

SV-LIB 1.0

A Standard Exchange Format for Software-Verification Tasks

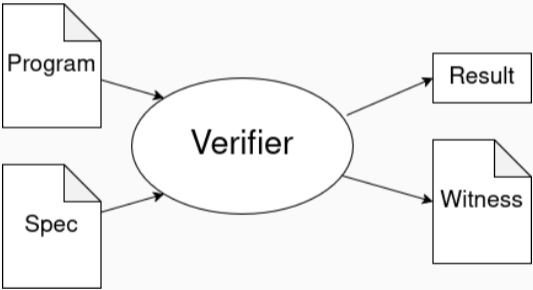
Marian Lingsch-Rosenfeld 

Dirk Beyer  Gidon Ernst  Martin Jonáš 

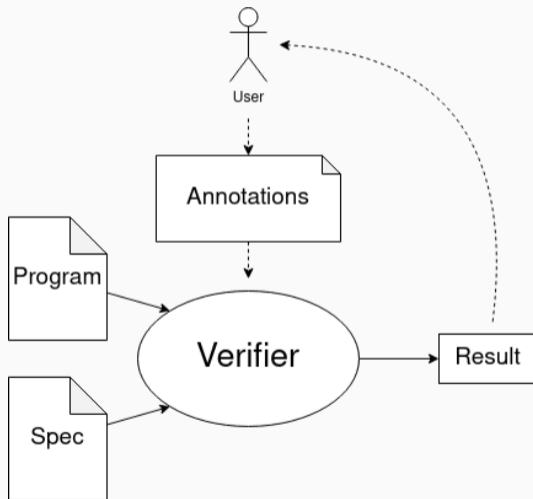
2026-04-12
LMU Munich, Germany



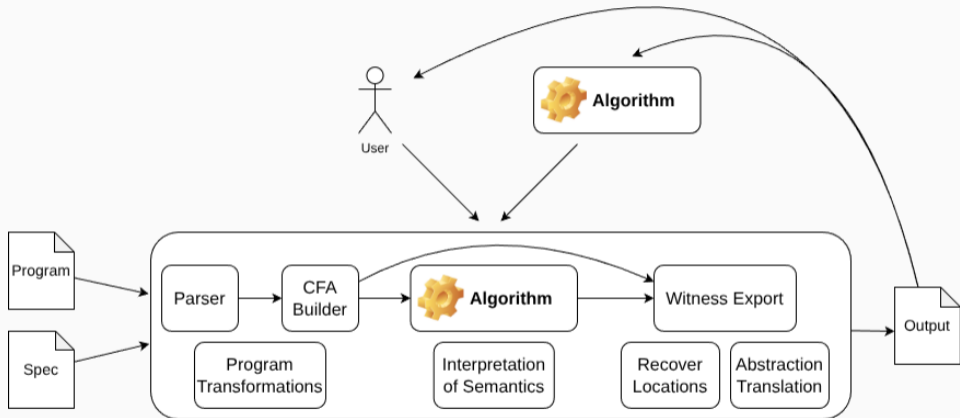
Automatic Software Verification



Deductive Software Verification

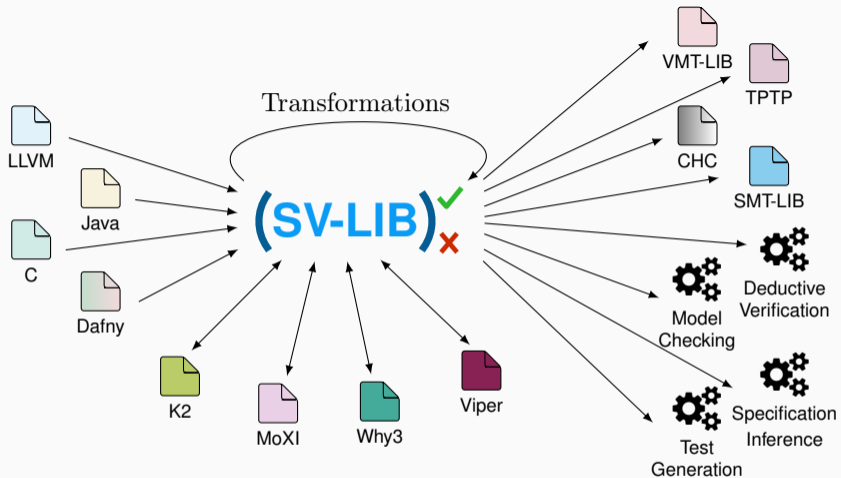


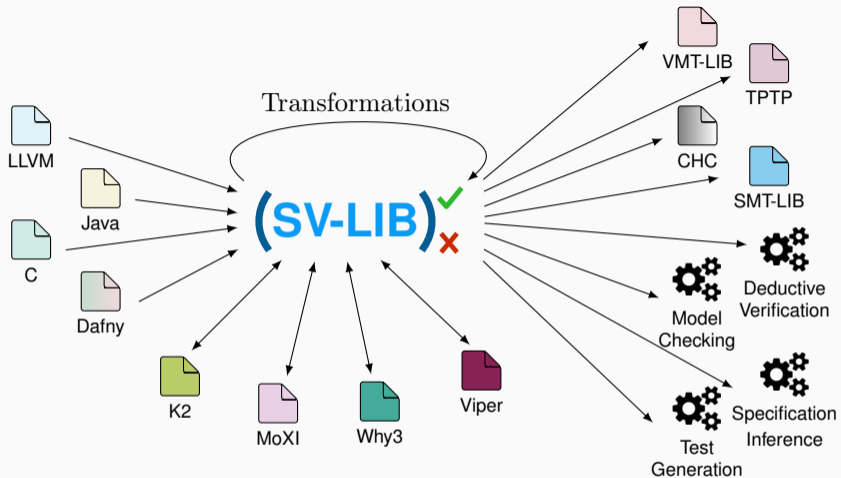
Cooperative Software Verification



Talking to the Algorithm has many extra steps!

SV-LIB





(a) Inspired by k2 [3]

(b) Based on SMT-LIB [1]

(c) MoXI [5] is for transition systems

(d) Btor2 [4] is for hardware circuits

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! [2]

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 information exchange between tools easier (e.g., witnesses, ...)
- (e) Make implementation of verification tools easier
- (f) 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 information exchange between tools easier (e.g., witnesses, ...)
- (e) Make implementation of verification tools easier
- (f) Strive for a software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, ...)

Syntax of SV-LIB

```
1 (set-logic LIA)
2
3 (define-proc
4   add
5   ((x0 Int) (y0 Int))
6   ((x Int))
7   ((y Int))
8   (! (sequence
9       (assign (x x0) (y y0))
10      (! (while
11          (< 0 y)
12          (assign
13            (x (+ x 1))
14            (y (- y 1))))))
15     :tag while-loop))
16   :tag proc-add))
```

```
17 (annotate-tag
18   proc-add
19   :requires (<= 0 y0)
20   :ensures (= x (+ x0 y0)))
21
22 (annotate-tag while-loop :not-recurring)
23
24 (declare-const x1 Int)
25 (declare-const y1 Int)
26
27 (verify-call add (x1 y1))
28
29 ((annotate-tag
30   while-loop
31   :invariant
32   (and
33     (<= 0 y)
34     (= (+ x y) (+ x0 y0)))
35   :decreases y))
```

Syntax of SV-LIB

```
1 (set-logic LIA)
2
3 (define-proc
4   add
5   ((x0 Int) (y0 Int))
6   ((x Int))
7   ((y Int))
8   (! (sequence
9       (assign (x x0) (y y0))
10      (! (while
11          (<= 0 y)
12          (assign
13              (x (+ x 1))
14              (y (- y 1))))))
15      :tag while-loop))
16   :tag proc-add))
17 [...]
```

```
((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 information exchange between tools easier (e.g., witnesses, ...)
- (e) Make implementation of verification tools easier
- (f) Strive for a software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, ...)

Keep Programs and Specifications Separate

```
1 (set-logic LIA)
2
3 (define-proc
4   add
5   ((x0 Int) (y0 Int))
6   ((x Int))
7   ((y Int))
8   (! (sequence
9       (assign (x x0) (y y0))
10      (! (while
11          (< 0 y)
12          (assign
13            (x (+ x 1))
14            (y (- y 1))))
15        :tag while-loop))
16   :tag proc-add))
```

```
17 (annotate-tag
18   proc-add
19   :requires (<= 0 y0)
20   :ensures (= x (+ x0 y0)))
21
22 (annotate-tag while-loop :not-recurring)
23
24 (declare-const x1 Int)
25 (declare-const y1 Int)
26
27 (verify-call add (x1 y1))
28
29 ((annotate-tag
30   while-loop
31   :invariant
32   (and
33     (<= 0 y)
34     (= (+ x y) (+ x0 y0)))
35   :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 information exchange between tools easier (e.g., witnesses, ...)
- (e) Make implementation of verification tools easier
- (f) Strive for a software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, ...)

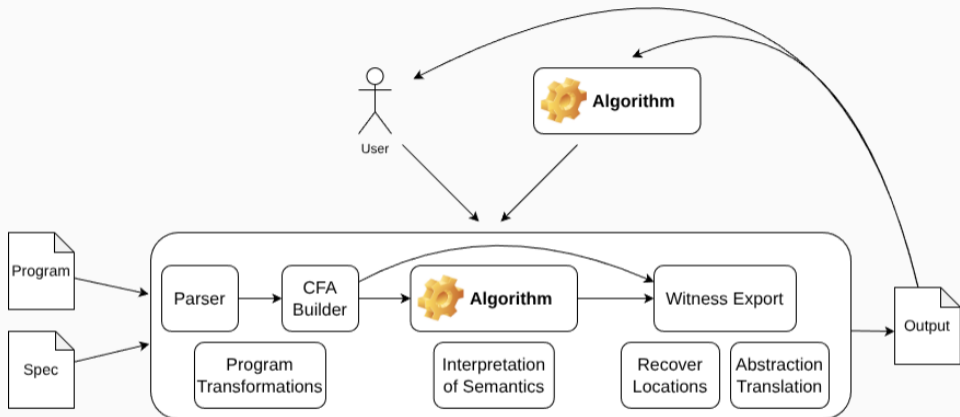
Witnesses as first-class citizens

```
1 (set-logic LIA)
2
3 (define-proc
4   add
5   ((x0 Int) (y0 Int))
6   ((x Int))
7   ((y Int))
8   (! (sequence
9       (assign (x x0) (y y0))
10      (! (while
11          (< 0 y)
12          (assign
13             (x (+ x 1))
14             (y (- y 1))))
15         :tag while-loop))
16     :tag proc-add))
17 (annotate-tag
18   proc-add
19   :requires (<= 0 y0)
20   :ensures (= x (+ x0 y0)))
21
22 (annotate-tag while-loop :not-recurring)
23
24 (declare-const x1 Int)
25 (declare-const y1 Int)
26
27 (verify-call add (x1 y1))
28
29 ((annotate-tag
30   while-loop
31   :invariant
32   (and
33     (<= 0 y)
34     (= (+ x y) (+ x0 y0)))
35   :decreases y))
```

Witnesses as first-class citizens

```
1 (set-logic LIA)
2
3 (define-proc
4   add
5   ((x0 Int) (y0 Int))
6   ((x Int))
7   ((y Int))
8   (! (sequence
9       (assign (x x0) (y y0))
10      (! (while
11          (< 0 y)
12          (assign
13            (x (+ x 1))
14            (y (- y 1))))
15         :tag while-loop))
16     :tag proc-add))
17 (annotate-tag
18   proc-add
19   :requires (<= 0 y0)
20   :ensures (= x (+ x0 y0))
21   :not-recurring)
22
23 (declare-const x1 Int)
24 (declare-const y1 Int)
25
26 ; Command taken from the witness
27 (annotate-tag
28   while-loop
29   :invariant
30   (and
31     (<= 0 y)
32     (= (+ x y) (+ x0 y0)))
33   :decreases y)
34
35 (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 information exchange between tools easier (e.g., witnesses, ...)**
- (e) Make implementation of verification tools easier
- (f) Strive for a software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, ...)



Communication with the algorithms is easier, and better!

- (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 information exchange between tools easier (e.g., witnesses, ...)
- (e) Make implementation of verification tools easier**
- (f) Strive for a software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, ...)

Making Implementation easier: SvLibChecker

Algorithm	Stand-Alone		Cumulative	
	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
<i>k</i> -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 information exchange between tools easier (e.g., witnesses, ...)
- (e) Make implementation of verification tools easier
- (f) Strive for a software ecosystem by providing building blocks (parser, linter, default validator, transformations, benchmarks, ...)**

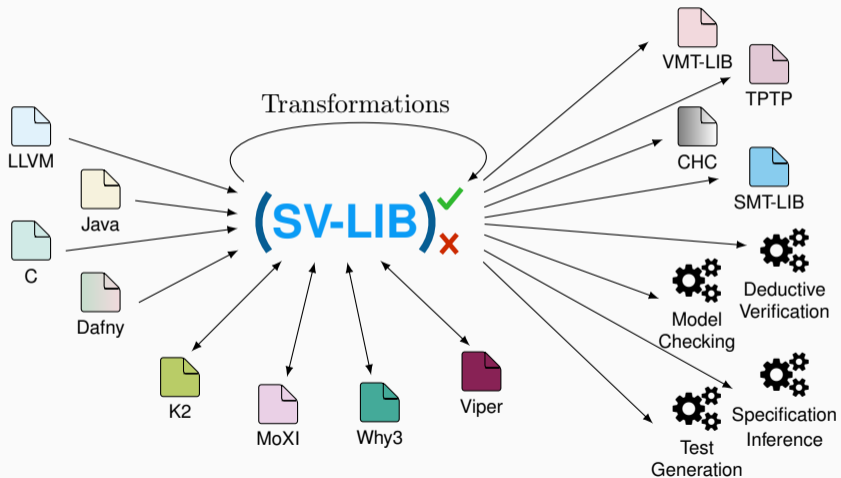
Goal: Software Ecosystem

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`
- (e) Try it out yourself!

<https://gitlab.com/sosy-lab/research/data/sv-lib-demo>

Future Work



Conclusion

- (a) SV-LIB 1.0 format available [2]!
- (b) Reduces engineering complexity for static analysis tools
- (c) First-class support for witnesses and validation
- (d) Makes information exchange between tools easier (e.g., witnesses, ...)
- (e) Tool support available



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

References i

- [1] Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. Tech. rep., University of Iowa (2010), <https://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r10.12.21.pdf>
- [2] 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>
- [3] Griggio, A., Jonáš, M.: KRATOS2: An SMT-based model checker for imperative programs. In: Proc. CAV. pp. 423–436. Springer (2023).
https://doi.org/10.1007/978-3-031-37709-9_20

References ii

- [4] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: BTOR2, BTORMC, and BOOLECTOR 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_32
- [5] Rozier, K.Y., Dureja, R., Irfan, A., Johannsen, C., Nukala, K., Shankar, N., Tinelli, C., Vardi, M.Y.: MoXI: An intermediate language for symbolic model checking. In: Proc. SPIN. pp. 26–46. LNCS 14624, Springer (2024). https://doi.org/10.1007/978-3-031-66149-5_2