Correctness Witnesses for Concurrent Programs
Bridging the Semantic Divide with Ghosts

2nd April 2025, VeWit 2025

Frank Schüssele Julian Erhard Manuel Bentele Matthias Heizmann Dominik Klumpp
Simmo Saan Michael Schwarz Helmut Seidl Sarah Tilscher Vesal Vojdani

Motivation

- Witness format for sequential programs (loop invariants, location invariants)[1]
- Goal: Format extension for concurrent programs
- Naive idea: location invariants

[1] Paulína Ayaziová et al. **"Software Verification Witnesses 2.0".** In: *SPIN*. vol. 14624. 2024.

Example

```
1  int used = 0;
2  mutex a;

3  main:
4    create(t1);
5    int y = 7;
6    lock(a);
7    assert(used == 0);
8    unlock(a);

9  t1:
10   lock(a);
11   used = nondet();
12   used = 0;
13   unlock(a);
```

Location invariant at line 6? Problem: We want to reason about other threads!

Solution: Ghost Variables

- Thread modular invariants with ghost variables + ghost updates
- Established concept[2]
- Suitable for different views of concurrent programs[3]

[2] Susan Owicki and David Gries. **"An Axiomatic Proof Technique for Parallel Programs I".** In: *Acta Informatica* 6 (1976), pp. 319–340.
[3] Julian Erhard et al. **"Correctness Witnesses for Concurrent Programs: Bridging the Semantic Divide with Ghosts".** In: *VMCAI (1)*. Vol. 15529. 2025.

Definition: Ghost Witness

A *ghost witness* for a program P is given by:

- $\mathcal{G}_{\text{👻}}$: Global ghost variables
- $\mathcal{L}_{\text{👻}}$: Local ghost variables
- $\mathbb{U}$: Updates of ghost variables associated to statements
- $\mathbb{I}$: Invariants associated to locations.

Example ghost witness W

```
1  int used = 0;
2  mutex a;

3  main:
4    create(t1);
5    int y = 7;
6    lock(a);
7    assert(used == 0);
8    unlock(a);

9  t1:
10   lock(a);
11   used = nondet();
12   used = 0;
13   unlock(a);
```

Goal: *"If mutex `a` is not locked, then `used==0`."*

- $\mathcal{G}_{👻} = \{👻 = 0\}$, $\mathcal{L}_{👻} = \emptyset$
- $\mathbb{U} = \{\texttt{lock(a)} \mapsto 👻 = 1,\ \texttt{unlock(a)} \mapsto 👻 = 0\}$
- $\mathbb{I} = \{\ell_6 \mapsto 👻 == 0 \implies \mathit{used} == 0\}$

Example of witness-instrumented program $P^W_{\text{👻}}$

```
int used = 0;
mutex a;
int 👻 = 0;

main:
  create(t1);
  atomic{ assert(👻 == 0 ⟹ used == 0); }
  int y = 7;
  atomic{ lock(a); 👻 = 1; }
  assert(used == 0);
  atomic{ unlock(a); 👻 = 0; }
```

```
t1:
  atomic{ lock(a); 👻 = 1; }
  used = nondet();
  used = 0;
  atomic{ unlock(a); 👻 = 0; }
```

Definition: Validity

A witness W for a program P is *valid*.

$\iff$

The witness-instrumented program $P^W_{\text{👻}}$ is safe.

Witnesses format for C programs

We extend the witness 2.0 format[a] — which is used by the Software Verification Competition.

```
- entry_type: ghost_instrumentation
    metadata: ...
    content:
    ghost_variables:
        - name: 👻
          type: int
          scope: global
          initial:
              value: 0
              format: c_expression
    ghost_updates:
        - location: { line: 4, ... }
          updates:
              - variable: 👻
                value: 1
                format: c_expression
```

[a] Paulína Ayaziová et al. **"Software Verification Witnesses 2.0".** In: *SPIN*. vol. 14624. 2024.

Semantics

Invariants:

- Atomic evaluation
- Only need to hold at well-defined points (*sequence points*)

Ghost Variables:

- Initialization: beginning of the program

Ghost Updates:

- Cannot assume that the whole statement at given location is atomic (e.g., may contain function call)
- Location for ghost updates restricted to specific statements (assignment or `pthread` library function call)
- Ghost update atomically with action after evaluating the expressions (i.e., arguments for function call / right-hand side of assignment)

Evaluation: Setup

- *ConcurrencySafety-Main* category of SV-COMP
- Witness Generation: GOBLINT[4] (thread modular abstract interpretation) with two analyses (protection, mutex-meet)
- Witness Validation: ULTIMATE GEMCUTTER[5] (model checking using commutativity on interleaving)

[4] Michael Schwarz et al. **"Improving Thread-Modular Abstract Interpretation".** In: *SAS*. vol. 12913. 2021.

[5] Azadeh Farzan, Dominik Klumpp, and Andreas Podelski. **"Sound sequentialization for concurrent program verification".** In: *PLDI*. ACM, 2022.

Evaluation: Witness confirmation

witnesses	protection-👻	mutexmeet-👻
witnesses for correct programs (348)		
total	181	217
confirmed	171	193
rejected	0	0
out of resources	5	19
witnesses for incorrect programs (337)		
total	282	288
confirmed	192	164
rejected	0	0
out of resources	85	117

Evaluation: Witness validation runtime

witnesses	protection-👻 (181)	mutexmeet-👻 (217)
validation	1:31:33	1:29:27
verification	1:07:27	1:18:29

How to add tool support?

Witness Generation:

- Interleaving: encode program counters[6]
  - Optimization: only necessary interleaving info
- Many more possibilities beyond encoding interleaving (e.g., mutex encoding)

Witness Validation:

- Transformation of program to instrument with ghosts
- Verification of transformed program

[6] Leslie Lamport. **"The 'Hoare Logic' of Concurrent Programs".** In: *Acta Informatica* 14 (1980).

Future Work

- Witness generation in Ultimate
- Witness validation in Goblint
- Further format extensions (*ghost update locations*)

Witness format documentation

https://ultimate-pa.github.io/concurrency-witnesses/