Trust in Formal Methods Toolchains

Arie Gurfinkel

Software Engineering Institute
Carnegie Mellon University

July 14, 2013
VeriSure

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

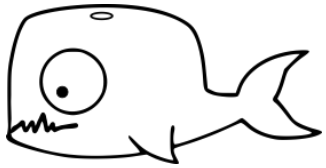
DM-0000507

About Me

Researcher at Carnegie Mellon Software Engineering Institute
Working on Software Model Checking and Static Analysis

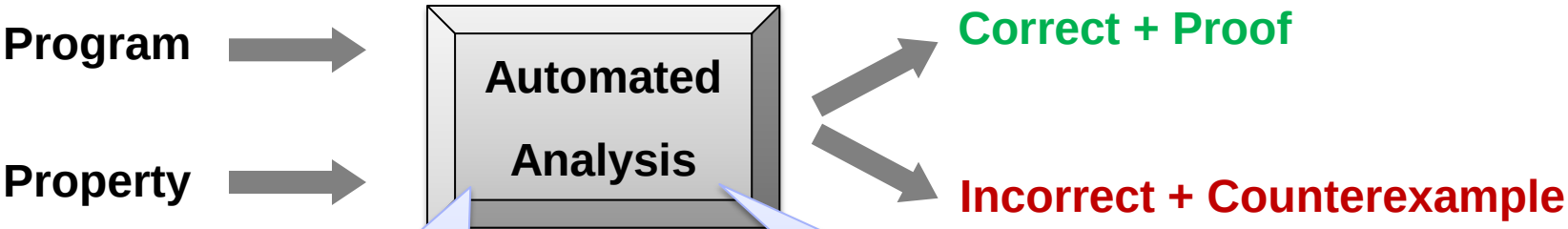
Developer of many verification tools and libraries

- Xchek
- TLQSolver
- Yasm
- Linear Decision Diagrams
- Whale
- UFO
- Vinta
- REK



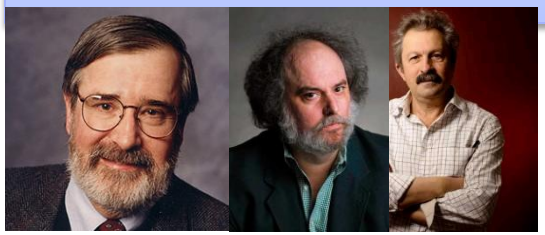
UFO/Vinta won the 2nd Software Verification Competition (SVCOMP)

Automated Software Analysis



Software Model Checking with Predicate Abstraction

e.g., Microsoft's SDV



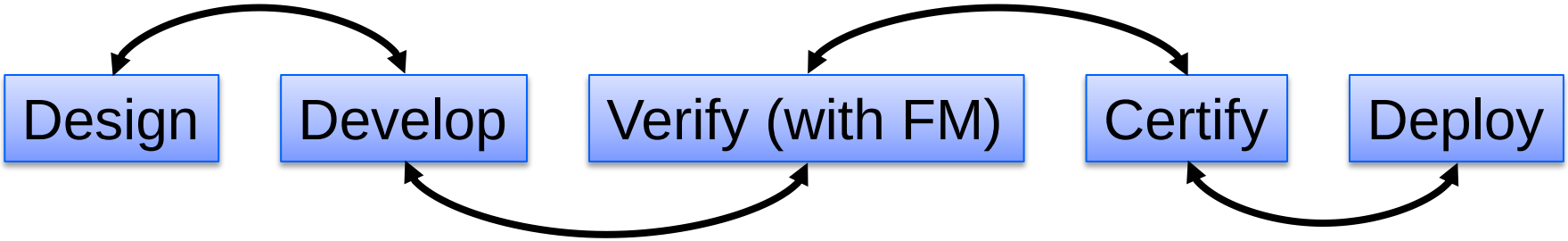
Abstract Interpretation with Numeric Abstraction

e.g., ASTREE, Polyspace

Software Certification: Methods and Tools

Dagstuhl Seminar 13051: http://www.dagstuhl.de/13051

Idealized Development w/ Formal Methods



No expensive testing!

- Verification is exhaustive

Simpler certification!

- Just check formal arguments

Can we trust formal methods tools? What can go wrong?

Trusting Automated Verification Tools

How should automatic verifiers be qualified for certification?

What is the basis for automatic program analysis (or other automatic formal methods) to replace testing?

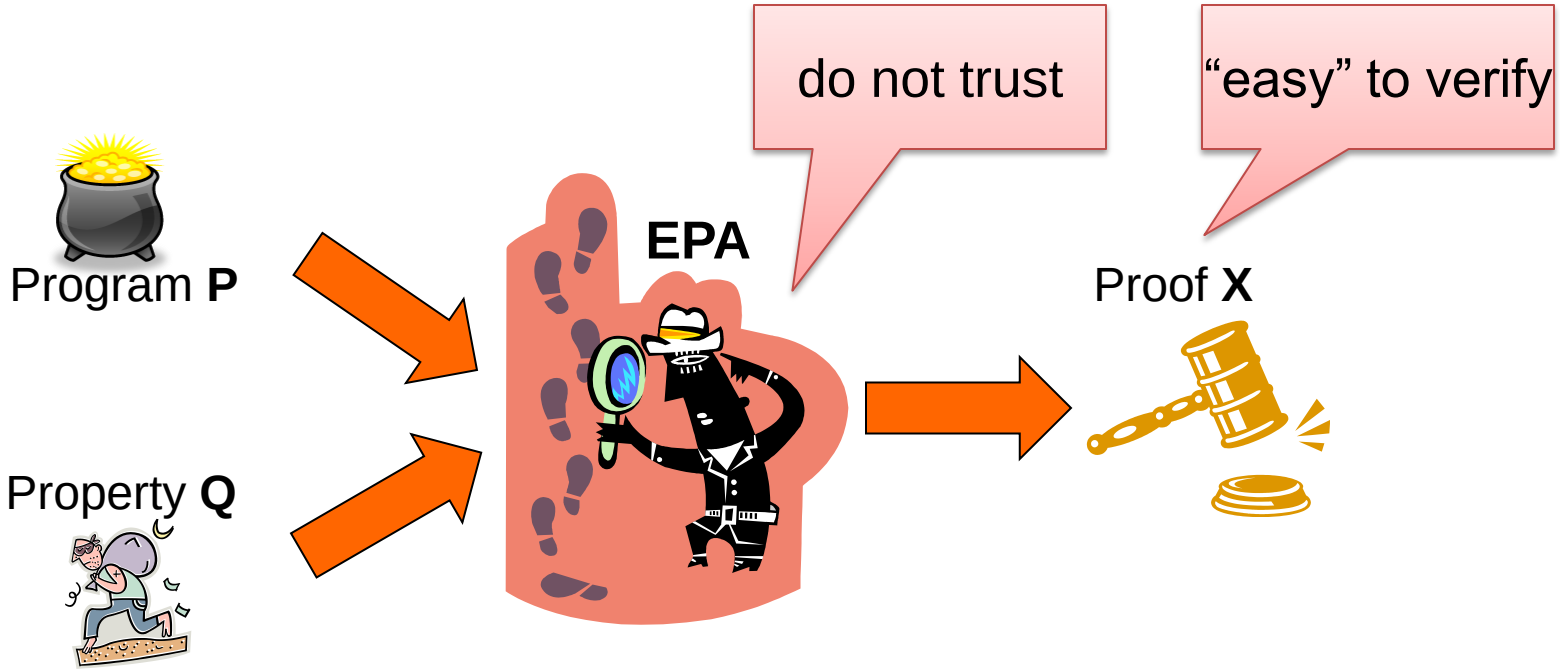
Verify the verifier

- (too) expensive
- verifiers are often very complex tools
- difficult to continuously adapt tools to project-specific needs

Proof-producing (or certifying) verifier

- Only the proof is important – not the tool that produced it
- Only the proof-checker needs to be verified/qualified
- Single proof-checker can be re-used in many projects

Evidence Producing Analysis



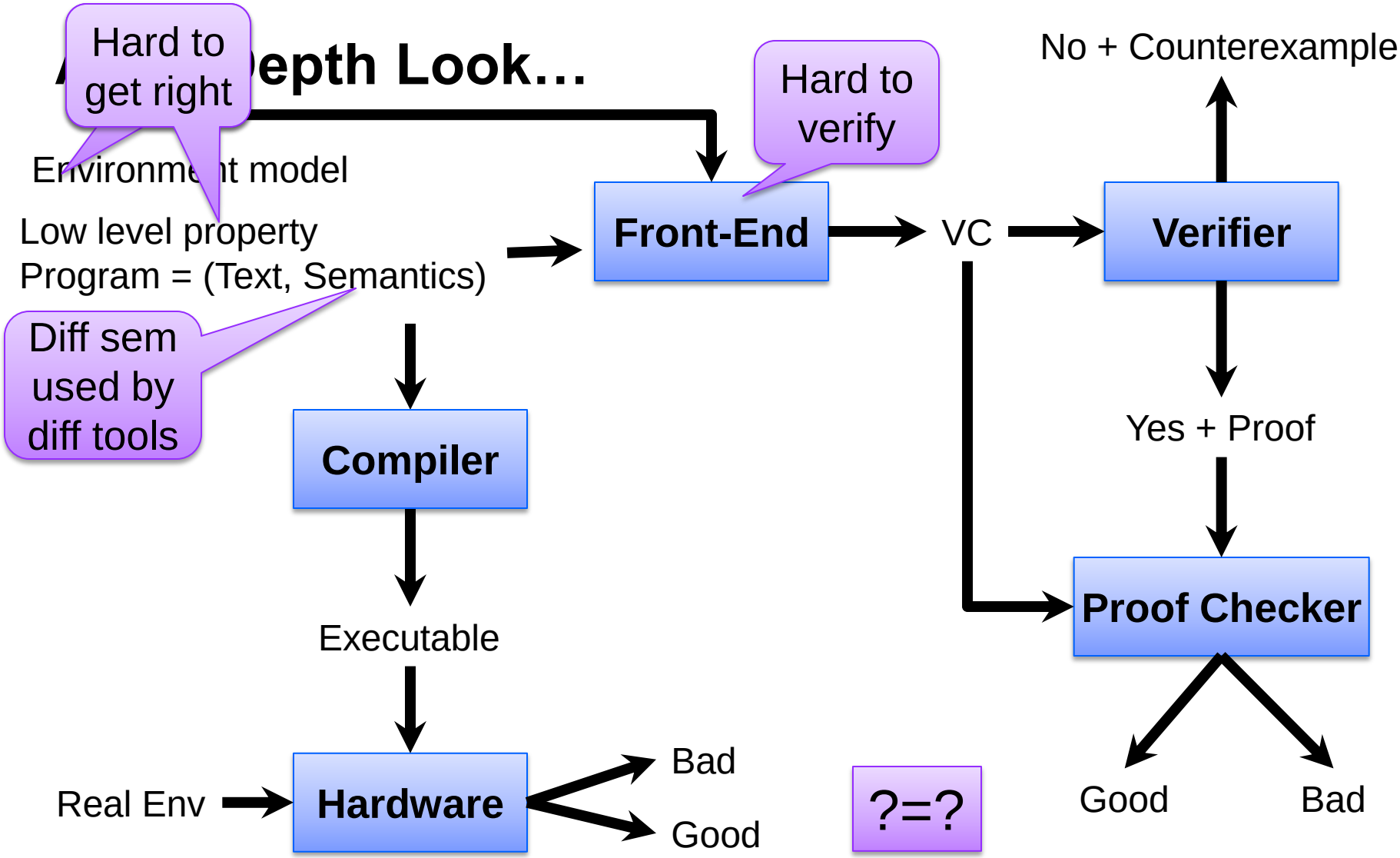
X witnesses that **P** satisfies **Q**. **X** can be objectively and independently verified.
Therefore, **EPA** is outside the Trusted Computing Base (**TCB**).

Active research area

- proof carrying code, certifying model checking, model carrying code etc.
- Few tools available. Some preliminary commercial application in the telecom domain.
- **Static** context. Good for **ensuring absence of problems.**
- Low automation. Applies to source or binary. High confidence.

Not that simple in practice !!!

A Depth Look…



Five Hazards (Gaps) of Automated Verification

Soundness Gap

- Intentional and unintentional unsoundness in the verification engine
- e.g., rational instead of bitvector arithmetic, simplified memory model, etc.

Semantic Gap

- Compiler and verifier use different interpretation of the programming language

Specification Gap

- Expressing high-level specifications by low-level verifiable properties

Property Gap

- Formalizing low-level properties in temporal logic and/or assertions

Environment Gap

- Too coarse / unsound / unfaithful model of the environment

Mitigating The Soundness Gap

Mitigating The Soundness Gap

Proof-producing verifier makes the soundness gap explicit

- the soundness of the proof can be established by a “simple” checker
- all assumptions are stated explicitly

Open questions:

- how to generate proofs for explicit Model Checking
  - e.g., SPIN, Java PathFinder
- how to generate partial proofs for non-exhaustive methods
  - e.g., KLEE, Sage
- how to deal with “intentional” unsoundness
  - e.g., rational arithmetic instead of bitvectors, memory models, …

Mitigating the Property Gap: Vacuity in Model Checking

Joint work with Marsha Chechik

Vacuity: Mitigating Property Gap

Model Checking Perspective: Never trust a *True* answer from a Model Checker

When a property is violated, a counterexample is a certificate that can be examined by the user for validity

When a property is satisfied, there is no feedback!

It is very easy to formally state something very trivial in a very complex way

```
MODULE main

VAR
  send : {s0,s1,s2};
  recv : {r0,r1,r2};

  ack : boolean;
  req : boolean;

ASSIGN
 init(ack):=FALSE;
 init(req):=FALSE;

 init(send):= s0;
 init(recv):= r0;
```

```
next (send) :=
    case
      send=s0:{s0,s1};
      send=s1:s2;
      send=s2&ack:s0;
      TRUE:send;
    esac;

  next (recv) :=
    case
      recv=r0&req:r1;
      recv=r1:r2;
      recv=r2:r0;
      TRUE: recv;
    esac;
```

```
next (ack) :=
   case
     recv=r2:TRUE;
     TRUE: ack;
   esac;

  next (req) :=
    case
      send=s1:FALSE;
      TRUE: req;
    esac;
```

```
SPEC AG (req -> AF ack)
```

Can A TRUE Result of Model Checker be Trusted

Antecedent Failure [Beatty & Bryant 1994]

- A temporal formula AG $(p \Rightarrow q)$ suffers an *antecedent failure* in model M iff $M \models AG\ (p \Rightarrow q)$ AND $M \models AG\ (\neg p)$

Vacuity [Beer et al. 1997]

- A temporal formula φ is satisfied *vacuously* by M iff there exists a sub-formula p of φ such that $M \models \varphi[p \leftarrow q]$ for *every* other formula q
- e.g., $M \models AG\ (r \Rightarrow AF\ a)$ and $M \models AG\ (r \Rightarrow AF\ \neg a)$ and $AG\ (r \Rightarrow AF\ \neg r)$ and $AG\ (r \Rightarrow AF\ FALSE)$, …

Vacuity Detection: Single Occurrence

φ is vacuous in M iff there exists an occurrence of a subformula p such that

- $M \models \varphi[p \leftarrow \text{TRUE}]$ and $M \models \varphi[p \leftarrow \text{FALSE}]$

$$\frac{M \models AG\,(req \Rightarrow AF\ TRUE)}{M \models AG\ TRUE} \qquad \frac{M \models AG\,(req \Rightarrow AF\ FALSE)}{M \models AG\ \neg req}$$

$$\frac{M \models AG\,(TRUE \Rightarrow AF\ ack)}{M \models AG\ AF\ ack} \qquad \frac{M \models AG\,(FALSE \Rightarrow AF\ ack)}{M \models AG\ TRUE}$$

Detecting Vacuity in Multiple Occurrences

Is AG (req ⇒ AF req) vacuous? Should it be?

$$\frac{M \models AG\,(TRUE \Rightarrow AF\ TRUE)}{M \models AG\ TRUE} \qquad \frac{M \models AG\,(FALSE \Rightarrow AF\ FALSE)}{M \models AG\ TRUE}$$

Is AG (req ⇒ AX req) vacuous? Should it be?

$$\frac{M \models AG\,(TRUE \Rightarrow AX\ TRUE)}{M \models AG\ TRUE} \qquad \frac{M \models AG\,(FALSE \Rightarrow AX\ FALSE)}{M \models AG\ TRUE}$$

Detecting Vacuity in Multiple Occurrences: ACTL

An *ACTL* φ is vacuous in M iff there exists an a subformula p such that

- $M \models \varphi[p \leftarrow x]$, where x is a non-deterministic variable

Is AG (req ⇒ AF req) vacuous? Should it be?

$$\frac{M \models AG\ (x \Rightarrow AF\ x)}{M \models AG\ TRUE}$$

Always vacuous!!!

Is AG (req ⇒ AX req) vacuous? Should it be?

$$\frac{M \models AG\ (x \Rightarrow AX\ x)}{\text{can't reduce}}$$

Not vacuous!!!

Mitigating the Environment Gap:

Environment Guarantee

Joint work with Marsha Chechik and Mihaela Gheorghiu

Validity of Vacuity Results

Properties may hold for wrong reasons

vacuity detection [Beer'97] – are all parts of property relevant?

- Example: "every time there is a request for a resource, it is fulfilled"
- ... holds if there are no requests!

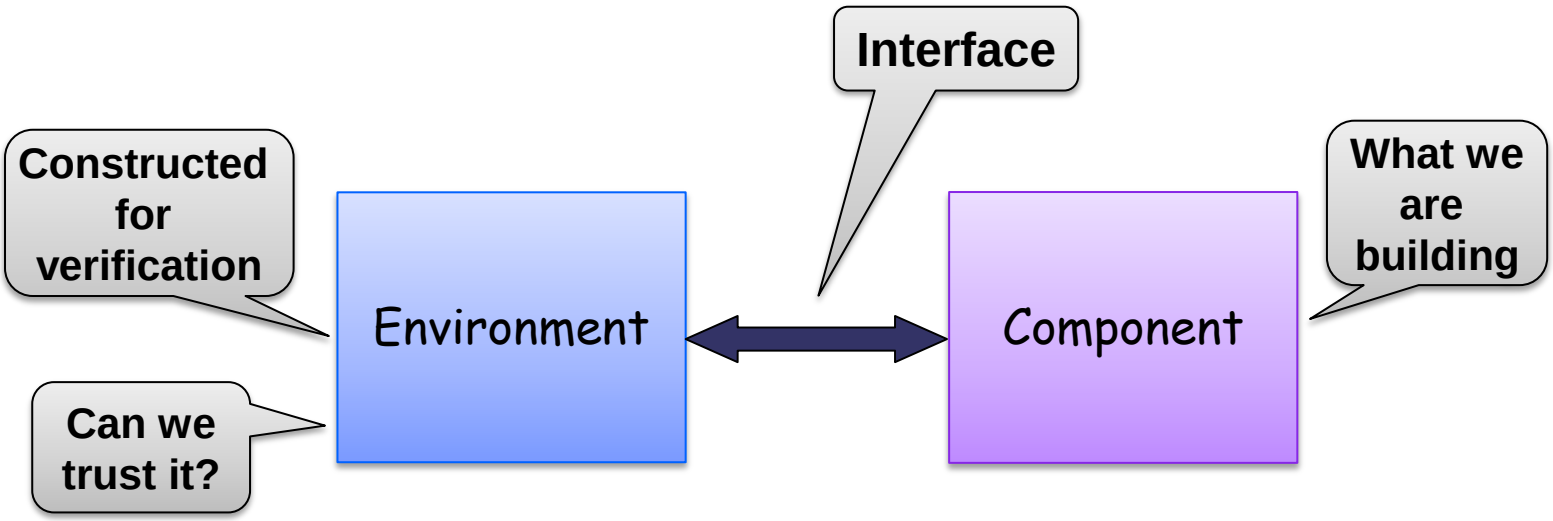
Pros

- May help identify errors

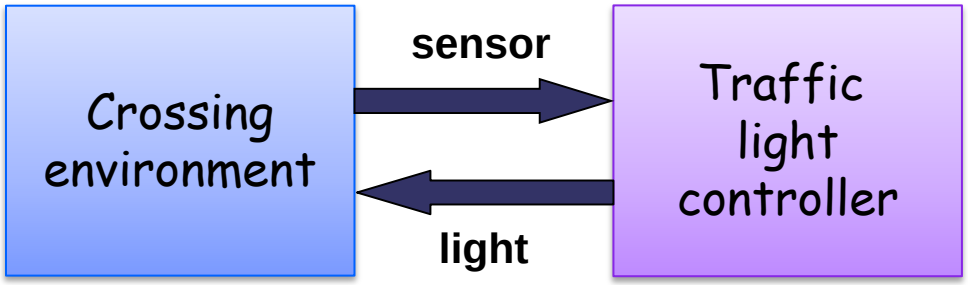
Cons

- General definition yields many false positives
  - Vacuous but not "wrong"
- Hard to go from a violation to a fix
- Property-centric

System Structure



Traffic Light Example:



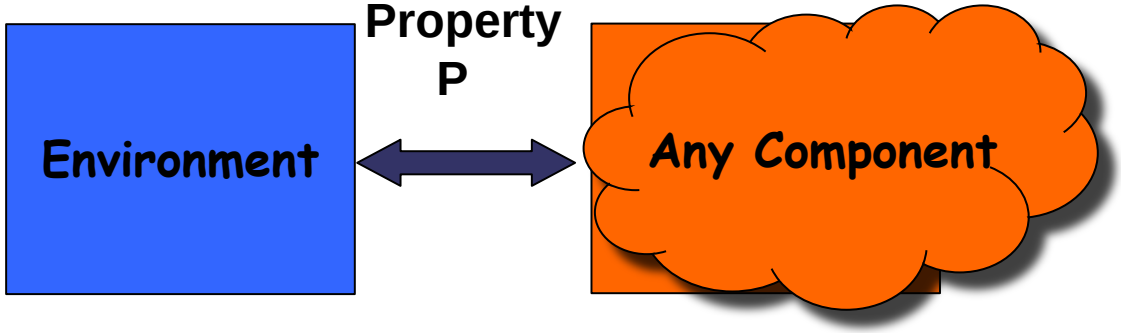
Our Goal

Model-centric identification of threats to validity

Pros:

- Ease of understanding
- No false positives
  - each reported error is present in the model

<u>Definition:</u> a property is guaranteed by environment if component is irrelevant for verification



Formalizing Environment Guarantees

An Environment E is a tuple (V, R)

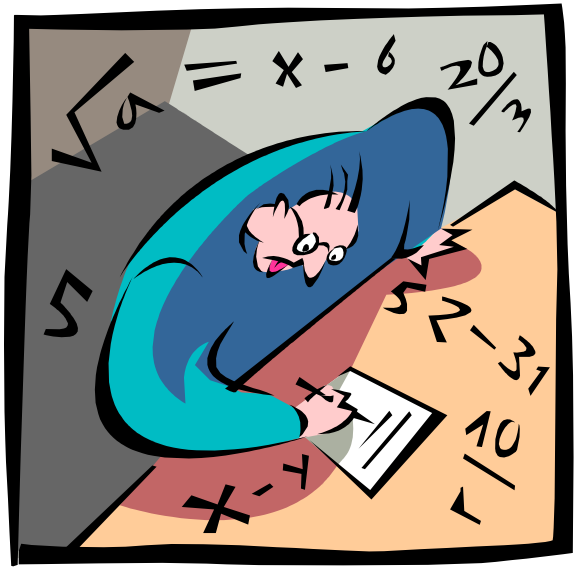
- V - environment and component variables (Ve,Vc)
- R – environment rules

A (Closed) Model M is a tuple (V, R, C)

- ... a closure of E with component rules C

Formal Definition:

- Environment E guarantees a property P iff
- for all closures M of E, M satisfies P



Special Case: Universal Properties

Universal properties: about all executions

- AG p ("in all states")
- AF p ("on all paths")

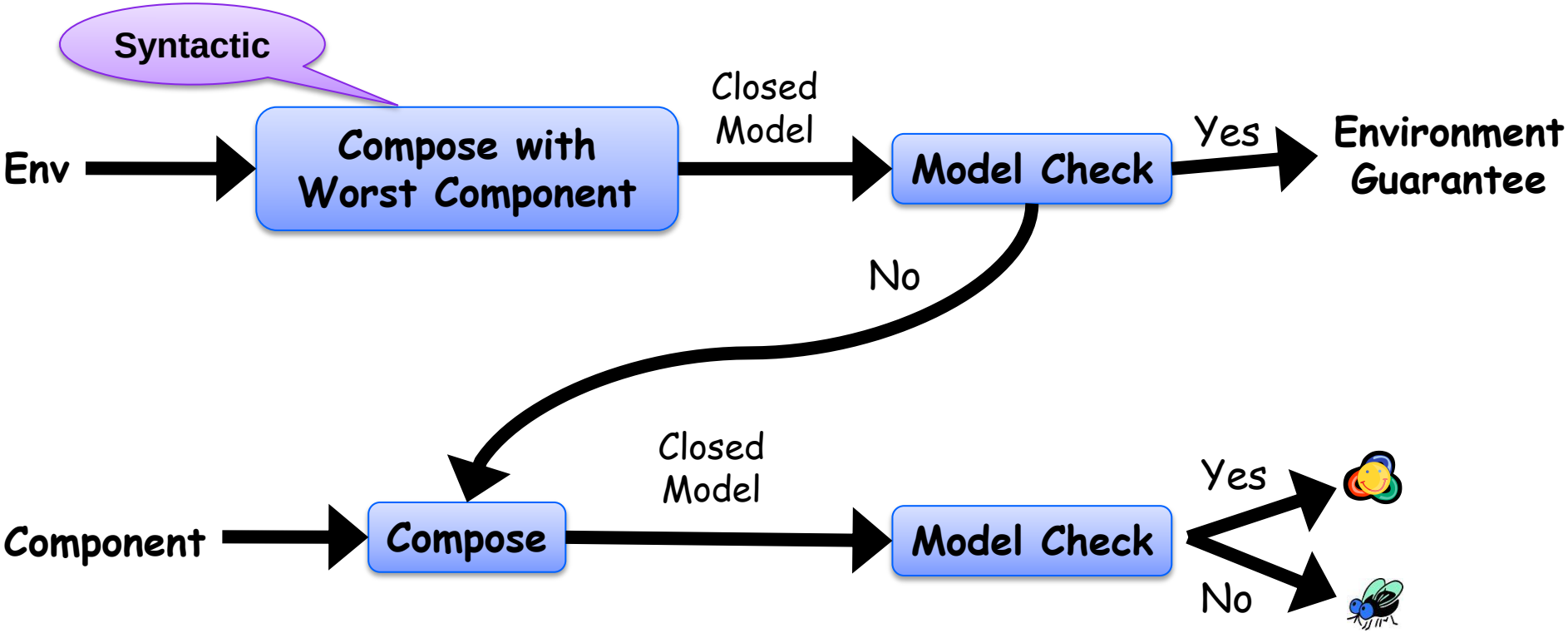
"Worst" component is the most non-deterministic

- Component variables change non-deterministically at each step
- = No constraints on component variables

Theorem

- A universal property is guaranteed by the environment iff it holds under the worst component.

Universal Properties: Implementation



Case Study: TCAS II

[Leveson *et al.* 94]

(Air) Traffic Collision Avoidance System

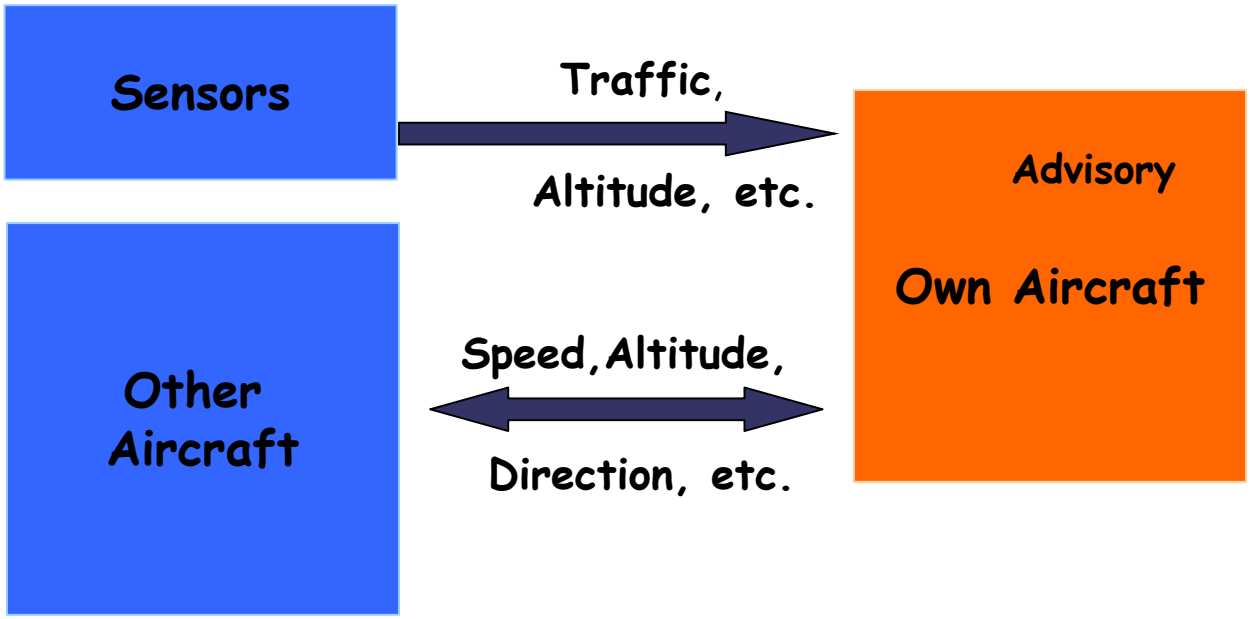
- Avoids collision between planes flying in the same space
- … by producing advisories to pilot for direction and strength of move

An example advisory:

- “CLIMB at 1500 to 2000 fpm”

TCAS II Model

NuSMV model [Chan et al. 98]



TCAS II Properties

Deterministic Advisory

- advisory changes in a deterministic fashion

Increase Climb/Descent

- increase climb does not change immediately to increase descent

Direction of the Other Aircraft

- direction does not change unless noticed by Own Aircraft

Advisories are Consistent

- direction (Up/Down) and strength (Pos/Neg) match

Experimental Results

Properties	Results		Time (sec.)		BDD nodes	
	Full	Env.	Full	Env.	Full	Env.
Reachability	-	-	1034.61	4.2	1349878	145246
Deterministic Advisory	TRUE	TRUE	20.63	3.8	173041	37846
Increase Climb/Descend	TRUE	TRUE	20.81	3.84	175280	37991
Other Aircraft Direction	TRUE	TRUE	27.83	3.87	341216	38352
Consistent Advisory	TRUE	FALSE	40.67	6.1	224611	39709

Mitigating the Semantic Gap

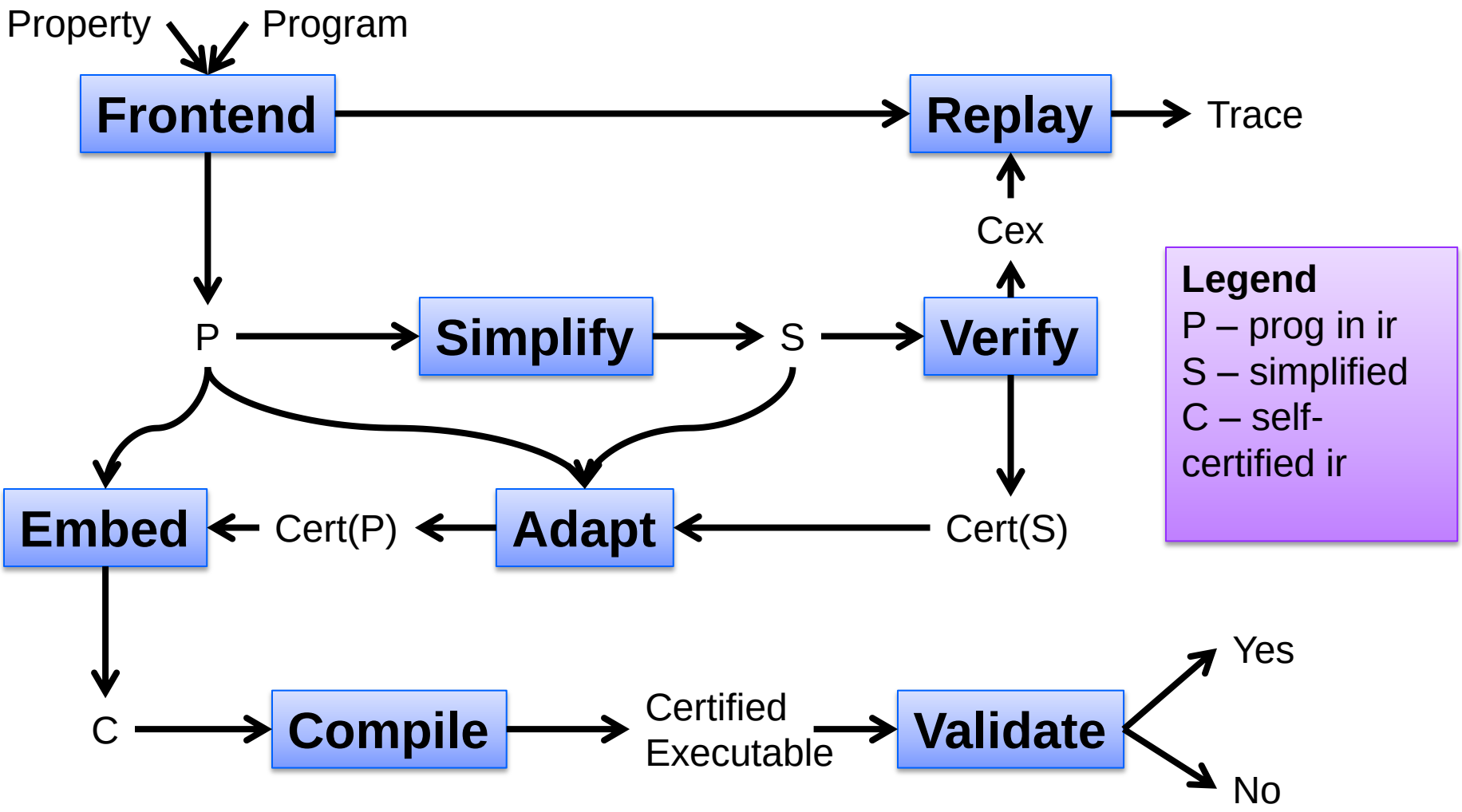
Mitigating the Semantics Gap

Combine a compile and a verifier in a single architecture

Use compiler intermediate representation as the “ground truth”

Propagate verification certificates from the verifier down to the compiled code

V&C Architecture (Verifier & Compiler)



Proofs are "Witnesses" to Success*

Program invariants are embedded as BEGIN and INV calls in compiled code.

```
while(n < 10) {
   BEGIN();

   INV((n >= 0) && (n < 10));

   n = n + 1;
}
```

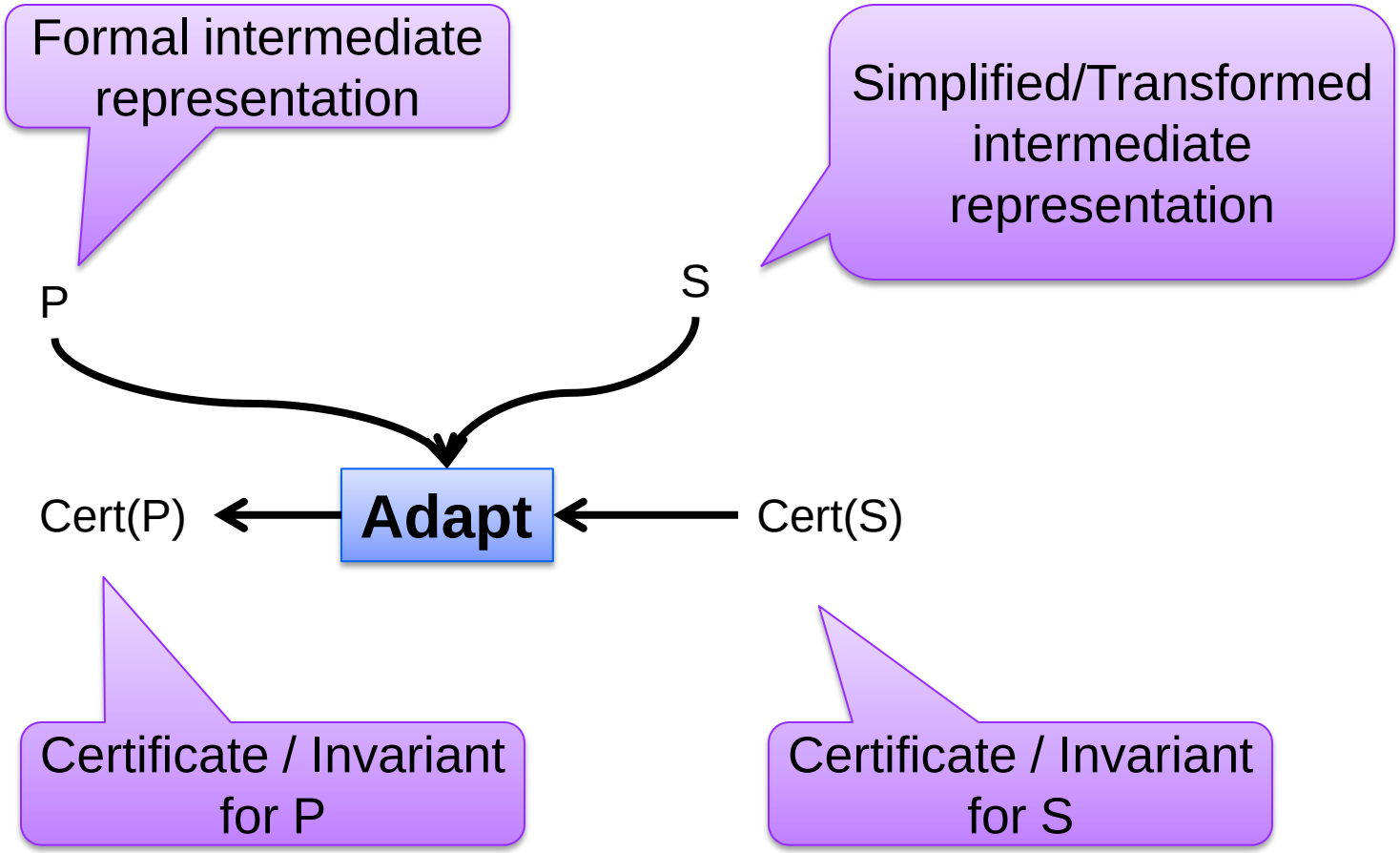
```
lwz %r0,8(%r31)
cmpwi %cr7,%r0,9
bgt %cr7,.L4
bl BEGIN
li %r0,0
stw %r0,16(%r31)
lwz %r0,8(%r31)
cmpwi %cr7,%r0,0
blt %cr7,.L5
lwz %r0,8(%r31)
cmpwi %cr7,%r0,9
bgt %cr7,.L5
li %r0,1
stw %r0,16(%r31)
.L5:
lwz %r3,16(%r31)
crxor 6,6,6
bl INV
lwz %r9,8(%r31)
addi %r0,%r9,1
stw %r0,8(%r31)
b .L3
```

These invariants are sufficient to construct a proof of conformance to a claim (policy)

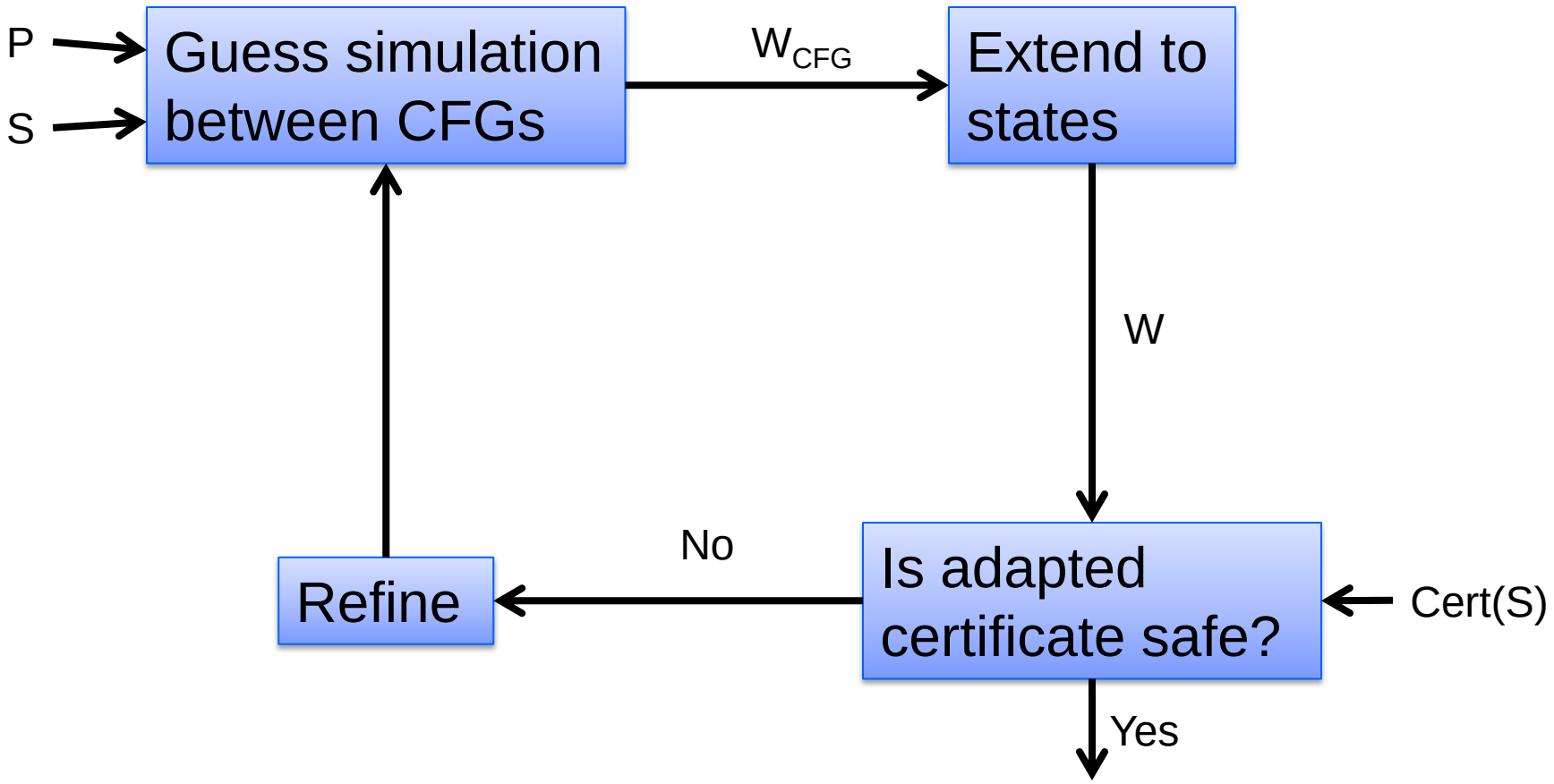
Embed

*Chaki, S., Ivers, J., Lee, P., Wallnau, K., Zeilberger, N., "Model-Driven Construction of Certified Binaries", Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, 2007, LNCS 4735, pp 666-681.

Adapt: From Low-Level to High-Level Certificate



Iteratively Guess a Simulation and Adapt Cert.



Five Hazards (Gaps) of Automated Verification

Soundness Gap

- Intentional and unintentional unsoundness in the verification engine
- e.g., rational instead of bitvector arithmetic, simplified memory model, etc.

Semantic Gap

- Compiler and verifier use different interpretation of the programming language

Specification Gap

- Expressing high-level specifications by low-level verifiable properties

Property Gap

- Formalizing low-level properties in temporal logic and/or assertions

Environment Gap

- Too coarse / unsound / unfaithful model of the environment

Contact Information

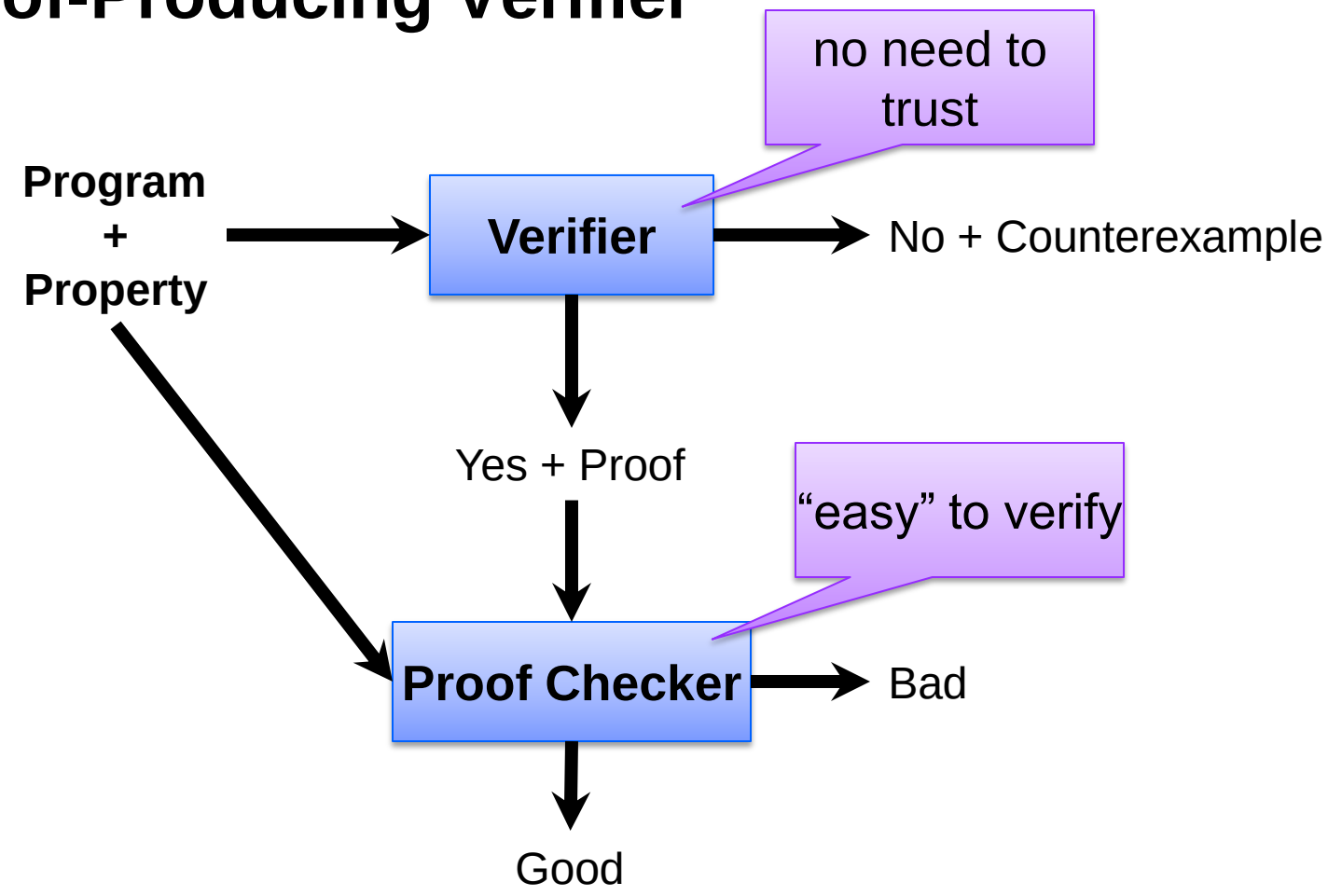
Presenter
Arie Gurfinkel
RTSS
Telephone: +1 412-268-5800
Email: arie@cmu.edu

U.S. mail:
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Web:
www.sei.cmu.edu
http://www.sei.cmu.edu/contact.cfm

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257

Proof-Producing Verifier



But things are not that simple in practice !!!