Experiments on automation of formal verification of devices at the binary level

Thomas Lacroix

INSA Lyon
Soutenance de PFE (Option R&D)

19/06/2019

Section 1

Motivation

Table of Contents

Security critical systems

Privacy

- Smartphones
- Smart TVs

Integrity

- Hospital equipment
- Traffic control systems
- Power plants

Security critical systems

Privacy

- Smartphones
- Smart TVs

Integrity

- Hospital equipment
- Traffic control systems
- Power plants

Problem: complex systems almost always contain bugs

Security critical systems - vulnerable

Security critical systems - vulnerable

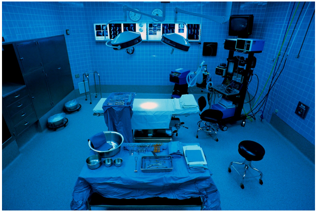


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

Security critical systems - vulnerable

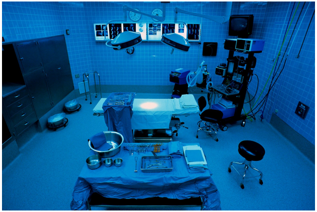


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

- Remote control of equipment

Security critical systems - vulnerable

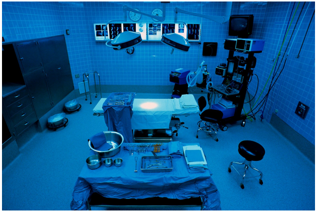


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

Figure: "Remote Exploitation of an Unaltered Passenger Vehicle" [1, 2]

- Remote control of equipment

Security critical systems - vulnerable

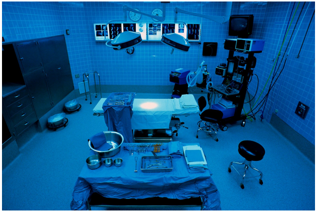


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

- Remote control of equipment

Figure: "Remote Exploitation of an Unaltered Passenger Vehicle" [1, 2]

- Total control of drive systems

Secure operating systems

[1] https://sel4.systems/Info/FAQ/proof.pml

Formal proof[1]:

- The binary code correctly implements its **abstract specification**.
- The specification guarantees **integrity** and **confidentiality**.

[1] https://sel4.systems/Info/FAQ/proof.pml

Formal proof[1]:

- The binary code correctly implements its **abstract specification**.
- The specification guarantees **integrity** and **confidentiality**.

- **Integrity**: data cannot be *changed* without permission.
- **Confidentiality**: data cannot be *read* without permission.

[1] https://sel4.systems/Info/FAQ/proof.pml

Secure operating systems

Proof assumptions[2]:

Security. Performance. Proof.

[2] https://docs.sel4.systems/FrequentlyAskedQuestions#is-sel4-proven-secure

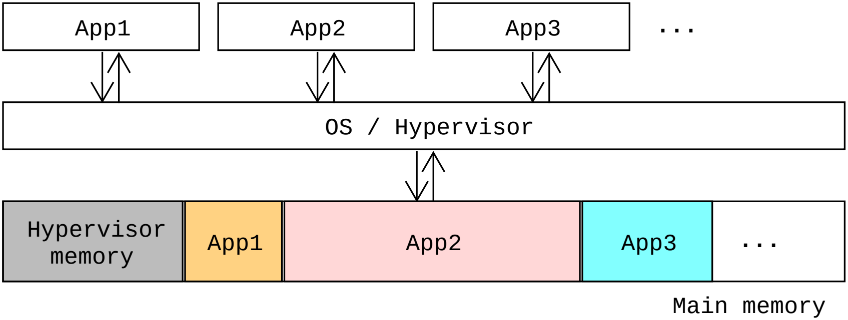
Secure operating systems

Security. Performance. Proof.

Proof assumptions[2]:

- Use of Direct Memory Access (DMA) is excluded, or only allowed for **trusted drivers that have to be formally verified by the user**.

[2] https://docs.sel4.systems/FrequentlyAskedQuestions#is-sel4-proven-secure


App1
App2
App3
...
OS / Hypervisor
Hypervisor memory
App1
App2
App3
...
Main memory

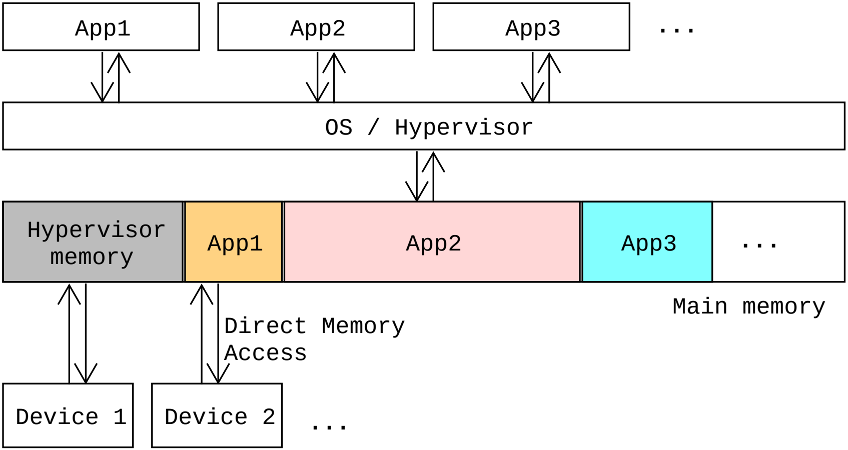

App1
App2
App3
...
OS / Hypervisor
Hypervisor memory
App1
App2
App3
...
Main memory
Direct Memory Access
Device 1
Device 2
...

Table of Contents

System software verification

Objective: show absence of errors in modelisation of real systems

TODO: Ideallement -> implémentation de systems concrets

System software verification

Objective: show absence of errors in modelisation of real systems

TODO: Ideallement -> implémentation de systems concrets

Formal proof

machine checkable proofs using

rigorous semantic

System software verification

Objective: show absence of errors in modelisation of real systems

TODO: Ideallement -> implémentation de systems concrets

Formal proof
machine checkable proofs using rigorous semantic

Non proof-producing verification
specialized programs or procedures that check a given property

Objective: show absence of errors in modelisation of real systems

TODO: Ideallement -> implémentation de systems concrets

Formal proof
machine checkable proofs using rigorous semantic

Use small reliable kernels
$\rightarrow$ produced theorems are trustworthy

Non proof-producing verification
specialized programs or procedures that check a given property

System software verification

Objective: show absence of errors in modelisation of real systems

TODO: Ideallement -> implémentation de systems concrets

Formal proof

machine checkable proofs using rigorous semantic

Use small reliable kernels
$\rightarrow$ produced theorems are trustworthy

Non proof-producing verification

specialized programs or procedures that check a given property

Classic bug-prone software
$\rightarrow$ need tests, less trustworthy

System software verification

Objective: show absence of errors in modelisation of real systems

TODO: Ideallement -> implémentation de systems concrets

Formal proof
machine checkable proofs using rigorous semantic

Use small reliable kernels
$\rightarrow$ produced theorems are trustworthy

Examples: HOL4, Coq, Isabelle

Non proof-producing verification
specialized programs or procedures that check a given property

Classic bug-prone software
$\rightarrow$ need tests, less trustworthy

System software verification

Objective: show absence of errors in modelisation of real systems

TODO: Ideallement -> implémentation de systems concrets

Formal proof

machine checkable proofs using rigorous semantic

Use small reliable kernels
$\rightarrow$ produced theorems are trustworthy

Examples: HOL4, Coq, Isabelle

Non proof-producing verification

specialized programs or procedures that check a given property

Classic bug-prone software
$\rightarrow$ need tests, less trustworthy

SMT solvers, model checkers

Table of Contents

Network Interface Controller (NIC)

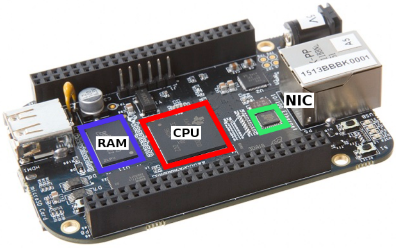
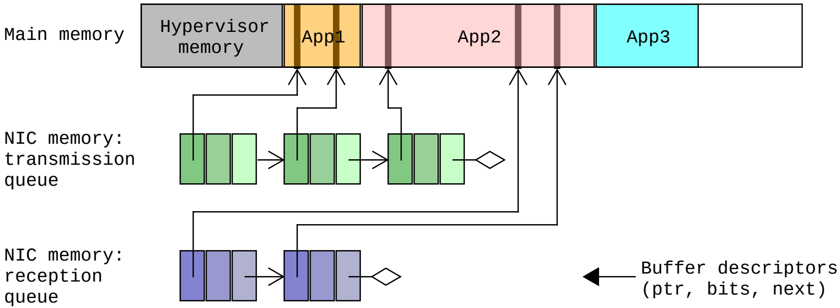
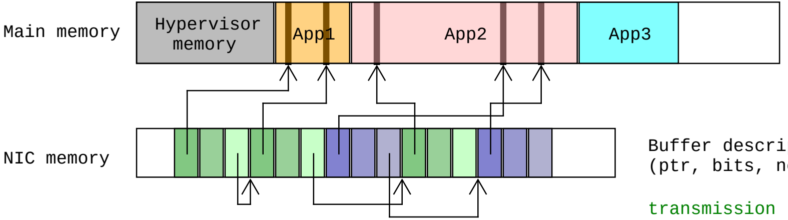


Figure: BeagleBone Black.

NIC: How it works



NIC: How it works

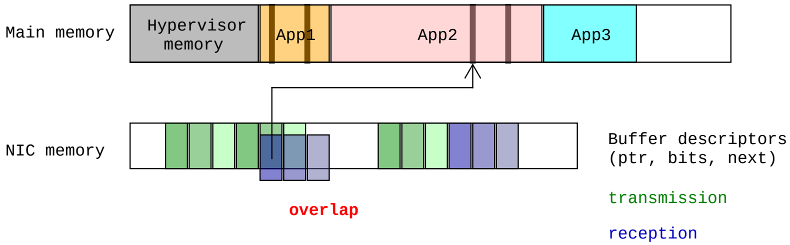


Buffer descriptors
(ptr, bits, next)

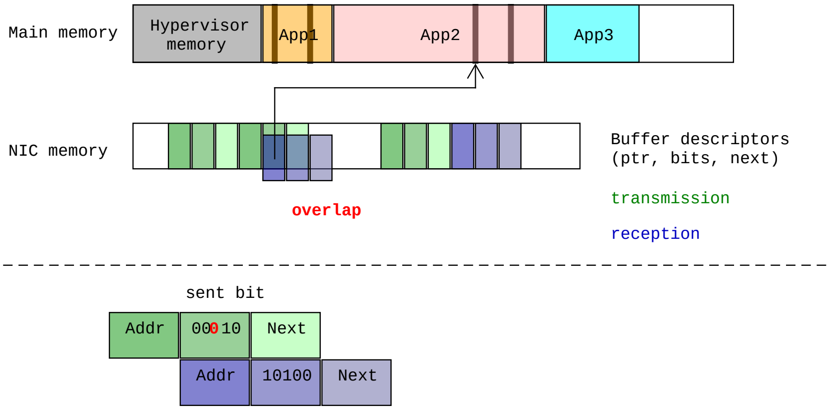
transmission

reception

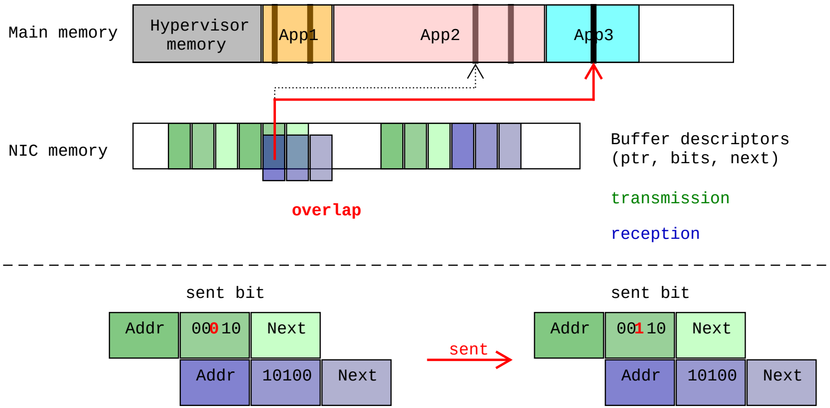
NIC: How it can fail



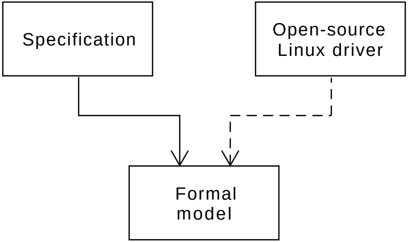
NIC: How it can fail



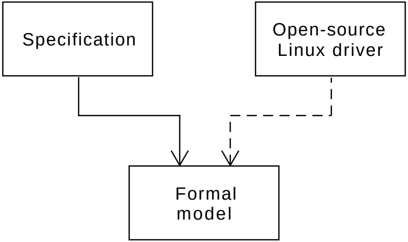
NIC: How it can fail



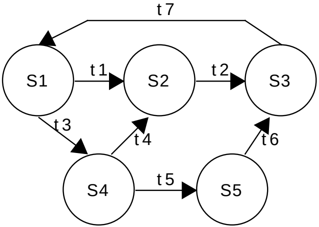
NIC: How it has been modeled



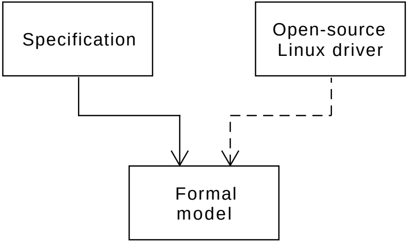
NIC: How it has been modeled



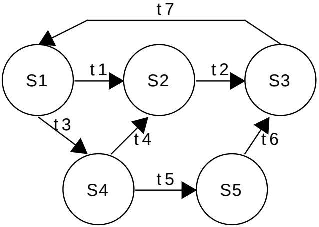
Transition system:



NIC: How it has been modeled



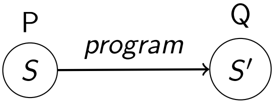
Transition system:



Unspecified behavior → "dead" state

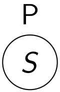
Hoare Triple

Hoare Triple



Hoare Triple

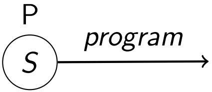
$$\forall S.\ P(S)$$



$$\{P\}$$

Hoare Triple

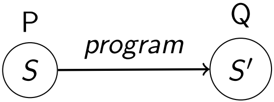
$$\forall S.\ P(S)\ \wedge\ S' = program(S)$$



$$\{P\}\ program$$

Hoare Triple

$$\forall S.\ P(S)\ \wedge\ S' = program(S)\ \implies\ Q(S')$$



$$\{P\}\ program\ \{Q\}$$

Weakest precondition

Weakest precondition *WP* such that:

$$\{WP\}\ program\ \{Q\}$$

Weakest precondition

Weakest precondition WP such that:

$$\{WP\}\ \mathit{program}\ \{Q\}$$

$$\Big(\forall S.\ P(S) \implies WP(S)\Big) \implies \{P\}\ \mathit{program}\ \{Q\}$$

Weakest precondition

Weakest precondition *WP* such that:

$$\{WP\}\ program\ \{Q\}$$

$$\Big(\forall S.\ P(S) \implies WP(S)\Big) \implies \{P\}\ program\ \{Q\}$$

$$WP = f(program,\ Q)$$

NIC: What the verification looks like

NIC: What the verification looks like

Low-level lemmas:

NIC: What the verification looks like

Low-level lemmas:

- $\{\neg dead \wedge well\_configured\}\ transition\ \{\neg dead\}$

NIC: What the verification looks like

Low-level lemmas:

- $\{\neg dead \land well\_configured\}$ $transition$ $\{\neg dead\}$
- $\{\neg overlapping \land \neg cyclic\}$ $transition$ $\{\neg overlapping\}$

NIC: What the verification looks like

Low-level lemmas:

- $\{\neg dead \land well\_configured\}\ transition\ \{\neg dead\}$
- $\{\neg overlapping \land \neg cyclic\}\ transition\ \{\neg overlapping\}$
- $\{\neg overlapping \land \neg cyclic\}\ transition\ \{\neg cyclic\}$

NIC: What the verification looks like

Low-level lemmas:

- $\{\neg dead \wedge well\_configured\}$ $transition$ $\{\neg dead\}$
- $\{\neg overlapping \wedge \neg cyclic\}$ $transition$ $\{\neg overlapping\}$
- $\{\neg overlapping \wedge \neg cyclic\}$ $transition$ $\{\neg cyclic\}$

Intermediate lemmas:

- $Invariant$: $rx\_invariant\_well\_defined$
- $Invariant$: $tx\_invariant\_well\_defined$

NIC: What the verification looks like

Low-level lemmas:

- $\{\neg dead \wedge well\_configured\}$ $transition$ $\{\neg dead\}$
- $\{\neg overlapping \wedge \neg cyclic\}$ $transition$ $\{\neg overlapping\}$
- $\{\neg overlapping \wedge \neg cyclic\}$ $transition$ $\{\neg cyclic\}$

Intermediate lemmas:

- $Invariant$: $rx\_invariant\_well\_defined$
- $Invariant$: $tx\_invariant\_well\_defined$

Security theorems:

- $\forall\ tx\_bd.\ readable(tx\_bd)$ $BD = Buffer\ Descriptor$
- $\forall\ rx\_bd.\ writable(rx\_bd)$

NIC: What the verification looks like

Low-level lemmas:

- $\{\neg dead \wedge well\_configured\}\ transition\ \{\neg dead\}$
- $\{\neg overlapping \wedge \neg cyclic\}\ transition\ \{\neg overlapping\}$
- $\{\neg overlapping \wedge \neg cyclic\}\ transition\ \{\neg cyclic\}$

Intermediate lemmas:

- *Invariant*: $rx\_invariant\_well\_defined$
- *Invariant*: $tx\_invariant\_well\_defined$

Security theorems:

- $\forall\ tx\_bd.\ readable(tx\_bd)$ $\quad BD = Buffer\ Descriptor$
- $\forall\ rx\_bd.\ writable(rx\_bd)$

Research question

Can we apply traditional software verification techniques and tools to show security properties of hardware devices?

HolBA: HOL4 Binary Analysis platform

- Verification platform at binary level
- Centered around its Intermediate Language, BIR
- Features proof-producing tools
  - Weakest precondition generation

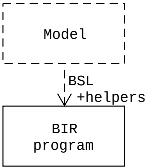
Section 2

Contract-based verification

Table of Contents

Contract-based verification pipeline

0. Translate the model in BIR

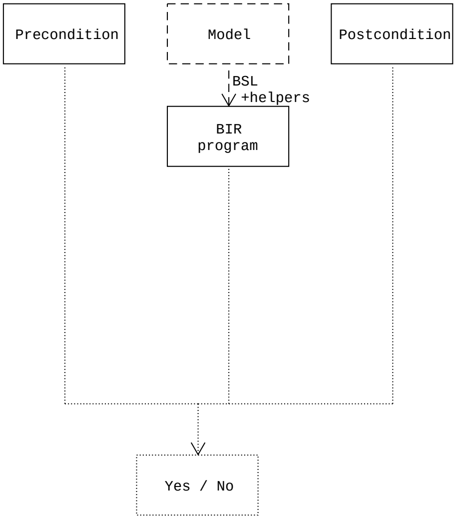


$transition_{BIR}$ **TODO: unique SMT-ready de l'implication**

Contract-based verification pipeline

0. Translate the model in BIR
1. Formulate a Hoare Triple

$\{P\}$ $transition_{BIR}$ $\{Q\}$ **TODO: unique SMT-ready de l'implication**

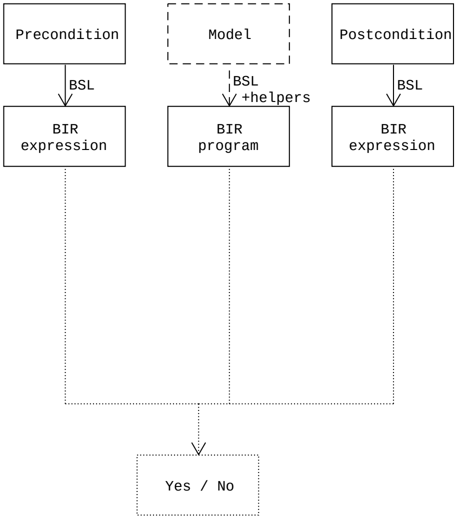


Contract-based verification pipeline

0. Translate the model in BIR
1. Formulate a Hoare Triple
2. Translate P and Q to BIR

$$\{P_{BIR}\}\ transition_{BIR}\ \{Q_{BIR}\}$$

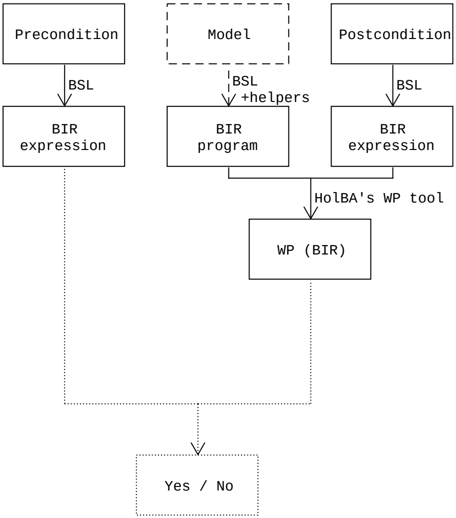
TODO: unique SMT-ready de l'implication



Contract-based verification pipeline

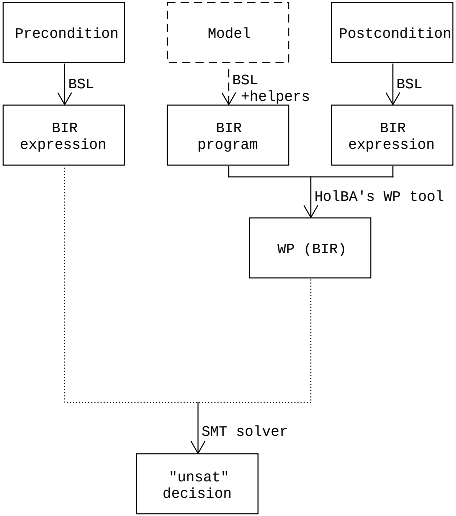
0. Translate the model in BIR
1. Formulate a Hoare Triple
2. Translate P and Q to BIR
3. Generate the WP

$P_{BIR}(S) \implies WP_{BIR}(S)$ **TODO: unique SMT-ready de l'implication**



Contract-based verification pipeline

0. Translate the model in BIR
1. Formulate a Hoare Triple
2. Translate P and Q to BIR
3. Generate the WP



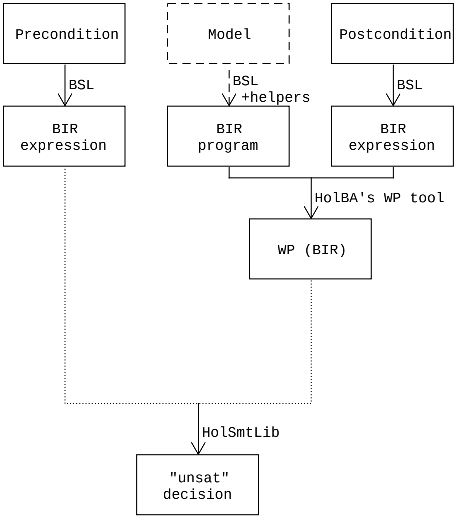
Satisfiability Modulo Theories

- external tools
- SMT-LIB 2.0

TODO: unique SMT-ready de l'implication

Contract-based verification pipeline

0. Translate the model in BIR
1. Formulate a Hoare Triple
2. Translate P and Q to BIR
3. Generate the WP



$$\neg\Big(P_{BIR}(S) \implies WP_{BIR}(S)\Big)$$

"unsat"? **TODO: unique SMT-ready de l'implication**

Contract-based verification pipeline

0. Translate the model in BIR
1. Formulate a Hoare Triple
2. Translate P and Q to BIR
3. Generate the WP
4. Translate the goal into a SMT-compatible expression

$$\neg\Big(P(S) \implies WP(S)\Big)_{SMT}$$

"unsat"? **TODO: unique SMT-ready de l'implication**

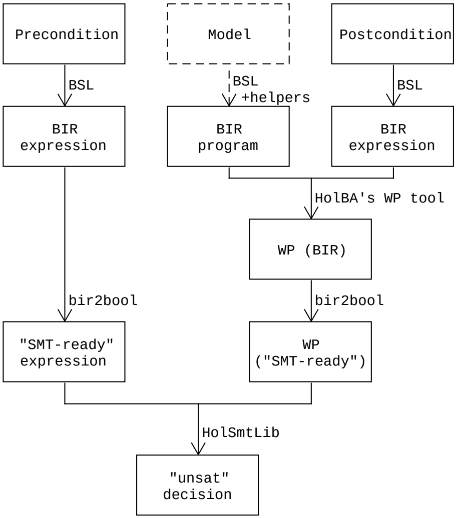
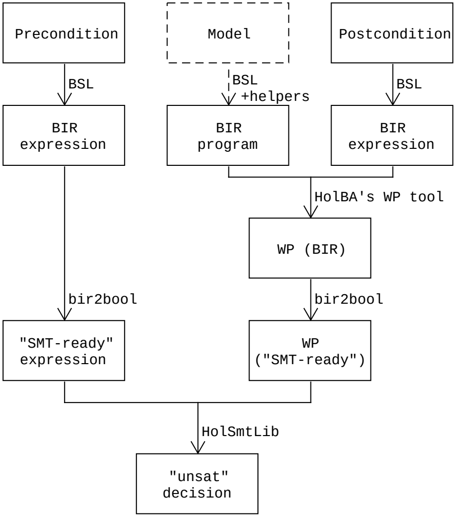


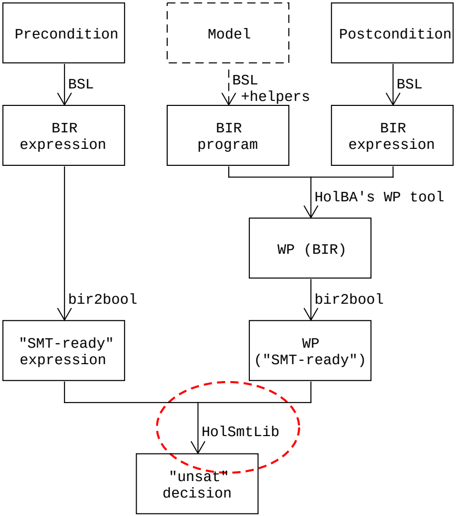
Table of Contents

How trustful is it?



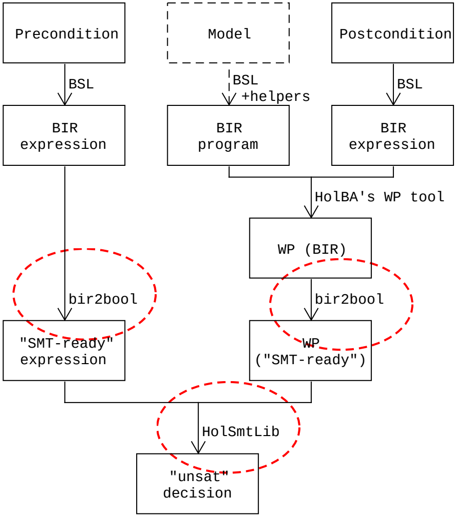
How trustful is it?

- SMT solvers don't produce proofs



How trustful is it?

- SMT solvers don't produce proofs
- bir2bool isn't proof-producing



How trustful is it?

- SMT solvers don't produce proofs
- bir2bool isn't proof-producing
- The BIR model may be wrong

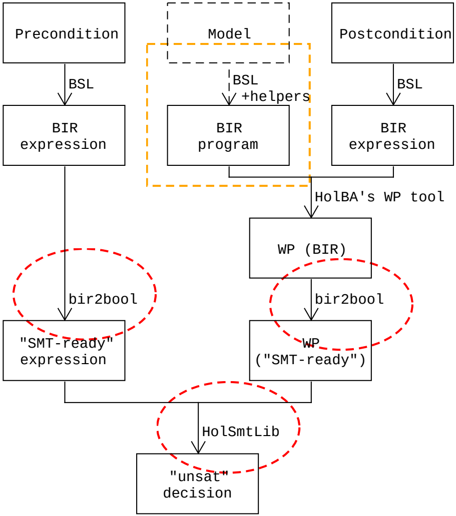


Table of Contents

How powerful is it?

How powerful is it?

Not proof-producing

Easier non-proof producing platforms exist

How powerful is it?

Not proof-producing

Easier non-proof producing platforms exist

Limited by SMT solvers' logics

How powerful is it?

Not proof-producing

Easier non-proof producing platforms exist

Limited by SMT solvers' logics

- $\{\neg \mathit{overlapping} \land \neg \mathit{cyclic}\}\ \mathit{transition}\ \{\neg \mathit{overlapping}\}$

How powerful is it?

Not proof-producing

Easier non-proof producing platforms exist

Limited by SMT solvers' logics

- $\{\neg overlapping \land \neg cyclic\}$ *transition* $\{\neg overlapping\}$
- SMT logic: **QF**_AUFBV → **Q**uantifier-**F**ree

How powerful is it?

Not proof-producing

Easier non-proof producing platforms exist

Limited by SMT solvers' logics

- $\{\neg \mathit{overlapping} \wedge \neg \mathit{cyclic}\}$ *transition* $\{\neg \mathit{overlapping}\}$
- SMT logic: **QF**_AUFBV → **Q**uantifier-**F**ree

Cannot compose theorems

How powerful is it?

Not proof-producing

Easier non-proof producing platforms exist

Limited by SMT solvers' logics

- $\{\neg overlapping \wedge \neg cyclic\}$ *transition* $\{\neg overlapping\}$
- SMT logic: **QF**_AUFBV $\rightarrow$ **Q**uantifier-**F**ree

Cannot compose theorems

- Partly a limitation of HolBA (work in progress)

Section 3

Proof-producing verification

Goal

Some theorems cannot be proved with previous pipeline
We would like to prove them anyway

Section 4

Conclusion

Questions

References I

Andy Greenberg.
Hackers remotely kill a jeep on the highway—with me in it.

Dr Charlie Miller and Chris Valasek.
Remote exploitation of an unaltered passenger vehicle.
page 91.

Thomas Tuerk.
Interactive theorem proving (ITP) course.

Kim Zetter.
It's insanely easy to hack hospital equipment.

Other tools for software verification

TODO: Jonas' MT, page 46 Section 2.5.4