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

The diagram shows the relationships between Interpolation, Induction (IC3), and Predicate Abstraction. The intersection of all three concepts represents the focus or the main topic of Part I: IC3.
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 $F$
  
  - $I(s) \Rightarrow F(s)$,
  - $F(s) \land T(s, s') \Rightarrow F(s')$
  - $F(s) \Rightarrow P(s)$
Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

\[ I \iff F_0 \] (1)

\[ \forall 0 \leq i < k . F_i \Rightarrow F_{i+1} \] (2)

\[ \forall 0 \leq i \leq k . F_i \Rightarrow P \] (3)

\[ \forall 0 \leq i < k . F_i \land T \Rightarrow F_{i+1}' \] (4)
Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

\begin{align*}
I & \iff F_0 \quad (1) \\
\forall 0 \leq i < k . F_i & \implies F_{i+1} \quad (2) \\
\forall 0 \leq i \leq k . F_i & \implies P \quad (3) \\
\forall 0 \leq i < k . F_i \land T & \implies F'_{i+1} \quad (4)
\end{align*}

(1) $F_0$ represents the initial states
Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

$I \Leftrightarrow F_0$ \hspace{1cm} (1)

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

$\forall 0 \leq i \leq k . F_i \Rightarrow P$ \hspace{1cm} (3)

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

(1) $F_0$ represents the initial states

(2+4) $F_i$ over-approximates states reachable in $\leq i$ steps
Approach: Construct sequence $F_0, F_1, \ldots, F_k$ of candidates

\[
\begin{align*}
I & \Leftrightarrow F_0 & (1) \\
\forall 0 \leq i < k . F_i & \Rightarrow F_{i+1} & (2) \\
\forall 0 \leq i \leq k . F_i & \Rightarrow P & (3) \\
\forall 0 \leq i < k . F_i \land T & \Rightarrow F_{i+1}' & (4)
\end{align*}
\]

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

\[
\begin{align*}
I & \iff F_0 \quad (1) \\
\forall 0 \leq i < k. F_i & \Rightarrow F_{i+1} \quad (2) \\
\forall 0 \leq i \leq k. F_i & \Rightarrow P \quad (3) \\
\forall 0 \leq i < k. F_i \land T & \Rightarrow F'_i \quad (4)
\end{align*}
\]

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 \textit{incrementally}
- Inductiveness is primary goal
\( I \Leftrightarrow F_0 \) \hspace{1cm} (1)
\( \forall 0 \leq i < k . F_i \Rightarrow F_{i+1} \) \hspace{1cm} (2)
\( \forall 0 \leq i \leq k . F_i \Rightarrow P \) \hspace{1cm} (3)
\( \forall 0 \leq i < k . F_i \land T \Rightarrow F'_i \) \hspace{1cm} (4)
\[ I \iff F_0 \quad (1) \]
\[ \forall 0 \leq i < k . F_i \Rightarrow F_{i+1} \quad (2) \]
\[ \forall 0 \leq i \leq k . F_i \Rightarrow P \quad (3) \]
\[ \forall 0 \leq i < k . F_i \land T \Rightarrow F'_{i+1} \quad (4) \]

Step 1: Check whether \( I \Rightarrow P \) and \( I \land T \Rightarrow P' \)
Step 1: Check whether $I \Rightarrow P$ and $I \land T \Rightarrow P'$
$I \Leftrightarrow F_0$ \hfill (1)

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

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

$\forall 0 \leq i < k . F_i \land T \Rightarrow F_{i+1}'$ \hfill (4)

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

✓ *Expand:* Add $F_1 \Leftrightarrow P$ to sequence of frames $F_0, \ldots$
Step 2: Check whether $F_1 \land T \Rightarrow P'$
$I \Leftrightarrow F_0$  \hspace{1cm} (1)

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

$\forall 0 \leq i \leq k . F_i \Rightarrow P$ \hspace{1cm} (3)

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

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

\textbf{✗} There’s a state $s$ such that $F_1 \land s \land T \land \neg P'$
IC3: Consecution

What do we know about $s$?

- $s \not\in F_0$, otherwise would have discovered $s$ earlier

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

If this doesn't hold, $s$ has a predecessor in $F_0$. 

$P \iff F_1$
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 $F_0$:

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

  consecution check

![Diagram showing $F_0$, $F_1$, and $s$]
IC3: Consecution

What do we know about $s$?

- $s \not\in F_0$, otherwise would have discovered $s$ earlier

Try to show that $s$ is unreachable from $F_0$:

- $F_0 \land \neg s \land T \Rightarrow \neg s'$
  - consecution check

- If this doesn’t hold, $s$ has a predecessor in $F_0$
IC3: Consecution

What do we know about $s$?

- $s \not\in F_0$, otherwise would have discovered $s$ earlier

Try to show that $s$ is unreachable from $F_0$:

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

  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

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

- We can replace $F_1$ with $F_1 \land \neg s$
- But that would only eliminate one state!
IC3: Generalization

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

- Try to generalize $s$:
  - $F_0 \land \neg s \land T \Rightarrow \neg s'$
  - Find $c \subseteq \neg s$ such that $F_0 \land c \land T \Rightarrow c'$
    (consider subsets of clause $\neg s$)
IC3: Generalization

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

- Try to generalize \( s \):
  - \( F_0 \land \lnot s \land T \Rightarrow \lnot s' \)
  - Find \( c \subseteq \lnot s \) such that \( F_0 \land c \land T \Rightarrow c' \)
    (consider subsets of clause \( \lnot s \))
  - \( F_1 := F_1 \land c \)
\[
\begin{align*}
I & \Leftrightarrow F_0 \quad (1) \\
\forall 0 \leq i < k . \ F_i & \Rightarrow F_{i+1} \quad (2) \\
\forall 0 \leq i \leq k . \ F_i & \Rightarrow P \quad (3) \\
\forall 0 \leq i < k . \ F_i \land T & \Rightarrow F_{i+1}' \quad (4)
\end{align*}
\]

Once no more bad states reachable from $F_1$, expand...
\[ I \iff F_0 \quad (1) \]

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

\[ \forall 0 \leq i \leq k . F_i \Rightarrow P \quad (3) \]

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

Once no more bad states reachable from \( F_2 \), expand…
$I \iff F_0$ \hfill (1)

\[ \forall 0 \leq i < k . F_i \Rightarrow F_{i+1} \] \hfill (2)

\[ \forall 0 \leq i \leq k . F_i \Rightarrow P \] \hfill (3)

\[ \forall 0 \leq i < k . F_i \wedge T \Rightarrow F'_{i+1} \] \hfill (4)

Once no more bad states reachable from $F_2$, expand...
\( I \iff F_0 \) \hspace{1cm} (1)
\[ \forall 0 \leq i < k . F_i \Rightarrow F_{i+1} \] \hspace{1cm} (2)
\[ \forall 0 \leq i \leq k . F_i \Rightarrow P \] \hspace{1cm} (3)
\[ \forall 0 \leq i < k . F_i \land T \Rightarrow F'_{i+1} \] \hspace{1cm} (4)

Until we eventually reach a fixed point.
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

Concrete domain

\[ s_0, s_1, s_2 \]

\[ a_0, a_1 \]
Abstract Domain: Set of Predicates

Map concrete states to abstract states by evaluating predicates:

- Concrete variable: $i$
- Predicates: $b_1 \equiv (i \neq 0)$ and $b_2 \equiv (i \leq 10)$
Example: Abstraction of $i++$ and $b_1 \equiv (i \neq 0)$

- We have to account for all possibilities!
Example: Abstraction of $i++$ and $b_1 \equiv (i \neq 0)$

- We have to account for all possibilities!
  - Even if there is just a single transition from $i \neq 0$ to $i = 0$!
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: $b_1 \equiv (i \neq 0)$, $b_2 \equiv (i \leq 10)$

Abstract domain

Concrete domain
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

![Diagram showing predicate abstraction in 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

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

Check consecution for $s$:

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

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

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:

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

\[
F_1 \land \neg \hat{s} \land T \Rightarrow \neg \hat{s}'
\]

\[
F_1 \land \neg s \land T \Rightarrow \neg s'
\]
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:

\[ F_1 \land \neg \hat{s} \land T \Rightarrow \neg \hat{s}' \]

\[ F_1 \land \neg s \land T \Rightarrow \neg s' \]

\[ P \]

\[ I \]

\[ F_1 \]

\[ F_2 \]

\[ F_3 \]
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):

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

![Diagram showing abstraction and transition]
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):

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

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

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

But what if abstract consecution fails?

But what if abstract consecution fails?
Abstract Consecution Failure

\[ F_1 \land \lnot \hat{s} \land T \Rightarrow \lnot \hat{s}' \ \times \]

\[ F_1 \land \lnot s \land T \Rightarrow \lnot s' \ \checkmark \]

Then \( \hat{s} \) has a concrete predecessor \( t \in F_1 \) that does not lead to \( s \) in one step.
Abstract Consecution Failure

\[ F_1 \land \neg \hat{s} \land T \Rightarrow \neg \hat{s}' \quad \text{X} \]

\[ F_1 \land \neg s \land T \Rightarrow \neg s' \quad \text{✓} \]

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 $A \implies B$:

- $A \implies I$ and $I \implies B$
- 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

\[ F_1 \land \neg \hat{s} \land T \Rightarrow \neg \hat{s}' \ x \]

\[ F_1 \land \neg s \land T \Rightarrow \neg s' \ \checkmark \]

How to save the day with interpolants:
Refinement for Abstract Consecution Failure

\[ F_1 \land \neg \hat{s} \land T \implies \neg \hat{s}' \]

How to save the day with interpolants:
Refinement for Abstract Consecution Failure

\[ F_1 \land \neg \hat{s} \land T \Rightarrow \neg \hat{s}' \]

How to save the day with interpolants:

1. Compute interpolant \( R' \)
   - \( F_1 \land \neg s \land T \Rightarrow R' \)
   - \( R' \Rightarrow \neg s' \)
Refinement for Abstract Consecution Failure

\[ F_1 \land \neg \hat{s} \land T \Rightarrow \neg \hat{s}' \]

How to save the day with interpolants:

1. Compute interpolant \( R' \)
   - \( F_1 \land \neg s \land T \Rightarrow R' \)
   - \( R' \Rightarrow \neg s' \)

2. Add \( \neg R \) to the abstract domain
   - Note: \( s \Rightarrow \neg R \), therefore \( \hat{s} \land \neg R \) is new abstraction of \( s \)
Refinement for Abstract Consecution Failure

\[ F_1 \land \neg \hat{s} \land T \Rightarrow \neg \hat{s}' \]

\[ \begin{array}{c}
A \\
F_1 \land \neg s \land T \Rightarrow \neg s' \\
B
\end{array} \]

How to save the day with interpolants:

1. Compute interpolant \( R' \)
   - \( F_1 \land \neg s \land T \Rightarrow R' \)
   - \( R' \Rightarrow \neg s' \)

2. Add \( \neg R \) to the abstract domain
   - Note: \( s \Rightarrow \neg R \), therefore \( \hat{s} \land \neg R \) is new abstraction of \( s \)
Refinement for Abstract Consecution Failure

\[ F_1 \land (\neg \hat{s} \lor R) \land T \Rightarrow (\neg \hat{s}' \lor R') \]

\[ F_1 \land \neg s \land T \Rightarrow \neg s' \]

How to save the day with interpolants:

1. Compute interpolant \( R' \)
   - \( F_1 \land \neg s \land T \Rightarrow R' \)
   - \( R' \Rightarrow \neg s' \)

2. Add \( \neg R \) to the abstract domain
   - Note: \( s \Rightarrow \neg R \), therefore \( \hat{s} \land \neg R \) 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!