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: \( \text{AG } p \)

“\( p \) holds in every reachable state of the system”

Using automata-theoretic methods, model checking of all safety properties reduces to checking \( \text{AG } p \)

**Reachability:** Does the transition system have a finite run ending in a state satisfying \( \neg p \) ?
Finite-state system modeled as \((V, \text{INIT}, T)\):

- **V** – finite set of Boolean variables
  - Four states \(\Rightarrow\) Boolean variables: \(v_1, v_2\)
- **INIT(V)** – describes the set of initial states
  - \(\text{INIT} = \neg v_1\)
- **T(V,V')** – describes the set of transitions
  - \(T = (v_1' \leftrightarrow \neg v_1 \lor (v_1 \land v_2)) \land (v_2' \leftrightarrow (v_1 \land v_2))\)

Property:

- **p(V)** - describes the set of states satisfying \(p\)
  - \(p = \neg v_1 \lor \neg v_2\) (Bad = \(\neg p = v_1 \land v_2\))
Induction for proving AG P

• The simple case: P is an inductive invariant
  – \text{INIT}(V) \Rightarrow P(V)
  – P(V) \land T(V, V') \Rightarrow 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 \land T \Rightarrow F'
  - F \Rightarrow P
Invariant Inference by Forward Reachability

\[ R_{i+1}(V') \equiv R_i(V') \lor \exists V \ (R_i(V) \land T(V, V')) \]

\[ = R_i \lor \text{Img}(R_i, T) \]

R is the strongest inductive invariant
Invariant Inference by Approximate Reachability

\[ F_{i+1}(V') \iff F_i(V) \land T(V,V') \]

If \( F_{k+1} \equiv F_k \) then \( F_k \) is an inductive invariant
IC3  (Bradley, VMCAI 2010)

• IC3 = Incremental Construction of Inductive Clauses for Indubitable Correctness

• The Goal: Find an Inductive Invariant stronger than P
  – Recall: F is an inductive invariant stronger than P if
    • INIT => F
    • F \land 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) \(<F_0,F_1,...,F_{k+1}>\) s.t.
  - \(F_0 = \text{INIT}\)
  - \(F_i \Rightarrow F_{i+1}\) \(F_i \subseteq F_{i+1}\)
  - \(F_i \land T \Rightarrow F'_{i+1}\) 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
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 \& c \& T => c'

• **Notation:**
  - cube s: conjunction of literals
    * v₁ \& v₂ \& ¬v₃ - Represents a state
  - s is a cube => ¬s is a clause (DeMorgan)
IC3 - Initialization

• Check satisfiability of the two formulas:
  – INIT ∧ ¬P
  – INIT ∧ T ∧ ¬P’

• If at least one is satisfiable: cex found

• If both are unsatisfiable then:
  – INIT => P
  – INIT ∧ T => P’

• Therefore
  – F₀ = INIT, F₁ = P

  • <F₀,F₁> is an OARS

OARS:

– F₀ = INIT
– Fᵢ => Fᵢ₊₁
– Fᵢ ∧ T => F’ᵢ₊₁
– Fᵢ => P
IC3 - Iteration

- Our OARS contains $F_0$ and $F_1$
- Initialize $F_2$ to $P$
  - If $P$ is an inductive invariant – done! 😊
  - Otherwise: $F_1 \land T \not\Rightarrow F'_2$
    => $F_1$ should be strengthened

OARS:
- $F_0 = \text{INIT}$
- $F_i \Rightarrow F_{i+1}$
- $F_i \land T \Rightarrow F'_{i+1}$
- $F_i \Rightarrow P$
IC3 - Iteration

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

OARS:
– $F_0 = \text{INIT}$
– $F_i \Rightarrow F_{i+1}$
– $F_i \land T \Rightarrow F'_{i+1}$
– $F_i \Rightarrow P$
IC3 - Iteration

• Is s reachable in one transition from the previous set?
  – backward search: Check $F_0 \land T \land s'$
  – If satisfiable, s is reachable from $F_0$: CEX
  – Otherwise, block s, i.e. remove it from $F_1$
    • $F_1 = F_1 \land \neg s$
IC3 - Iteration

- Iterate this process until $F_1 \land T \land \neg P'$ becomes unsatisfiable
  - $F_1 \land T \Rightarrow P'$ holds
    - $(F \land T \land \neg P')$ unsat IFF $(F \land T \Rightarrow P')$ valid
  - $< F_0, F_1, F_2 >$ is an OARS
IC3 - Iteration

- New iteration, initialize $F_3$ to $P$, check $F_2 \land T \land \neg P'$
  - If satisfiable, get $s$ that can reach $\neg P$
  - Now check if $s$ can be reached from $F_1$ by $F_1 \land T \land s'$
  - If it can be reached, get $t$ and try to block it
IC3 - Iteration

• To block t, check $F_0 \land T \land t'$
  - If satisfiable, a CEX
  - If not, t is blocked, get a “new” $t^*$ by $F_1 \land T \land s'$ and try to block $t^*$
IC3 - Iteration

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

......You get the picture 😊
If $s_k$ is reachable (in $k$ steps): counterexample

If $s_k$ is unreachable: strengthen $F_k$ to exclude $s_k$
General Iteration

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

i.e. $F_k \land T \implies P'$
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 $F_{k-1} \land T \Rightarrow \neg s'$ holds
  – Therefore s is not reachable in k transitions
  – $F_k := F_k \land \neg s$
• 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 $F_{k-1} \land T \Rightarrow \neg s'$
- And, $\neg s$ is a clause
- Generalization:
  Find a sub-clause $c \subseteq \neg s$ s.t. $F_{k-1} \land T \Rightarrow c'$
  - Sub clause means less literals
  - Less literals implies less satisfying assignments
    - $(a \lor b)$ vs. $(a \lor b \lor c)$
    - $c \Rightarrow \neg s$ i.e. $c$ is a stronger fact
- $F_k := F_k \land c$
  - 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. $F_{k-1} \land T \Rightarrow c'$?

• Trial and Error
  – Try to remove literals from $\neg s$ while $F_{k-1} \land T \land \neg c'$ remains unsatisfiable

• Use the UnSAT Core
  – $F_{k-1} \land T \land s'$ 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 $F_{k-1}$ (!!)
  – $F_{k-1} \land T \Rightarrow F_k$
  – $F_k \Rightarrow P$
  – 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 $F_{k-1} \land T \Rightarrow \neg s'$ holds
  - $s$ is not reachable in $k$ transitions
- So, this is equivalent to $F_{k-1} \land \neg s \land T \Rightarrow \neg s'$
- Further INIT $\Rightarrow \neg s$
  - Otherwise, CEX! (INIT $\neq \neg s$ IFF $s$ is in INIT)
- This looks familiar!
  - $\neg s$ is inductive relative to $F_{k-1}$
Inductive Generalization

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

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

- c is also inductive relative to $F_{k-1}, F_{k-2}, \ldots, F_0$
  - Add c to all of these sets
  - $F_i^* = F_i \land c$

- $F_i^* \land T \Rightarrow F_{i+1}^*$ 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 $j > i$ ($F_i \Rightarrow F_j$)
  - 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 $c$
  – $F_i = F_i \land c$
• $c$ is a clause learnt at level i
• try to push $c$ forward for $j > i$
  – If $F_j \land c \land T \Rightarrow c'$ holds
    • $c$ is inductive in level $j$
    • $F_{j+1} = F_{j+1} \land c$
  – 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
If $\text{INIT} \land \lnot P$ is SAT return false; // CEX
If $\text{INIT} \land T \land \lnot P'$ is SAT return false; // CEX
$\text{OARS} = \langle \text{INIT}, P \rangle$; // $\langle F_0, F_1 \rangle$
k = 1
while ($\text{OARS}.\text{is\_fixpoint}() == \text{false}$) do
    extend($\text{OARS}$);  // $F_{k+1} = P$
    while ($F_k \land T \land \lnot P'$ is SAT) do
        $s = \text{get\_state}()$
        If (block\_state($s, k) == \text{false}$) // 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

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); // F_k+1 = P
    while (F_k ∧ T ∧ ¬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
General Iteration

\[ \text{SAT}(F_k \land T \land \lnot P') \]

\[ F_k, F_{k+1} = P \]

\[ S_k \]

\[ \text{Bad} \]
General Iteration: Alternative

\[ \text{SAT}(F_{k+1} \land \neg P') ? \]
\[ \text{SAT}(F_k \land T \land s') ? \]

\[ F_{k+1} = \text{true} \]
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
{ h is a list }

```java
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:
  \[ V = \langle h, i, j, \text{null}, n(\cdot, \cdot), C(\cdot) \rangle \]

Program state:

- first-order structure
  \[ D = \{e_1, \ldots, e_5\} \]
  \[ l(h) = e_1 \]
  \[ l(i) = e_3 \]
  \[ l(j) = e_2 \]
  \[ l(\text{null}) = e_5 \]
  \[ l(n) = \{(e_1, e_2), (e_2, e_3)\ldots\} \]
Filter Example: Assertions

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

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

V= < h, i, j, null, 
  n(\cdot, \cdot), C(\cdot), 
  H, L(\cdot, \cdot) >

Auxiliary symbols

\{ \forall z. h\neq\text{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 \( \text{Init}(V), \text{Bad}(V) \)
Filter Example

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

```java
void filter(Node h)
    Node i:=h; j:=null;
    while (i \neq null)
        if \neg 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 \text{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) \}
Filter Example

```c
void filter(Node h){
    Node i:=h; j:=null;
    { H = h ∧ i=h ∧ j =null ∧ ∀x,y. n*(x,y) ↔ L(x,y) }
    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;
    }
    { i=null → ∀z. h≠null ∧ n*(h,z) → C(z)
      ∀z. L(H,z) ∧ C(z) → n*(h,z)
      ∀x,y. L(H,x) ∧ L(x,y) ∧ C(x) ∧ C(y) → n*(x,y) }
}
```
From Programs to Transition Systems

\[ H = h \land \forall x,y. \ n^*(x,y) \iff L(x,y) \]

\[ \forall z. \ h \neq \text{null} \land n^*(h,z) \rightarrow C(z) \]
\[ \forall z. \ L(H,z) \land C(z) \rightarrow n^*(h,z) \]
\[ \forall x,y. \ L(H,x) \land L(x,y) \land C(x) \land C(y) \rightarrow n^*(x,y) \]

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 $n$:
  \[ V = \langle h, i, j, \text{null}, n^*(\cdot, \cdot), C(\cdot) \rangle \]

• Axiomatize $n^*$:

\[
\Gamma_{\text{linOrd}} = \forall \alpha, \beta: n^*(\alpha, \beta) \land n^*(\beta, \alpha) \iff \alpha = \beta \land \\
\forall \alpha, \beta, \gamma: n^*(\alpha, \beta) \land n^*(\beta, \gamma) \rightarrow n^*(\alpha, \gamma) \land \\
\forall \alpha, \beta, \gamma: n^*(\alpha, \beta) \land n^*(\alpha, \gamma) \rightarrow (n^*(\beta, \gamma) \lor n^*(\gamma, \beta))
\]

Effectively Propositional (EPR)

• Satisfiability is decidable

• Finite model property
Filter Example

```markdown
void filter(Node h){
    Node i:=h; j:=null;
    { H = h ∧ i=h ∧ j=null ∧ ∀x,y. n*(x,y) ⇔ L(x,y) }
    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;
    }
}

{ i=null → ∀z. h≠null ∧ n*(h,z) → C(z)
    ∀z. L(H,z) ∧ C(z) → n*(h,z)
    ∀x,y. L(H,x) ∧ L(x,y) ∧ C(x) ∧ C(y) → n*(x,y) }
```
void filter(Node h){
    Node i:=h; j:=null;
    { H = h \land i=h \land j =null \land \forall x,y. n^*(x,y) \leftrightarrow L(x,y) }
    while {i} (i \neq null){
        if \neg C(i) then {
            if i = h then h:=i.n
            else j.n:=i.n;
        }
        else j:=i;
        i:=i.n;
    }
    { i=null \rightarrow \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) }
}
Inductive Invariants

• Setting
  – $V$ – relational vocabulary
  – $\text{TR}(V, V')$ – transition relation
  – $\text{Init}(V)$ – initial states
  – $\text{Bad}(V)$ – bad states (determined by assertions)

• $I(V)$ is an inductive invariant if:
  – $\text{Init} \implies I$
  – $I(V) \land \text{TR}(V, V') \implies I(V')$
  – $I \implies \neg \text{Bad}$

• Infer inductive invariant with PDR (IC3)
Universal Property Directed Reachability

• Given: \( V, \text{TR}(V,V'), \text{Init}(V), \text{Bad}(V) \)
• UPDR searches for inductive invariant \( I(V) \) in the form of a universal formula

\[
\forall \bar{x} \ (l_{1,1}(\bar{x}) \lor \ldots \lor l_{1,1}(\bar{x})) \land \ldots \land \forall \bar{x} \ (l_{n,1}(\bar{x}) \lor \ldots \lor l_{n,m}(\bar{x}))
\]

Clause / lemma

• iteratively infers universal lemmas until fixpoint
IC3 General Iteration

\[ F_i := F_i \land \neg \sigma_i \]
\[ F_{i-1} := F_{i-1} \land \neg \sigma_{i-1} \]
\[ \vdots \]

\[ F_i \]
\[ F_{i-1} \]
\[ \sigma_{i-1} \]
\[ \sigma_i \]

But now \( \sigma \) is not a formula!
Universal PDR (UPDR)

- “bad state” $\sigma$ is a finite first-order model
  - use $\text{Diag}(\sigma)$ as an abstraction of $\sigma$:

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

$\sigma' \models \text{Diag}(\sigma)$ iff $\sigma$ is a sub-structure of $\sigma'$
Diagrams as Abstractions

- $\sigma' \models \text{Diag}(\sigma)$ iff $\sigma$ is a sub-structure of $\sigma'$

![Diagram]

$\sigma$

$h$

$i$

$j$

$null$

$n^*$

$\sigma'$

$h$

$i$

$j$

$null$

$n^*$
If $\text{Diag}(\sigma_{i+1})$ is reachable from $F_i$: continue backwards
If $\text{Diag}(\sigma_j)$ is unreachable from $F_{j-1}$: $F_j := F_j \land \neg \text{Diag}(\sigma_j)$
Why Diagrams?

• If there exists a universal inductive invariant I and $\sigma_j$ is a “bad state”, then 
  all states in $\text{Diag}(\sigma_j)$ are unreachable from Init
  – Blocking will succeed

• $F_j := F_j \land \neg \text{Diag}(\sigma_j)$

  universal clause

=> $F_1, F_2, \ldots$ are universal formulas
More Intuition

• If there exists a universal inductive invariant:

\[ I = \bigwedge \forall \bar{x} \, (l_{1,1}(\bar{x}) \lor \ldots \lor l_{1,1}(\bar{x})) \land \ldots \land \forall \bar{x} \, (l_{n,1}(\bar{x}) \lor \ldots \lor l_{n,m}(\bar{x})) \]

Then:

\[ \neg I \equiv \exists \bar{x} \, (\neg l_{1,1}(\bar{x}) \land \ldots \land \neg l_{1,1}(\bar{x})) \lor \ldots \lor \exists \bar{x} \, (\neg l_{n,1}(\bar{x}) \land \ldots \land \neg l_{n,m}(\bar{x})) \]

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
void filter(Node h){
    Node i:=h; j:=null;
    { H = h ∧ i=h ∧ j =null ∧ ∀x,y. n*(x,y) ↔ L(x,y) }
    while {l} (i ≠ null){
        if ¬C(i) then {
            if i = h then h:=i.n
            else j.n:=i.n;
        }
        else j:=i;
        i:=i.n
    }
    { i=null → ∀z. h≠null ∧ n*(h,z) → C(z)
        ∀z. L(H,z) ∧ C(z) → n*(h,z)
        ∀x,y. L(H,x) ∧ L(x,y) ∧ C(x) ∧ C(y) → n*(x,y) }
Filter Example: Frame 2

Bad = i=null ∧ ¬ (∀z. h≠null ∧ n*(h,z) → C(z))
∀z. L(H,z) ∧ C(z) → n*(h,z)
∀x,y. L(H,x) ∧ L(x,y) ∧ C(x) ∧ C(y) → n*(x,y)
Inferred Invariant

- \( i \neq h \land i \neq \text{null} \rightarrow n^*(j; i) \)
- \( i \neq h \rightarrow C(h) \)
- \( n^*(h, j) \lor i \neq j \)
- \( \forall x. i \neq h \land n^*(j, x) \land x \neq j \rightarrow n^*(i, x) \)
- \( i \neq h \rightarrow C(j) \)
- \( \forall x. x = h \lor j = \text{null} \lor \neg n^*(h, x) \lor \neg n^*(h, j) \lor \neg C(j) \)
- \( \forall x. j \neq \text{null} \land n^*(h, x) \land x \neq h \land \neg C(x) \rightarrow n^*(j, x) \)
Summary

Property Directed Reachability

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