Ultimate Taipan and Race Detection in Ultimate

Daniel Dietsch, Matthias Heizmann, Dominik Klumpp✉,
Frank Schüssele, Andreas Podelski

University of Freiburg, Germany

SV-COMP 2023

Specifications in Ultimate

- In Ultimate C programs are translated to the intermediate language Boogie

Specifications in Ultimate

- In Ultimate C programs are translated to the intermediate language Boogie
- Different C specifications are encoded as assertions in Boogie, the tools check reachability of those asserts

Specifications in Ultimate

- In Ultimate C programs are translated to the intermediate language Boogie
- Different C specifications are encoded as assertions in Boogie, the tools check reachability of those asserts
- Goal: Encode data-races also as assertions

Data races

A program written in C contains a data race if there are two different thread, s.t.

1. one thread writes to a memory location and the other thread writes to or reads from the same memory location,
2. and at least one of the accesses is not atomic,
3. and neither access *happens-before* the other.

From Data Races to Reachability

- Introduce global boolean variables `race_x` for every global variable x

From Data Races to Reachability

- Introduce global boolean variables `race_x` for every global variable x
- Add statements for these variables in the translation

From Data Races to Reachability

- Introduce global boolean variables race_x for every global variable x
- Add statements for these variables in the translation
  For actions that read x:

```
race_x := true;
<read(x)>
assert race_x == true;
```

From Data Races to Reachability

- Introduce global boolean variables race_x for every global variable x
- Add statements for these variables in the translation

  For actions that read x:

```
race_x := true;
<read(x)>
assert race_x == true;
```

  For actions that write x:

```
havoc tmp; // nondeterministic assignment
race_x := tmp;
<write(x)>
assert race_x == tmp;
```

Atomicity

- For an action *a*, we call the sequence of Boogie statements that results from this wrapping $block(a)$

Atomicity

- For an action a, we call the sequence of Boogie statements that results from this wrapping $block(a)$
- If a is part of an atomic block, then the entire $block(a)$ falls inside that atomic block in the translation

Atomicity

- For an action *a*, we call the sequence of Boogie statements that results from this wrapping $block(a)$
- If *a* is part of an atomic block, then the entire $block(a)$ falls inside that atomic block in the translation
- This way the translation ensures that there are no data-races between two atomic statements

SV-COMP Results

	Tool	Score
1	UGemCutter	151
2	UTaipan	139
3	Goblint	124
4	UAutomizer	120
5	CSeq	39

(a)

	Tool	Score
1	Goblint	1304
2	Deagle	1211
3	Dartagnan	768
4	UAutomizer	756
5	UGemcutter	732
6	UTaipan	612

(b)

Figure: Results of the Ultimate tools in the NoDataRace category in (a) SV-COMP 2022 and (b) SV-COMP 2023