Lab: Abstract Interpretation

(Week 9)

Yulei Sui, Mohamad Barbar, Hanyuan Li, Angela He

School of Computer Science and Engineering

University of New South Wales, Australia

Assignment-3 Spec Released

- Remember to `git pull` or `docker pull` to get the code template for **Assignment-3**
- If you are using Python for implementation, please update `pysvf` via
  - `pip install pysvf -U`
  - `pip install -U pysvf --force-reinstall`

Quiz-3 + Lab-Exercise-3 + Assignment-3

- Quiz-3 (5 points) (due date: **23:59, Wednesday, Week 10**)
  - Abstract domain and soundness, widening and narrowing
- Lab-Exercise-3 (5 points) (due date: **23:59, Wednesday, Week 10**)
  - **Goal:** Manually updating abstract states for assertion verification
  - **Specification:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-3`
- Assignment-3 (25 points) (due date: **23:59, Wednesday, Week 11**)
  - **Goal:** Automated abstract interpretation and bug detection.
  - **Specification:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-3`
  - **Three levels of internal tests** (Levels 1-3) for marking (from small to medium to large programs).
  - We provide **only** Levels 1 and 2 public tests for **dry run** and basic tests.
    - Create **your own tests** (note: really large tests may be very memory hungry!).
  - Submit **both Assignment-3.h and Assignment-3.cpp** in C++ implementation

MyExperience Survey

The myExperience survey is open:

`https://go.blueja.io/aEQocdN6j0-MHp2wvUA8KA`

It would mean a lot to **our teaching team** if you could take a moment to leave your feedback/comments, especially if you liked the lectures, tutors, labs, the content and interactions.

Software Verification Competition (SV-COMP)

Optional for Interested Students.

Cameron McGowan & Angela He & 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.

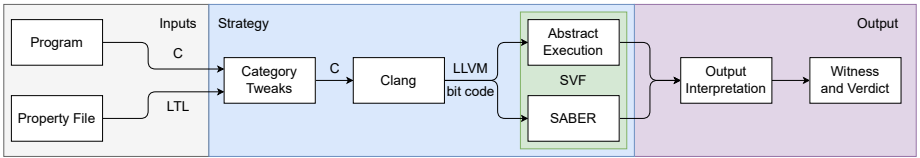
How does SV-COMP work?

- **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.

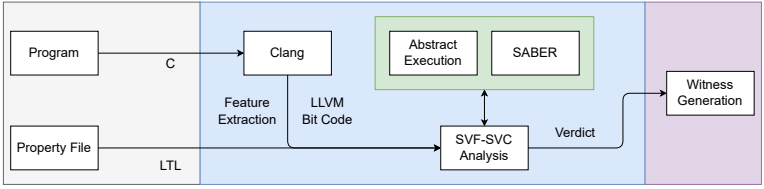
SVF-SVC in SV-COMP 2025

- 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.



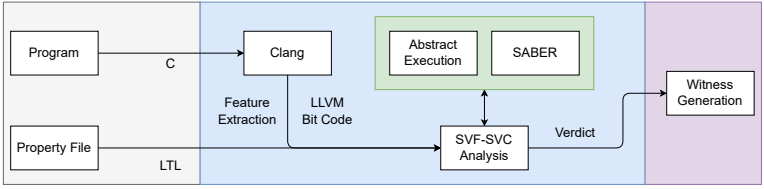
SVF-SVC in SV-COMP 2026

- In the 2026 competition, **SVF-SVC** moved to a white-box architecture built directly on SVF's internal APIs.
  - Instead of manipulating C files and shelling out to SVF, SVF-SVC 2.0 analyzes code directly at the LLVM level.
  - We competed in the ReachSafety, MemSafety and SoftwareSystems categories.



SVF-SVC in SV-COMP 2026

- In the 2026 competition, **SVF-SVC** moved to a white-box architecture built directly on SVF's internal APIs.
  - Instead of manipulating C files and shelling out to SVF, SVF-SVC 2.0 analyzes code directly at the LLVM level.
  - We competed in the ReachSafety, MemSafety and SoftwareSystems categories.



- Our competition results were weaker than 2025's debut, largely due to the cost of the architectural rewrite. Our short tooling paper was submitted, but was not accepted for publication this year.

SVF-SVC in SV-COMP 2027

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

SVF-SVC in SV-COMP 2027

- We are looking to participate again, this time with your help!
- We are stabilising the SVF-SVC revamp 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 2027

- We are looking to participate again, this time with your help!
- We are stabilising the SVF-SVC revamp 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 2027

- We are looking to participate again, this time with your help!
- We are stabilising the SVF-SVC revamp 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 2026: Tool registration.
  - November 2026: Final tool submission.
  - December 2026: Paper submission.
  - January 2027: Paper notification and final edits.

SVF-SVC in SV-COMP 2027

Why should I participate?

- The competition provides an excellent opportunity to apply the skills developed in this course and work on a real-world application.
- The competition allows the opportunity to make a research publication which is greatly beneficial for any future research work or career in academia.
- The competition involves working as part of a team on a software project which looks excellent on your resume when applying for computer science, software engineering or cybersecurity related roles.
- You will improve your ability to program, design algorithms, interface with real-world systems and collaborate and work with a team.

SVF-SVC in SV-COMP 2027

How can I participate?

- Completing COMP6131 will give you all the prerequisite knowledge required for the competition.
- **Expected Timeline**:
  - Term 2 2026: Finish learning the foundations from COMP6131.
  - Term 2-3 break 2026: Organise the team, plan out the project, begin implementation.
  - Term 3: Continue and complete implementation.
  - Summer break 2026-27: Receive competition results.
- We are looking for students who can make some commitment in the Term 2-3 break and/or Term 3.

Competition Format

- **Competition website**: `https://sv-comp.sosy-lab.org/`

Competition Format

- **Competition website**: `https://sv-comp.sosy-lab.org/`
- Let's take a deeper look at the competition categories and properties to check. `https://sv-comp.sosy-lab.org/2027/rules.php`

Competition Format

- **Competition website**: `https://sv-comp.sosy-lab.org/`
- Let's take a deeper look at the competition categories and properties to check. `https://sv-comp.sosy-lab.org/2027/rules.php`
- Let's look at how programs and witnesses work in SV-COMP.
- Example program: `https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/blob/main/c/loop-acceleration/multivar_1-1.c`
- Example witness: `https://gitlab.com/sosy-lab/benchmarking/sv-witnesses/-/blob/main/examples/multivar_1-1.witness-2.0.yml`

Competition Format

- **Competition website**: https://sv-comp.sosy-lab.org/
- Let's take a deeper look at the competition categories and properties to check. https://sv-comp.sosy-lab.org/2027/rules.php
- Let's look at how programs and witnesses work in SV-COMP.
- Example program: https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/blob/main/c/loop-acceleration/multivar_1-1.c
- Example witness: https://gitlab.com/sosy-lab/benchmarking/sv-witnesses/-/blob/main/examples/multivar_1-1.witness-2.0.yml
- **SVF-SVC 2025 Paper**: https://link.springer.com/chapter/10.1007/978-3-031-90660-2_21
- **SVF-SVC 2025 Github**: https://github.com/Lasagnenator/svf-svc-comp

Competition Summary

- Let's take a brief look at our submission for last year: `https://github.com/Lasagnenator/svf-svc-comp`
- **Participate**: Contact us at angela.he@student.unsw.edu.au and/or cameron.mcgowan@unsw.edu.au to participate.

Competition Summary

- Let's take a brief look at our submission for last year: `https://github.com/Lasagnenator/svf-svc-comp`
- **Participate**: Contact us at angela.he@student.unsw.edu.au and/or cameron.mcgowan@unsw.edu.au to participate.
- **Final Q & A**: Please ask any questions about the competition now or email us later.