Goblint

Autotuning Thread-Modular Abstract Interpretation

Simmo Saan[1(✉)] Michael Schwarz[2] Julian Erhard[2]
Manuel Pietsch[2] Helmut Seidl[2] Sarah Tilscher[2] Vesal Vojdani[1]

[1]University of Tartu, Tartu, Estonia
simmo.saan@ut.ee

[2]Technische Universität München, Garching, Germany

TACAS 2023

Goblint

- Static analyzer for C programs
  - Based on abstract interpretation — sound!
  - Specializes in multi-threaded programs — best in *NoDataRace*!
- Implemented in OCaml
  - CIL fork for frontend
  - Apron for relational analyses
  - No other dependencies
- MIT license
- Developed by:

Advances since SV-COMP 2021

1. New numeric abstract domains
   - Integer relations with Apron
   - Floating-point intervals
   - Integer congruences
2. Concurrency
   - Suite of non-relational value analyses
   - Novel relational value analysis (ESOP 2023)
   - *May-happen-in-parallel* analysis for data-race detection
3. Loop unrolling

Advances since SV-COMP 2021

1. New numeric abstract domains
   - Integer relations with Apron
   - Floating-point intervals
   - Integer congruences

2. Concurrency
   - Suite of non-relational value analyses
   - Novel relational value analysis (ESOP 2023)
   - *May-happen-in-parallel* analysis for data-race detection

3. Loop unrolling

Advances since SV-COMP 2021

1. New numeric abstract domains
   - Integer relations with Apron
   - Floating-point intervals
   - Integer congruences

2. Concurrency
   - Suite of non-relational value analyses
   - Novel relational value analysis (ESOP 2023)
   - *May-happen-in-parallel* analysis for data-race detection

3. Loop unrolling

Autotuning

- Cheap *syntactic* heuristics
- Increase precision, decrease resource usage

1. Disable concurrency analyses in single-threaded programs
2. Toggle integer domains
   - E.g. presence of congruences, enums
3. Choose octagon variables
4. Choose widening thresholds
   - Interval, octagon
   - Especially useful for flow-insensitive concurrent value invariants
5. Unroll loops
   - Up-to static bounds or tunable unrolled size
   - Prefer allocation, spawning or error function calls

Autotuning

- Cheap *syntactic* heuristics
- Increase precision, decrease resource usage

1. Disable concurrency analyses in single-threaded programs
2. Toggle integer domains
   - E.g. presence of congruences, enums
3. Choose octagon variables
4. Choose widening thresholds
   - Interval, octagon
   - Especially useful for flow-insensitive concurrent value invariants
5. Unroll loops
   - Up to static bounds or feasible unrolled size
   - Prefer allocation, spawning or error function call

Autotuning

- Cheap *syntactic* heuristics
- Increase precision, decrease resource usage

1. Disable concurrency analyses in single-threaded programs
2. Toggle integer domains
   - E.g. presence of congruences, enums
3. Choose octagon variables
4. Choose widening thresholds
   - Interval, octagon
   - Especially useful for flow-insensitive concurrent value invariants
5. Unroll loops
   - Up to static bounds or feasible unrolled size
   - Prefer allocation, spawning or error function call

Autotuning

- Cheap *syntactic* heuristics
- Increase precision, decrease resource usage

1. Disable concurrency analyses in single-threaded programs
2. Toggle integer domains
   - E.g. presence of congruences, enums
3. Choose octagon variables
4. Choose widening thresholds
   - Interval, octagon
   - Especially useful for flow-insensitive concurrent value invariants
5. Unroll loops
   - Up to static bounds or feasible unrolled size
   - Prefer allocation, spawning or error function call

Autotuning

- Cheap *syntactic* heuristics
- Increase precision, decrease resource usage

1. Disable concurrency analyses in single-threaded programs
2. Toggle integer domains
   - E.g. presence of congruences, enums
3. Choose octagon variables
4. Choose widening thresholds
   - Interval, octagon
   - Especially useful for flow-insensitive concurrent value invariants
5. Unroll loops
   - Up to static bounds or feasible unrolled size
   - Prefer allocation, spawning or error function call

Autotuning

- Cheap *syntactic* heuristics
- Increase precision, decrease resource usage

1. Disable concurrency analyses in single-threaded programs
2. Toggle integer domains
   - E.g. presence of congruences, enums
3. Choose octagon variables
4. Choose widening thresholds
   - Interval, octagon
   - Especially useful for flow-insensitive concurrent value invariants
5. Unroll loops
   - Up to static bounds or feasible unrolled size
   - Prefer allocation, spawning or error function call

Impact of autotuning

Based on own preliminary comparative evaluation

Greatest improvements

1. Disabling concurrency analyses
   - Reduced CPU time 16%, memory 4% for *unreach-call*
2. Octagon analysis
   - 104 additional correct results in *NoOverflows*
   - Automatic variable selection better than all

Overall

- Performance improvement canceled out by precision improvement
- More tasks solved at same level of efficiency

Impact of autotuning

Based on own preliminary comparative evaluation

Greatest improvements

1. Disabling concurrency analyses
   - Reduced CPU time 16%, memory 4% for *unreach-call*
2. Octagon analysis
   - 104 additional correct results in *NoOverflows*
   - Automatic variable selection better than all

Overall

- Performance improvement canceled out by precision improvement
- More tasks solved at same level of efficiency

Impact of autotuning

Based on own preliminary comparative evaluation

Greatest improvements

1. Disabling concurrency analyses
   - Reduced CPU time 16%, memory 4% for *unreach-call*
2. Octagon analysis
   - 104 additional correct results in *NoOverflows*
   - Automatic variable selection better than all

Overall

- Performance improvement canceled out by precision improvement
- More tasks solved at same level of efficiency

Impact of autotuning

Based on own preliminary comparative evaluation

Greatest improvements

1. Disabling concurrency analyses
   - Reduced CPU time 16%, memory 4% for *unreach-call*
2. Octagon analysis
   - 104 additional correct results in *NoOverflows*
   - Automatic variable selection better than all

Overall

- Performance improvement canceled out by precision improvement
- More tasks solved at same level of efficiency

Further reading

Saan, S., Schwarz, M., Erhard, J., Pietsch, M., Seidl, H., Tilscher, S., Vojdani, V.
GOBLINT: Autotuning Thread-Modular Abstract Interpretation

https://goblint.in.tum.de

https://github.com/goblint/analyzer