Automated Software Verification

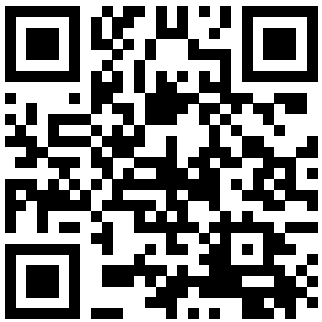
Can Machines Help Us Improve Software Quality?

Simmo Saan Vesal Vojdani

Department of Computer Science
University of Tartu

Digit 2025 (May 9, Tartu)

Get Dev Container to follow along later:
/sws-lab/digit2025-infer

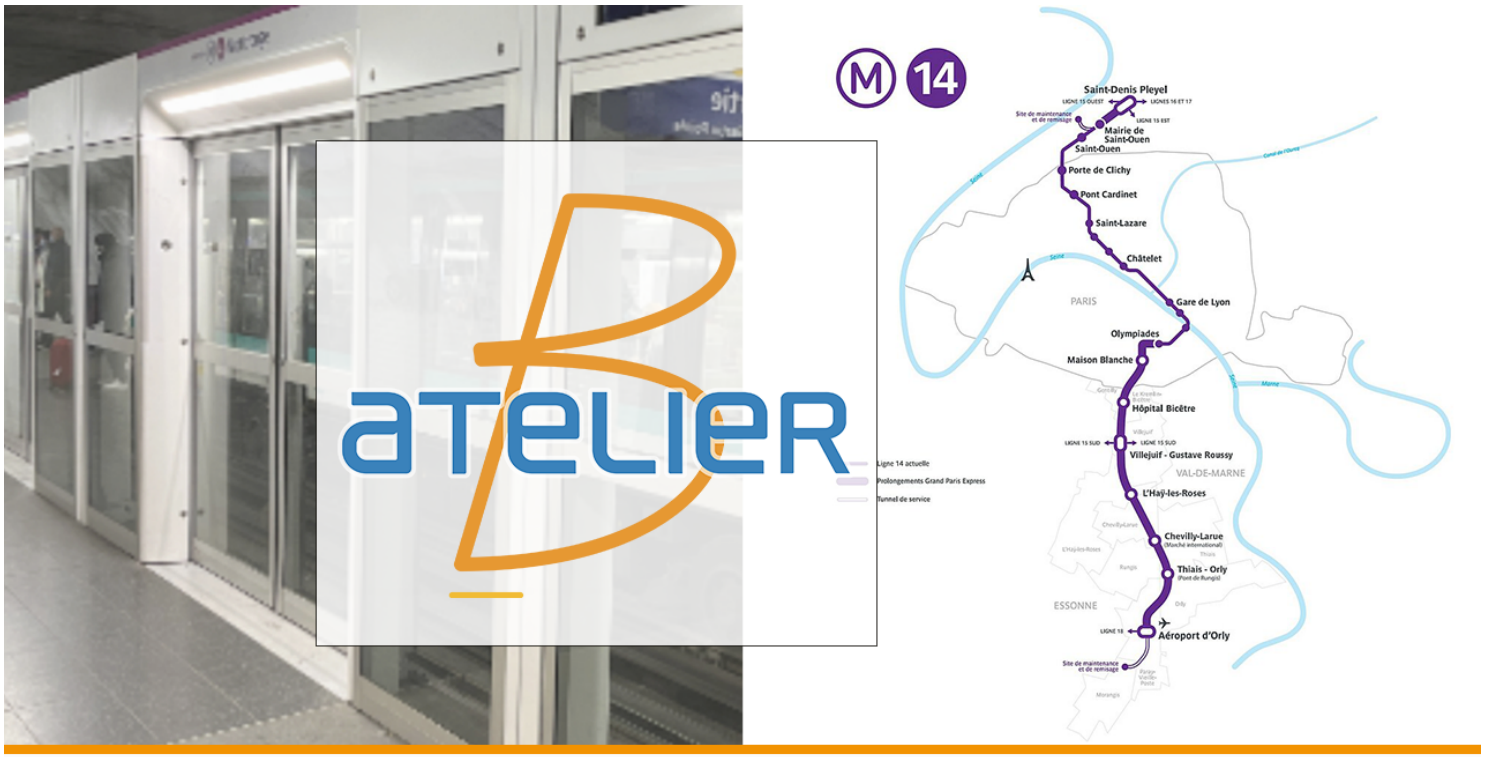


Motivation

- Quality assurance saves money! (Airbus)
- But it is easier to generate code, than to verify it! (AI is currently making this even worse. . .)
- “Closing The Software Understanding Gap”

The Solution: Use Formal Verification!

But. . . that's for safety-critical systems only?!



CLEARSY
Safety Solutions Designer

A Spectrum of (Semi-)Formal Methods

We focus on methods with high levels of automation!

1. How Amazon built Cedar
   Deductive methods in practice.

2. Automated Verification with SV-COMP tools
   State-of-the-art in automated software verification.

3. Scalable Analysis with Facebook Infer
   Introduction to abstract interpretation!

1. Deductive Verification

How Amazon Built Cedar with Verification-Guided Development (VGD)

Amazon Cedar (Dafny & Lean): Verified authorization engine of AWS Verified Permissions.

Project Everest (F*): Verified cryptography library (HACL*/EverCrypt) deployed in Firefox, Windows, Linux kernel, WireGuard.

EverParse (F*): Verified parsers securing every network packet in Azure Hyper-V.

Hoare Triplets (aka Code Contracts)

$$\{\!|\ \phi\ |\!\}\ S\ \{\!|\ \psi\ |\!\}$$

- For each state satisfying pre-condition ϕ,
- if execution reaches the end of S,
- the resulting state satisfies post-condition ψ.

So let's look at simple Dafny example

```
int abs(int i) {
    if (0 <= i)
        r := i;
    else
        r := -i;
}
```

- Prove: for any input ($\mathtt{i} \in \mathbb{Z}$), returns $\mathtt{r} \geqslant 0$.
- Does this also hold for Java's `int`?

How to express this in Java?

abs

```
public static int abs(int a)
```

Returns the absolute value of an `int` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Integer.MIN_VALUE`, the most negative representable `int` value, the result is that same value, which is negative. In contrast, the `absExact(int)` method throws an `ArithmeticException` for this value.

absExact

```
public static int absExact(int a)
```

Returns the mathematical absolute value of an `int` value if it is exactly representable as an `int`, throwing `ArithmeticException` if the result overflows the positive `int` range.

Since the range of two's complement integers is asymmetric with one additional negative value (JLS 4.2.1), the mathematical absolute value of `Integer.MIN_VALUE` overflows the positive `int`

Key Points

- Partial specs suffice to uncover edge cases!
- Central focus of contemporary PL research: making signatures more expressive.

```
method Abs(i: int32) returns (r: int32)
  ensures 0 <= r || r == -0x8000_0000

method AbsExact(i: int32) returns (r: int32)
  requires i != -0x8000_0000
  ensures 0 <= r
```

- Java: `int abs(int a)` and `int absExact(int a)`

Cedar Policy (cedarpolicy.com)

A Language and Evaluation Engine for Access Control Policies

```
permit(
  principal,
  action in [Action::"edit", ...],
  resource
)
when {
  resource.owner == principal.id
};
```

Cedar Authorization Algorithm

1. If any forbid policy evaluates to true, then the final result is `Deny`.
2. Else, if any permit policy evaluates to true, then the final result is `Allow`.
3. Otherwise (i.e., no policy is satisfied), the final result is `Deny`.

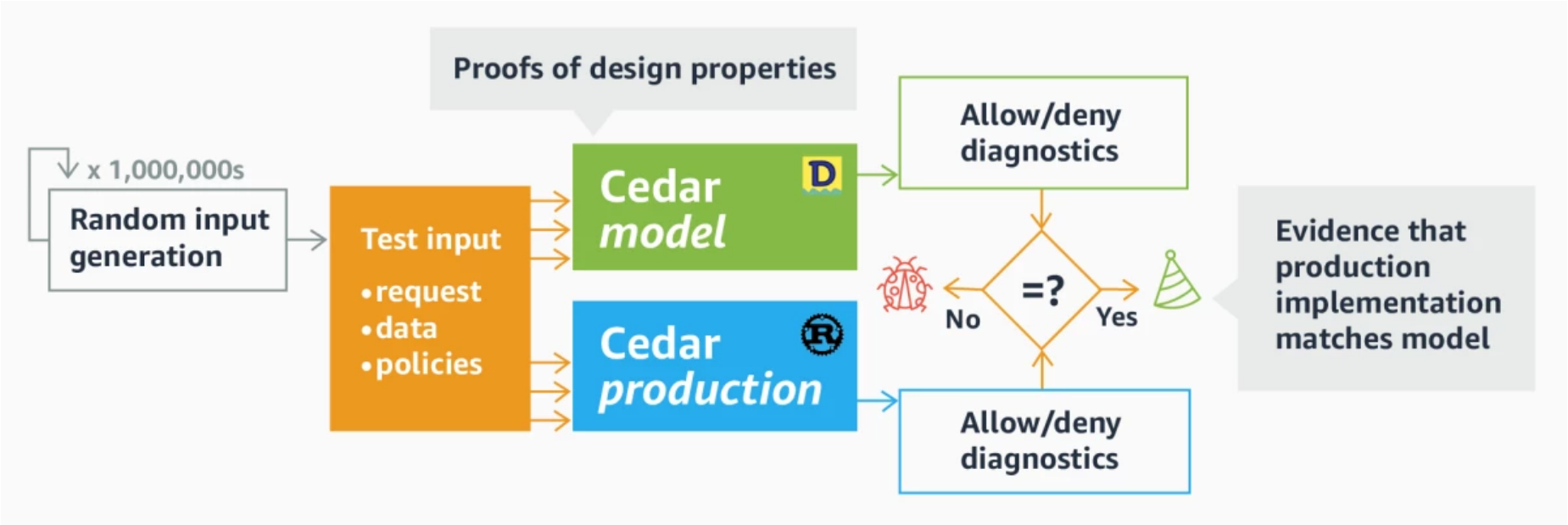
Proving Properties about Cedar

/cedar-policy/cedar-spec/blob/v3.0.1/cedar-dafny/thm/basic.dfy

```
lemma ForbidTrumpsPermit(request: Request, store: Store)
requires // If some forbid policy is satisfied, then
  exists f ::
    f in store.policies.policies.Keys &&
    store.policies.policies[f].effect == Forbid &&
    Authorizer(request, store).satisfied(f)
ensures // the request is denied.
  Authorizer(request, store).isAuthorized().decision == Deny
{
  var f :| f in Authorizer(request, store).forbids();
}
```

Differential Random Testing (DRT)

The Key Component of Verification-Guided Development



- Reference implementation in Dafny (then Lean).
- Production implementation in Rust.
- The model is used as a test oracle for coverage-guided fuzzing.

Was it worth it?

https://www.amazon.science/publications/how-we-built-cedar-a-verification-guided-approach

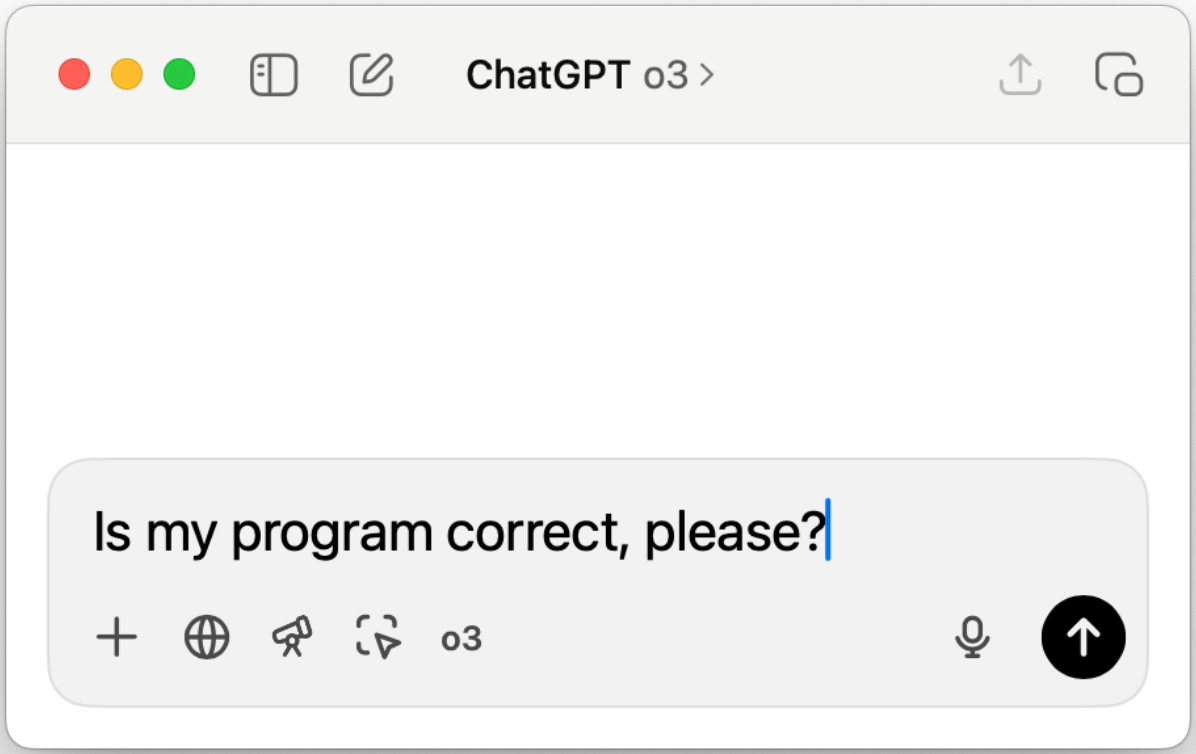
While carrying out proofs, we found and fixed 4 bugs in Cedar's policy validator, and DRT and PBT helped us find and fix 21 additional bugs in various parts of Cedar.

This is practical because...

- They formalized a complicated part of their code.
- The production code is still in a mainstream language.
- The Verification-Guided Development process unveiled many bugs in the code.
- The production team does not depend on esoteric languages and/or knowledge.

2. Automated Software Verification?

(NB! This slide is a joke. In this house, we believe in truthful methods.)



As an aside (if interested): ChatGPT verification.

SV-COMP

International Competition on Software Verification

- Automated tools compete to find bugs and prove correctness of C and Java programs.
- Driver of innovation in verification techniques!
- Largest competition for software verification tools SV-COMP 2025 key statistics:

62 Tools

33 Teams

12 Countries

Goblint

https://goblint.in.tum.de

- Our tool at SV-COMP
- Static analyzer for C programs
- Specializes in concurrency
- Based on abstract interpretation
- Sound (on SV-COMP)
- Implemented in OCaml

Formal Methods Tools Repository

https://fm-tools.sosy-lab.org

- SV-COMP tools are uploaded to Zenodo.
- These are powerful verification engines (but focused on smaller programs).
- You can use these tools! If you extract part of your code, these tools can find real vulnerabilities.
- Our student extracted reported vulnerabilities and attempted to verify the fixes.
  - Flaws in patched versions (e.g., CVE-2025-32776).
  - We are working on improving the extraction process.

The SV-COMP Workflow (Simplified)

1. Verification task:
   - Program – source code
   - Property – memory safety, overflow, data race, memory leak, termination, reachability (assert)
   - Expected verdict – correct (*true*) or incorrect (*false*)

The SV-COMP Workflow (Simplified)

1. Verification task:
   - Program – source code
   - Property – memory safety, overflow, data race, memory leak, termination, reachability (assert)
   - Expected verdict – correct (*true*) or incorrect (*false*)
2. Verifier analyzes:
   - 15 min CPU time, 4 CPU cores, 15 GB RAM

The SV-COMP Workflow (Simplified)

1. Verification task:
   - Program – source code
   - Property – memory safety, overflow, data race, memory leak, termination, reachability (assert)
   - Expected verdict – correct (*true*) or incorrect (*false*)
2. Verifier analyzes:
   - 15 min CPU time, 4 CPU cores, 15 GB RAM
3. Verifier output:
   - Verdict – correct, incorrect or unknown

The SV-COMP Workflow (Simplified)

1. Verification task:
   - Program – source code
   - Property – memory safety, overflow, data race, memory leak, termination, reachability (assert)
   - Expected verdict – correct (*true*) or incorrect (*false*)
2. Verifier analyzes:
   - 15 min CPU time, 4 CPU cores, 15 GB RAM
3. Verifier output:
   - Verdict – correct, incorrect or unknown
4. Verifier scores points if
   - Verdict is the same as expected

3. Abstract Interpretation

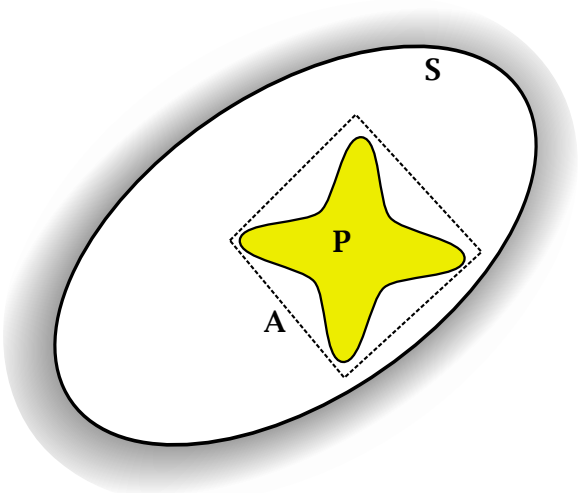
- A theoretical framework for static analysis.
- Simulates program on abstract values.
- Ensuring we cover every possible execution path (soundness).
- Famous Examples:
  - Astrée (ENS & AbsInt)
  - IKOS (NASA)
  - Goblint (TUM & Tartu)
- Used to verify safety-critical software!

Soundness!

Illustration from Antoine Miné (MPRI M2-6: Abstract interpretation)

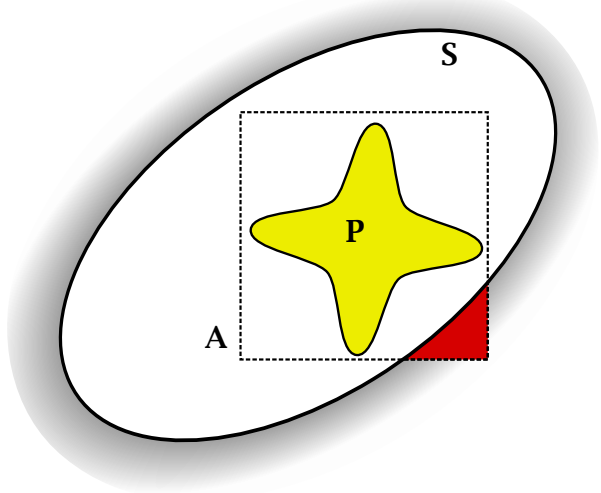
- Goal: program satisfy spec ($\mathrm{P} \subseteq S$).
- Over-approximate P by abstract states $\mathcal{A}$.

If $\mathcal{A} \subseteq S$, program correct:



$A \subseteq S$
program proved

If $\mathcal{A} \not\subseteq \mathrm{P}$, potential bug:



$A \not\subseteq S$
false alarm

Infer: More Practical Usage . . .

Moving Fast with Software Verification

- A framework for modular static analysis (Integrated into CI pipelines at Facebook.)
- It is based on abstract interpretation.
- Occasionally abandons soundness. (Gasp!)

Running Infer

Demo on JFreeChart 1.5.0

- Get Dev Container to follow along:

  /sws-lab/digit2025-infer

- ```
  $ make install
  $ cd examples/jfreechart-1.5.0/
  $ infer run -- mvn compile
  ```
  ```

Running Infer

Demo on JFreeChart 1.5.0

- Get Dev Container to follow along:

  /sws-lab/digit2025-infer

- ```
  $ make install
  $ cd examples/jfreechart-1.5.0/
  $ infer run -- mvn compile
  ```

- Results on JFreeChart 1.5.0:

Found 60 issues

Issue Type(ISSUED_TYPE_ID)	#
Thread Safety Violation(THREAD_SAFETY_VIOLATION)	36
Null Dereference(NULLPTR_DEREFERENCE)	19
Inefficient Keyset Iterator(INEFFICIENT_KEYSET_ITERATOR)	4
Resource Leak(PULSE_RESOURCE_LEAK)	1
  ```

Tutorial: Resource-Counting Domain

- The abstract state is a non-negative integer $\mathfrak{n} \in \mathbb{N}$ (the number of resources currently held).
- The transfer functions for acquire/release are simply increment/decrement.

Resource-Counting: Example 1

```
void basicLeakBad() throws /* ... */ {
  new FileInputStream("file.txt");
} // Leaked 1 resource
```

Resource-Counting: Example 1

```
void basicLeakBad() throws /* ... */ {
  new FileInputStream("file.txt");
} // Leaked 1 resource
```

```
void doubleLeakBad() throws /* ... */ {
  new FileInputStream("file1.txt");
  new FileInputStream("file2.txt");
} // Leaked 2 resources
```

Resource-Counting: Example 1

```
void basicLeakBad() throws /* ... */ {
  new FileInputStream("file.txt");
} // Leaked 1 resource
```

```
void doubleLeakBad() throws /* ... */ {
  new FileInputStream("file1.txt");
  new FileInputStream("file2.txt");
} // Leaked 2 resources
```

```
void basicReleaseOk() throws /* ... */ {
  FileInputStream stream =
      new FileInputStream("file.txt");
  stream.close();
} // Leaked 0 resources
```

Resource-Counting: Example 2

```
void acquireTwoForgetOneBad() throws /* ... */ {
  FileInputStream stream1 =
      new FileInputStream("file.txt");
  FileInputStream stream2 =
      new FileInputStream("file.txt");
  stream1.close();
} // Leaked 1 resource
```

Resource-Counting: Example 2

```
void acquireTwoForgetOneBad() throws /* ... */ {
  FileInputStream stream1 =
      new FileInputStream("file.txt");
  FileInputStream stream2 =
      new FileInputStream("file.txt");
  stream1.close();
} // Leaked 1 resource
```

```
void acquireTwoThenReleaseOk() throws /* ... */ {
  FileInputStream stream1 =
      new FileInputStream("file.txt");
  FileInputStream stream2 =
      new FileInputStream("file.txt");
  stream1.close();
  stream2.close();
} // Leaked 0 resources
```

Tutorial: Resource-Counting Domain

- The abstract state is a non-negative integer $\mathfrak{n} \in \mathbb{N}$ (the number of resources currently held).
- The transfer functions for acquire/release are simply increment/decrement.
- States are ordered numerically: $0 \sqsubseteq 1 \sqsubseteq 2 \sqsubseteq \ldots$.
- Paths are joined by taking the larger: $\mathfrak{n}_1 \sqcup \mathfrak{n}_2 = \max(\mathfrak{n}_1, \mathfrak{n}_2)$.

Resource-Counting: Example 3

```
void mayLeakBad(Boolean b) throws /* ... */ {
  FileInputStream stream;
  if (b) {
    stream = new FileInputStream("file.txt");
  }
} // Leaked 1 resource
```

Resource-Counting: Example 3

```
void mayLeakBad(Boolean b) throws /* ... */ {
  FileInputStream stream;
  if (b) {
    stream = new FileInputStream("file.txt");
  }
} // Leaked 1 resource
```

```
void choiceCloseOk(Boolean b) throws /* ... */ {
  FileInputStream stream =
      new FileInputStream("file.txt");
  if (b) {
    stream.close();
  } else {
    stream.close();
  }
} // Leaked 0 resources
```

Conclusion / Take-Away Message

- You can apply formal verification gracefully without committing to esoteric languages and tools . . .
- May need some guidance: industrial master's thesis with us?
- Get Dev Container for Infer demos:

  /sws-lab/digit2025-infer