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

Thomas Lacroix

INSA Lyon
Soutenance de PFE (Option R&D)

Wednesday, June 19, 2019

Section 1

Motivation

Table of Contents

Security critical systems - vulnerable

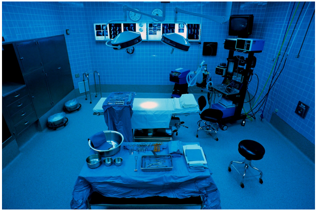


Figure: "It's Insanely Easy to Hack Hospital Equipment" [7]

Figure: "Remote Exploitation of an Unaltered Passenger Vehicle" [3, 5]

Security critical systems - vulnerable

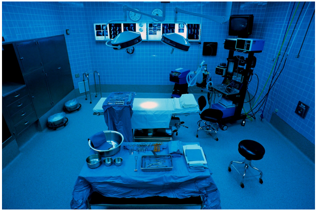


Figure: "It's Insanely Easy to Hack Hospital Equipment" [7]

Figure: "Remote Exploitation of an Unaltered Passenger Vehicle" [3, 5]

Problem: complex systems almost always contain bugs

Secure execution platforms

seL4

Security. Performance. Proof.

INTEGRITY MULTIVISOR

MINIX 3

Microsoft

Hyper-V

Secure execution platforms

Secure execution platforms

Security. Performance. Proof.

Formal proof [2]:

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

Secure execution platforms

Formal proof [2]:

- 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.

Secure operating systems

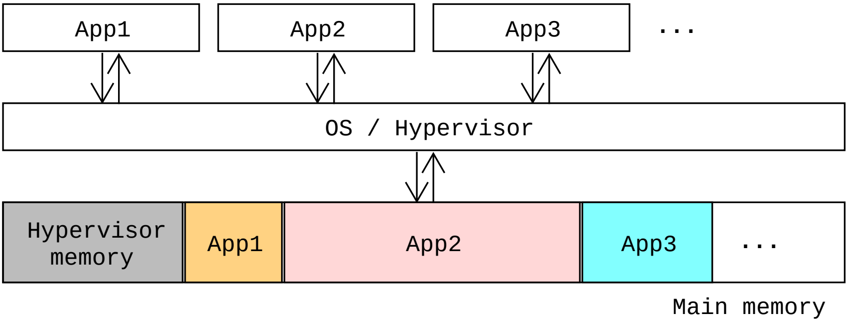
Proof assumptions [1]:

Secure operating systems

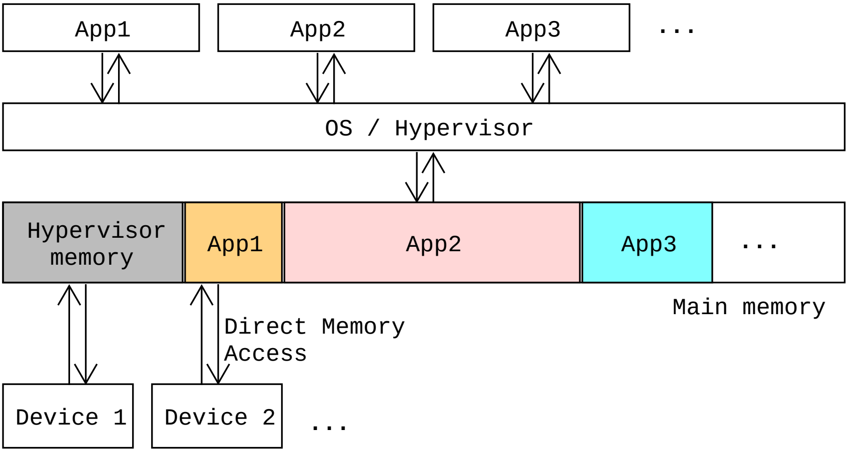
Proof assumptions [1]:

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

What is DMA?


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

What is DMA?



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

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof
machine checkable proofs using
rigorous semantic

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

Examples: HOL4, Coq, Isabelle

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof
machine checkable proofs using rigorous semantic

Use small reliable kernels
→ 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
→ need tests, less trustworthy

SMT solvers, model checkers

Table of Contents

Network Interface Controller (NIC)

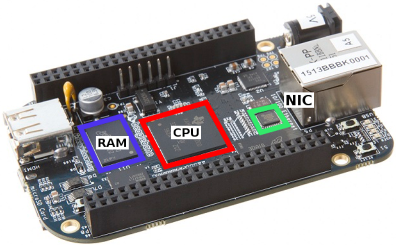
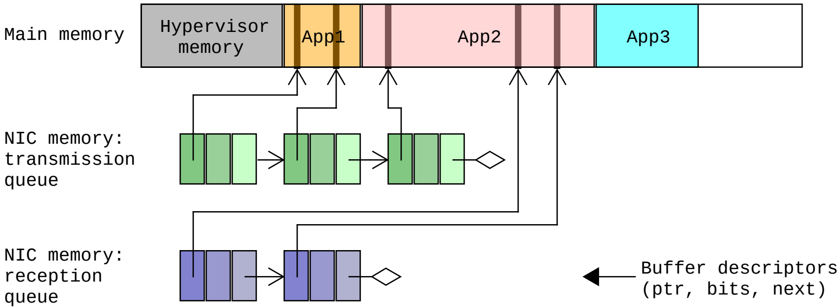
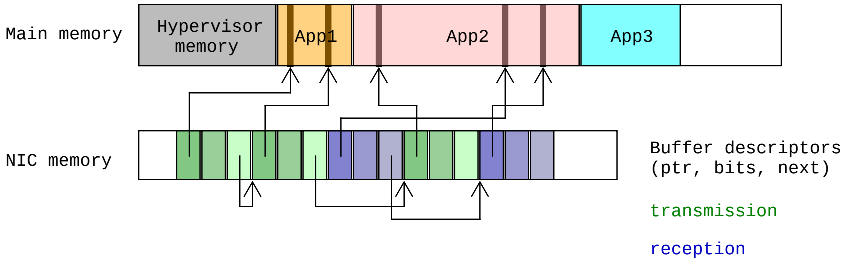


Figure: BeagleBone Black.

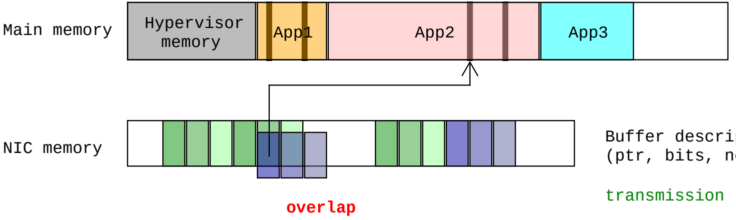
NIC: How it works



NIC: How it works



NIC: How it can fail

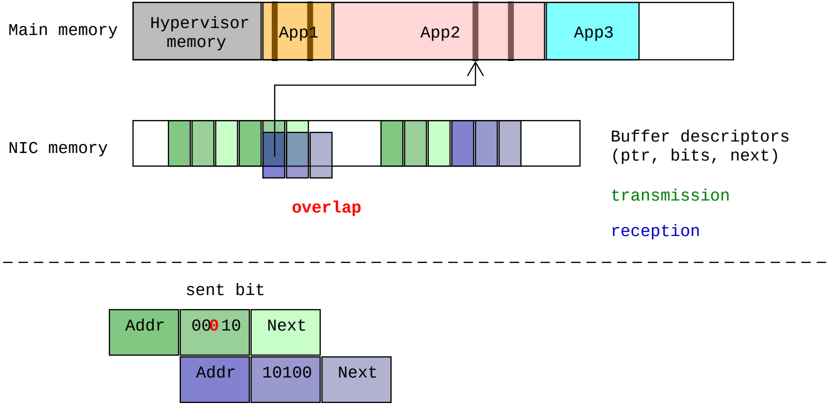


Buffer descriptors
(ptr, bits, next)

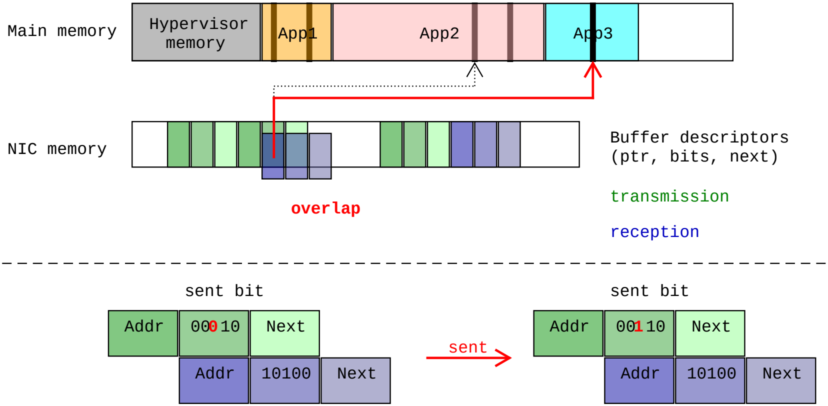
transmission

reception

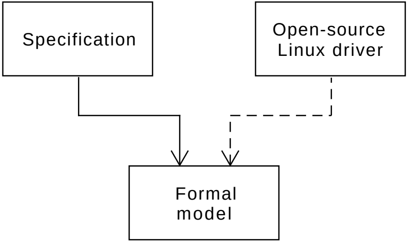
NIC: How it can fail



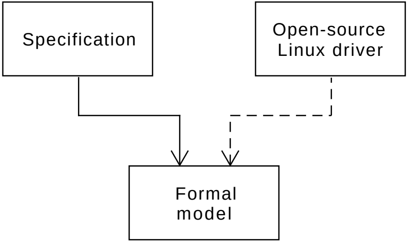
NIC: How it can fail



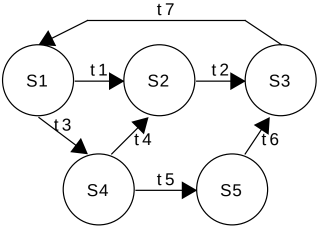
NIC: How it has been modeled [4]



NIC: How it has been modeled [4]

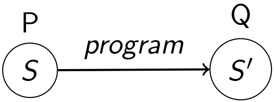


Transition system:



Hoare Triple

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



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

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 [4]

NIC: What the verification looks like [4]

Low-level lemmas:

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

NIC: What the verification looks like [4]

Low-level lemmas:

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

Intermediate lemmas:

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

NIC: What the verification looks like [4]

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 [4]

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*
- $\forall\ rx\_bd.\ writable(rx\_bd)$

Research question

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

HolBA: HOL4 Binary Analysis platform

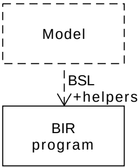
- Verification platform at binary level
- Centered around its Intermediate Language, BIR
- Features proof-producing tools
  - Weakest precondition generation

Section 2

Automatic contract-based verification

Table of Contents

0. Translate the model in BIR

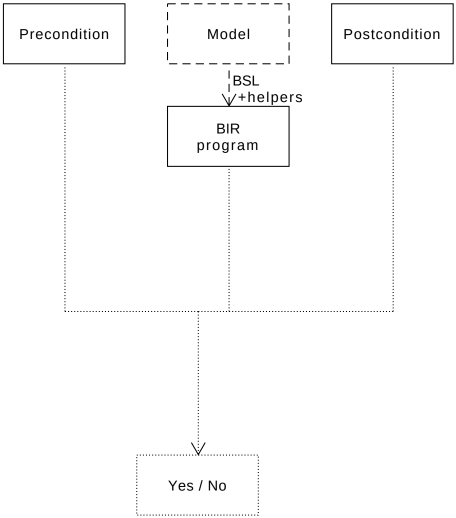


$transition_{BIR}$

Contract-based verification pipeline

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

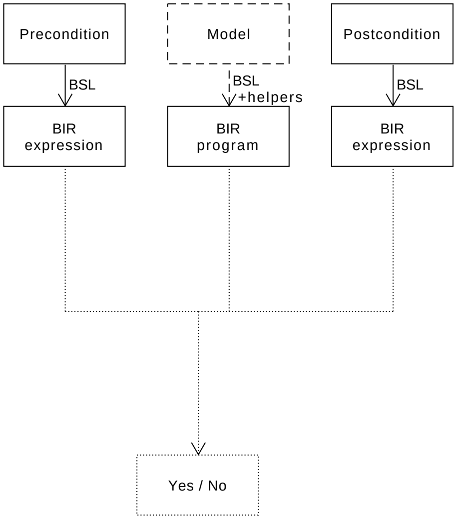
$$\{P\}\ transition_{BIR}\ \{Q\}$$



Contract-based verification pipeline

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

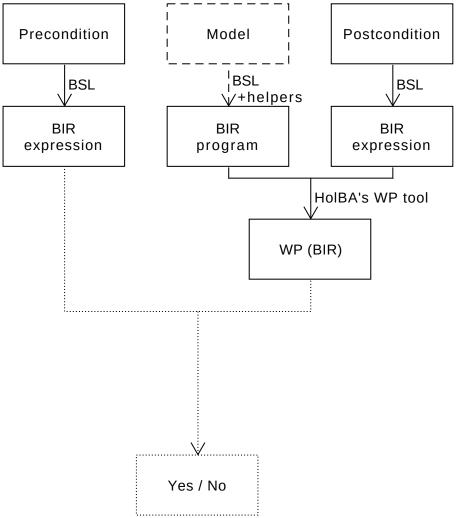
$$\{P_{BIR}\}\ transition_{BIR}\ \{Q_{BIR}\}$$



Contract-based verification pipeline

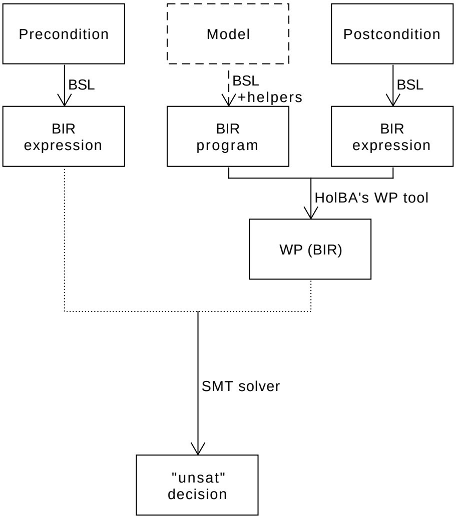
0. Translate the model in BIR
1. Formulate a Hoare Triple
2. Translate P and Q to BIR
3. Generate the WP

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



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 **M**odulo **T**heories

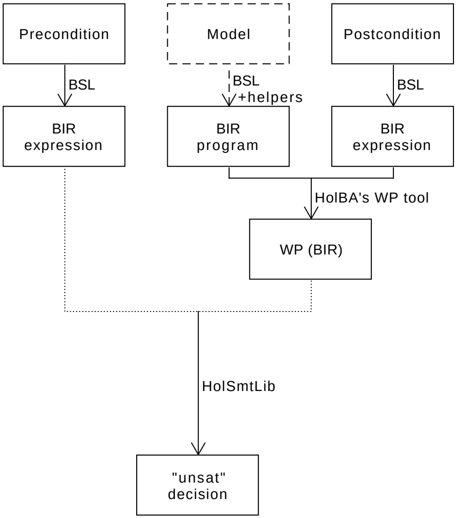
- external tools
- SMT-LIB 2.0

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

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

"unsat"?



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"?

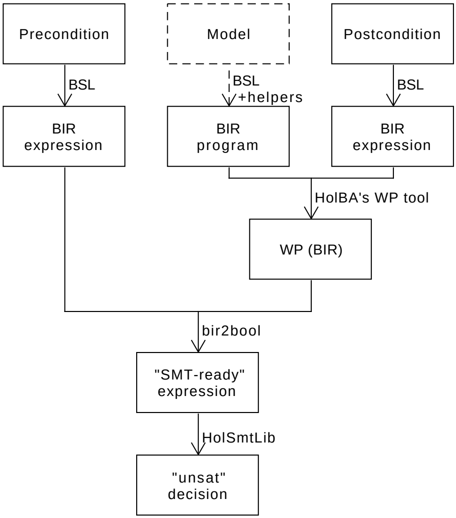
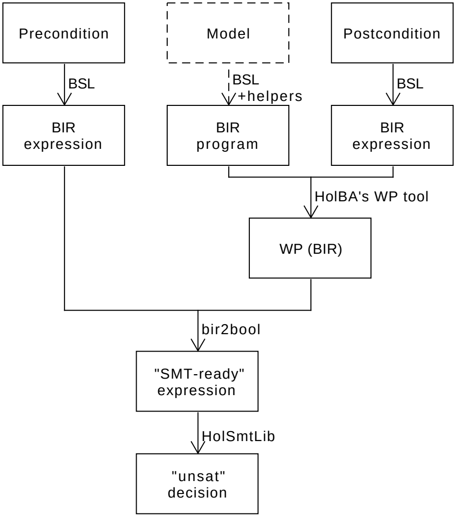


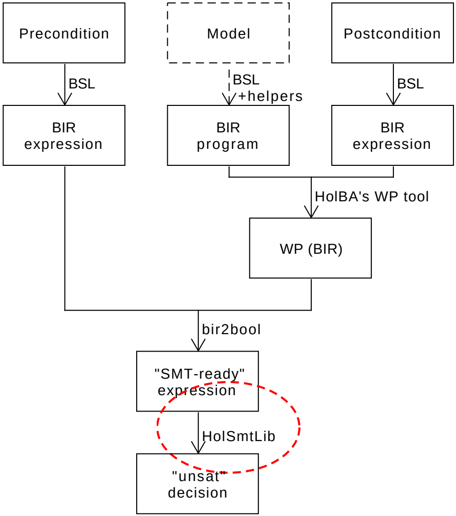
Table of Contents

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

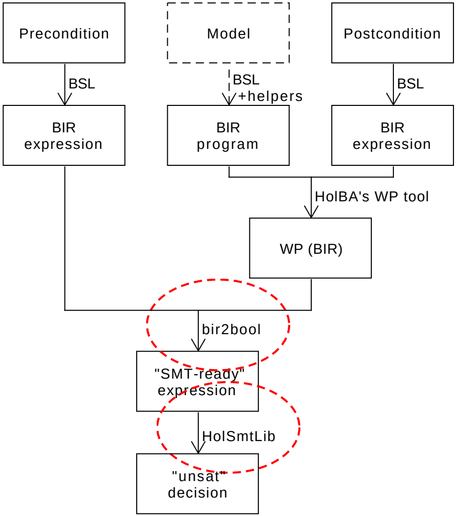


Table of Contents

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

- $\{\neg overlapping \wedge \neg cyclic\}$ *transition* $\{\neg overlapping\}$

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

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 $\rightarrow$ **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

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

Cannot compose theorems

- Work in progress in HolBA

Section 3

Proof-producing verification

Goal

Goal

→ Some theorems cannot be proved with previous pipeline

Goal

→ Some theorems cannot be proved with previous pipeline

→ We would like to prove them anyway

Goal

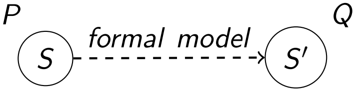
- → Some theorems cannot be proved with previous pipeline
- → We would like to prove them anyway
- → We want to use them with the current proof

Goal

→ Some theorems cannot be proved with previous pipeline

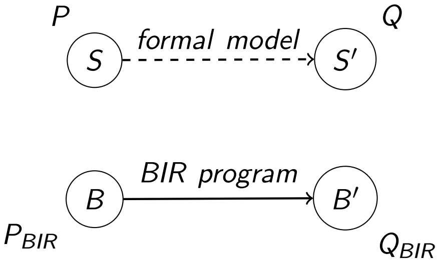
→ We would like to prove them anyway

→ We want to use them with the current proof



Goal

- → Some theorems cannot be proved with previous pipeline
- → We would like to prove them anyway
- → We want to use them with the current proof

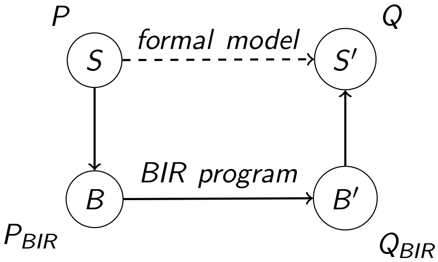


Goal

→ Some theorems cannot be proved with previous pipeline

→ We would like to prove them anyway

→ We want to use them with the current proof

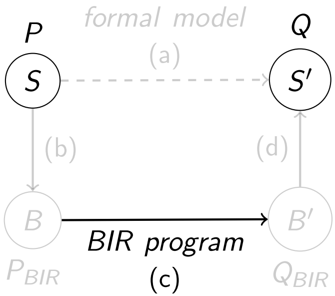


Goal

$$\{P\}\ BIR\_prog\ \{Q\}$$

$$\equiv$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \implies \textbf{P}\ S \implies \textbf{Q}\ S'$$



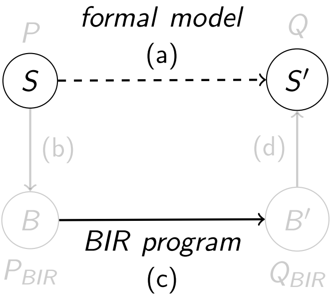
Goal

$$\{P\}\ BIR\_prog\ \{Q\}$$

$$\equiv$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \implies \textbf{P}\ S \implies \textbf{Q}\ S'$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=} \forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B \wedge \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$$



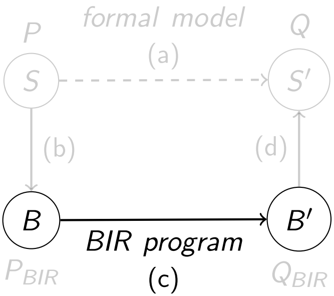
Goal

$$\{P\}\ BIR\_prog\ \{Q\}$$

$$\equiv$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \implies \textbf{P}\ S \implies \textbf{Q}\ S'$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=} \forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B \land \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$$



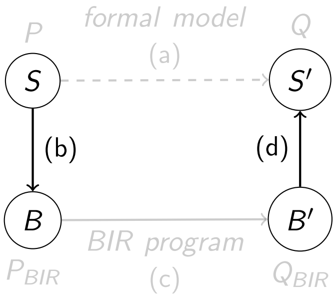
Goal

$$\{P\}\ BIR\_prog\ \{Q\}$$

$$\equiv$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \implies \textbf{P}\ S \implies \textbf{Q}\ S'$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=} \forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B \wedge \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$$



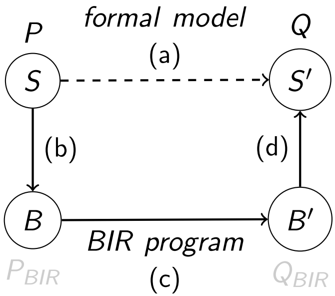
Goal

$$\{P\}\ BIR\_prog\ \{Q\}$$

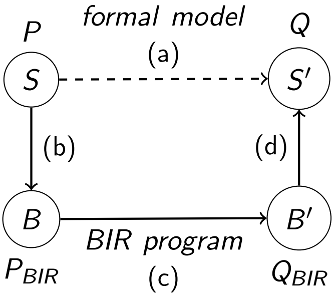
$$\equiv$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \implies \textbf{P}\ S \implies \textbf{Q}\ S'$$

$$\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=} \forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B \wedge \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$$

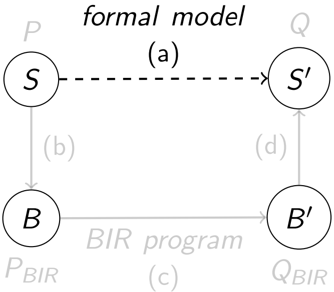


Proof overview



Proof overview

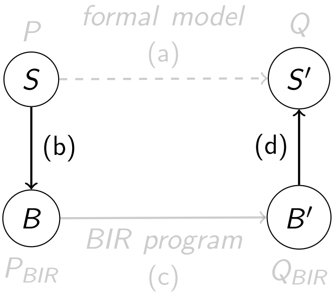
(a) $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=}$
$\forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B$
$\wedge\ \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$



Proof overview

(a) $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=} \forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B \wedge \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$

– Relation between S and B: $\textbf{R}\ S\ B$

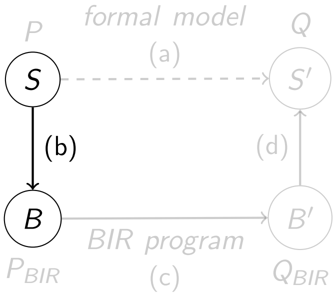


Proof overview

(a) $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=}$
$\forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B$
$\wedge\ \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$

- Relation between S and B: $\textbf{R}\ S\ B$

(b) Injectivity: $\forall S.\ \exists B.\ \textbf{R}\ S\ B$

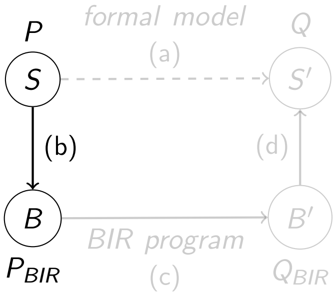


(a) $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=}$
$\forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B$
$\wedge\ \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$

– Relation between S and B: $\textbf{R}\ S\ B$

(b) Injectivity: $\forall S.\ \exists B.\ \textbf{R}\ S\ B$

(b) $\forall B.\ (\exists S.\ \textbf{R}\ S\ B \wedge \textbf{P}\ S) \implies \textbf{P}_{\textbf{BIR}}\ B$



Proof overview

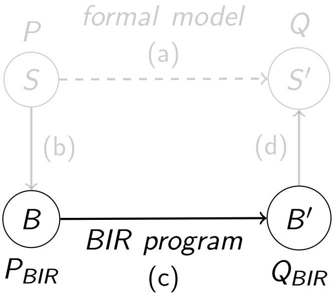
(a) $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=}$
$\forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B$
$\wedge\ \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$

– Relation between S and B: $\textbf{R}\ S\ B$

(b) Injectivity: $\forall S.\ \exists B.\ \textbf{R}\ S\ B$

(b) $\forall B.\ (\exists S.\ \textbf{R}\ S\ B \wedge \textbf{P}\ S) \implies \textbf{P}_{\textbf{BIR}}\ B$

(c) $\forall B\ B'.\ (\textbf{P}_{\textbf{BIR}}\ B \wedge B' =$
$\textbf{BIR\_exec}\ BIR\_prog\ B) \implies \textbf{Q}_{\textbf{BIR}}\ B'$



Proof overview

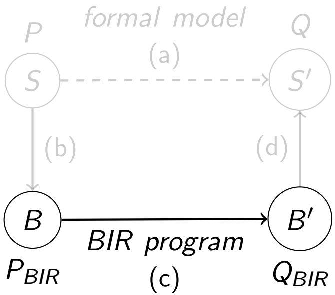
(a) $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=}$
$\forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B$
$\wedge\ \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$

– Relation between S and B: $\textbf{R}\ S\ B$

(b) Injectivity: $\forall S.\ \exists B.\ \textbf{R}\ S\ B$

(b) $\forall B.\ (\exists S.\ \textbf{R}\ S\ B \wedge \textbf{P}\ S) \implies \textbf{P}_{\textbf{BIR}}\ B$

(c) $\{P_{BIR}\}\ BIR\ program\ \{Q_{BIR}\}$



Proof overview

(a) $\forall S\ S'.\ \mathbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=} \forall B\ B'.(B' = \mathbf{BIR\_exec}\ BIR\_prog\ B \land \mathbf{R}\ S\ B) \implies \mathbf{R}\ S'\ B'$

– Relation between S and B: $\mathbf{R}\ S\ B$

(b) Injectivity: $\forall S.\ \exists B.\ \mathbf{R}\ S\ B$

(b) $\forall B.\ (\exists S.\ \mathbf{R}\ S\ B \land \mathbf{P}\ S) \implies \mathbf{P_{BIR}}\ B$

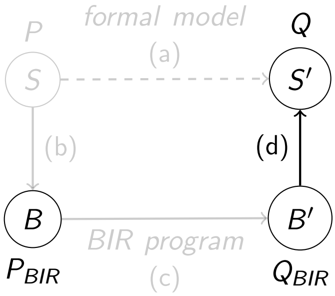
(c) $\{P_{BIR}\}\ BIR\ program\ \{Q_{BIR}\}$

(d) $\forall B'.\ \mathbf{Q_{BIR}}\ B' \implies (\forall S\ S'\ B.\ \mathbf{P_{BIR}}\ B \land \mathbf{R}\ S\ B\ \land\ \mathbf{R}\ S'\ B' \implies \mathbf{Q}\ S\ S')$

$$\begin{array}{ccccc} & P & \text{formal model} & Q & \\ & S & \xrightarrow{(a)} & S' & \\ (b) & \downarrow & & \uparrow & (d) \\ & B & \xrightarrow[(c)]{BIR\ program} & B' & \\ & P_{BIR} & & Q_{BIR} & \end{array}$$

Proof overview

Notation: $\mathbf{P_{BIR}}\ B \stackrel{def}{=} \mathbf{BIR\_eval}\ P_{BIR}\ B$



Proof overview

(a) $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \overset{def}{=}$
$\forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B$
$\wedge\ \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$

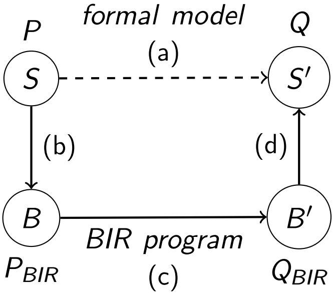
- Relation between S and B: $\textbf{R}\ S\ B$

(b) Injectivity: $\forall S.\ \exists B.\ \textbf{R}\ S\ B$

(b) $\forall B.\ (\exists S.\ \textbf{R}\ S\ B \wedge \textbf{P}\ S) \implies \textbf{P}_{\textbf{BIR}}\ B$

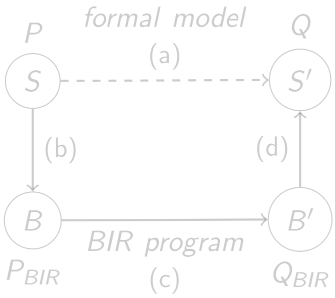
(c) $\{P_{BIR}\}\ BIR\ program\ \{Q_{BIR}\}$

(d) $\forall B'.\ \textbf{Q}_{\textbf{BIR}}\ B' \implies$
$(\forall S\ S'\ B.\ \textbf{P}_{\textbf{BIR}}\ B \wedge \textbf{R}\ S\ B\ \wedge\ \textbf{R}\ S'\ B'$
$\implies \textbf{Q}\ S\ S')$



Proof overview

1. $\forall S\ S'.\ \textbf{exec}\ S\ BIR\_prog\ S' \stackrel{def}{=}$
   $\forall B\ B'.(B' = \textbf{BIR\_exec}\ BIR\_prog\ B$
   $\wedge\ \textbf{R}\ S\ B) \implies \textbf{R}\ S'\ B'$
2. Relation between S and B: $\textbf{R}\ S\ B$
3. Injectivity: $\forall S.\ \exists B.\ \textbf{R}\ S\ B$
4. $\forall B.\ (\exists S.\ \textbf{R}\ S\ B \wedge \textbf{P}\ S) \implies \textbf{P}_{\textbf{BIR}}\ B$
5. $\{P_{BIR}\}\ BIR\ program\ \{Q_{BIR}\}$
6. $\forall B'.\ \textbf{Q}_{\textbf{BIR}}\ B' \implies$
   $(\forall S\ S'\ B.\ \textbf{P}_{\textbf{BIR}}\ B \wedge \textbf{R}\ S\ B\ \wedge\ \textbf{R}\ S'\ B'$
   $\implies \textbf{Q}\ S\ S')$



Proof overview - automate?

Theorem	Length of proof (LoC)	Ease to automate
1. def $S \to S'$ *(a)*	–	*Hard? (lifter)*
2. def relation *(R)*	–	Easy
3. Injectivity	10	Very easy
4. $P \to P_{BIR}$ *(b)*	4	Very easy
5. Hoare Triple *(c)*	151	*Medium?* $*^2$
6. $Q_{BIR} \to S$ *(d)*	48	Should be easy $*^1$

Proof overview - automate?

Theorem	Length of proof (LoC)	Ease to automate
1. def $S \to S'$ *(a)*	–	*Hard? (lifter)*
2. def relation *(R)*	–	Easy
3. Injectivity	10	Very easy
4. $P \to P_{BIR}$ *(b)*	4	Very easy
5. Hoare Triple *(c)*	151	*Medium?* $*^2$
6. $Q_{BIR} \to S$ *(d)*	48	Should be easy $*^1$

$*^1$ Need 2 simple tactics

Proof overview - automate?

Theorem	Length of proof (LoC)	Ease to automate
1. def $S \to S'$ *(a)*	–	*Hard? (lifter)*
2. def relation *(R)*	–	Easy
3. Injectivity	10	Very easy
4. $P \to P_{BIR}$ *(b)*	4	Very easy
5. Hoare Triple *(c)*	151	*Medium?* *[2]
6. $Q_{BIR} \to S$ *(d)*	48	Should be easy *[1]

*[1] Need 2 simple tactics

*[2] Need smart tactics (multi-pass, goal aware)

Section 4

Conclusion

Conclusion

- Automation is feasible
- Can reduce proof lengths and complexity
- Trustworthy if proof-producing

Questions

References I

[1] Is seL4 proven secure? | FAQ | seL4 docs.

[2] What is proved and what is assumed | seL4.

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

[4] Jonas Haglund.
Formal verification of systems software.

[5] Dr Charlie Miller and Chris Valasek.
Remote exploitation of an unaltered passenger vehicle.

[6] Thomas Tuerk.
Interactive theorem proving (ITP) course.

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

HolBA overview

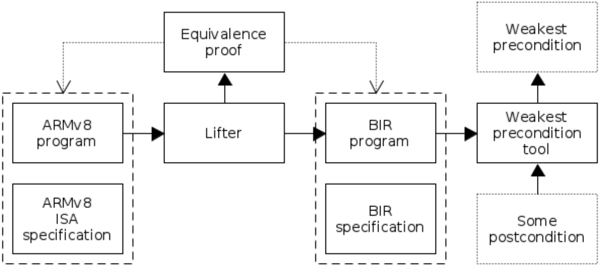


Figure: Overview of the HolBA framework (lifter and WP tool)

Fringe

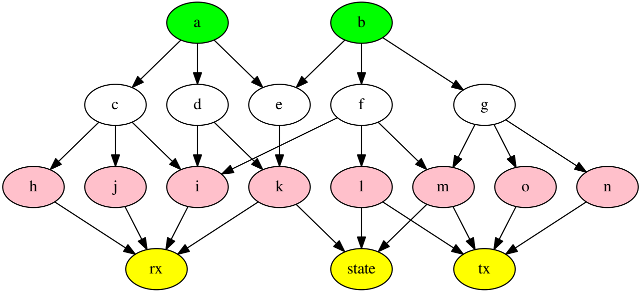


Figure: Fringe of an ideally-shaped proof

Pipeline public interface

```
val thm = prove_contract "cjmp"
  cjmp_prog_def
  (* Precondition *) (blabel_str "entry", btrue)
  (* Postcondition *) (
    [blabel_str "end"],
    beq ((bden o bvarimm32) "y", bconst32 100)
  )
```

BSL: BIR Simple Language

```
bite (
  borl [
    ble ((bden o bvarimm64) "x", bconst64 100),
    bnot (ble (bplus ((bden o bvarimm64) "y", bconst64 1),
                 bconst64 10)),
    ble (bplus ((bden o bvarimm64) "x",
                 (bden o bvarimm64) "y"),
             bconst64 20)
  ],
  bmult ((bden o bvarimm64) "x", bconst64 2),
  bplus (bmult ((bden o bvarimm64) "x", bconst64 3),
             bconst64 1)
)
```

BIR pretty-printer - disabled

```
BExp_IfThenElse
  (BExp_BinExp BIExp_Or
     (BExp_BinExp BIExp_Or
        (BExp_BinPred BIExp_LessOrEqual
           (BExp_Den (BVar "x" (BType_Imm Bit64))) (BExp_Const (Imm64 100w)))
        (BExp_UnaryExp BIExp_Not
           (BExp_BinPred BIExp_LessOrEqual
              (BExp_BinExp BIExp_Plus (BExp_Den (BVar "y" (BType_Imm Bit64)))
                 (BExp_Const (Imm64 1w))) (BExp_Const (Imm64 10w)))))
     (BExp_BinPred BIExp_LessOrEqual
        (BExp_BinExp BIExp_Plus (BExp_Den (BVar "x" (BType_Imm Bit64)))
           (BExp_Den (BVar "y" (BType_Imm Bit64)))) (BExp_Const (Imm64 20w))))
  (BExp_BinExp BIExp_Mult (BExp_Den (BVar "x" (BType_Imm Bit64)))
     (BExp_Const (Imm64 2w)))
  (BExp_BinExp BIExp_Plus
     (BExp_BinExp BIExp_Mult (BExp_Den (BVar "x" (BType_Imm Bit64)))
        (BExp_Const (Imm64 3w))) (BExp_Const (Imm64 1w)))
```

BIR pretty-printer - enabled

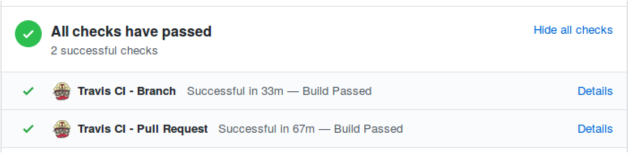
```
BExp_If
  (BExp_Or
     (BExp_LessOrEqual
        (BExp_Den (BVar "x" (BType_Imm Bit64))) (BExp_Const (Imm64 100w)))
     (BExp_Not
        (BExp_LessOrEqual
           (BExp_Plus
              (BExp_Den (BVar "y" (BType_Imm Bit64))) (BExp_Const (Imm64 1w)))
           (BExp_Const (Imm64 10w))))
     (BExp_LessOrEqual
        (BExp_Plus
           (BExp_Den (BVar "x" (BType_Imm Bit64)))
           (BExp_Den (BVar "y" (BType_Imm Bit64))))
        (BExp_Const (Imm64 20w))))
BExp_Then
  (BExp_Mult (BExp_Den (BVar "x" (BType_Imm Bit64))) (BExp_Const (Imm64 2w)))
BExp_Else
  (BExp_Plus
     (BExp_Mult
        (BExp_Den (BVar "x" (BType_Imm Bit64))) (BExp_Const (Imm64 3w)))
     (BExp_Const (Imm64 1w)))
```

Exception pretty-printer and LogLib

```
[TRACE @ nic_helpersLib::prove_p_imp_wp] smt_ready_tm:
¬(if (nic_dead = 0w) ∧ (nic_init_state = 2w) then 1w else 0w) ‖
(¬(if nic_init_state = 1w then 1w else 0w) ‖ if 1w = 0w then 1w else 0w) &&
((if nic_init_state = 1w then 1w else 0w) ‖
 (¬(if nic_init_state = 2w then 1w else 0w) ‖ if nic_dead = 0w then 1w else 0w) &&
 ((if nic_init_state = 2w then 1w else 0w) ‖
  (¬(if nic_init_state = 3w then 1w else 0w) ‖ if 1w = 0w then 1w else 0w) &&
  ((if nic_init_state = 3w then 1w else 0w) ‖
   (¬(if nic_init_state = 4w then 1w else 0w) ‖ if 1w = 0w then 1w else 0w) &&
   if (nic_init_state = 4w) ∨ (1w = 0w) then 1w else 0w))) =
1w
Handled exception: [init_automaton_doesnt_die] Z3_ORACLE_PROVE failed
HOL_ERR:
 - Structure: nic_helpersLib
 - Function: prove_p_imp_wp
 - Message: at Z3.Z3_SMT_Oracle:
Z3 not configured: set the HOL4_Z3_EXECUTABLE environment variable to point to the Z3 executable file.

[DEBUG @ nic_helpersLib::prove_p_imp_wp] Asking Z3 for a SAT model...
[DEBUG @ nic_helpersLib::prove_p_imp_wp] Failed to compute a SAT model. Ignoring.
error in quse /NOBACKUP/tholac/holba-reborn/examples/nic/test-early-wp.sml : HOL_ERR {message = "at Z3.Z3_SMT
_Oracle:\nZ3 not configured: set the HOL4_Z3_EXECUTABLE environment variable to point to the Z3 executable fi
le.", origin_function = "prove_p_imp_wp", origin_structure = "nic_helpersLib"}
error in load test-early-wp : HOL_ERR {message = "at Z3.Z3_SMT_Oracle:\nZ3 not configured: set the HOL4_Z3_EX
ECUTABLE environment variable to point to the Z3 executable file.", origin_function = "prove_p_imp_wp", origi
n_structure = "nic_helpersLib"}
Uncaught exception: HOL_ERR {message = "at Z3.Z3_SMT_Oracle:\nZ3 not configured: set the HOL4_Z3_EXECUTABLE e
nvironment variable to point to the Z3 executable file.", origin_function = "prove_p_imp_wp", origin_structur
e = "nic_helpersLib"}
```

Continuous Integration (CI) - tests



Continuous Integration (CI) - static analysis

holba-bot commented 5 days ago

🤖 Results of: `scripts/ci/static-analysis.sh`

Grep-cheat analysis:

- Found 23 occurences of `cheat` in `src/` and 0 in `examples/`.

Grep-todo analysis:

- Found 52 occurences of `TODO/FIXME` in `src/` and 0 in `examples/`.

Note: I'm a script, and I'm simple, so I may be missing something or show false positives. You can review the script here.