Software Model Checking

with

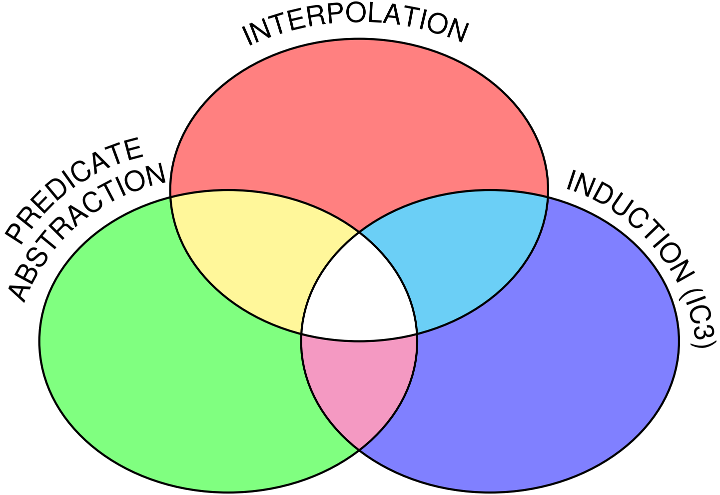
Predicate Abstraction, Interpolation, & IC3

Johannes Birgmeier, Aaron Bradley,
Georg Weissenbacher

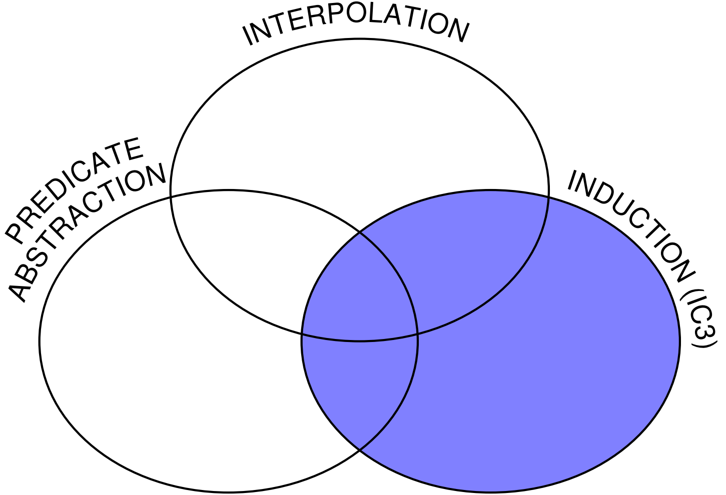
Challenges in (Software) Model Checking

1. Finding Inductive Invariants
2. Scalability (State Space Explosion)

How we will address these challenges



Part I: IC3



Incremental Construction of Inductive Clauses for Indubitable Correctness

- Verification of *finite state systems*
- Aaron Bradley
  SAT-Based Model Checking without Unrolling [VMCAI'11]
- Given: Finite State Transition System
  - Initial states $I \subseteq S$
  - Transition relation $T \subseteq S \times S$
  - Safety property P

Incremental Construction of Inductive Clauses for Indubitable Correctness

- Verification of *finite state systems*
- Aaron Bradley
  SAT-Based Model Checking without Unrolling [VMCAI'11]
- Given: Finite State Transition System
  - Initial states $I \subseteq S$
  - Transition relation $T \subseteq S \times S$
  - Safety property P
- Goal: **Inductive** invariant F
  - $I(s) \Rightarrow F(s)$,
  - $F(s) \wedge T(s, s') \Rightarrow F(s')$
  - $F(s) \Rightarrow P(s)$

IC3

Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

$$I \Leftrightarrow F_0 \tag{1}$$

$$\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} \tag{2}$$

$$\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P \tag{3}$$

$$\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} \tag{4}$$

IC3

Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

$$\boxed{\begin{array}{lr} I \Leftrightarrow F_0 & (1) \\ \forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} & (2) \\ \forall 0 \leq i \leq k \,.\, F_i \Rightarrow P & (3) \\ \forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} & (4) \end{array}}$$

(1) F_0 represents the initial states

IC3

Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

$$
\begin{array}{ll}
I \Leftrightarrow F_0 & (1) \\
\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} & (2) \\
\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P & (3) \\
\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} & (4)
\end{array}
$$

(1) F_0 represents the initial states

(2+4) F_i *over-approximates* states reachable in $\leq i$ steps

IC3

Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

$$
\begin{aligned}
&I \Leftrightarrow F_0 && (1)\\
&\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} && (2)\\
&\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P && (3)\\
&\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} && (4)
\end{aligned}
$$

(1) F_0 represents the initial states

(2+4) F_i *over-approximates* states reachable in $\leq i$ steps

(3) All F_i are *safe*

IC3

Sequence $F_0, F_1, \ldots, F_k$ of candidates for invariant

$$
\begin{aligned}
& I \Leftrightarrow F_0 && (1)\\
& \forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} && (2)\\
& \forall 0 \leq i \leq k \,.\, F_i \Rightarrow P && (3)\\
& \forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} && (4)
\end{aligned}
$$

Important properties of algorithm:

- New frame F_{k+1} is added if F_k is “safe”, k increased
- Over-approximation $F_0, F_1, \ldots, F_k$ is refined *incrementally*
- Inductiveness is primary goal

IC3

$$I \Leftrightarrow F_0 \quad (1)$$

$$\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} \quad (2)$$

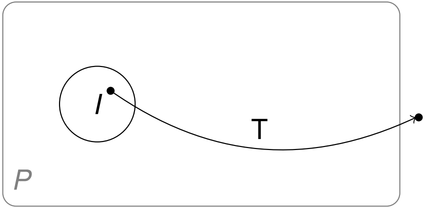
$$\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P \quad (3)$$

$$\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} \quad (4)$$

IC3

$$
\begin{aligned}
& I \Leftrightarrow F_0 && (1)\\
& \forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} && (2)\\
& \forall 0 \leq i \leq k \,.\, F_i \Rightarrow P && (3)\\
& \forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} && (4)
\end{aligned}
$$

Step 1: Check whether $I \Rightarrow P$ and $I \wedge T \Rightarrow P'$



IC3

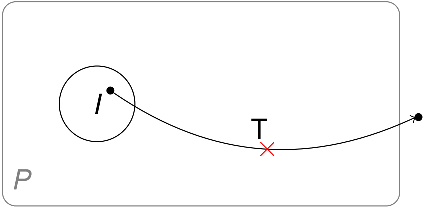
$$I \Leftrightarrow F_0 \quad (1)$$

$$\forall 0 \leq i < k \, . \, F_i \Rightarrow F_{i+1} \quad (2)$$

$$\forall 0 \leq i \leq k \, . \, F_i \Rightarrow P \quad (3)$$

$$\forall 0 \leq i < k \, . \, F_i \land T \Rightarrow F'_{i+1} \quad (4)$$

Step 1: Check whether $I \Rightarrow P$ and $I \land T \Rightarrow P'$



IC3

$$I \Leftrightarrow F_0 \quad (1)$$

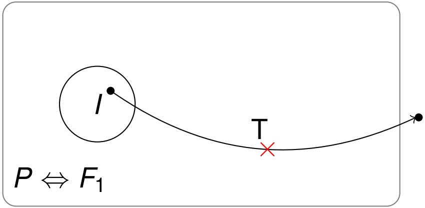
$$\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} \quad (2)$$

$$\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P \quad (3)$$

$$\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} \quad (4)$$

Step 1: Check whether $I \Rightarrow P$ and $I \wedge T \Rightarrow P'$

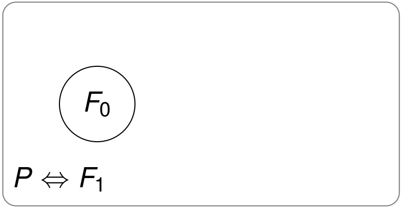
✓ *Expand:* Add $F_1 \Leftrightarrow P$ to sequence of frames $F_0, \ldots$



IC3

$$
\begin{aligned}
& I \Leftrightarrow F_0 && (1) \\
& \forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} && (2) \\
& \forall 0 \leq i \leq k \,.\, F_i \Rightarrow P && (3) \\
& \forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} && (4)
\end{aligned}
$$

Step 2: Check whether $F_1 \wedge T \Rightarrow P'$

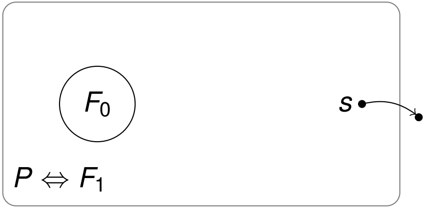


IC3

$$I \Leftrightarrow F_0 \quad (1)$$
$$\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} \quad (2)$$
$$\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P \quad (3)$$
$$\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} \quad (4)$$

Step 2: Check whether $F_1 \wedge T \Rightarrow P'$

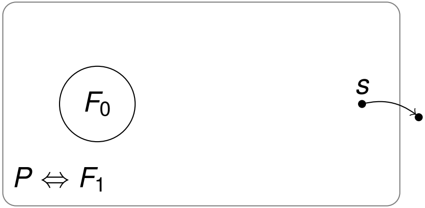
✗ There's a state s such that $F_1 \wedge s \wedge T \wedge \neg P'$



IC3: Consecution

What do we know about s?

- $s \notin F_0$, otherwise would have discovered s earlier



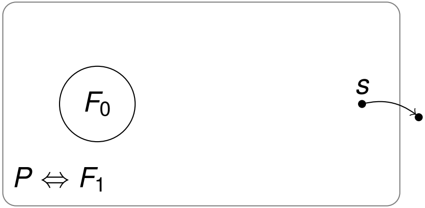
IC3: Consecution

What do we know about s?

- $s \notin F_0$, otherwise would have discovered s earlier

Try to show that s is unreachable from F_0:

- $\underbrace{F_0 \wedge \neg s \wedge T \Rightarrow \neg s'}_{\text{consecution check}}$



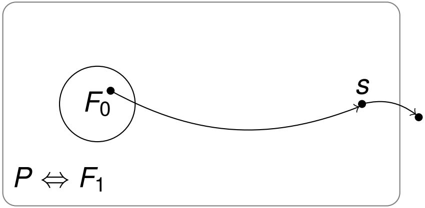
IC3: Consecution

What do we know about s?

- $s \notin F_0$, otherwise would have discovered s earlier

Try to show that s is unreachable from F_0:

- $\underbrace{F_0 \land \neg s \land T \Rightarrow \neg s'}_{\text{consecution check}}$
- If this doesn't hold, s has a predecessor in F_0 ↯



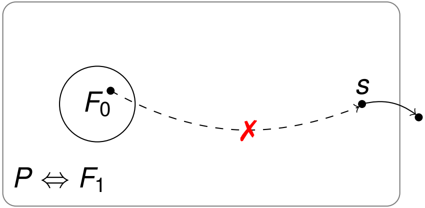
IC3: Consecution

What do we know about s?

- $s \notin F_0$, otherwise would have discovered s earlier

Try to show that s is unreachable from F_0:

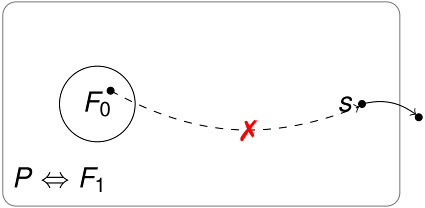
- $\underbrace{F_0 \wedge \neg s \wedge T \Rightarrow \neg s'}_{\text{consecution check}}$
- If this holds, s is *inductive relative to* F_0



IC3: Relative Inductiveness

$$F_0 \land \neg s \land T \Rightarrow \neg s'$$

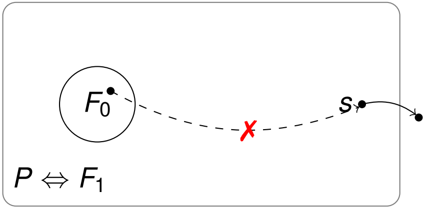
- We can replace F_1 with $F_1 \land \neg s$



IC3: Relative Inductiveness

$$F_0 \wedge \neg s \wedge T \Rightarrow \neg s'$$

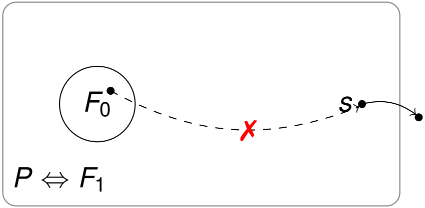
- We can replace F_1 with $F_1 \wedge \neg s$
- But that would only eliminate one state!



IC3: Generalization

Could eliminate s from F_1. But we can do better!

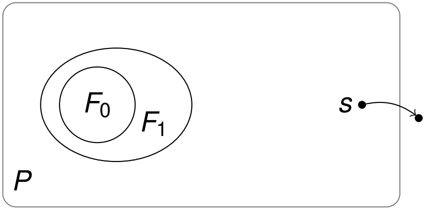
- Try to generalize s:
  - ✓ $F_0 \wedge \neg s \wedge T \Rightarrow \neg s'$
  - Find $c \subseteq \neg s$ such that $F_0 \wedge c \wedge T \Rightarrow c'$ (consider subsets of clause $\neg s$)



IC3: Generalization

Could eliminate s from F_1. But we can do better!

- Try to generalize s:
  - ✓ $F_0 \land \neg s \land T \Rightarrow \neg s'$
  - Find $c \subseteq \neg s$ such that $F_0 \land c \land T \Rightarrow c'$ (consider subsets of clause $\neg s$)
  - $F_1 := F_1 \land c$



IC3

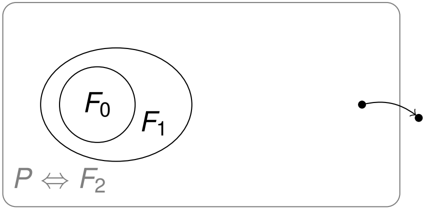
$$I \Leftrightarrow F_0 \quad (1)$$

$$\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} \quad (2)$$

$$\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P \quad (3)$$

$$\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} \quad (4)$$

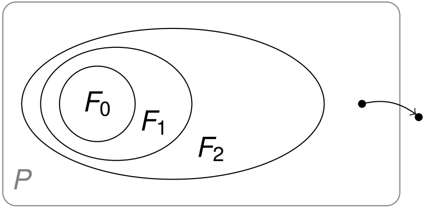
Once no more bad states reachable from F_1, *expand...*



IC3

$$
\begin{aligned}
&I \Leftrightarrow F_0 && (1)\\
&\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} && (2)\\
&\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P && (3)\\
&\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} && (4)
\end{aligned}
$$

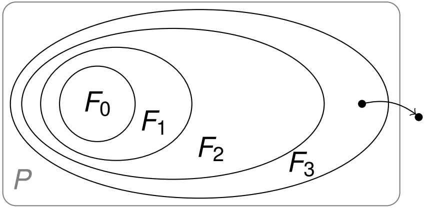
Once no more bad states reachable from F_2, *expand...*



IC3

$$
\begin{aligned}
&I \Leftrightarrow F_0 && (1)\\
&\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} && (2)\\
&\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P && (3)\\
&\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} && (4)
\end{aligned}
$$

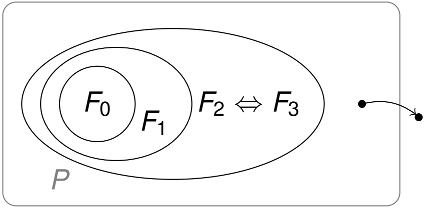
Once no more bad states reachable from F_2, *expand...*



IC3

$$I \Leftrightarrow F_0 \quad (1)$$
$$\forall 0 \leq i < k \,.\, F_i \Rightarrow F_{i+1} \quad (2)$$
$$\forall 0 \leq i \leq k \,.\, F_i \Rightarrow P \quad (3)$$
$$\forall 0 \leq i < k \,.\, F_i \wedge T \Rightarrow F'_{i+1} \quad (4)$$

Until we eventually reach a fixed point.

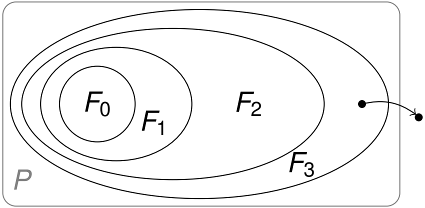


IC3

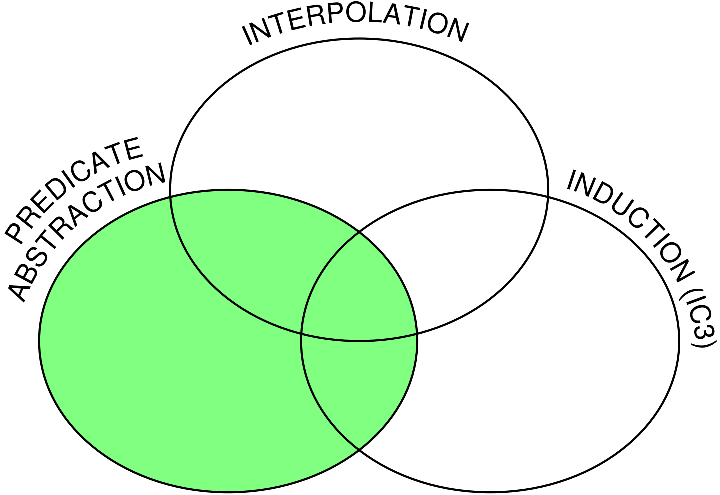
Does this work for software?

Yes; simply replace SAT solver with SMT solver, but:

- State space much larger or infinite
- Will painstakingly eliminate single/small sets of states
- High risk of divergence

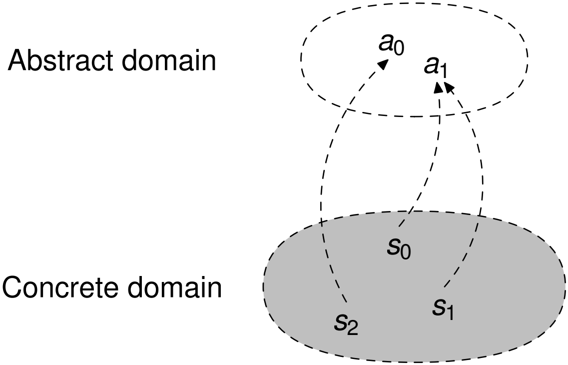


Part II: Predicate Abstraction



Predicate Abstraction: A Form of Abstract Interpretation

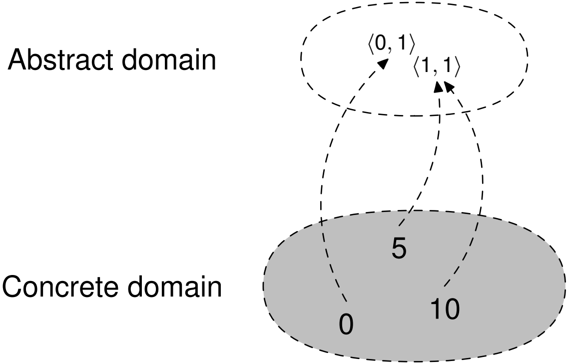
- Map concrete states to abstract states
- Reduce size of state space
  - Obtain finite representation



Abstract Domain: Set of Predicates

Map concrete states to abstract states by evaluating predicates:

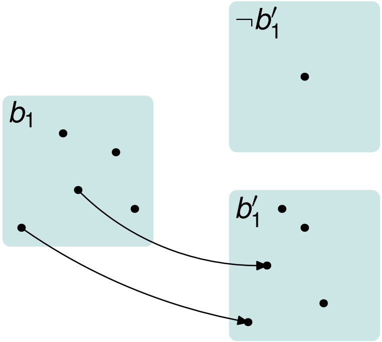
- Concrete variable: `i`
- Predicates: $b_1 \equiv (\mathtt{i} \neq 0)$ and $b_2 \equiv (\mathtt{i} \leq 10)$



Predicate Abstraction: Explicit Abstract Transition Relation

Example: Abstraction of `i++` and $b_1 \widehat{=} (\texttt{i} \neq 0)$

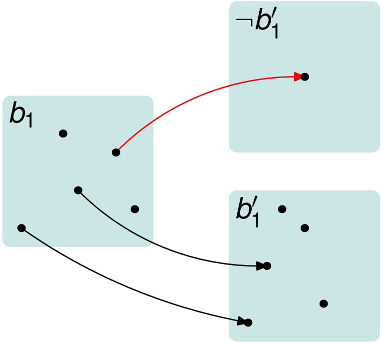
- We have to account for all possibilities!



Predicate Abstraction: Explicit Abstract Transition Relation

Example: Abstraction of `i++` and $b_1 \widehat{=} (\mathtt{i} \neq 0)$

- We have to account for all possibilities!
  - Even if there is just a single transition from $\mathtt{i} \neq 0$ to $\mathtt{i} = 0$!



Predicate Abstraction IC3 Style

Construction of explicit abstract transition relation

- requires many calls to SMT solver
- is computationally expensive

Predicate Abstraction IC3 Style

Construction of explicit abstract transition relation

- requires many calls to SMT solver
- is computationally expensive
- contrary to the spirit of IC3 (focus on single states)

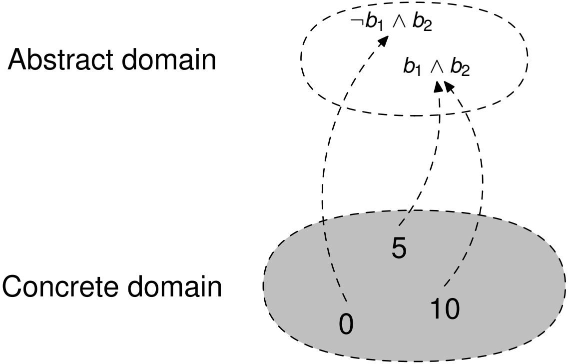
Predicate Abstraction IC3 Style

Construction of explicit abstract transition relation

- requires many calls to SMT solver
- is computationally expensive
- contrary to the spirit of IC3 (focus on single states)

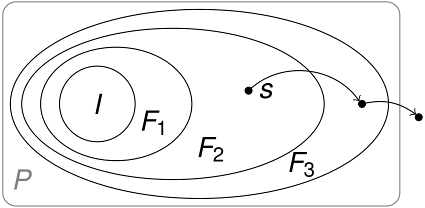
Abstraction of single states is computationally cheap!

- Predicates: $b_1 \equiv (\texttt{i} \neq 0)$, $b_2 \equiv (\texttt{i} \leq 10)$



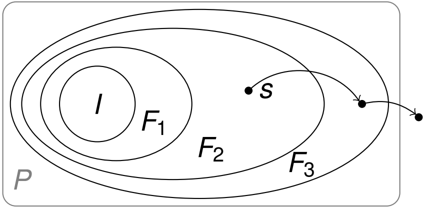
Predicate Abstraction IC3 Style

- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula



Predicate Abstraction IC3 Style

- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

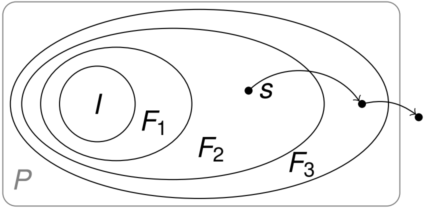


Predicate Abstraction IC3 Style

- $F_0, F_1, \dots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Check consecution for s:

$$F_1 \wedge \neg s \wedge T \Rightarrow \neg s'$$



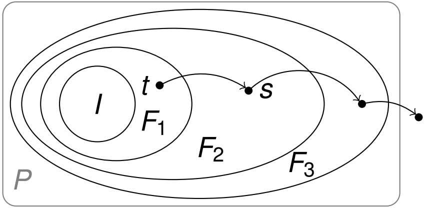
Predicate Abstraction IC3 Style

- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Check consecution for s:

$$F_1 \wedge \neg s \wedge T \Rightarrow \neg s'$$

If s *not* relative inductive, proceed with predecessor t

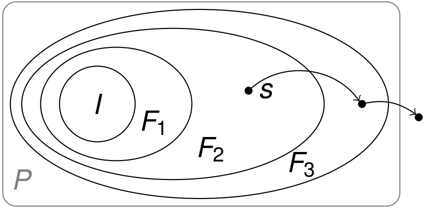


Predicate Abstraction / Abstract Consecution

- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Consecution:

$$F_1 \wedge \neg s \wedge T \Rightarrow \neg s'$$



Predicate Abstraction / Abstract Consecution

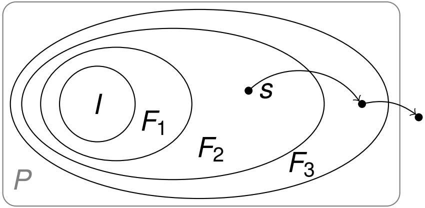
- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Abstract Consecution:

$$F_1 \land \neg\hat{s} \land T \Rightarrow \neg\hat{s}'$$

$$\uparrow \qquad\qquad \uparrow$$

$$F_1 \land \neg s \land T \Rightarrow \neg s'$$



Predicate Abstraction / Abstract Consecution

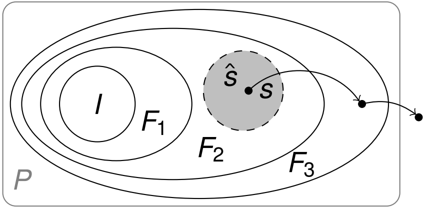
- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Abstract Consecution:

$$F_1 \wedge \neg \hat{s} \wedge T \Rightarrow \neg \hat{s}'$$

$$\uparrow \qquad\qquad \uparrow$$

$$F_1 \wedge \neg s \wedge T \Rightarrow \neg s'$$

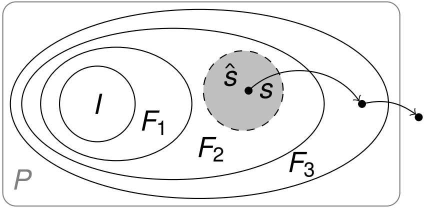


Predicate Abstraction / Abstract Consecution

- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Check *abstract* consecution (instead of concrete):

$$F_1 \land \neg\hat{s} \land T \Rightarrow \neg\hat{s}' \quad \checkmark$$



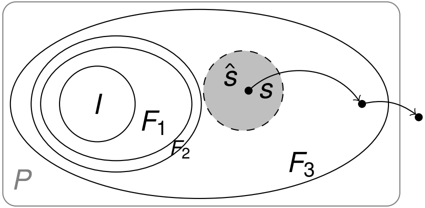
Predicate Abstraction / Abstract Consecution

- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Check *abstract* consecution (instead of concrete):

$$F_1 \land \neg\hat{s} \land T \Rightarrow \neg\hat{s}' \quad \checkmark$$

Replace F_2 with $F_2 \land c$, where clause $c \subseteq \neg\hat{s}$



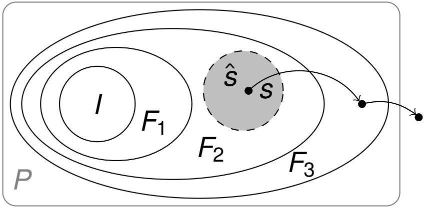
Abstract Consecution Failure

- $F_0, F_1, \ldots F_k$: CNF over *predicates*
- Transition relation T: program as SMT formula
- state s: *concrete* predecessor of bad state

Check consecution:

$$F_1 \land \neg s \land T \Rightarrow \neg s' \qquad ✗$$

But what if *abstract consecution* fails?



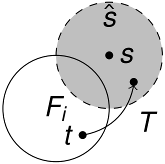
Abstract Consecution Failure

$$F_1 \wedge \neg\hat{s} \wedge T \Rightarrow \neg\hat{s}' \text{ ✗}$$

$$\uparrow \qquad\qquad \uparrow$$

$$F_1 \wedge \neg s \wedge T \Rightarrow \neg s' \text{ ✓}$$

Then $\hat{s}$ has a concrete predecessor $t \in F_1$ that does not lead to s in one step.



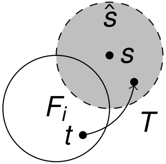
Abstract Consecution Failure

$$F_1 \wedge \neg\hat{s} \wedge T \Rightarrow \neg\hat{s}' \text{ ✗}$$

$$\uparrow \qquad\qquad \uparrow$$

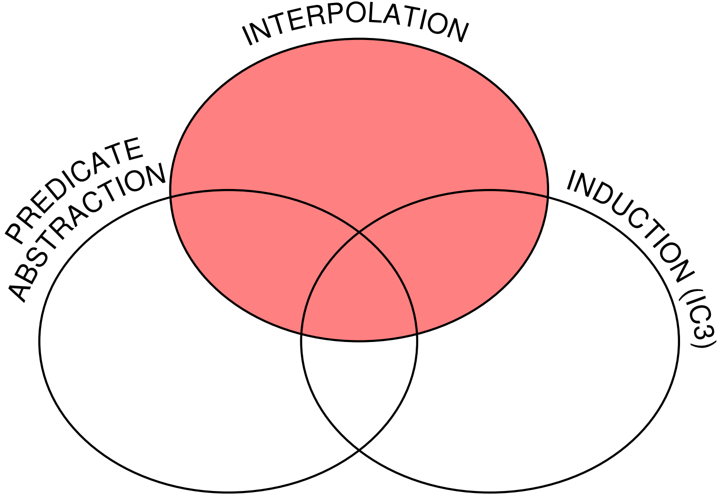
$$F_1 \wedge \neg s \wedge T \Rightarrow \neg s' \text{ ✓}$$

Then $\hat{s}$ has a concrete predecessor $t \in F_1$ that does not lead to s in one step.



- Our *abstract domain* is too *imprecise*

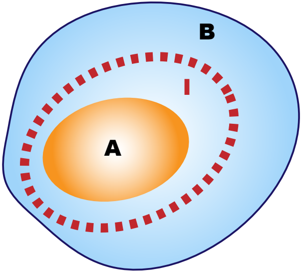
Part III: Craig Interpolation



What is a Craig Interpolant?

Craig interpolant I for formula $A \Rightarrow B$:

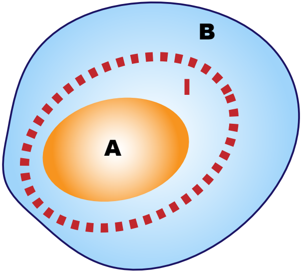
- $A \Rightarrow I$ and $I \Rightarrow B$
- all non-logical symbols in I occur in A *as well as* in B



What is a Craig Interpolant?

Craig interpolant I for formula $A \Rightarrow B$:

- $A \Rightarrow I$ and $I \Rightarrow B$
- all non-logical symbols in I occur in A *as well as* in B



Can be provided by contemporary SMT solvers for many theories

Refinement for Abstract Consecution Failure

$$F_1 \wedge \neg \hat{s} \wedge T \Rightarrow \neg \hat{s}' \; ✗$$

$$\uparrow \qquad\qquad \uparrow$$

$$F_1 \wedge \neg s \wedge T \Rightarrow \neg s' \; ✓$$

How to save the day with interpolants:

Refinement for Abstract Consecution Failure

$$F_1 \wedge \neg \hat{s} \wedge T \Rightarrow \neg \hat{s}' \text{ ✗}$$

$$\uparrow \qquad\qquad \uparrow$$

$$\underbrace{F_1 \wedge \neg s \wedge T}_{A} \Rightarrow \underbrace{\neg s'}_{B} \text{ ✓}$$

How to save the day with interpolants:

Refinement for Abstract Consecution Failure

$$F_1 \wedge \neg\hat{s} \wedge T \Rightarrow \neg\hat{s}' \text{ ✗}$$

$$\uparrow \qquad\qquad \uparrow$$

$$\underbrace{F_1 \wedge \neg s \wedge T}_{A} \Rightarrow \underbrace{\neg s'}_{B} \text{ ✓}$$

How to save the day with interpolants:

1. Compute interpolant R'
   - $F_1 \wedge \neg s \wedge T \Rightarrow R'$
   - $R' \Rightarrow \neg s'$

Refinement for Abstract Consecution Failure

$$F_1 \wedge \neg\hat{s} \wedge T \Rightarrow \neg\hat{s}' \text{ ✗}$$

$$\uparrow \qquad\qquad \uparrow$$

$$\underbrace{F_1 \wedge \neg s \wedge T}_{A} \Rightarrow \underbrace{\neg s'}_{B} \text{ ✓}$$

How to save the day with interpolants:

1. Compute interpolant R'
   - $F_1 \wedge \neg s \wedge T \Rightarrow R'$
   - $R' \Rightarrow \neg s'$
2. Add $\neg R$ to the abstract domain
   - Note: $s \Rightarrow \neg R$, therefore $\hat{s} \wedge \neg R$ is new abstraction of s

Refinement for Abstract Consecution Failure

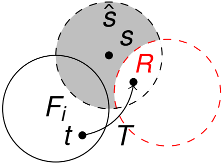
$$F_1 \wedge \neg\hat{s} \wedge T \Rightarrow \neg\hat{s}' \ ✗$$

$$\uparrow \qquad\qquad \uparrow$$

$$\underbrace{F_1 \wedge \neg s \wedge T}_{A} \Rightarrow \underbrace{\neg s'}_{B} \ ✓$$

How to save the day with interpolants:

1. Compute interpolant R'
   - $F_1 \wedge \neg s \wedge T \Rightarrow R'$
   - $R' \Rightarrow \neg s'$
2. Add $\neg R$ to the abstract domain
   - Note: $s \Rightarrow \neg R$, therefore $\hat{s} \wedge \neg R$ is new abstraction of s



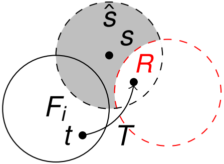
Refinement for Abstract Consecution Failure

$$F_1 \wedge (\neg \hat{s} \vee R) \wedge T \Rightarrow (\neg \hat{s}' \vee R') \checkmark$$

$$\underbrace{F_1 \wedge \neg s \wedge T}_{A} \Rightarrow \underbrace{\neg s'}_{B} \checkmark$$

How to save the day with interpolants:

1. Compute interpolant R'
   - $F_1 \wedge \neg s \wedge T \Rightarrow R'$
   - $R' \Rightarrow \neg s'$
2. Add $\neg R$ to the abstract domain
   - Note: $s \Rightarrow \neg R$, therefore $\hat{s} \wedge \neg R$ is new abstraction of s



Refinement IC3 Style

Refinement via Craig Interpolation

- without unrolling! (unlike most other SMC approaches)
- therefore extremely light-weight

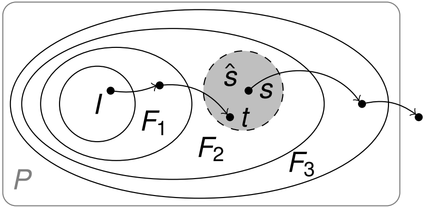
Refinement IC3 Style

Refinement via Craig Interpolation

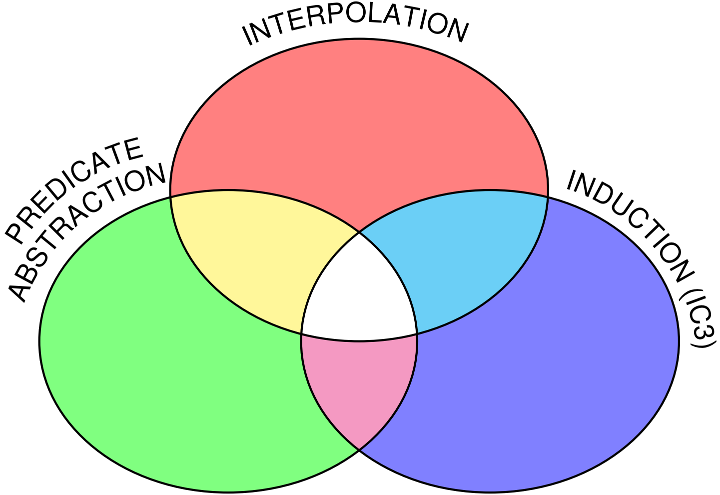
- without unrolling! (unlike most other SMC approaches)
- therefore extremely light-weight

Also: Refinement can be *delayed*!

- Spurious state may be eliminated later without refinement



Conclusion: IC3 + Predicate Abstraction + Interpolation



Conclusion: IC3 + Predicate Abstraction + Interpolation

Evaluation of prototype implementation:

- on InvGen, Dagger, "Beautiful Interpolants" benchmarks
  - using mostly linear arithmetic
- solve substantially more problems than CPAChecker
  - details in our CAV'14 paper!
- delaying refinement pays off (evaluated several strategies)

Conclusion: IC3 + Predicate Abstraction + Interpolation

Evaluation of prototype implementation:

- on INVGEN, DAGGER, "Beautiful Interpolants" benchmarks
  - using mostly linear arithmetic
- solve substantially more problems than CPAChecker
  - details in our CAV'14 paper!
- delaying refinement pays off (evaluated several strategies)

Lessons learned:

- Induction focus of IC3 successfully transferred to software
- Predicate abstraction in this setting is *cheap*
- Refinement doesn't require unrolling!