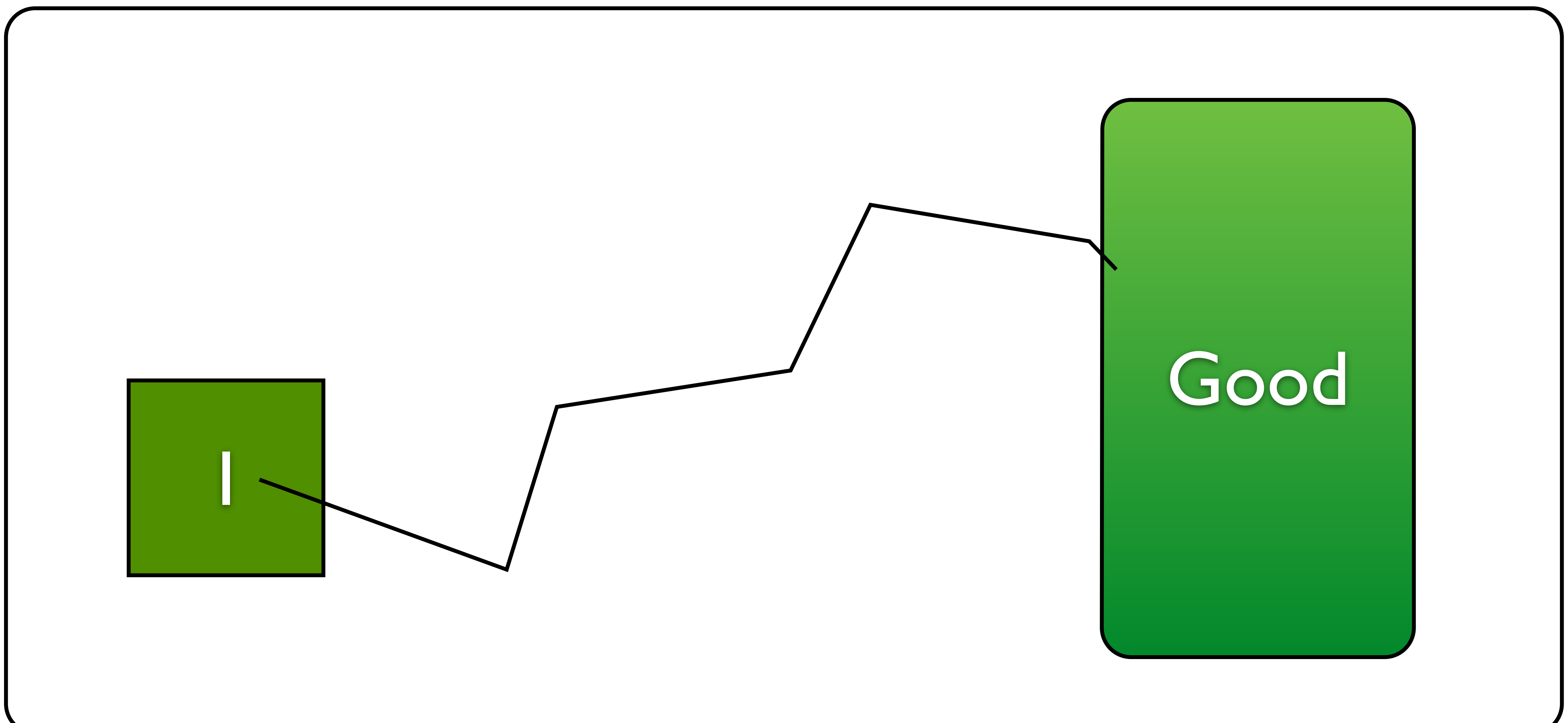
Typical verification questions

- **Safety**: do all the executions of the system avoid a given set of **bad** states ?



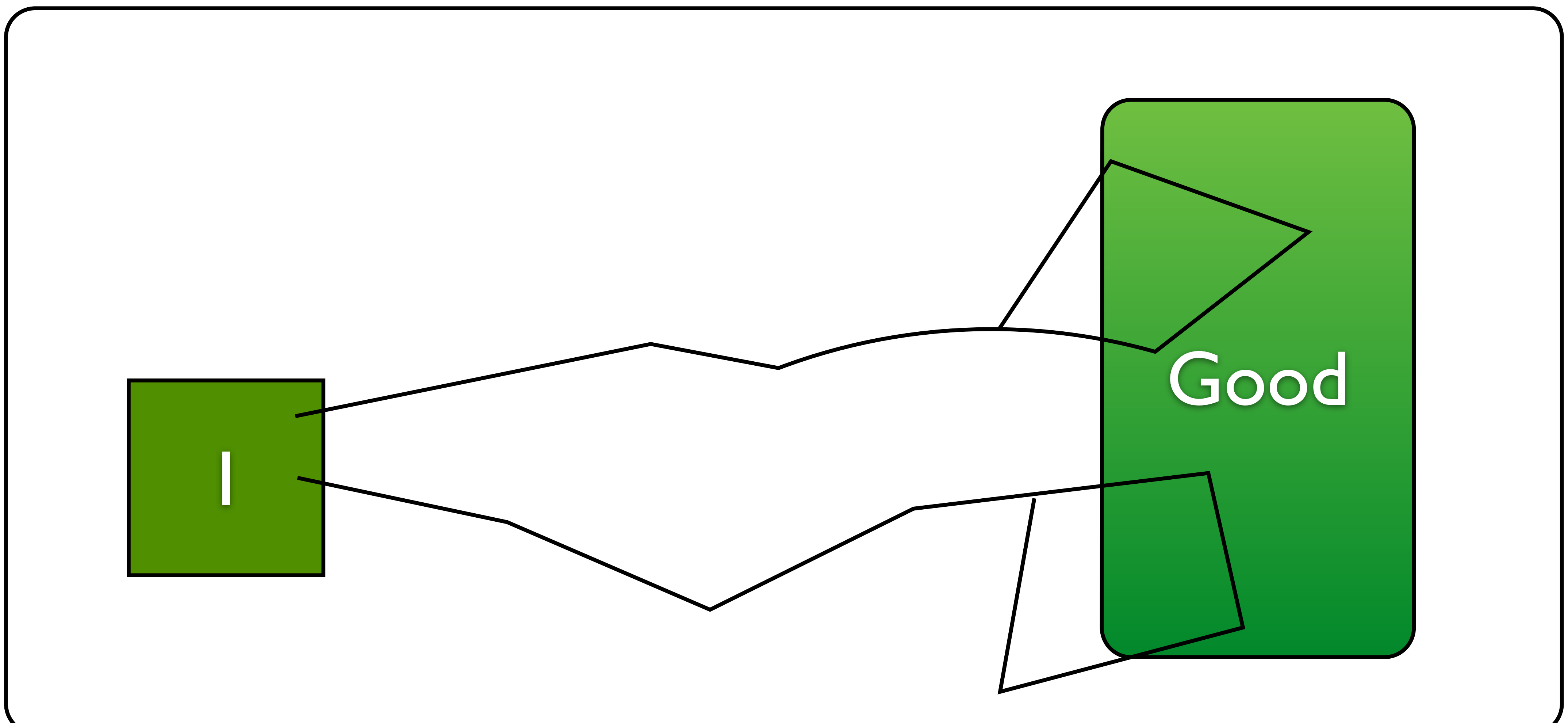
Typical verification questions

- **Reachability**: is there an execution of the system that reaches **good** states ? dual of safety

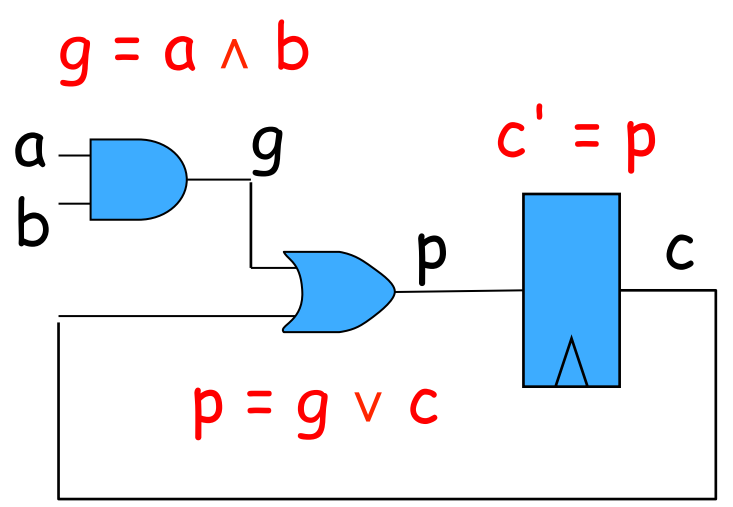


Typical verification questions

- **Liveness:** are all the executions of the system doing eventually/repeatedly something good ?



Circuit Example



From McMillan03

Model:

$C = \{$
$g = a \wedge b,$
$p = g \vee c,$
$c' = p$
$\}$

Can we reach a state of the circuit in which $c \wedge \neg p$ holds ?

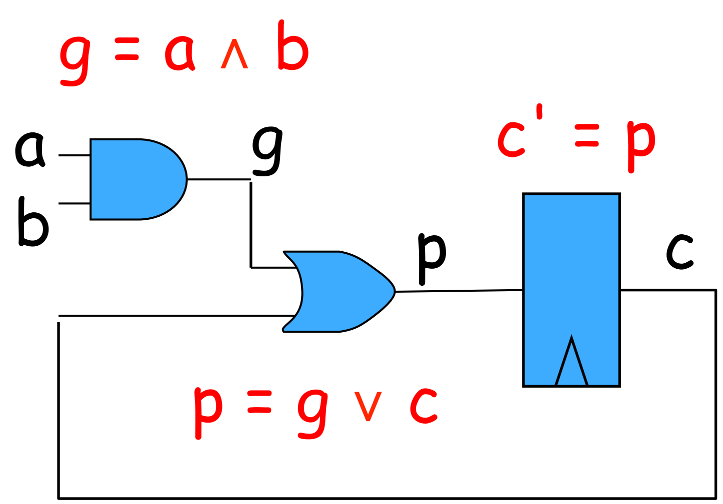
Bounded model-checking [BCC+99]

Bounded model-checking

- Falsifying **safety properties**
- Let STS=(X,I,T) and Bad ∈ $\mathfrak{B}$(X)
- Is there a ⟦T⟧-path from ⟦I⟧ to ⟦Bad⟧ ?
- **Bound**:

Is there a ⟦T⟧-**path from** ⟦**I**⟧ **to** ⟦**Bad**⟧
of length at most k ?

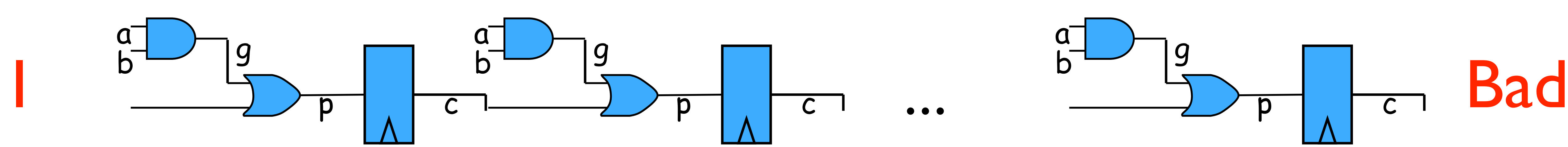
System unfolding



Model:

$C = \{$
$g = a \wedge b,$
$p = g \vee c,$
$c' = p$
$\}$

k unfolding



Can the circuit reach a state where c is true in at most k steps ?

Unfolding of T

- **Unfolding** of T k times:

  $T(X_0,X_1) \wedge T(X_1,X_2) \wedge \ldots \wedge T(X_{k-2},X_{k-1})$

- Use SAT solver to check **satisfiability** of

  $I(X_0) \wedge T(X_0,X_1) \wedge T(X_1,X_2) \wedge \ldots \wedge T(X_{k-2},X_{k-1}) \wedge \bigvee_{i=0..k-1} Bad(X_i)$

- A satisfying assignment corresponds to a path of length at most k from $[\![I]\!]$ to $[\![Bad]\!]$, i.e. a **counter-example** to the safety property

- Formulas above can easily be expressed as **sets of clauses** and so can be readily analyzed by a **Boolean SAT solver**

Completeness threshold

- **Diameter** of a system = length of the longest simple path in the transition system
- Bounded model-checking for safety property with a bound k=diameter of the system ensures **completeness**
- Unfortunately, computing the diameter of a symbolic transition system is hard. Indeed deciding if the diameter of a symbolic transition system is equal to k is **PSpace-C** (so as hard as the verification problem itself)

Beyond safety

- Let Good ∈ $\mathfrak{B}$(x)
- Given an infinite path ρ in TS, we note **Inf**(ρ) the set of states that appear infinitely many times along ρ
- An infinite path in TS is **good** if Inf(ρ)∩⟦Good⟧≠∅
- **Liveness**: check that all paths in TS are **good**
- Counter-examples are **lasso-path** such that the cycle does not contain any good states
- **Bound**: find a lasso-path of length at most k that does not cross ⟦Good⟧ in the lasso part

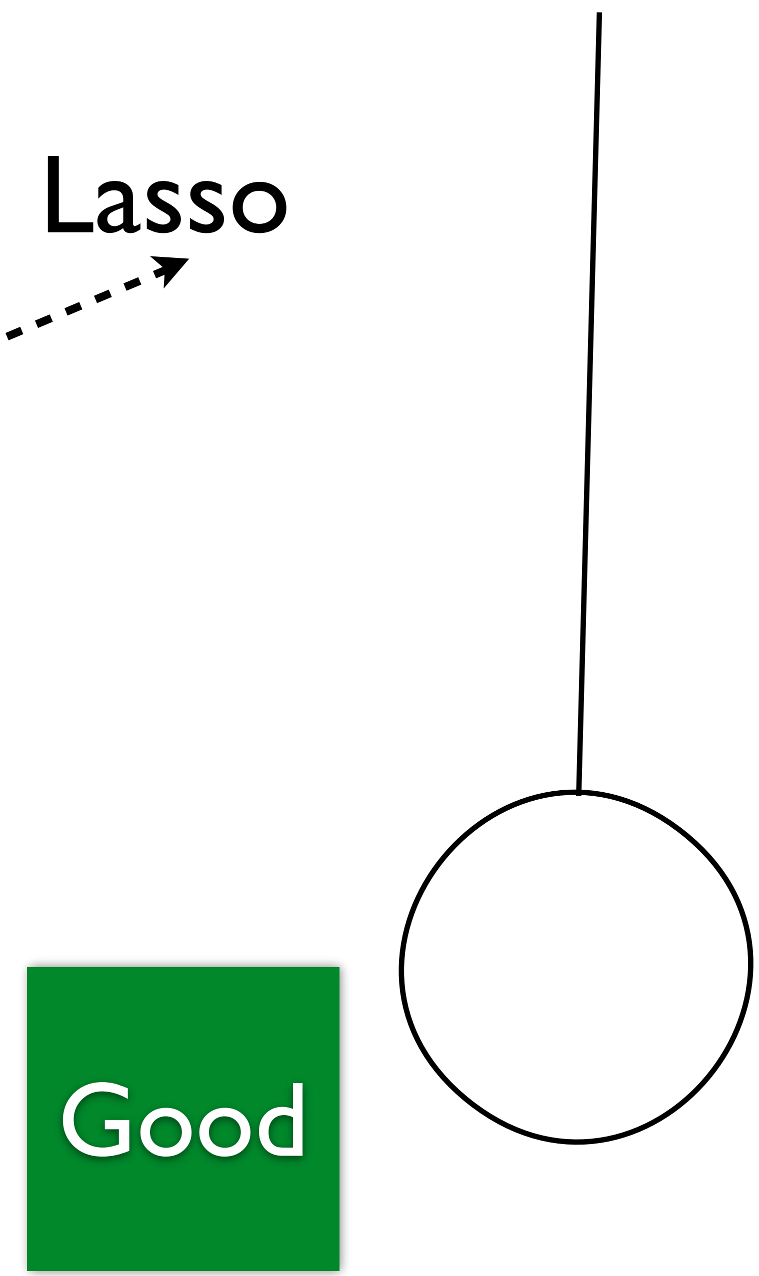
Beyond safety

- Encoding in SAT:

$$I(X_0)$$
$$\wedge T(X_0,X_1) \wedge \ldots \wedge T(X_{k-2},X_{k-1})$$
$$\wedge \bigvee_{m=0..k-1} T(X_{k-1},X_m)$$
$$\wedge_{j=m..k-1} \neg Good(Xj)$$

Lasso

Liveness is violated



Beyond safety

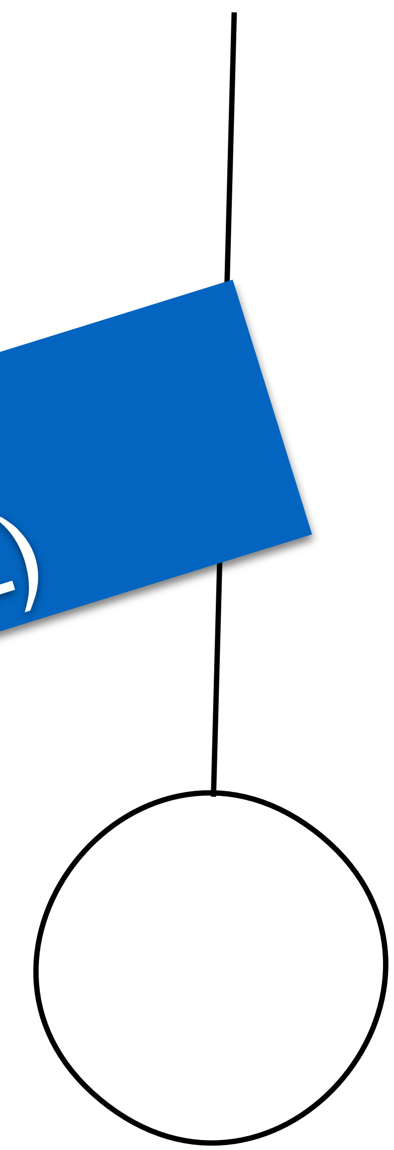
- Encoding in SAT:

Lasso

$I(X_0)$
$\wedge T(X_0,X_1) \wedge \ldots \wedge T(X_{k-2},X_{k-1})$
$\wedge \ \bigvee_{m=0..k-1} T(X_{k-1},X_m)$

$\wedge_{i=}$

...s violated



Good

All this can be extended to
linear temporal logic specifications (LTL)

Unbounded Model-Checking

Four examples of unbounded SAT based MC

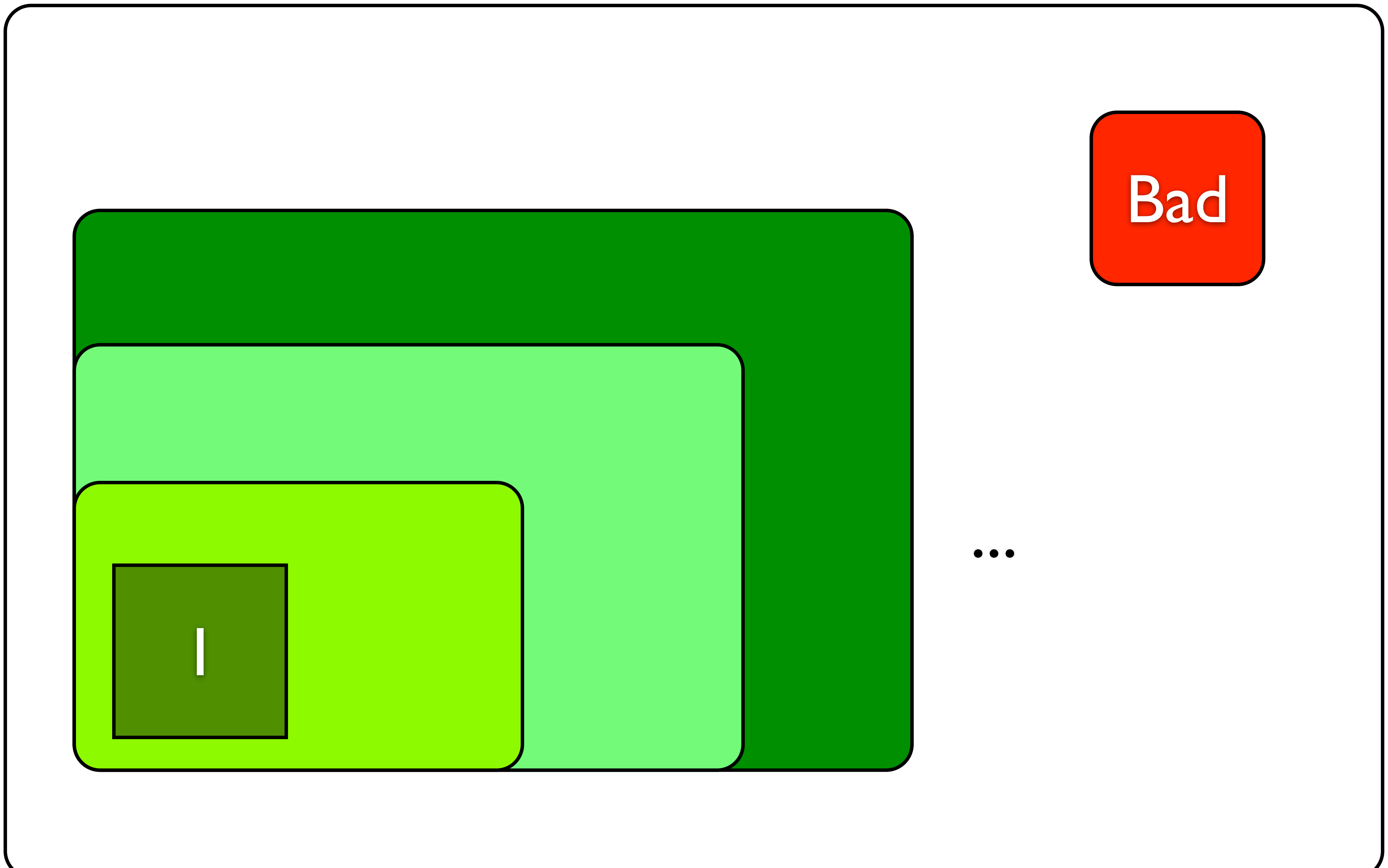
- Symbolic Reachability Analysis based on SAT Solvers [ABE00]
- Unbounded Sat-based model-checking with abstractions [CCKSVW02] + McMillan variant
- Interpolation and unbounded SAT-based model-checking [McMillan03]
- Discovering inductive invariants in subset constructions

Symbolic Reachability Analysis based on SAT Solvers [ABE00]

Symbolic Forward/Backward Reachability

- Let STS=(X,I,T) and let Bad ∈ $\mathfrak{B}$(X)
- **ReachFwd**(I) is the least set of states R such that R=⟦I⟧∪Post⟦T⟧(R)

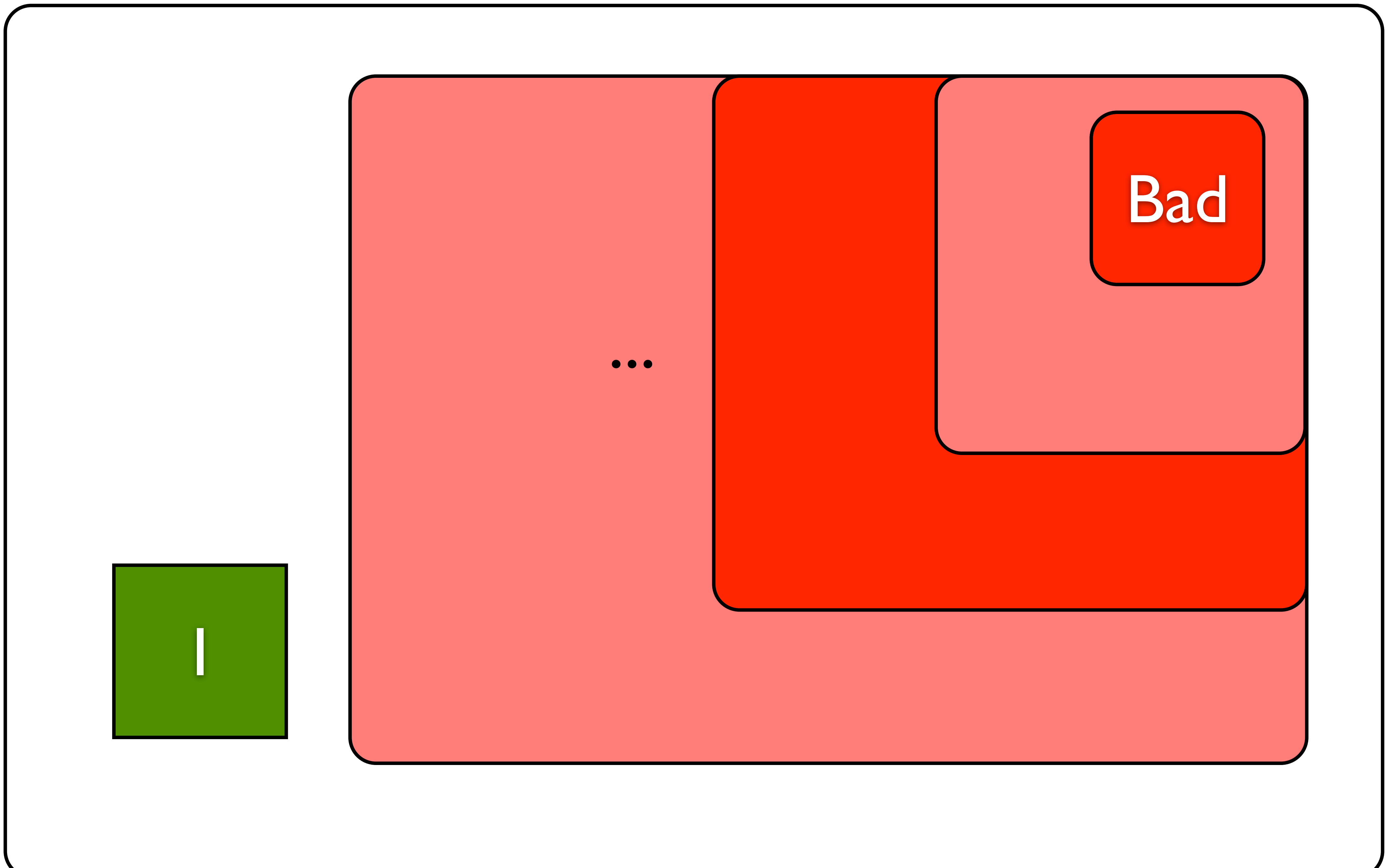
Forward exploration
Post



Symbolic Forward/Backward Reachability

- Let STS=(X,I,T) and let Bad ∈ $\mathfrak{B}$(X)
- **ReachFwd**(I) is the least set of states R such that R=⟦I⟧∪Post⟦T⟧(R)
- **ReachBack**(Bad) is the least set of states B such that B=⟦Bad⟧∪Pre⟦T⟧(B)

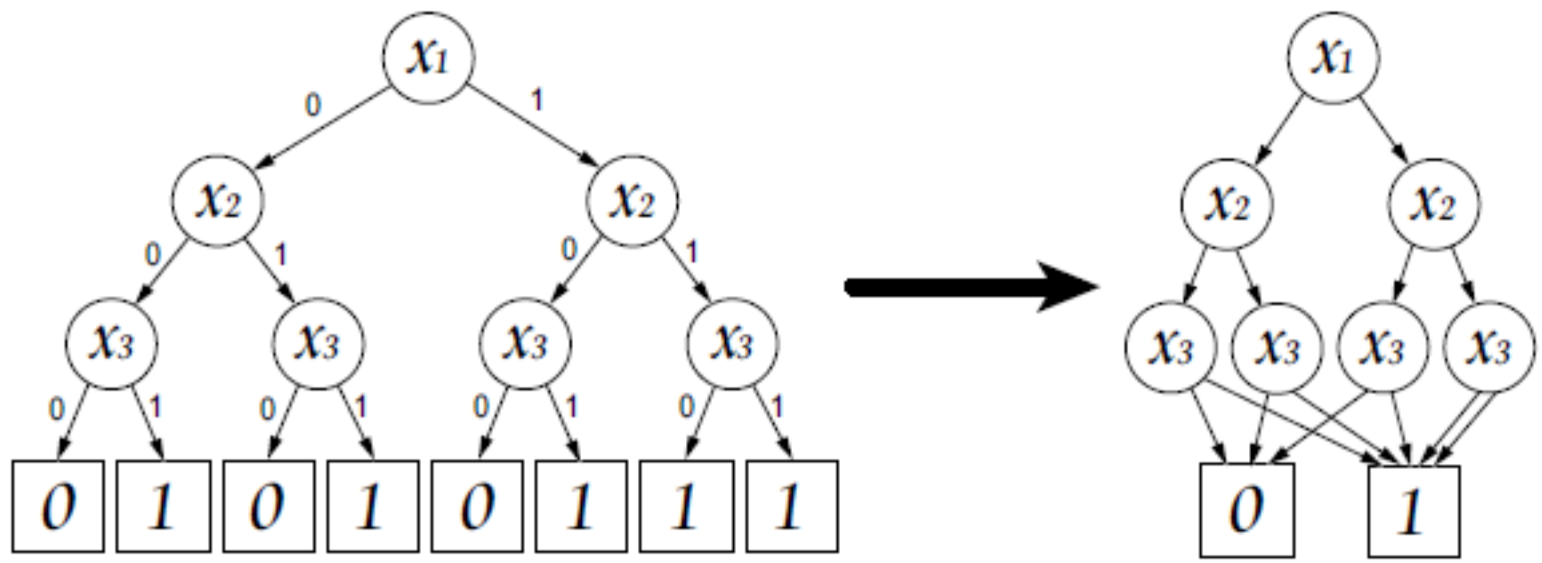
Backward exploration
Pre



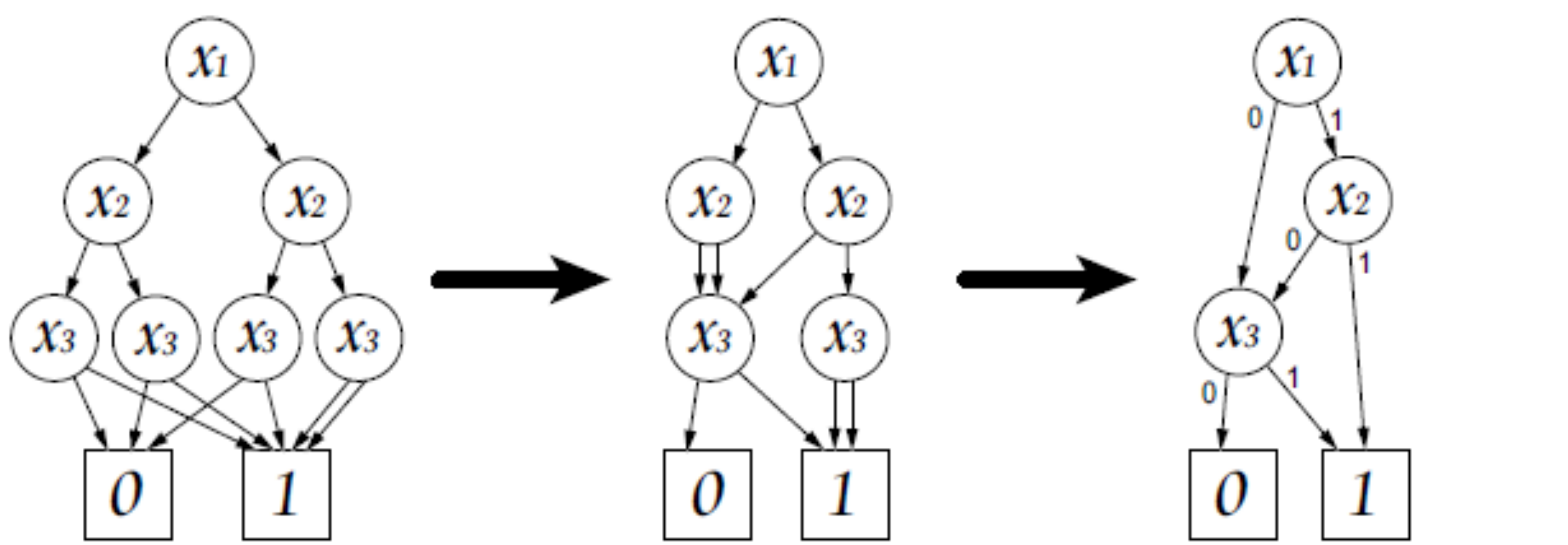
Symbolic Forward/Backward Reachability

- Let STS=(X,I,T) and let Bad $\in \mathfrak{B}(X)$
- **ReachFwd**(I) is the least set of states R such that $R = [\![I]\!] \cup Post[\![T]\!](R)$
- **ReachBack**(Bad) is the least set of states B such that $B = [\![Bad]\!] \cup Pre[\![T]\!](B)$
- Symbolic MC: fixpoints+**data structures** for manipulating sets symbolically

BDDs



share suffixes

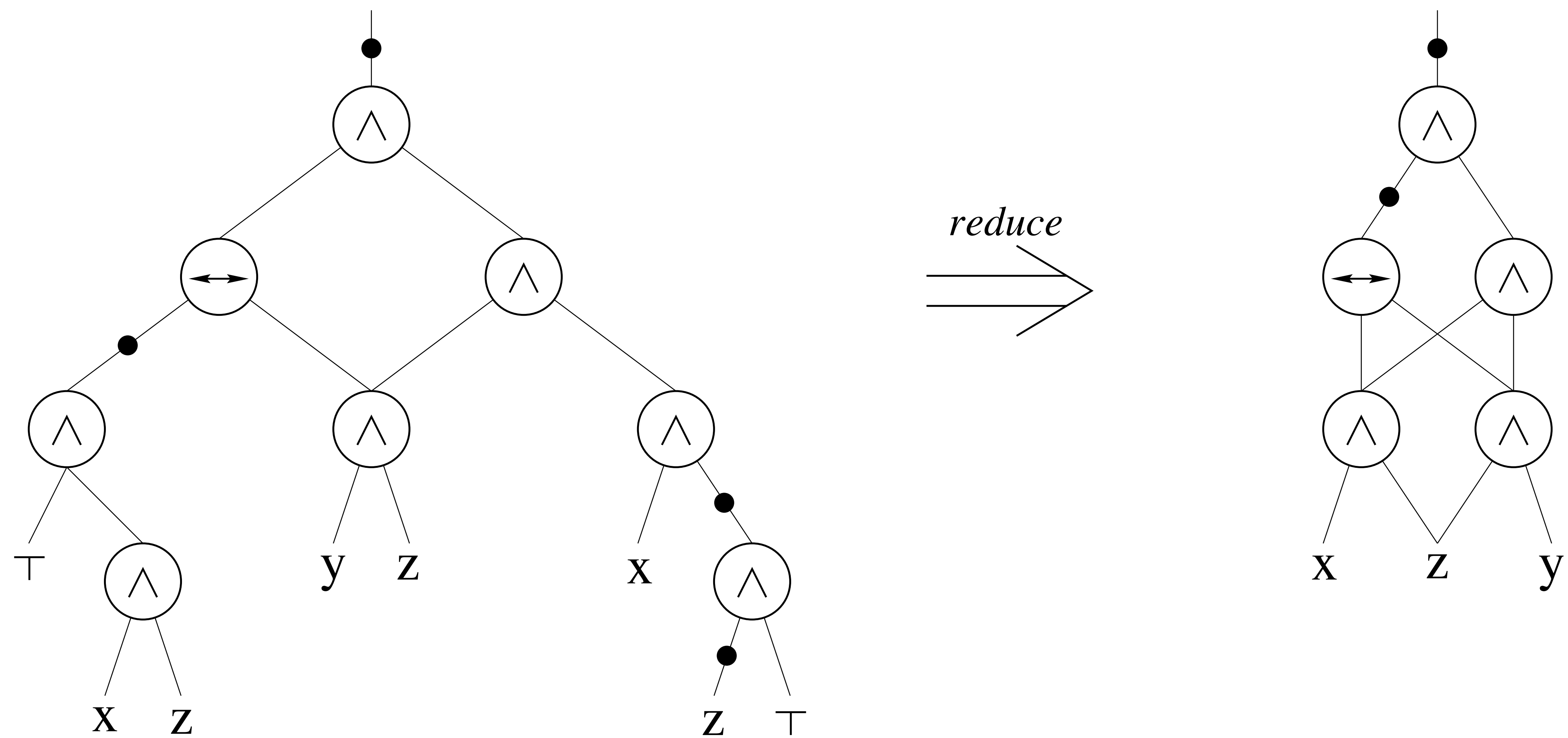


remove unnecessary tests

BDDs - Canonicity and Succinctness

- BDDs are **canonical** representation for Boolean functions
- Make very **easy** to check fixed-point
- Fact: some Boolean functions have **provably large** BDD representations, e.g. binary multiplication
- **Idea**: use potentially more compact representations... at the expense of **canonicity** and (maybe) some algorithmic efficiency

Boolean circuits



Boolean circuits

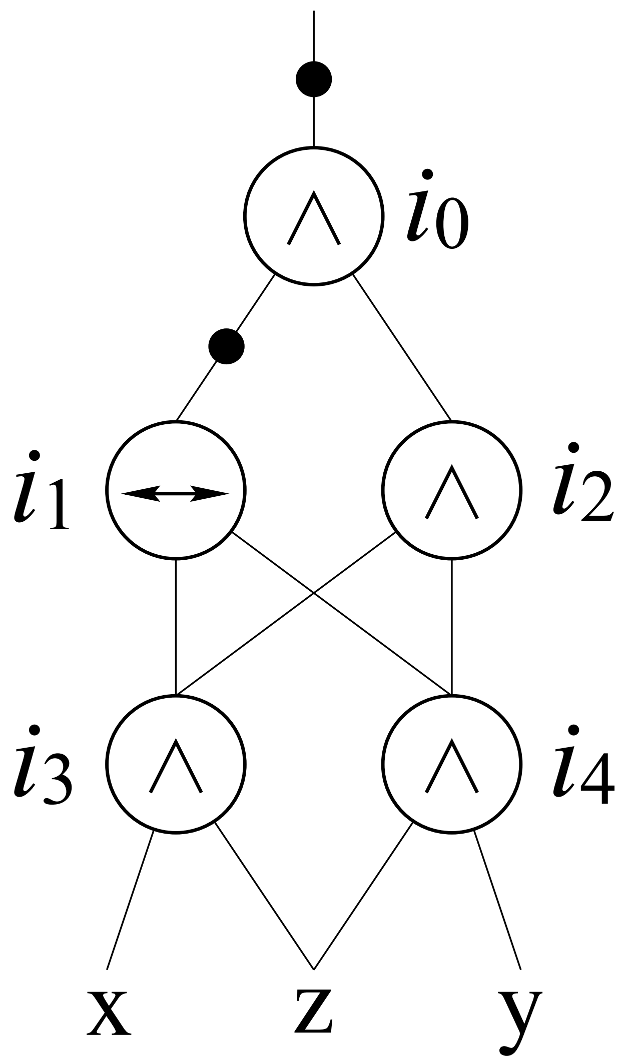
- As BDDs, **Boolean circuits** represent sets of valuations (=states)
- There is **no** (useful) canonical form
- There are often **more compact** than BDDs
- Algorithms exists for Boolean op. (obviously) and for computing PRE and POST images
- Satisfiability is **NP-Complete**

Boolean circuits

- As BDDs, **Boolean circuits** represent sets of valuations (=states)
- There is **no** (useful) canonical form
- There are often **more compact** than BDDs
- Algorithms exists for Boolean op. (obviously) and for computing PRE and POST images
- Satisfiability is **NP-Complete**

→ **use SAT**

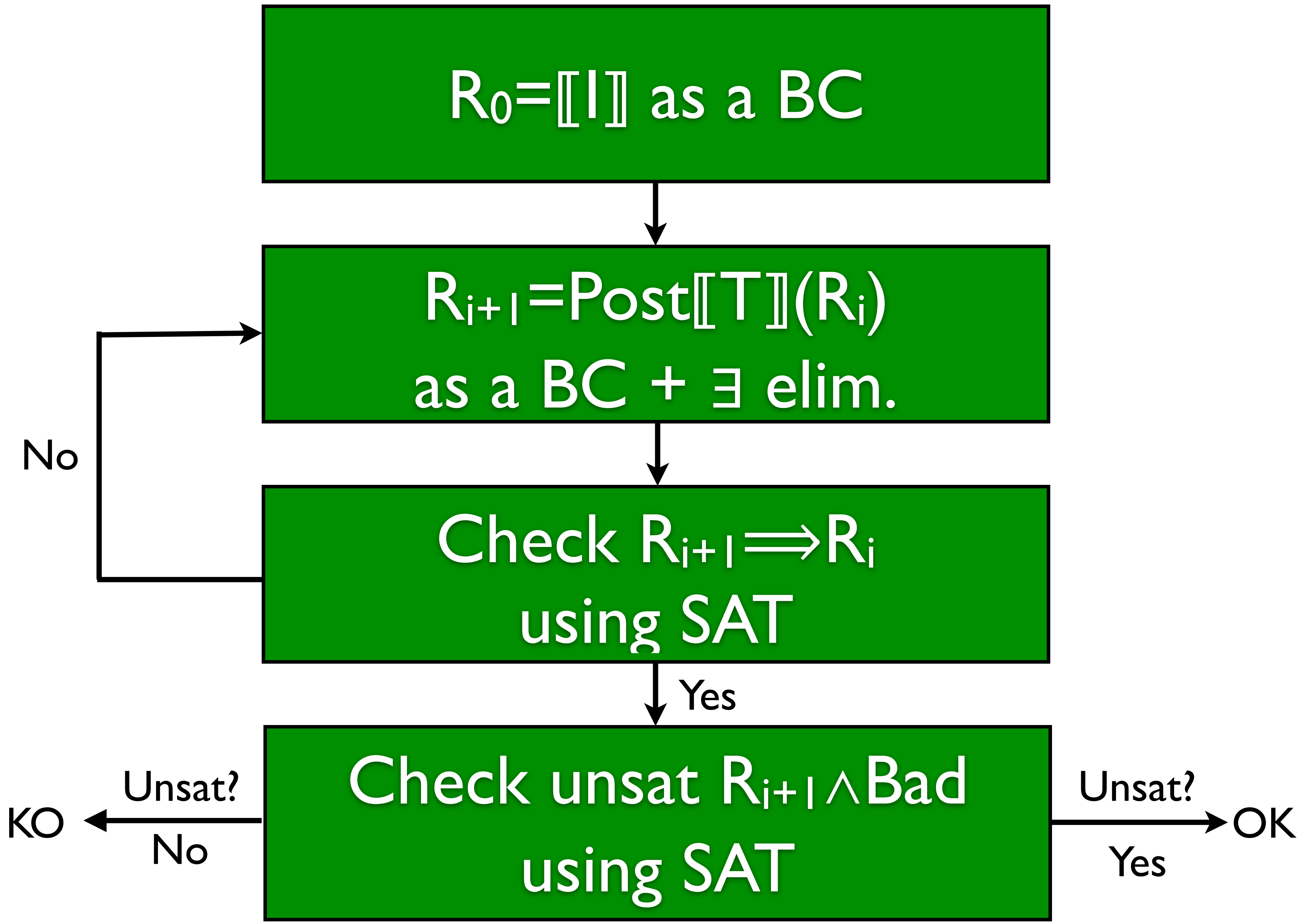
Checking satisfiability of Boolean circuits with SAT



$$
\begin{aligned}
&(i_0 \leftrightarrow \neg i_1 \wedge \ i_2) \\
\wedge\ &(i_1 \leftrightarrow i_3 \leftrightarrow i_4) \\
\wedge\ &(i_2 \leftrightarrow i_3 \wedge i_4) \\
\wedge\ &(i_3 \leftrightarrow x \wedge z) \\
\wedge\ &(i_4 \leftrightarrow z \wedge y) \\
\wedge\ &\neg i_0
\end{aligned}
$$

Not equivalent but **satisfiability** is maintained

SMC algorithm using BC and SAT

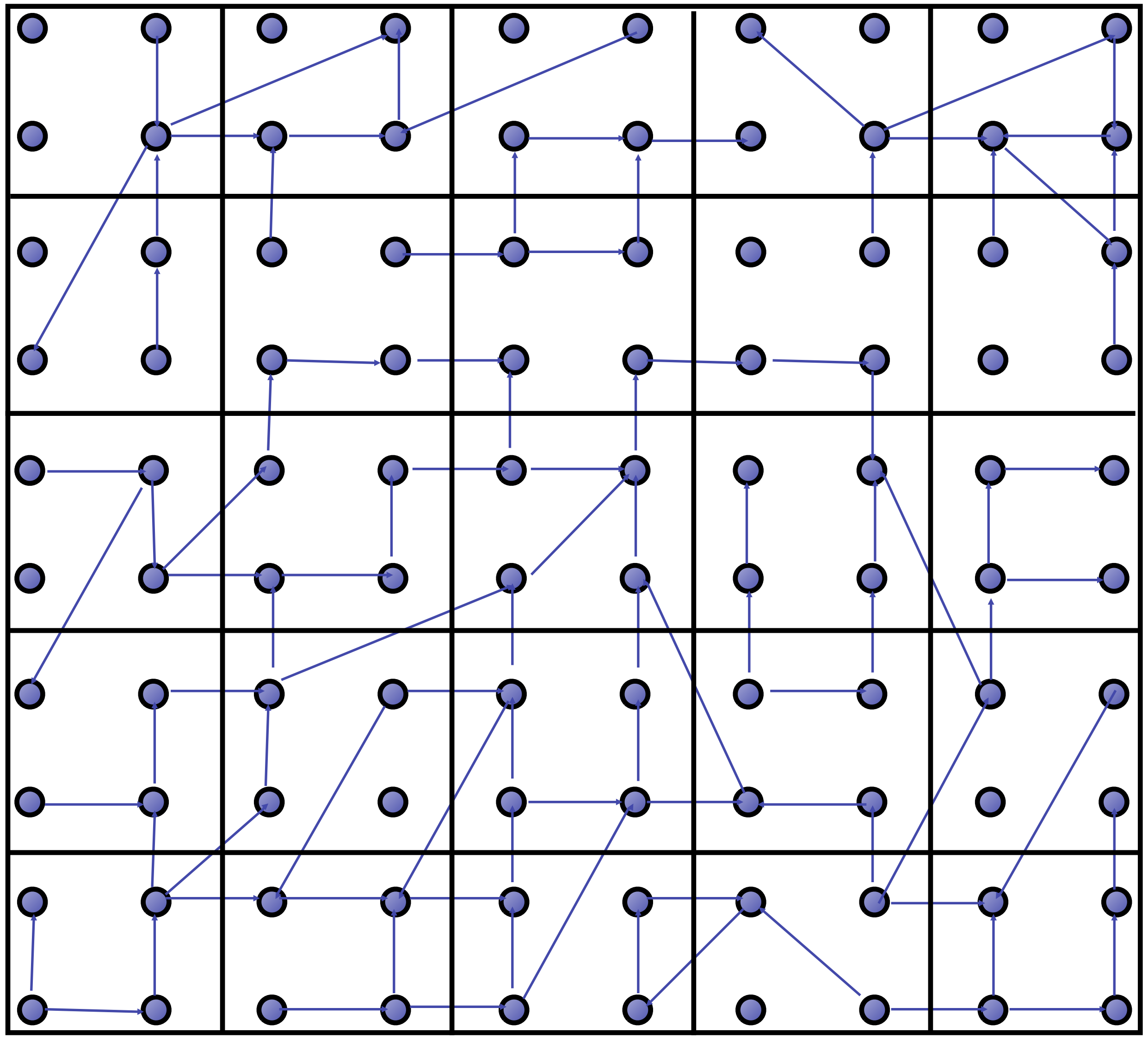


Unbounded SAT-based model-checking with abstractions [CCKSVW02]

Abstractions

- Symbolic model-checking sensitive to the **number** of Boolean variables (symbolic state explosion problem)
- But (coarse) abstractions are often **sufficient** to prove correctness
- Try to **lower the number of variables** using abstraction

State-space partitioning

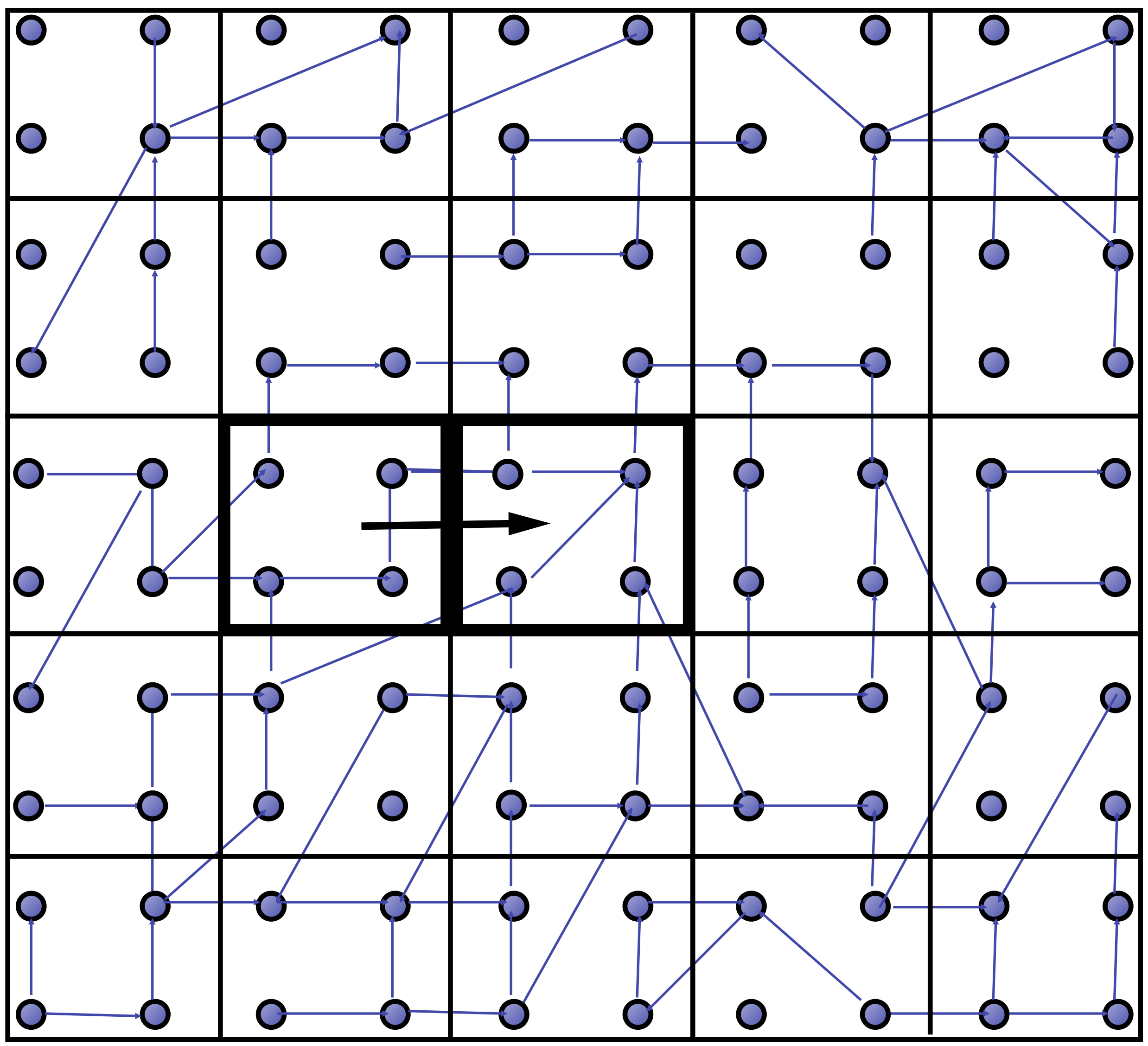


☛ **Predicates** on program/circuit state space

☛ States satisfying the same predicates are (considered) **equivalent**

☛ Merged into one **abstract** state

State-space partitioning



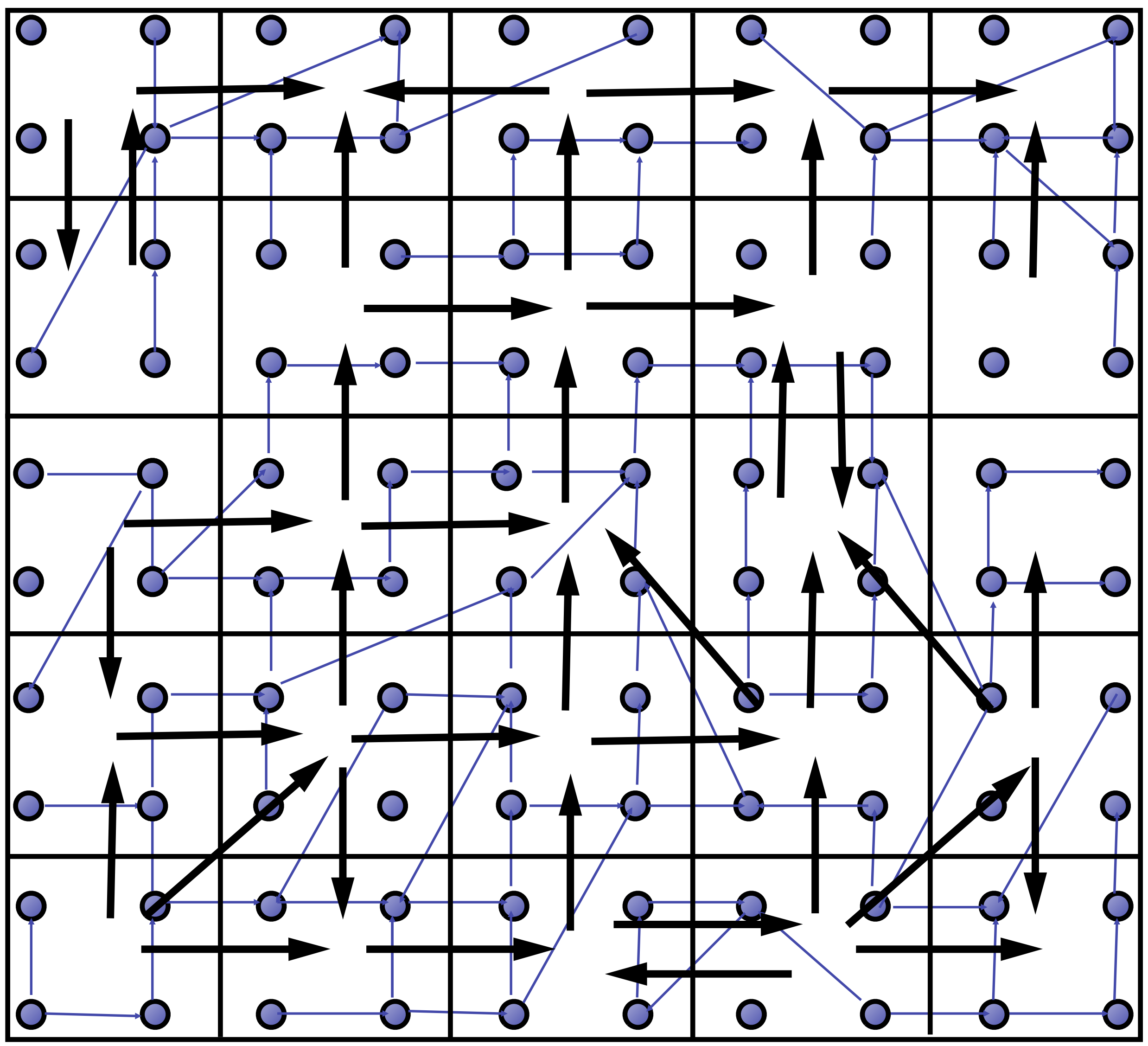
Abstract transition relation

$$T^{\alpha}(A_1, A_2)$$

iff

$$\exists_{s_1 \in A_1} \cdot \exists_{s_2 \in A_2} \cdot T(s_1, s_2)$$

State-space partitioning



Existential Lifting

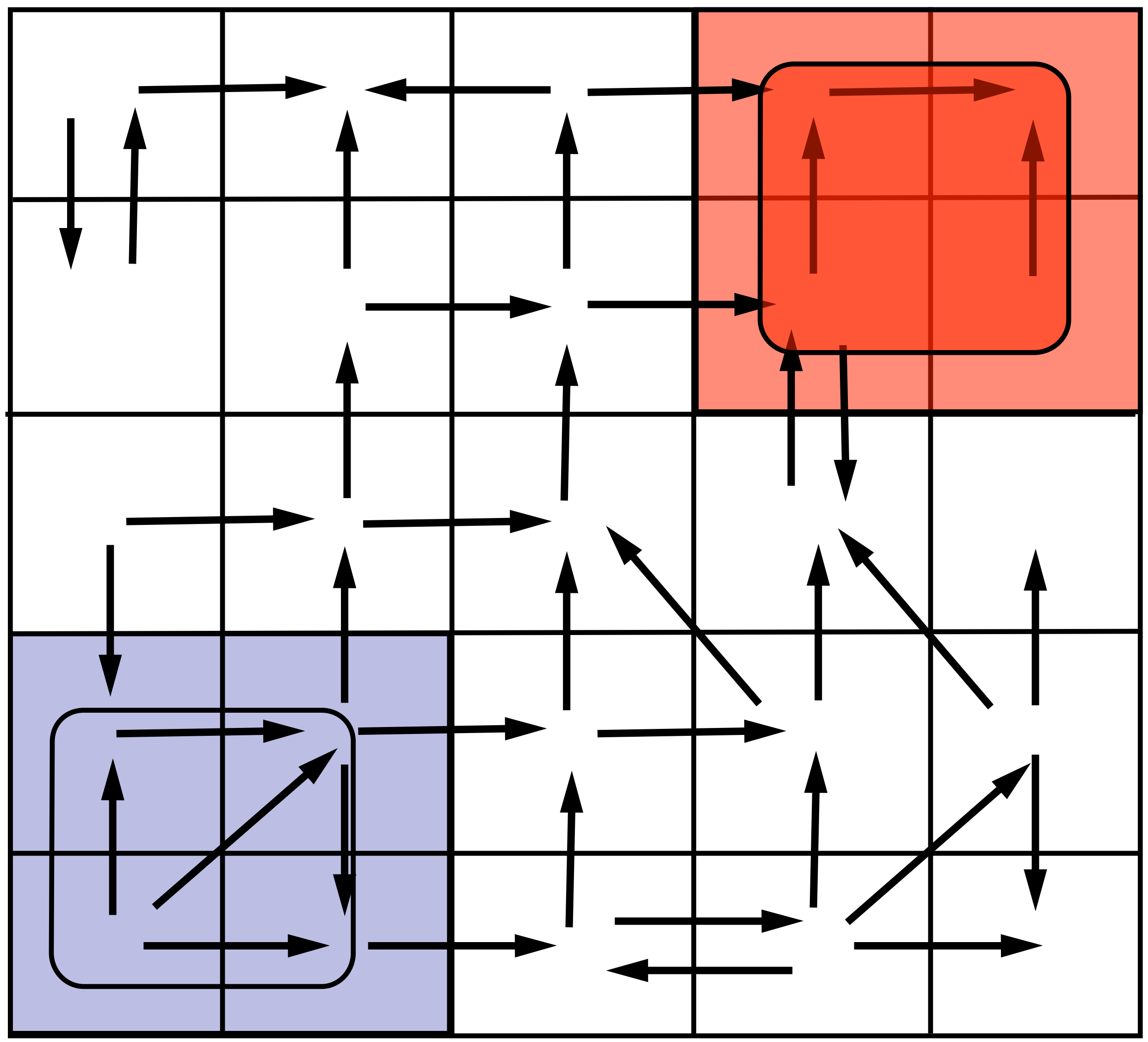
Abstract transition relation

$T^{\alpha}(A_1,A_2)$

iff

$\exists_{s_1 \in A_1} \cdot \exists_{s_2 \in A_2} \cdot T(s_1,s_2)$

State-space partitioning



Analyze the abstract graph

Overapproximation:

Safe ⇒ System Safe

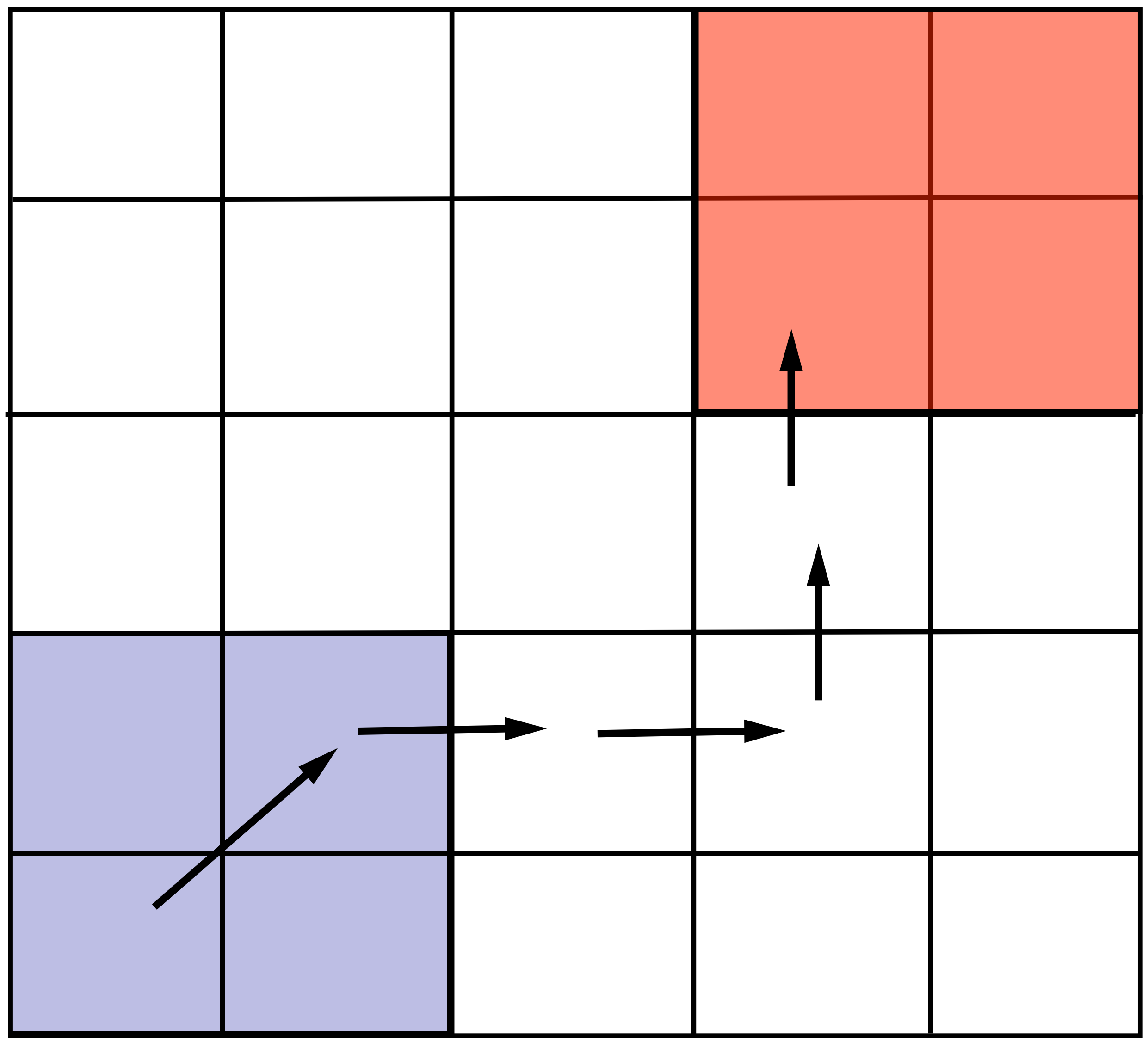
No false positives

Problem

Spurious counterexamples

Counterex.-Guided Refinement

[Kurshan et al93] [Clarke et al 00][Ball-Rajamani 01]

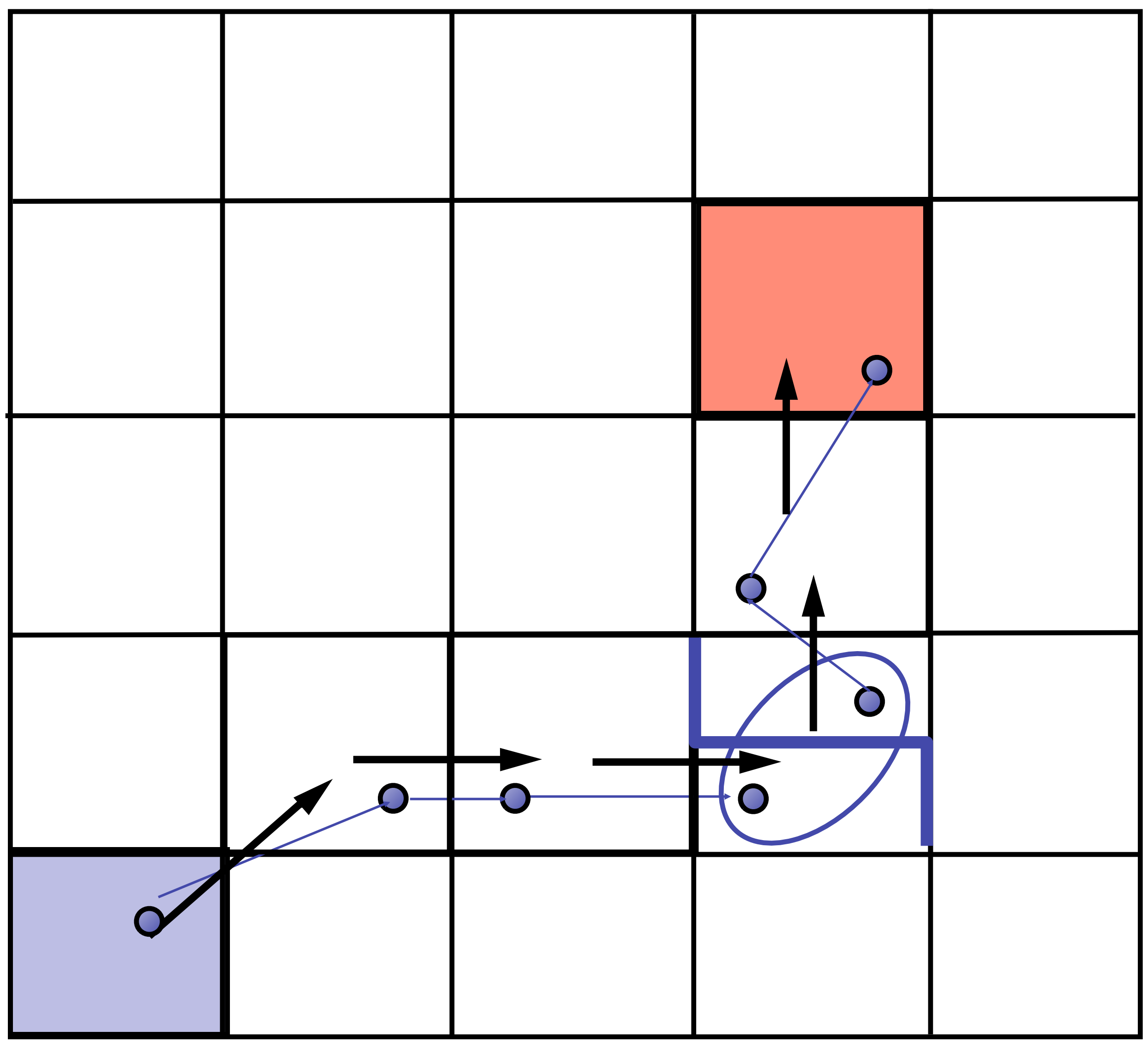


Solution

Use spurious counterexamples to refine abstraction !

Counterex.-Guided Refinement

[Kurshan et al93] [Clarke et al 00][Ball-Rajamani 01]



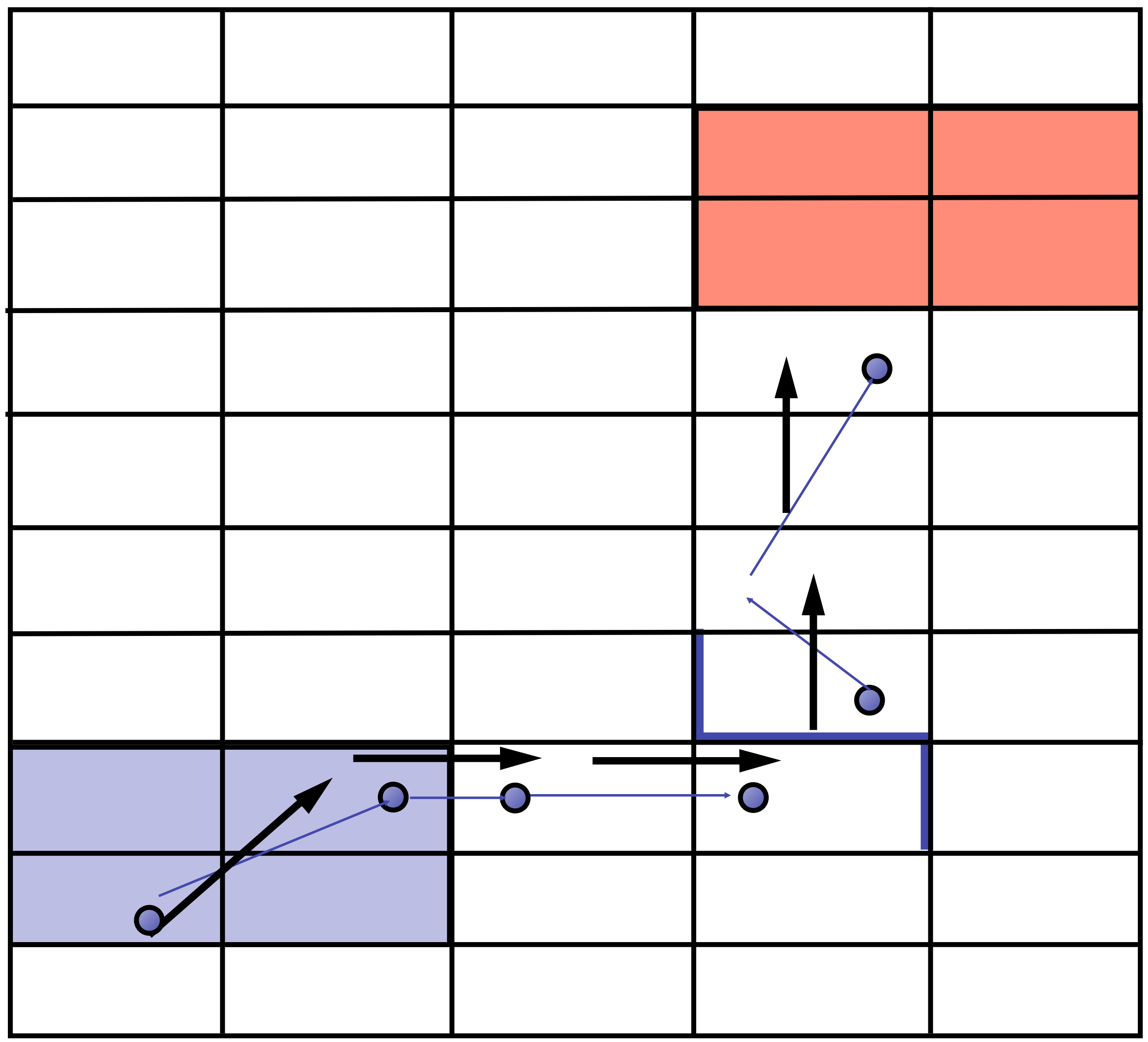
Solution

Use spurious **counterexamples** to **refine** abstraction

1. **Add predicates** to distinguish states across **cut**
2. Build **refined** abstraction

Imprecision due to **merge**

Iterative Abstraction-Refinement



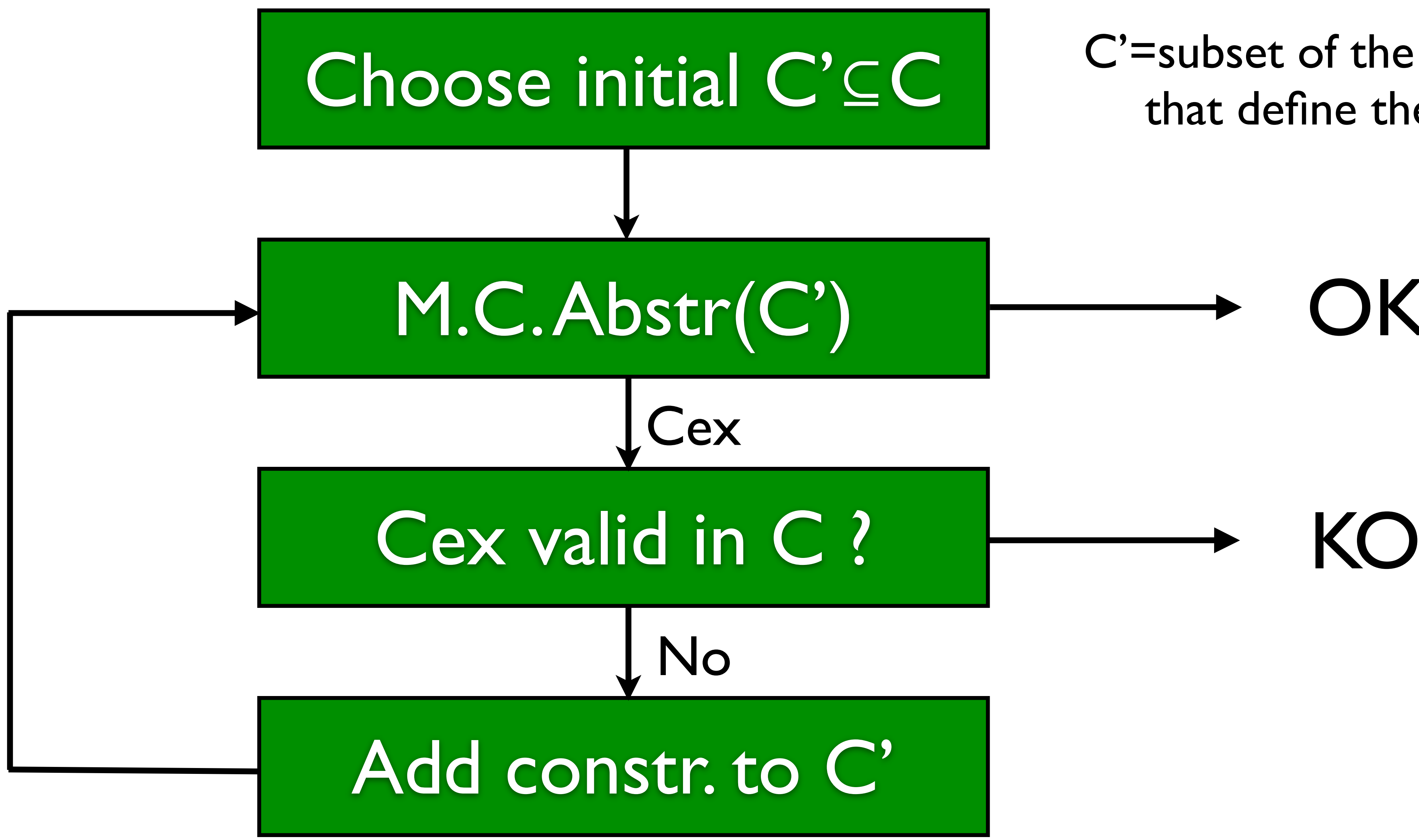
Solution

Use spurious **counterexamples** to **refine** abstraction

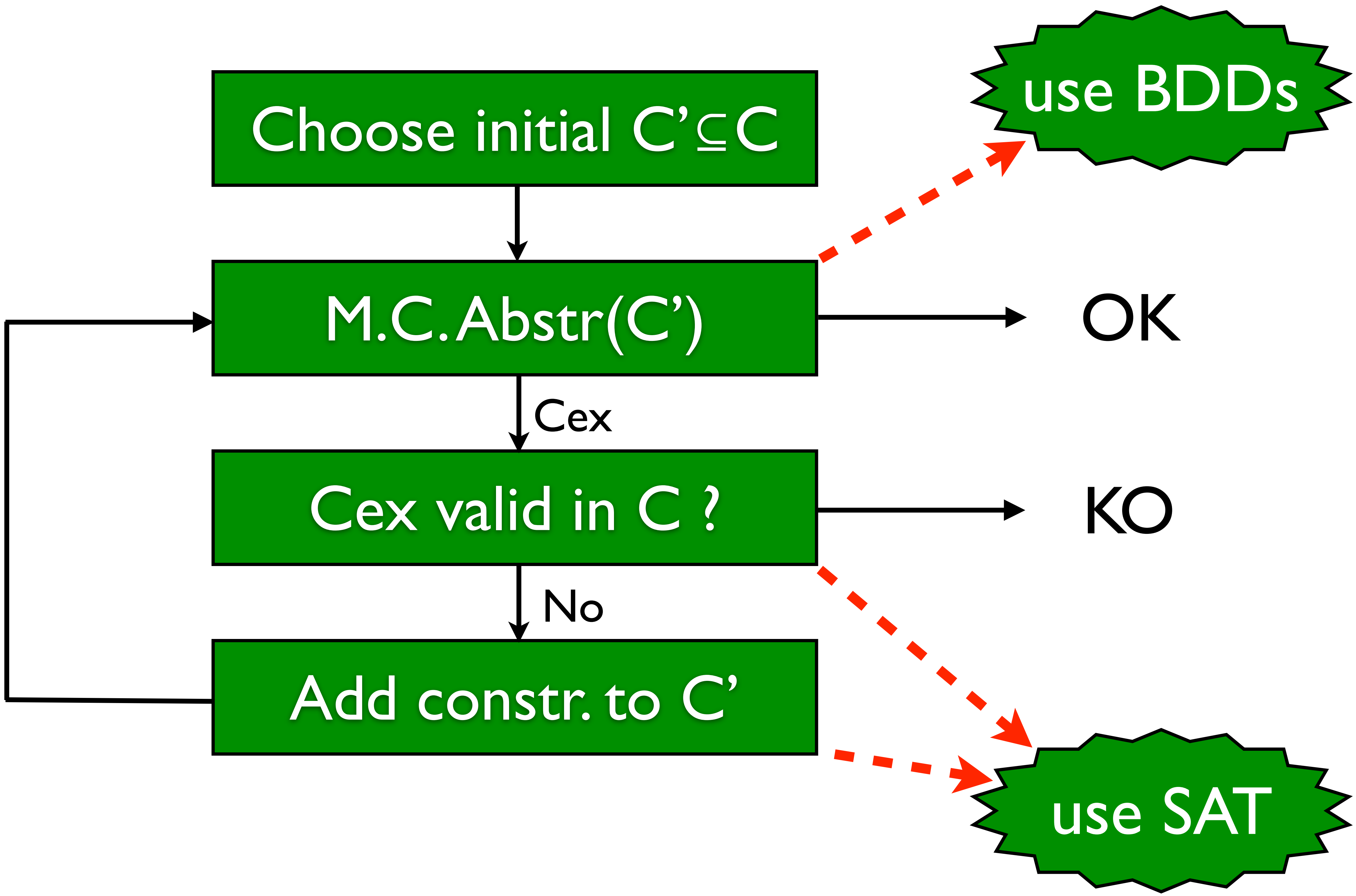
1. Add predicates to distinguish states across **cut**
2. Build **refined** abstraction
   - -eliminates counterexample
3. **Repeat** search
   Till real counterexample or system proved safe

Abstraction refinement

C'=subset of the constraints
that define the system



Abstraction refinement



Abstract Cex - Safety

- **Abstract variables** Y=Support(C',I,Bad)
- Abstract system is model-checked using BDD-based symbolic MC with variables in Y only and $|Y| \ll |X|$
- Abstract counter-example is a truth assignment to $\{ y_t \mid y \in Y \wedge 0 \leq t \leq \mathbf{k} \}$ where k is the number of steps in the counter-example

Concretization of Cex

- The abstract Cex $\mathbf{A}^\alpha$ satisfies:
  $\mathbf{A}^\alpha\mathbf{(Y)}= I(Y_0) \wedge T_{0..k-1}(Y_0,\ldots,Y_{k-1}) \wedge \bigvee_{i=0..k-1} \mathbf{Bad}(Y_i)$
- Search for a concrete A consistent with $\mathbf{A}^\alpha$:

  $\mathbf{A}^\alpha\mathbf{(Y)} \wedge I(X_0) \wedge T_{0..k-1}(X_0,\ldots,X_{k-1}) \wedge \bigvee_{i=0..k-1} \mathbf{Bad}(X_i)$

  =BMC but **guided** by the abstract Cex
- If unsat Cex cannot be made concrete and it is **spurious**

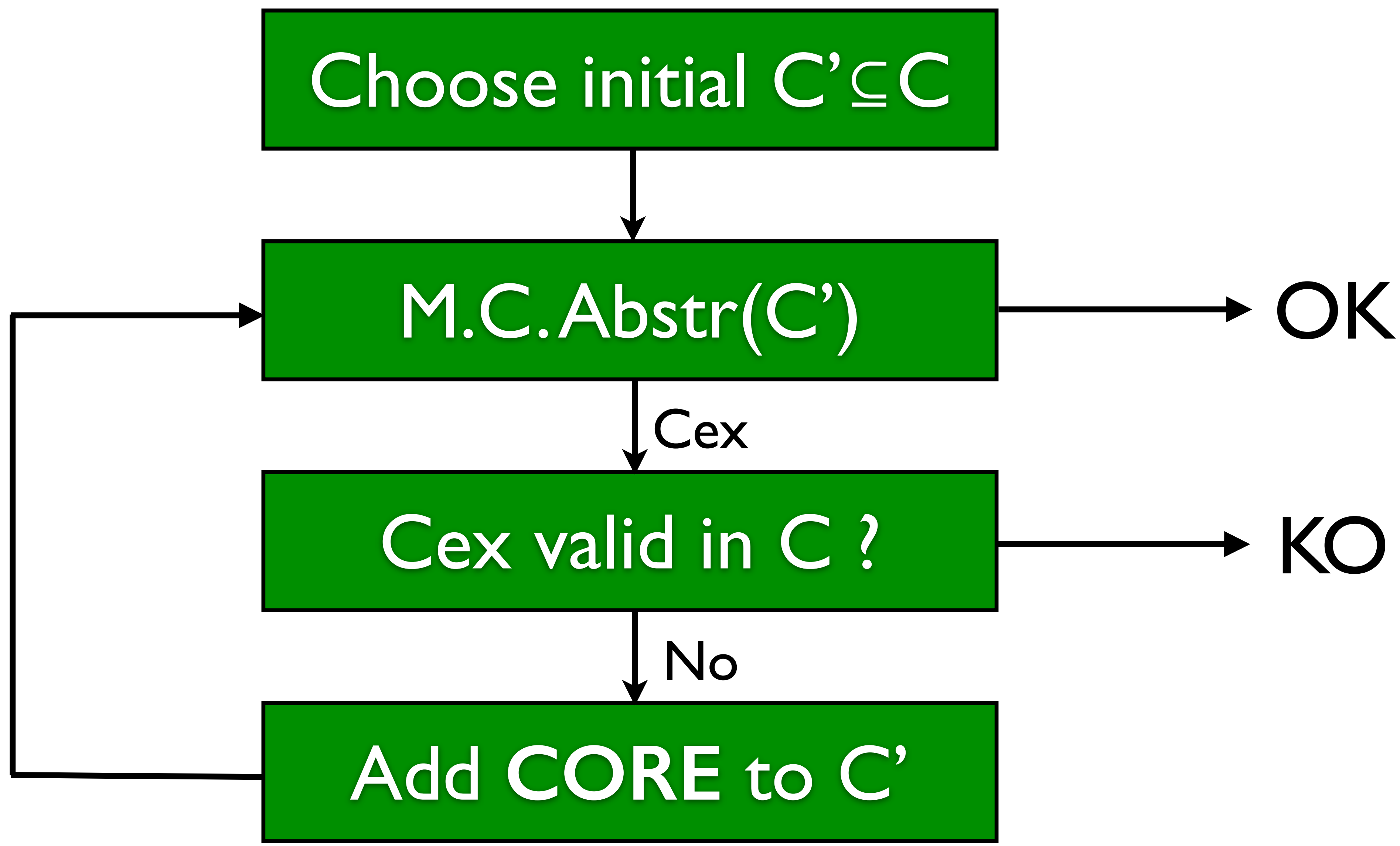
Refinement

- Refinement: **add** constraints to C'
- Goal: to **eliminate** the Cex in the next abstract model
- There are many technics for that
- One based on SAT machinery: use **resolution based refutation** of the unsat formula that defines the concretization of the abstract counter-example

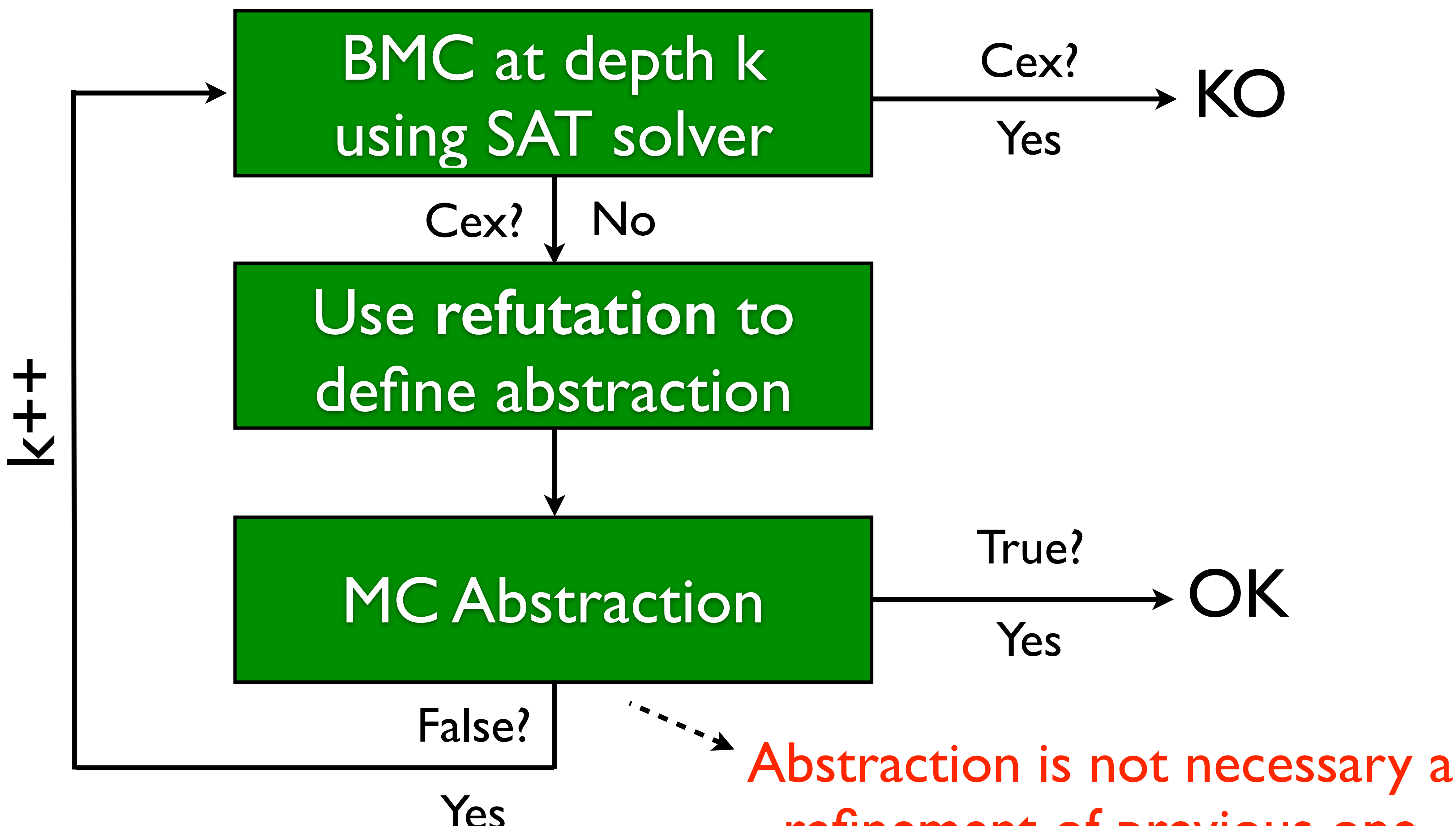
Resolution based refinement

- $A^\alpha(Y) \wedge I(X_0) \wedge T_{0..k-1}(X_0,\ldots,X_{k-1}) \wedge \vee_{i=0..k-1} Bad(X_i)$ is **unsatisfiable**
- SAT solver returns unsatisfiable and produce an **UNSAT CORE**
- A^α cannot be extended to a concrete Cex: CORE is sufficient to prove it
- Add CORE to C’

Abstraction refinement



Variation [McMillan03]



Conclude when k is large enough

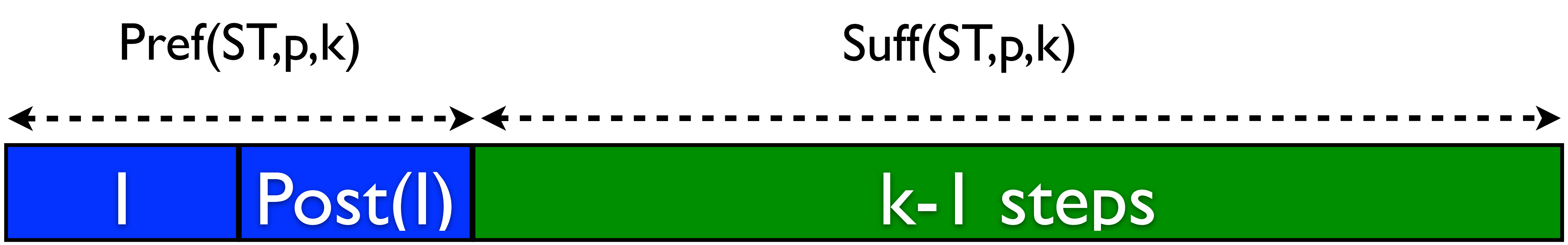
Interpolation based unbounded Sat-based model-checking [McMillan03]

Interpolant

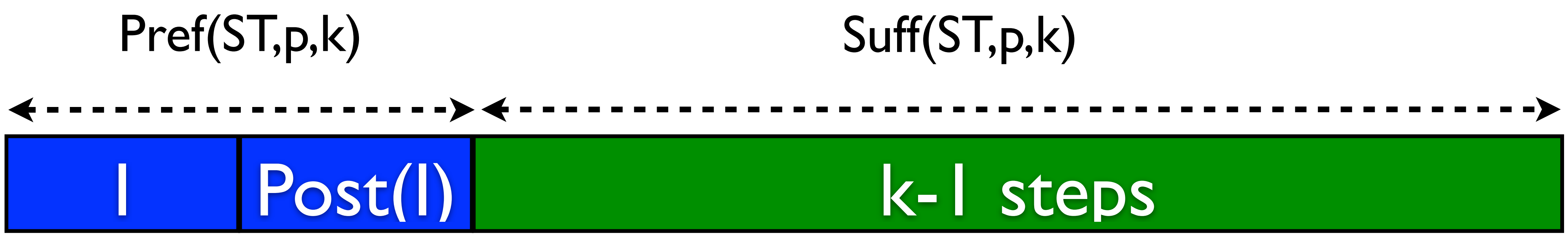
- An **interpolant** $\mathbb{I}$ for an unsatisfiable formula $A \wedge B$ is a formula such that
  - $A \Longrightarrow \mathbb{I}$ % $\mathbb{I}$ overapproximates A
  - $\mathbb{I} \wedge B$ is unsatisfiable
  - $\mathbb{I}$ **only** refers to the common variables of A and B
- Ex: $A \equiv p \wedge q$, $B \equiv \neg q \wedge r$, $\mathbb{I} \equiv q$
- Intuitively, $\mathbb{I}$ is the set of facts that the SAT solver considers relevant to prove $A \wedge B$ unsatisfiable

Interpolation and SAT-MC

- First, call **BMC**(ST,p,k)   p=invariant ?
- Decompose BMC(ST,p,k) into Pref(ST,p,k)∧Suff(ST,p,k), where
  - Pref(ST,p,k)≡init+first transition
  - Suff(ST,p,k)≡k-1 last transitions+¬p
  - if formula is SAT, we have Cex
- Otherwise, compute I for Pref(ST,p,k)∧Suff(ST,p,k)



Interpolation and SAT-MC



Fact: the interpolant $\mathbb{I}$ **overapproximates** the set of initial states and those accessible in one step and that do **not** lead to bad states within k steps (quality of the overapproximation)

Idea: iterate from a new set of initial states : $\mathbb{I}$

Interpolation procedure

procedure interpolation (M, p)
1. initialize k
2. while *true* do
3.   if $BMC(M, p, k)$ is SAT then return *counterexample*
4. $R = I$
5.   while true do
6. $M' = (S, R, T, L)$
7.   let $C = \mathit{Pref}(M', p, k) \wedge \mathit{Suff}(M', p, k)$
8.   if C is SAT then break (goto line 15)
9.   /* C is UNSAT */
10.   compute interpolant $\mathcal{I}$ of $\mathit{Pref}(M', p, k) \wedge \mathit{Suff}(M', p, k)$
11. $R' = \mathcal{I}$ is an over-approximation of states reachable from R in one step.
12.   if $R \Rightarrow R'$ then return *verified*
13. $R = R \vee R'$
14.   end while
15.   increase k
16. end while

end

Interpolation procedure

Discover negative instances

procedure interpolation (M, p)
1. initialize k
2. while *true* do
3.   if $BMC(M, p, k)$ is SAT then return *counterexample*
4. $R = I$
5.   while true do
6. $M' = (S, R, T, L)$
7.   let $C = \mathit{Pref}(M', p, k) \wedge \mathit{Suff}(M', p, k)$
8.   if C is SAT then break (goto line 15)
9.   /* C is UNSAT */
10.   compute interpolant $\mathcal{I}$ of $\mathit{Pref}(M', p, k) \wedge \mathit{Suff}(M', p, k)$
11. $R' = \mathcal{I}$ is an over-approximation of states reachable from R in one step.
12.   if $R \Rightarrow R'$ then return *verified*
13. $R = R \vee R'$
14.   end while
15.   increase k
16. end while

end

Interpolation procedure

procedure interpolation (M, p)
1. initialize k
2. while *true* do
3.   if $BMC(M, p, k)$ is SAT then return *counterexample*
4. $R = I$
5.   while true do
6. $M' = (S, R, T, L)$
7.   let $C = Pref(M', p, k) \wedge Suff(M', p, k)$
8.   if C is SAT then break (goto line 15)
9.   /* C is UNSAT */
10.   compute interpolant $\mathcal{I}$ of $Pref(M', p, k) \wedge Suff(M', p, k)$
11. $R' = \mathcal{I}$ is an over-approximation of states reachable from R in one step.
12.   if $R \Rightarrow R'$ then return *verified*
13. $R = R \vee R'$
14.   end while
15.   increase k
16. end while
end

Potentially spurious counter-example due to over-approximation

Interpolation procedure

procedure interpolation (M, p)
1. initialize k
2. while *true* do
3. if $BMC(M, p, k)$ is SAT then return *counterexample*
4. $R = I$
5. while true do
6. $M' = (S, R, T, L)$
7. let $C = \mathit{Pref}(M', p, k) \wedge \mathit{Suff}(M', p, k)$
8. if C is SAT then break (goto line 15)
9. /* C is UNSAT */
10. compute interpolant $\mathcal{I}$ of $\mathit{Pref}(M', p, k) \wedge \mathit{Suff}(M', p, k)$
11. $R' = \mathcal{I}$ is an over-approximation of states reachable from R in one step.
12. if $R \Rightarrow R'$ then return *verified*
13. $R = R \vee R'$
14. end while
15. increase k
16. end while
end

Abstract fixpoint computation through interpolants

Interpolation procedure

procedure interpolation (M, p)
1. initialize k
2. while *true* do
3. if $BMC(M, p, k)$ is SAT then return *counterexample*
4. $R = I$
5. while true do
6. $M' = (S, R, T, L)$
7. let $C = Pref(M', p, k) \wedge Suff(M', p, k)$
8. if C is SAT then break (goto line 15)
9. /* C is UNSAT */
10. compute interpolant $\mathcal{I}$ of $Pref(M', p, k) \wedge Suff(M', p, k)$
11. $R' = \mathcal{I}$ is an over-approximation of states reachable from R in one step.
12. if $R \Rightarrow R'$ then return *verified*
13. $R = R \vee R'$
14. end while
15. increase k
16. end while
end

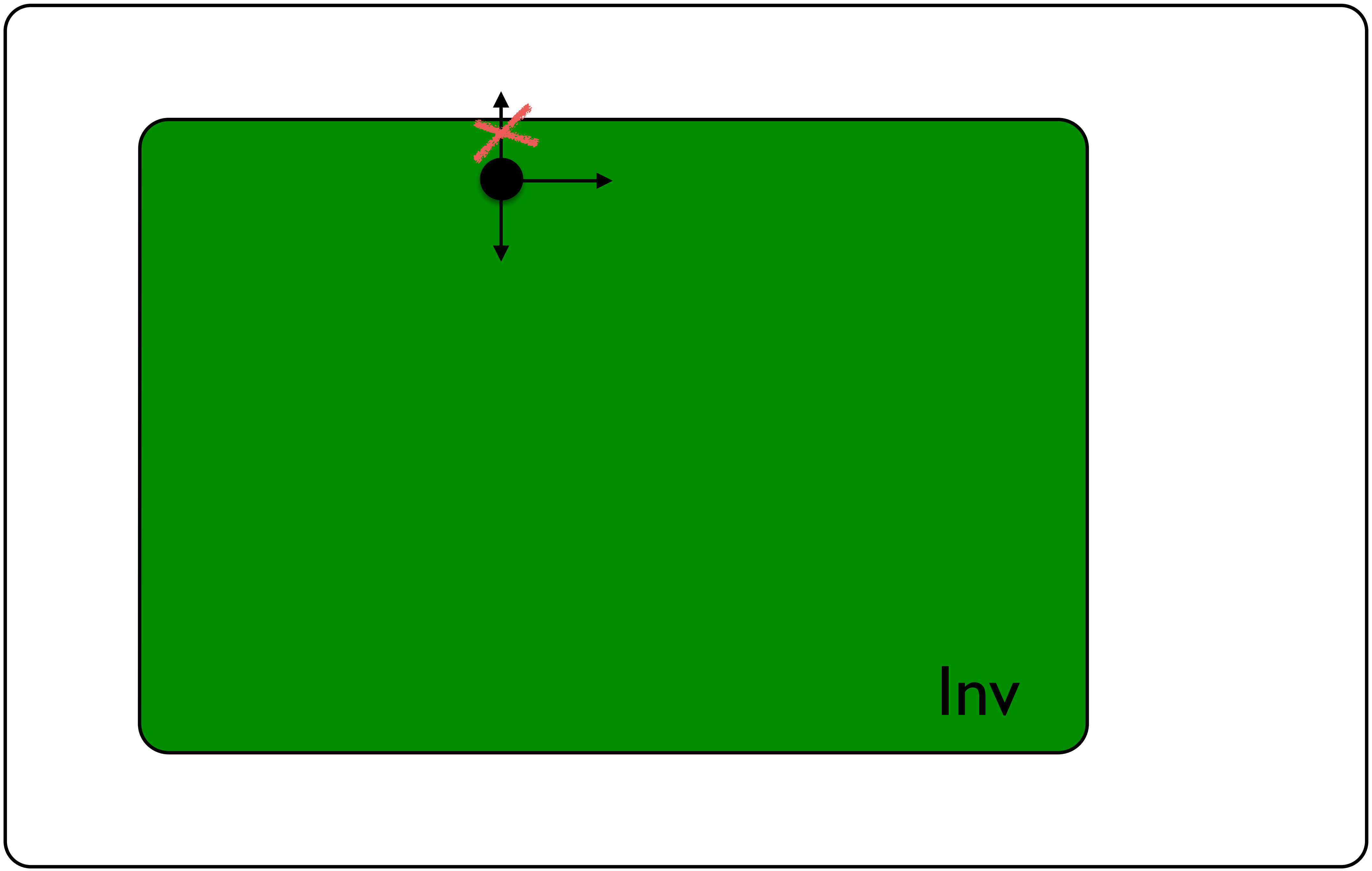
when k=diameter, the abstract algorithm concludes !

But most often it concludes **much earlier** !

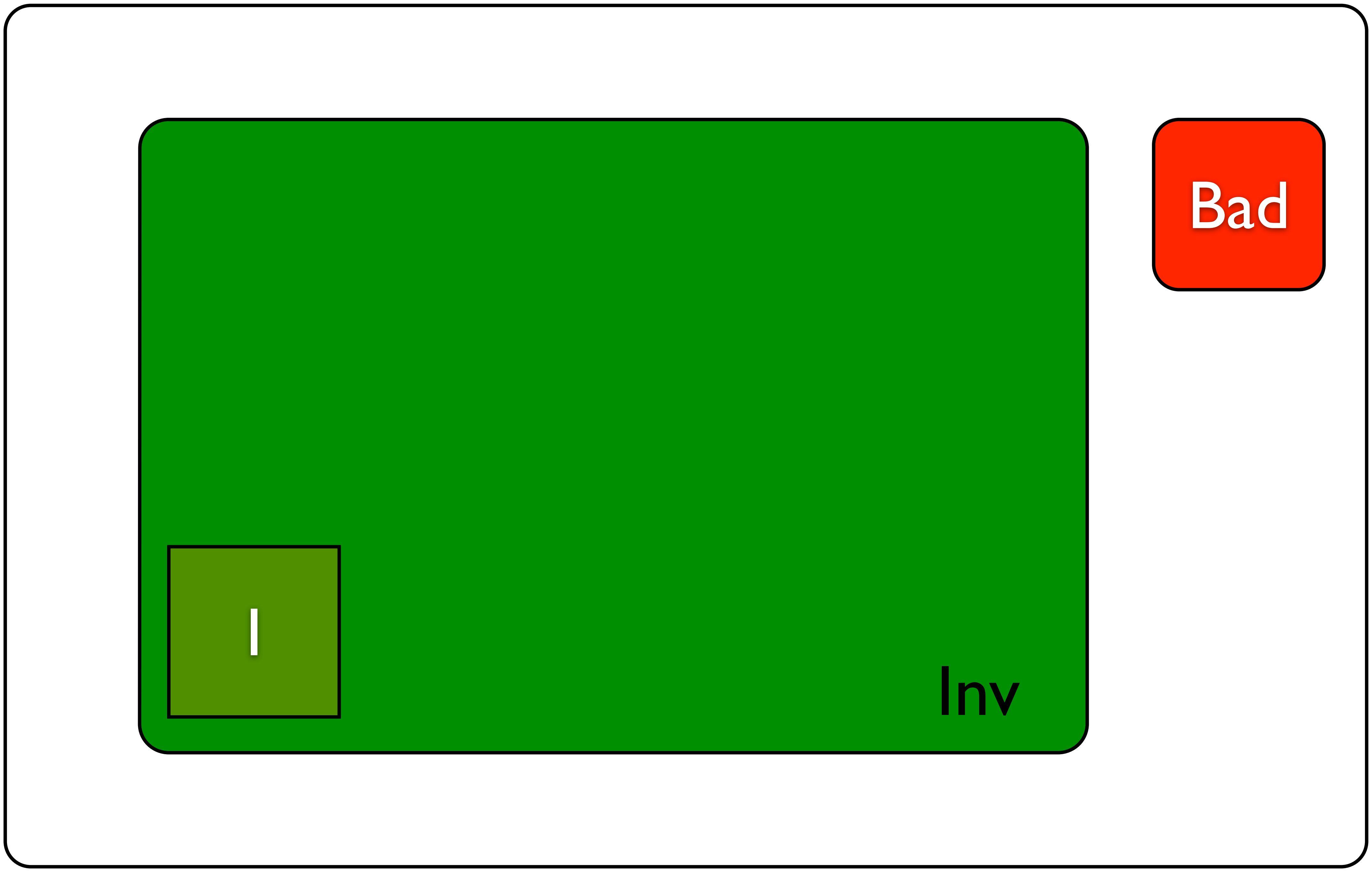
This is a complete framework !

Discovering inductive invariants in subset constructions

Inductive invariants



Inductive invariants



Verifying inductive invariants

- Let STS=(X,I,T) be a symbolic transition system
- Inv $\in \mathfrak{B}$(X) is an **inductive invariant**

iff

Inv(X) $\wedge$ T(X,X') $\Longrightarrow$ Inv(X')

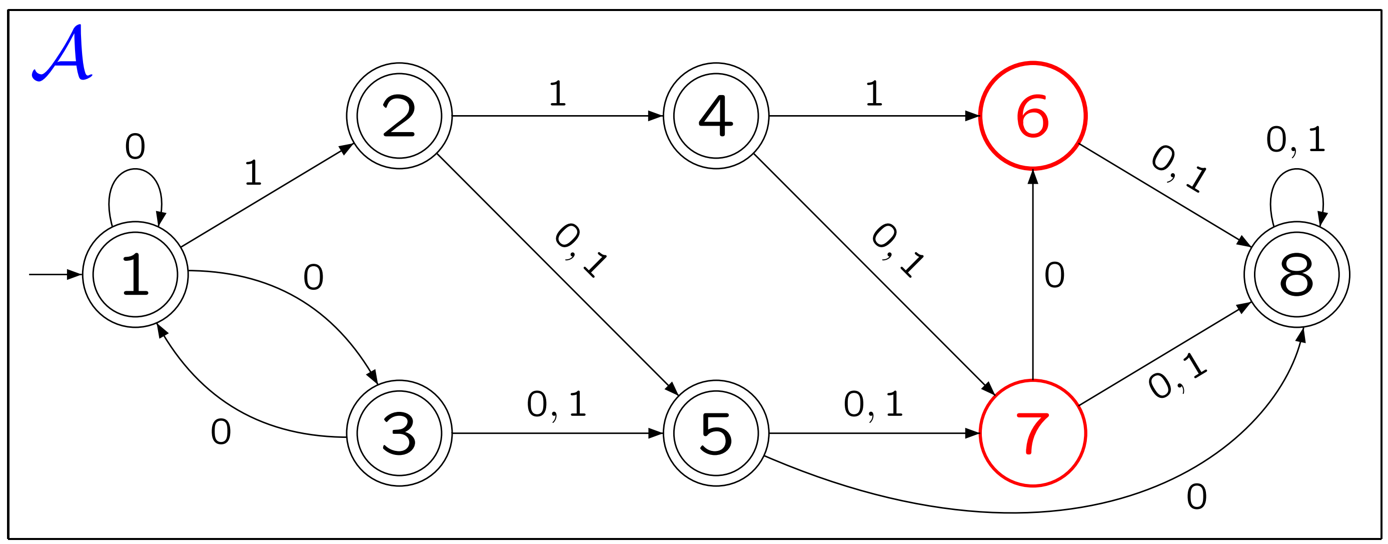
iff

$\neg$ (Inv(X) $\wedge$ T(X,X') $\Longrightarrow$ Inv(X')) is **UNSAT**

How to discover inductive invariants ?

Universality of NFA

- Nond. finite automata $A=(Q,\Sigma,q_0,\delta,F)$



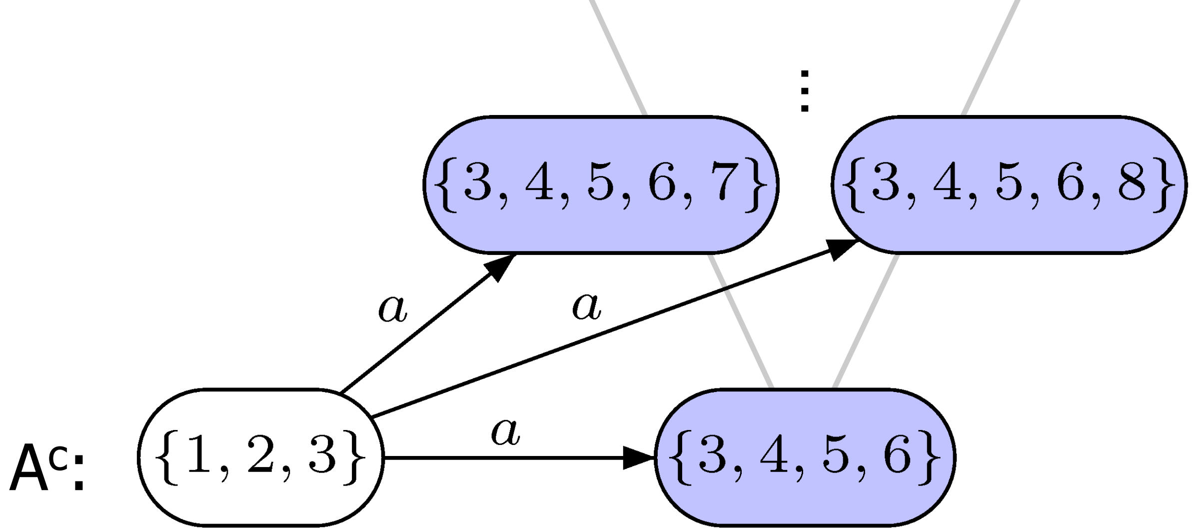
- $L(A) \neq \Sigma^*$ iff there exists a word w such that all runs on w end up in Q\F.
- Special case for $L(A) \subseteq^? L(B)$, PSpace-C.

Universality of NFA

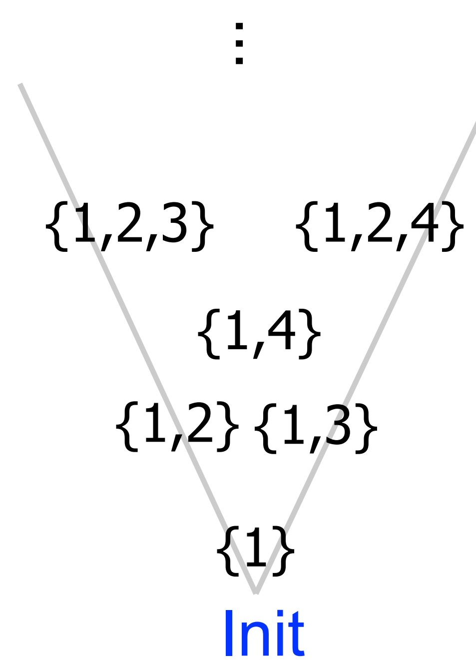
- Can be solved through reachability in STS (subset construction)
- Hard because one Boolean variable per state of the automaton - BDDs do not scale
- But special class of STS: monotonicity
- There are practical alternative algorithms to BDDs, based on antichains for example

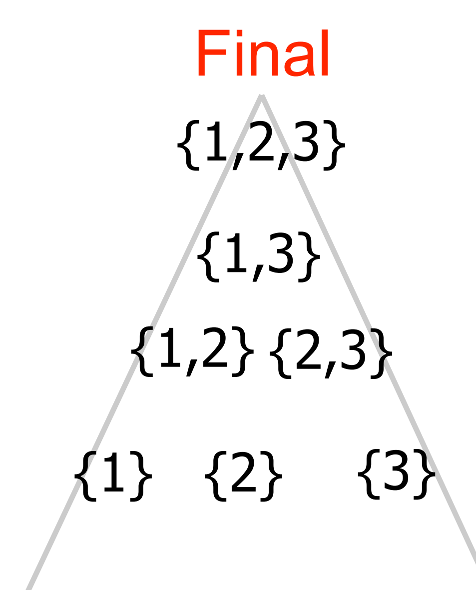
“Closed” subset construction

Transition relation can be “closed” without changing the language.



those sets
can be added safely

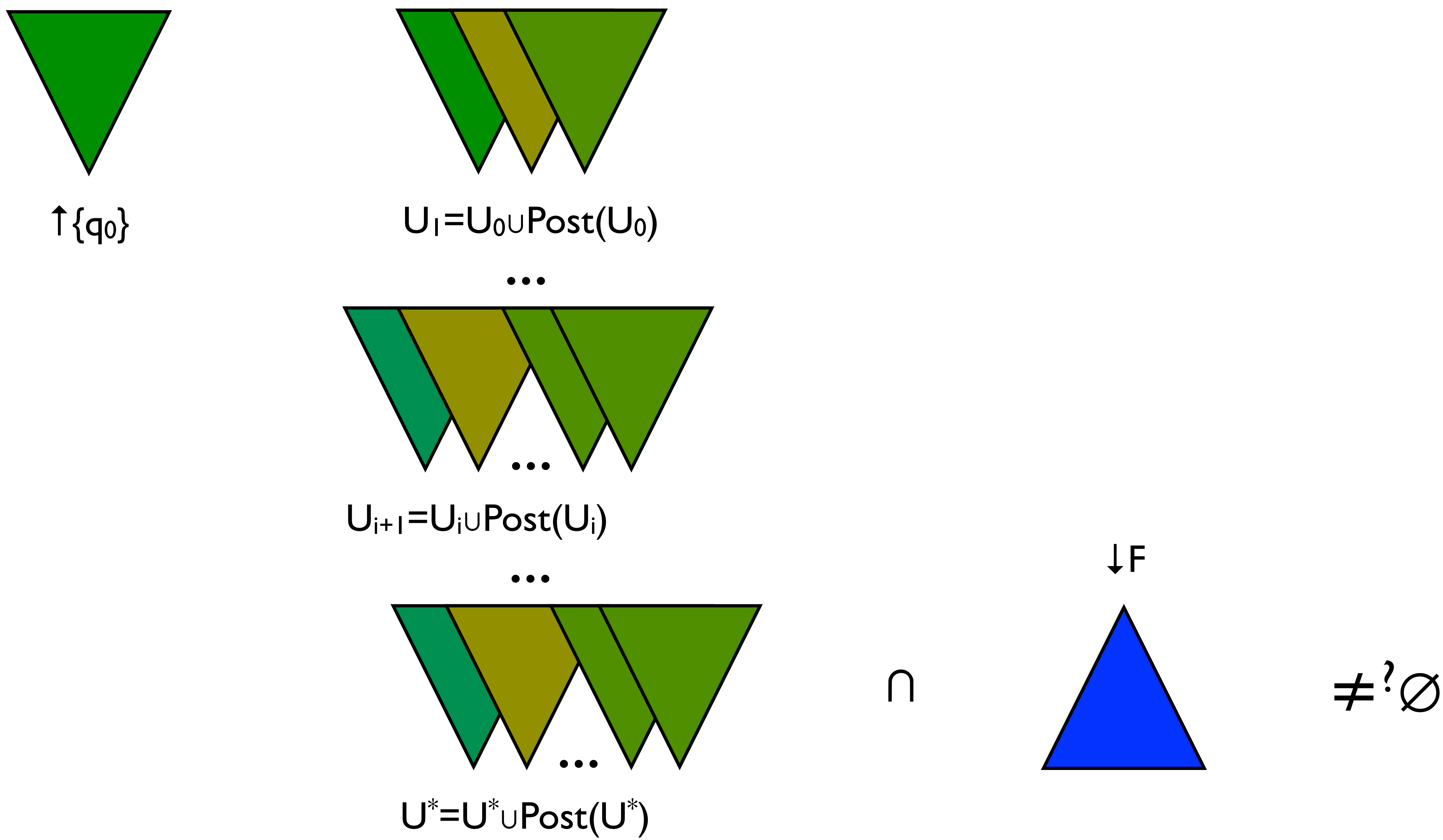




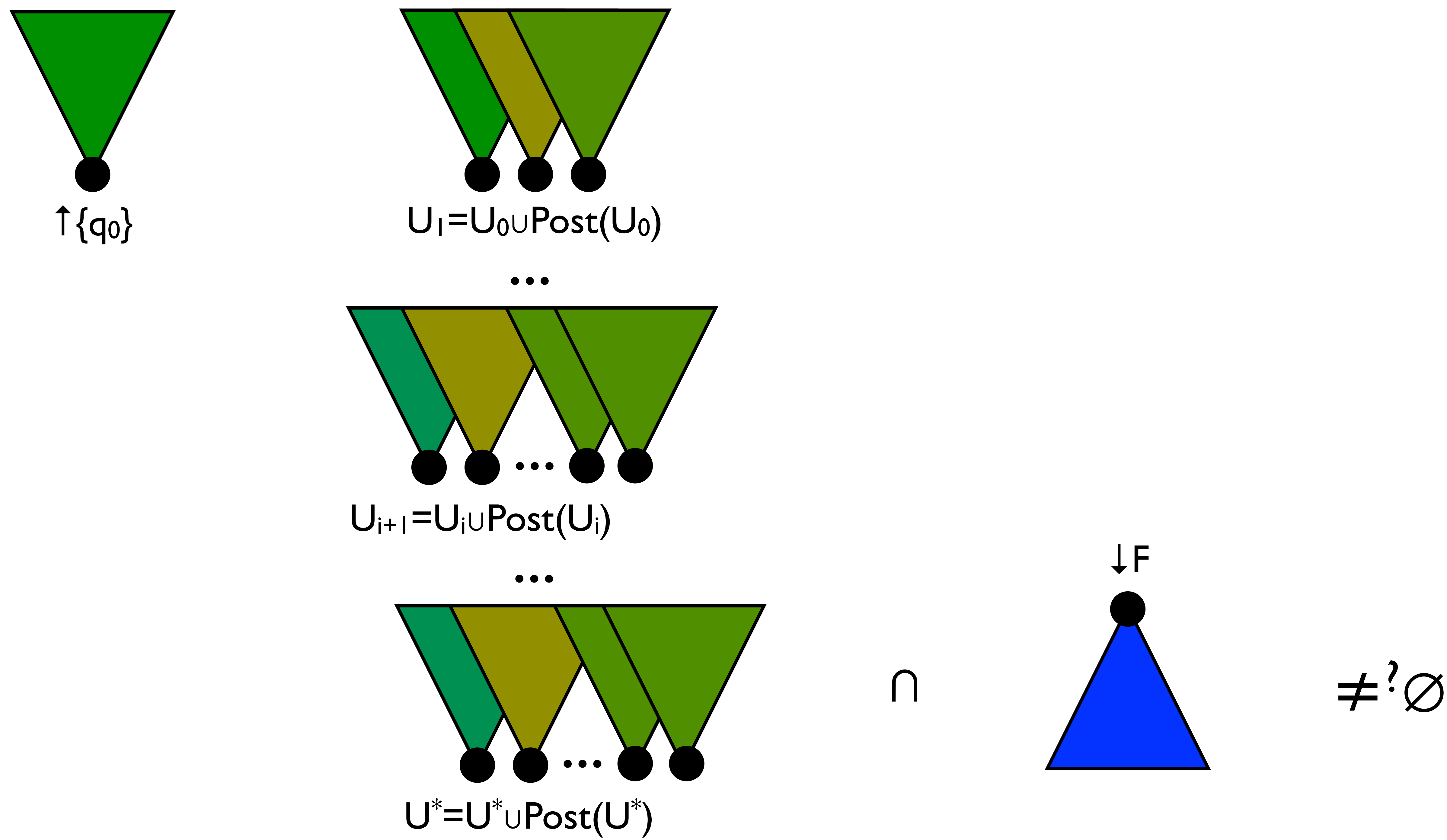
Init: sets containing initial states of A

Final: sets containing **no** accepting states of A

Forward analysis



Forward analysis



Forward analysis with antichains

$\uparrow\{q_0\}$

$U_1 = U_0 \cup Post(U_0)$

⊆-Upward-closed sets are canonically represented by their ⊆-minimal elements

Cab be very compact
Orders of magnitude faster than BDDs

$U_{i+1} = U_i \cup Post(U_i)$

…

$\downarrow F$

$\cap$

$\neq^? \emptyset$

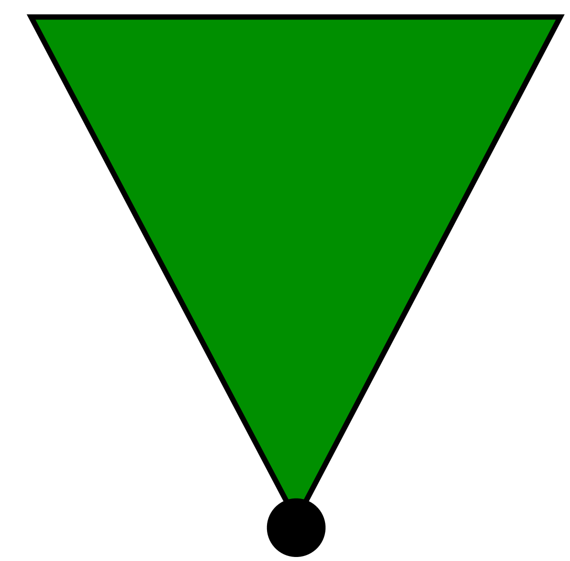
$U^* = U^* \cup Post(U^*)$

Discover post-fixpoint using SAT

- A set of sets $\mathbb{S} \subseteq 2^Q$ is a post-fixpoint of Post⟦A⟧ if:
  - $\{q_0\} \in \mathbb{S}$
  - $Post[\![A]\!](\mathbb{S}) \subseteq \mathbb{S}$
- Problem: find $\mathbb{S}$ such that $\mathbb{S} \cap F = \varnothing$
- Rely on the **antichain representation** of $\mathbb{S}$

Using SAT to synthesize $\mathbb{S}$

- Fix k the number of sets in the antichain
- $X = \{ (q,i) \mid q \in Q \wedge 1 \leq i \leq k \}$
- any $v : X \rightarrow \{0,1\}$ represent an antichain



$\{ q \mid v(q,i)=1 \}$

set nr. i of the antichain

Boolean encoding

- $\mathbb{S}$ is a post-fixpoint of Post⟦A⟧ and $\mathbb{S}$ does not intersect with ↓F

- $\bigwedge_{i=1}^{i=k} \bigwedge_{\sigma\in\Sigma} \bigvee_{j=1}^{j=k} \bigwedge_{(q,i)\in X} (q,i) \rightarrow \bigwedge_{(q,j)\mid q\in\delta(q,\sigma)} (q,j)$
- $(q_0, 1)$
- $\bigwedge_{i=1}^{i=k} \bigvee_{q\in F} \neg(q,i)$

Check that it is a post fixpoint for POST

Boolean encoding

- $\mathbb{S}$ is a post-fixpoint of Post⟦A⟧ and $\mathbb{S}$ does not intersect with ↓F

  - $\bigwedge_{i=1}^{i=k} \bigwedge_{\sigma \in \Sigma} \bigvee_{j=1}^{j=k} \bigwedge_{(q,i) \in X} (q, i) \rightarrow \bigwedge_{(q,j) | q \in \delta(q,\sigma)} (q, j)$
  - $(q_0, 1)$
  - $\bigwedge_{i=1}^{i=k} \bigvee_{q \in F} \neg(q, i)$

Check that initial state of automaton is contained

Boolean encoding

- $\mathbb{S}$ is a post-fixpoint of Post⟦A⟧ and $\mathbb{S}$ does not intersect with ↓F

  - $\bigwedge_{i=1}^{i=k} \bigwedge_{\sigma \in \Sigma} \bigvee_{j=1}^{j=k} \bigwedge_{(q,i) \in X} (q, i) \rightarrow \bigwedge_{(q,j) | q \in \delta(q,\sigma)} (q, j)$
  - $(q_0, 1)$
  - $\bigwedge_{i=1}^{i=k} \bigvee_{q \in F} \neg(q, i)$

Check universality

Boolean encoding

- $\mathbb{S}$ is a post-fixpoint of Post⟦A⟧ and $\mathbb{S}$ does not intersect with ↓F

  - $\bigwedge_{i=1}^{i=k} \bigwedge_{\sigma\in\Sigma} \bigvee_{j=1}^{j=k} \bigwedge_{(q,i)\in X} (q,i) \rightarrow \bigwedge_{(q,i)}$
  - $(q_0, 1)$
  - $\bigwedge^{i=k}$

Similar to template based inductive invariant generation using SMT solvers

Conclusion

- There are **several uses** of SAT solvers **beyond Bounded MC**
- SAT can be used **to help** SMC
- **UNSAT Core** are important and rich objects, useful for **abstraction refinements**
- **Interpolation** pushes the idea further (**no** more BDDs)
- Direct construction of **inductive invariants** can be useful too

Pointers to bibliography

- Kenneth L. McMillan: **Interpolation and SAT-Based Model Checking**. CAV 2003.
- Nina Amla, Xiaoqun Du, Andreas Kuehlmann, Robert P. Kurshan and Kenneth L. McMillan: **An Analysis of SAT-based Model Checking Techniques in an Industrial Environment**. CHARME05.
- Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu: **Symbolic Model Checking without BDDs**. TACAS 1999.
- Parosh Aziz Abdulla, Per Bjesse, Niklas Eén: **Symbolic Reachability Analysis Based on SAT-Solvers**. TACAS 2000.
- Laurent Doyen, Jean-François Raskin: **Antichain Algorithms for Finite Automata**. TACAS 2010.
- **Handbook of Model-Checking**. Springer. 2015.