Configurable Software Model Checking CPAchecker

Dirk Beyer
Software Verification

C Program

```c
int main() {
    int a = foo();
    int b = bar(a);
    assert(a == b);
}
```

Verification Tool

**TRUE**
i.e., specification is satisfied

**FALSE**
i.e., bug found

General method:
Create an overapproximation of the program states / compute program invariants
CPAchecker History

- 2002: BLAST with lazy abstraction refinement [9, 27]
- 2003: Multi-threading support [25]
- 2004: Test-case generation, interpolation, spec. lang. [9, 1]
- 2005: Memory safety, predicated lattices [24, 8]
- Maintenance and extensions became extremely difficult because of design choices that were not easy to revert
- 2007: Configurable program analysis [12, 13], CPAchecker was started as complete reimplementation from scratch [14]
CPAchecker History (2)

- 2009: Large-block encoding [2, FMCAD '09]
- 2010: Adjustable-block encoding [15, FMCAD '10]
- 2012: Conditional model checking [10, FSE '12], PredAbs vs. Impact [20, FMCAD '12]
- 2013: Explicit-state MC [16, FASE '13], BDDs [19, STTT '14], precision reuse [17, FSE '13]
- ...

Iterative fixpoint (forward) post computation
Iterative fixpoint (forward) post computation
Software Verification by Model Checking
[23, 31, Clarke/Emerson, Queille/Sifakis 1981]

Iterative fixpoint (forward) post computation

\[ ... \]

\[ \text{Iterative fixpoint (forward) post computation} \]
Iterative fixpoint (forward) post computation
Software Model Checking

\[
Reached, \ Frontier := \{ e_0 \}
\]

\[
\textbf{while} \ Frontier \neq \emptyset \ \textbf{do}
\]

\[
\text{remove } e \text{ from } Frontier
\]

\[
\textbf{for all } e' \in \text{post}(e) \ \textbf{do}
\]

\[
\textbf{if} \ \neg \text{stop}(e', \ Reached) \ \textbf{then}
\]

\[
\text{add } e' \text{ to } Reached, \ Frontier
\]

\[
\textbf{return} \ Reached
\]
Fixpoint computation on the CFG
Software Verification by Data-Flow Analysis

[29, Kildall 1973]

Fixpoint computation on the CFG
Fixpoint computation on the CFG
Software Verification by Data-Flow Analysis

[29, Kildall 1973]

Fixpoint computation on the CFG
Fixpoint computation on the CFG
Software Verification by Data-Flow Analysis

[29, Kildall 1973]

Fixpoint computation on the CFG
Software Verification by Data-Flow Analysis

[29, Kildall 1973]

Fixpoint computation on the CFG
Software Verification by Data-Flow Analysis
[29, Kildall 1973]

Fixpoint computation on the CFG
Software Verification by Data-Flow Analysis
[29, Kildall 1973]

Fixpoint computation on the CFG
Reached, Frontier := \{e_0\}
while Frontier \neq \emptyset do
    remove e from Frontier
    for all e' \in \text{post}(e) do
        e'' := \text{merge}(e', e'')
        if e'' \neq e'' then
            replace e'' in Reached, Frontier by e''
    if \neg \text{stop}(e', \text{Reached}) then
        add e' to Reached, Frontier
return Reached
Configurable Program Analysis

\[ \text{Reached}, \text{Frontier} := \{ e_0 \} \]

\textbf{while} Frontier \neq \emptyset \textbf{do}

\begin{align*}
\text{remove } e \text{ from Frontier} \\
\text{for all } e' \in \text{post}(e) \textbf{ do} \\
\quad \text{for all } e'' \in \text{Reached} \textbf{ do} \\
\quad \quad e''_{\text{new}} := \text{merge}(e', e'') \\
\quad \quad \text{if } e''_{\text{new}} \neq e'' \textbf{ then} \\
\quad \quad \quad \text{replace } e'' \text{ in Reached, Frontier by } e''_{\text{new}} \\
\quad \quad \text{if } \neg \text{stop}(e', \text{Reached}) \textbf{ then} \\
\quad \quad \quad \text{add } e' \text{ to Reached, Frontier} \\
\end{align*}

\textbf{return} \text{ Reached}
Better combination of abstractions
→ Configurable Program Analysis

Unified framework that enables intermediate algorithms
Dynamic Precision Adjustment

Lazy abstraction refinement: [26, Henzinger/Jhala/Majumdar/Sutre POPL '02]

- Different predicates per location and per path
- Incremental analysis instead of restart from scratch after refinement
Better fine tuning of the precision of abstractions
→ Adjustable Precision
[13, Beyer/Henzinger/Theoduloz ASE’08]

Unified framework enables:
▶ switch on and off different analysis, and can
▶ adjust each analysis separately

• Not only refine, also abstract!
Adjustable Block-Encoding

- Handle loop-free blocks of statements at once
- Abstract only between blocks
  (less abstractions, less refinements)

[2, Beyer/Cimatti/Griggio/Keremoglu/Sebastiani FMCAD '09]
[15, Beyer/Keremoglu/Wendler FMCAD '10]
CPAchecker

[14, Beyer/Keremoglu CAV '11]
CPA – Summary

- Unification of several approaches → reduced to their essential properties
- Allow experimentation with new configurations that we could never think of
- Flexible implementation CPAchecker
Framework for Software Verification — current status

- Written in Java
- Open Source: Apache 2.0 License
- ~80 contributors so far from 15 universities/institutions
- 470,000 lines of code
  (300,000 without blank lines and comments)
- Started 2007

https://cpachecker.sosy-lab.org
Input language C (experimental: Java)

Web frontend available:
https://vcloud.sosy-lab.org/cpachecker/webclient/run

Counterexample output with graphs

Benchmarking infrastructure available
(with large cluster of machines)

Cross-platform: Linux, Mac, Windows
Among world’s best software verifiers:
https://sv-comp.sosy-lab.org/2021/results/

Continuous success in competition since 2012
(66 medals: 19x gold, 22x silver, 25x bronze)

Awarded Gödel medal
by Kurt Gödel Society

Used for Linux driver verification
with dozens of real bugs found and fixed in Linux [28, 18]
CAchecker: Concepts

- Included Concepts:
  - CEGAR [21]
  - Interpolation [16, 7]
  - Adjustable-block encoding [15]
  - Conditional model checking [10]
  - Verification witnesses [5, 4]

- Further available analyses:
  - IMPACT algorithm [30, 20, 7]
  - Bounded model checking [22, 7]
  - k-Induction [6, 7]
  - Property-directed reachability [3]
Completely modular, and thus flexible and easily extensible

Every abstract domain is implemented as a "Configurable Program Analysis" (CPA)

E.g., predicate abstraction, explicit-value analysis, intervals, octagon, BDDs, memory graphs, and more

Algorithms are central and implemented only once

Separation of concerns

Combined with Composite pattern
CPAAlgorithm is the core algorithm for reachability analysis / fixpoint iteration.

Other algorithms can be added if desired, e.g.,
- CEGAR
- Double-checking counterexamples
- Sequential combination of analyses
CPAchecker: Architecture

Source Code → Parser & CFA Builder → k-induction Algorithm → CEGAR Algorithm → CPA Algorithm

Spec CPA → Location CPA → Callstack CPA → Predicate CPA

Results
Online at SoSy-Lab VerifierCloud:
https://vcloud.sosy-lab.org/cpachecker/webclient/run

Download for Linux/Windows:
https://cpachecker.sosy-lab.org

- Run scripts/cpa.sh | scripts\cpa.bat
- -default <FILE>
- Windows/Mac need to disable bitprecise analysis:
  - -predicateAnalysis-linear
  - -setprop solver.solver=smtinterpol
  - -setprop analysis.checkCounterexamples=false

Open graphical report in browser: output/*.html

Open .dot files with dotty / xdot (www.graphviz.org/)
Model Checkers check only what you specified

CPAchecker's default:
- Label ERROR
- Calling functionᵣ _assert_fail()
- assert(pred) needs to be pre-processed

SV-COMP:
- Calling functionᵣ _VERIFIER_error() / reach_error()
- -spec sv-comp-reachability
Want to implement your own analysis?

▶ Easy, just write a CPA in Java
▶ Implementations for 10 interfaces needed
▶ But for 8, we have default implementations
  → Minimal configuration:
    abstract state and
    abstract post operator
The CPA framework is flexible:

- Many components are provided as CPAs:
  - Location / program counter tracking
  - Callstack tracking
  - Specification input (as automata)
  - Pointer-aliasing information

- CPAs can be combined, so your analysis doesn’t need to care about these things
References I


References IV


