Block Abstraction Memoization with Copy-On-Write Refinement

Karlheinz Friedberger

LMU Munich, Germany

05.09.2017
Introduction
Basic Problem with Software Verification

Problem:
- computation of abstract state space \textit{at once} is expensive
Introduction

Basic Problem with Software Verification

Problem:
- computation of abstract state space \textit{at once} is expensive

Approaches of BAM:
- split into smaller problems and solve them separately
- use a \textit{cache} for intermediate results
Introduction

Basic Problem with Software Verification

Problem:
▶ computation of abstract state space \textit{at once} is expensive

Approaches of BAM:
▶ split into smaller problems and solve them separately
▶ use a \textit{cache} for intermediate results

Benefits of BAM:
▶ implemented as top-level CPA
▶ independent of sub-analysis (PA, VA, IA,... and combinations)
▶ modular approach: optimization and heuristics
Introduction
Basics of BAM: Structure and Components

CFA divided into *blocks*

- functions or loops as block size
- block size defines entry and exit nodes
Introduction
Basics of BAM: Structure and Components

CFA divided into blocks
- functions or loops as block size
- block size defines entry and exit nodes

BAMCPA
- manage the analysis and the cache
- optimize cache access by using a Reducer
Introduction

Basics of BAM: Structure and Components

CFA divided into *blocks*
  - functions or loops as block size
  - block size defines entry and exit nodes

BAMCPA
  - manage the analysis and the cache
  - optimize cache access by using a *Reducer*

Combine with other components:
  - CEGAR: specialized refinement (over several ARGs)
  - Exporter: ARG & Graphml
Introduction

Overview of the CPAchecker Framework

Source Code ➔ Parser & CFA Builder ➔ CEGAR Algorithm ➔ Results

CPA Algorithm ➔ sub-analysis

BAM CPA

Spec ➔ Spec CPA

Location CPA ➔ Callstack CPA ➔ Predicate CPA

Karlheinz Friedberger

LMU Munich, Germany

4 / 15
CEGAR with Lazy Refinement

- spurious error path found $\rightarrow$ refinement procedure
  $\rightarrow$ determines a new precision and a cutpoint
  $\rightarrow$ only a "minimal" part of the ARG is remove
CEGAR with Lazy Refinement

- spurious error path found → refinement procedure
  → determines a new precision and a cutpoint
  → only a "minimal" part of the ARG is remove

BAM Refinement

- determine precision and cutpoint over several nested ARGs
- depends only on underlying analysis
- refine the "minimal" set of ARGs
- several heuristics:
  - refine one, all, or some ARGs along error-path
  - merge precisions from different sources
CEGAR with Lazy Refinement

Default state space exploration in BAM with refinement, refinement applied with an *in-place* update of the ARG
CEGAR with Lazy Refinement

Default state space exploration in BAM with refinement, refinement applied with an *in-place* update of the ARG.
CEGAR with Lazy Refinement

Default state space exploration in BAM with refinement, refinement applied with an *in-place* update of the ARG.
Problem: Repeated Counterexamples

What is a *repeated counterexample*?

- an error path cannot be excluded from repeated exploration
- cycles of error paths (and refinements)
  → no progress in CEGAR
Problem: Repeated Counterexamples

Observation

▶ problem mostly appears with "big" programs, e.g. with many blocks and several refinements
▶ small changes in programs cause large differences in runtime of BAM
Problem: Repeated Counterexamples

Observation

► problem mostly appears with "big" programs, e.g. with many blocks and several refinements
► small changes in programs cause large differences in runtime of BAM

Manual analysis shows possible reasons

► deleting block abstractions (holes in the ARG)
► imprecise caching (aggressive caching) → heuristics
► imprecise reducer (Predicate Analysis) → heuristics
Problem: Repeated Counterexamples

The old Approach

And after the refinement?

- start exploration again
- when accessing a missing block,
  recomputing it or use another block abstraction from cache
Problem: Repeated Counterexamples

The old Approach

And after the refinement?

- start exploration again
- when accessing a missing block, recompute it or use another block abstraction from cache

Problem?

- interfering with other refinements
  - precision for a missing block?
  - re-compute nested blocks or take from cache?
- exporting incomplete data (witnesses, ARGs, statistics)
Problem: Repeated Counterexamples

The old Approach

And after the refinement?

► start exploration again
► when accessing a missing block,
  recompute it or use another block abstraction from cache

Problem?

► interfering with other refinements
  → precision for a missing block?
  → re-compute nested blocks or take from cache?
► exporting incomplete data (witnesses, ARGs, statistics)

Idea: do not delete computed block abstractions
Improved Refinement Strategy

Use *Copy-on-Write* for Updates of the ARG
Improved Refinement Strategy

Use *Copy-on-Write* for Updates of the ARG
Improved Refinement Strategy

Use Copy-on-Write for Updates of the ARG
Improved Refinement Strategy
Use *Copy-on-Write* for Updates of the ARG

Computational overhead?
  - old approach: removing a subtree needs $O(N)$ time
  - new approach: copying a subtree needs $O(N)$ time
  - only small increase in memory consumption:
    $\rightarrow$ *flat copy* of ARG states
Improved Refinement Strategy

Use *Copy-on-Write* for Updates of the ARG

Computational overhead?

- old approach: removing a subtree needs $O(N)$ time
- new approach: copying a subtree needs $O(N)$ time
- only small increase in memory consumption:  
  $\rightarrow$ flat copy of ARG states

More benefits

- no need to re-computeate deletes blocks
- *all* information available at end of analysis
- immutable ARGs (after finished sub-analysis)
Evaluation ($\leq 1$ refinements)

Runtime of refinement approaches of BAM with predicate analysis

*tasks with up to one refinement* → no difference expected!
Evaluation (>1 refinements)
Runtime of refinement approaches of BAM with predicate analysis

tasks with more than one refinement

CPU time (s)

n-th fastest result

- copy-on-write
- in-place
Evaluation (\(\leq 1\) and \(> 1\) refinements combined)

Runtime of refinement approaches of BAM with predicate analysis

![Graph showing CPU time vs. n-th fastest result for copy-on-write and in-place methods.]
Conclusion

Current status:

- works for most tasks
- slower on some tasks, faster on more tasks
- PA benefits most, VA only on some files

Future work:

- some heuristics might no longer be beneficial
- new: choose from several cache-entries for the same key?
- merge into trunk, maybe soon :-)

Karlheinz Friedberger
LMU Munich, Germany
Conclusion

Current status:

▶ works for most tasks
▶ slower on some tasks, faster on more tasks
▶ PA benefits most, VA only on some files

Future work:

▶ some heuristics might no longer be beneficial
▶ new: choose from several cache-entries for the same key?
▶ merge into trunk, maybe soon :-)