Incremental Construction of Inductive Clauses for Indubitable Correctness

or simply: IC3

A Simplified Description

Based on “SAT-Based Model Checking without Unrolling”

Aaron Bradley, VMCAI 2011

“Efficient Implementation of Property Directed Reachability”

Niklas Een, Alan Mishchenko, Robert Brayton, FMCAD 2011

Safety Properties

Safety property: AG p

"p holds in every reachable state of the system"

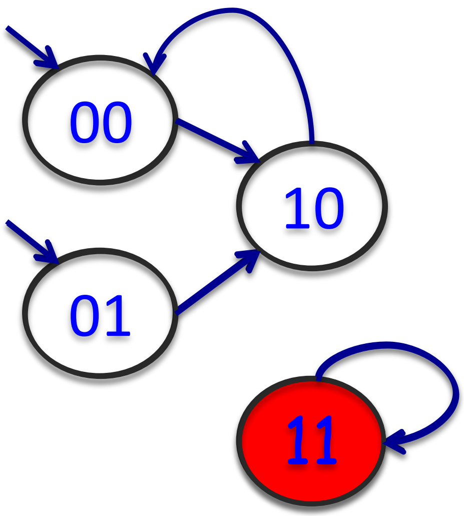
Using automata-theoretic methods, model checking of all safety properties reduces to checking AG p

Reachability: Does the transition system have a finite run ending in a state satisfying $\neg p$?

Modeling with Propositional Formulas

Finite-state system modeled as (V, INIT, T):

- V – finite set of Boolean variables
  - Four states ➔ Boolean variables: v_1 v_2
- INIT(V) – describes the set of initial states
  - $INIT = \neg v_1$
- T(V,V') – describes the set of transitions
  - $T = (v_1' \leftrightarrow \neg v_1 \vee (v_1 \wedge v_2)) \wedge (v_2' \leftrightarrow (v_1 \wedge v_2))$



state = valuation to variables

Property:

- p(V) - describes the set of states satisfying p
  - $p = \neg v_1 \vee \neg v_2$ ($Bad = \neg p = v_1 \wedge v_2$)

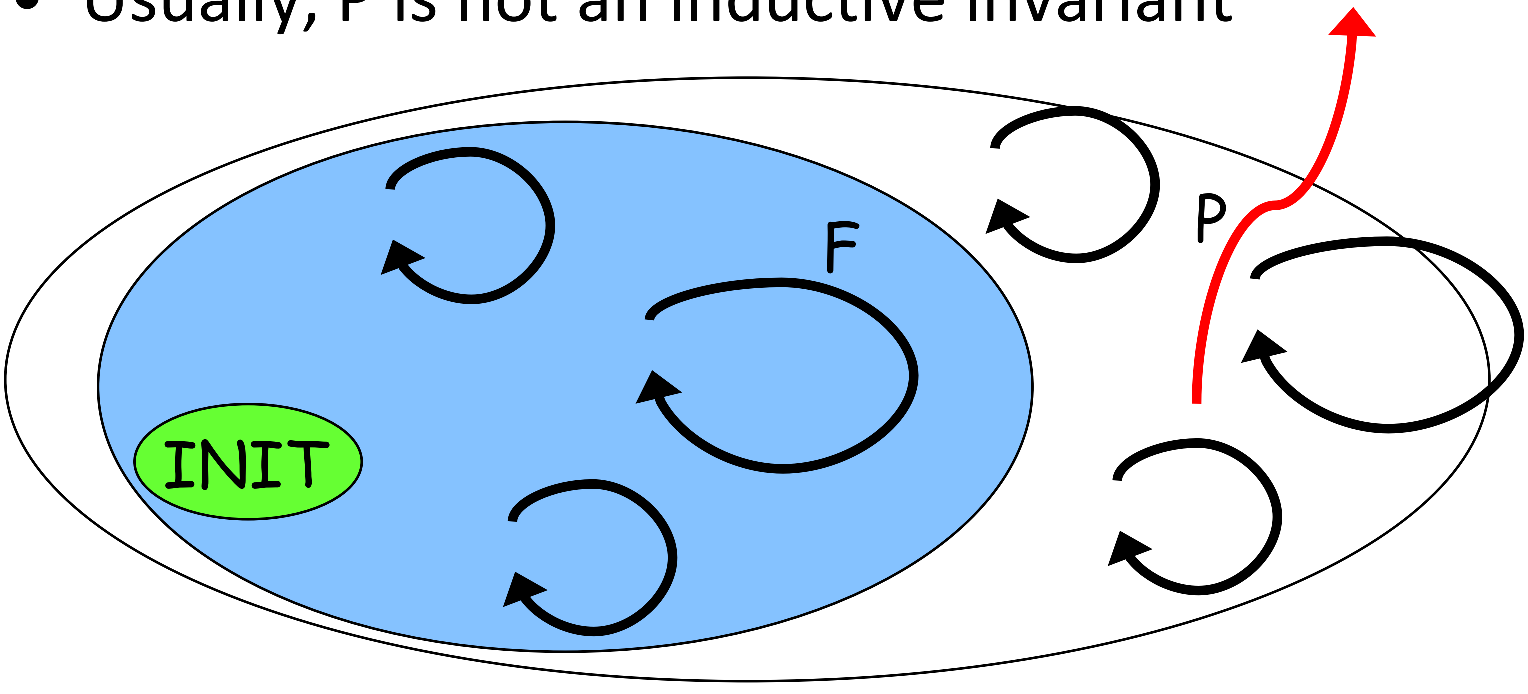
Induction for proving AG P

- The simple case: P is an inductive invariant
  - INIT(V) => P(V)
  - P(V) $\wedge$ T(V, V') => P(V')

- P(V') – the value of P in the next state

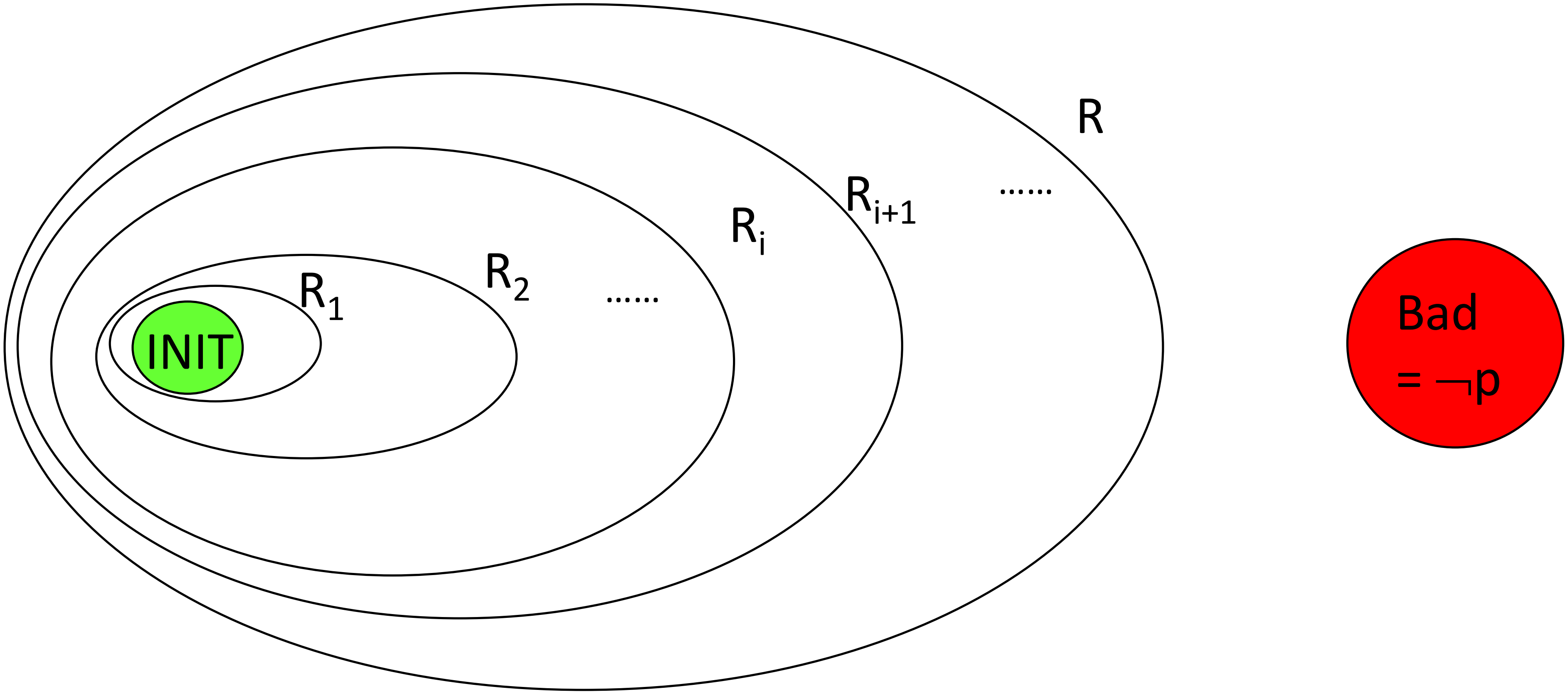
Induction for proving AG P

- Usually, P is not an inductive invariant



- BUT – a stronger inductive invariant F may exist
  - INIT => F
  - F $\wedge$ T => F'
  - F => P

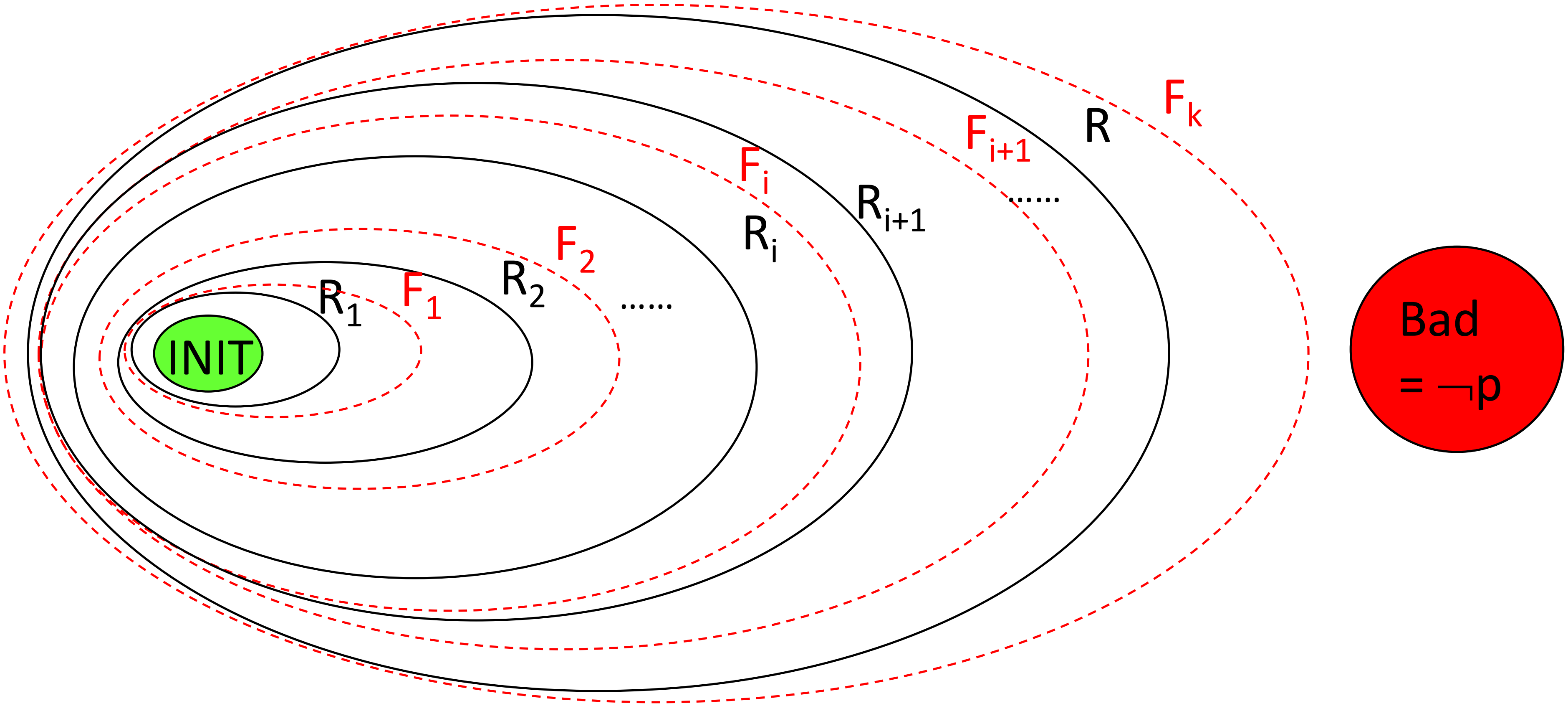
Invariant Inference by Forward Reachability



$R_{i+1}(V') \equiv R_i\ (V') \vee \exists V\ (R_i\ (V) \wedge T(V,V'))$ $\quad = R_i \vee Img(R_i,T)$

R is the strongest inductive invariant

Invariant Inference by Approximate Reachability



$F_{i+1}(V') \Leftarrow F_i(V) \wedge T(V,V')$

If $F_{k+1} \equiv F_k$ then F_k is an inductive invariant

IC3 (Bradley, VMCAI 2010)

- IC3 = **I**ncremental **C**onstruction of **I**nductive **C**lauses for **I**ndubitable **C**orrectness

- The Goal: Find an Inductive Invariant stronger than P
  - Recall: F is an inductive invariant stronger than P if
    - INIT => F
    - F $\wedge$ T => F'
    - F => P
- by learning relatively inductive facts (incrementally)
- In a property directed manner
  - Also called "Property Directed reachability" (PDR)

What Makes IC3 Special?

- All previous SAT-based approaches require unrolling of the transition relation T
  - Searching for an inductive invariant
  - Unrolling used to strengthen the invariant

- IC3 performs no unrolling
  - strengthens by learning relatively inductive facts locally

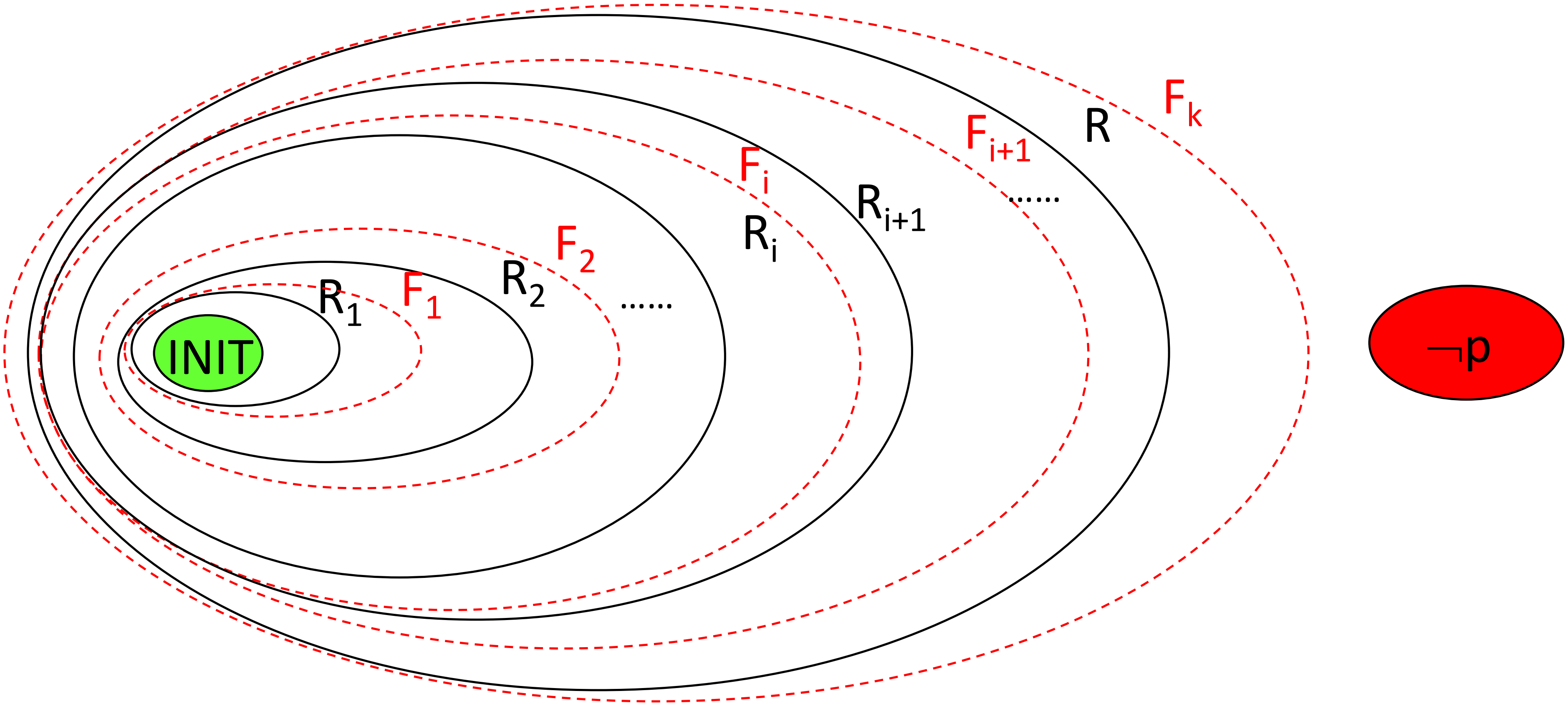
IC3 Basics

- Iteratively compute Over-Approximated Reachability Sequence (OARS) $<F_0,F_1,\ldots,F_{k+1}>$ s.t.
  - F_0 = INIT
  - $F_i \Rightarrow F_{i+1}$ — $F_i \subseteq F_{i+1}$
  - $F_i \wedge T \Rightarrow F'_{i+1}$ — Simulates one forward step
  - $F_i \Rightarrow P$ — p is an invariant up to k+1

- F_i - CNF formula given as a set of clauses
- F_i over-approximates R_i

- If $F_{i+1} \Rightarrow F_i$ then fixpoint

OARS



$F_{i+1}(V') \Leftarrow F_i(V) \wedge T(V,V')$

If $F_{k+1} \equiv F_k$ then F_k is an inductive invariant

IC3 Basics (cont.)

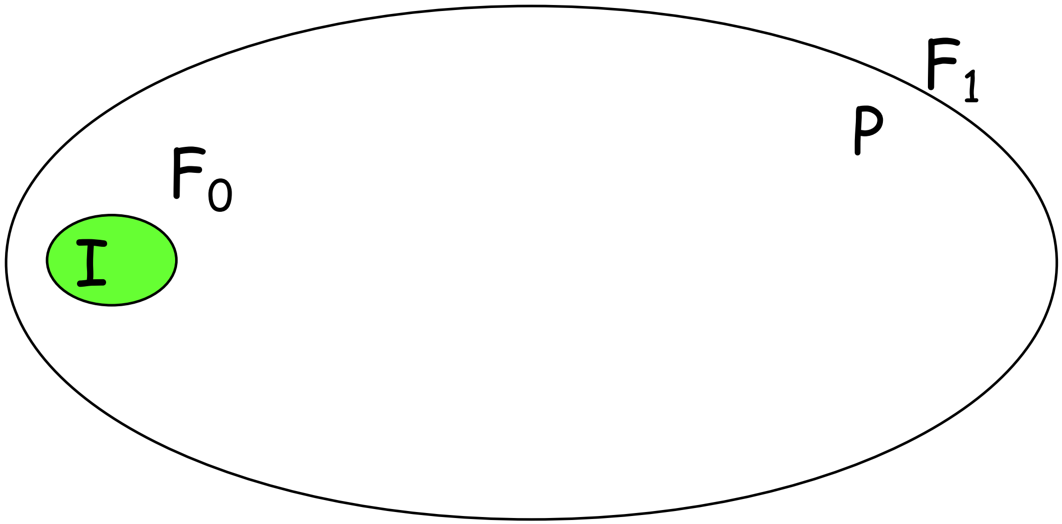
- c is inductive relative to F if
  - INIT => c
  - $F \wedge c \wedge T \Rightarrow c'$
- Notation:
  - cube s: conjunction of literals
    - $v_1 \wedge v_2 \wedge \neg v_3$ - Represents a state
  - s is a cube => **¬s is a clause** (DeMorgan)

IC3 - Initialization

OARS:

- F_0 = INIT
- F_i => F_{i+1}
- $F_i \wedge T$ => F'_{i+1}
- F_i => P

- Check satisfiability of the two formulas:
  - INIT $\wedge$ ¬P
  - INIT $\wedge$ T $\wedge$ ¬P'
- If at least one is satisfiable: cex found
- If both are unsatisfiable then:
  - INIT => P
  - INIT $\wedge$ T => P'
- Therefore
  - F_0 = INIT, F_1 = P
    - <F_0,F_1> is an OARS



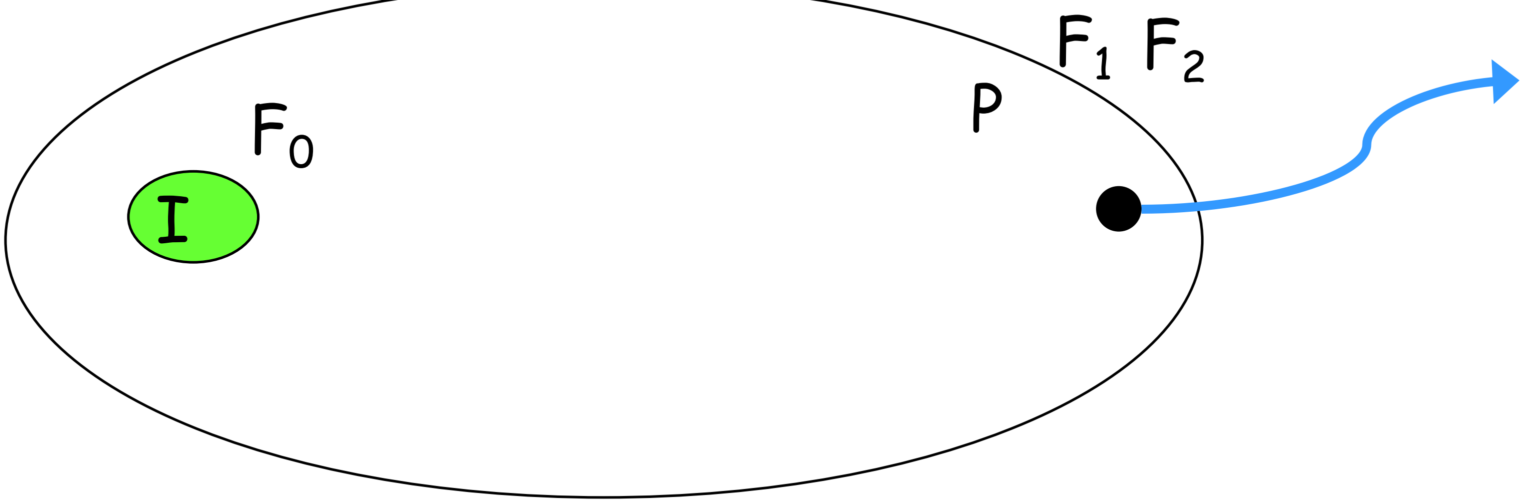
IC3 - Iteration

OARS:

- F_0 = INIT
- F_i => F_{i+1}
- $F_i \wedge T$ => F'_{i+1}
- F_i => P

- Our OARS contains F_0 and F_1
- Initialize F_2 to P
  - If P is an inductive invariant – done! ☺
  - Otherwise: $F_1 \wedge T \not=> F'_2$

    => F_1 should be strengthened

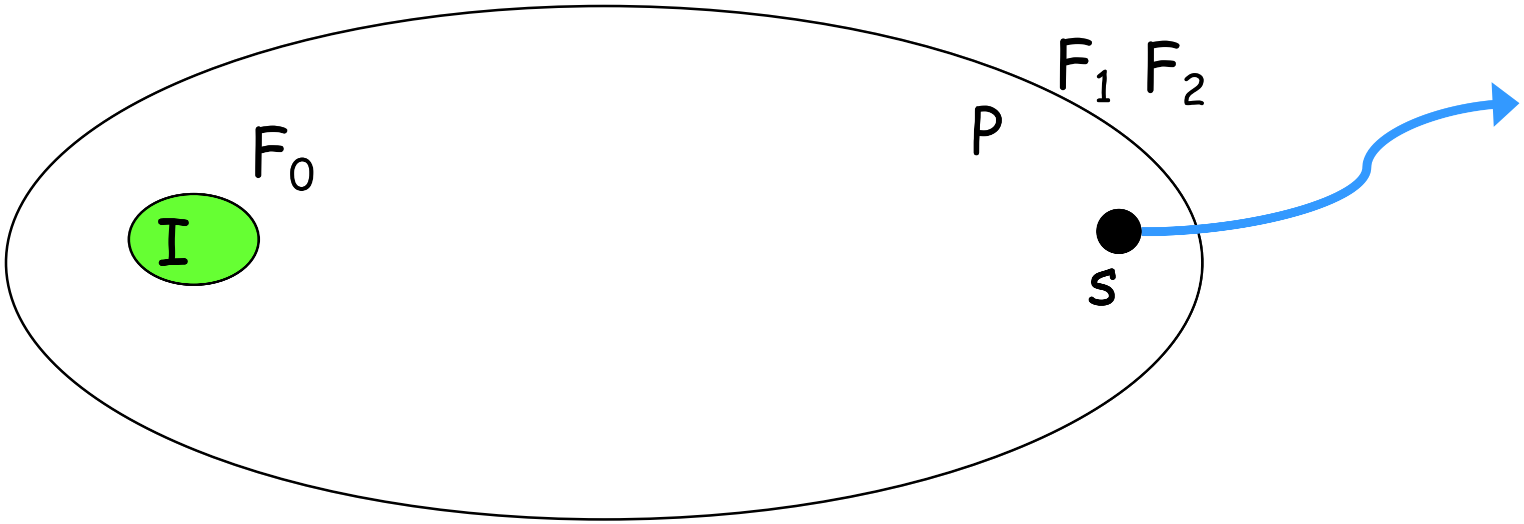


IC3 - Iteration

OARS:

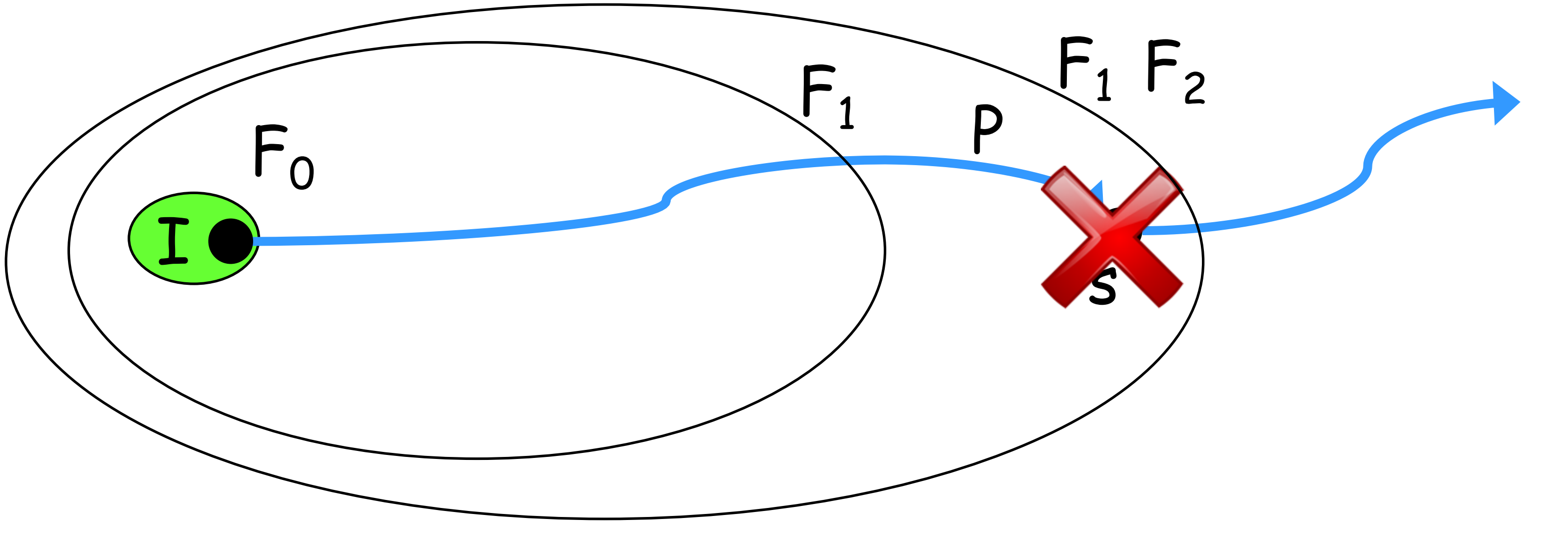
- F_0 = INIT
- $F_i \Rightarrow F_{i+1}$
- $F_i \wedge T \Rightarrow F'_{i+1}$
- $F_i \Rightarrow P$

- If P is not an inductive invariant
  - $F_1 \wedge T \wedge \neg P'$ is satisfiable
  - From the satisfying assignment get a state s that can reach the bad states



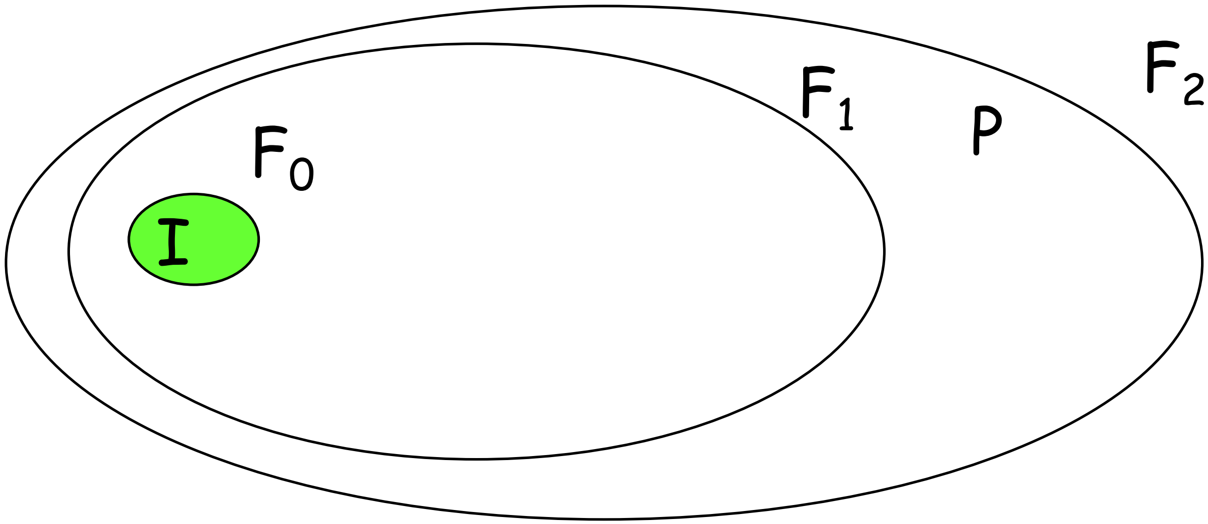
IC3 - Iteration

- Is s reachable in one transition from the previous set?
  - backward search: Check $F_0 \wedge T \wedge s'$
  - If satisfiable, s is reachable from F_0 : CEX
  - Otherwise, block s, i.e. remove it from F_1
    - $F_1 = F_1 \wedge \neg s$



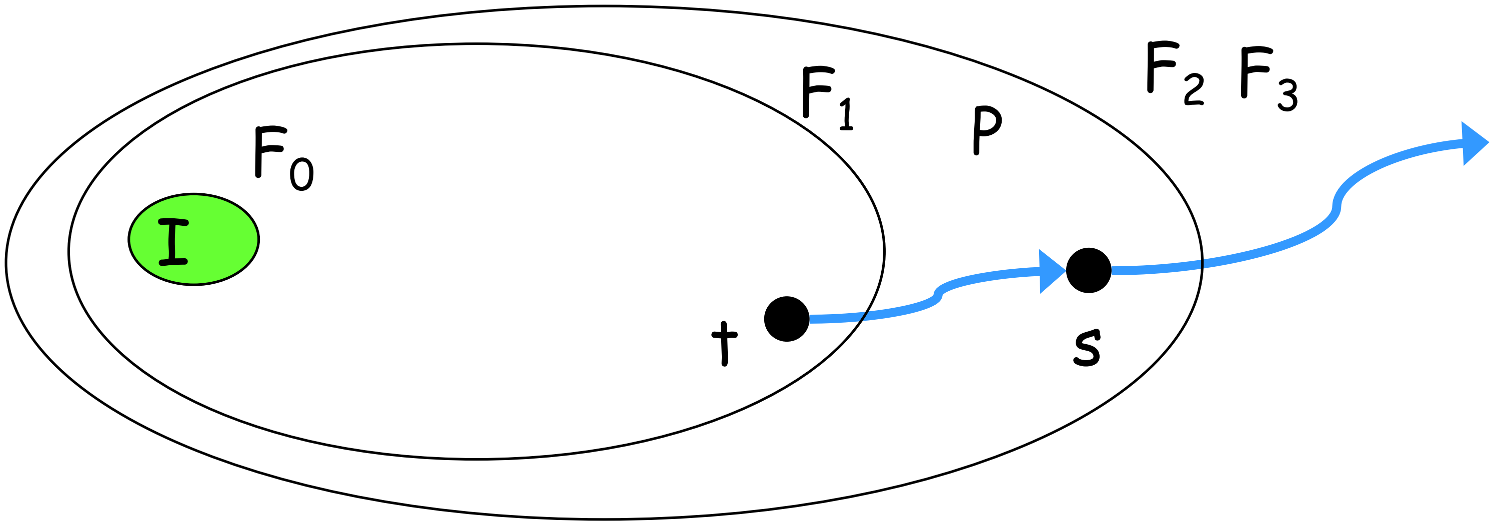
IC3 - Iteration

- Iterate this process until $F_1 \wedge T \wedge \neg P'$ becomes unsatisfiable
  - $F_1 \wedge T \Rightarrow P'$ holds
    - $(F \wedge T \wedge \neg P')$ unsat IFF $(F \wedge T \Rightarrow P')$ valid
  - $< F_0, F_1, F_2 >$ is an OARS



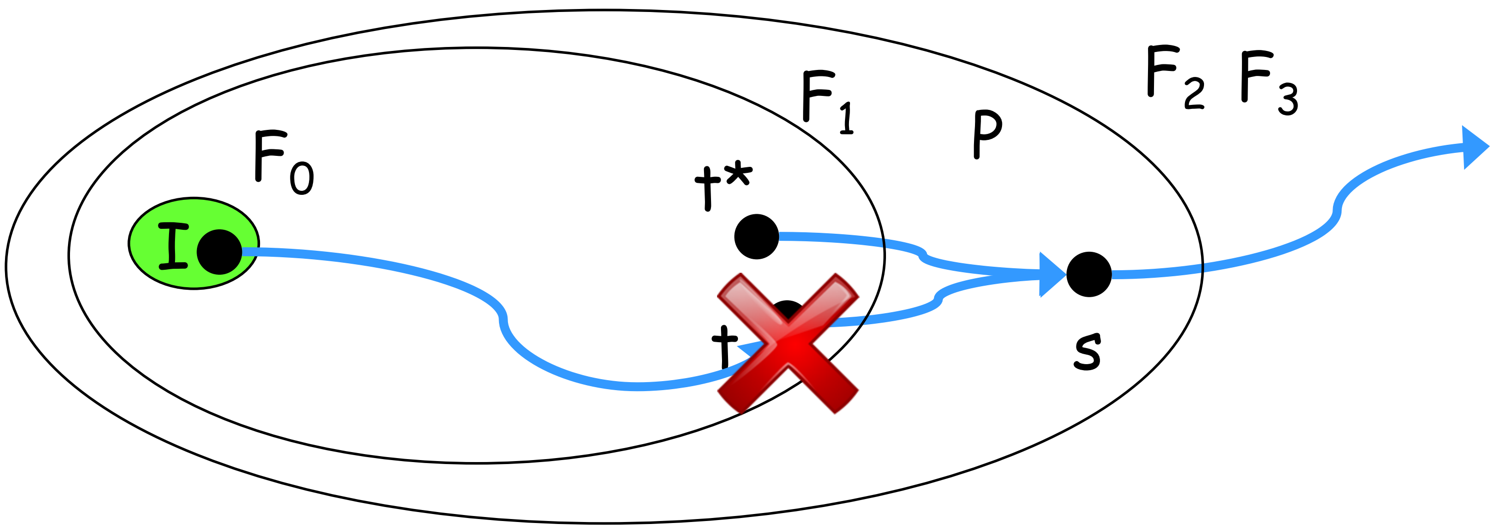
IC3 - Iteration

- New iteration, initialize F_3 to P, check $F_2 \wedge T \wedge \neg P'$
  - If satisfiable, get s that can reach ¬P
  - Now check if s can be reached from F_1 by $F_1 \wedge T \wedge s'$
  - **If it can be reached, get t and try to block it**



IC3 - Iteration

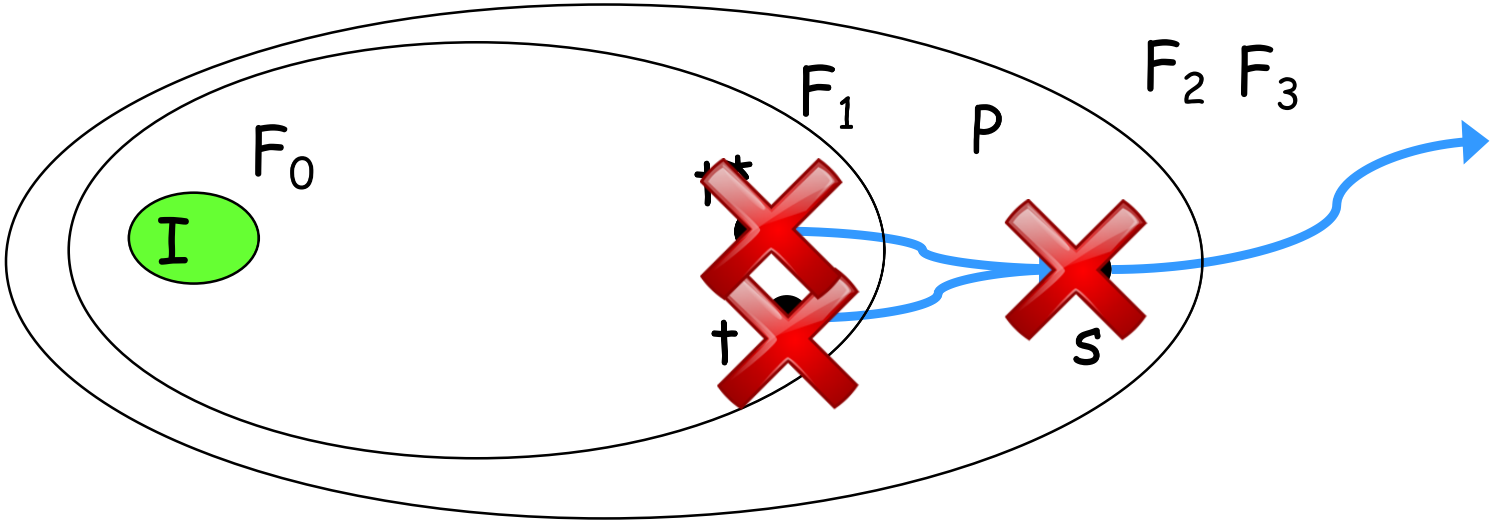
- To block t, check $F_0 \wedge T \wedge t'$
  - If satisfiable, a CEX
  - If not, t is blocked, get a "new" t* by $F_1 \wedge T \wedge s'$ and try to block t*



IC3 - Iteration

- When $F_1 \wedge T \wedge s'$ becomes unsatisfiable
  - s is blocked, get a “new” s* by $F_2 \wedge T \wedge \neg P'$ and try to block s*

……You get the picture ☺

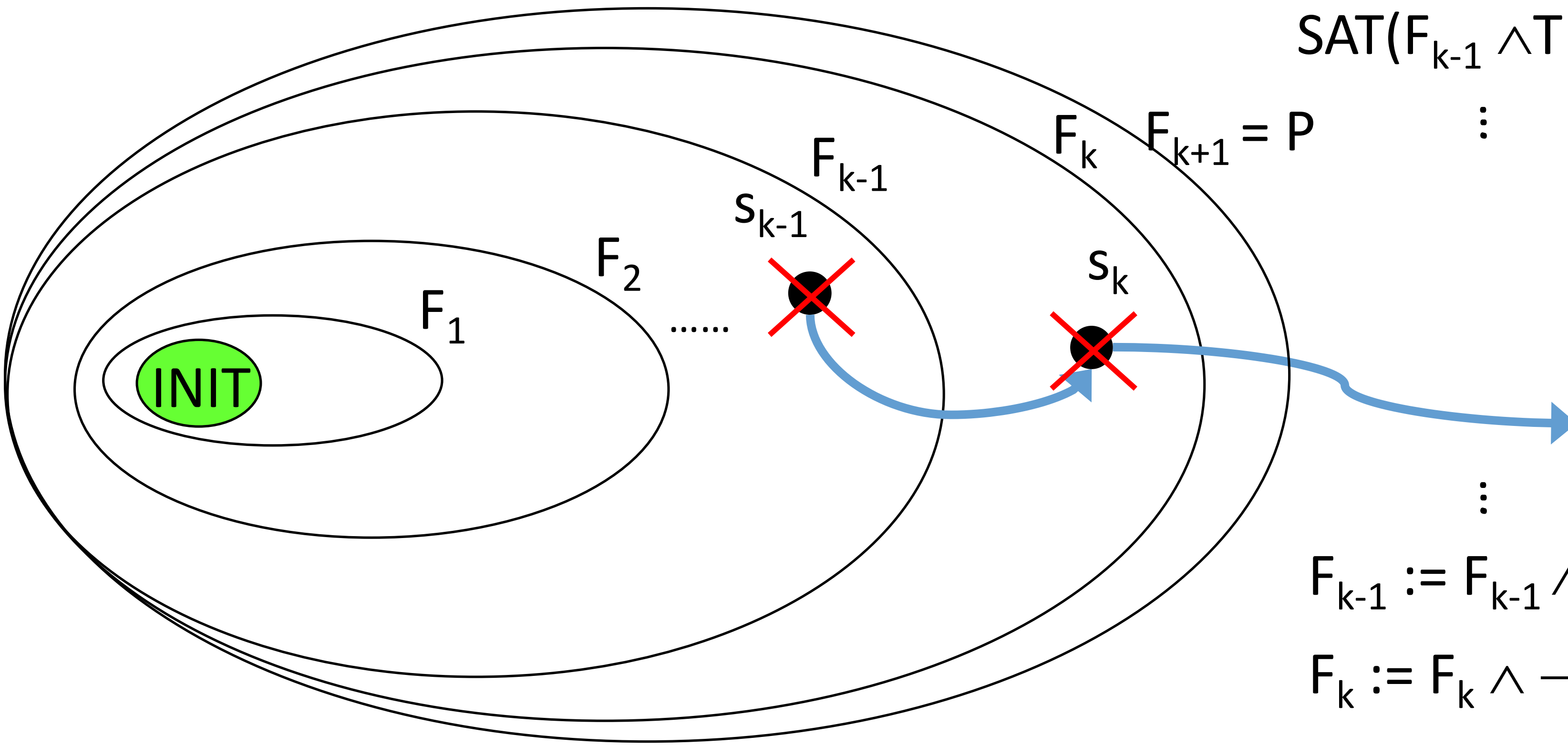


General Iteration

$SAT(F_k \wedge T \wedge \neg P')$?

$SAT(F_{k-1} \wedge T \wedge s_k')$?

⋮



⋮

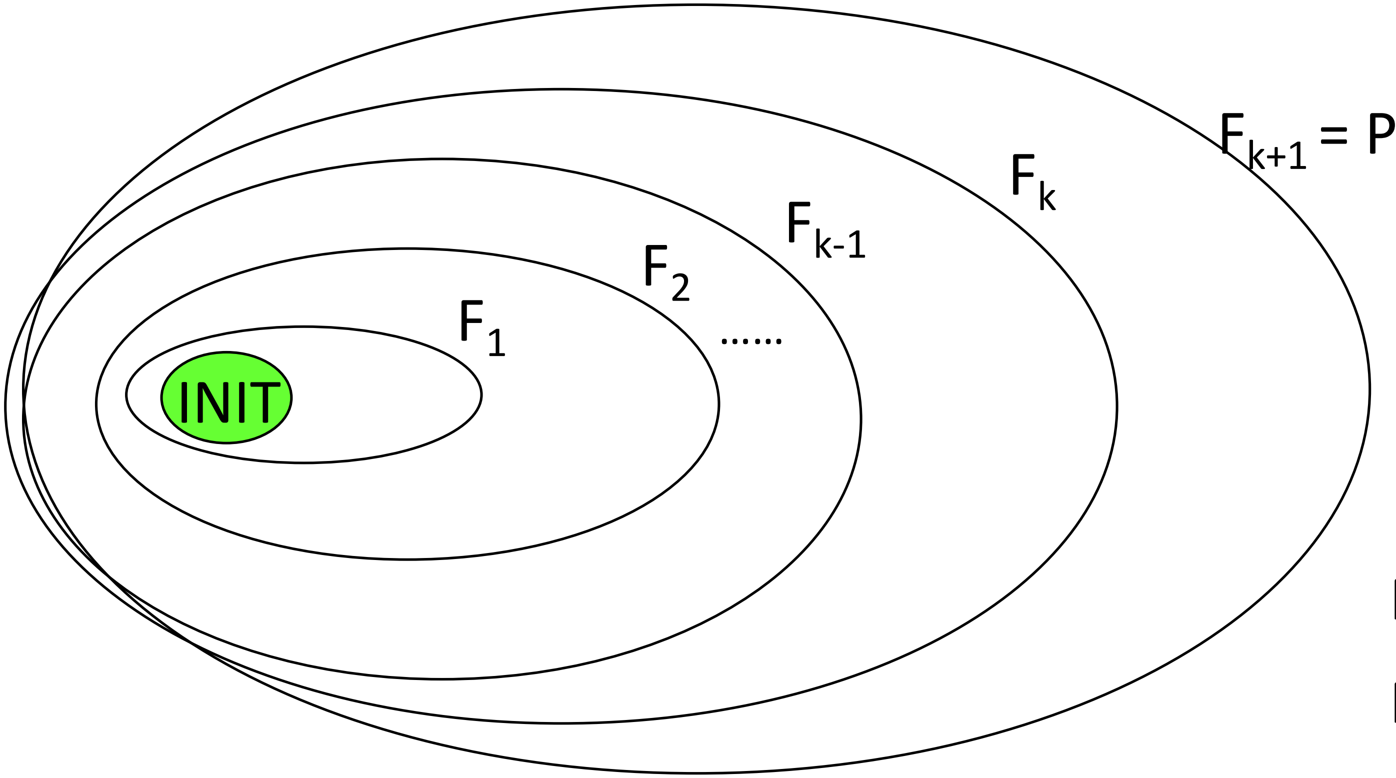
$F_{k-1} := F_{k-1} \wedge \neg s_{k-1}$

$F_k := F_k \wedge \neg s_k$

If s_k is reachable (in k steps): counterexample

If s_k is unreachable: strengthen F_k to exclude s_k

General Iteration



⋮

$F_{k-1} := F_{k-1} \wedge \neg s_{k-1}$

$F_k := F_k \wedge \neg s_k$

Until $F_k \wedge T \wedge \neg P'$ is unsatisfiable

i.e. $F_k \wedge T \Rightarrow P'$

IC3 - Iteration

- Given an OARS $<F_0,F_1,\ldots,F_k>$, define $F_{k+1} = P$
- Apply a backward search
  - Find predecessor s_k in F_k that can reach a bad state
    - $F_k \wedge T \not\Rightarrow P'$ ($F_k \wedge T \wedge \neg P'$ is sat)
  - If none exists, move to next iteration
  - If exists, try to find a predecessor s_{k-1} to s_k in F_{k-1}
    - $F_{k-1} \wedge T \not\Rightarrow \neg s_k'$ ($F_{k-1} \wedge T \wedge s_k'$ is sat)
  - If none exists, s_k can be removed from F_k
    - $F_k := F_k \wedge \neg s_k$
  - Otherwise: Recur on (s_{k-1},F_{k-1})
    - We call $(s_{k-1},k-1)$ a proof obligation
- If we reach INIT, a CEX exists

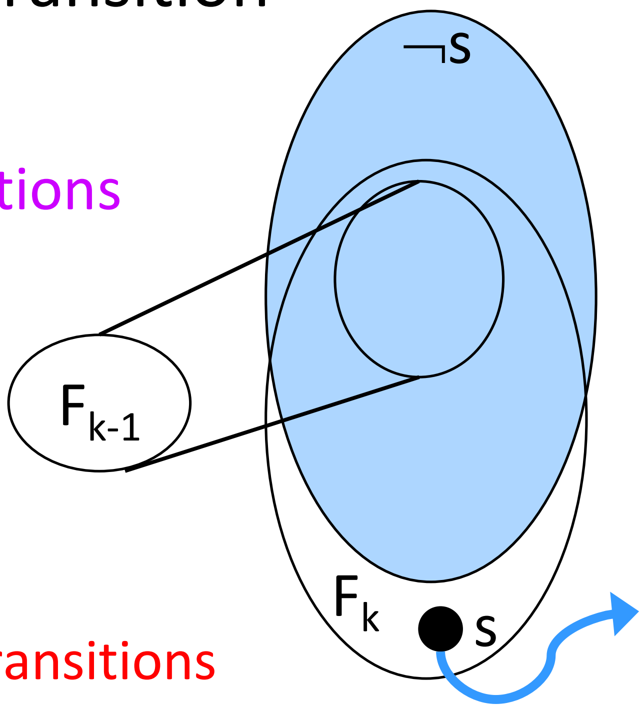
That Simple?

- Looks simple
- But this “simple” does NOT work
- Simple = State Enumeration
  - Too many states…
- Are we enumerating states?
  - No – removing more than one state at a time
  - But, yes (when IC3 doesn’t perform well)

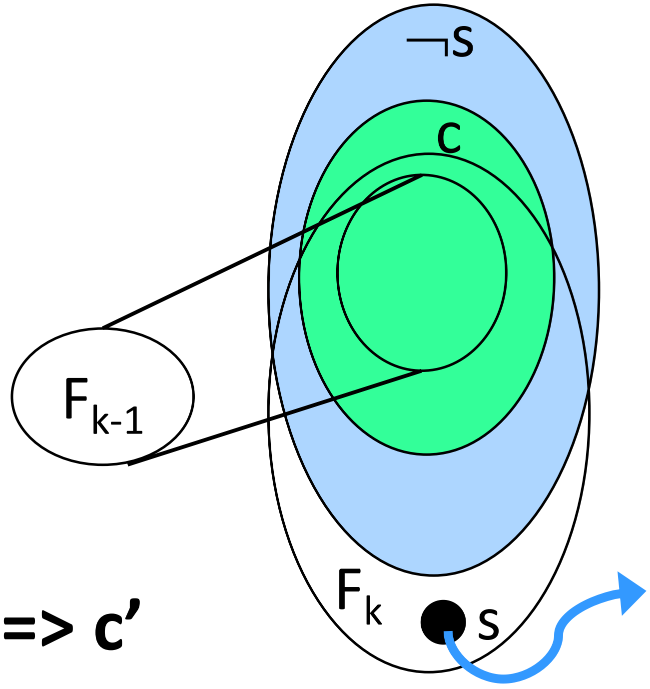
Generalization

Try to deduce a general fact from a blocked state

- s in F_k can reach a bad state in one transition
- But $F_{k-1} \wedge T \Rightarrow \neg s'$ holds
  - Therefore s is not reachable in k transitions
  - $F_k := F_k \wedge \neg s$
- We want to generalize this fact
  - s is a single state
  - Goal: learn a stronger fact
    - Find a set of states, unreachable in k transitions



Generalization



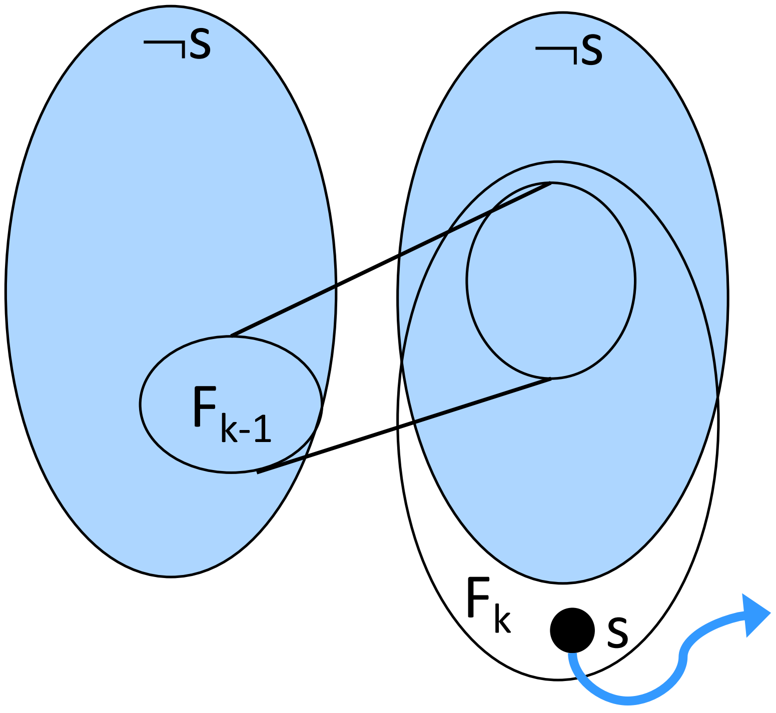
- We know $F_{k-1} \wedge T \Rightarrow \neg s'$
- And, $\neg s$ is a clause
- Generalization:
  Find a sub-clause $c \subseteq \neg s$ s.t. $\mathbf{F_{k-1} \wedge T \Rightarrow c'}$
  - Sub clause means less literals
  - Less literals implies less satisfying assignments
    - $(a \vee b)$ vs. $(a \vee b \vee c)$
  - $c \Rightarrow \neg s$ i.e. c is a stronger fact
- $F_k := F_k \wedge c$
  - More states are removed from F_k, making it stronger/more precise (closer to R_k)

Generalization

- How do we find a sub-clause $c \subseteq \neg s$ s.t. $\mathbf{F_{k-1} \wedge T \Rightarrow c'}$**?**
- Trial and Error
  - Try to remove literals from ¬s while $\mathbf{F_{k-1}} \wedge \mathbf{T} \wedge \neg c'$ remains unsatisfiable
- Use the UnSAT Core
  - $\mathbf{F_{k-1}} \wedge \mathbf{T} \wedge s'$ is unsatisfiable
  - Conflict clauses can also be used

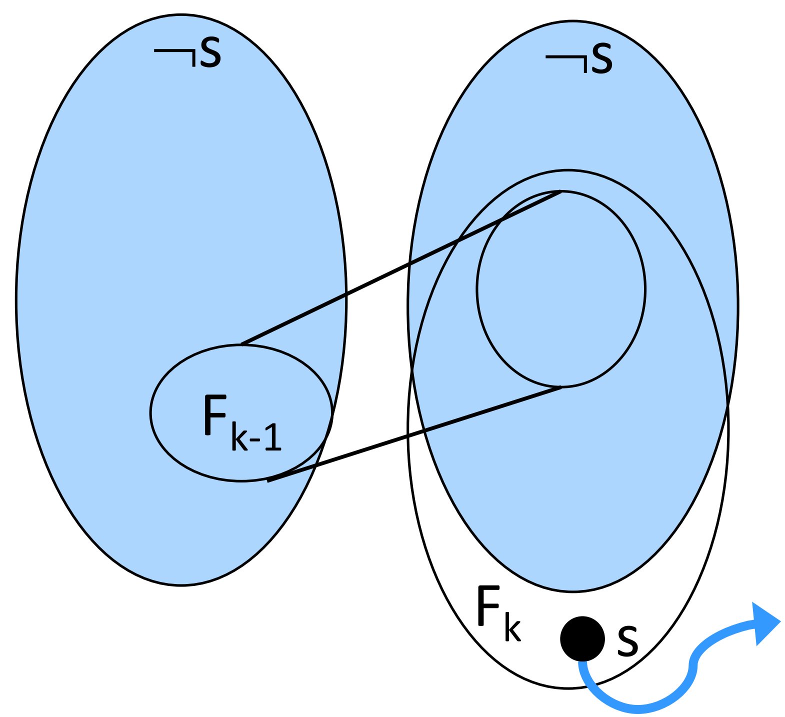
Observation 1

- Assume a state s in F_k can reach a bad state in a number of transitions
- Important Fact: **s is not in F_{k-1}** (!!)
  - $F_{k-1} \wedge T \Rightarrow F_k$
  - $F_k \Rightarrow P$
  - If s was in F_{k-1} we would have found it in an earlier iteration
- Therefore: $F_{k-1} \Rightarrow \neg s$



Observation 1

- Assume a state s in F_k can reach a bad state in a number of transitions
- Therefore: $F_{k-1} \Rightarrow \neg s$
- Assume $F_{k-1} \wedge T \Rightarrow \neg s'$ holds
  - s is not reachable in k transitions
- So, this is equivalent to
  $F_{k-1} \wedge \neg s \wedge T \Rightarrow \neg s'$
- Further INIT => ¬s
    - Otherwise, CEX!
      (INIT ≠> ¬s IFF s is in INIT)
- This looks familiar!
  - ¬s is inductive relative to F_{k-1}



Inductive Generalization

- We now know that ¬s is inductive relative to F_{k-1}
- And, ¬s is a clause
- Inductive Generalization:
  Find sub-clause $c \subseteq \neg s$ s.t.
  $\mathbf{F_{k-1} \wedge c \wedge T \Rightarrow c'}$ (and INIT => c)
  - Stronger inductive fact
- $F_k := F_k \wedge c$
  - It may be the case that $\mathbf{F_{k-1} \wedge T \Rightarrow F_k}$ no longer holds
    - Why?

Inductive Generalization

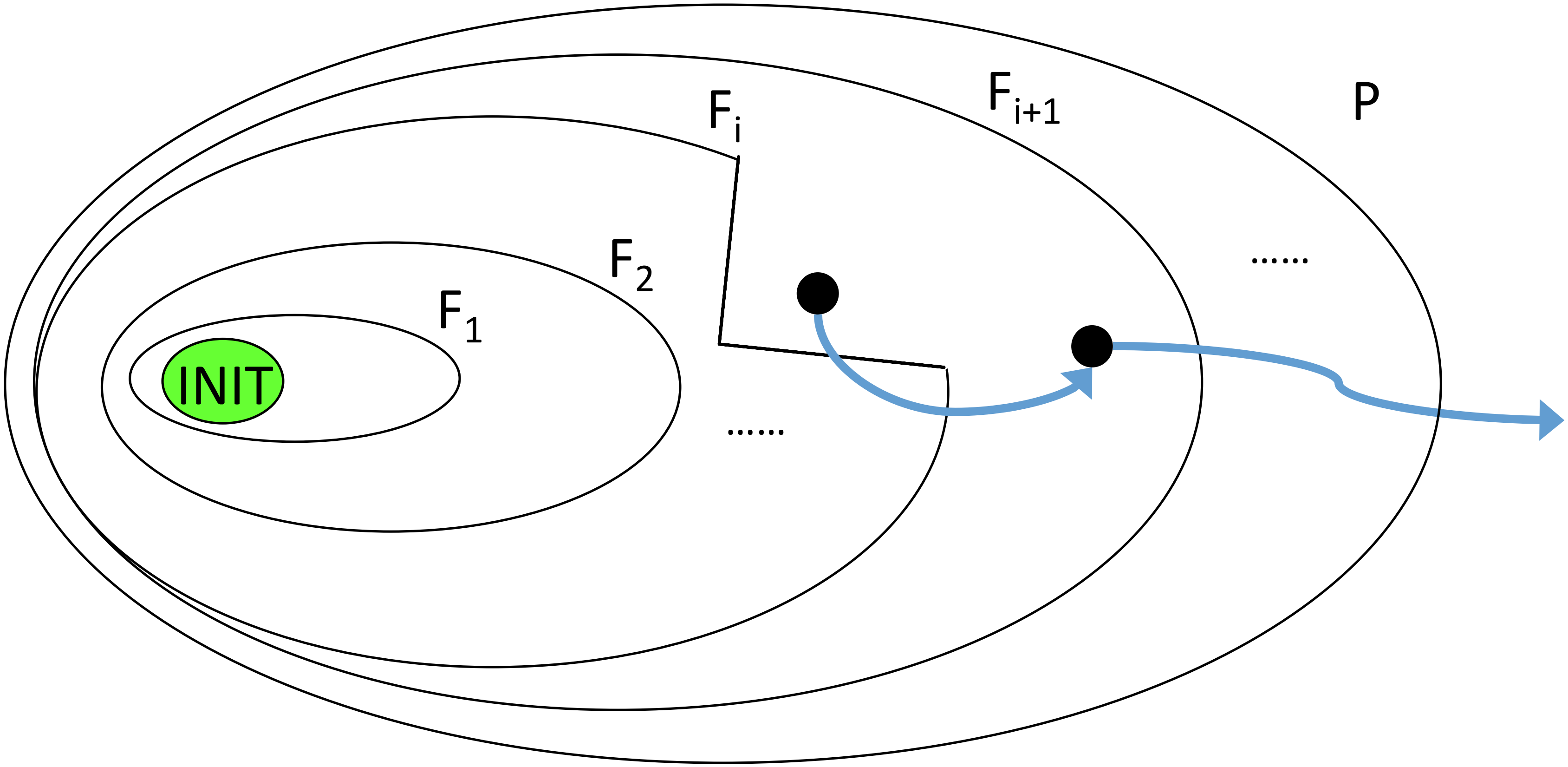
- $F_{k-1} \wedge c \wedge T \Rightarrow c'$ and $INIT \Rightarrow c$ hold
- $F_k := F_k \wedge c$

- c is also inductive relative to $F_{k-1}, F_{k-2},\ldots,F_0$
  - Add c to all of these sets
  - $F_i^* = F_i \wedge c$
- $\mathbf{F_i^* \wedge T \Rightarrow F_{i+1}}^*$ holds

Observation 2

- Assume state s in F_i can reach a bad state in a number of transitions
- s is also in F_j for j > i   (F_i => F_j)
  - a longer CEX may exist
  - s may not be reachable in i steps, but it may be reachable in j steps
- If s is blocked in F_i, it must be blocked in F_j for j > i
  - Otherwise, a CEX exists

Push Forward



Push Forward

- Suppose s is removed from F_i
  - by conjoining a sub-clause c
  - $F_i = F_i \wedge c$
- c is a clause learnt at level i
- try to push c forward for j > i
  - If $F_j \wedge c \wedge T \Rightarrow c'$ holds
    - c is inductive in level j
    - $F_{j+1} = F_{j+1} \wedge c$
  - Else: s was not blocked at level j > i
    - Add a proof obligation (s,j)
    - If s is reachable from INIT in j steps, CEX!

IC3 – Key Ingredients

- Backward Search
  - Find a state s that can reach a bad state in a number of steps
  - s may not be reachable (over-approximations)
- Block a State
  - Do it efficiently, block more than s
    - Generalization
- Push Forward
  - An inductive fact at frame i, may also be inductive at higher frames
  - If not, a longer CEX is found

IC3 – High Level Alg

If INIT $\wedge$ ¬P is SAT return false; // CEX

If INIT $\wedge$ T $\wedge$ ¬P’ is SAT return false; // CEX

OARS = <INIT,P>; // <F_0,F_1>

k=1

while (OARS.is_fixpoint() == false) do

    extend(OARS); // F_{k+1} = P

    while (F_k $\wedge$ T $\wedge$ ¬P’ is SAT) do

        s = get_state();

        If (block_state(s, k) == false) // recursive function

            return false; // CEX

    push_forward();

    k = k+1

return valid;

F_i represented by set of clauses.
Check implication by set inclusion

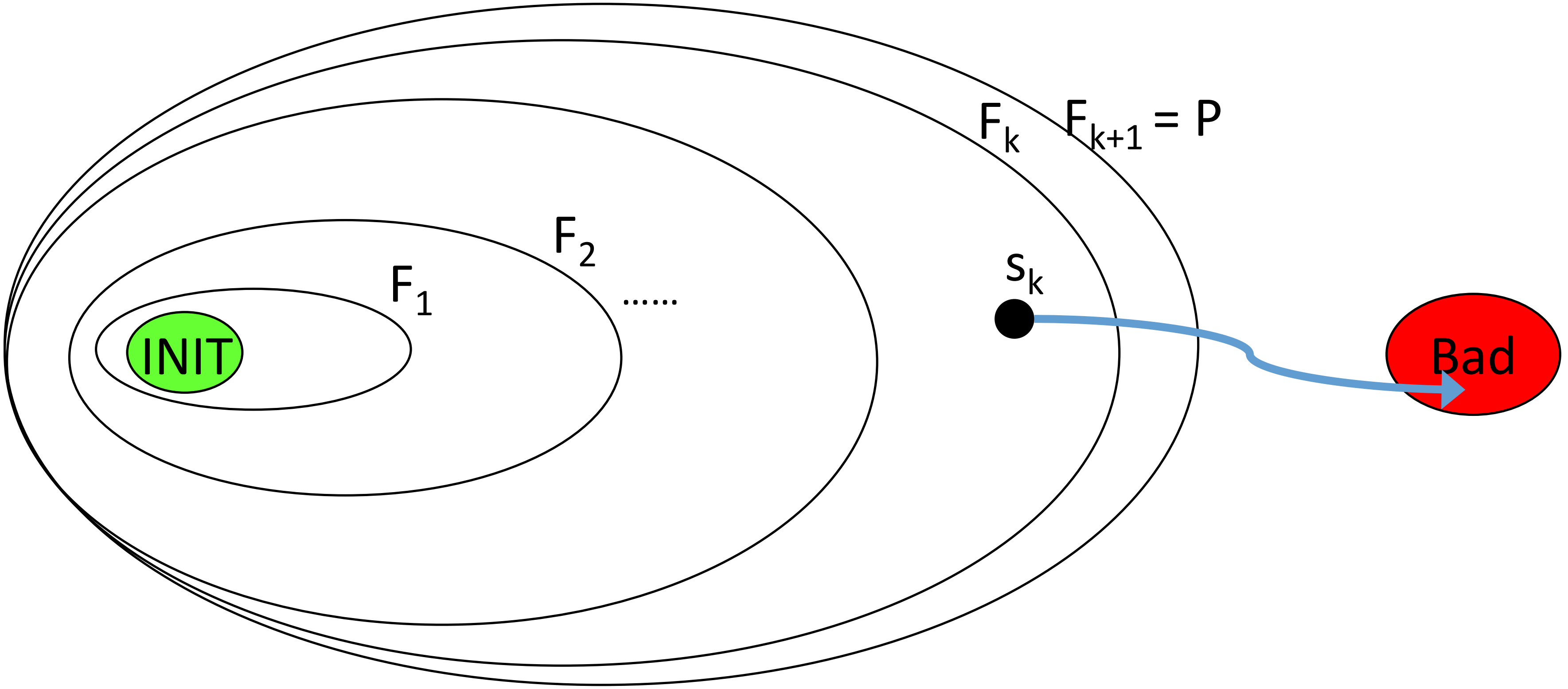
IC3 – Alternative Description

F_i represented by set of clauses.
Check implication by set inclusion

```
If INIT ∧ ¬P is SAT return false; // CEX
If INIT ∧ T ∧ ¬P' is SAT return false; // CEX
OARS = <INIT,P>;  // <F0,F1>
k=1
while (OARS.is_fixpoint() == false) do
      extend(OARS);    // ~~Fk+1 = P~~        Fk+1 = true
      ~~while (Fk ∧ T ∧ ¬P' is SAT) do~~     while (Fk+1 ∧ ¬P  is SAT) do
            s = get_state();
            If (block_state(s, k) == false) // recursive function
                        return false;  // CEX
      push_forward();
      k = k+1
return valid;
```

General Iteration

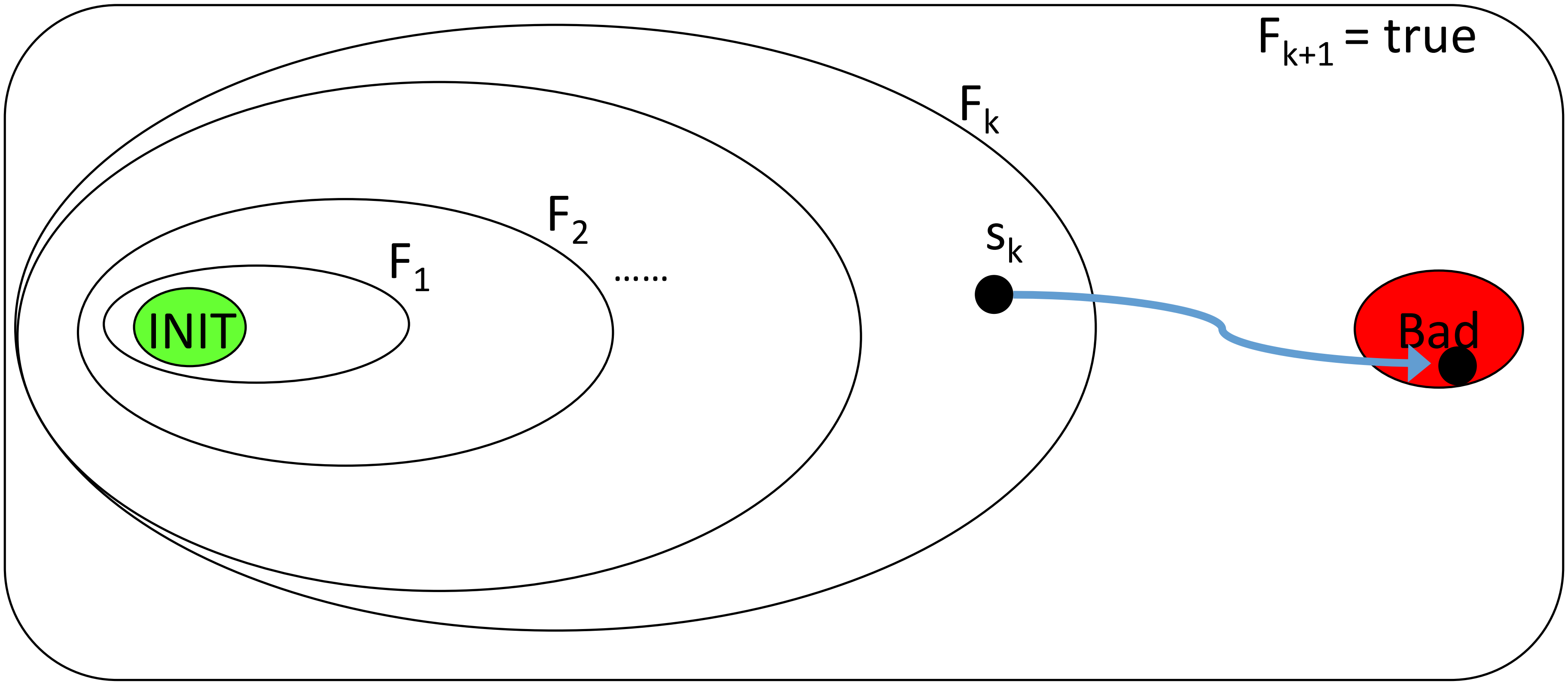
$SAT(F_k \wedge T \wedge \neg P')$?



General Iteration: Alternative

SAT($F_{k+1} \wedge \neg P'$) ?

SAT($F_k \wedge T \wedge s'$) ?



Correctness

PDR vs. CEGAR

CEGAR:

- computes strongest inductive invariant (least fixpoint) with respect to given abstraction
  - Invariant computation is **not** property guided
  - But the abstraction and refinement are property guided
- Requires abstract transformer
- Requires refinement mechanism that reveals new predicates (e.g., interpolation)
- Counterexample analysis uses unrolling of TR

What About Infinite State Systems?

- Use first-order logic instead of propositional logic

Filter Example

{ h is a list }

```
void filter(Node h){
  Node i:=h; j:=null;
  while (i ≠ null){
     if ¬C(i) then {
       if i = h then h:=i.n
       else j.n:=i.n;
     }
     else j:=i;
     i:=i.n;
}}
```

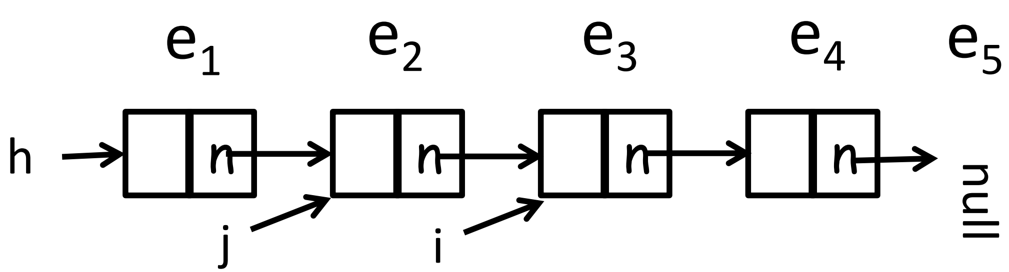
{ post-condition: all C-elements were removed, other remained while preserving original order }

From Programs to Logic

- Vocabulary:

$$V = \langle \underbrace{h, i, j, null,}_{\text{constants}}\ \underbrace{n(\cdot,\cdot), C(\cdot)}_{\text{relations}} \rangle$$

Program state:



first-order structure

$D = \{e_1, \ldots, e_5\}$

$I(h) = e_1$

$I(i) = e_3$

$I(j) = e_2$

$I(null) = e_5$

$I(n) = \{(e_1,e_2), (e_2,e_3) \ldots \}$

Filter Example: Assertions

$\{H = h \land \forall x,y.\ n^*(x,y) \leftrightarrow L(x,y)\ \}$

```
void filter(Node h){
  Node i:=h; j:=null;
  while (i ≠ null){
    if ¬C(i) then {
      if i = h then h:=i.n
      else j.n:=i.n;
    }
    else j:=i;
    i:=i.n;
}}
```

$V = \langle h, i, j, null, n(\cdot,\cdot), C(\cdot), H, L(\cdot,\cdot) \rangle$

Auxiliary symbols: $H, L(\cdot,\cdot)$

$\{\ \forall z.\ h \neq null \land n^*(h,z) \rightarrow C(z)$

$\forall z.\ L(H,z) \land C(z) \rightarrow n^*(h,z)$

$\forall x,y.\ L(H,x) \land L(x,y) \land C(x) \land C(y) \rightarrow n^*(x,y)\ \}$

From Programs to Transition Systems

- Transition relation:
  first-order formula TR(V,V’) describing loop body

- Initial and Bad states:
  first-order formulas Init(V), Bad(V)

Filter Example

$\{H = h \wedge \forall x,y.\ n^*(x,y) \leftrightarrow L(x,y)\ \}$

```
void filter(Node h){
  Node i:=h; j:=null;
  while (i ≠ null){
     if ¬C(i) then {
       if i = h then h:=i.n
       else j.n:=i.n;
     }
     else j:=i;
     i:=i.n;
}}
```

$\{\ \ \forall z.\ \ h \neq null\ \ \wedge n^*(h,z) \rightarrow C(z)$

$\forall z.\ \ \ L(H,z) \wedge C(z) \rightarrow n^*(h,z)$

$\forall x,y.\ L(H,x) \wedge L(x,y) \wedge C(x) \wedge C(y) \rightarrow n^*(x,y)\ \ \}$

Filter Example

```
void filter(Node h){
  Node i:=h; j:=null;
```

$\{ H = h \wedge i{=}h \wedge j = null \wedge \forall x,y.\ n^*(x,y) \leftrightarrow L(x,y) \}$ — Init(V)

```
  while (i ≠ null){
     if ¬C(i) then {
        if i = h then h:=i.n
        else j.n:=i.n;
     }
     else j:=i;
     i:=i.n;
}}
```

(loop body: TR(V,V'))

$$\{\ i{=}null \rightarrow \forall z.\ h \neq null \wedge n^*(h,z) \rightarrow C(z)$$
$$\forall z.\ L(H,z) \wedge C(z) \rightarrow n^*(h,z)$$
$$\forall x,y.\ L(H,x) \wedge L(x,y) \wedge C(x) \wedge C(y) \rightarrow n^*(x,y)\ \}$$

P(V)

From Programs to Transition Systems

$H = h \wedge \forall x,y.\ n^*(x,y) \leftrightarrow L(x,y)$

$\forall z.\ \ h \neq null\ \ \wedge n^*(h,z) \rightarrow C(z)$
$\forall z.\ \ \ L(H,z) \wedge C(z) \rightarrow n^*(h,z)$
$\forall x,y.\ L(H,x) \wedge L(x,y) \wedge C(x) \wedge C(y) \rightarrow n^*(x,y)$

Problems:

- FOL + transitive closure is undecidable
- McCarthy assignment rule for wlp does not work for heap manipulations x.n := e

Reachability Predicates

- Use n* instead of n:

$$V = < h, i, j, null, n^*(\cdot, \cdot), C(\cdot) >$$

- Axiomatize n*:

Acyclicity + reflexivity
Transitivity
linearity

$$\Gamma_{\text{linOrd}} = \forall \alpha, \beta: n^*(\alpha, \beta) \wedge n^*(\beta, \alpha) \leftrightarrow \alpha = \beta \wedge$$
$$\forall \alpha, \beta, \gamma: n^*(\alpha, \beta) \wedge n^*(\beta, \gamma) \rightarrow n^*(\alpha, \gamma) \wedge$$
$$\forall \alpha, \beta, \gamma: n^*(\alpha, \beta) \wedge n^*(\alpha, \gamma) \rightarrow (n^*(\beta, \gamma) \vee n^*(\gamma, \beta))$$

Effectively Propositional (EPR)

- Satisfiability is deciadable
- Finite model property

Filter Example

```
void filter(Node h){
  Node i:=h; j:=null;
 { H = h ∧ i=h ∧ j =null ∧ ∀x,y. n*(x,y) ↔ L(x,y) }      } Init(V)
  while (i ≠ null){
     if ¬C(i) then {
        if i = h then h:=i.n
        else j.n:=i.n;                                   } TR(V,V')
     }
     else j:=i;
     i:=i.n;
}}
```

$\{\ i = null \rightarrow \forall z.\ h \neq null \wedge n^*(h,z) \rightarrow C(z)$
$\quad \forall z.\ L(H,z) \wedge C(z) \rightarrow n^*(h,z)$
$\quad \forall x,y.\ L(H,x) \wedge L(x,y) \wedge C(x) \wedge C(y) \rightarrow n^*(x,y)\ \}$ — P(V)

Filter Example

```
void filter(Node h){
   Node i:=h; j:=null;
  { H = h ∧ i=h ∧ j =null ∧ ∀x,y. n*(x,y) ↔ L(x,y) }        } Init(V)
   while {I} (i ≠ null){
      if ¬C(i) then {
         if i = h then h:=i.n
         else j.n:=i.n;
      }                                                     } TR(V,V')
      else j:=i;
      i:=i.n;
}}
```

$\{\ i{=}null \rightarrow \forall z.\ \ h \neq null\ \wedge n^*(h,z) \rightarrow C(z)$

$\forall z.\ \ L(H,z) \wedge C(z) \rightarrow n^*(h,z)$

$\forall x,y.\ L(H,x) \wedge L(x,y) \wedge C(x) \wedge C(y) \rightarrow n^*(x,y)\ \ \}$ } P(V)

Inductive Invariants

- Setting
  - V – relational vocabulary
  - TR(V, V’) – transition relation
  - Init(V) – initial states
  - Bad(V) – bad states (determined by assertions)
- I(V) is an inductive invariant if:
  - $Init \Rightarrow I$
  - $I(V) \wedge TR(V,V') \Rightarrow I(V')$
  - $I \Rightarrow \neg Bad$
- Infer inductive invariant with PDR (IC3)

Universal Property Directed Reachability

- Given: V, TR(V,V'), Init(V), Bad(V)
- UPDR searches for inductive invariant I(V) in the form of a universal formula

$$\underbrace{\forall \bar{x}\,(l_{1,1}(\bar{x}) \vee \ldots \vee l_{1,1}(\bar{x}))}_{\text{Clause / lemma}} \wedge \ldots \wedge \forall \bar{x}\,(l_{n,1}(\bar{x}) \vee \ldots \vee l_{n,m}(\bar{x}))$$

- iteratively infers universal lemmas until fixpoint

IC3 General Iteration

$F_i := F_i \wedge \neg\sigma_i$

$F_{i-1} := F_{i-1} \wedge \neg\sigma_{i-1}$

⋮

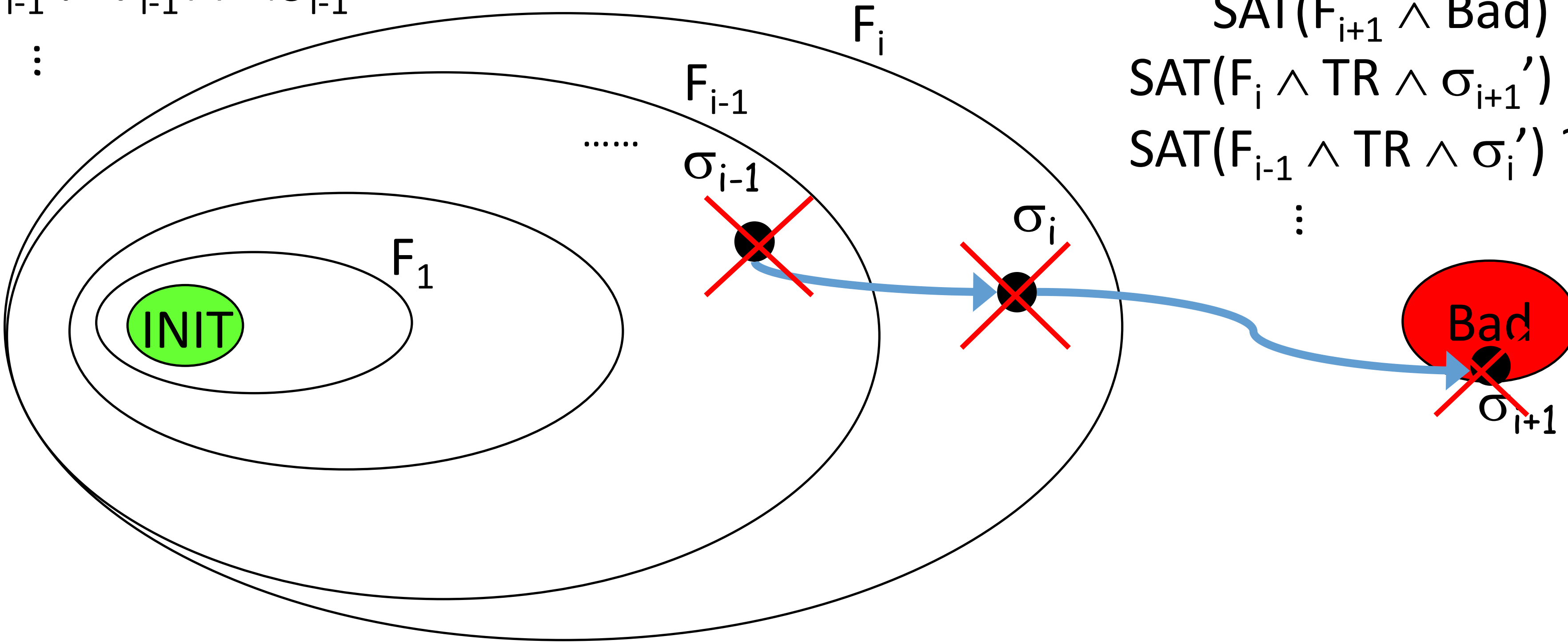
$F_{i+1} = true$ $F_{i+1} = true \wedge \neg\sigma_{i+1}$

$SAT(F_{i+1} \wedge Bad)$?

$SAT(F_i \wedge TR \wedge \sigma_{i+1}')$?

$SAT(F_{i-1} \wedge TR \wedge \sigma_i')$?

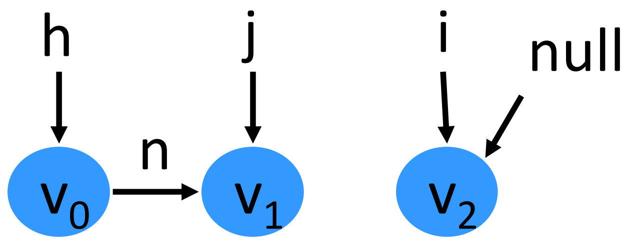
⋮



But now σ is not a formula!

Universal PDR (UPDR)

- “bad state” σ is a finite first-order model
  - use Diag(σ) as an abstraction of σ :

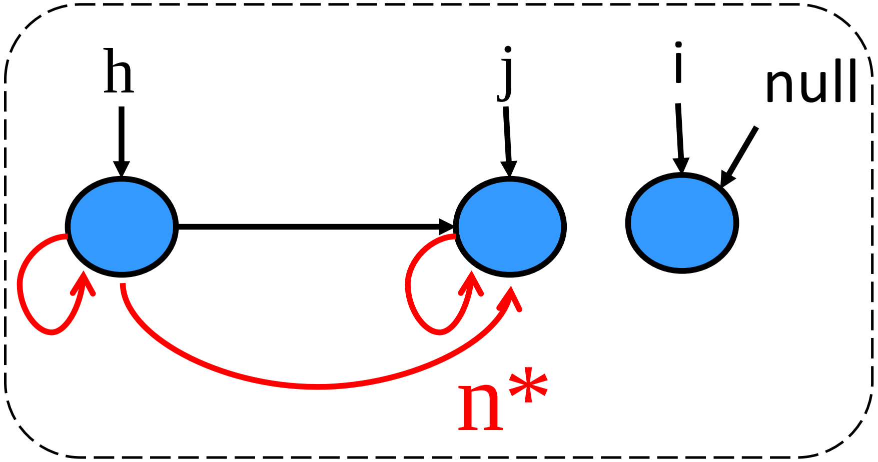


$$\exists x_0,x_1,x_2.\ x_0 \neq x_1 \wedge x_0 \neq x_2 \wedge x_1 \neq x_2 \wedge$$
$$h = x_0 \wedge j = x_1 \wedge i = x_2 \wedge null = x_2 \wedge$$
$$n^*(x_0,x_0) \wedge n^*(x_1,x_1) \wedge n^*(x_2,x_2) \wedge n^*(x_0,x_1) \wedge$$
$$\neg n^*(x_0,x_2) \wedge \neg n^*(x_1,x_0) \wedge \ldots$$

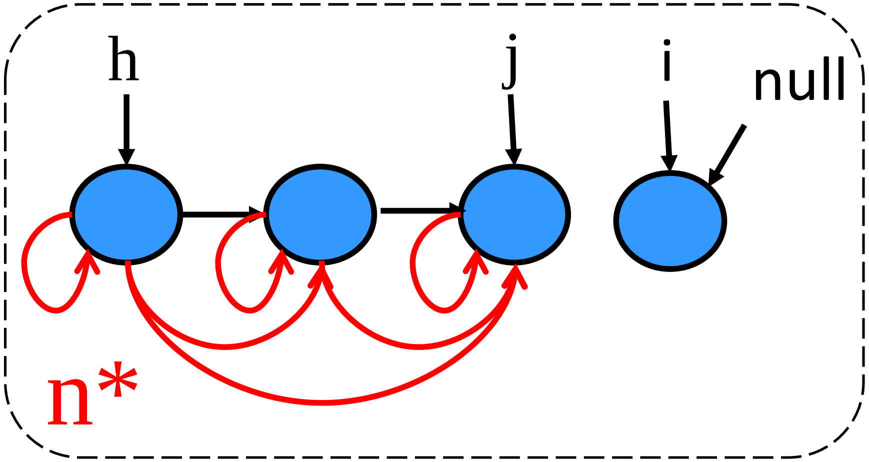
$\sigma' \models$ Diag(σ) iff σ is a sub-structure of σ'

Diagrams as Abstractions

- σ' |= Diag(σ)   iff   σ is a sub-structure of σ'



σ



σ'

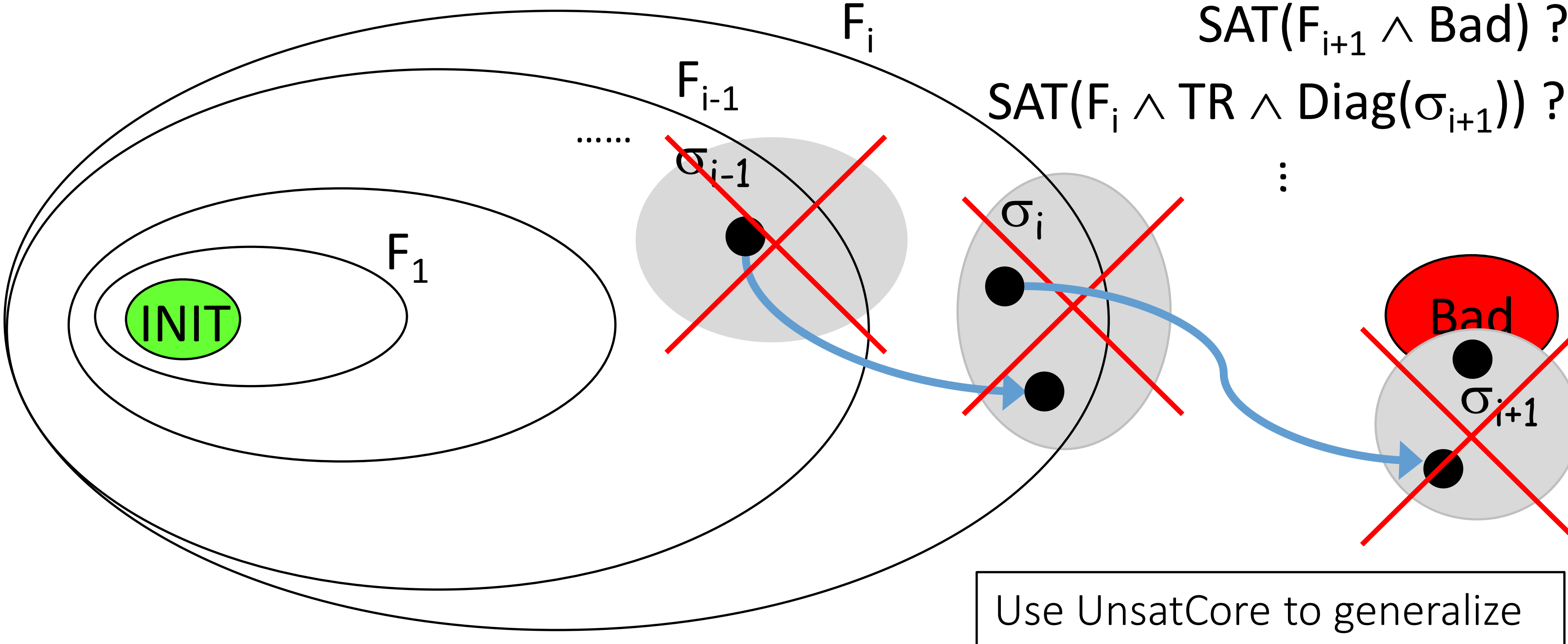
UPDR

F_{i+1} = true F_{i+1} = true $\wedge \neg \mathrm{Diag}(\sigma_{i+1})$

SAT($F_{i+1} \wedge$ Bad) ?

SAT($F_i \wedge TR \wedge \mathrm{Diag}(\sigma_{i+1})$) ?

⋮



If $\mathrm{Diag}(\sigma_{i+1})$ is reachable from F_i: continue backwards

If $\mathrm{Diag}(\sigma_j)$ is unreachable from F_{j-1}: $F_j := F_j \wedge \neg \mathrm{Diag}(\sigma_j)$

Why Diagrams?

- If there exists a universal inductive invariant I and σ_j is a "bad state", then *all* states in Diag(σ_j) are unreachable from Init
  - Blocking will succeed

- $F_j := F_j \land \neg \mathrm{Diag}(\sigma_j)$

  $\neg \mathrm{Diag}(\sigma_j)$: universal clause

=> F_1, F_2, ... are universal formulas

More Intuition

- If there exists a universal inductive invariant:

$$I = \underbrace{\forall \bar{x}\ (l_{1,1}(\bar{x}) \vee \ldots \vee l_{1,1}(\bar{x}))}_{\text{Clause / lemma}} \wedge \ldots \wedge \forall \bar{x}\ (l_{n,1}(\bar{x}) \vee \ldots \vee l_{n,m}(\bar{x}))$$

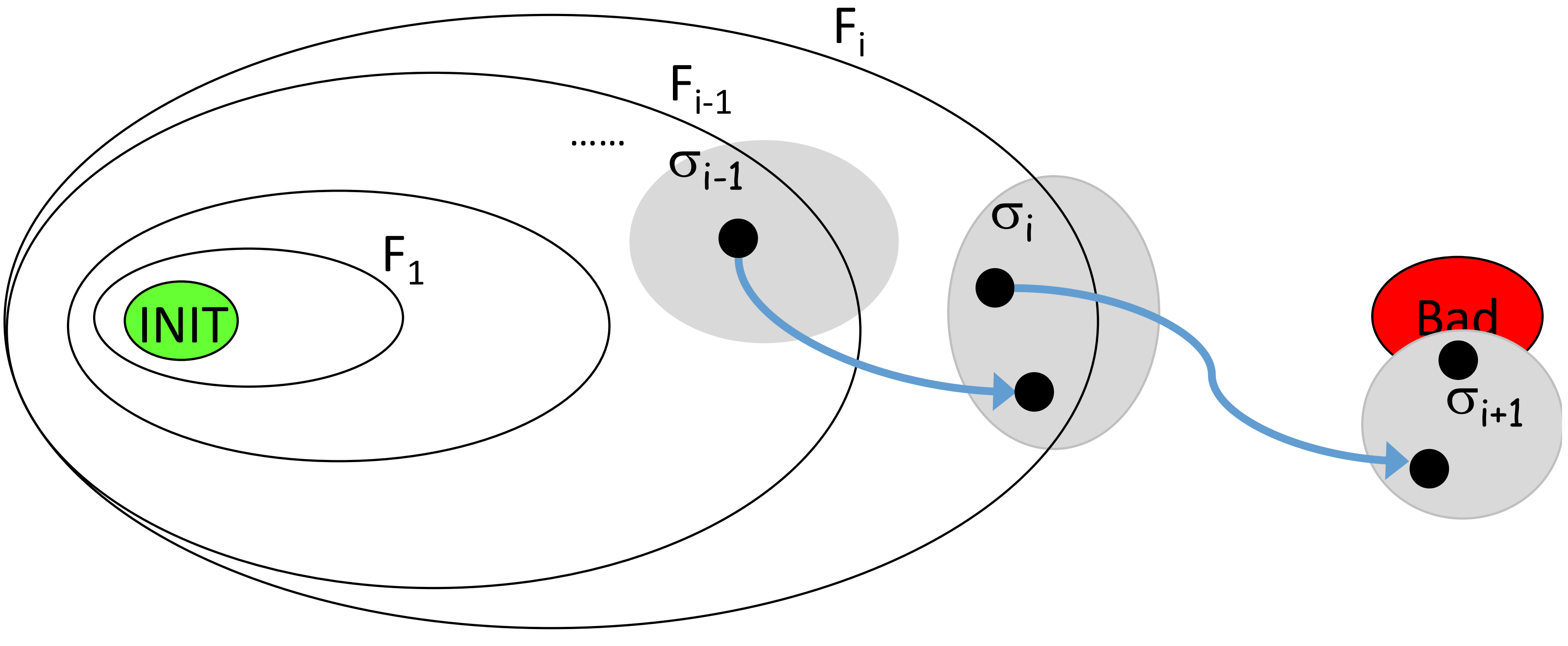
Then:

$$\neg I \equiv \underbrace{\exists \bar{x}\ (\neg l_{1,1}(\bar{x}) \wedge \ldots \wedge \neg l_{1,1}(\bar{x}))}_{\text{Cube}} \vee \ldots \vee \exists \bar{x}\ (\neg l_{n,1}(\bar{x}) \wedge \ldots \wedge \neg l_{n,m}(\bar{x}))$$

UPDR tries to generate and block cex models that "cover" all cubes in $\neg I$

UPDR: Possible Outcomes

- Fixpoint: universal inductive invariant found
- Abstract counterexample:



UPDR: Possible Outcomes (cont.)

- Fixpoint: universal inductive invariant found
- Abstract counterexample:

  Check if spurious using bounded model checking

  - If concrete counterexample found:
    - program is unsafe
  - If counterexample is spurious:
    - Unknown whether the program is safe, but
    - No universal inductive invariant exists
- Divergence

Filter Example

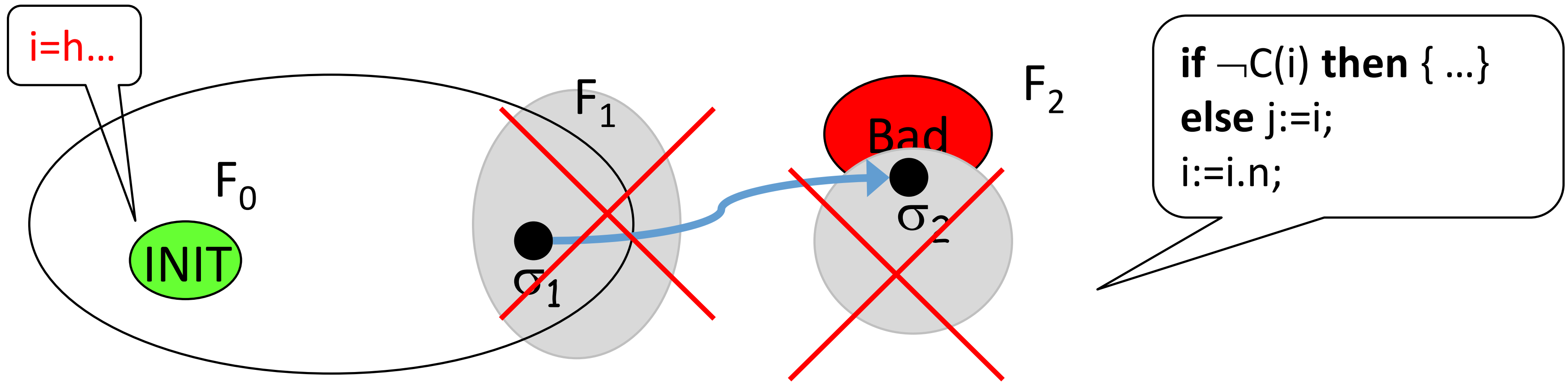
```
void filter(Node h){
  Node i:=h; j:=null;
 { H = h ∧ i=h ∧ j =null ∧ ∀x,y. n*(x,y) ↔ L(x,y) }        } Init(V)
  while {I} (i ≠ null){
     if ¬C(i) then {
        if i = h then h:=i.n
        else j.n:=i.n;
     }                                                    } TR(V,V')
     else j:=i;
     i:=i.n
}}
```

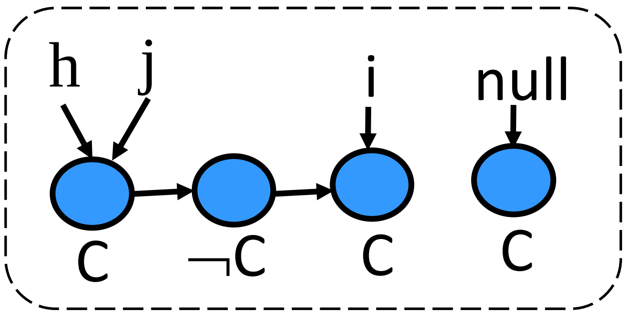
$\{ \; i=null \rightarrow \forall z.\; h \neq null \wedge n^*(h,z) \rightarrow C(z)$

$\forall z.\; L(H,z) \wedge C(z) \rightarrow n^*(h,z)$

$\forall x,y.\; L(H,x) \wedge L(x,y) \wedge C(x) \wedge C(y) \rightarrow n^*(x,y) \; \}$ — P(V)

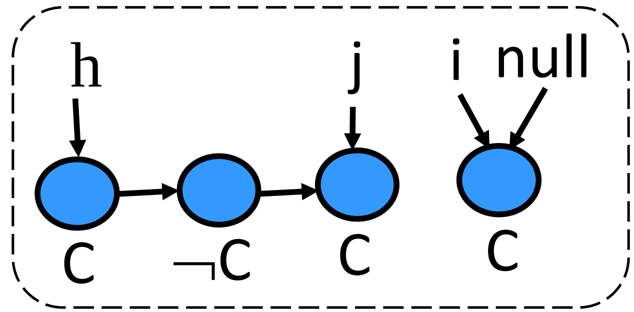
Filter Example: Frame 2





$\neg\exists x.\ (j \neq x \wedge \neg n^*(i, x) \wedge n^*(j,x))$

$\equiv \forall x.\ (j=x \vee n^*(i, x) \vee \neg n^*(j, x))$



$\forall x.\ (i \neq null \vee n^*(i, x) \vee \neg n^*(h, x) \vee h=x)$

$$\text{Bad} = \quad i=null \wedge \neg\ (\forall z.\ \ h \neq null \ \wedge n^*(h,z) \rightarrow C(z)$$

$$\forall z.\ \ L(H,z) \wedge C(z) \rightarrow n^*(h,z)$$

$$\forall x,y.\ L(H,x) \wedge L(x,y) \wedge C(x) \wedge C(y) \rightarrow n^*(x,y)$$

Inferred Invariant

- $i \neq h \wedge i \neq null \rightarrow n^*(j ; i)$
- $i \neq h \rightarrow C(h)$
- $n^*(h, j) \vee i \neq j$
- $\forall x.\ i \neq h \wedge n^*(j, x) \wedge x \neq j \rightarrow n^*(i, x)$
- $i \neq h \rightarrow C(j)$
- $\forall x.\ x = h \vee j = null \vee \neg n^*(h, x) \vee \neg n^*(h, j) \vee \neg C(j)$
- $\forall x.\ j \neq null \wedge n^*(h, x) \wedge x \neq h \wedge \neg C(x) \rightarrow n^*(j, x)$

Summary

Property Directed Reachability

- SAT-based
- Performs local reasoning, no unrolling
- Complete for finite state systems
- No need for predefined predicates