Software Verification using Predicate Abstraction and Iterative Refinement

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Sagar Chaki
March 16, 2011
This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

NO WARRANTY

THIS MATERIAL OF CARNEGIE MELLON UNIVERSITY AND ITS SOFTWARE ENGINEERING INSTITUTE IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.
Outline

Overview of Model Checking

Creating Models from C Code: Predicate Abstraction

Eliminating spurious behaviors from the model: Abstraction Refinement

Concluding remarks: research directions, tools etc.
Model Checking

Algorithm for answering queries about behaviors of state machines
• Given a state machine $M$ and a query $\phi$ does $M \models \phi$?

Standard formulation:
• $M$ is a Kripke structure
• $\phi$ is a temporal logic formula
  – Computational Tree Logic (CTL)
  – Linear Temporal Logic (LTL)
• We’ll use (slight) variants

Discovered independently by Clarke & Emerson and Queille & Sifakis in the early 1980’s
Models: Doubly Labeled State Machines

\[ \Sigma(M) = \{ a, b, c, d, e, f \} \quad AP = \{ p, q, r \} \]
Query 1: State-Event LTL Formulas

Whenever $p$ holds, $e$ happens some time in the future: $G ( p \Rightarrow F e )$
Query 2: State-Event LTL Formulas

$p$ and $r$ are never true at the same time: $G(\neg p \lor \neg r)$
Query 3: State-Event LTL Formulas

Whenever $p$ holds, $a$ happens some time in the future: $G ( p \Rightarrow F \ a )$
Query 3: State-Event LTL Formulas

Whenever $p$ holds, $a$ happens some time in the future: $G ( p \Rightarrow F a )$
Scalability of Model Checking

Explicit statespace exploration: early 1980s

- Tens of thousands of states

Symbolic statespace exploration: millions of states

- Binary Decision Diagrams (BDD) : early 1990’s
- Bounded Model Checking: late 1990’s
  - Based on propositional satisfiability (SAT) technology

Abstraction and compositional reasoning

- $10^{120}$ to effectively infinite statespaces (particularly for software)
Models of C Code

```c
if (x) {
    y = x;
} else {
    y = x + 1;
}
assert (y);
```

Program: Syntax  Control Flow Graph  Model: Semantics  Infinite State
**Abstraction**

**Partition** concrete statespace into abstract states

- Each abstract state $S$ corresponds to a set of concrete states $s$
- We write $\alpha(s)$ to mean the abstract state corresponding to $s$
- We define $\gamma(S) = \{ s \mid S = \alpha(s) \}$

**Fix the transitions existentially**

- $S \rightarrow S' \iff \exists s \in \gamma(S). \exists s' \in \gamma(S') . s \rightarrow s'$
- $S \rightarrow S' \iff \exists s \in \gamma(S). \exists s' \in \gamma(S') . s \rightarrow s'$

**Abstraction is conservative**

- If a State/Event-LTL property holds on the abstraction, it also holds on the program
- However, the converse is not true: a property that fails on the abstraction may still hold on the program
Example
Predicate Abstraction

if (x)

no

y = x + 1

yes

y = x

assert (y)

Partition the statespace based on values of a finite set of predicates on program variables
Predicate Abstraction

if (x)

no

y = x + 1

y = x

yes

assert (y)

States where y ≠ 0

¬P

States where y = 0

P ≡ (y == 0)

P

ERROR

Actions are unimportant
Predicate Abstraction

\[
\text{if (x)}
\]

\[
\text{no} \quad \text{yes}
\]

\[
y = x + 1 \quad y = x
\]

\[
\text{assert (y)}
\]

\[
\neg P \quad P
\]

\[
P \equiv ( y == 0 )
\]

States where \( y \neq 0 \)

States where \( y = 0 \)

Call SAT Checker

ERROR
Predicate Abstraction

if (x)
y = x + 1
y = x

no yes

assert (y)

States where y ≠ 0

SAT Checker Query:
y ≠ 0 ∧
x = 0 ∧
x′ = x ∧
y′ = y ∧
y′ ≠ 0

SAT Checker Answer:
SAT and here’s a solution
x=0, y=1, x′=0, y′=1

P ≡ (y == 0)
Predicate Abstraction

if (x)

no

y = x + 1

yes

y = x

assert (y)

SAT Checker Query:

\[ y \neq 0 \land x \neq 0 \land x' = x \land y' = y \land y' \neq 0 \]

SAT Checker Answer:

SAT and here’s a solution:

\[ x=1, y=1, x'=1, y'=1 \]

\[ P \equiv ( y == 0 ) \]
Predicate Abstraction

\[ y = x + 1 \]
\[ \text{if (}x\text{)} \]
\[ y = x \]

no

y = x + 1

yes

y = x

assert (y)

States where y \neq 0

SAT Checker Query:
\[ y \neq 0 \land x' = x \land y' = x + 1 \land y' \neq 0 \]

SAT Checker Answer:
SAT and here’s a solution
\[ x=1, y=1, x'=1, y'=2 \]

\[ P \equiv (y == 0) \]
Predicate Abstraction

if (x)

no

yes

y = x + 1

y = x

assert (y)

States where y \neq 0

SAT Checker Query:

y \neq 0 \land
x' = x \land
y' = x \land
y' \neq 0

SAT Checker Answer:

SAT and here’s a solution
x=1, y=1, x'=1, y'=1

P \equiv (y == 0)
Predicate Abstraction

```
y = x + 1
if (x)
y = x
no yes
```

States where \( y \neq 0 \)

States where \( y = 0 \)

\( P \equiv ( y == 0 ) \)

No predicates about \( x \)

© 2006 Carnegie Mellon University
Imprecision due to Predicate Abstraction

Counterexamples generated by model checking the abstract model may be spurious, i.e., not concretely realizable.

Need to refine the abstraction iteratively by changing the set of predicates.

Can infer new set of predicates by analyzing the spurious counterexample:

- Lot of research in doing this effectively
- Counterexample Guided Abstraction Refinement (CEGAR)
- A.K.A. Iterative Abstraction Refinement
- A.K.A. Iterative Refinement
Model Checking

\[ \phi = G(\neg \text{ERROR}) \]

\[ P \equiv (y == 0) \]
Model Checking

\[
\phi = G(\neg \text{ERROR})
\]

\[
P \equiv (y = 0)
\]
Model Checking

if (x)

\[ y = x \]

assert (y)

\[ P \equiv (y == 0) \]
Counterexample Validation

- Simulate counterexample symbolically
- Call SAT Checker to determine if the post-condition is satisfiable
- In our case, Counterexample is spurious
- New set of predicates \{x==0,y==0\}
Counterexample Validation

if (x)

yes

y = x

assert (y)

SAT Checker Query:

- \( x \neq 0 \)
- \( y' = x \)
- \( y' = 0 \)

SAT Checker Answer:

UNSAT and here’s an UNSAT core

\{x \neq 0, y' = x, y' = 0\}

• Used to derive new predicate (x=0)
• Different heuristics used in practice
Predicate Abstraction: 2\textsuperscript{nd} Iteration

```
if (x)
  no
  y = x + 1
  assert (y)
  \neg P \neg Q
  x \neq 0 y \neq 0

  yes
  y = x
  \neg P Q

ERROR
P \equiv (x == 0) Q \equiv (y == 0)
```

```
x = 0 y = 0
P Q
```

ERROR
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\begin{itemize}
  \item if \((x)\)
    \begin{itemize}
      \item no
      \begin{itemize}
        \item \(y = x + 1\)
      \end{itemize}
      \end{itemize}
    \end{itemize}
  \begin{itemize}
    \item yes
    \begin{itemize}
      \item \(y = x\)
    \end{itemize}
  \end{itemize}

\begin{itemize}
  \item assert \((y)\)
  \begin{itemize}
    \item \(-P\)
    \item \(-Q\)
  \end{itemize}
\end{itemize}

\[x \neq 0 \land y \neq 0\]

\[P \equiv (x == 0)\]
\[Q \equiv (y == 0)\]

\[x = 0 \land y = 0\]
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\textbf{if} (x)

\begin{align*}
\text{no} & \\
\text{y} & = x + 1 \\
& \text{assert} (y) \\
& x \neq 0 \ y \neq 0
\end{align*}

\begin{align*}
\text{yes} & \\
\text{y} & = x \\
& \text{ERROR}
\end{align*}

\[ \neg P \ \neg Q \]

\[ \neg P \ Q \]

\[ P \ Q \]

\[ P \equiv (x == 0) \ Q \equiv (y == 0) \]

\[ X = 0 \ y = 0 \]
Predicate Abstraction: 2nd Iteration

if (x)

no

y = x + 1

assert (y)

x \neq 0 \ y \neq 0

yes

y = x

ERROR

\neg P \neg Q

\neg P \ Q

ERROR

P \equiv ( x == 0 ) \ Q \equiv ( y == 0 )

X = 0 \ y = 0

\neg P \ Q

P \neg Q

P \ Q
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\begin{itemize}
\item \textbf{if (x)}
\item \textbf{no} \quad \textbf{y = x + 1}
\item \textbf{yes} \quad \textbf{y = x}
\item \textbf{assert (y)}
\item \textbf{x \neq 0 \ y \neq 0}
\end{itemize}

\textbf{ERROR}

\textbf{x = 0 \ y = 0}

\textbf{P \equiv (x \equiv 0)}\quad \textbf{Q \equiv (y \equiv 0)}

\begin{itemize}
\item \textbf{\neg P \ \neg Q}
\item \textbf{\neg P \ Q}
\item \textbf{P \ \neg Q}
\item \textbf{P \ Q}
\end{itemize}
Model Checking: 2\textsuperscript{nd} Iteration

\[ \phi = G(\neg \text{ERROR}) \]

\[ P \equiv (x == 0) \quad Q \equiv (y == 0) \]
Iterative Refinement: Summary

Choose an initial set of predicate, and proceed iteratively as follows:

1. **Abstraction**: Construct an abstract model $M$ of the program using the predicate abstraction

2. **Verification**: Model check $M$. If model checking succeeds, exit with success. Otherwise, get counterexample $CE$.

3. **Validation**: Check $CE$ for validity. If $CE$ is valid, exit with failure.

4. **Refinement**: Otherwise, update the set of predicates and repeat from Step 1.
Iterative Refinement

Program → Predicate Abstraction

Initial Predicates → Better Predicates

Predicate Abstraction → Abstract Model

Abstract Model → Model Checking

Model Checking → System OK

SAT Checker → Candidate Counterexample

Candidate Counterexample → Counterexample Valid?

Counterexample Valid? → Problem Found

Predicate Refinement → Better Predicates

Better Predicates → Predicate Refinement

Predicate Refinement → Abstract Model

Abstract Model → Model Checking

Model Checking → System OK

System OK → Yes

Yes → Localization Reduction, Kurshan, Bell Labs

Localization Reduction, Kurshan, Bell Labs → Counterexample-guided Abstraction Refinement for Symbolic Model Checking, Clarke et al., CMU

Counterexample-guided Abstraction Refinement for Symbolic Model Checking, Clarke et al., CMU → Software Model Checking, SLAM Project, Microsoft, Ball & Rajamani

Software Model Checking, SLAM Project, Microsoft, Ball & Rajamani → Problem Found
Predicate Abstraction: Optimizations

1. Construct transitions on-the-fly
2. Different set of predicates at different control locations

```
if (x)
  no
  P ≡ ( x == 0 )

  y = x + 1
  Q ≡ ( y == 0 )

  y = x

  P ≡ ( x == 0 )

yes

assert (y)
```

3. Avoid exponential number of theorem-prover calls
Research Areas

Finding “good” predicates

- Technically as hard as finding “good” loop invariants
- Complexity is linear in LOC but exponential in number of predicates

Combining with static analysis

- Alias analysis, invariant detection, constant propagation
- Inexpensive, and may make subsequent model checking more efficient

Bounded model checking
Software Model Checking Tools

Iterative Refinement

- SLAM, BLAST, MAGIC, Copper, …

Bounded Model Checking

- CBMC, …

Others

- Engines: MOPED, BEBOP, BOPPO, …
- Java: Java PathFiner, Bandera, BOGOR, …
- C: CMC, …
Bibliography

**Predicate Abstraction:** Construction of abstract state graphs with PVS, S. Graf, H. Saidi, Proceedings of Computer Aided Verification (CAV), 1997

**Abstraction Refinement for C:** Automatically Validating Temporal Safety Properties of Interfaces, T. Ball, S. Rajamani, Proceedings of the SPIN Workshop, 2001

**Software Model Checking Technology Transfer:** SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft, T. Ball, B. Cook, V. Levin, S. Rajamani, Proceedings of Integrated Formal Methods, 2004