Lab: Code Verification and Z3 Theorem Prover

(Week 7)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Quiz-2, Exercise-2 and Assignment-2

- Quiz-2 with 25 questions (5 points), **due date: 23:59 Tuesday, Week 7**
  - Logical formula and predicate logic
  - Z3's knowledge and translation rules
- Lab-Exercise-2 (5 points), **due date: 23:59 Tuesday, Week 7**
  - **Goal:** Manually translate code into z3 formulas/constraints and verify the assertions embedded in the code.
  - **Specification:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-2`
  - **SVF Z3 APIs:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-Z3-API`
- Assignment-2 (25 points) **due date: 23:59 Tuesday, Week 8**
  - **Goal:** automatically perform assertion-based verification for code using static symbolic execution.
  - **Specification:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-2`

Methods to Be Implemented

You need to implement the following four functions in `Assignment-2.cpp`:

- `SSE::reachability`
- `SSE::collectAndTranslatePath`
- `SSE::handleCall`
- `SSE::handleRet`
- `SSE::handleNonBranch`
- `SSE::handleBranch`
- The required implementation parts are indicated with TODO comments and you only need to fill up the code template if a method is partially implemented.

Software Verification Competition (SV-COMP)

Optional for Interested Students.

Cameron McGowan & Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

What is SV-COMP?

- SV-COMP is an annual **software verification competition** held as part of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (**TACAS**).

What is SV-COMP?

- SV-COMP is an annual **software verification competition** held as part of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (**TACAS**).
- The **competition goals** are as follows:
  - Provide a **snapshot** of the state-of-the-art in software verification to the community. This makes it easy to compare the efficacy of different tools for different problems when selecting one to use.
  - Increase the **visibility and credits** that tool developers receive. This encourages the development of verifiers in research and provides a forum for students to share their work.
  - Establish a set of **benchmarks** for software verification in the community. This means that researchers with a new technique can easily compare it to established literature.

What is SV-COMP?

- SV-COMP is an annual **software verification competition** held as part of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (**TACAS**).
- The **competition goals** are as follows:
  - Provide a **snapshot** of the state-of-the-art in software verification to the community. This makes it easy to compare the efficacy of different tools for different problems when selecting one to use.
  - Increase the **visibility and credits** that tool developers receive. This encourages the development of verifiers in research and provides a forum for students to share their work.
  - Establish a set of **benchmarks** for software verification in the community. This means that researchers with a new technique can easily compare it to established literature.
- **Competition website**: `https://sv-comp.sosy-lab.org/`

How does SV-COMP work?

- **Verification tasks**:
  - A verification task consists of a **C program** and a **specification**. A verification run is a **non-interactive execution** of a competition candidate on a single verification task, in order to check if the following statement is correct: “The program satisfies the specification.”

How does SV-COMP work?

- **Verification tasks**:
  - A verification task consists of a **C program** and a **specification**. A verification run is a **non-interactive execution** of a competition candidate on a single verification task, in order to check if the following statement is correct: “The program satisfies the specification.”
  - The **result** of a verification run is a triple **(ANSWER, WITNESS, TIME)**. ANSWER is one of the following outcomes:

TRUE + Witness	The specification is satisfied and a correctness witness is produced.
FALSE + Witness	The specification is violated and and a violation witness is produced.
UNKNOWN	The tool cannot decide the problem or terminates by a tool crash, time-out, or out-of-memory.

How does SV-COMP work?

- **Witnesses**:
  - The witness has to be written to a file witness.graphml or witness.yml, which is given to a **witness validator** to check validity. The result is counted as correct only if **at least one validator** successfully validated it.

How does SV-COMP work?

- **Witnesses**:
  - The witness has to be written to a file witness.graphml or witness.yml, which is given to a **witness validator** to check validity. The result is counted as correct only if **at least one validator** successfully validated it.
  - **Correctness witnesses**: We provide a path of invariants which hold and prove the specification is met.

How does SV-COMP work?

- **Witnesses**:
  - The witness has to be written to a file witness.graphml or witness.yml, which is given to a **witness validator** to check validity. The result is counted as correct only if **at least one validator** successfully validated it.
  - **Correctness witnesses**: We provide a path of invariants which hold and prove the specification is met.
  - **Violation witnesses**: We provide a path of concrete inputs which violate the specification.

How does SV-COMP work?

- **Witnesses**:
  - The witness has to be written to a file witness.graphml or witness.yml, which is given to a **witness validator** to check validity. The result is counted as correct only if **at least one validator** successfully validated it.
  - **Correctness witnesses**: We provide a path of invariants which hold and prove the specification is met.
  - **Violation witnesses**: We provide a path of concrete inputs which violate the specification.
- **Scoring**:

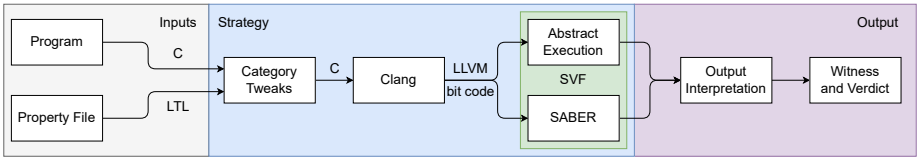
Points	Reported Result	Description
0	UNKNOWN	Failure to compute a verification result.
+1	FALSE correct	Error found violation witness was confirmed.
-16	FALSE incorrect	Error reported for a correct program.
+2	TRUE correct	Correctness reported and validated.
-32	TRUE incorrect	Correctness reported but error was present.

SVF-SVC in SV-COMP 2025

- In the 2025 competition, **SVF-SVC** participated in SV-COMP for the first time.
  - We built a Python wrapper around SVF which translated C files into an appropriate input format for SVF.
  - We used the specification category information to call SVF with the appropriate flags.
  - We interpreted SVF's output to generate witnesses using a basic format.

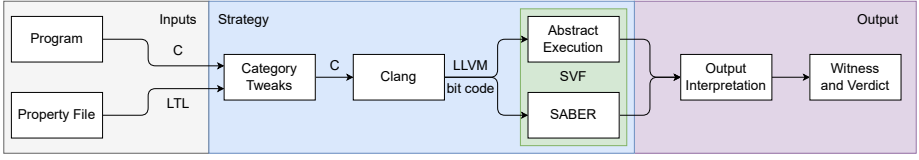
SVF-SVC in SV-COMP 2025

- In the 2025 competition, **SVF-SVC** participated in SV-COMP for the first time.
  - We built a Python wrapper around SVF which translated C files into an appropriate input format for SVF.
  - We used the specification category information to call SVF with the appropriate flags.
  - We interpreted SVF's output to generate witnesses using a basic format.



SVF-SVC in SV-COMP 2025

- In the 2025 competition, **SVF-SVC** participated in SV-COMP for the first time.
  - We built a Python wrapper around SVF which translated C files into an appropriate input format for SVF.
  - We used the specification category information to call SVF with the appropriate flags.
  - We interpreted SVF's output to generate witnesses using a basic format.



- SVF-SVC qualified for the competition and our short tooling paper was published in TACAS 2025:
  `https://link.springer.com/chapter/10.1007/978-3-031-90660-2_21`

SVF-SVC in SV-COMP 2026

- We are looking to participate again, this time with your help!

SVF-SVC in SV-COMP 2026

- We are looking to participate again, this time with your help!
- We are redesigning the SVF-SVC to allow our wrapper to support assignment style inputs and outputs.
- We aim to more extensively support SV-COMP test cases and provide more robust witness formats.

SVF-SVC in SV-COMP 2026

- We are looking to participate again, this time with your help!
- We are redesigning the SVF-SVC to allow our wrapper to support assignment style inputs and outputs.
- We aim to more extensively support SV-COMP test cases and provide more robust witness formats.
- We are looking for groups of students to compete with the SVF-SVC team to:
  - Reduce the number of incorrect assertions rather than UNKNOWN outputs made to address the harsh penalties associated with incorrect results.
  - Expand SVF-SVC to compete in more categories.
  - Better utilise our time given the competition constraints.
  - Optimize or add to existing algorithms to improve overall performance.

SVF-SVC in SV-COMP 2026

- We are looking to participate again, this time with your help!
- We are redesigning the SVF-SVC to allow our wrapper to support assignment style inputs and outputs.
- We aim to more extensively support SV-COMP test cases and provide more robust witness formats.
- We are looking for groups of students to compete with the SVF-SVC team to:
  - Reduce the number of incorrect assertions rather than UNKNOWN outputs made to address the harsh penalties associated with incorrect results.
  - Expand SVF-SVC to compete in more categories.
  - Better utilise our time given the competition constraints.
  - Optimize or add to existing algorithms to improve overall performance.
- **Expected deadlines**:
  - October 2025: Tool registration.
  - November 2025: Final tool submission.
  - December 2025: Paper submission.
  - January 2026: Paper notification and final edits.