# Domain-Independent Interprocedural Program Analysis using Block-Abstraction Memoization

**Dirk Beyer** and Karlheinz Friedberger

LMU Munich, Germany

Proc. ESEC/FSE 2020
doi:10.1145/3368089.3409718
Presented at ESEC/FSE 2022

# Software Verification

C Program

```
void main() {
  uint a = nondet();
  uint b = nondet(a);
  uint s = sum(a, b);
  if (s != a + b) {
    error();
}}
```

Verification Tool

TRUE

specification is satisfied

FALSE

bug found

# Software Verification

C Program

```
void main() {
  uint a = nondet();
  uint b = nondet(a);
  uint s = sum(a, b);
  if (s != a + b) {
    error();
}}
```

Verification Tool

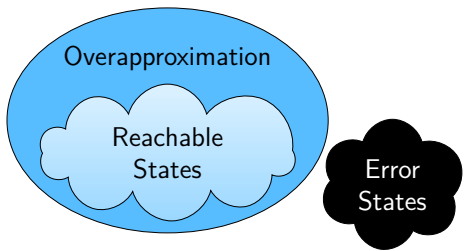
TRUE

specification is satisfied

FALSE

bug found

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

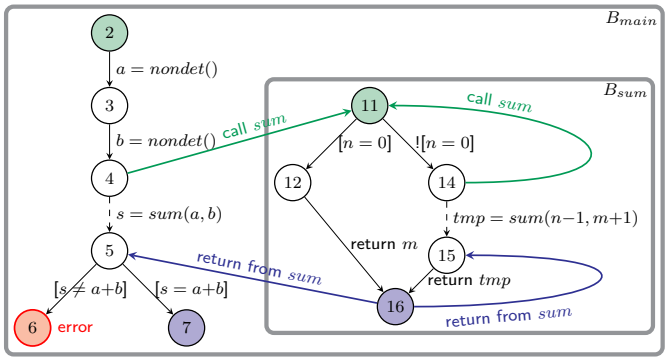


# Example of an Analysis

```
1  void main(void) {
2    uint a = nondet();
3    uint b = nondet();
4    uint s = sum(a, b);
5    if (s != a + b) {
6      error ();
7    }
8  }
9
10 uint sum(uint n, uint m) {
11   if (n == 0) {
12     return m;
13   } else {
14     uint tmp = sum(n - 1, m + 1);
15     return tmp;
16   }
17 }
```

# Example of an Analysis

```
1  void main(void) {
2    uint a = nondet();
3    uint b = nondet();
4    uint s = sum(a, b);
5    if (s != a + b) {
6      error ();
7    }
8  }
9
10 uint sum(uint n, uint m) {
11   if (n == 0) {
12     return m;
13   } else {
14     uint tmp = sum(n − 1, m + 1);
15     return tmp;
16   }
17 }
```




control-flow automaton

# Configurable Program Analysis (CPA)

[1, Beyer/Henzinger/Théoduloz, 2007]

CPA algorithm explores the abstract state space and defines operators for each specific domain

- *transfer*: successor computation
- *merge*: combination of two abstract states
- *stop*: coverage of abstract states

# Configurable Program Analysis (CPA)

[1, Beyer/Henzinger/Théoduloz, 2007]

CPA algorithm explores the abstract state space and defines operators for each specific domain

- *transfer*: successor computation
- *merge*: combination of two abstract states
- *stop*: coverage of abstract states

✓ Independent of used domain: explicit values, intervals, or predicates

# Configurable Program Analysis (CPA)

[1, Beyer/Henzinger/Théoduloz, 2007]

CPA algorithm explores the abstract state space
and defines operators for each specific domain

- *transfer*: successor computation
- *merge*: combination of two abstract states
- *stop*: coverage of abstract states

✓ Independent of used domain: explicit values, intervals, or predicates

X Not using block summaries: one analysis for the whole program

# Block-Abstraction Memoization (BAM)

[2, Wonisch/Wehrheim, 2012]

- split large verification task into smaller problems and solve them separately
- use CPA algorithm for a domain-specific analysis
- cache intermediate analysis results

# Block-Abstraction Memoization (BAM)

[2, Wonisch/Wehrheim, 2012]

- split large verification task into smaller problems and solve them separately
- use CPA algorithm for a domain-specific analysis
- cache intermediate analysis results

- ✓ Independent of domain-specific analysis
- ✓ Simple way of reusing intermediate results

# Block-Abstraction Memoization (BAM)

[2, Wonisch/Wehrheim, 2012]

- split large verification task into smaller problems and solve them separately
- use CPA algorithm for a domain-specific analysis
- cache intermediate analysis results

✓ Independent of domain-specific analysis

✓ Simple way of reusing intermediate results

x Not interprocedural:
  - context relevant for block abstractions
  - colliding variable names from different procedure scopes

# Interprocedural Block-Abstraction Memoization

- based on BAM Intraprocedural

# Interprocedural Block-Abstraction Memoization

- based on BAM Intraprocedural

Operators defined for each specific domain

- *reduce*: abstraction at block entry
- *expand*: concretization at block exit
- *rebuild*: restore context information

# Interprocedural Block-Abstraction Memoization

- based on BAM Intraprocedural

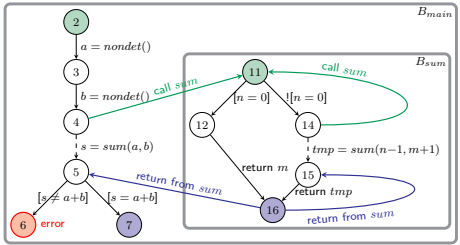
Operators defined for each specific domain

- *reduce*: abstraction at block entry
- *expand*: concretization at block exit
- *rebuild*: restore context information

Fixed-point algorithm:

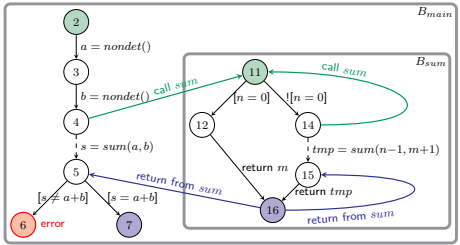
- sound overapproximation of the recursive procedure

# Example of an Analysis

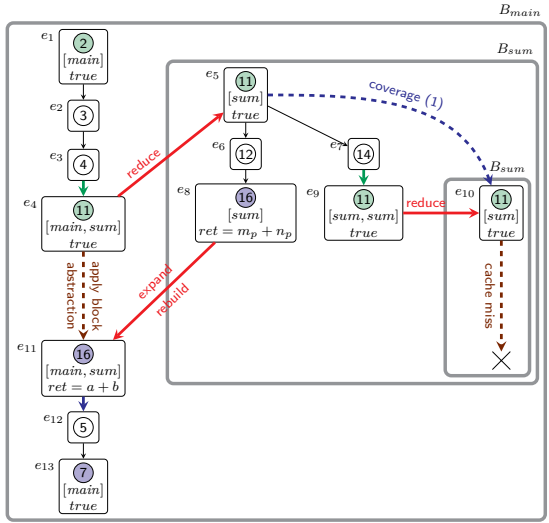



control-flow automaton

# Example of an Analysis



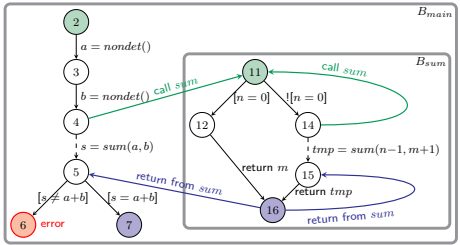

control-flow automaton




abstract reachability graph

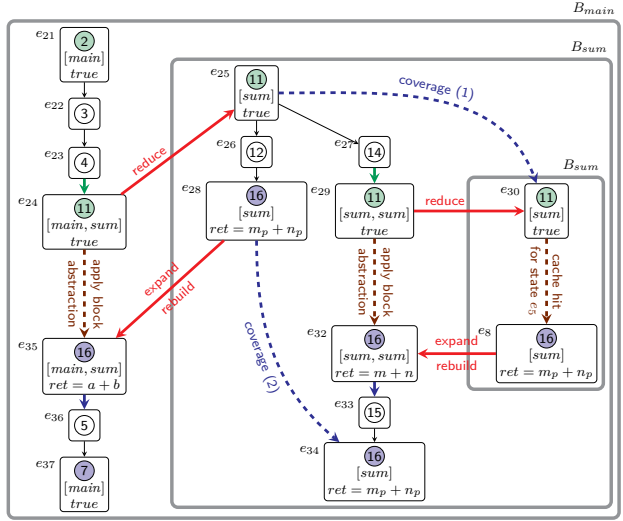
Fixed-point algorithm
(first iteration)

# Example of an Analysis




control-flow automaton

Fixed-point algorithm
(second iteration)




abstract reachability graph

# Contribution

- formalization of BAM Interprocedural

# Contribution

- formalization of BAM Interprocedural
- support for several abstract domains
  - value, predicate, interval domain
  - combination of domains

# Contribution

- formalization of BAM Interprocedural
- support for several abstract domains
  - value, predicate, interval domain
  - combination of domains
- combination with other approaches
  - CEGAR
  - witness validation

# Contribution

- formalization of BAM Interprocedural
- support for several abstract domains
  - value, predicate, interval domain
  - combination of domains
- combination with other approaches
  - CEGAR
  - witness validation
- implementation in the open-source framework CPAchecker

# Evaluation: CPAchecker vs. Tools of SV-COMP 2020

| Verifier | CPU time (s) | Proofs | Bugs |
|---|---|---|---|
| CBMC | 662 | 32 | 47 |
| CPACHECKER (Value+Predicate) | 2 180 | 37 | 46 |
| DIVINE | 1 190 | 32 | 42 |
| ESBMC | 941 | 33 | 47 |
| MAP2CHECK | 23 600 | 34 | 37 |
| PINAKA | 237 | 31 | 31 |
| SYMBIOTIC | 138 | 33 | 45 |
| UAUTOMIZER | 2 160 | 41 | 37 |
| VERIABS | 7 630 | 41 | 46 |

# Conclusion

- modular domain-independent interprocedural analysis

# Conclusion

- modular domain-independent interprocedural analysis
- based on an intraprocedural analysis

# Conclusion

- modular domain-independent interprocedural analysis
- based on an intraprocedural analysis
- support for recursive tasks

# Conclusion

- modular domain-independent interprocedural analysis
- based on an intraprocedural analysis
- support for recursive tasks
- negligible overhead

# Conclusion

- modular domain-independent interprocedural analysis
- based on an intraprocedural analysis
- support for recursive tasks
- negligible overhead
- competitive performance on a large set of benchmarks

# References

- Domain-Independent Interprocedural Program Analysis using Block-Abstraction Memoization
  Dirk Beyer and Karlheinz Friedberger, Proc. ESEC/FSE 2020
  doi:10.1145/3368089.3409718

- Reproduction Package:
  doi:10.5281/zenodo.4024268


- CPAchecker
  https://cpachecker.sosy-lab.org/

# References I


[1] Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable software verification: Concretizing the convergence of model checking and program analysis. In: Proc. CAV. pp. 504–518. LNCS 4590, Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_51

[2] Wonisch, D., Wehrheim, H.: Predicate analysis with block-abstraction memoization. In: Proc. ICFEM. pp. 332–347. LNCS 7635, Springer (2012). https://doi.org/10.1007/978-3-642-34281-3_24