Assurance Cases for Formally Demonstrated Conformance Relations

Arie Gurfinkel
Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University

joint work with Sagar Chaki, Kurt Wallnau, and Charles Weinstock

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Value of Conformance Evidence: A Proposition

Two Axioms*:

- *Evidence* that software satisfies (implements) claimed behavior has economic value
- The value of evidence increases as it moves nearer to binary (executable) code.

Assumption for this presentation:

- We care about demonstrating behavioral conformance, i.e., software does or does not behave according to some specified policy

Definitions:

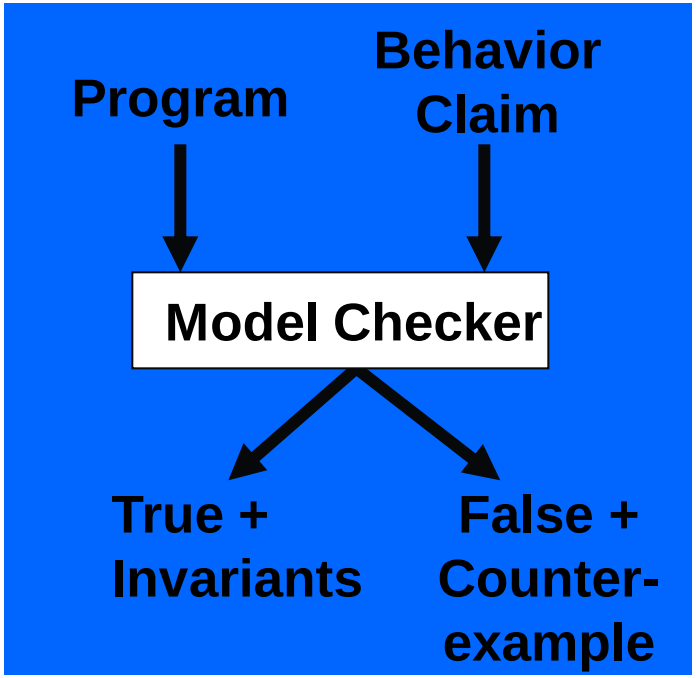
- A claim asserts a behavior, typically some combination of safety (behavior X *never* happens) and liveness (X *eventually* happens)
- An invariant is a condition that holds for all program executions

See http://www.sei.cmu.edu/pacc/ as background, and http://www.sei.cmu.edu/pacc/dowloads.html for publicly released software described in this presentation.

Certifying Software Model Checking (CSMC)

What is software model checking?

- Type of static analysis that aims to provide precise answers to “deep” claims about the runtime behavior of programs
- Provides an error trace if a failure is found, and in some cases a proof certificate that a claim is satisfied*
- Traditionally used by hardware industry e.g., Intel, IBM, Cadence (non-certifying)
- Increasingly used in the software industry e.g., Microsoft, NEC (non-certifying)
- Exhaustive & automated, but scale is a real challenge, especially for software

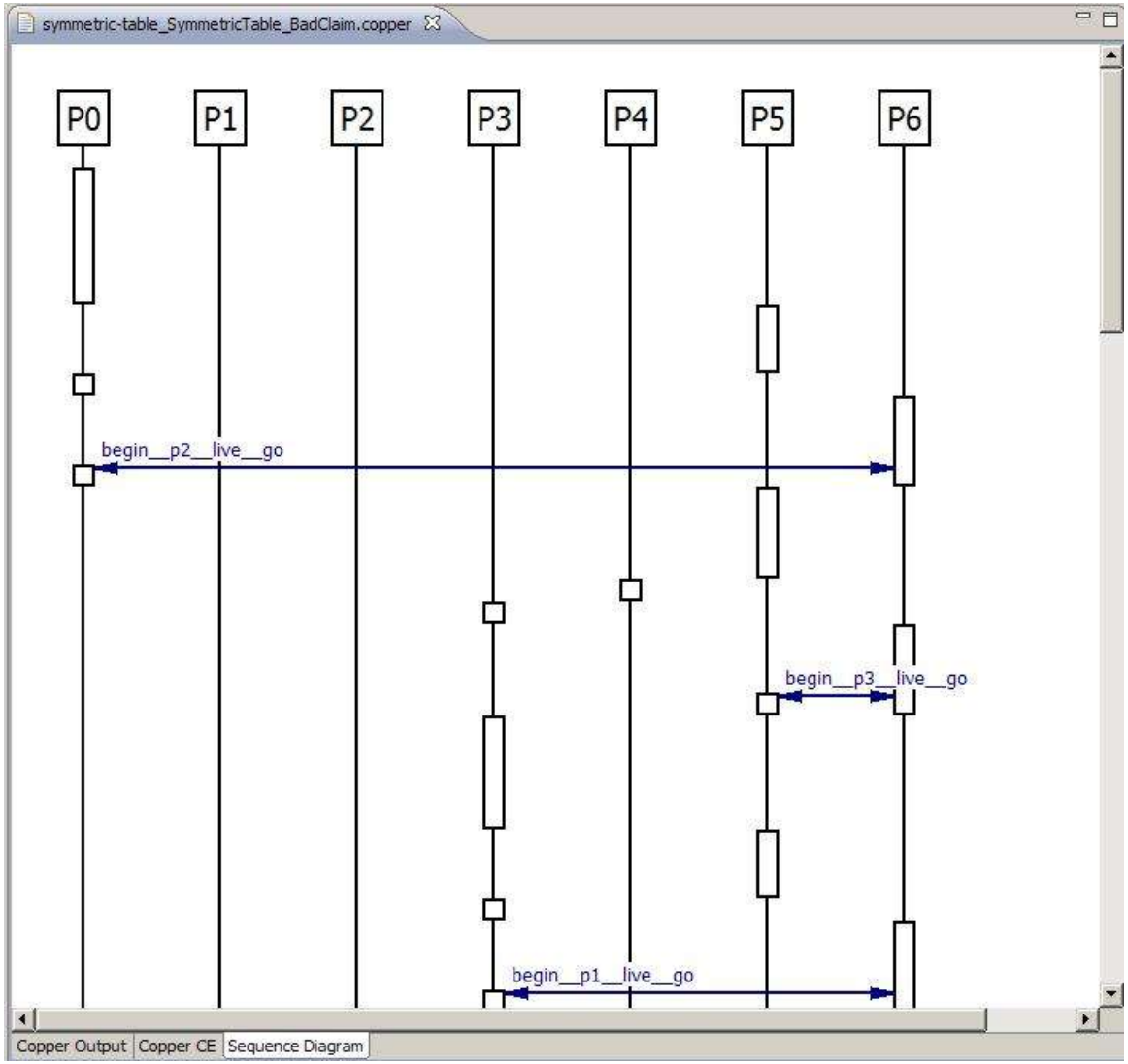


*Chaki, S., "SAT-Based Software Certification". pp. 151 - 166. Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006). Vienna, Austria, March 25 - April 2, 2006. Notes in Computer Science, Volume 3920, 2006.

Typical “Claims” Checked by Model Checker

1. An assertion deep in the program is never violated
   1. A string buffer that is manipulated in all sorts of tricky ways does not overflow.
   2. A null pointer is never dereferenced
2. A program follows a complicated API usage pattern
   1. A lock is never acquired twice without being released
3. Concurrency is used correctly
   1. Absence of deadlocks/livelocks
4. Every request is eventually serviced
   1. A program always accepts a reset command and returns to a safe state
   2. A device driver always returns control to the operating system
5. Secure information flow
   1. A hospital management software does not show a patient’s SSN to a nurse
   2. Amazon does not show credit card information of one customer to another

Counterexamples are “Witnesses” to Failure



Model checkers describe execution paths through a transition system when they encounter failures.

These counterexamples can be mapped back to the original code and can be confirmed.

Confirmation is usually quite simple, and can often be reproduced by a test case.

We do not need to trust the model checker since we have a witness.

Certificate Production

Program
P

Certifying
SMC

Policy
S

Invariant

$VC\text{-}Gen_P$

Proof
Pr

Theorem
Prover

VC_P

Trusted
Computing
Base

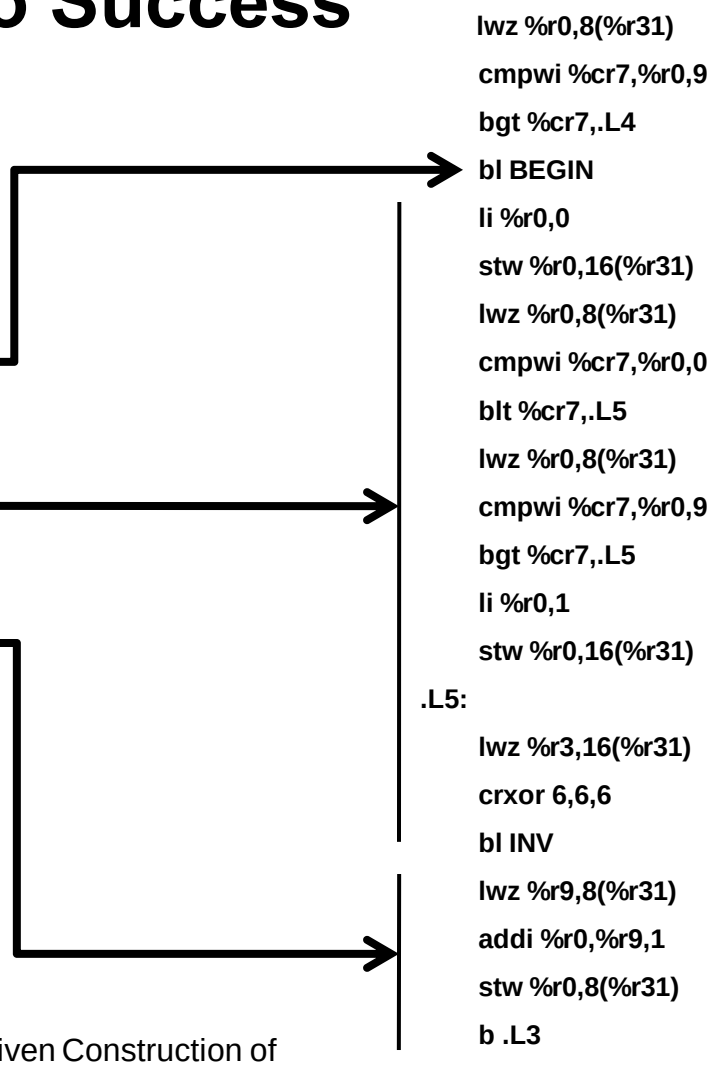
Proofs are "Witnesses" to Success*

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

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

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

   n = n + 1;
}
```



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

*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.

Scenario

You are a consumer who has been presented with executable software

- The software includes an embedded proof of claim C

The vendor asserts C is satisfied and that you do not need to trust

- the developers (their identities or their motives)
- the processes developers used to design or build the software
- the compiler or any tools used to design or build the software
- the software model checker that supplied proof invariants
- any tools (theorem provers, etc.) used to construct the proof
- the security of any distribution channels, i.e, if code or proof is tampered with:
  - proof checking will fail, or
  - the code conforms to policies in C

Do you believe the vendor?

Assurance Cases

Evidence without argument is unexplained

Argument without evidence is unfounded

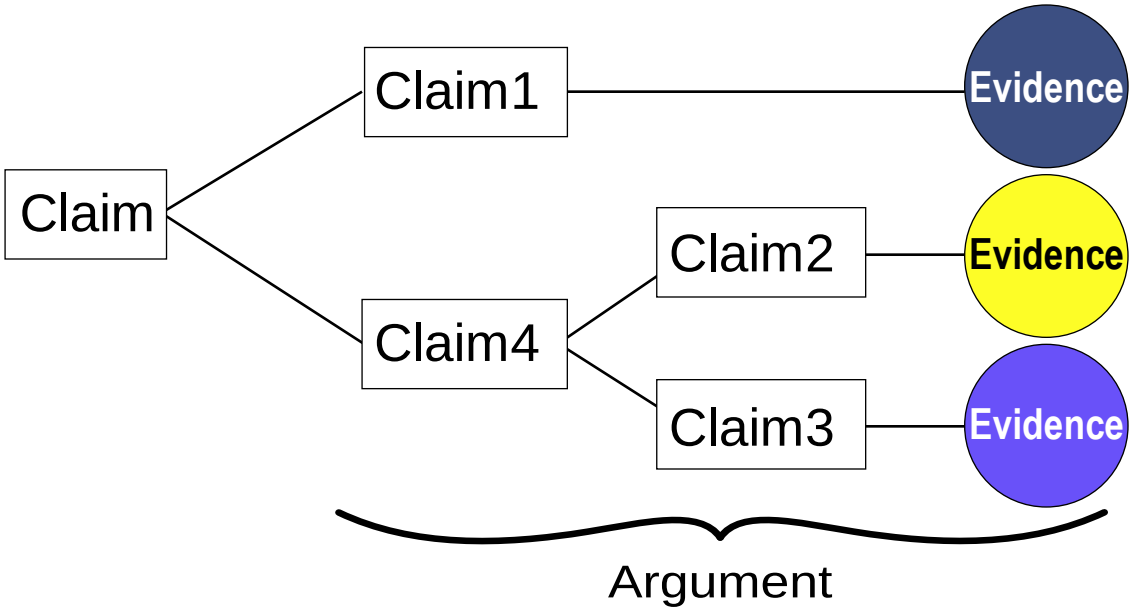
An assurance case requires claims, evidence, and an argument

- Evidence
  - Results of observing, analyzing, testing, simulating, and estimating the properties of a system that provide fundamental information from which the presence of some system property can be inferred
- Argument
  - Explanation of how the available evidence can be reasonably interpreted as indicating acceptable operation, usually by demonstrating compliance with requirements, sufficient and avoidance of hazards, etc.

What is an Assurance Case?

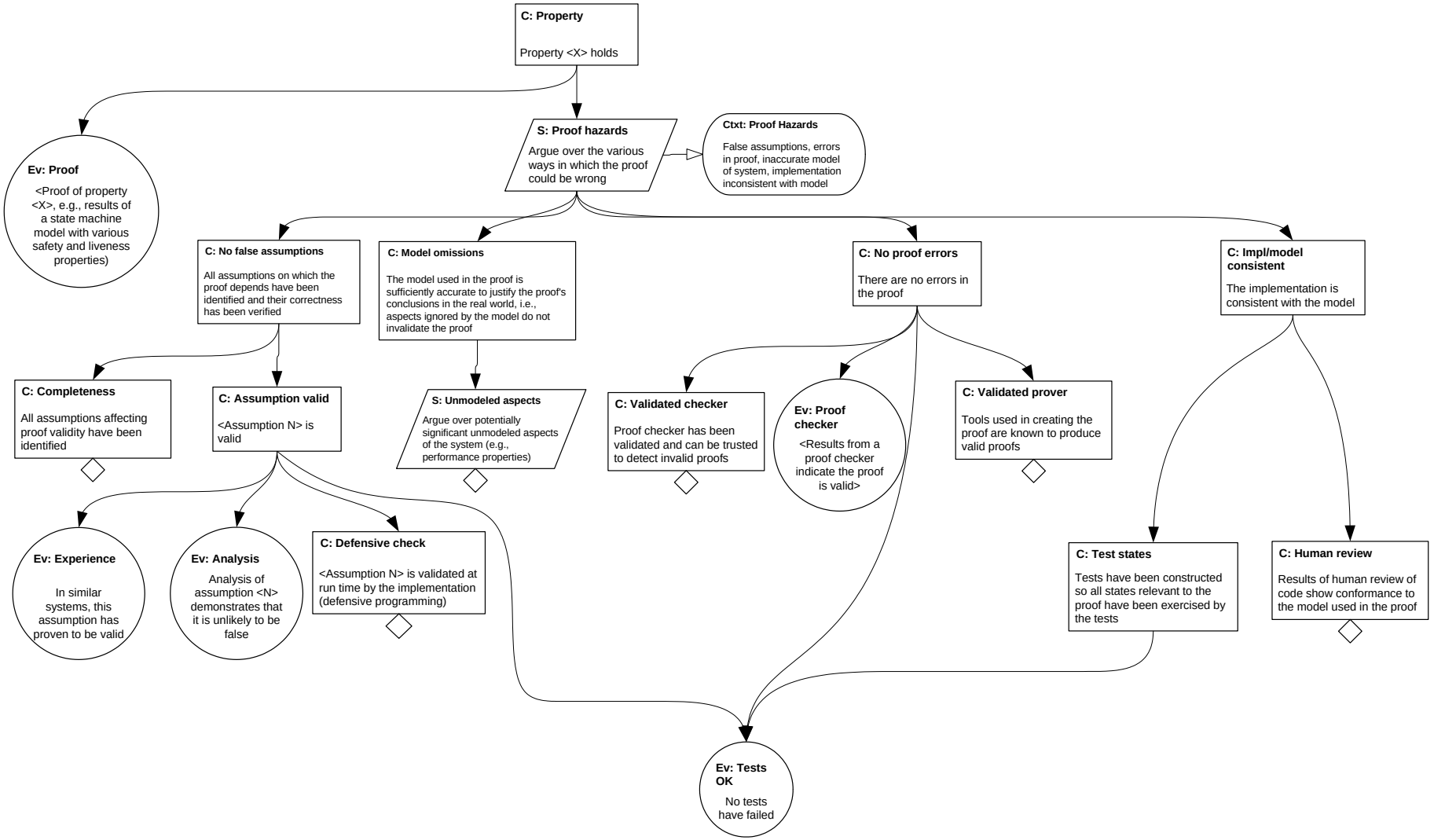
A structured demonstration that a system is acceptably safe, secure, reliable, etc.

- A comprehensive presentation of **evidence** linked (by **argument**) to a **claim** of some policy



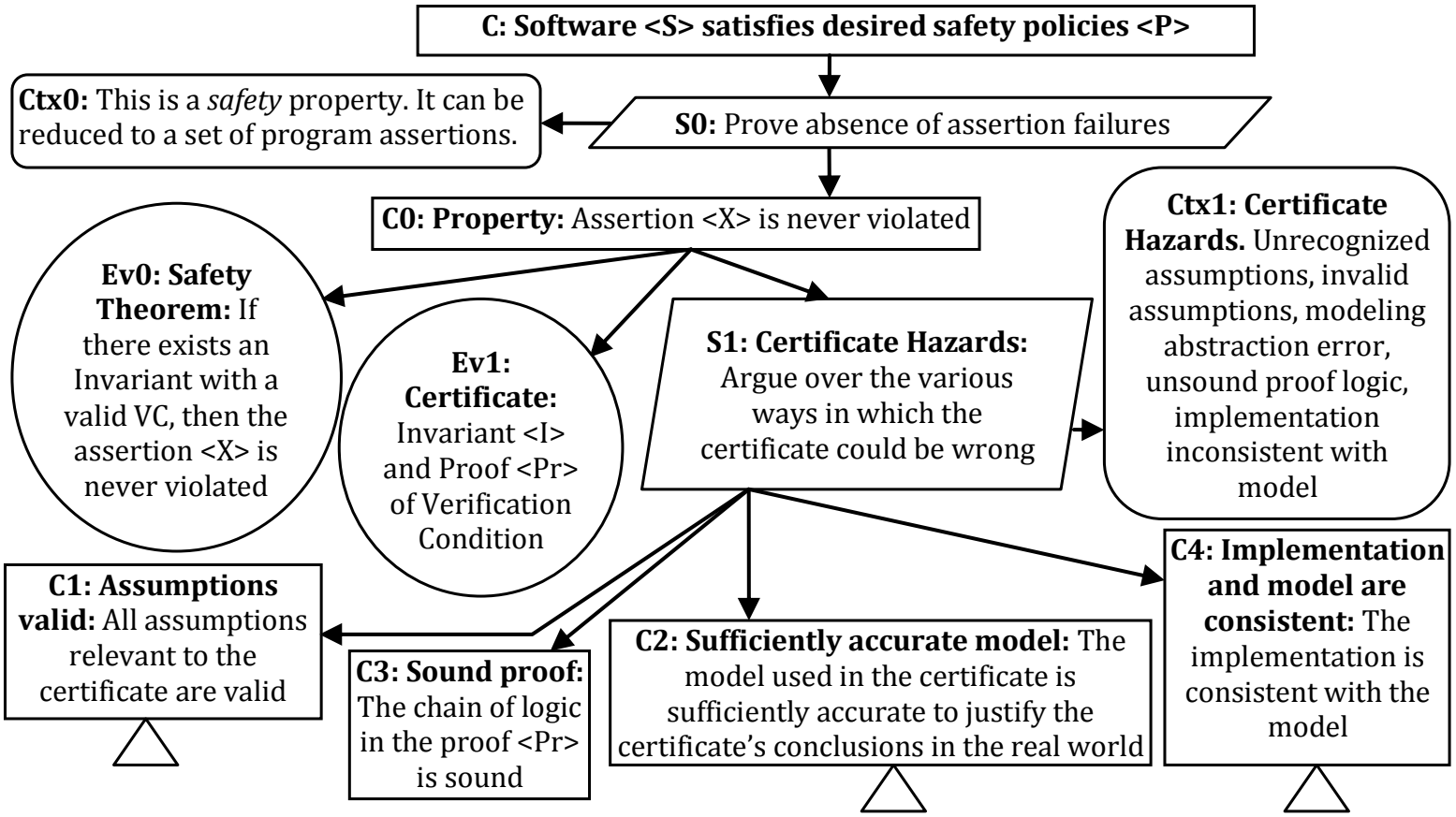
IF ● **THEN** Claim1; **IF** ● **THEN** Claim2; **IF** ● **THEN** Claim3;
IF Claim2 **and** Claim3 **THEN** Claim4; **IF** Claim1 **and** Claim4 **THEN** Claim

An Assurance Pattern for Proof Evidence*

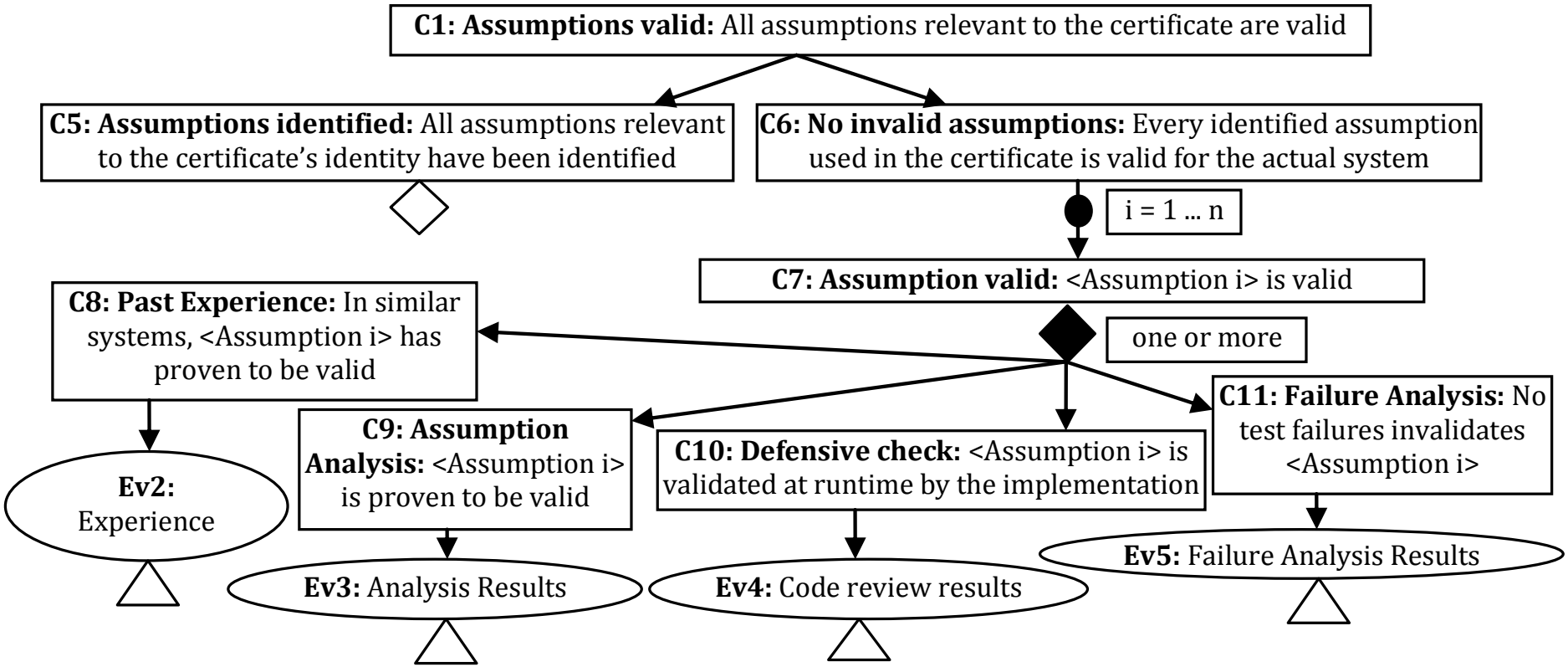


*Goodenough, J., Weinstock, C., "Hazards to Evidence: Demonstrating the Quality of Evidence in an Assurance Case", Technical Note CMU/SEI-2008-TN-016 , Software Engineering Institute, Carnegie Mellon University, 2008 (in preparation)

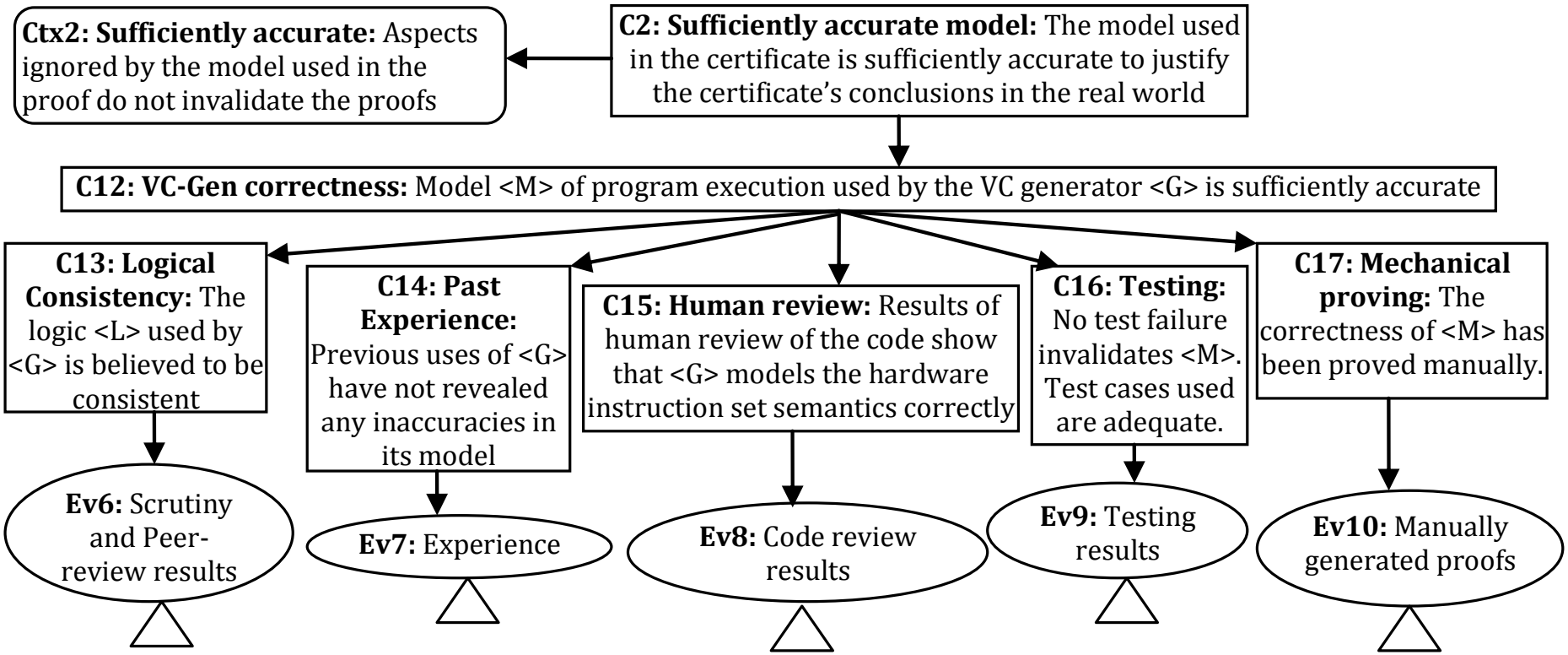
Top-Level Assurance Case



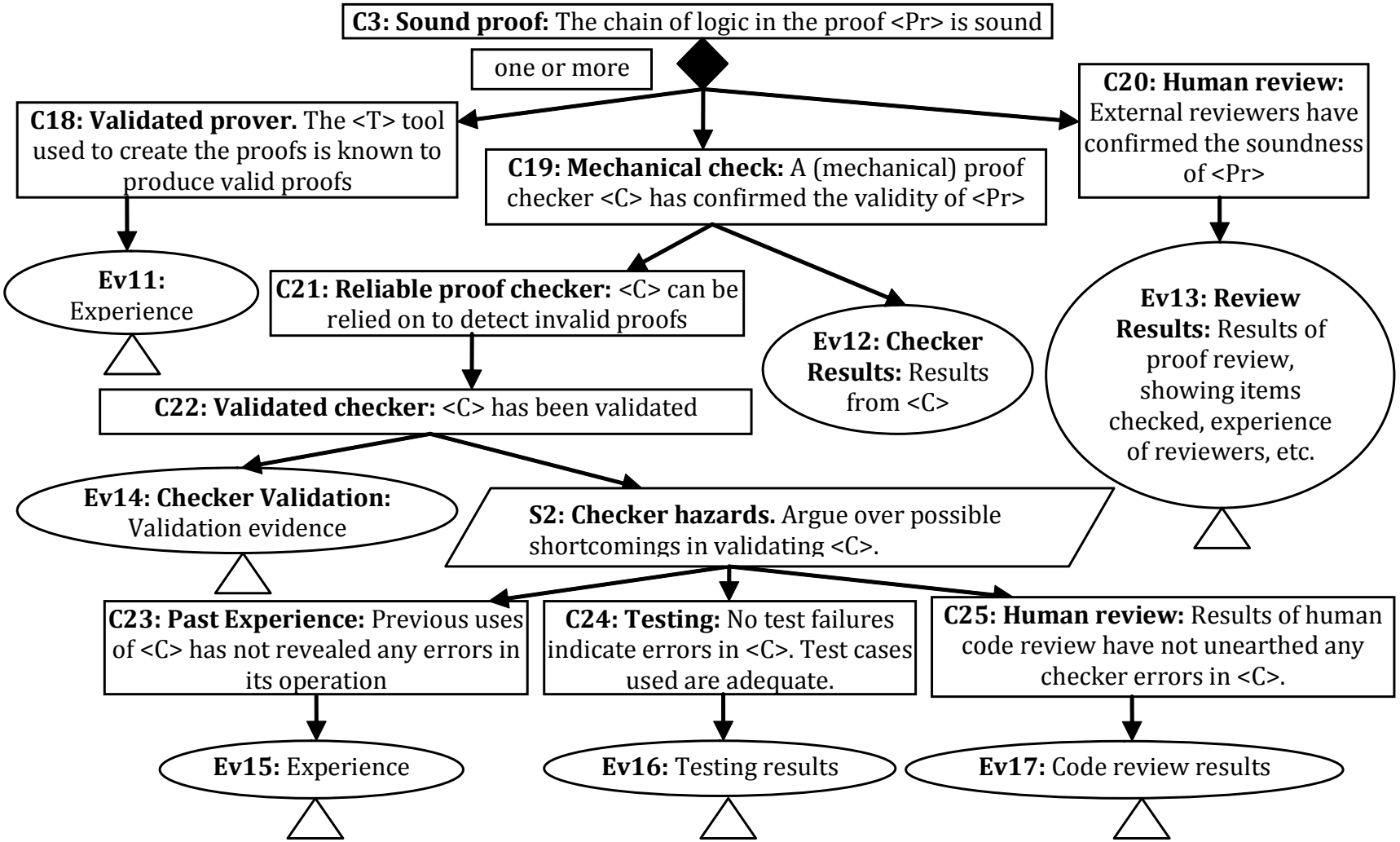
C1: Assumptions valid



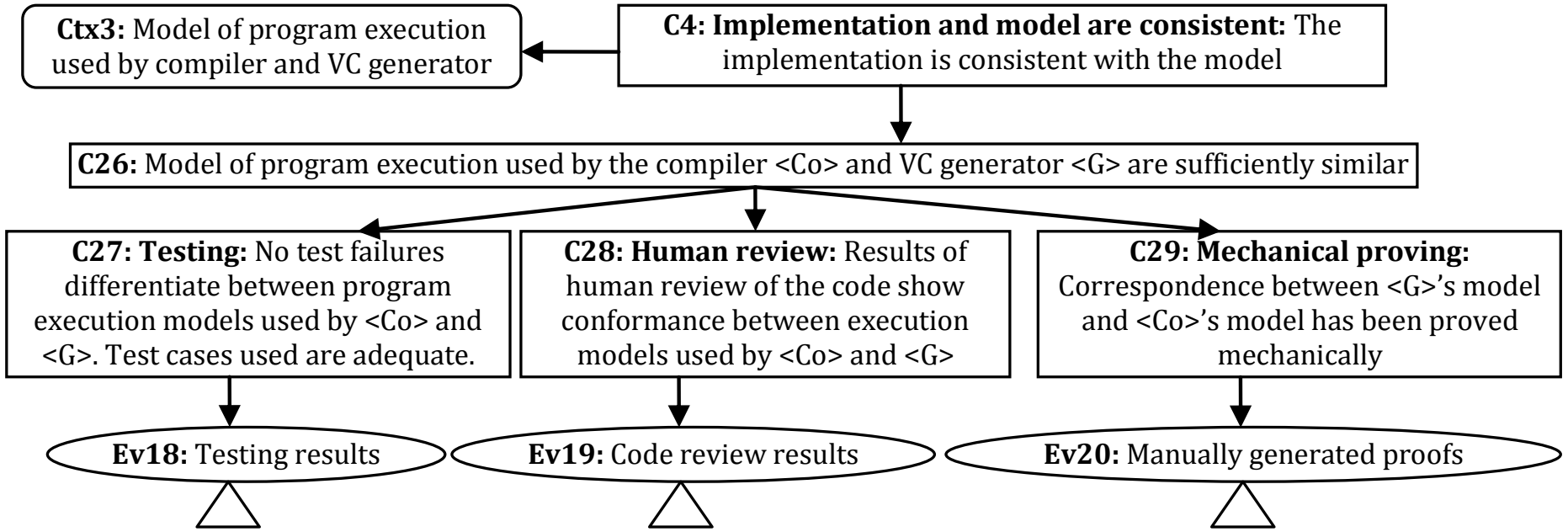
C2: Sufficiently Accurate Model



C3: Sound Proof



C4: Implementation and Model are Consistent



What Might This Mean for Certification?

A claim of conformance can be only as strong as the policy is precise

- Good news: we know how to specify machine-checkable behavior
- Bad news: it can be costly to specify precisely required software behavior

Software producers can shoulder the "burden of proof" for conformance

- Good news: proof-strength evidence can be mechanized for small but critical code (e.g., embedded software, trusted security kernel, …)
- Bad news: software acquisition and development practice must change if we are to generate, and use evidence of software quality in a cost-effective way

How much evidence is enough, and how strong must the evidence be?

- A proof is not a proof unless it is believed to be a proof, and different standards of "*believability*" obtain in different settings
- Assurance cases are one way of calibrating "*believability*"

END