Towards practical and highly assured information flow control for mobile apps

David Naumann

Ongoing work with Andrey Chudnov, Anindya Banerjee, Stan Rosenberg, Mohammad Nikouei, Mounir Assaf, Gary Leavens, ...
Partially supported by NSF CNS 1228930 and Dept. Homeland Security.

You have zero assurance anyway. Get over it.

- Burgeoning mobile app ecosystem is rife with dangers: highly interdependent software components from disparate providers, bugs, misfeatures, mass-market malware, advanced persistent threats,...
- The high stakes for personal privacy and for mission-critical information is motivating R&D for tools to improve security and privacy.
  - well automated taint-trackers for standard policies [Andromeda,FlowDroid,Sparta...]
  - special tools for special policies (latitude/longitude, street address) [Cortesi,Ferrara,Pistoia,Tripp]
  - improving ways to prioritize warnings [Tripp et al CCS'14] and to explore [Chong et al PLDI'15]

Lack high assurance for end-to-end properties of most software, despite strides in verification [seL4; Ironclad Apps; Verified Software Toolchain].

At least do no harm

Why focus on confidentiality and integrity? Regardless of whether it does anything useful, I would like to know my phone isn't subverting my organization's mission or my personal privacy.

- Crisp and clear requirements for information flow: what is sensitive, where does it come from and where can it go?
- Reasonable and explicit assumptions. (This talk: hardware, OS, and language runtime trustworthy.)
- Find flaws and vulnerabilities, and evidence of trustworthiness.

Ideal evidence: formal specs, verified software stack, attested by hw.

Challenges and threats to assurance

- Scale and complexity of systems. A simple property like "doesn't display CC number" depends on many properties of the sw/hw platform and its environment.
- What even is "the platform"?
  mobile/web app ecosystem, code injection, ...
- Analysts and developers have resources and expertise to specify/annotate/analyze critical apps and their configuration —but not to undertake comprehensive interactive proof.

Challenges and threats to assurance

- Scale and complexity of systems. A simple property like "doesn't display CC number" depends on many properties of the sw/hw platform and its environment.
- What even is "the platform"?
  mobile/web app ecosystem, code injection, ...
- Analysts and developers have resources and expertise to specify/annotate/analyze critical apps and their configuration —but not to undertake comprehensive interactive proof.
- Fragmentation of R&D. Plentiful funding, near-term needs.

Are researchers doing all we can to give engineers the science they need in the long run?

Assertions (Verification 101)

- Assert conditions at control points:
  - Abstract interpreters infer value ranges, type constraints etc. [Cousot&Cousot'77]
  - Suited to precise verification using SMT solvers
  - Developers write preconditions; checked dynamically/statically.
- High assurance: prove, in an accurate model of system behavior and environment, that an assertion always holds (or will always be checked). [Floyd'67; Klein et al'13, Appel et al'15]

Assertions (Verification 101)

- Assert conditions at control points:
  - Abstract interpreters infer value ranges, type constraints etc. [Cousot&Cousot'77]
  - Suited to precise verification using SMT solvers
  - Developers write preconditions; checked dynamically/statically.
- High assurance: prove, in an accurate model of system behavior and environment, that an assertion always holds (or will always be checked). [Floyd'67; Klein et al'13, Appel et al'15]

Security 101:

- Requirements are specified, as access controls.
- Assertions are for safety properties — but information flow is a "hyperproperty" [Clarkson,Schneider'08]

Being hyper about confidentiality assurance

We analyze app executions

- May send GPS coords to local map service
- Never sends GPS coords unless user auth'd within past 5 min
- May send current_city to adware
- Never tries to send GPS to adware

We model environment/adversary

- Adversary controls/observes net
- Adversary controls some apps
- Adversary knows the code and reasons about correlations in multiple (low-)observations

Being hyper about confidentiality assurance

We analyze app executions

- May send GPS coords to local map service
- Never sends GPS coords unless user auth'd within past 5 min
- May send current_city to adware
- Never tries to send GPS to adware

We model environment/adversary

- Adversary controls/observes net
- Adversary controls some apps
- Adversary knows the code and reasons about correlations in multiple (low-)observations

What is confidentiality of GPS?

For every observed run, and every fine location, there is another run with that location but the same (low-)observables.

Epistemic property [Askarov,Sabelfeld'07; Balliu'13],
2-safety hyperproperty [Terauchi'05; Clarkson/Schneider'08]

— so it needs complex or novel specification notation?

Specifying information flow requirements (Flowspecs)

Baseline policies:

- fix channel labels
  (sources/sinks, e.g., derived from access permissions)
- declare flow-to relation to designate allowed dependencies, e.g.,
  GPS is secret, secrets cannot flow to network.

Specifying information flow requirements (Flowspecs)

Baseline policies:

- fix channel labels
  (sources/sinks, e.g., derived from access permissions)
- declare flow-to relation to designate allowed dependencies, e.g., GPS is secret, secrets cannot flow to network.

Downgrading policies:

- give conditions under which additional flows allowed
- what is released/endorsed, by whom, where, when

Many conditions expressible in terms of program data, sanitizer functions, and implicit state represented by control point.
E.g., high order bits of GPS coords can flow to adware, if "coarse location" currently selected and log entry has been written.

How to specify policies

Micro Heartbleed — example violation a taint tracker might detect

```
char * buf = malloc(256);
char * rsp = malloc(256);
loadKeys(buf); //buf[245..255] := secret key
readSocket(buf); //buf [0..99] := untrusted input
int rspLen = (int) buf[0];
memcpy(rsp, buf, rspLen); //require trusted length
...
```

How to specify policies

Micro Heartbleed fix — becomes false positive?

```
char * buf = malloc(256);
char * rsp = malloc(256);
loadKeys(buf); //buf[245..255] := secret key
readSocket(buf); //buf [0..99] := untrusted input
int rspLen = (int) buf[0];
if (rspLen < 100) {
    memcpy(rsp, buf, rspLen); //require trusted length
                              //but is rspLen still tainted?
    ...
}
```

How to specify policies

A precise policy for the fix — using assertions

```
char * buf = malloc(256);
char * rsp = malloc(256);
loadKeys(buf);
readSocket(buf);
assume 𝔹 (buf[0] < 100) ⟹ 𝔸 buf[0] // cond'l endorsement
int rspLen = (int) buf[0];
if (rspLen < 100) {
    assert 𝔸 rspLen; //require length to be trusted
    memcpy(rsp, buf, rspLen);
    ...
}
```

Intermediate assumptions specify endorsement and declassification.

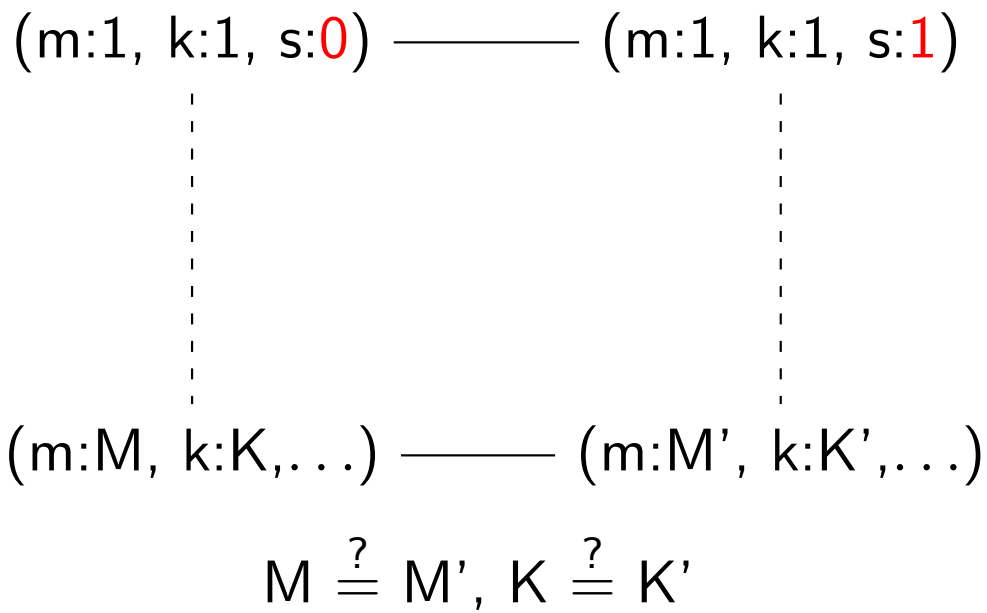
$\mathbb{B}$ means both runs; $\mathbb{A}$ means alternate runs agree on value.

Noninterference

memcpy_len: low (trusted)
keys: low (trusted)
socket: high (untrusted)

high shouldn't influence low

$\mathbb{A}$ means two states agree

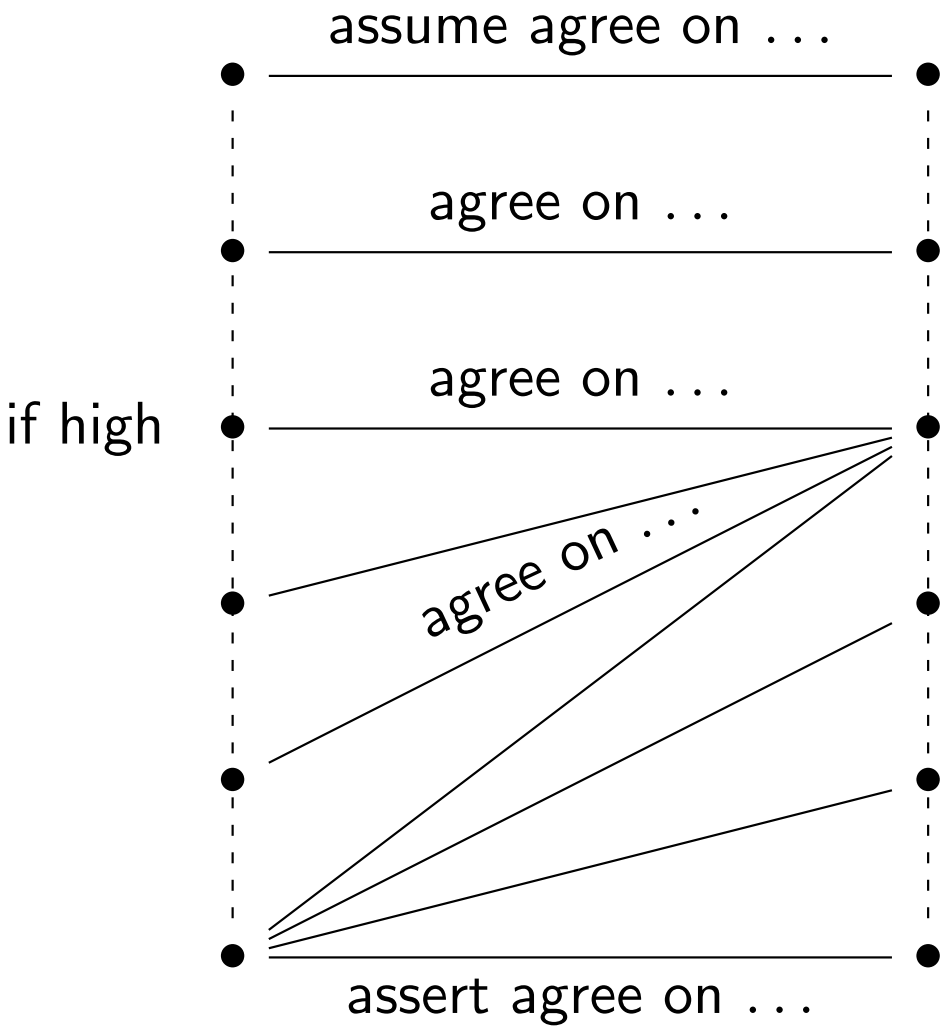


Precondition
Assume $\mathbb{A}m \wedge \mathbb{A}k$

Postcondition
Assert $\mathbb{A}m \wedge \mathbb{A}k$

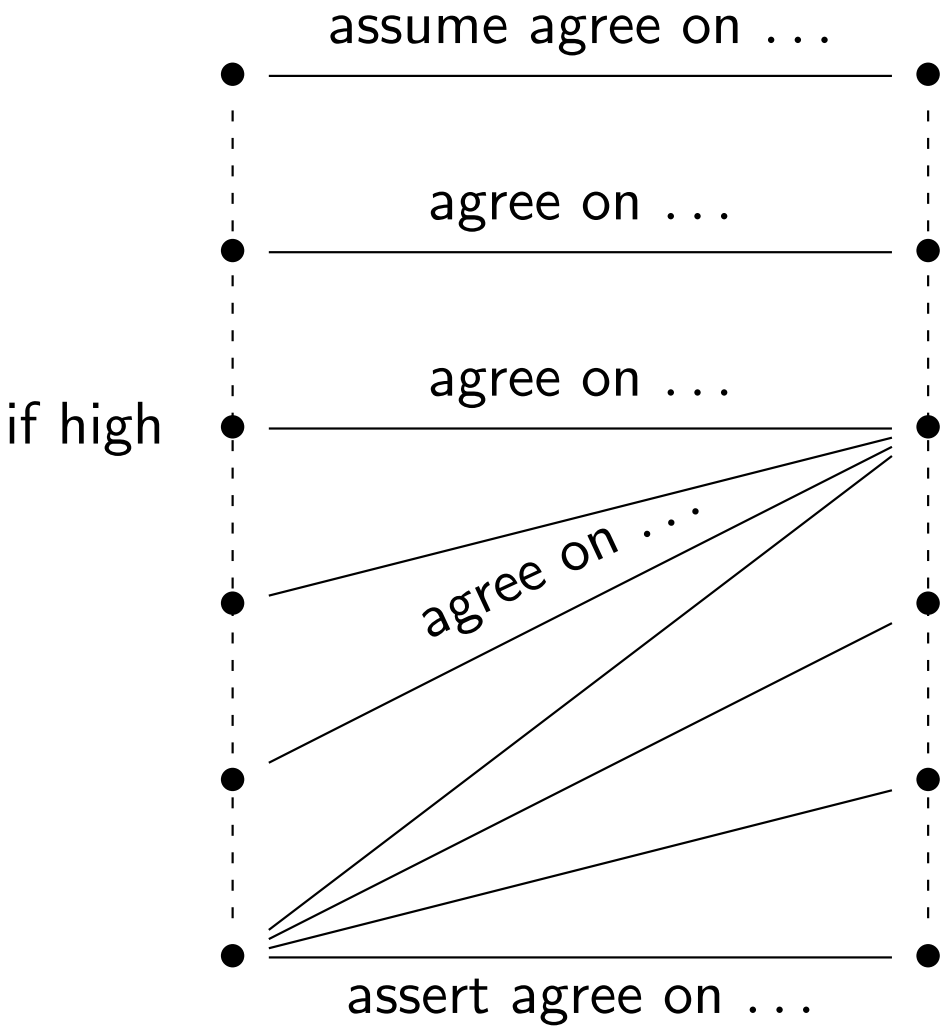
Reasoning about noninterference

Proofs establish stronger properties (unwinding conditions)



Reasoning about noninterference

Proofs establish stronger properties (unwinding conditions)



- Correctness of static analysis
  [Volpano,Smith'97]
- Static analysis for 2-safety
  [Kovács,Seidl,Finkbeiner CCS'13]
- Information flow in logical form
  [Amtoft,Banerjee,Bandhakavi'06],
  [Nanevsky,Banerjee,Garg'13]
- Epistemic properties
  [Banerjee,DN,Rosenberg'08; . . .]

Semantics of small-step relational logic

y := trusted_input();
Assume $\mathbb{A}$ y; // baseline policy

Assume $\mathbb{A}$ e; // endorse e
x := e;

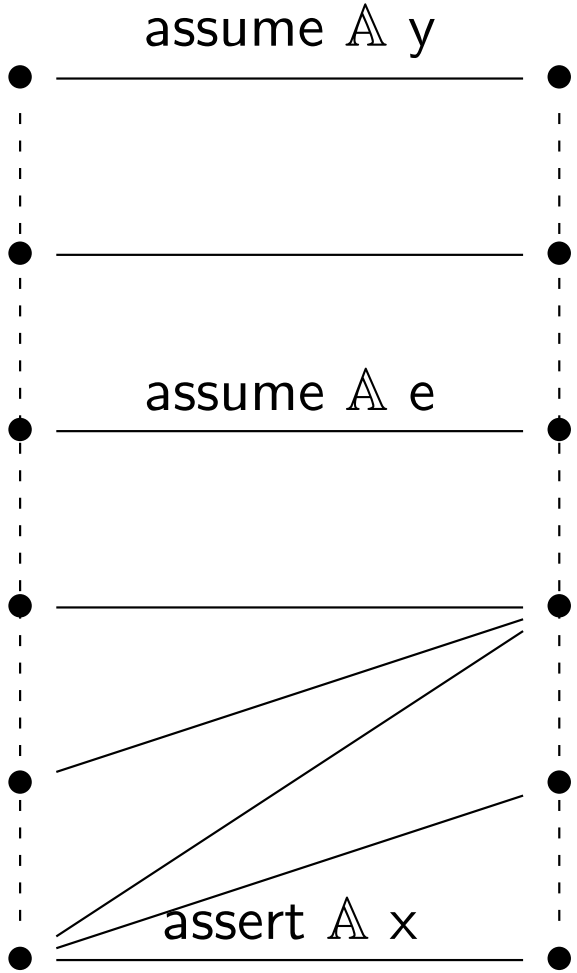
Assert $\mathbb{A}$ x; // baseline policy
trusted_output := x;

Semantics of small-step relational logic

y := trusted_input();
Assume $\mathbb{A}$ y; // baseline policy

Assume $\mathbb{A}$ e; // endorse e
x := e;

Assert $\mathbb{A}$ x; // baseline policy
trusted_output := x;

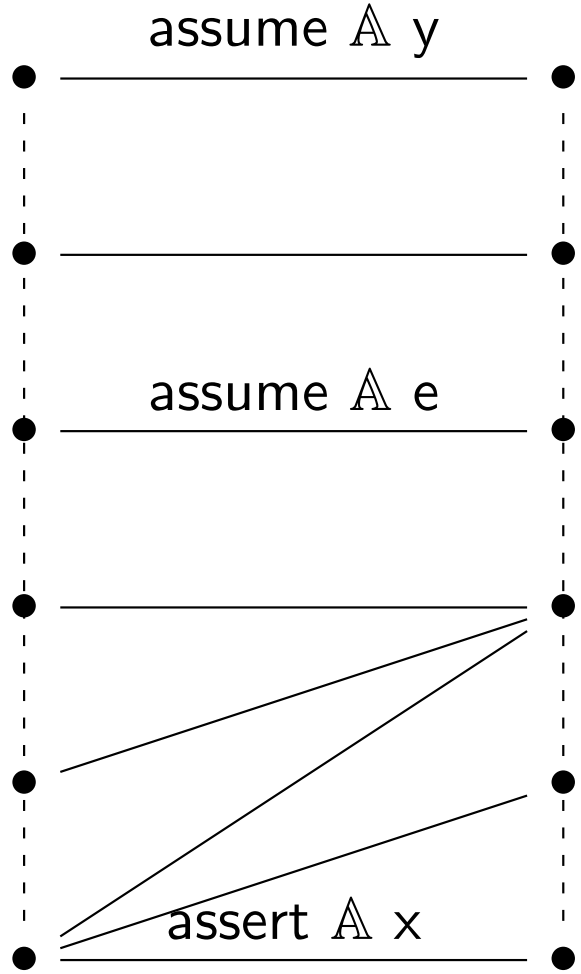


Semantics of small-step relational logic

y := trusted_input();
Assume $\mathbb{A}$ y; // baseline policy

Assume $\mathbb{A}$ e; // endorse e
x := e;

Assert $\mathbb{A}$ x; // baseline policy
trusted_output := x;



Security property: $\forall$ pairs of traces $\exists$ alignment such that:
if an assertion fails then there's a preceding assumption failure.

Relational logic: 2-safety and beyond

Relational formulas Φ interpreted in a pair of states. E.g.,
$\mathbb{B}\mathit{root} \neq \mathit{null} \Rightarrow \mathbb{A}\mathit{root}.\mathit{nxt}^*.\mathit{val}$

Baseline (by typing) + relational specs $\Longrightarrow$ epistemic security for stateful declassification policies [Banerjee,DN,Rosenberg'08]

Relational logic: 2-safety and beyond

Relational formulas Φ interpreted in a pair of states. E.g.,
$\mathbb{B}root \neq null \Rightarrow \mathbb{A}root.nxt^*.val$

Baseline (by typing) + relational specs $\Longrightarrow$ epistemic security for stateful declassification policies [Banerjee,DN,Rosenberg'08]

General form relates two programs: $\{\Phi\}C \sim C'\{\Psi\}$
Many applications: conformance of a model of platform code; compiler/analyzer optimizations, ...

Relational logic: 2-safety and beyond

Relational formulas Φ interpreted in a pair of states. E.g., $\mathbb{B}root \neq null \Rightarrow \mathbb{A}root.nxt^*.val$

Baseline (by typing) + relational specs $\Longrightarrow$ epistemic security for stateful declassification policies [Banerjee,DN,Rosenberg'08]

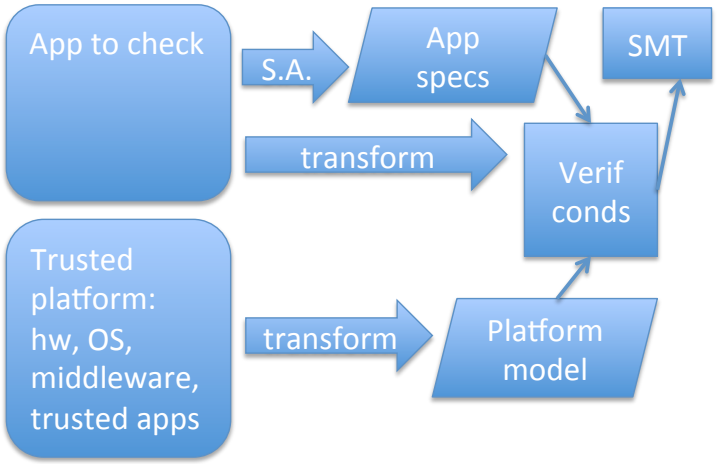
General form relates two programs: $\{\Phi\}C \sim C'\{\Psi\}$
Many applications: conformance of a model of platform code; compiler/analyzer optimizations, ...

Logic/verifier needs guidance for alignments:

- syntactic interleaving of code ("self-composition")
- synchronization guards

[Banerjee,DN,Nikouei'15]

Architecture sketch for assurance via static verification



Standard assertion checking, plus modular relational verifacation for assured downgrading.

Static Analysis to infer candidate invariants, infer labeling, alignments, . . .

Transforms used at many stages — can (in principle) be assured by relational verification.

Assurance via monitoring

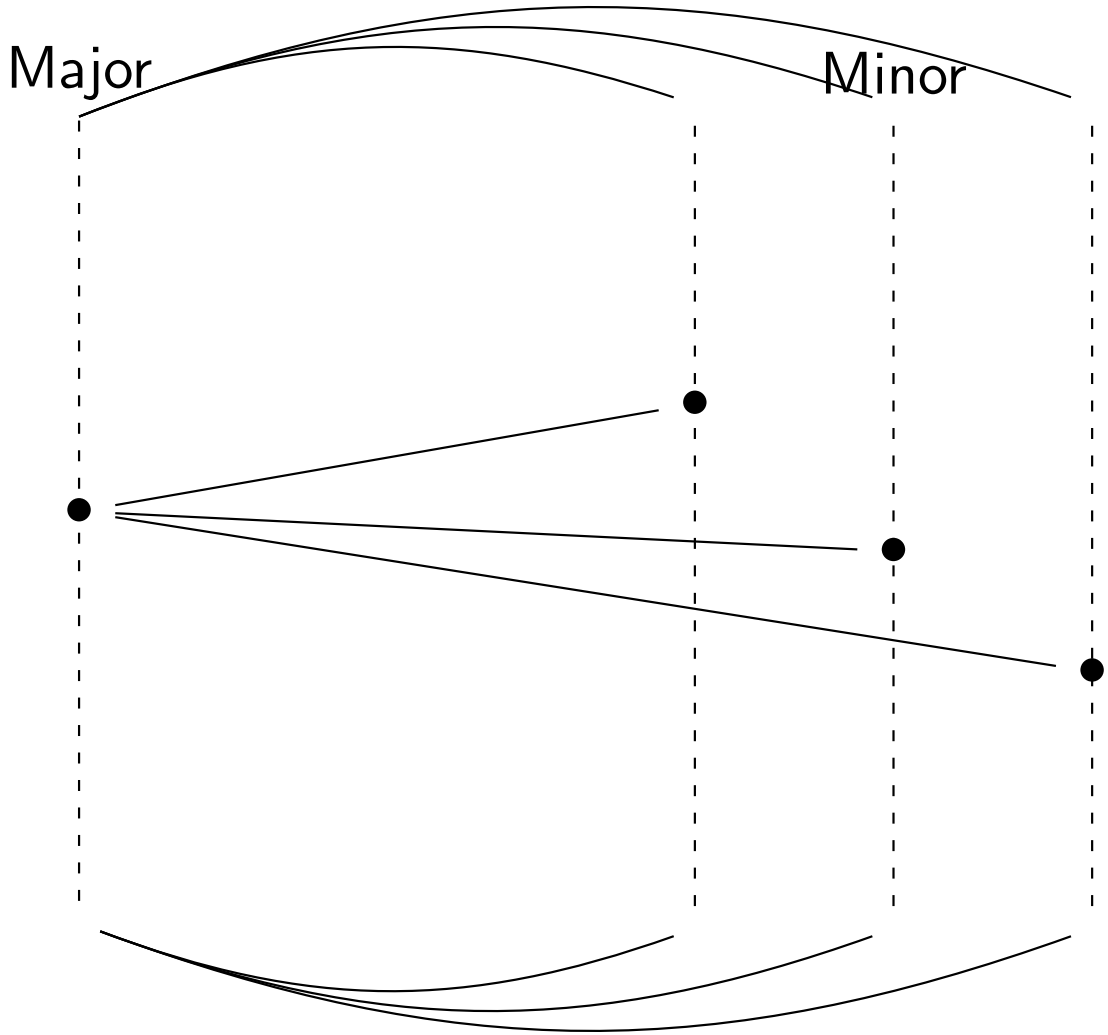
Why monitor?

- Verification isn't feasible/practical for some components. E.g., JavaScript in a WebView.
- Not all runs are secure.
- Many policy-relevant conditions are already checked at runtime.
- Many policy-relevant events are already logged at runtime.

Evidence needed for assurance:

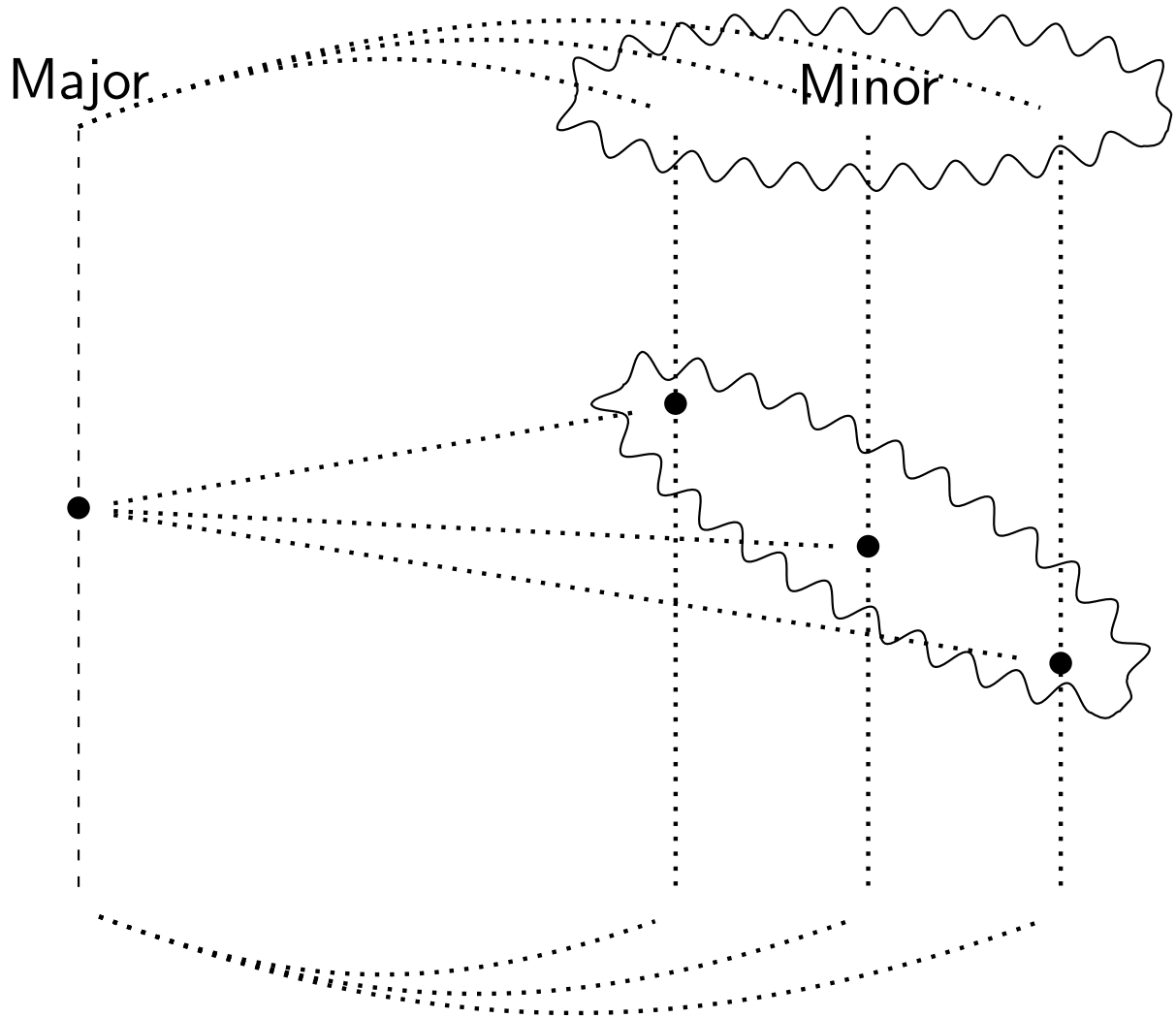
- Connection between what's monitored and what's statically verified.
- Correctness of the monitor — but it's a hyperproperty!

How to design a monitor

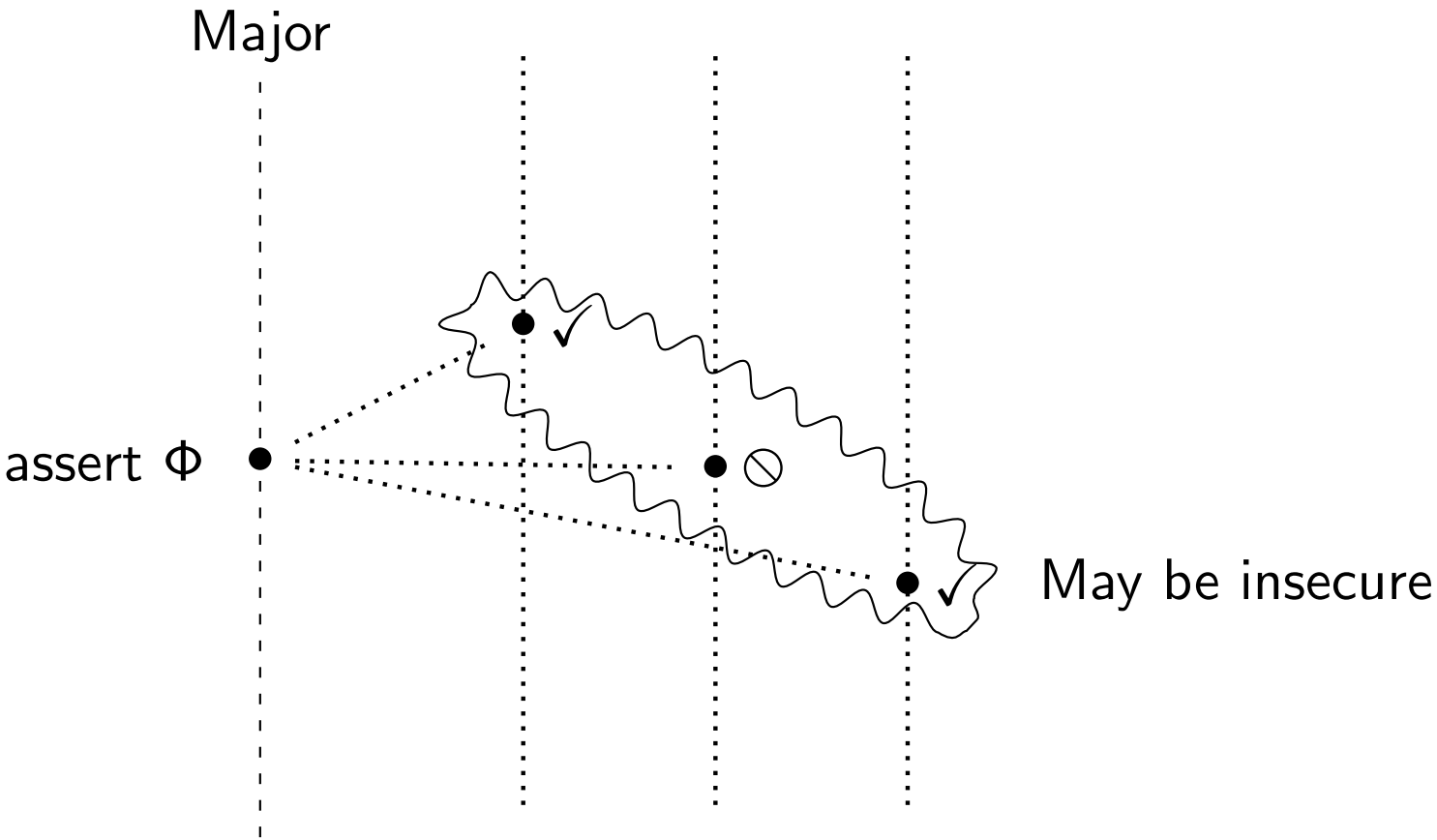


How to design a monitor

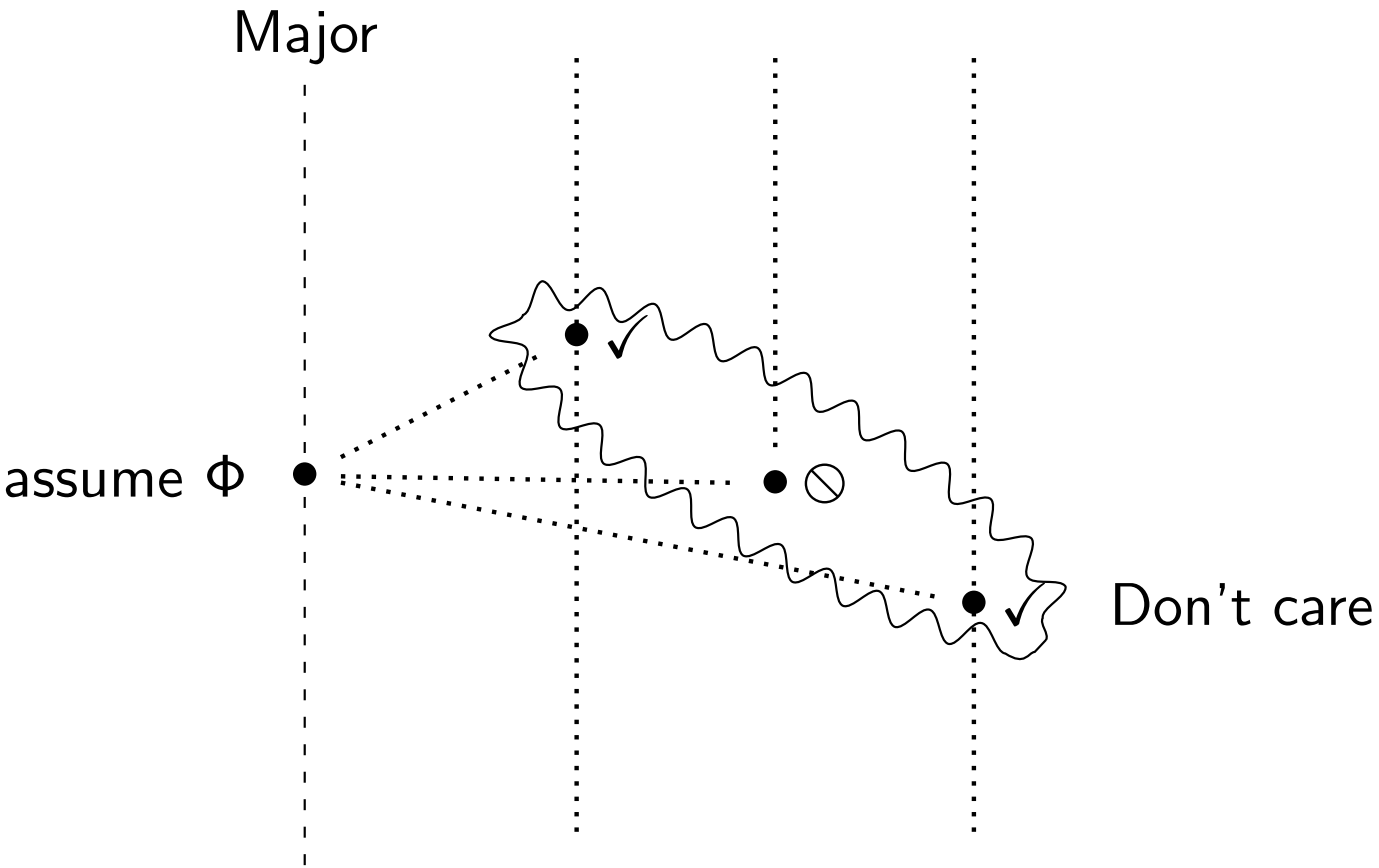
Monitor's view:



How to design a monitor (assert)



How to design a monitor (assume)



How to design a monitor

Monitor state from previous work (e.g., [Russo,Sabelfeld'10])

Variable/location labels

λ : Variables $\rightarrow \{\mathbf{lo}, \mathbf{hi}\}$

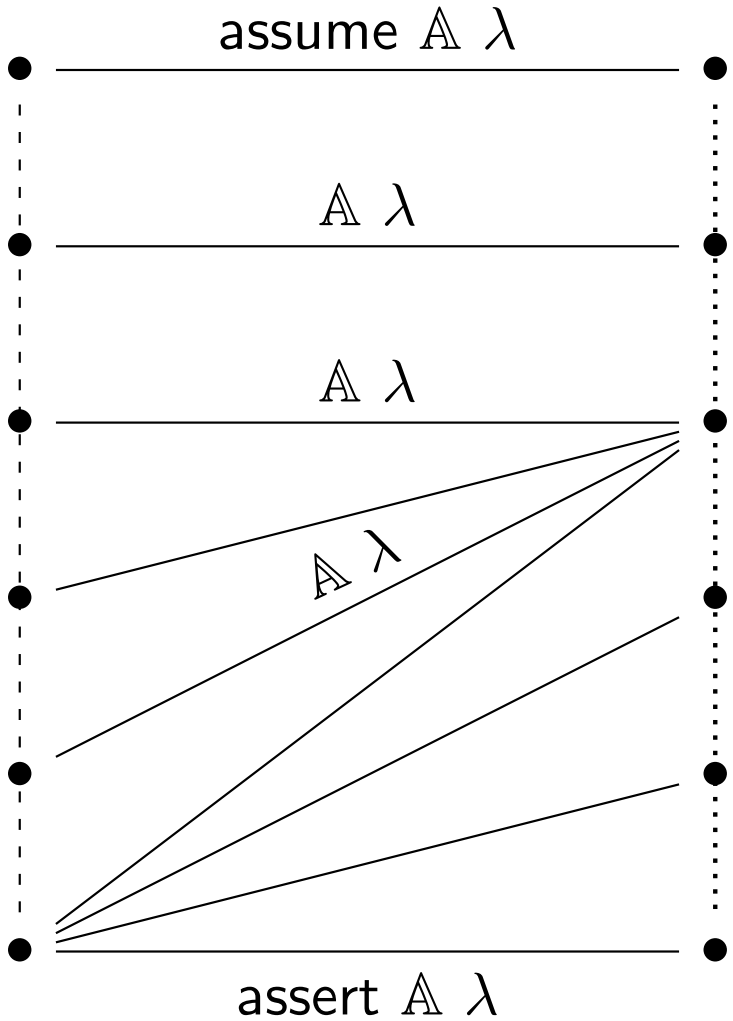
PC level stack

PC : List of levels paired with control join points.
PC provides for compensating for branches not taken, based on static analysis of write effects.

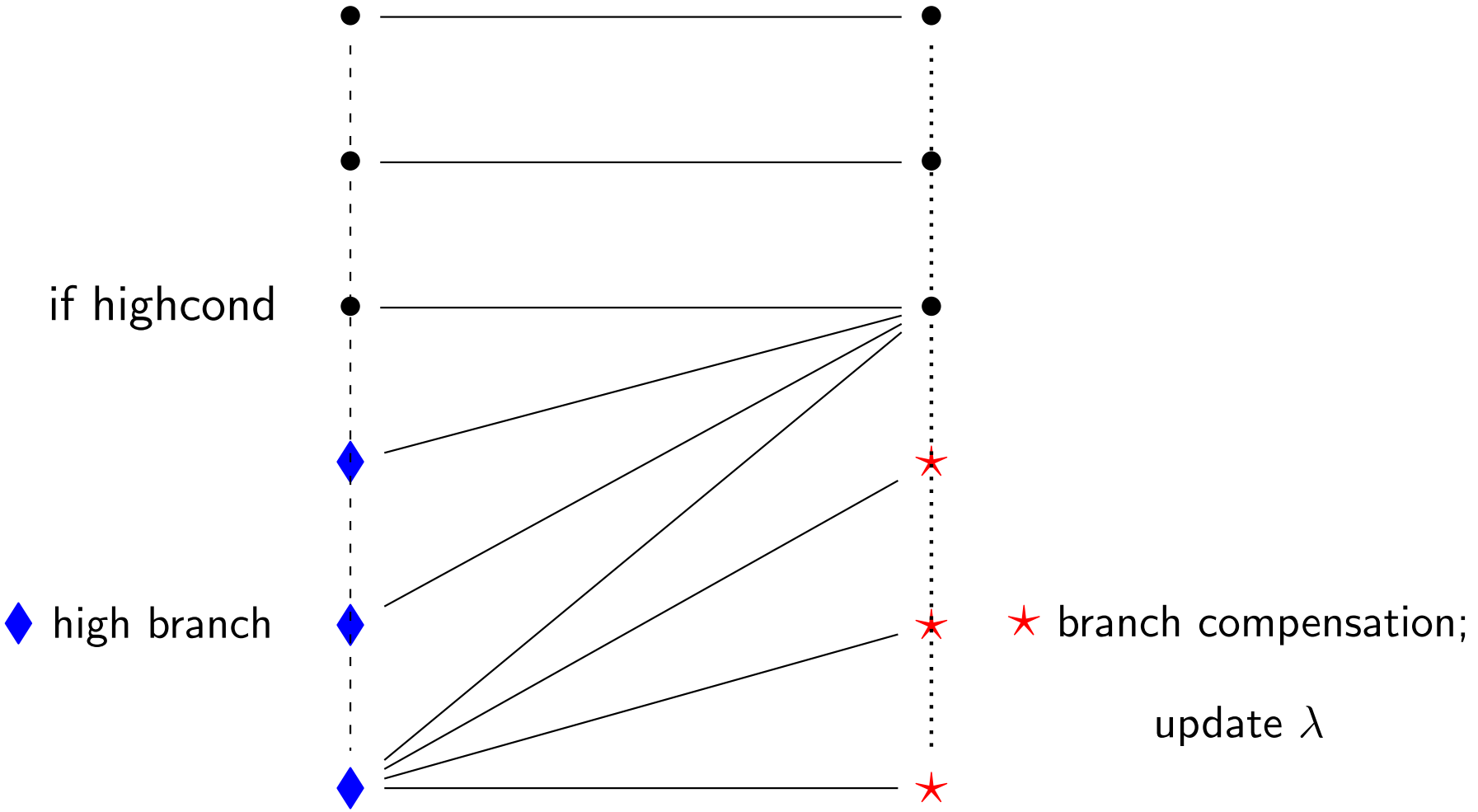
Variable levels tell about assumed agreements

$\mathbb{A}x$ where $\lambda(x) = \mathbf{lo}$

Track agreements at every step, not just assert/assume.



PC level stack tells about alignments



Tracking of assumptions

λ : *Variables* $\rightarrow$ *Levels*

PC : list-of (*Levels* $\times$ *JoinPoints*)

Δ : the set of assumed formulas still true.

Tracking of assumptions

$\lambda : \mathit{Variables} \to \mathit{Levels}$
PC : list-of $(\mathit{Levels} \times \mathit{JoinPoints})$
Δ : the set of assumed formulas still true.

Selected monitor transitions (in low context):

assume Φ	$\Delta := \Delta \cup \{\Phi\}$
$x := e$	$\lambda(x) :=$ **lo** if $\mathbb{A}\lambda \wedge \Delta \implies \mathbb{A}e$, else **hi** $\Delta := \mathit{approxStrongestPost}((x := e)^2, \Delta)$
assert Φ	$\Delta := \Delta \cup \{\Phi\}$ if $\mathbb{A}\lambda \wedge \Delta \implies \Phi$, else security violation

A "derived" monitor [Chudnov,Kuan,DN CSF'14]

Run it — or verify the monitored program.

$$\langle c,\ \sigma,\ \lambda, \pi, \Delta\rangle \mapsto \langle c',\ \sigma',\ \lambda', \pi', \Delta'\rangle$$

c	π	π'	λ'	Δ'
$x{:=}e;\ d$	**lo**	π	$[\lambda \mid x : l]$ where $l = \begin{cases}\textbf{lo} \text{ if } \lambda(e) = \textbf{lo} \vee \\ \quad chkVal(\mathbb{A}\lambda \wedge \Delta \implies \mathbb{A}e) \\ \textbf{hi} \text{ otherwise}\end{cases}$	$\{\Phi \mid \Phi \in \Delta \wedge x \notin FV(\Phi)\}$
$x{:=}e;\ d$	$(\textbf{hi}, \_)$	π	$[\lambda \mid x : \textbf{hi}]$	$\{\Phi \mid \Phi \in \Delta \wedge x \notin FV(\Phi)\}$
while e do p; d	$\_$	π	λ	Δ
skip; d	**lo**	π	λ	Δ
skip; d	$(\textbf{hi}, (b, d) : [])$	**lo**	$lift(\lambda, targets(b))$	$\{\Phi \mid \Phi \in \Delta \wedge$ $targets(b) \cap FV(\Phi) = \emptyset\}$
skip; d	$(\textbf{hi}, (b, d) : r)$	$(\textbf{hi}, r)$	$lift(\lambda, targets(b))$	$\{\Phi \mid \Phi \in \Delta \wedge$ $targets(b) \cap FV(\Phi) = \emptyset\}$
skip; d	$(\textbf{hi}, (b, c') : \_)$	π	λ	Δ
if e then b_{tt} else b_{ff}; d	**lo**	$\begin{cases}\textbf{lo} \text{ if } \lambda(vars(e)) = \textbf{lo} \vee \\ \quad chkVal(\mathbb{A}\lambda \wedge \Delta \implies \mathbb{A}e) \\ (\textbf{hi}, (b_{\neg\sigma(e)}, d) : []) \text{ otherwise}\end{cases}$	λ	Δ
if e then b_{tt} else b_{ff}; d	$(\textbf{hi}, \mathrm{st})$	$(\textbf{hi}, (b_{\neg\sigma(e)}, d) : st)$	λ	Δ
assert Ψ; d	**lo**	failure, if $check(\Psi, \sigma, \lambda, \Delta) \neq \mathrm{tt}$; otherwise, see next row:		
		π	λ	Δ, Ψ
assert Ψ; d	$(\textbf{hi}, \_)$	failure		
assume Ψ; d	**lo**	π	λ	Δ, Ψ
assume Ψ; d	$(\textbf{hi}, \_)$	failure		

Monitor reasoning example

Endorsement conditioned on password check

assume $(\mathbb{B}\ (\text{guess} = \text{password}) \implies \mathbb{A}\,\text{untrusted})$
$\wedge\, \mathbb{A}\,(\text{guess} = \text{password})$
if guess $=$ password
then
  assert $\mathbb{B}\,(\text{guess} = \text{password})$; // succeeds
  trusted $:=$ untrusted; //λ(trusted):=low (from $\mathbb{A}$ untrusted)
else skip;
assert $\mathbb{A}$ trusted // succeeds

In conclusion

It's early days in the science of security and "hyperproperties" are a crazy big class. But 2-safety encompasses a big range of info flow policies. Results using toy languages suggest guiding principles.

- Usable specification notations (assertions, access labels/types)
- Systematic derivation of 2-safety monitors [Cousot&Cousot'79]
- Validation of platform models, to leverage existing tools towards high assurance

In conclusion

It's early days in the science of security and "hyperproperties" are a crazy big class. But 2-safety encompasses a big range of info flow policies. Results using toy languages suggest guiding principles.

- Usable specification notations (assertions, access labels/types)
- Systematic derivation of 2-safety monitors [Cousot&Cousot'79]
- Validation of platform models, to leverage existing tools towards high assurance

Highly assured 2-safety monitor, as abstract interpretation

- of a 1-safety property (indexed by the major run)
- Going further: quantitative info flow
- Can abstract interpreters/tools infer alignments to facilitate SMT-based relational logic verifiers?

In conclusion

It's early days in the science of security and "hyperproperties" are a crazy big class. But 2-safety encompasses a big range of info flow policies. Results using toy languages suggest guiding principles.

- Usable specification notations (assertions, access labels/types)
- Systematic derivation of 2-safety monitors [Cousot&Cousot'79]
- Validation of platform models, to leverage existing tools towards high assurance

Highly assured 2-safety monitor, as abstract interpretation

- of a 1-safety property (indexed by the major run)
- Going further: quantitative info flow
- Can abstract interpreters/tools infer alignments to facilitate SMT-based relational logic verifiers?

~~Punchline buried in false positives or smothered by label creep?~~