Non-termination Witnesses and Their Validation

Zsófia Ádám, Paulína Ayaziová, Levente Bajczi, Dirk Beyer, Marek Jankola, Marian Lingsch-Rosenfeld, and Jan Strejček

{adamzsofi,levente.bajczi}@edu.bme.hu
{dirk.beyer,marek.jankola,marian.lingsch}@sosy.ifi.lmu.de
{strejcek,xayaziov}@fi.muni.cz

Introduction

Instead of producing an output y for an input x, a certifying algorithm produces as output for x not only y but also a witness w. Witnesses and their validation also exist in the area of automatic software verification, and a large number of tools support verification witnesses [5]. However, there is no format with a clear definition and semantics for witnesses of non-termination. In our work, we close the gap by presenting extension of the witness format in version 2.0 [1] to support program non-termination.

Example

```
1  int main() {
2    int i = nondet();
3    while (i > 0) {
4        if (i != 5) {
5            i = i-1;
6        }
7    }
8    return 0;
9  }
```

```
- entry_type: "violation_sequence"
  metadata: ...
  content:
  - segment:
    - waypoint:
        type: "assumption"
        constraint:
          value: "i == 5"
          format: "c_expression"
        location:
          file_name: Ex02_sim.c
          line: 3
        action: "follow"
  - segment:
    - waypoint:
        type: "branching"
        constraint:
          value: "true"
        location:
          file_name: Ex02_sim.c
          line: 3
        action: "cycle"
```

Validation

- CPAchecker [3] is a verification tool that performs validation by liveness-to-safety reduction [6] and bounded model checking. It uses the assumptions from witness to strengthen the formulas expressing program paths.
- Theta [8] uses the same algorithmic configurations for non-termination validation, as for verification by annotating it's internal CFA with the information from the waypoints of the witness. Furthermore, it is capable of modular validation, which can confirm witnesses faster, but is not able to refute them.
- Witch [2] is violation witness validator based on symbolic execution, using information from the witness to restrict the state-space of the program. We add a liveness-to-safety transformation to support non-termination witnesses.

Syntax and Semantics

The basic building blocks of non-termination witnesses are *waypoints*:

- `type` $\in$ {`branching`, `assumption`, `function_enter`, `function_return`}
- `action` $\in$ {`follow`, `avoid`, `cycle`}
- `location` $\in L$, where L is a set of program locations
- `constraint` on the state-space of the program.

A *segment* then consists of arbitrarily many waypoints with action `avoid` followed by exactly one waypoint with action `follow` or `cycle`.

Part of a program execution *matches* a segment if

- it does not pass any `avoid`-action waypoint of the segment,
- it ends when the `follow`/`cycle` waypoint is first visited, and
- the `follow`/`cycle` waypoint of the segment is passed.

A *non-termination witness* is a sequence $n_1 n_2 \dots n_i c_1 c_2 \dots c_j$ of $i \geq 0$ `follow` segments $n_1, \dots, n_i$ followed by $j > 0$ `cycle` segments $c_1, \dots, c_j$. The witness represents each program execution that can be divided into infinitely many non-empty parts such that, for each k, the k-th execution part matches the k-th segment of the sequence $n_1 n_2 \dots n_i (c_1 c_2 \dots c_j)^\omega$.

Construction

- TransVer [4] is a program-transformation framework that can reduce termination to reachability. We implemented a transformation of a violation witness for the transformed program to a non-termination witness for the original program.
- Theta [8] has a modular architecture, which enables a diverse set of verification algorithms on a wide selection of input formats. Verification of non-termination can be done by a lasso checking CEGAR, bounded techniques with L2S reduction, or by transformation to a CHC problem.
- Symbiotic [7] is a program analyzer utilizing program transformation, slicing, and symbolic execution. We implemented export of non-termination witnesses in the new format.

Trivial Witness

```
- entry_type: "violation_sequence"
  metadata: ...
  content:
  - segment:
    - waypoint:
        type: "assumption"
        constraint:
          value: "1"
          format: "c_expression"
        location:
          file_name: ...
          line: 123
        action: "cycle"
```

Artifact

Results

Validators	EmergenTheta 151 witnesses	Theta 205 witnesses	Thorn 221 witnesses	TransVer-CPAchecker 71 witnesses	TransVer-UAutomizer 73 witnesses	Symbiotic 658 witnesses
CPAchecker	115 / 0	**156** / 0	172 / 0	**69** / 0	**66** / 0	204 / 0
EmergenTheta	62 / 1	37 / 1	61 / 1	39 / 1	41 / 0	32 / 3
Theta	**131** / 1	109 / 2	**177** / 2	39 / 0	28 / 0	158 / 0
Theta-modular	23 / 0	15 / 0	25 / 0	0 / 0	0 / 0	241 / 0
Thorn	56 / 1	36 / 0	61 / 1	39 / 2	37 / 2	43 / 3
Witch	104 / 0	148 / 0	164 / 0	57 / 0	64 / 0	**658** / 0
Total confirmed	143	173	199	71	73	658

References

[1] Ayaziová, P., Beyer, D., Lingsch-Rosenfeld, M., Spiessl, M., Strejček, J.: Software verification witnesses 2.0. In: Proc. SPIN. pp. 184–203. LNCS 14624 (2024)
[2] Ayaziová, P., Strejček, J.: Witch 3: Validation of violation witnesses in the witness format 2.0 (competition contribution). In: Proc. TACAS (3). pp. 341–346. LNCS 14572 (2024)
[3] Baier, D., Beyer, D., Chien, P.C., Jakobs, M.C., Jankola, M., Kettl, M., Lee, N.Z., Lemberger, T., Lingsch-Rosenfeld, M., Wachowitz, H., Wendler, P.: Software verification with CPAchecker 3.0: Tutorial and user guide (extended version). arXiv/CoRR (2024)
[4] Beyer, D., Jankola, M., Lingsch-Rosenfeld, M., Xia, T., Zheng, X.: TransVer: A modular program-transformation framework for reduction to reachability. In: Proc. SPIN. LNCS 15945 (2025)
[5] Beyer, D., Strejček, J.: Improvements in software verification and witness validation: SV-COMP 2025. In: Proc. TACAS (3). pp. 151–186. LNCS 15698 (2025)
[6] Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. In: Proc. FMICS. pp. 160–177. No. 2 in ENTSC 66 (2002)
[7] Jonáš, M., Kumor, K., Novák, J., Sedláček, J., Trtík, M., Zaoral, L., Ayaziová, P., Strejček, J.: Symbiotic 10: Lazy memory initialization and compact symbolic execution (competition contribution). In: Proc. TACAS (3). pp. 406–411. LNCS 14572 (2024)
[8] Telbisz, C., Bajczi, L., Szekeres, D., Vörös, A.: Theta: Various approaches for concurrent program verification (competition contribution). In: Proc. TACAS (3). pp. 260–265. LNCS 15698 (2025)