Software Verification using Predicate Abstraction and Iterative Refinement: Part 1

15-414 Bug Catching: Automated Program Verification and Testing

Sagar Chaki
November 28, 2011
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)

Discovered independently by Clarke & Emerson and Queille & Sifakis in the early 1980’s
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

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

Program: Syntax  Control Flow Graph  Model: Semantics

Infinite State
Existential 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

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

Existential Abstraction is conservative [ClarkeGrumbergLong94]

- If a ACTL* property holds on the abstraction, it also holds on the program
  - LTL is a subset of ACTL*
- However, the converse is not true: a property that fails on the abstraction may still hold on the program
- Existential abstraction can be viewed as a form of abstract interpretation

Strong & sometimes not computable

Weak: computable
Example of Existential Abstraction

Abstract Initial State

Concrete Initial State

Concrete State

Abstract State

Concrete Transition

Abstract Transition

Abstractly Reachable but Concretely Unreachable

Abstractly and Concretely Unreachable

Abstractly Unreachable
Example of Existential Abstraction
Predicate Abstraction

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

\[ y = x + 1 \]

if (x)

\[ y = x \]

no yes

assert (y)

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

States where \( y \neq 0 \)

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

States where \( y = 0 \)

\[ \neg P \]

ERROR

Software Verification
Sagar Chaki, March 16, 2011
© 2006 Carnegie Mellon University
Predicate Abstraction

\[
y = x + 1\\
\text{if (x)}\\
\begin{cases}
  \text{no} & y = x + 1 \\
  \text{yes} & y = x
\end{cases}
\]

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

\[
\neg P
\]

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

States where \( y \neq 0 \)

States where \( y = 0 \)

Call SAT Checker

ERROR
Predicate Abstraction

\[
y = x + 1
\]

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

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

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

SAT Checker Answer:
SAT and here’s a solution
\[
x=0, y=1, x'=0, y'=1
\]

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

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

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

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

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

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

States where \( y \neq 0 \)

assert (y)

if (x)

no

y = x + 1

yes

y = x

x=1 y=1

x=1 y=2

\neg P
Predicate Abstraction

if \( x \)

no \[ y = x + 1 \]

yes \[ y = x \]

assert (y)

States where \( y \neq 0 \)

SAT Checker Query:
\[
\begin{align*}
& y \neq 0 \land \\
& x' = x \land \\
& y' = x \land \\
& y' \neq 0
\end{align*}
\]

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

assert (y)

States where \( y \neq 0 \)

\( \neg P \)

States where \( y = 0 \)

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

No predicates about x

ERROR

© 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

yes

assert (y)

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

\( \neg P \)

ERROR

\( P \)
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:

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

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

\begin{align*}
\text{if (x)} & \quad \text{no} \quad \text{yes} \\
y = x + 1 & \quad y = x \\
\text{assert (y)} & \quad \neg P \quad \neg Q \\
x \neq 0 \quad y \neq 0 & \\
\neg P \quad Q & \\
P \equiv (x == 0) \quad Q \equiv (y == 0) & \\
X = 0 \quad y = 0 &
\end{align*}
Predicate Abstraction: 2\textsuperscript{nd} Iteration

if (x)

no

y = x + 1

assert (y)

\neg P \neg Q

\neg Q\neg P

x \neq 0 y \neq 0

x = 0 y = 0

y = x

yes

ERROR

P \equiv (x \equiv 0)

Q \equiv (y \equiv 0)

P \quad \neg Q

P \quad Q

ERROR

Software Verification
Sagar Chaki, March 16, 2011
© 2006 Carnegie Mellon University
Predicate Abstraction: 2\textsuperscript{nd} Iteration

if (x)  

no:  
y = x + 1  
assert (y)  
x ≠ 0 \ y ≠ 0

yes:  
y = x  

ERROR

¬P \ ¬Q  

¬P \ Q  
P \ ¬Q  
P \ Q

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

X = 0 \ y = 0

Software Verification
Sagar Chaki, March 16, 2011
© 2006 Carnegie Mellon University
Predicate Abstraction: 2nd Iteration

if (x)

no

y = x + 1
y = x

assert (y)

x ≠ 0 y ≠ 0

yes

¬P ¬Q

¬P Q

P ≡ (x == 0) Q ≡ (y == 0)

ERROR

P ¬Q

ERROR

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

if \(x\)

no

\(y = x + 1\)

\(y = x\)

yes

assert (y)

\(\neg P \neg Q\)

\(\neg P Q\)

ERROR

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

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

\(x = 0\)

\(y = 0\)

ERROR
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\begin{align*}
\text{if } (x) & \\
\text{no} & \quad \text{yes} \\
\text{no} & \quad \text{assert } (y) \\
\text{no} & \quad \text{ERROR} \\
\text{X = 0 } & \quad \text{Y = 0} \\
\neg P & \quad \neg Q \\
\neg P & \quad Q \\
\neg P & \quad \neg Q \\
P & \quad Q \\
\neg P & \quad \neg Q \\
P & \quad Q \\
X = 0 & \quad Y = 0
\end{align*}
Model Checking: 2\textsuperscript{nd} Iteration

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

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

SUCCESS

\[ \neg P \quad \neg Q \quad \neg P \quad Q \quad P \quad \neg Q \quad P \quad Q \quad P \quad Q \]
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

Predicate Abstraction → Abstract Model → Model Checking

Model Checking → SAT Checker

SAT Checker → Candidate Counter-example

Candidate Counter-example → Counterexample Valid? → Yes

Counterexample Valid? → Yes → Problem Found

Program → Initial Predicates → Better Predicates

Better Predicates → Predicate Refinement → Predicate Abstraction

Predicate Abstraction → Abstract Model → Model Checking

Model Checking → System OK

System OK → Yes

Localization Reduction, Kurshan, Bell Labs

Counterexample-guided Abstraction Refinement for Symbolic Model Checking, Clarke et al., CMU

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

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

\[
\begin{align*}
P &\equiv (x == 0) \\
\text{if (x)} &\quad \text{no} \\
&\quad \text{yes} \\
y &\equiv x + 1 \\
y &\equiv x \\
Q &\equiv (y == 0) \\
\text{assert (y)}
\end{align*}
\]

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, SATABS, ...

Bounded Model Checking
- CBMC, ...

Others
- Engines: MOPED, BEBOP, BOPPO, ...
- Java: Java PathFiner, Bandera, BOGOR, ...
- C: CMC, CPAChecker, ...
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 Intergrated Formal Methods, 2004