# SV-LIB 1.0

## A Standard Exchange Format for Software-Verification Tasks

**Marian Lingsch-Rosenfeld**

Dirk Beyer Gidon Ernst Martin Jonáš

2026-03-17

LMU Munich, Germany

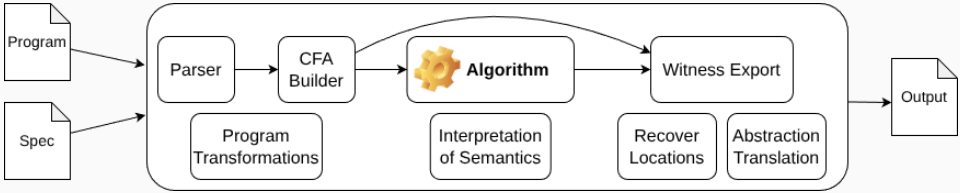
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

(SV-LIB)

SoSy-Lab Software Systems

# Implementing a static analyzer is difficult!

# Why is it difficult?




Algorithmic vs. Engineering complexity

# Motivation for another IVL

(a) Reduce engineering complexity (e.g., easy to parse and interpret)

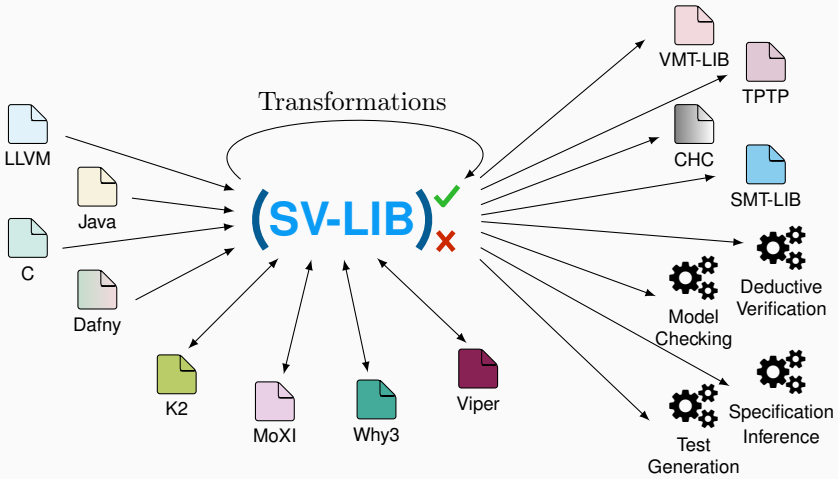
(b) Complexities of the input language transfer to complexity in presenting verification results (e.g., witnesses)

(c) Information exchange between tools is difficult (e.g., witnesses, ...)

(d) Have formal semantics without undefined behavior

(e) many more [1]

# SV-LIB


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

# SV-LIB: Goals

(a) Consolidate existing IVLs from an engineering perspective by taking well-understood constructs, and avoiding semantic complexity

(b) Keep programs and specifications separable and the latter extensible

(c) Make witnesses first-class citizens by providing a standard format for them

(d) Make implementation of verification tools easier

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

# SV-LIB: Goals

(a) Consolidate existing IVLs from an engineering perspective by taking well-understood constructs, and avoiding semantic complexity

(b) Keep programs and specifications separable and the latter extensible

(c) Make witnesses first-class citizens by providing a standard format for them

(d) Make implementation of verification tools easier

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

# Syntax of SV-LIB

```
(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))
```

# Syntax of SV-LIB

```
(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)))))
```

(a) Consolidate existing IVLs from an engineering perspective by taking well-understood constructs, and avoiding semantic complexity

(b) Keep programs and specifications separable and the latter extensible

(c) Make witnesses first-class citizens by providing a standard format for them

(d) Make implementation of verification tools easier

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

# 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))
```

(a) Consolidate existing IVLs from an engineering perspective by taking well-understood constructs, and avoiding semantic complexity

(b) Keep programs and specifications separable and the latter extensible

(c) Make witnesses first-class citizens by providing a standard format for them

(d) Make implementation of verification tools easier

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

# Witnesses as first-class citizens

```
(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))
```

# Witnesses as first-class citizens

```
(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))
```

(a) Consolidate existing IVLs from an engineering perspective by taking well-understood constructs, and avoiding semantic complexity

(b) Keep programs and specifications separable and the latter extensible

(c) Make witnesses first-class citizens by providing a standard format for them

(d) Make implementation of verification tools easier

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

# Making Implementation easier: SvLibChecker

| | Stand-Alone | | Cumulative | |
|---|---|---|---|---|
| **Algorithm** | **LOC** | **Comments** | **LOC** | **Comments** |
| Trace Abstraction | 113 | 62 | 196 | 131 |
| Symbolic Execution | 101 | 27 | 379 | 223 |
| BMC | 87 | 19 | 365 | 215 |
| Predicate Abstraction | 86 | 50 | 461 | 276 |
| Value Analysis | 75 | 25 | 353 | 221 |
| $k$-Induction | 59 | 22 | 424 | 237 |
| All Algorithms | 604 | 274 | 896 | 431 |
| Total Code Base | 2597 | 722 | 2597 | 722 |

(a) Consolidate existing IVLs from an engineering perspective by taking well-understood constructs, and avoiding semantic complexity

(b) Keep programs and specifications separable and the latter extensible

(c) Make witnesses first-class citizens by providing a standard format for them

(d) Make implementation of verification tools easier

(e) Strive for a 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

(d) Lightweight Model-Checker SvLibChecker

# 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
```

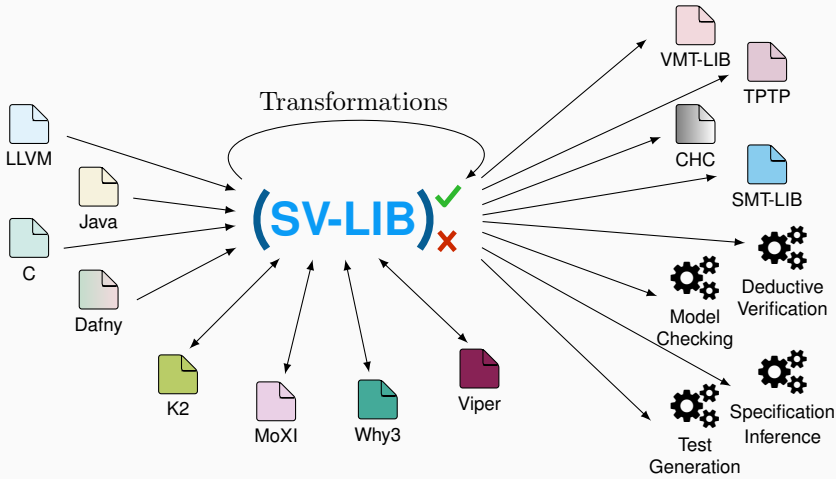
# Demo: CPAchecker

```
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
```

# Demo: SvLibChecker

```
git clone https://gitlab.com/sosy-lab/benchmarking/sv-lib.git
cd sv-lib/examples/core-verification
git checkout 48d4beedec0f86a004b8aaa29c8193d727a94d7a
git clone https://gitlab.com/sosy-lab/software/svlibchecker.git
git checkout f29766853548086e306bdbdd6c253e4428898239
cd svlibchecker && make package CLEAN=0 && cd ..
mv svlibchecker/package/svlibchecker-f2976685 .
cd svlibchecker-f2976685
./svlibchecker.py --config config/predicateAnalysisCegar.toml \
  ../loop-simple-safe.svlib # expected ‘correct‘ as output
./svlibchecker.py --config config/predicateAnalysisCegar.toml \
  ../loop-simple-unsafe.svlib # expected ‘incorrect‘ as output
```

# 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

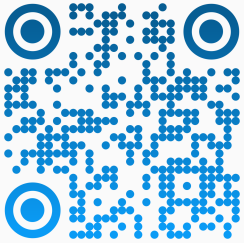
(a) SV-LIB 1.0 format available to read [1]

(b) Reduces engineering complexity for static analysis tools

(c) Addresses multiple challenges in the SV-COMP community

(d) First-class support for witnesses and validation

(e) 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