1 Ablauf

13:00 – 13:15 Intro
13:15 – 13:45 pySMT und BMC (Breakout Rooms)
13:45 – 13:55 Besprechung
13:55 – 14:00 Pause
14:00 – 14:10 Einführung CPAs
14:10 – 14:45 CPAs (Breakout Rooms)
14:35 – 14:45 Besprechung CPAs
14:45 – 14:50 Pause
14:50 – 15:20 CPAs (Breakout Rooms)
15:20 – 15:30 Nachbesprechung

1.1 Teambildung

2er-Teams. Wer kann Python?

2 SMT Solvers

Satisfiability modulo theories.

Theories:

- Arrays
- Arithmetic (Integer, Float, Bitvector)
- Undefined functions
- ...

3 Configurable Program Analysis

3.1 Semi-Lattice

Semi-lattice \( \mathcal{E} = (E, \sqsubseteq, \sqcup, \top) \) over elements of a set \( E \), if:

- \( \sqsubseteq \) : \( E \times E \) partial order over \( E \),
- every subset \( M \subseteq E \) has a least upper bound \( e \in E \),
- \( \sqcup : E \times E \to E \) denotes the least upper bound of two elements,
- top element \( \top \) is the least upper bound of \( E \).
3.2 CPAs

A CPA \( \mathcal{D} = (D, \rightsquigarrow, \text{merge}, \text{stop}) \) for a CFA \((L, l_0, G)\) consists of the following components:

- Abstract domain \( D = (C, \mathcal{E}, \llbracket \cdot \rrbracket) \) with concrete states \( C \), semi-lattice \( \mathcal{E} = (E, \sqsubseteq, \sqcup, \top) \), and concretization function \( \llbracket \cdot \rrbracket : E \rightarrow 2^C \)
- Abstract transfer relation \( \rightsquigarrow : E \times G \times E \) assigns to each abstract state \( e \in E \) possible abstract successors \( e' \in E \), labelled with a corresponding CFA edge \( g \in G \).
- Merge operator \( \text{merge} : E \times E \rightarrow E \) combines two abstract states into a new one
- Termination check \( \text{stop} : E \times 2^E \rightarrow \mathbb{B} \) checks whether an abstract state is already covered by a set of given abstract states

Some CPAs you should know:

1. Location CPA
2. Observer Analysis
3. Value Abstraction
4. Predicate Abstraction

3.3 CPA Algorithm

**Algorithm 2 CPA(\( \mathcal{D}, e_0 \))**

**Input:** a CPA \( \mathcal{D} = (D, \rightsquigarrow, \text{merge}, \text{stop}), \)

an initial abstract state \( e_0 \in E \), where \( E \) denotes the set of elements of the lattice of \( D \)

**Output:** a set of reachable abstract states

**Variables:** a set reached \( \subseteq E \), a set waitlist \( \subseteq E \)

1: waitlist := \{e_0\}
2: reached := \{e_0\}
3: while waitlist \( \neq \{\} \) do
4: choose \( e \) from waitlist
5: waitlist := waitlist \( \setminus \{e\} \)
6: for each \( e' \) with \( e \rightsquigarrow e' \) do
7: \hspace{1cm} for each \( e'' \in \text{reached} \) do
8: \hspace{2cm} // combine with existing abstract state
9: \hspace{2cm} \( e_{\text{new}} := \text{merge}(e', e'') \)
10: \hspace{2cm} if \( e_{\text{new}} \neq e'' \) then
11: \hspace{3cm} waitlist := (waitlist \( \cup \{e_{\text{new}}\}) \setminus \{e''\}
12: \hspace{3cm} reached := (reached \( \cup \{e_{\text{new}}\}) \setminus \{e''\}
13: \hspace{1cm} \text{if } \neg \text{stop}(e', \text{reached}) \text{ then}
14: \hspace{2cm} waitlist := waitlist \( \cup \{e'\}\)
15: \hspace{2cm} reached := reached \( \cup \{e'\}\)
16: return reached
3.4 Linear Temporal Logic (LTL)

As a reminder, the syntax of LTL:

Formula $\varphi ::= \text{true} \mid \text{false} \mid A \quad \text{atomic propositions}$

$\mid \neg \varphi \mid \varphi \land \psi \mid \ldots \quad \text{junctors over LTL formulae}$

$\mid \diamond \varphi \quad (N \varphi) \quad \varphi\text{ is true in the next state (/next time step)}$

$\mid \Box \varphi \quad (G \varphi) \quad \varphi\text{ is true all the time, from now on}$

$\mid \phi U \psi \quad \phi\text{ is true until } \psi\text{, and } \psi\text{ must be true at some point}$

$\mid \phi W \psi \quad \phi\text{ is true until } \psi\text{ is true, and } \psi\text{ may always stay false}$

An LTL formula is evaluated over an infinite sequence of steps. In each step, each atomic proposition may change its value.

The definition of a (time) step is arbitrary. In our application, a time step is often defined as one transition in the CFA.

4 Verification-Result Witnesses

https://github.com/sosy-lab/sv-witnesses

4.1 Violation Witnesses

4.2 Correctness Witnesses