# Goblint: Abstract Interpretation for Memory Safety and Termination

Simmo Saan[1] Julian Erhard[2,3] **Michael Schwarz**[2]
Stanimir Bozhilov[2] Karoliine Holter[1] Sarah Tilscher[2,3]
Vesal Vojdani[1] Helmut Seidl[2]

[1]University of Tartu [2]TUM [3]LMU
`m.schwarz@tum.de`

TACAS 2024

# Goblint in 4 bullet points

- Static analyzer for C programs
  - Based on abstract interpretation — sound!
  - Overapproximating — no violations!
  - Specializes in multi-threaded programs — best in *NoDataRace*!

# SV-COMP 2024: Branching out

Termination + MemorySafety

# SV-COMP 2024: Branching out

Termination + MemorySafety

# Termination

- Two sources for non-termination
    - Infinite loops
    - Recursion

# Termination

- Two sources for non-termination
    - Infinite loops
        - Syntactically instrument loops with a counter
        - Prove counter is bounded
        - Leverage existing numerical domains
    - Recursion

# Termination

- Two sources for non-termination
  - Infinite loops
    - Syntactically instrument loops with a counter
    - Prove counter is bounded
    - Leverage existing numerical domains
  - Recursion
    - Goblint already computes call graph including dynamic calls
    - Check this context-sensitive call graph for cycles

# Termination

- Two sources for non-termination
  - Infinite loops
    - Syntactically instrument loops with a counter
    - Prove counter is bounded
    - Leverage existing numerical domains
  - Recursion
    - Goblint already computes call graph including dynamic calls
    - Check this context-sensitive call graph for cycles

Result

A basic termination analysis in a few hundred lines of code.

# SV-COMP24: Branching out

Termination + MemorySafety

# SV-COMP24: Branching out

Termination + MemorySafety

# SV-COMP24: Branching out

Termination + MemorySafety

# MemorySafety

- Points-to analysis ✓
- Identify heap objects by allocation sites & counters[1] and compute uniqueness ✓

[1] Nod to Tomáš Dacík who added counters to GOBLINT during his exchange

# MemorySafety

- Points-to analysis ✓
- Identify heap objects by allocation sites & counters[1] and compute uniqueness ✓

- Use-After-Free
- . . .
- . . .

[1] Nod to Tomáš Dacík who added counters to GOBLINT during his exchange

# MemorySafety

- Points-to analysis ✓
- Identify heap objects by allocation sites & counters[1] and compute uniqueness ✓

- Use-After-Free in a multi-threaded setting
- ...
- ...

[1] Nod to Tomáš Dacík who added counters to GOBLINT during his exchange

# Use-After-Free

Check at each access to `h` that no calls to `free(h)`

- have happened before

# Use-After-Free

Check at each access to `h` that no calls to `free(h)`

- have happened before from any thread

# Use-After-Free

Check at each access to `h` that no calls to `free(h)`

- have happened before from any thread

Reason about behavior of multiple threads.

# Use-After-Free

Check at each access to `h` that no calls to `free(h)`

- have happened before from any thread

Reason about behavior of multiple threads.

**Challenge:** How to check this thread-modularly?

MHP information (for races)

- finite abstractions of reaching traces encoding aspects of the history, e.g., set of joined threads. [S. et al, ESOP '23]

MHP information (for races)

- finite abstractions of reaching traces encoding aspects of the history, e.g., set of joined threads. [S. et al, ESOP '23]

## Approach

- Per abstract heap object:
  - Accumulate MHP information of all `frees` flow-insensitively
  - For an access, check that none of the frees can happen before

MHP information (for races)

- finite abstractions of reaching traces encoding aspects of the history, e.g., set of joined threads. [S. et al, ESOP '23]

## Approach

- Per abstract heap object:
  - Accumulate MHP information of all `frees` flow-insensitively
  - For an access, check that none of the frees can happen before

## Example

- $t_2$ accesses $h$ (MHP: $\top$)
- $t_1$ calls free $h$ (MHP: $t_2$ must-joined)

MHP information (for races)

- finite abstractions of reaching traces encoding aspects of the history, e.g., set of joined threads. [S. et al, ESOP '23]

Approach

- Per abstract heap object:
  - Accumulate MHP information of all `frees` flow-insensitively
  - For an access, check that none of the frees can happen before

Example

- $t_2$ accesses $h$ (MHP: $\top$)
- $t_1$ calls free $h$ (MHP: $t_2$ must-joined)

MHP information (for races)

- finite abstractions of reaching traces encoding aspects of the history, e.g., set of joined threads. [S. et al, ESOP '23]

## Approach

- Per abstract heap object:
  - Accumulate MHP information of all `frees` flow-insensitively
  - For an access, check that none of the frees can happen before

## Example

- $t_2$ accesses $h$ (MHP: $\top$)
- $t_1$ calls free $h$ (MHP: $t_2$ must-joined)
- ✓

# First Attempt

- Poses many exciting challenges, e.g.,
  - more expressive MHP abstractions
  - more expressive (relational) heap domains
  - . . .

# Thank you!

- Support for termination
- Support for memory safety, also for concurrent programs

- Only **sound** tool to support all properties
- Second best score in **ConcurrencySafety-MemSafety** (after DEAGLE)

/goblint/analyzer

# Further reading

Saan, S., Erhard, J., Schwarz, M., Bozhilov, S., Holter, K., Tilscher, S., Vojdani, V., Seidl, H.
Goblint: Abstract Interpretation for Memory Safety and Termination
In: TACAS 2024. pp. 381–386. Springer (2024).
DOI: `https://doi.org/10.1007/978-3-031-57256-2_25`

`https://goblint.in.tum.de`

`https://github.com/goblint/analyzer`