SV-LIB 1.0

A Standard Exchange Format for Software-Verification Tasks

Gidon Ernst **Marian Lingsch-Rosenfeld**

Dirk Beyer Martin Jonáš

2025-12-03

LMU Munich, Germany

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

(SV-LIB)

SoSy-Lab Software Systems

VerifyThis

On-site competitions (at ETAPS): hands on, personal exchange

VerifyThis

On-site competitions (at ETAPS): hands on, personal exchange

Great experience! But what if we focus on collaboration? [Huisman+ 20]

VerifyThis

On-site competitions (at ETAPS): hands on, personal exchange

Great experience! But what if we focus on collaboration? [Huisman+ 20]

The **goals of the long-term challenges** are

1. to foster collaboration between researchers and their tools,
2. to demonstrate practical value of formal methods, and
3. to evaluate the capabilities of methods and tools.

The emphasis on collaboration comes with the need and at the same time the opportunity to make progress on long-standing open issues in formal methods [12]

4. to develop approaches that bridge between specification paradigms and
5. to work towards conceptual and technical integration of verification tools.

Discussion over recent years

This event series

- Challenge: Hagrid, Casino, Memcached
- Presentations, discussions, and tutorials
- ...

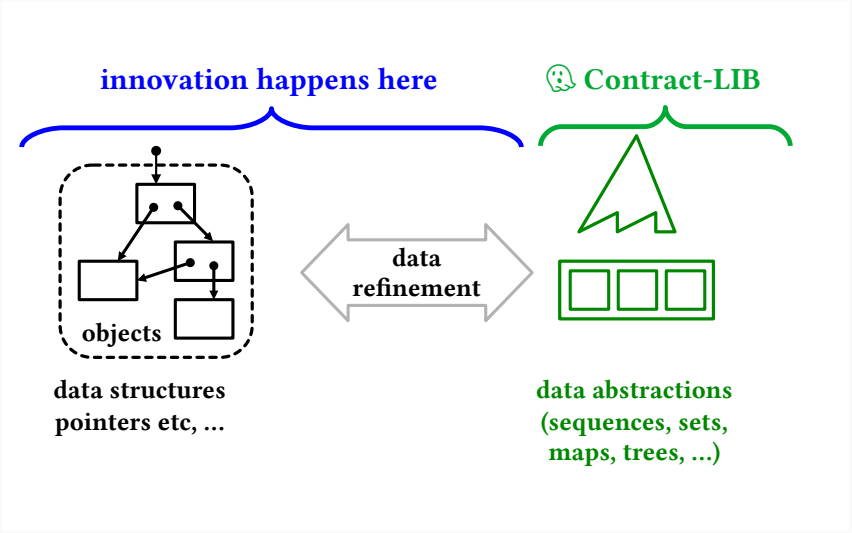
Dagstuhl and Lorentz Seminar(s):

- combine heterogeneous tools and theories
- software contracts vs. system contracts

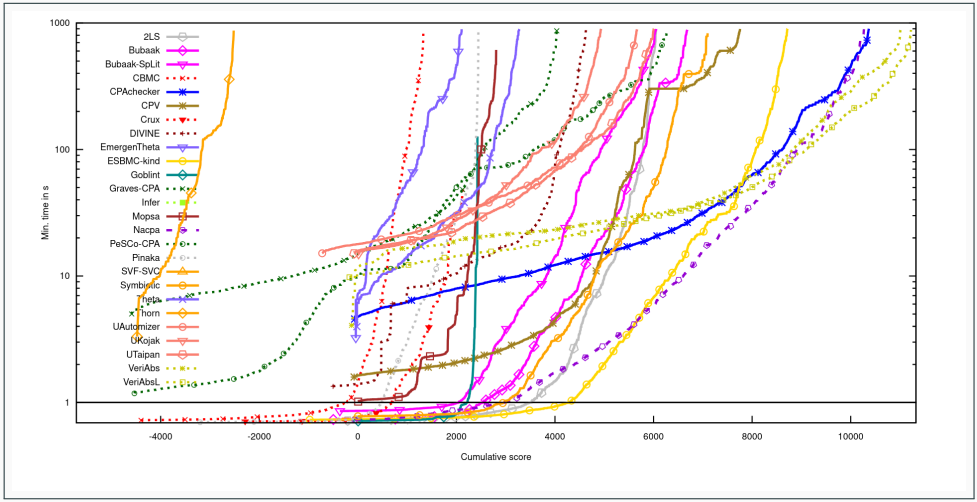
CAV award 2019: Intermediate Verification Languages (Leino, Filliâtre):

- Why3, Boogie, Viper (procedural), MoXI, VMT (state-transition), K2 (hybrid)

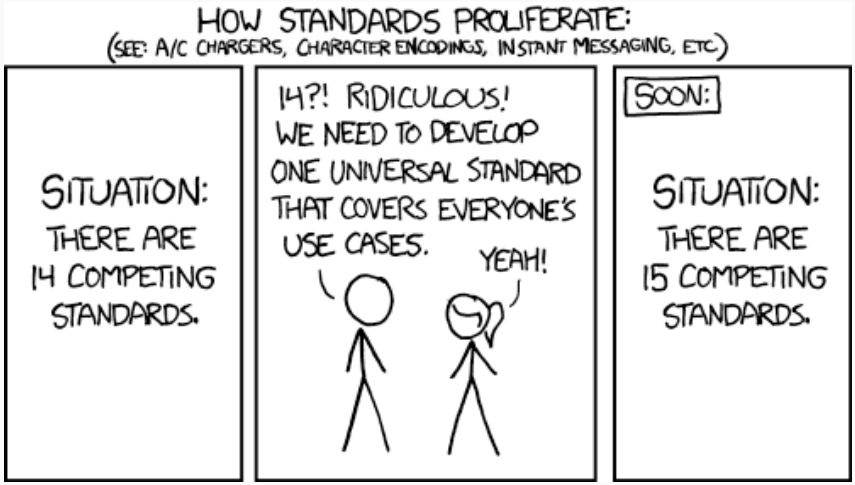
Contract-LIB: Interface Specifications [ISoLA 2024]



SV-COMP: Intl. Competition on Software Verification [Beyer+]



https://sv-comp.sosy-lab.org/2025/results/results-verified/



Taken from xkcd.com/927

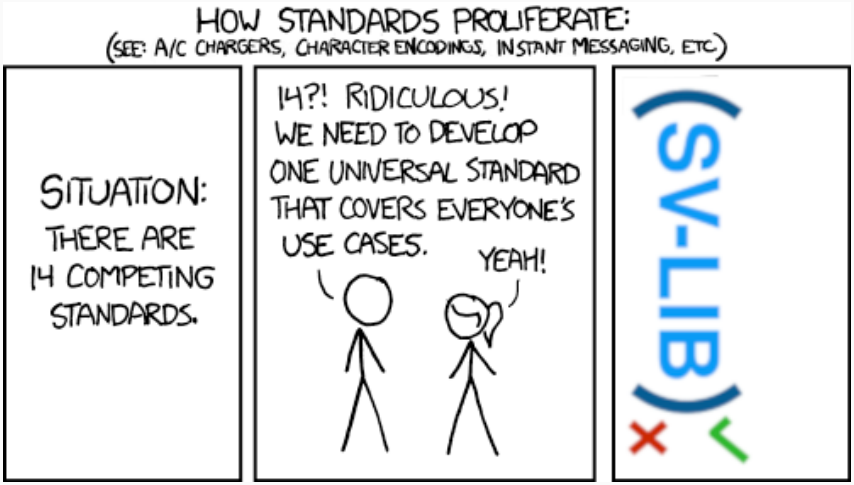
Motivation for another IVL

(a) Challenge in SV-COMP and VerifyThis: Tools need to support C as well as implement the verification methodology

(b) Challenge in SV-COMP: The complexities of C transfer into certification of the correctness of tool output

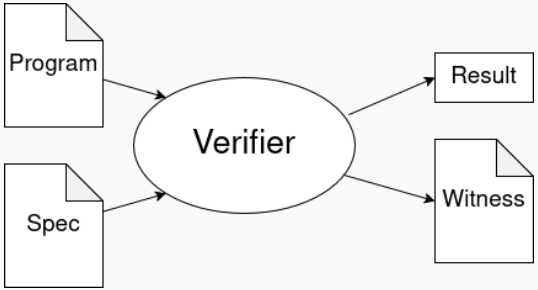
(c) Challenge in SV-COMP: Exchange information between tools (e.g., witnesses, specifications, . . .)

(d) Challenge in SV-COMP and VerifyThis: Bridge the gap between the SV-COMP and VerifyThis communities

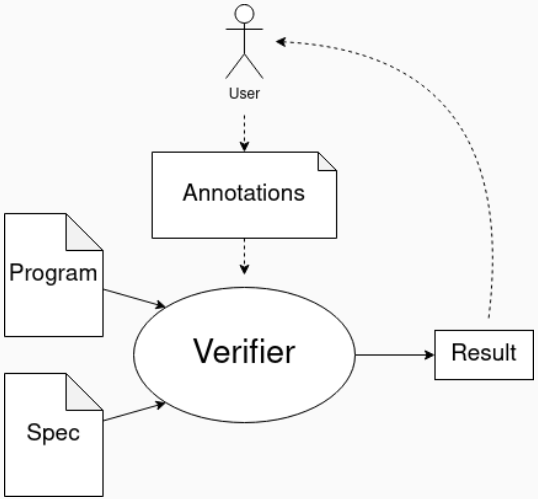


Taken from xkcd.com/927

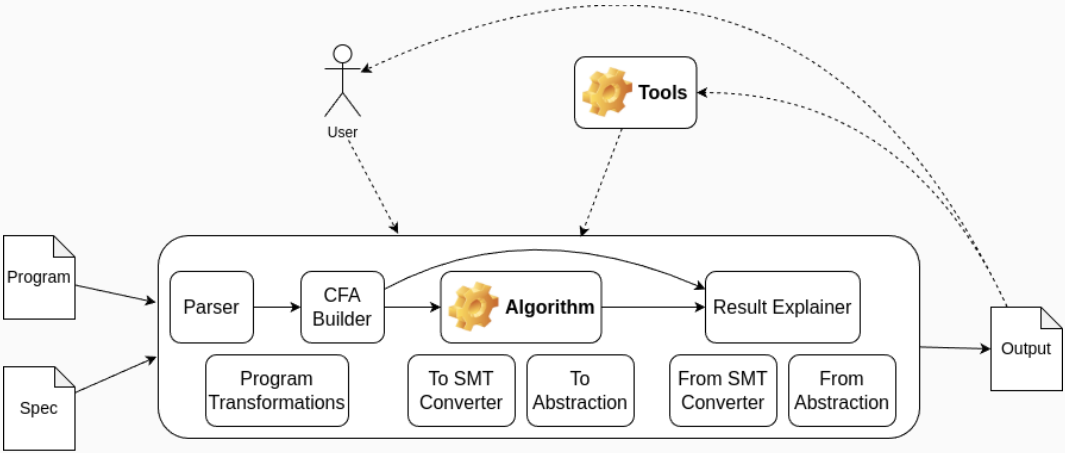
Automatic Software Verification (SV-COMP)



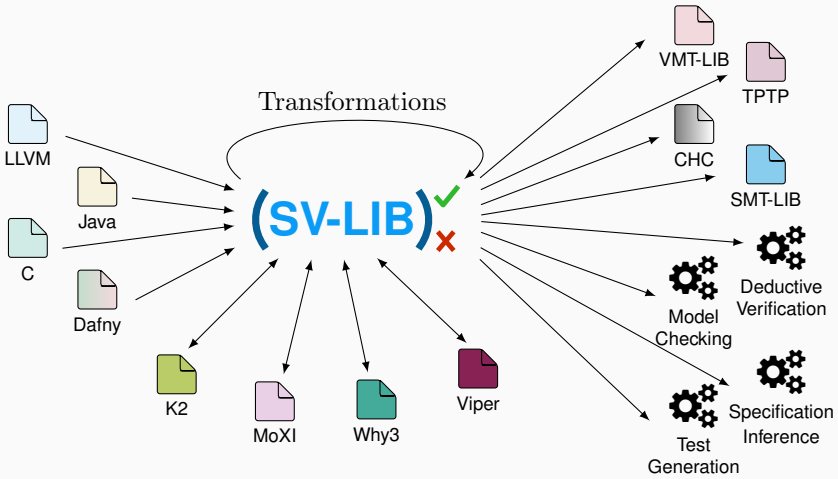
Autoactive Software Verification (VerifyThis)



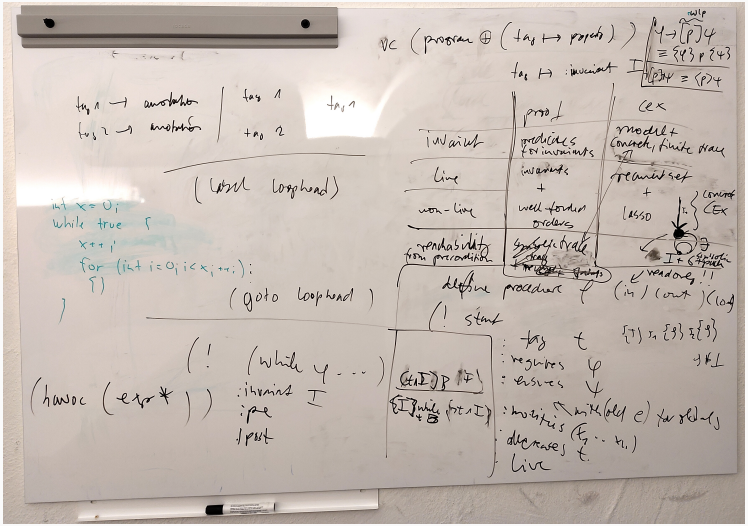
Inside a Verifier



SV-LIB


Transformations
LLVM
Java
C
Dafny
(SV-LIB)
K2
MoXI
Why3
Viper
VMT-LIB
TPTP
CHC
SMT-LIB
Deductive Verification
Model Checking
Specification Inference
Test Generation

SV-LIB: Initial Design



SV-LIB: Initial Design

Syntax of Imperative Commands

Expressions and Sorts: as in SMT-LIB

Commands include global mutable variables and procedure definitions

```
c ::= (declare-var symbol sort)
    |  (define-proc symbol ; name
                   ((symbol sort)*)  ; inputs
                   ((symbol sort)*)  ; outputs
                   ((symbol sort)*)) ; local variables
```

Statements

```
s ::= skip ; (assume true)
    |  (assign ((symbol expr)*)
    |  (havoc  ((symbol)*)
    |  (assume expr)
    |  (event :kw (expr*))
    |  (call symbol (expr*) (symbol*))
    |  (label :kw )  |  (goto :kw )
    |  (if expr s1 s2?)  |  (while expr s)  |  (star s)
    |  (sequential s1 ... sn)
    |  (concurrent s1 ... sn)  |  (atomic s1 ... sn)
    |  (choice s1 ... sn)
```

https://docs.google.com/document/d/1HAcfPubY7oKVtuIDRUZZ44omsKPl0pxThkPGXEjuC3g/edit

SV-LIB: Goals

(a) Consolidate existing IVLs from an engineering perspective, with focus on tools and interchange: take well-understood constructs, avoiding semantic complexity (procedural extension of SMT-LIB)

(b) Keep programs and specifications separable and the latter extensible: achieved by a robust mechanism to refer to and annotate program points

(c) Validation ≈ Verification: have full spectrum of potential witnesses standardized to enable and foster interchange across tools and use-cases

(d) Strive for a nice software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, . . .)

SV-LIB: Goals

(a) Consolidate existing IVLs from an engineering perspective, with focus on tools and interchange: take well-understood constructs, avoiding semantic complexity (procedural extension of SMT-LIB)

(b) Keep programs and specifications separable and the latter extensible: achieved by a robust mechanism to refer to and annotate program points

(c) Validation ≈ Verification: have full spectrum of potential witnesses standardized to enable and foster interchange across tools and use-cases

(d) Strive for a nice software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, . . .)

Goal: Consolidation of Existing IVLs

```
(set-logic LIA)

(define-proc
  add
  ((x0 Int) (y0 Int))
  ((x Int))
  ((y Int))
  (! (sequence
       (assign (x x0) (y y0))
       (! (while
            (< 0 y)
            (assign
              (x (+ x 1))
              (y (- y 1))))
        :tag while-loop))
    :tag proc-add))
```

```
(annotate-tag
  proc-add
  :requires (<= 0 y0)
  :ensures  (= x (+ x0 y0)))

(annotate-tag while-loop :not-recurring)

(declare-const x1 Int)
(declare-const y1 Int)

(verify-call add (x1 y1))
```

```
((annotate-tag
   while-loop
   :invariant
     (and
       (<= 0 y)
       (= (+ x y) (+ x0 y0)))
   :decreases y))
```

Goal: Consolidation of Existing IVLs

```
(set-logic LIA)

(define-proc
  add
  ((x0 Int) (y0 Int))
  ((x Int))
  ((y Int))
  (! (sequence
       (assign (x x0) (y y0))
       (! (while
            (<= 0 y)
            (assign
              (x (+ x 1))
              (y (- y 1))))
        :tag while-loop))
    :tag proc-add))
```

```
((select-trace
   (model
     (define-fun x1 () Int 1)
     (define-fun y1 () Int 1))
   (init-global-vars)
   (entry-proc add)
   (steps
     (init-proc-vars add
        ; initialization of x and y
        ; not necessary
        ))
   (incorrect-annotation
     proc-add
     :ensures
     (= x (+ x0 y0)))))
```

SV-LIB: Goals

(a) Consolidate existing IVLs from an engineering perspective, with focus on tools and interchange: take well-understood constructs, avoid semantic complexity (procedural extension of SMT-LIB)

(b) Keep programs and specifications separable and the latter extensible: achieved by a robust mechanism to refer to and annotate program points

(c) Validation ≈ Verification: have full spectrum of potential witnesses standardized to enable and foster interchange across tools and use-cases

(d) Strive for a nice software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, . . .)

Goal: Keep Programs and Specifications Separate

```
(set-logic LIA)

(define-proc
  add
  ((x0 Int) (y0 Int))
  ((x Int))
  ((y Int))
  (! (sequence
       (assign (x x0) (y y0))
       (! (while
            (< 0 y)
            (assign
              (x (+ x 1))
              (y (- y 1))))
        :tag while-loop))
    :tag proc-add))
```

```
(annotate-tag
  proc-add
  :requires (<= 0 y0)
  :ensures  (= x (+ x0 y0)))

(annotate-tag while-loop :not-recurring)

(declare-const x1 Int)
(declare-const y1 Int)

(verify-call add (x1 y1))
```

```
((annotate-tag
   while-loop
   :invariant
     (and
       (<= 0 y)
       (= (+ x y) (+ x0 y0)))
   :decreases y))
```

SV-LIB: Goals

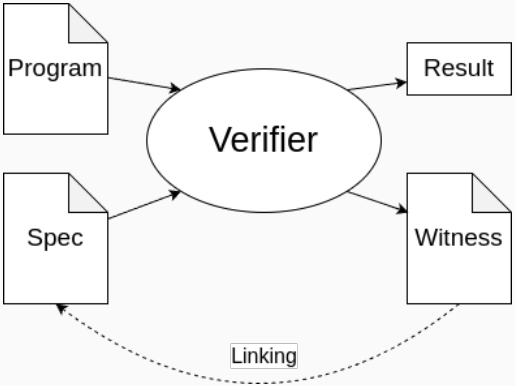
(a) Consolidate existing IVLs from an engineering perspective, with focus on tools and interchange: take well-understood constructs, avoid semantic complexity (procedural extension of SMT-LIB)

(b) Keep programs and specifications separable and the latter extensible: achieved by a robust mechanism to refer to and annotate program points

(c) Validation ≈ Verification: have full spectrum of potential witnesses standardized to enable and foster interchange across tools and use-cases

(d) Strive for a nice software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, . . .)

Goal: Validation ≈ Verification



(a) adding a witness from a sound tool can only make the specification more precise

(b) adding wrong invariants leads to additional ways to violate the spec

(c) adding a wrong CEX can be characterized meaningfully and a witness exported for such a task

Goal: Validation ≈ Verification

```
(set-logic LIA)

(define-proc
  add
  ((x0 Int) (y0 Int))
  ((x Int))
  ((y Int))
  (! (sequence
       (assign (x x0) (y y0))
       (! (while
            (< 0 y)
            (assign
              (x (+ x 1))
              (y (- y 1))))
        :tag while-loop))
    :tag proc-add))
```

```
(annotate-tag
  proc-add
  :requires (<= 0 y0)
  :ensures  (= x (+ x0 y0)))

(annotate-tag while-loop :not-recurring)

(declare-const x1 Int)
(declare-const y1 Int)

(verify-call add (x1 y1))
```

```
((annotate-tag
   while-loop
   :invariant
     (and
       (<= 0 y)
       (= (+ x y) (+ x0 y0)))
   :decreases y))
```

Goal: Validation ≈ Verification

```
(set-logic LIA)

(define-proc
  add
  ((x0 Int) (y0 Int))
  ((x Int))
  ((y Int))
  (! (sequence
       (assign (x x0) (y y0))
       (! (while
            (< 0 y)
            (assign
              (x (+ x 1))
              (y (- y 1))))
        :tag while-loop))
    :tag proc-add))
```

```
(annotate-tag
  proc-add
  :requires (<= 0 y0)
  :ensures  (= x (+ x0 y0))
  :not-recurring)

(declare-const x1 Int)
(declare-const y1 Int)

; Command taken from the witness
(annotate-tag
  while-loop
  :invariant
    (and
      (<= 0 y)
      (= (+ x y) (+ x0 y0)))
  :decreases y)

(verify-call add (x1 y1))
```

SV-LIB: Goals

(a) Consolidate existing IVLs from an engineering perspective, with focus on tools and interchange: take well-understood constructs, avoid semantic complexity (procedural extension of SMT-LIB)

(b) Keep programs and specifications separable and the latter extensible: achieved by a robust mechanism to refer to and annotate program points

(c) Validation ≈ Verification: have full spectrum of potential witnesses standardized to enable and foster interchange across tools and use-cases

(d) Strive for a nice software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, . . .)

Goal: Software Ecosystem

Aim to provide building blocks to facilitate adoption:

(a) ANTLR Grammar for parsing SV-LIB

(b) PySvLib: Python library for working with SV-LIB programs

(c) Integration of SV-LIB into CPAchecker

Demo: PySvLib

```
git clone https://gitlab.com/sosy-lab/benchmarking/sv-lib.git
cd sv-lib/examples/core-verification
git checkout 48d4beedec0f86a004b8aaa29c8193d727a94d7a
pip install git+https://gitlab.com/sosy-lab/benchmarking/sv-lib.git\
  @48d4beedec0f86a004b8aaa29c8193d727a94d7a#subdirectory=pysvlib
pysvlib lint loop-simple-safe.svlib # expected exit code 0
pysvlib lint loop-simple-unsafe.svlib # expected exit code 0
```

Demo: CPAchecker

```
git clone https://gitlab.com/sosy-lab/benchmarking/sv-lib.git
cd sv-lib/examples/core-verification
git checkout 48d4beedec0f86a004b8aaa29c8193d727a94d7a
apt install cpachecker # https://doi.org/10.48550/arXiv.2409.02094
cpachecker loop-simple-safe.svlib # expected ‘true‘ as output
cpachecker loop-simple-unsafe.svlib # expected ‘false‘ as output
```

SV-LIB: Goals Recap

(a) Consolidate existing IVLs from an engineering perspective, with focus on tools and interchange: take well-understood constructs, avoid semantic complexity (procedural extension of SMT-LIB)

(b) Keep programs and specifications separable and the latter extensible: achieved by a robust mechanism to refer to and annotate program points

(c) Validation $\approx$ Verification: have full spectrum of potential witnesses standardized to enable and foster interchange across tools and use-cases

(d) Strive for a nice software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, ...)

Lessons and Insights

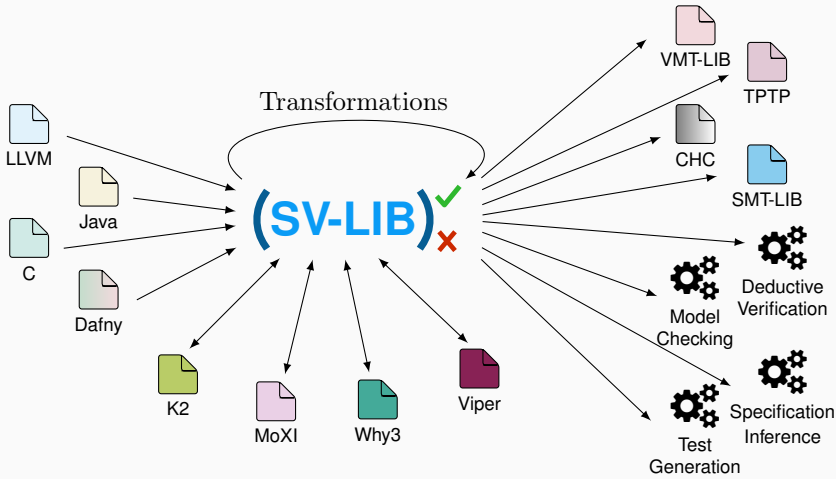
(a) inductiveness: what does it really mean in practice? leap as part of CEX

(b) Verification = Validation (with the right problem formulation)

(c) surprisingly interesting: monotonicity of collecting witnesses when ground truth is not available: how to accomodate multi-source, possibly conflicting evidence on the properties of the program

(d) outlook: general liveness properties

Future Work


Transformations
(SV-LIB)
LLVM
Java
C
Dafny
K2
MoXI
Why3
Viper
VMT-LIB
TPTP
CHC
SMT-LIB
Deductive Verification
Model Checking
Specification Inference
Test Generation

Conclusion

(a) SV-LIB 1.0 format available to read [1]

(b) Addresses multiple challenges in the SV-COMP and VerifyThis communities

(c) Designed to cover both automated and autoactive verification use-cases

(d) Tool support available

https://gitlab.com/sosy-lab/benchmarking/sv-lib

References i

[1] Beyer, D., Ernst, G., Jonáš, M., Lingsch-Rosenfeld, M.: SV-LIB 1.0: A standard exchange format for software-verification tasks. arXiv/CoRR **2511**(21509) (December 2025). https://doi.org/10.48550/arXiv.2511.21509