How Good Are Your Invariants: Witness Validation for Hardware via Circuit Instrumentation with Software Invariants

Zsófia Ádám

Supervisors: Nian-Ze Lee, Po-Chun Chien, Dirk Beyer

MŰEGYETEM 1782

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

ftsrg Critical Systems Research Group

Introduction

Software **and** Hardware Verification?

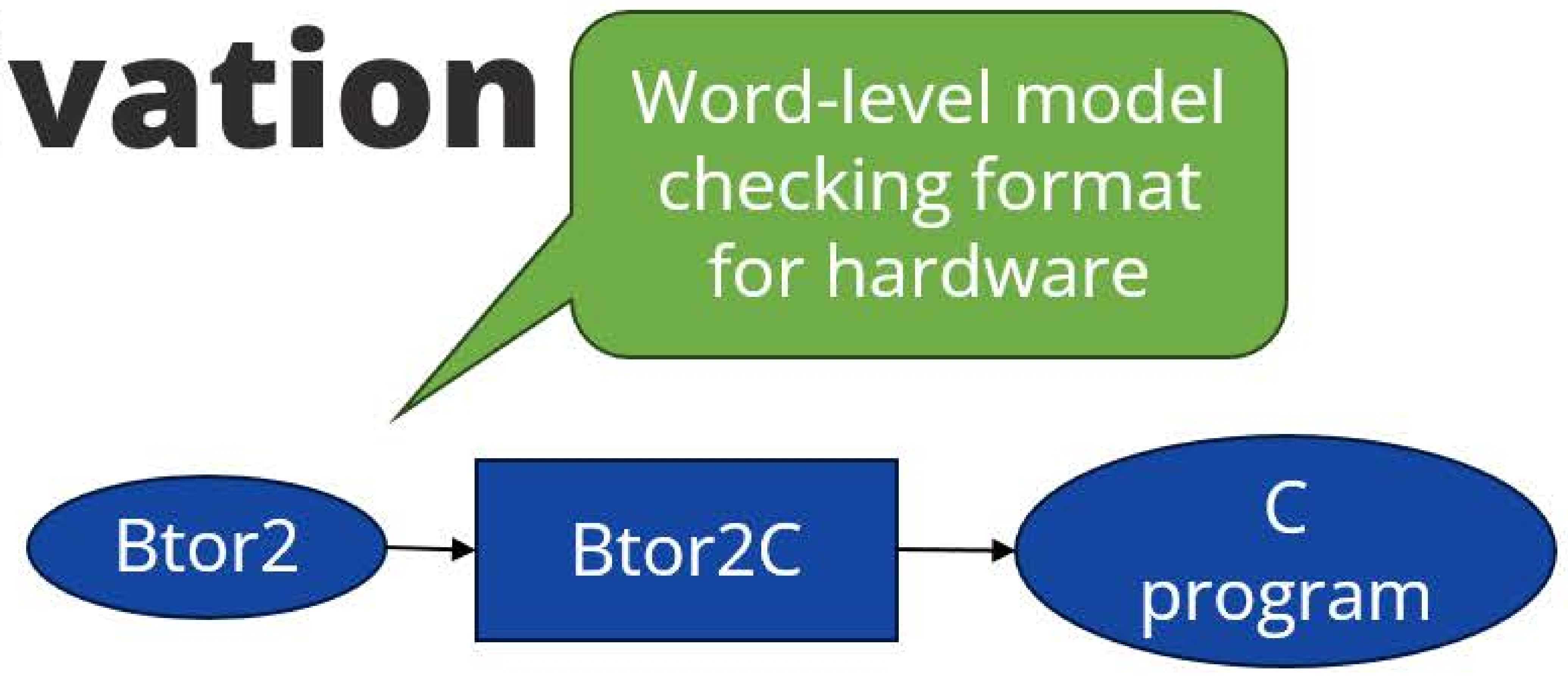
Motivation

Motivation

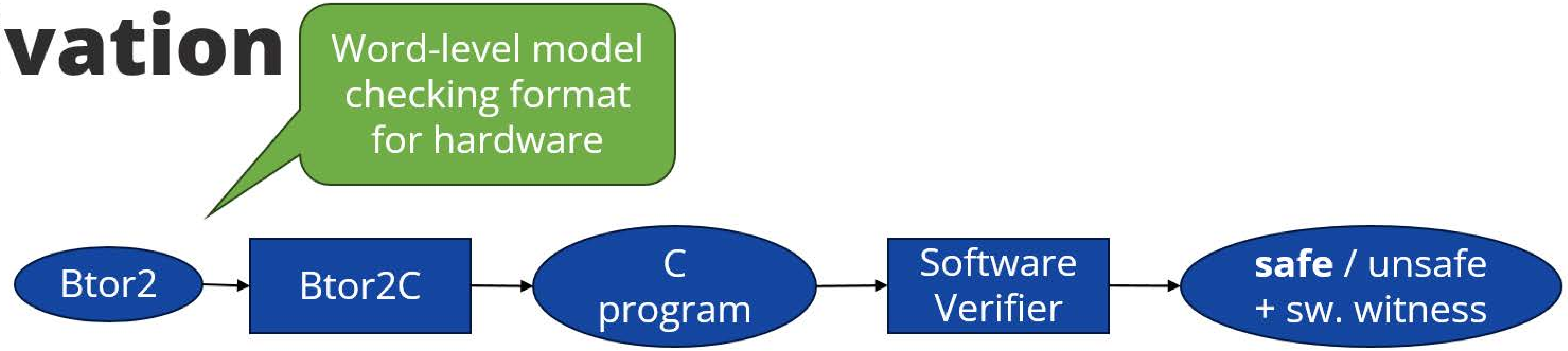
Word-level model checking format for hardware

Btor2

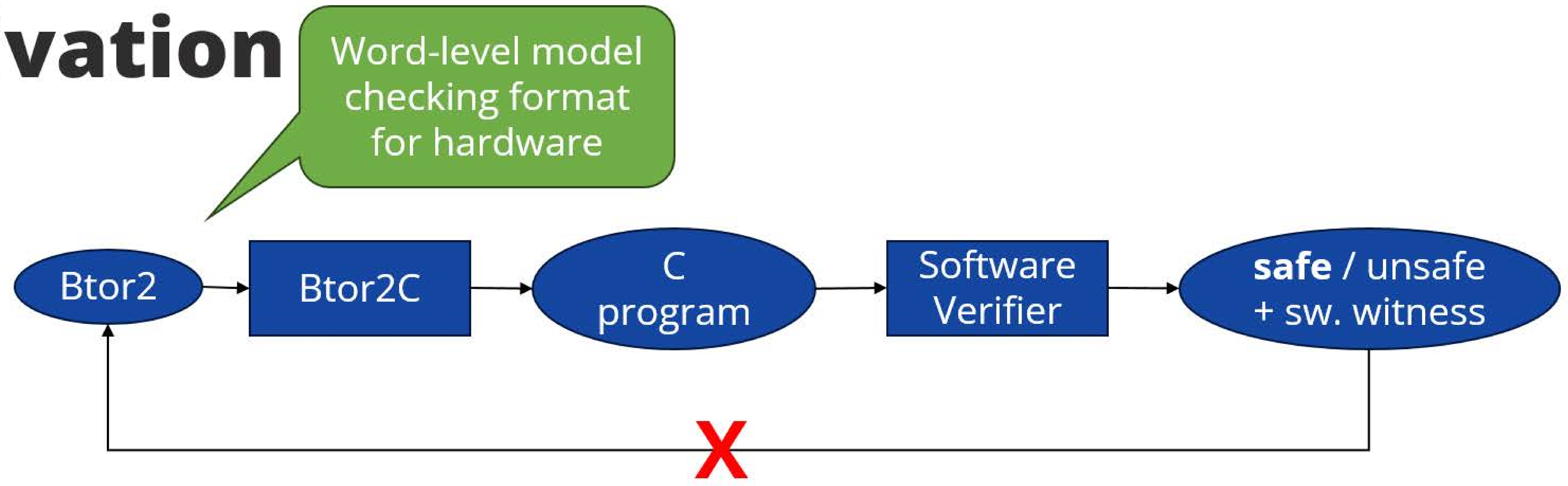
Motivation


Word-level model checking format for hardware
Btor2
Btor2C
C program

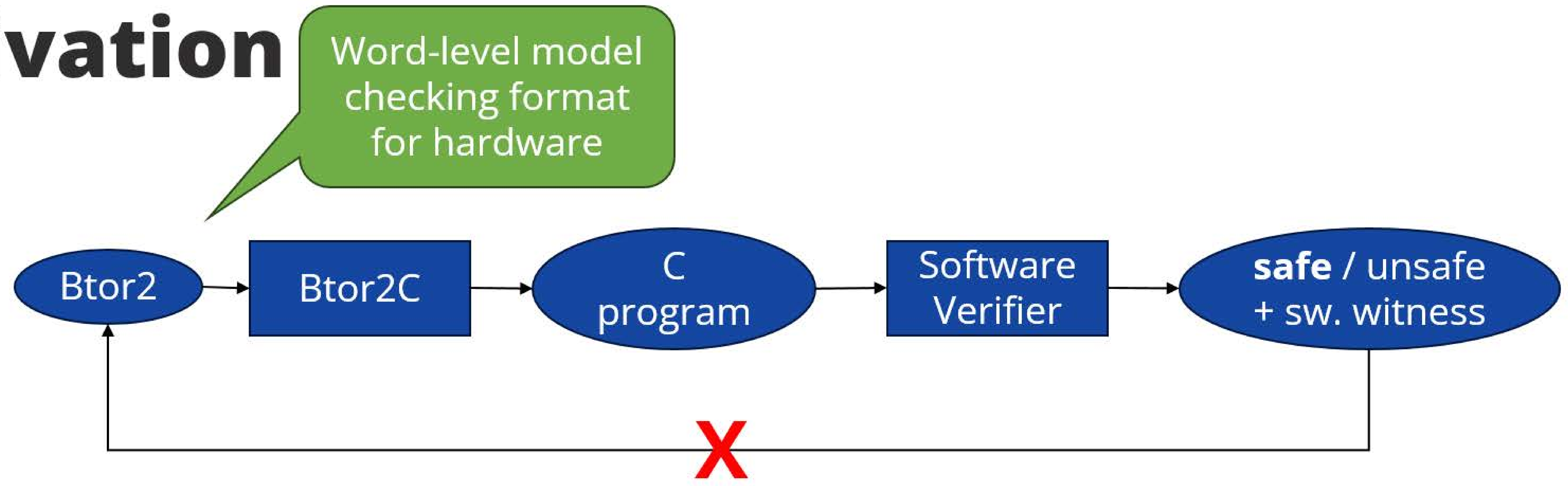
Motivation


Word-level model checking format for hardware
Btor2
Btor2C
C program
Software Verifier
safe / unsafe + sw. witness

Motivation

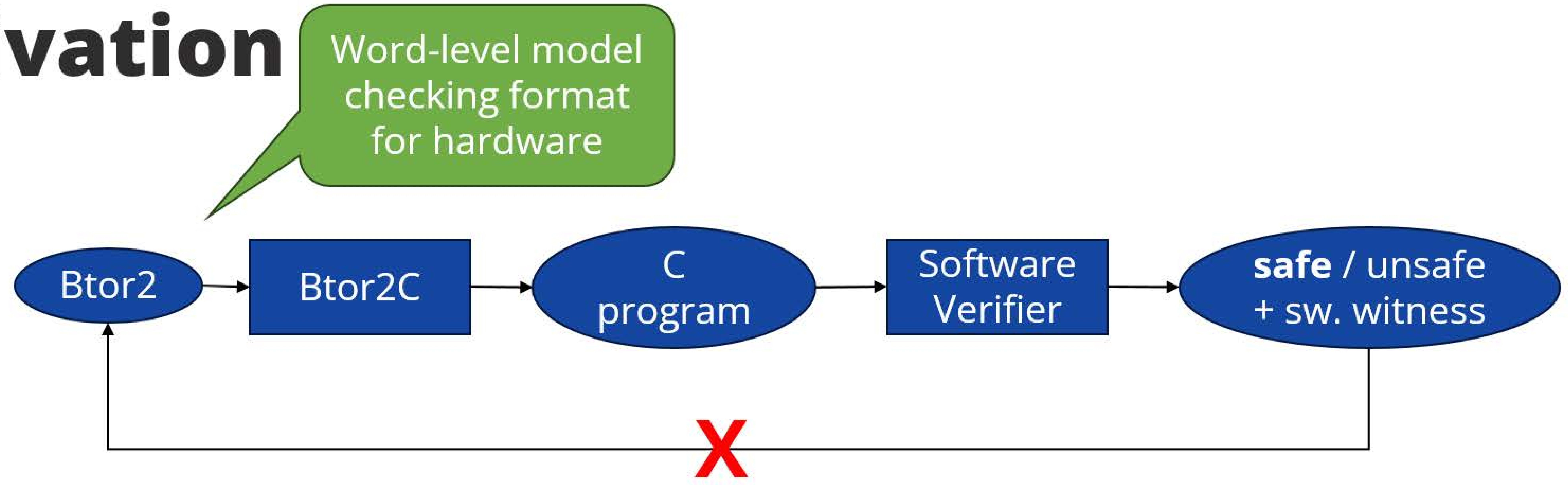

Word-level model checking format for hardware
Btor2
Btor2C
C program
Software Verifier
safe / unsafe + sw. witness
X

Motivation


Word-level model checking format for hardware
Btor2
Btor2C
C program
Software Verifier
safe / unsafe + sw. witness
X

Validation of Software Correctness Witnesses for Btor2C

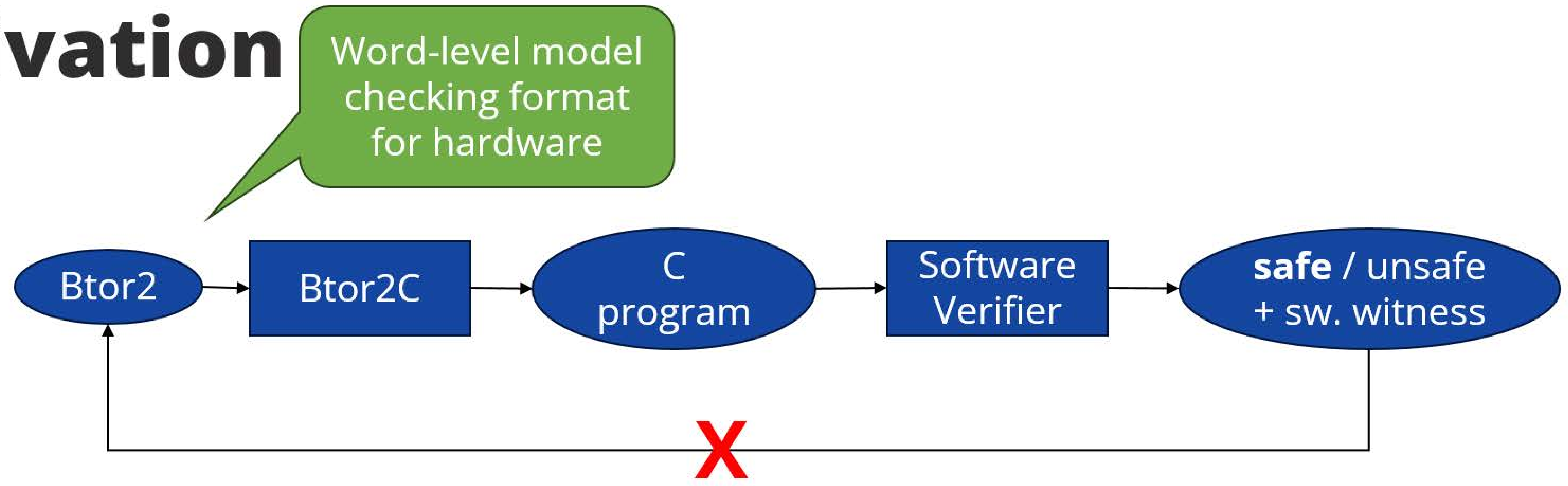
Motivation


Word-level model checking format for hardware
Btor2
Btor2C
C program
Software Verifier
safe / unsafe + sw. witness
X

Validation of Software Correctness Witnesses for Btor2C

- Validate **software correctness witnesses** (invariants)
  - and provide insight on the "hardware side"

Motivation


Word-level model checking format for hardware
Btor2
Btor2C
C program
Software Verifier
safe / unsafe + sw. witness
X

Validation of Software Correctness Witnesses for Btor2C

- Validate **software correctness witnesses** (invariants)
  - and provide insight on the "hardware side"
- Show that there is **no discrepancy** inbetween C program and circuit
  - or find the issues

Validation Approach

From software back to Hardware

Challenges and Approach

Challenges and Approach

- No Btor2 correctness witness format

Challenges and Approach

- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself

Challenges and Approach

- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself
- Software Witness: automaton with invariants, loose constraints on format

Challenges and Approach

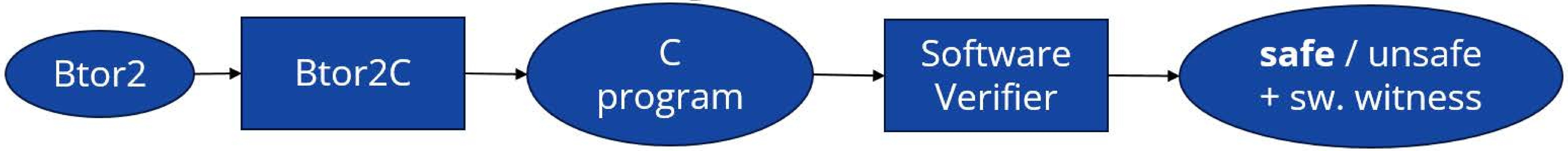
- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself
- Software Witness: automaton with invariants, loose constraints on format
  - Btor2C **C program structure fixed** – invariants required (in general)

Challenges and Approach

- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself
- Software Witness: automaton with invariants, loose constraints on format
  - Btor2C **C program structure fixed** – invariants required (in general)
- **Don’t re-verify** - Validation should be easy and fast
  - In practice: invariants might not be inductive, witnesses are automata, validators are often verifiers reconfigured

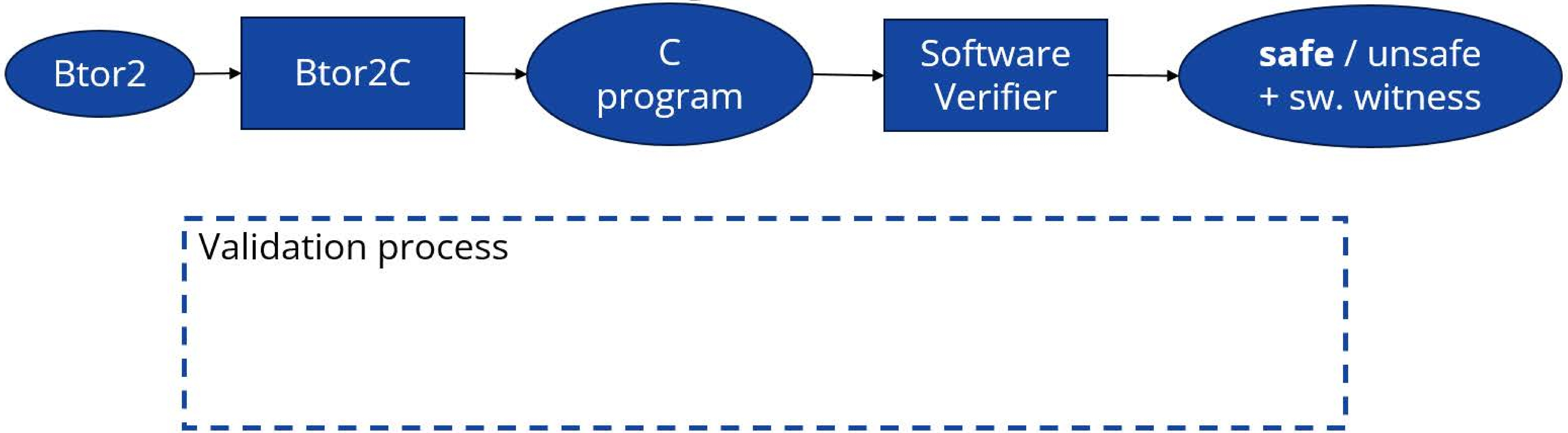
Challenges and Approach

- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself
- Software Witness: automaton with invariants, loose constraints on format
  - Btor2C **C program structure fixed** – invariants required (in general)
- **Don't re-verify** - Validation should be easy and fast
  - In practice: invariants might not be inductive, witnesses are automata, validators are often verifiers reconfigured



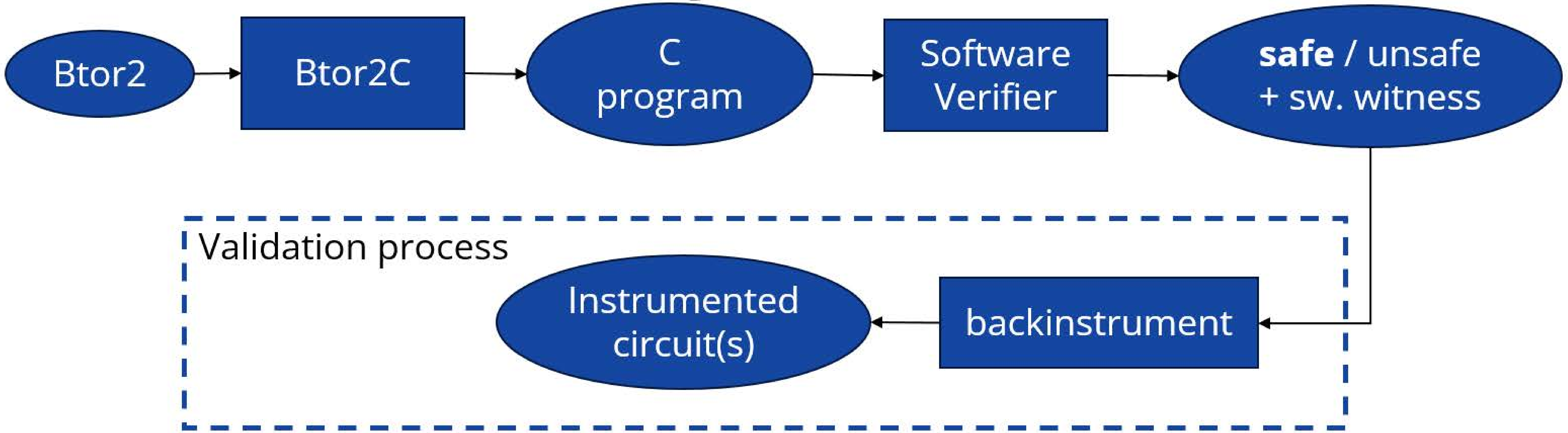
Challenges and Approach

- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself
- Software Witness: automaton with invariants, loose constraints on format
  - Btor2C **C program structure fixed** – invariants required (in general)
- **Don't re-verify** - Validation should be easy and fast
  - In practice: invariants might not be inductive, witnesses are automata, validators are often verifiers reconfigured



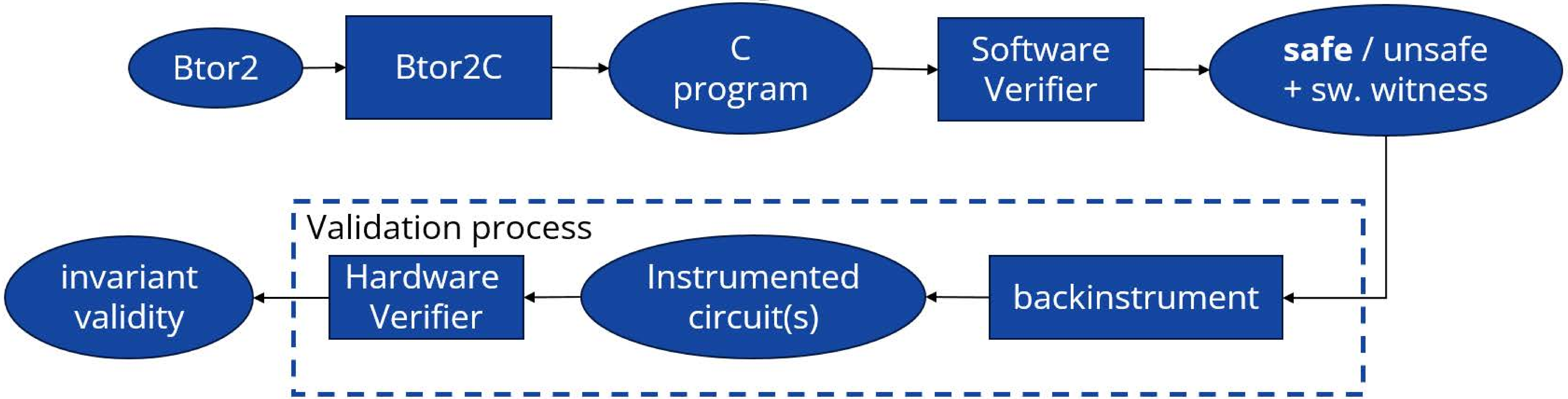
Challenges and Approach

- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself ✓
- Software Witness: automaton with invariants, loose constraints on format
  - Btor2C **C program structure fixed** – invariants required (in general) ✓
- **Don't re-verify** - Validation should be easy and fast
  - In practice: invariants might not be inductive, witnesses are automata, validators are often verifiers reconfigured



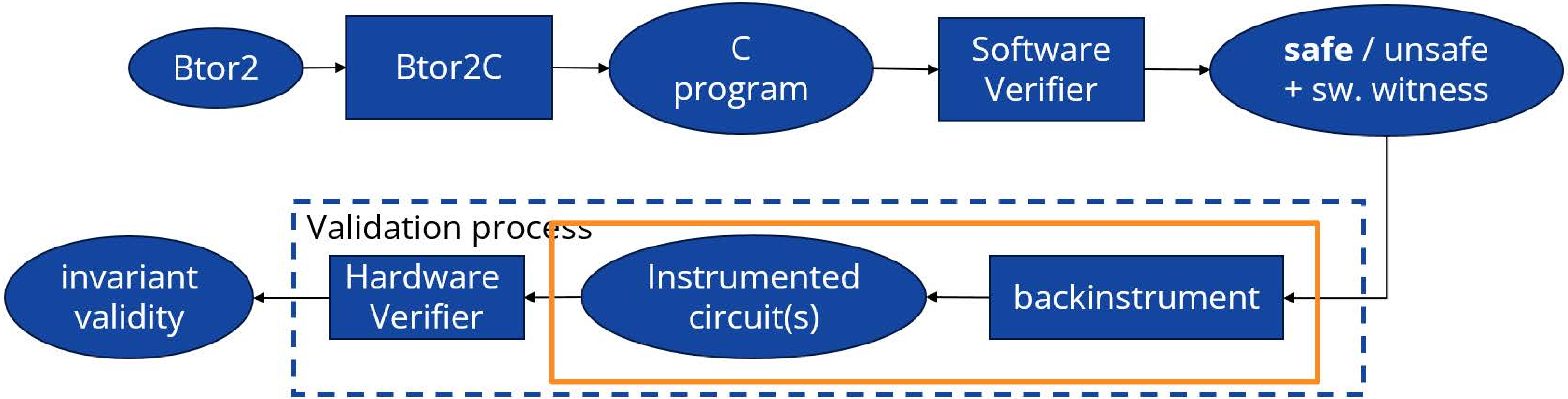
Challenges and Approach

- No Btor2 correctness witness format
  - Not needed! Just **instrument the circuit** itself ✓
- Software Witness: automaton with invariants, loose constraints on format
  - Btor2C **C program structure fixed** – invariants required (in general) ✓
- **Don't re-verify** - Validation should be easy and fast
  - In practice: invariants might not be inductive, witnesses are automata, validators are often verifiers reconfigured

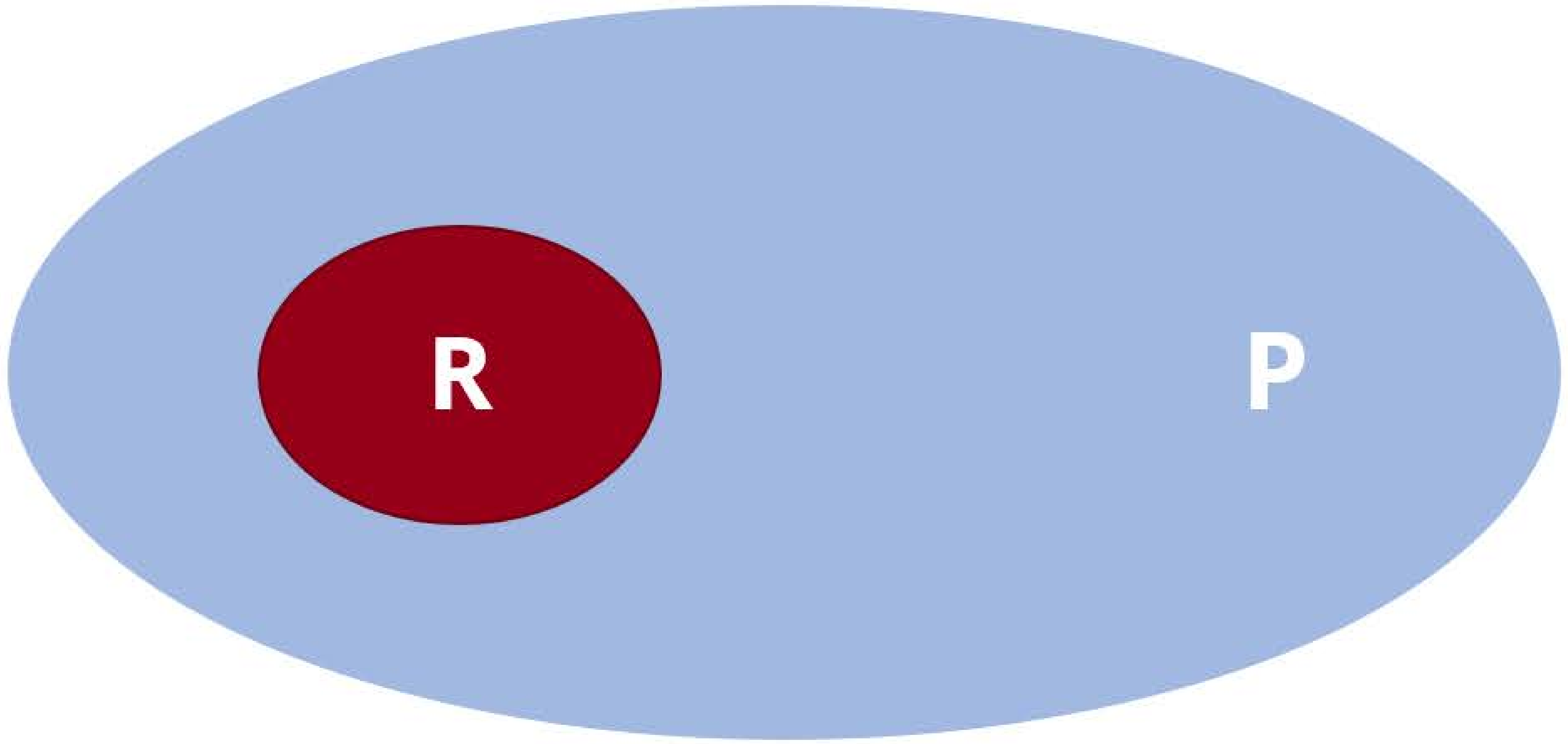


Challenges and Approach

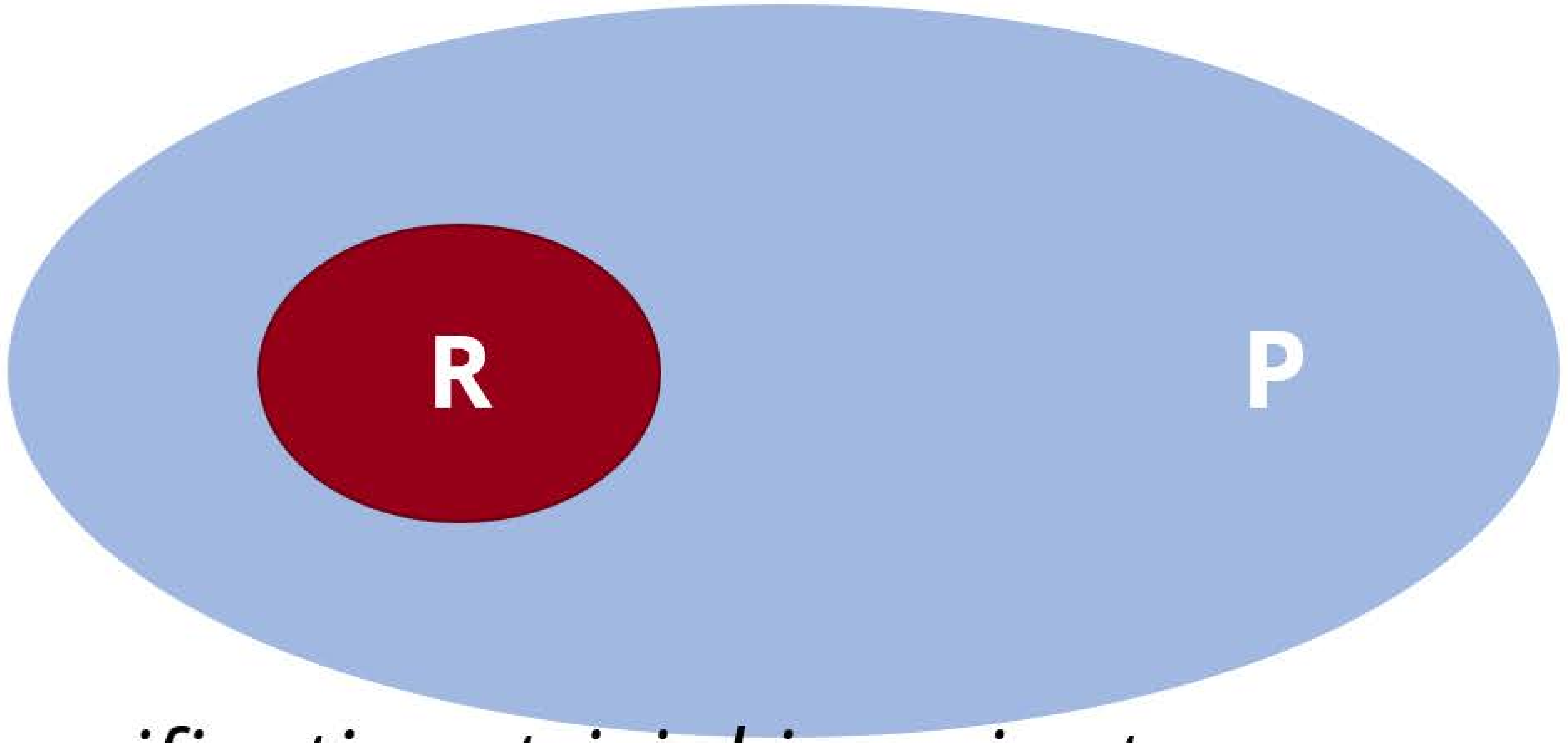
- No Btor2 correctness witness format ✓
  - Not needed! Just **instrument the circuit** itself
- Software Witness: automaton with invariants, loose constraints on format ✓
  - Btor2C **C program structure fixed** – invariants required (in general)
- **Don't re-verify** - Validation should be easy and fast ?
  - In practice: invariants might not be inductive, witnesses are automata, validators are often verifiers reconfigured



How good is your invariant?

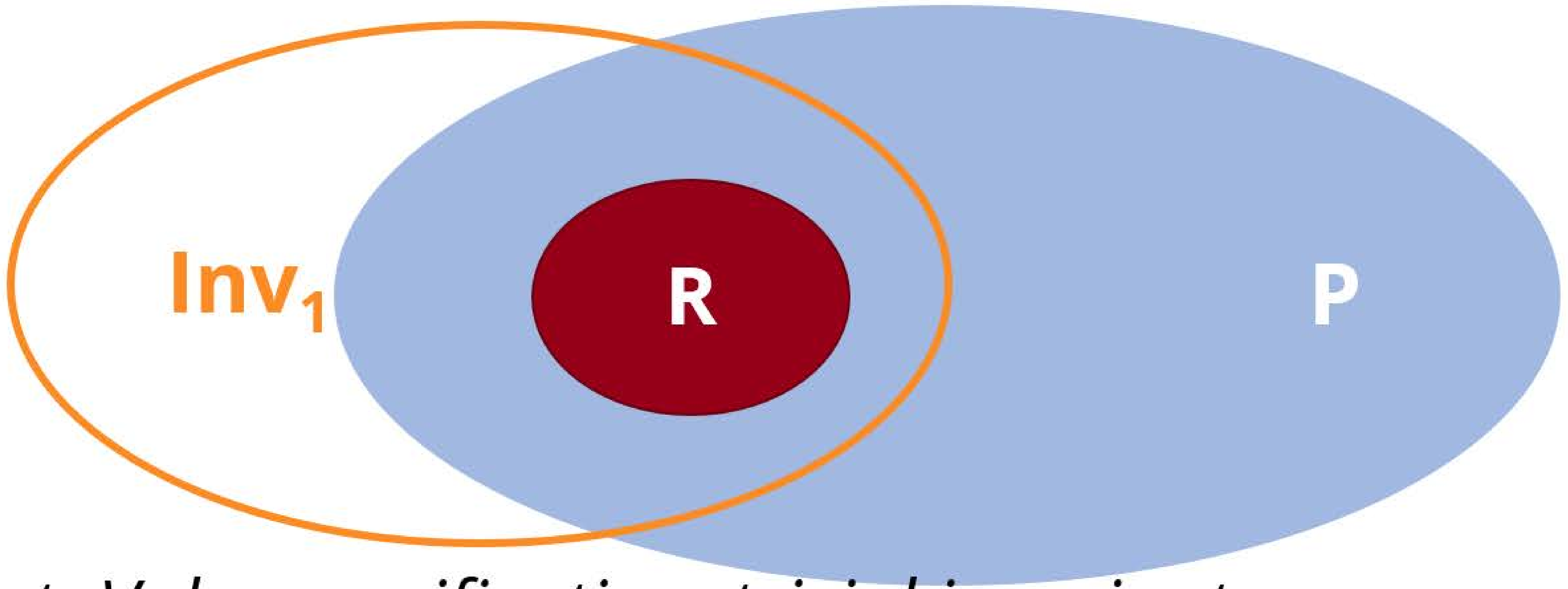


How good is your invariant?



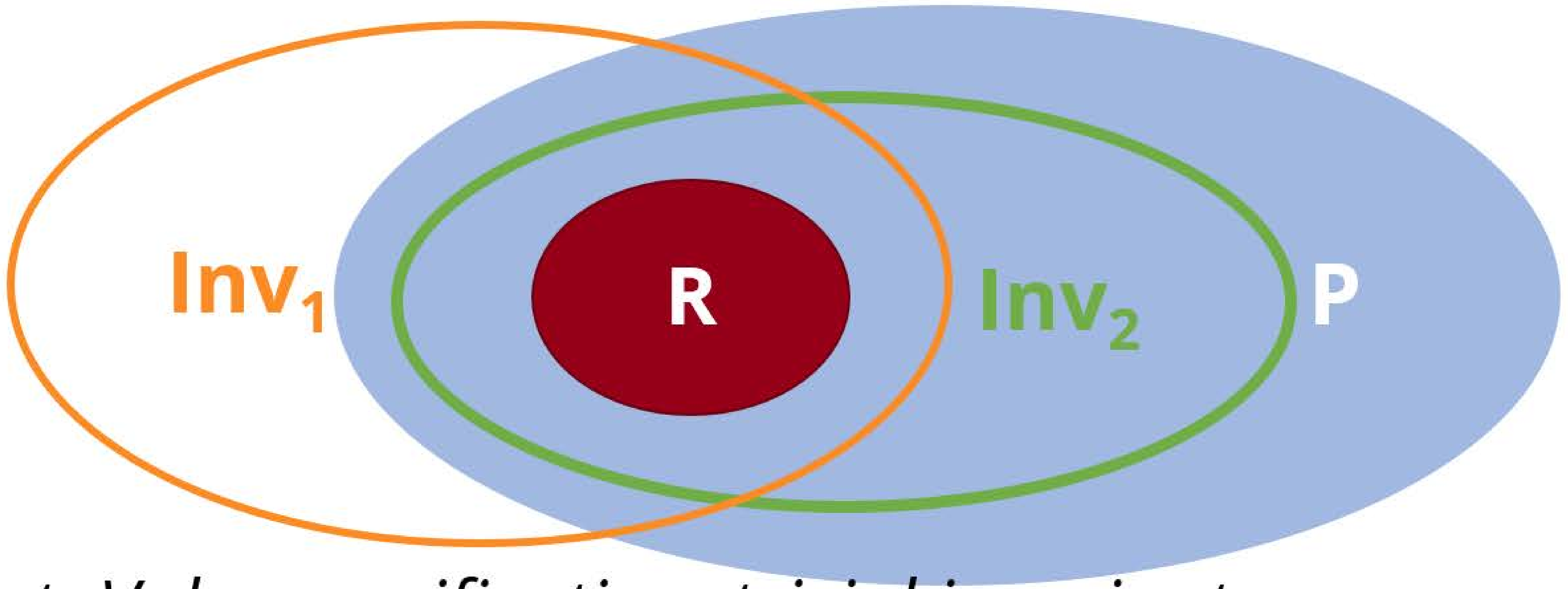
0. R(s) => P(s) ∧ Inv(s) *MetaVal, re-verification, trivial invariants*

How good is your invariant?



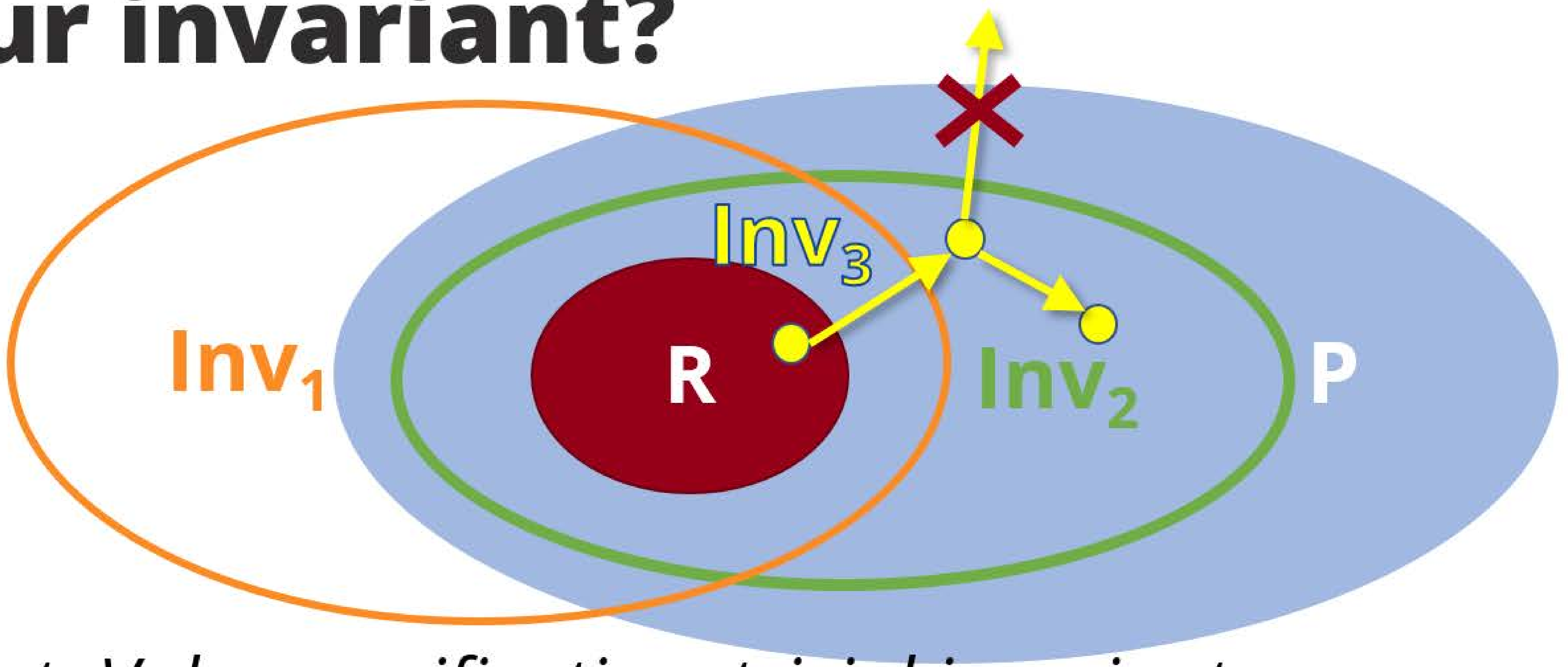
0. **R(s) => P(s) ∧ Inv(s)** *MetaVal, re-verification, trivial invariants*
1. **R(s) => Inv(s)** *Invariant, without P, not validation*

How good is your invariant?



0. **R(s) => P(s) ∧ Inv(s)** *MetaVal, re-verification, trivial invariants*
1. **R(s) => Inv(s)** *Invariant, without P, not validation*
2. **R(s) => Inv(s) ∧ Inv(s) => P(s)** *Safe Invariant, second half SAT*

How good is your invariant?



0. R(s) => P(s) ∧ Inv(s) *MetaVal, re-verification, trivial invariants*

1. R(s) => Inv(s) *Invariant, without P, not validation*

2. R(s) => Inv(s) ∧ Inv(s) => P(s) *Safe Invariant, second half SAT*

3. Inductive Invariant

0) Inv(s) => P(s)

1) I(s) => Inv(s)

2) Inv(s) ∧ T(s, s') => Inv(s’)

Example: (1) Invariant Check

R(s) => Inv(s) - Invariant Check (1)

- Basic check for correctness witness
- Does not prove anything about P, but no re-verification

Example: (1) Invariant Check

R(s) => Inv(s) - Invariant Check (1)

Basic check for correctness witness

Does not prove anything about P,
but no re-verification

```
...
<node id="N43">
<data key="invariant">
state_23 == 200
</data>
<data key="invariant.scope">main</data>
</node>
...
```

Example: (1) Invariant Check

R(s) => Inv(s) - Invariant Check (1)

Basic check for correctness witness
Does not prove anything about P,
but no re-verification

```
...
<node id="N43">
<data key="invariant">
state_23 == 200
</data>
<data key="invariant.scope">main</data>
</node>
...
```

→

(1..39 is the original Btor2 circuit)

1 sort bitvec 1
2 sort bitvec 20
... **; remove original property**

Example: (1) Invariant Check

R(s) => Inv(s) - Invariant Check (1)

- Basic check for correctness witness
- Does not prove anything about P, but no re-verification

```
...
<node id="N43">
<data key="invariant">
state_23 == 200
</data>
<data key="invariant.scope">main</data>
</node>
...
```

→

(1..39 is the original Btor2 circuit)

```
1 sort bitvec 1
2 sort bitvec 20
... ; remove original property
40 constd 2 200 ; create a constant (200)
```

Example: (1) Invariant Check

R(s) => Inv(s) - Invariant Check (1)

Basic check for correctness witness

Does not prove anything about P, but no re-verification

```
...
<node id="N43">
<data key="invariant">
state_23 == 200
</data>
<data key="invariant.scope">main</data>
</node>
...
```

→

(1..39 is the original Btor2 circuit)

```
1 sort bitvec 1
2 sort bitvec 20
... ; remove original property
40 constd 2 200 ; create a constant (200)
41 eq 1 23 40 ; state_23 == 200
```

Example: (1) Invariant Check

R(s) => Inv(s) - Invariant Check (1)

Basic check for correctness witness

Does not prove anything about P,
but no re-verification

```
...
<node id="N43">
<data key="invariant">
state_23 == 200
</data>
<data key="invariant.scope">main</data>
</node>
...
```

→

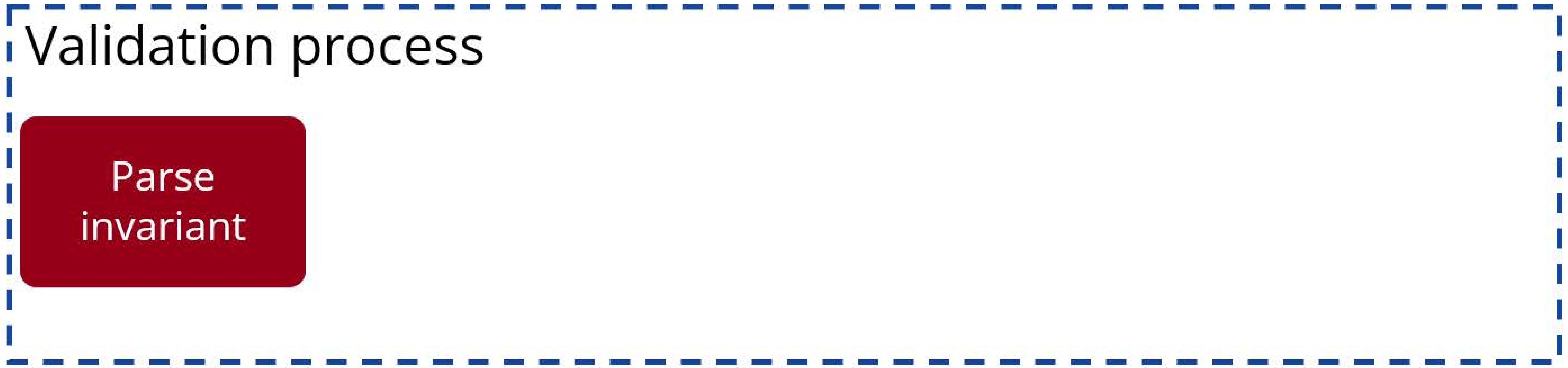
(1..39 is the original Btor2 circuit)

```
1 sort bitvec 1
2 sort bitvec 20
... ; remove original property
40 constd 2 200 ; create a constant (200)
41 eq 1 23 40 ; state_23 == 200
42 neg 1 41 ; negate the property
43 bad 42
; bad = „negation of safety property”
```

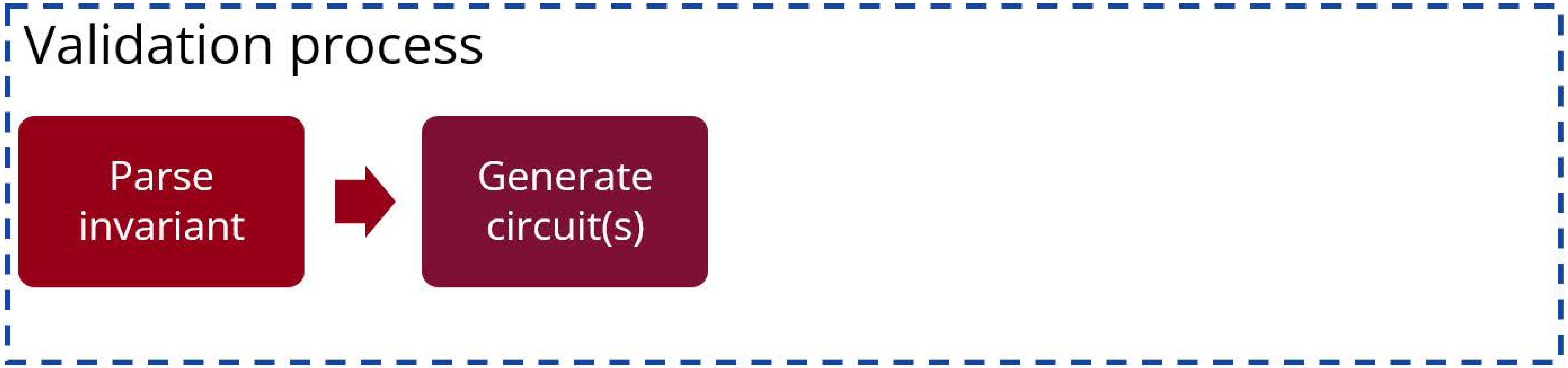
Full Validation Process

Validation process

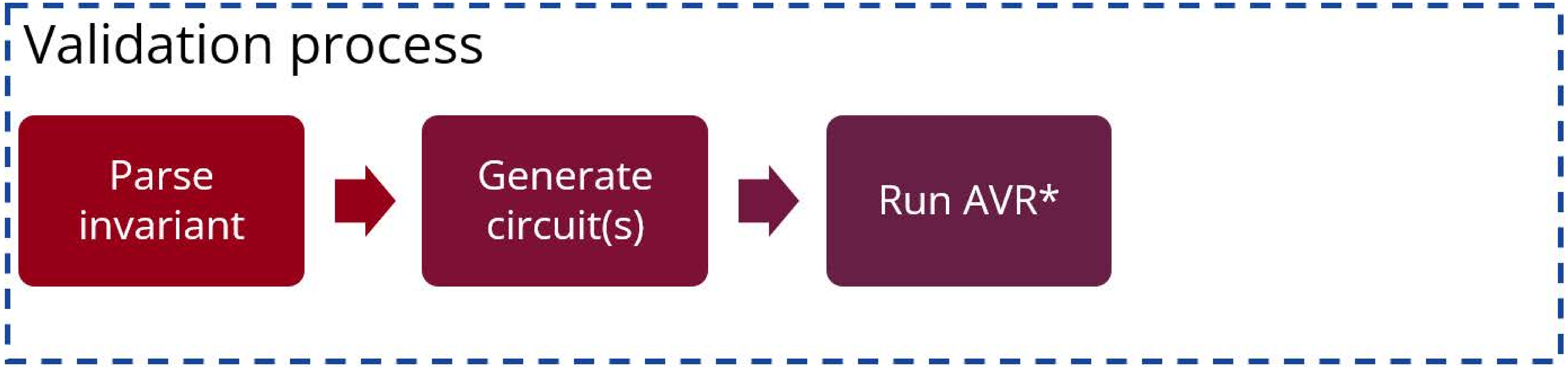
Full Validation Process



Full Validation Process

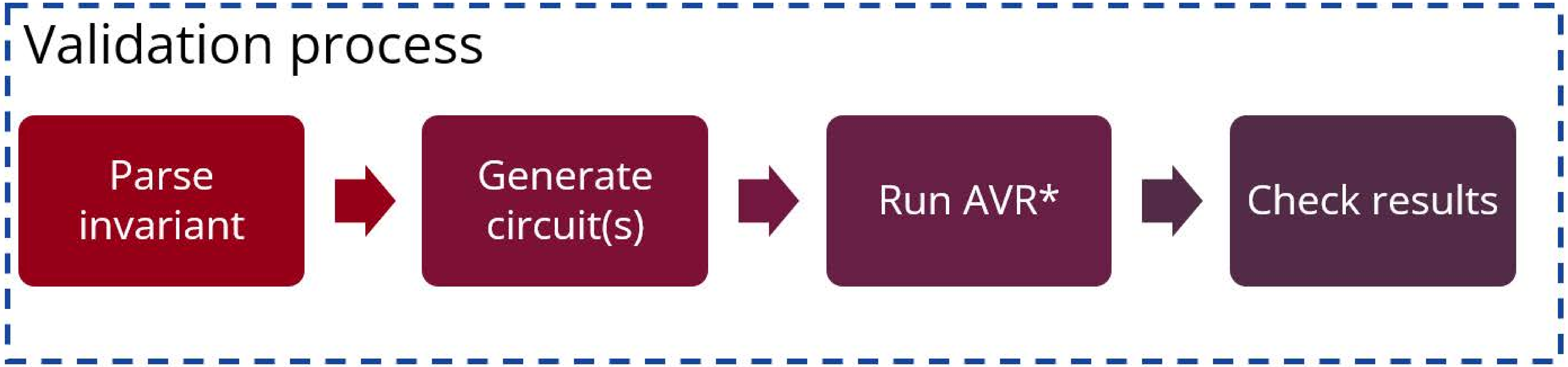


Full Validation Process



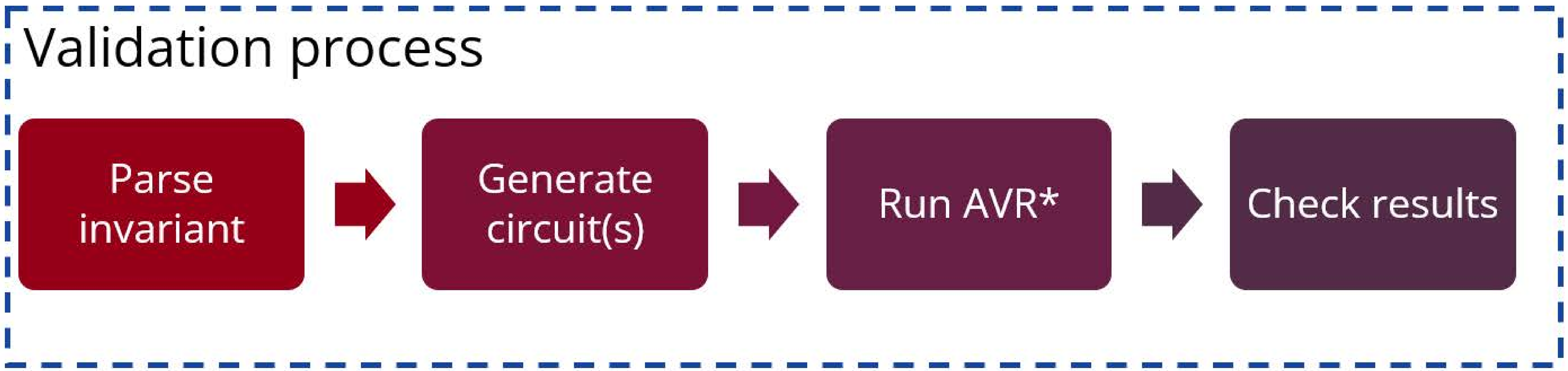
* github.com/aman-goel/avr

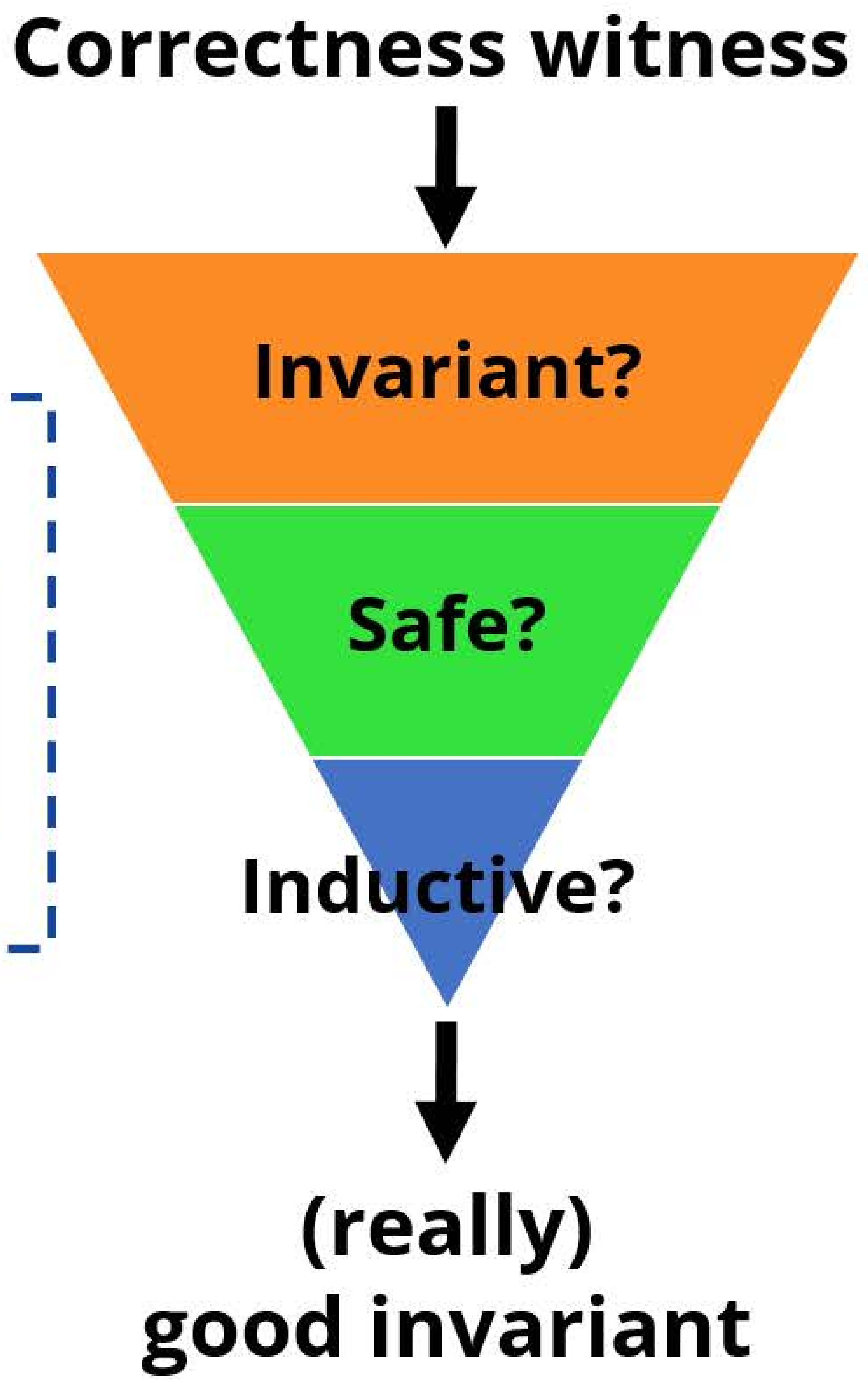
Full Validation Process



* github.com/aman-goel/avr

Full Validation Process



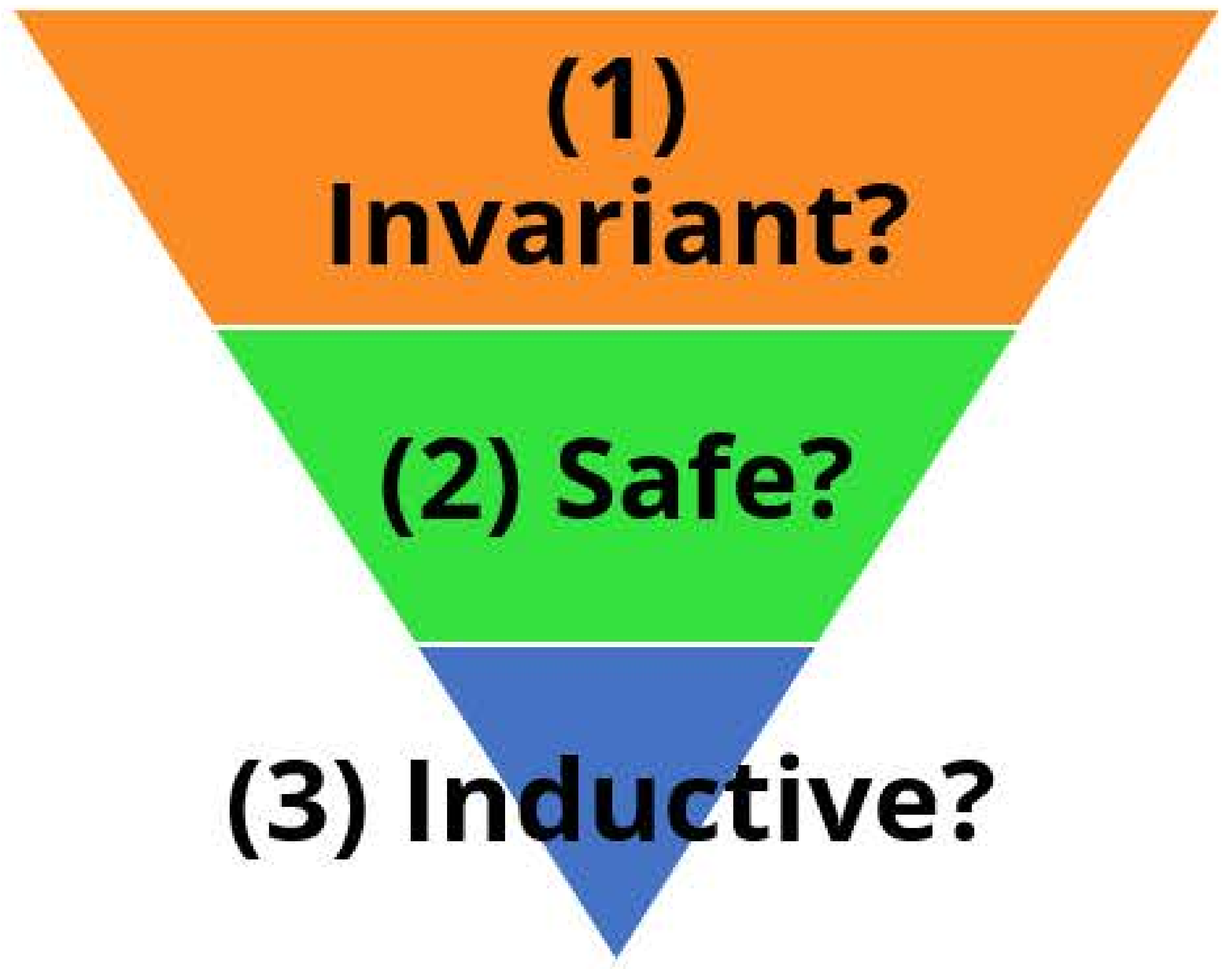


* github.com/aman-goel/avr

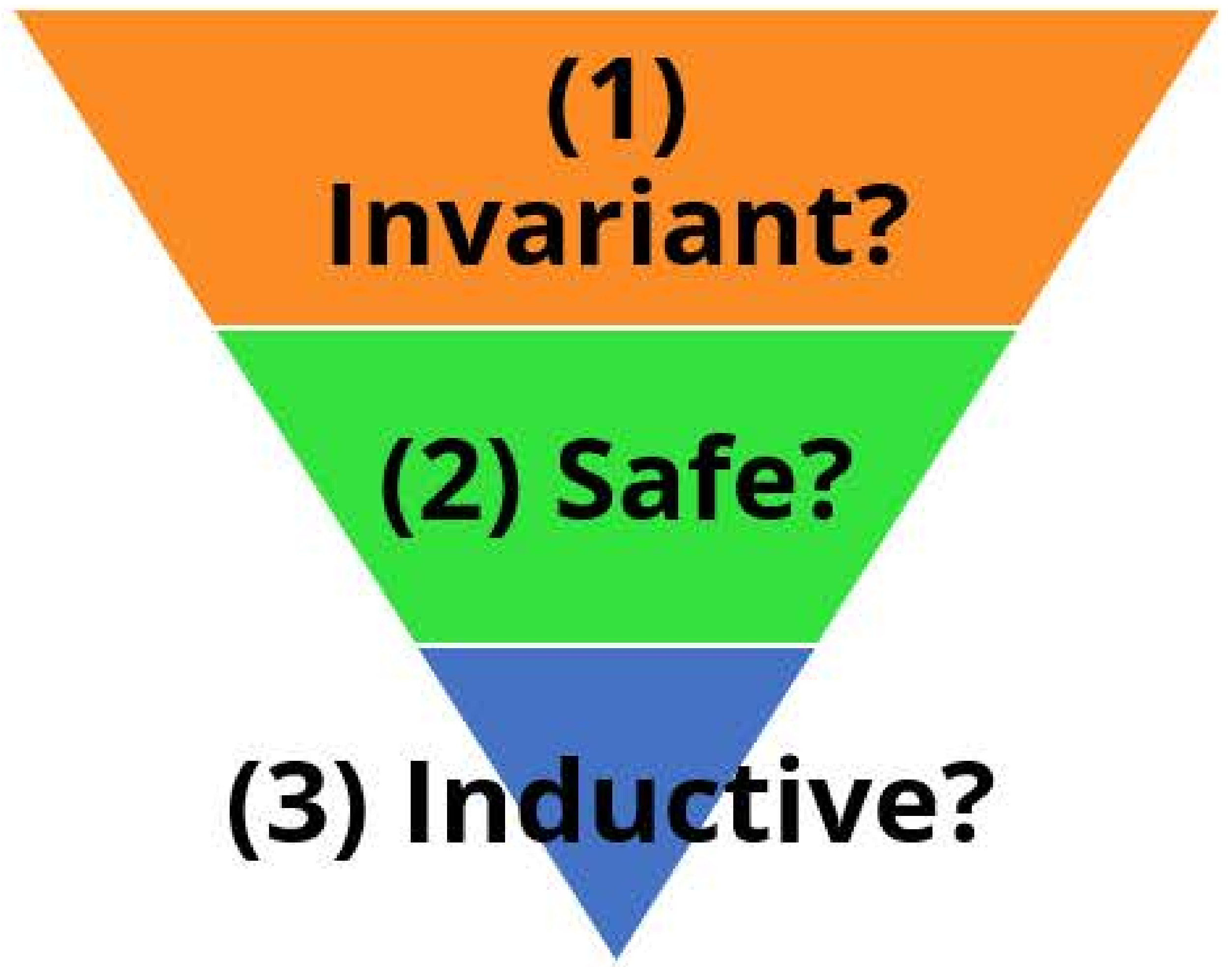
Results so far

Preliminary Benchmarks

Preliminary Results

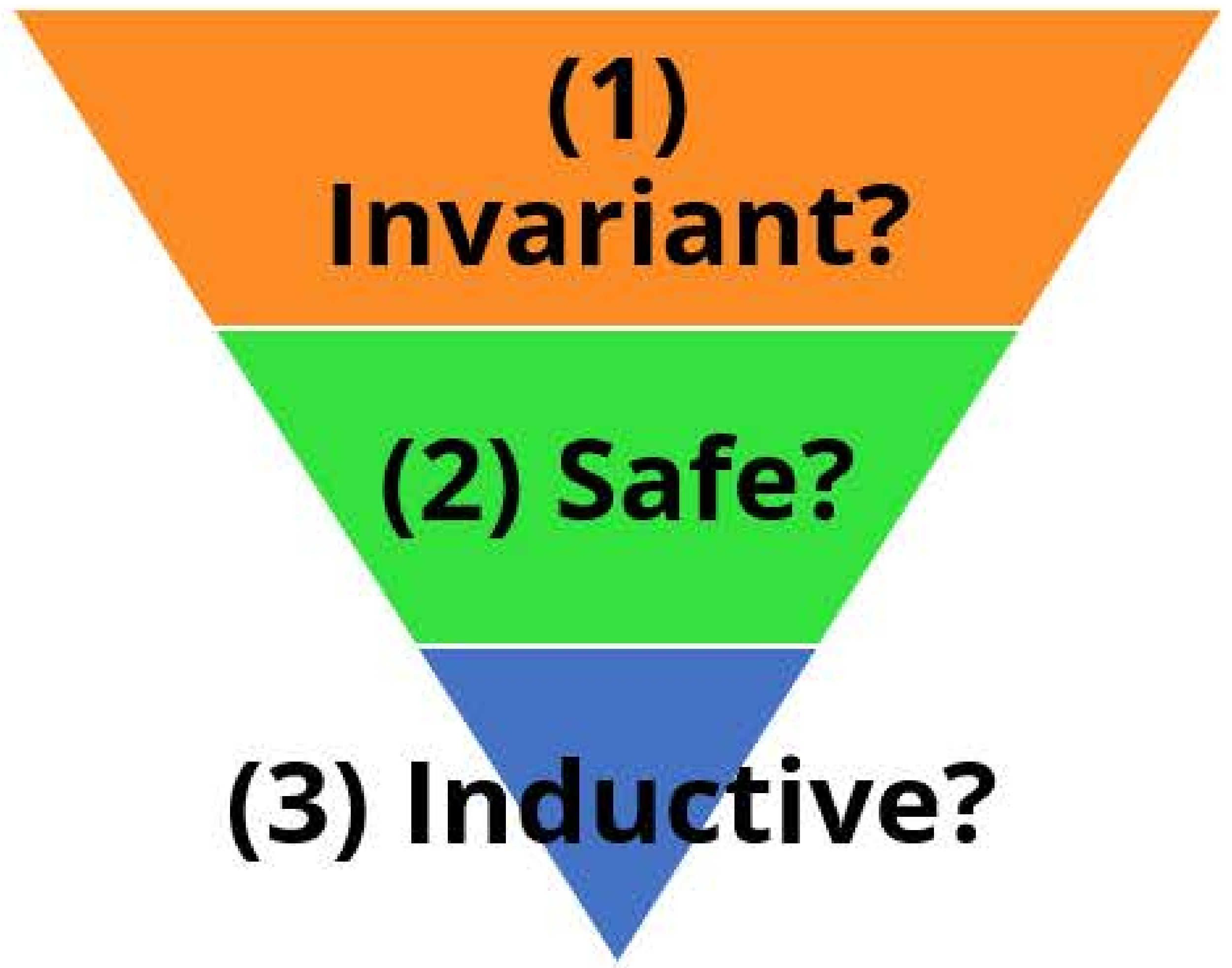


Preliminary Results



- **867** safe Btor2C benchmarks *(superset of SV-COMP)*

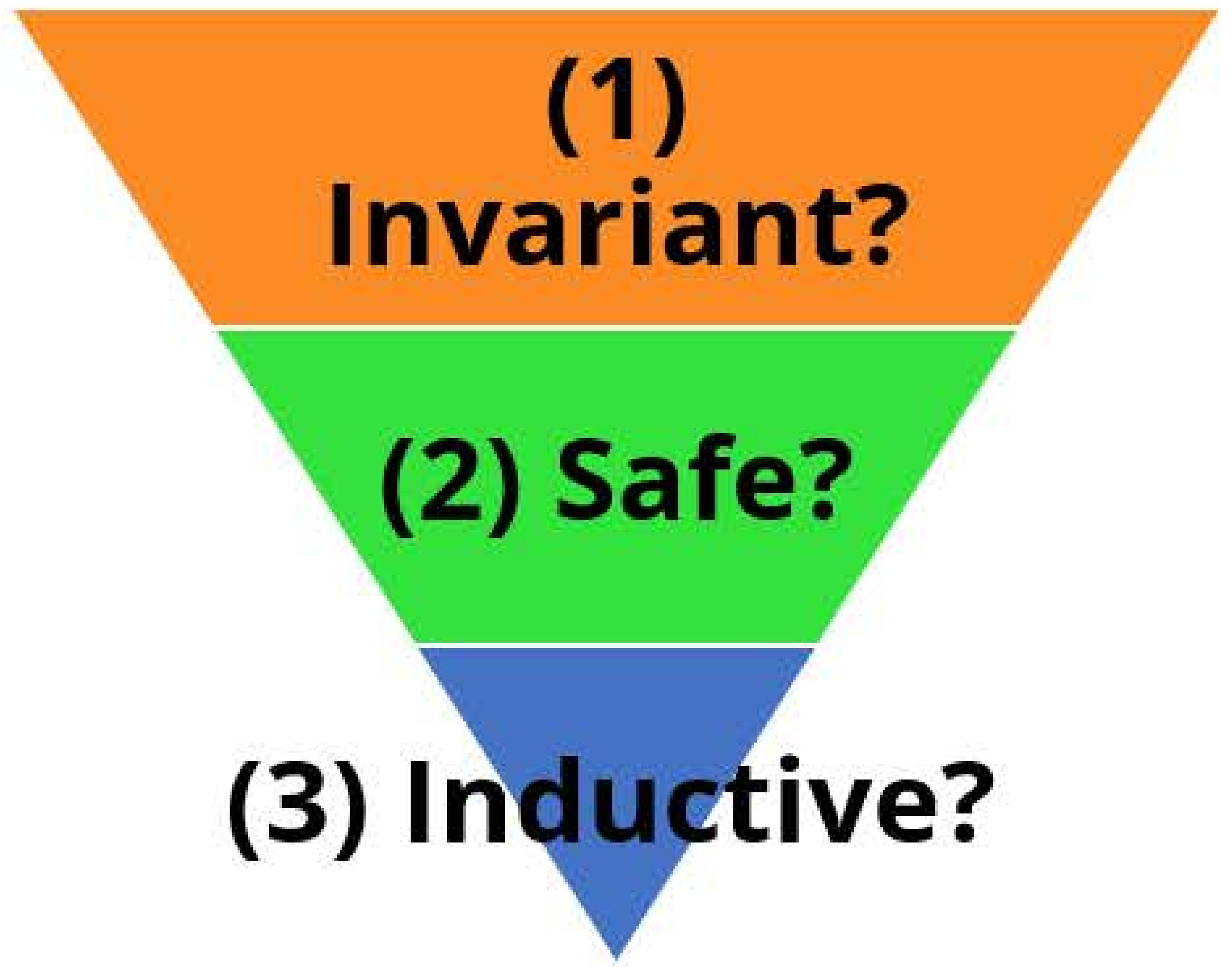
Preliminary Results



- **867** safe Btor2C benchmarks *(superset of SV-COMP)*
- **CPAchecker***
  - Predicate abstraction
- **UAutomizer****
  - Default configuration

* cpachecker.sosy-lab.org
** www.ultimate-pa.org

Preliminary Results



- **867** safe Btor2C benchmarks *(superset of SV-COMP)*
- **CPAchecker***
  - Predicate abstraction
- **UAutomizer****
  - Default configuration
- **90 seconds** for validation of witnesses

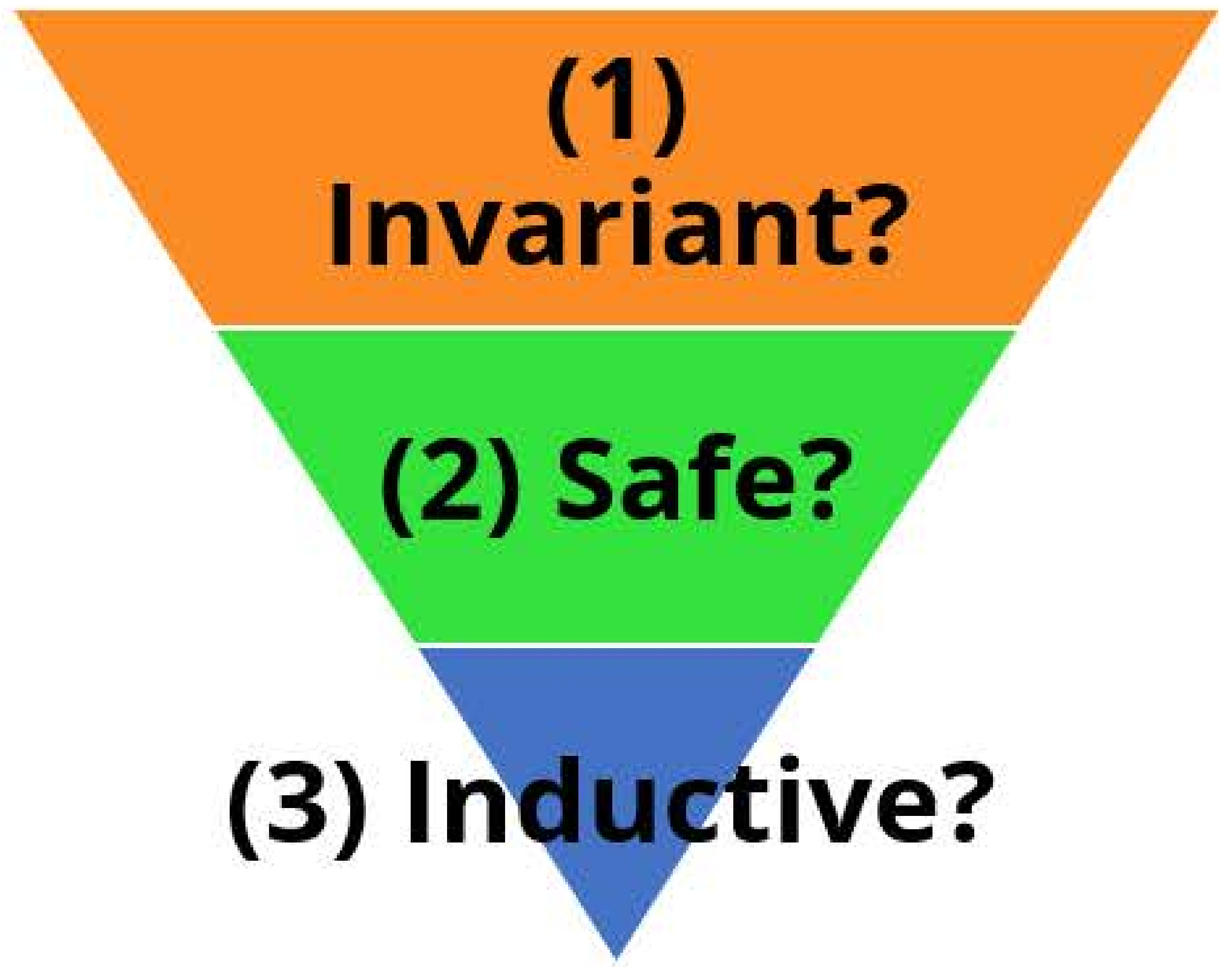
* cpachecker.sosy-lab.org
** www.ultimate-pa.org

Preliminary Results

- **867** safe Btor2C benchmarks *(superset of SV-COMP)*
- **CPAchecker***
  - Predicate abstraction
- **UAutomizer****
  - Default configuration
- **90 seconds** for validation of witnesses

* cpachecker.sosy-lab.org
** www.ultimate-pa.org

	UAutomizer	CPAchecker
No invariant	25	7
No valid invariant	42	0
AVR issue	0	78
Timeout	0	25
(1) Invariant only	0	1
(2) Safe only	0	0
(3) Inductive	12	70



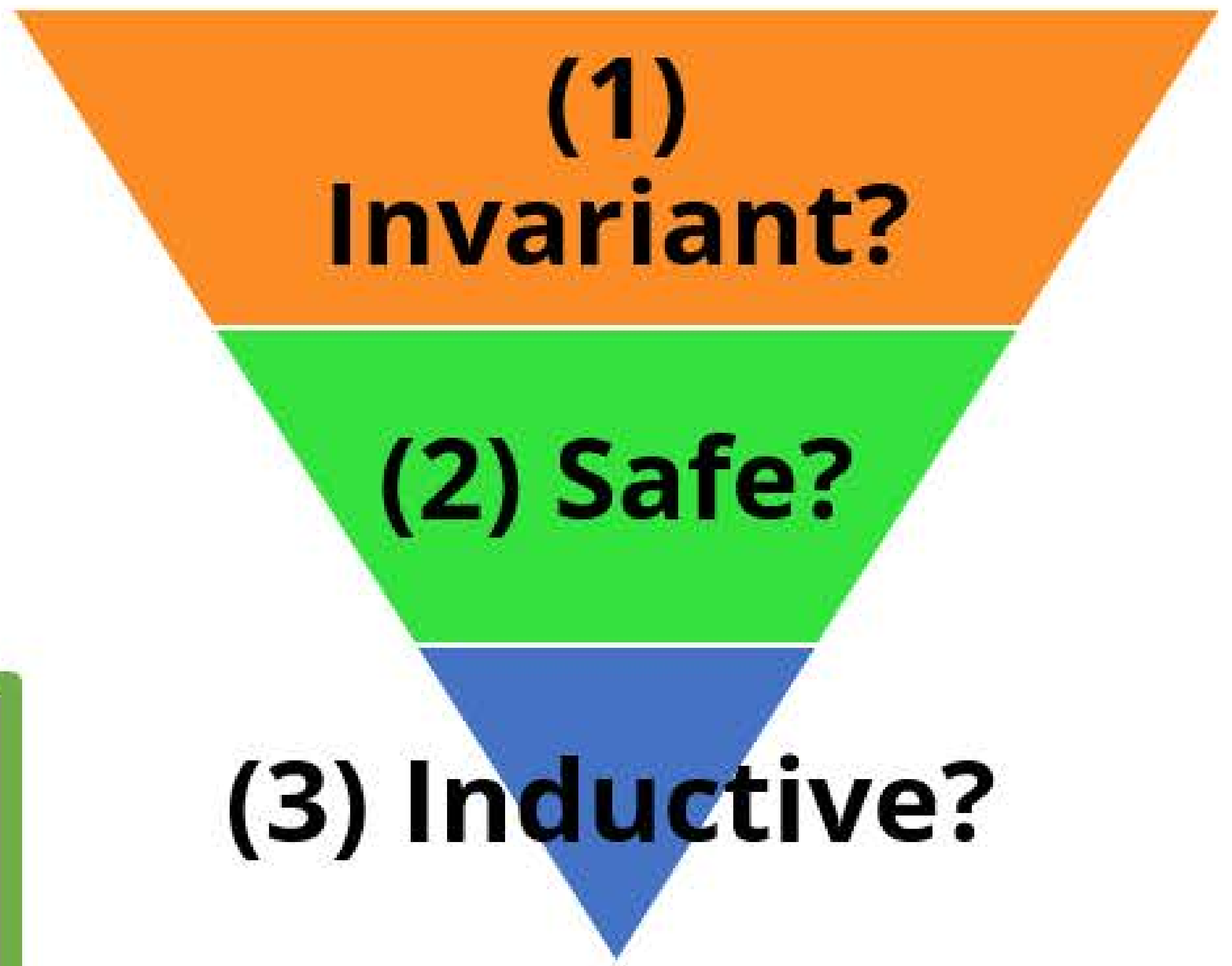
Preliminary Results

- **867** safe Btor2C benchmarks *(superset of SV-COMP)*
- **CPAchecker***
  - Predicate abstraction
- **UAutomizer****
  - Default configuration
- **90 seconds** for validation of witnesses

* cpachecker.sosy-lab.org
** www.ultimate-pa.org

After fixing issues

	UAutomizer	CPAchecker
No invariant	25	7
No valid invariant	42	0
AVR issue	0	78
Timeout	0	25
(1) Invariant only	0	1
(2) Safe only	0	0
(3) Inductive	12	70

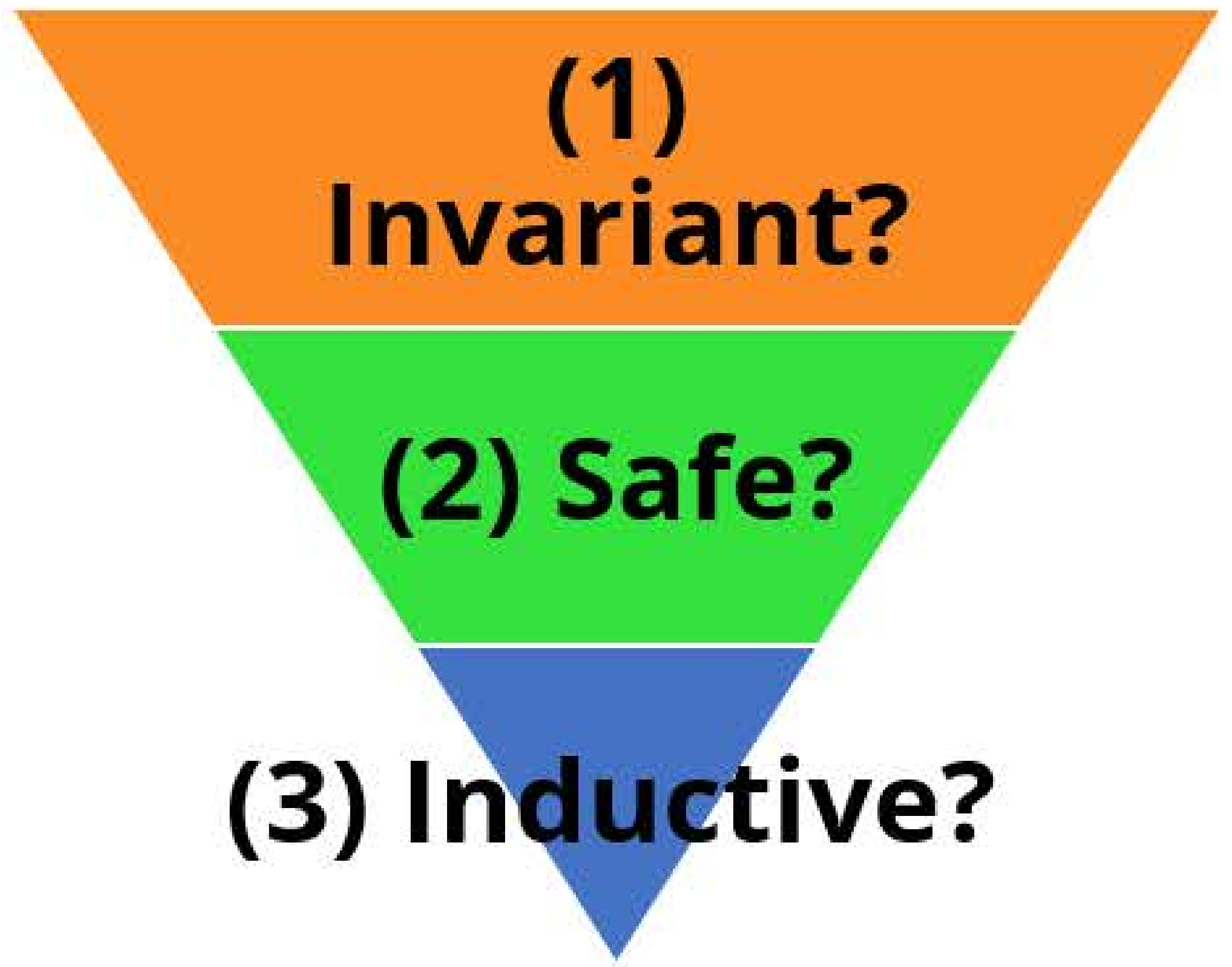


Preliminary Results

- **867** safe Btor2C benchmarks *(superset of SV-COMP)*
- **CPAchecker***
  - Predicate abstraction
- **UAutomizer****
  - Default configuration
- **90 seconds** for validation of witnesses

After fixing issues

	UAutomizer	CPAchecker
No invariant	25	7
No valid invariant	42	0
AVR issue	0	78
Timeout	0	25
(1) Invariant only	0	1
(2) Safe only	0	0
(3) Inductive	12	70



Known AVR issues, huge invariants

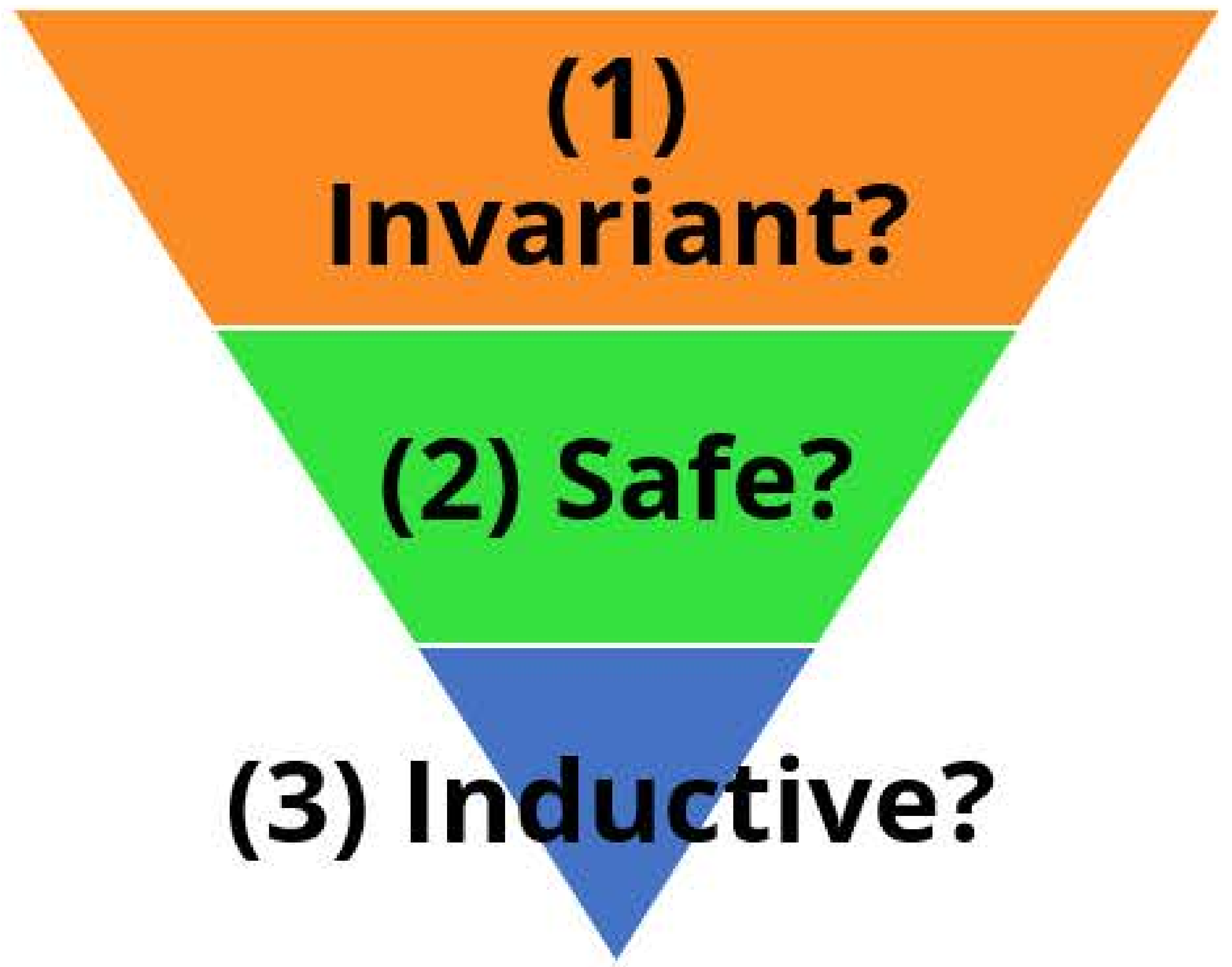
* cpachecker.sosy-lab.org
** www.ultimate-pa.org

Preliminary Results

- **867** safe Btor2C benchmarks *(superset of SV-COMP)*
- **CPAchecker***
  - Predicate abstraction
- **UAutomizer****
  - Default configuration
- **90 seconds** for validation of witnesses

After fixing issues

	UAutomizer	CPAchecker
No invariant	25	7
No valid invariant	42	0
AVR issue	0	78
Timeout	0	25
(1) Invariant only	0	1
(2) Safe only	0	0
(3) Inductive	12	70

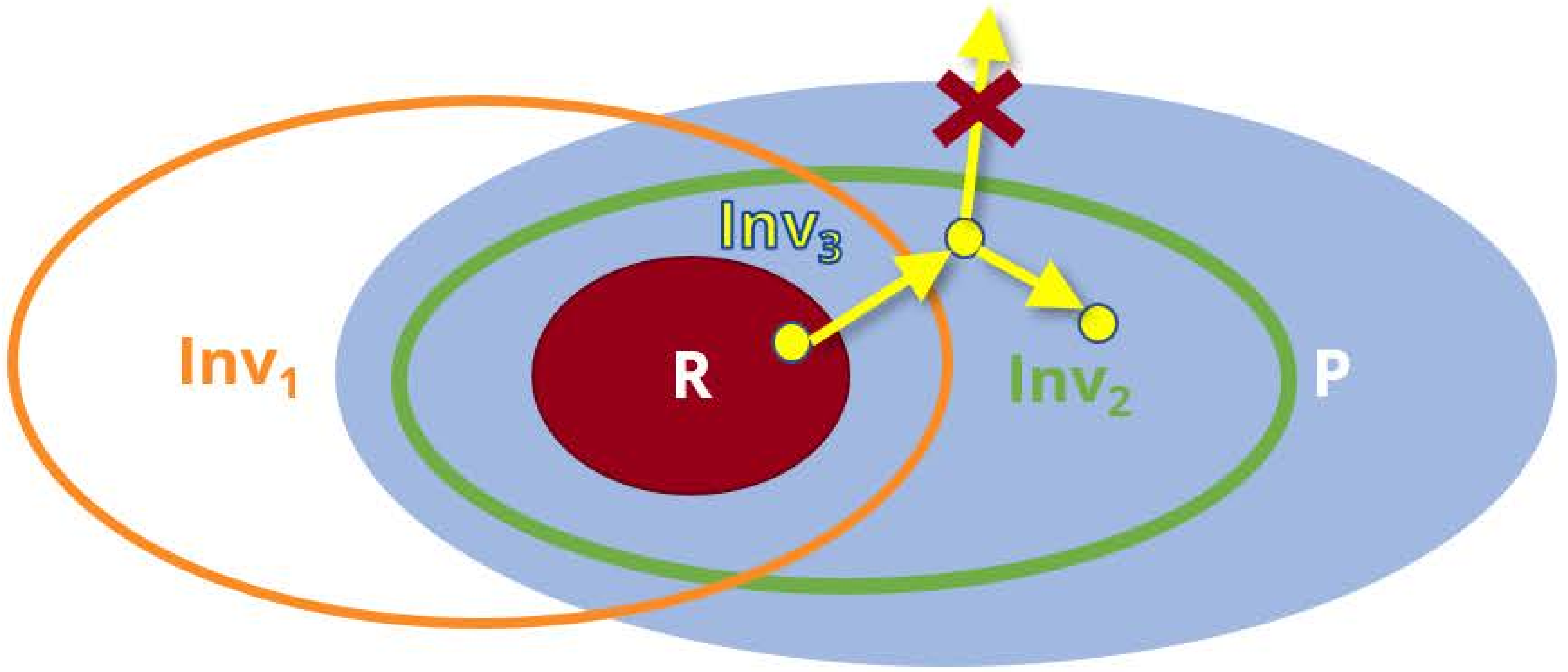


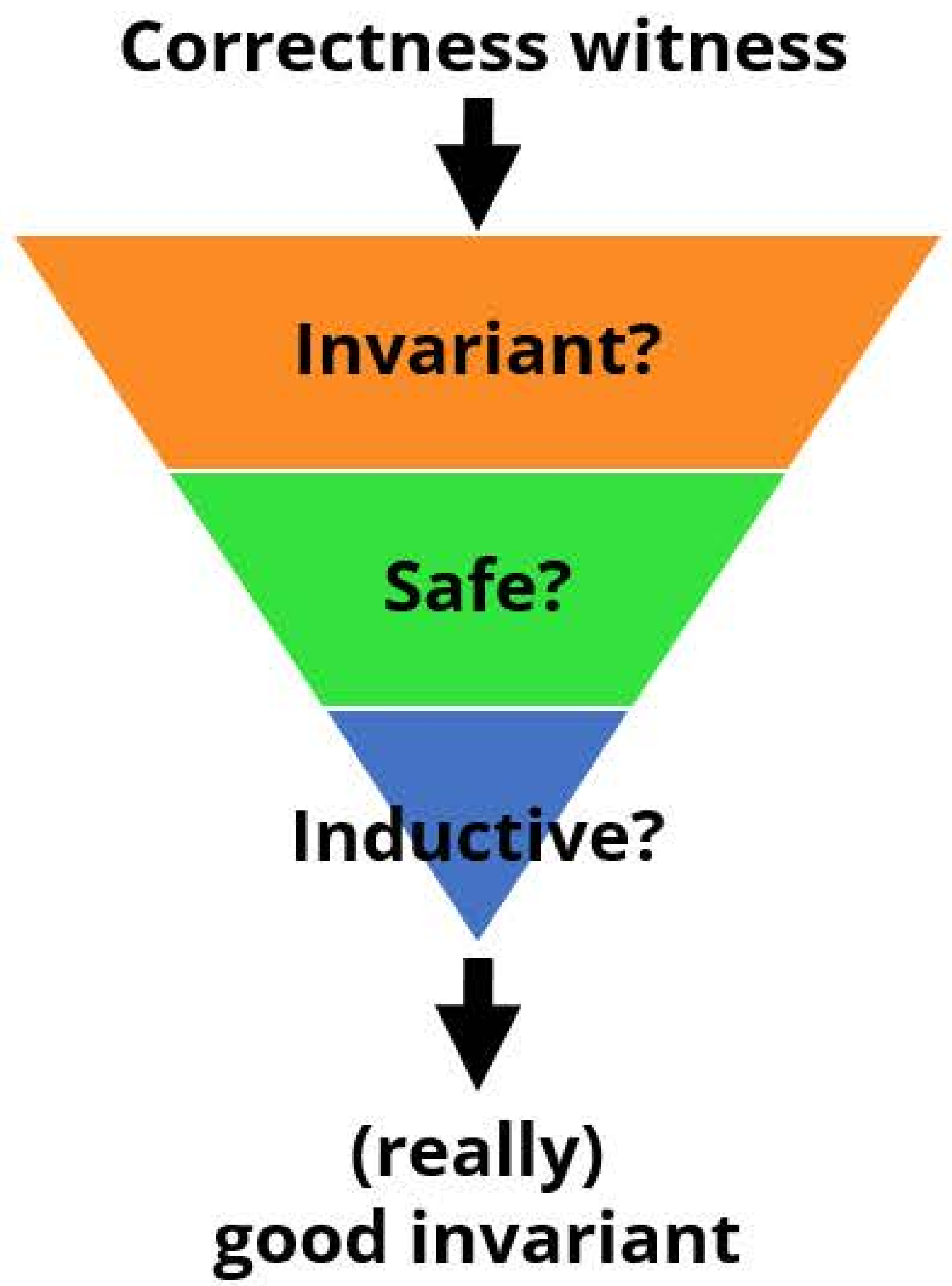
Known AVR issues, huge invariants

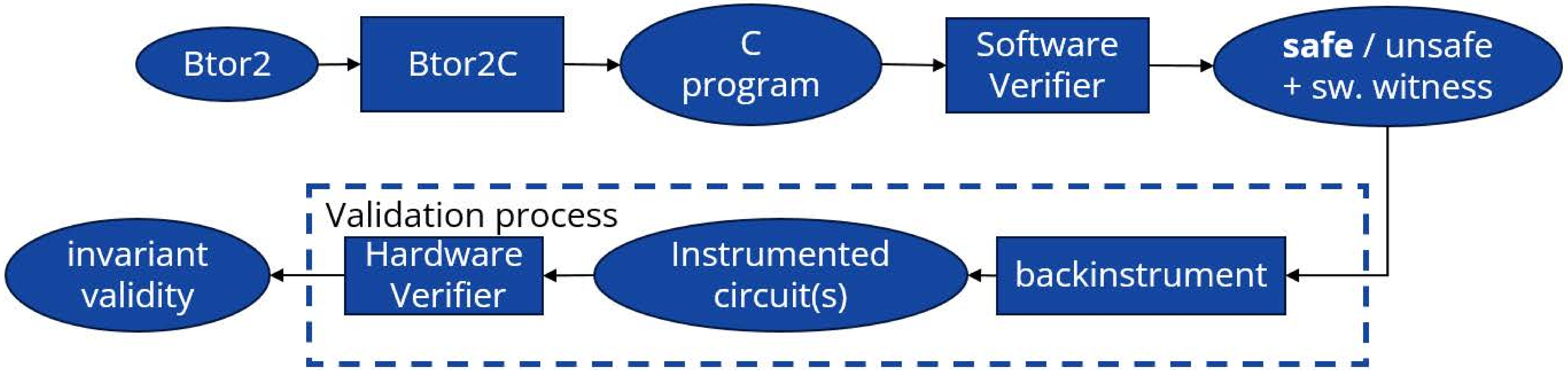
Good quality invariants

* cpachecker.sosy-lab.org
** www.ultimate-pa.org

Conclusion


Inv$_3$
Inv$_1$
R
Inv$_2$
P


Correctness witness
Invariant?
Safe?
Inductive?
(really)
good invariant


Btor2
Btor2C
C program
Software Verifier
safe / unsafe + sw. witness
Validation process
backinstrument
Instrumented circuit(s)
Hardware Verifier
invariant validity

Findings, So Far

- **CPAchecker** (cpachecker.sosy-lab.org)
  - Some **missing and invalid invariants** due to (complex) implementation issues
- **Uautomizer** (www.ultimate-pa.org)
  - Some **invalid** (empty or syntax error) invariants
- **2LS** (github.com/diffblue/2ls)
  - Some **trivial or invalid** invariants
- **Btor2C** (gitlab.com/sosy-lab/software/btor2c)
  - Optimization – corner case issues
- **Some fixed already**

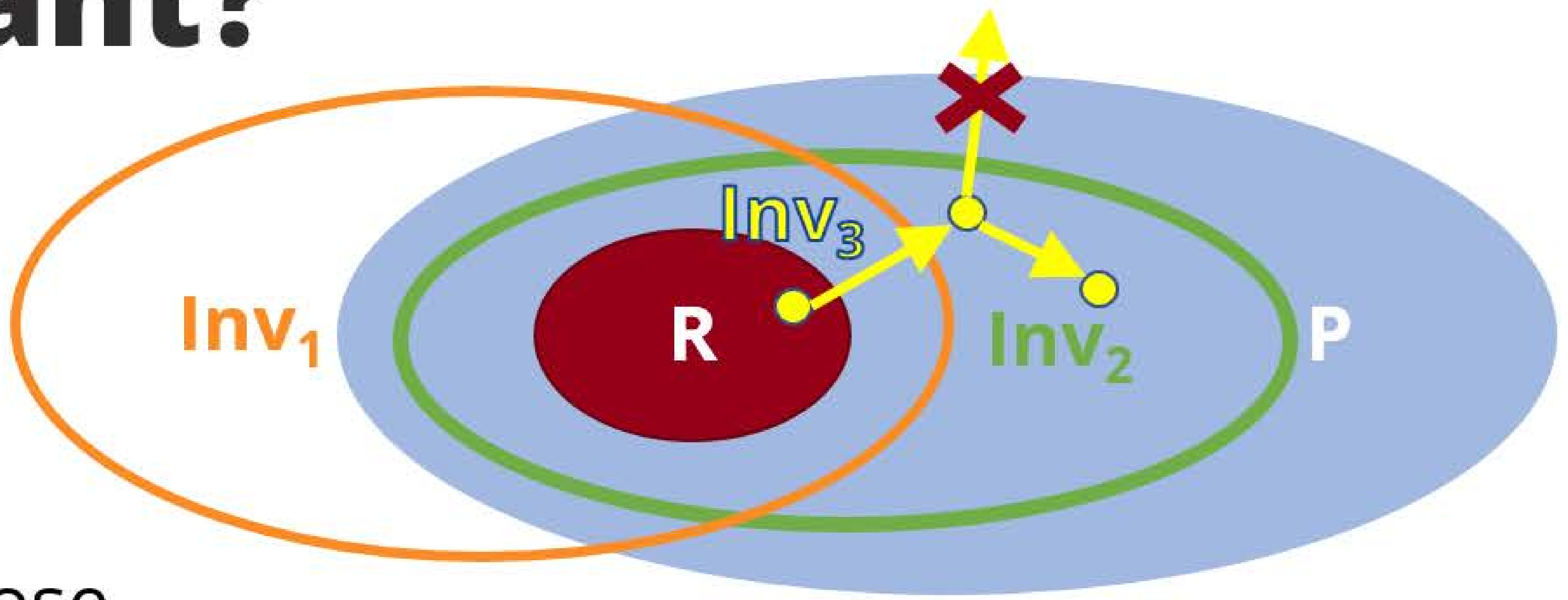
Software Witnesses

- GraphML
- Violation and **Correctness Witness** formats
- Correctness Witness Automaton:
  - Branching, assumptions, enterLoopHead, ...
  - **Invariants**

```
<node id="N42">
<node id="N43">
        <data key="invariant">
        ( ( ( mask_SORT_1 == 1 ) && ( ( state_23 == 200 )
        || ( ( ( ( ( ! ( var_10 == ( mask_SORT_4 & state_6 ) ) )
        && ( ( ( ( mask_SORT_1 & var_18 ) == 0 )
        …
        </data>
        <data key="invariant.scope">main</data>
</node>

<edge source=”N42" target=”N43">
        <data key=”startline">13</data>
        <data key=”enterLoopHead">true</data>
</edge>
```

How good is your invariant?



- **R(s) => P(s) ∧ Inv(s)** (0) *(MetaVal)*
  - „loose" check – allows trivial invariants (⊤)
  - Issue of re-verification
- **R(s) => Inv(s)** - Invariant Check (1)
  - Basic check for correctness witness, still loose
  - Does not prove anything about P, but no re-verification
- **R(s) => Inv(s) ∧ Inv(s) => P(s)** - Safe Invariant Check (2)
  - **Inv(s) => P(s)** is a SAT problem, not model checking
- Inductive Invariant Check (3)

  0) Inv(s) => P(s)

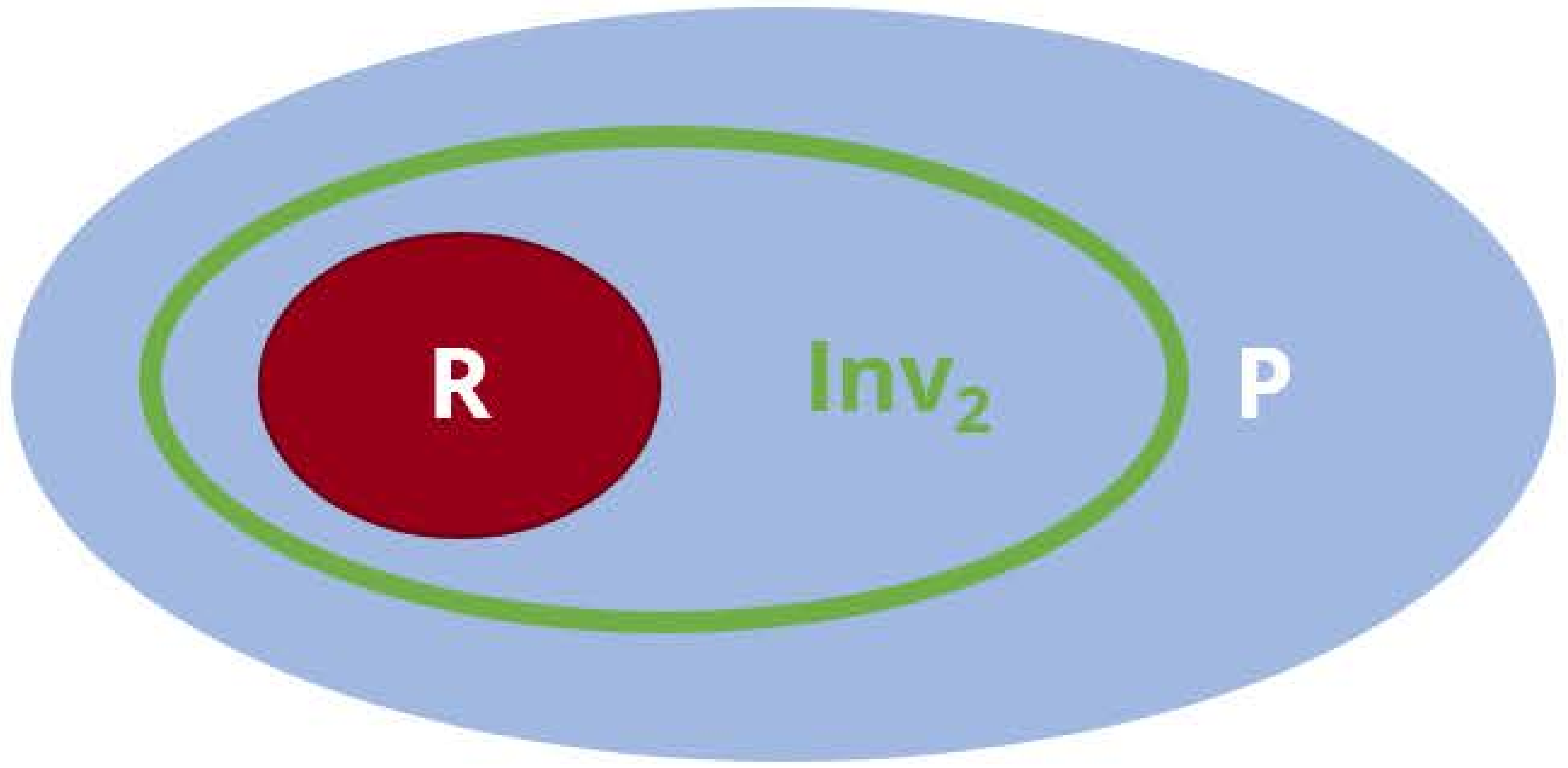
  1) I(s) => Inv(s)

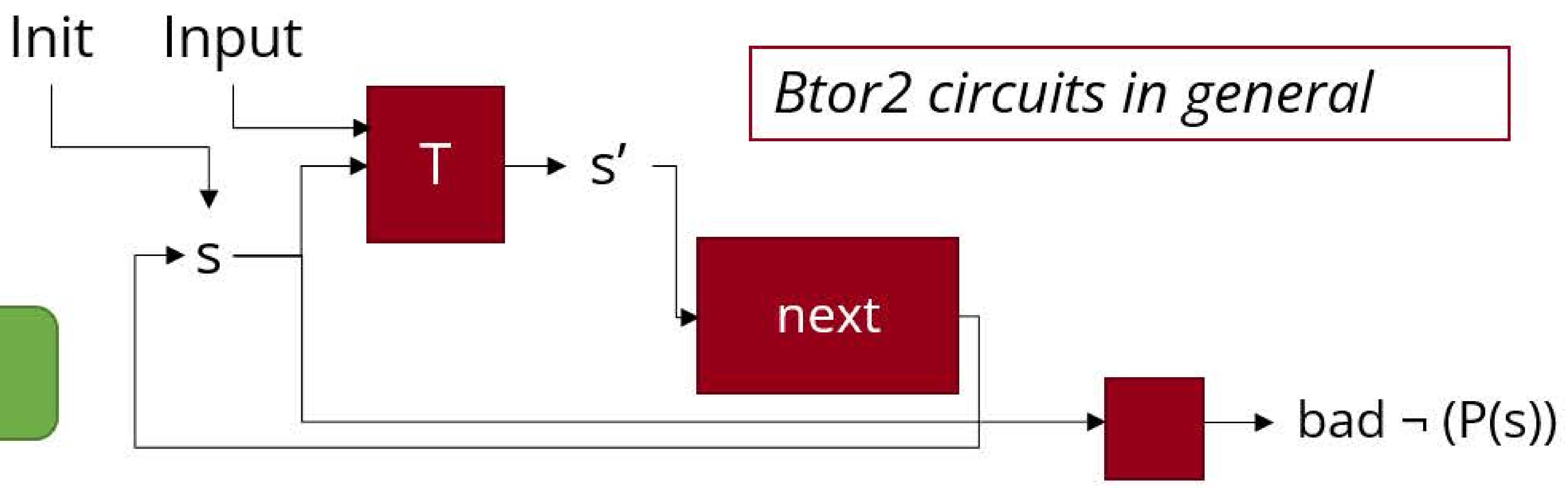
  2) Inv(s) ∧ T(s, s') => Inv(s')

  - Pure SAT solving
  - Mature HW algorithms return inductive invariants *(IMC, PDR)*

(2) Safe Invariant Check

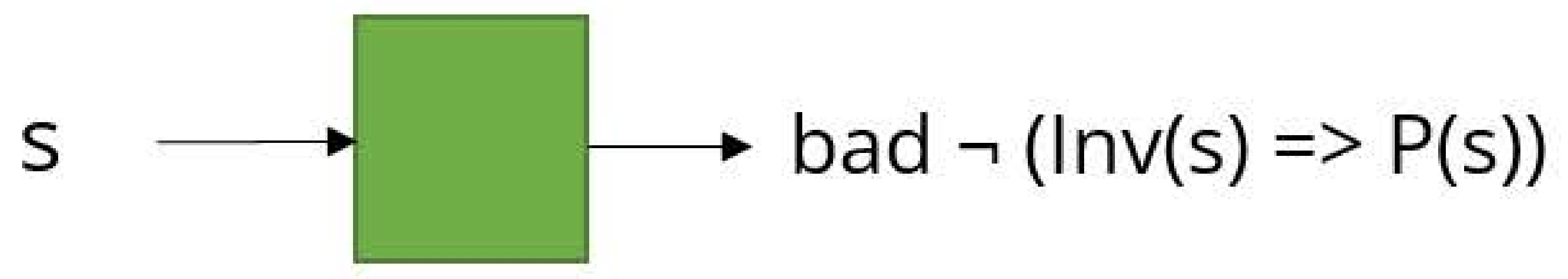
R(s) => Inv(s) ∧ Inv(s) => P(s)
Safe Invariant Check
*(**Inv(s) => P(s)** is SAT solving only)*





How to check

- Do check (1) – **R(s) => Inv(s)**
- **Inv(s) => P(s)**
    - Add new property:
      ¬(Inv(s) => P(s))
    - Remove **next, init**
    - Remove property **¬P(s)**



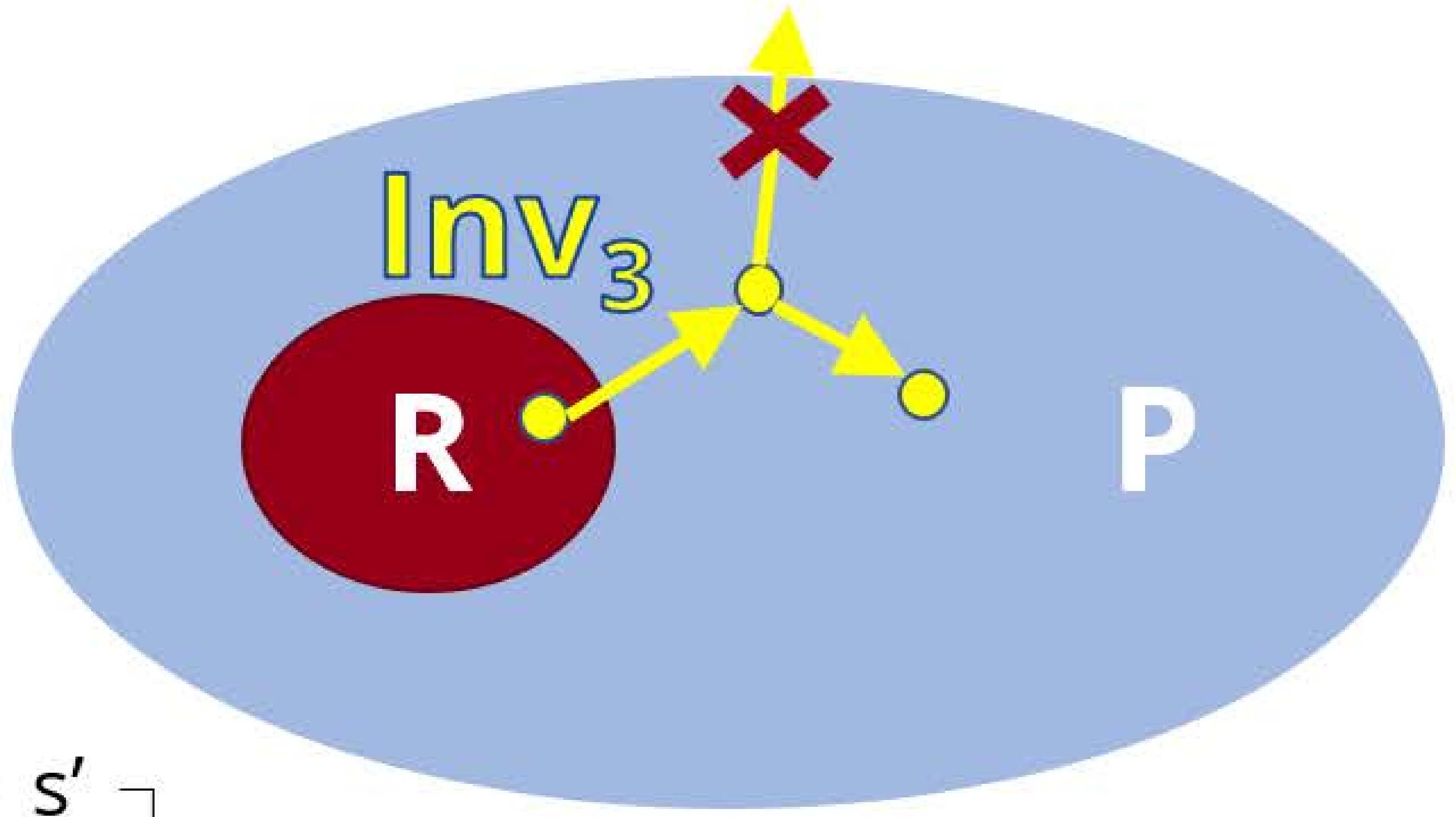
(3) Inductive Invariant Check

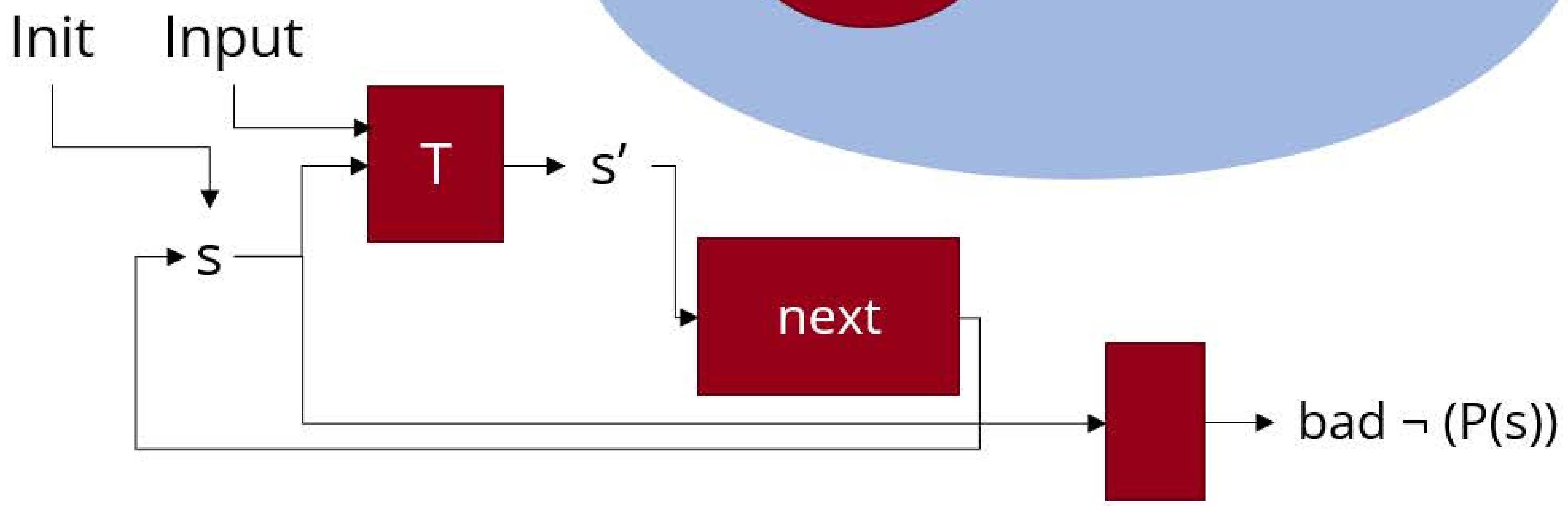
(Safe and) Inductive Invariant Check

0) Inv(s) => P(s)

1) I(s) => Inv(s)

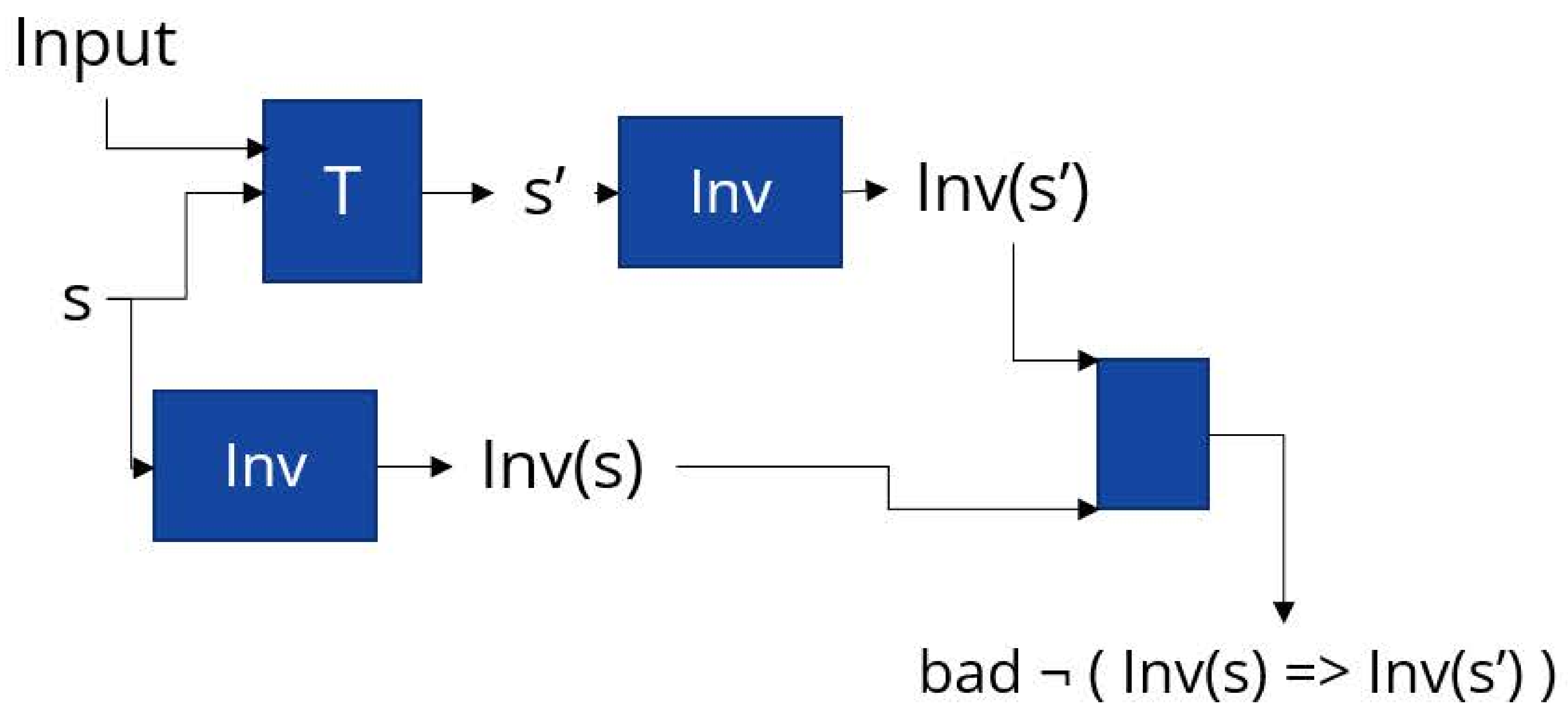
2) Inv(s) ∧ T(s, s') => Inv(s')





How to do

0) see check **(2)**
1) remove **next**,
   change property to **¬Inv(s)**
2) remove **next**, **init**,
   change property to
   ¬ (Inv(s) => Inv(s'))


Input
T
s'
Inv
Inv(s')
s
Inv
Inv(s)
bad ¬ (Inv(s) => Inv(s'))