Open Group, Sydney, Australia, 17 April 2013
based on Dagstuhl Seminar 13051, January 2013
Software Certification: Methods and Tools

Logic and Epistemology in Assurance Cases

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Assurance Cases as a Framework for Certification

- No matter how certification is (or should be) actually organized and undertaken. . .
- We can describe, understand, and evaluate it within the framework of an assurance case
    - Claims
    - Argument
    - Evidence
- For example, in objectives-based guidelines such as DO-178C, the claims are largely established by regulation, guidelines specify the evidence to be produced, and the argument was presumably hashed out in the committee meetings that produced the guidelines
    - Though absent a documented argument, it's not clear what some of the evidence is for: e.g., MC/DC testing)
    - Need to reconstruct the argument for purpose of evaluation (FAA has tasked NASA to do this)

My Dream

- Is to be able to evaluate the certification argument for a system by systematic and substantially automated methods
  - cf. Leibniz' Dream: "let us calculate"
- So that the precious resource of human insight and wisdom can be focused on just the areas that need it
- Also a step toward an intellectual foundation for certification arguments
- Caveat: I'm concentrating on (functional and safety) requirements
  - All aviation software incidents arise in the transition from system to software requirements
  - Implementation assurance is fairly well managed, modulo "derived requirements"

Assurance Cases and Verification

- The argument aims to justify the claims, based on the evidence
- This is a bit like logic
    - A proof justifies a conclusion, based on given assumptions and axioms
- Formal verification provides ways to automate the evaluation, and sometimes the construction, of a proof
- So what's the difference between an assurance case and a formal verification?
- An assurance case also considers why we should believe the assumptions and axioms, and the interpretation of the formalized assumptions and claims
- As an exercise, consider my formal verification in PVS of Anselm's Ontological Argument (for the existence of God)

Logic And The Real World

- Software is logic
- But it interacts with the world
  - What it is supposed to do (i.e., requirements)
  - The actual semantics of its implementation
  - Uncertainties and hazards posed by sensors, actuators, devices, the environment, people, other systems
- So we must consider what we know about all these
- That's epistemology

Epistemology

- This is the study of knowledge
- What we know, how we know it, etc.
  - Traditionally taken as justified true belief
  - But that's challenged by Gettier examples
  - And other objections
  - So there are alternative characterizations
  - e.g., . . . obtained by a generally reliable method (Ramsey)
- I'd hoped that philosophy would provide some help
  - It does provide insight and challenges
  - Philosophy of law, in particular, raises relevant issues
  - But no answers
- At issue here is the accuracy and completeness of our knowledge of the world
  - Insofar as it interacts with the system of interest
  - Seems an engineering question, not philosophical

Logic and Epistemology in Assurance Cases

- We have just two sources of doubt in an assurance case
- Logic doubt: the validity of the argument
    - Can be eliminated by formal verification
    - Subject to caveats discussed elsewhere
    - Automation allows what-if experimentation to bolster reviewer confidence
    - We can also allow "because I say so" proof rules
- Epistemic doubt: the accuracy and completeness of our knowledge of the world in its interaction with the system
    - This is where we need to focus
- Same distinction underlies Verification and Validation (V&V)

Epistemology And Models

- We use formal verification to eliminate logic doubt
- That means we must present our assumptions in logic also
- This is where and how we encode our knowledge about the world
    - As models described in logic
- So our epistemic doubt then focuses on these models

Sometimes Less Is More

- Detail is not necessarily a good thing
- Because then we need to be sure the detail is correct
- For example, Byzantine faults
  - Completely unspecified, no epistemic doubt
- *vs.* highly specific fault models
  - Epistemic doubt whether real faults match the model

An Aside: Resilience

- To some extent, it is possible to trade epistemic and logic doubts
  - Weaker assumptions, fewer epistemic doubts
  - *vs.* more complex implementations, more logic doubt
- I claim resilience is about favoring weaker assumptions
- And it is the way of the future

Reducing Epistemic Doubt: Validity

- We have a model and we want to know if it is valid
- One way is to run experiments against it
- That's why simulation models are popular (e.g., Simulink)
- But models that support simulation are not so useful in formal verification nor, I think, in certification
    - To be executable, have to include a lot of detail
    - But our task is to describe assumptions about the world, not implement them
- Hence should prefer models described by constraints
- Recent advances in formal verification support this
    - Infinite bounded model checking, enabled by SMT solving
    - Allows use of uninterpreted functions
    - With axioms/constraints spec'd as synchronous observers
    - While still enjoying full automation

Reducing Epistemic Doubt: Completeness

- In addition to validity, we are concerned with the completeness of models
- E.g., have we recorded all hazards, all failure modes, etc.
- Traditional approaches: follow generally reliable procedure
  - E.g., ISO 14971 for hazard analysis in medical devices
  - Or HAZOP, FMEA, FTA etc.
- Most of these can be thought of as manual ways to do model checking (state exploration) with some heuristic focus that directs attention to the paths most likely to be informative
- With suitable models we can do automated model checking and cover the entire modeled space
  - e.g., infinite bounded model checking, again
  - `check: FORMULA (system || assumptions) |- G(AOK => safe)`
  - Counterexamples guide refinements to system design and/or assumptions

Aside: Formal Verification

Formal Analysis: The Basic Idea

- Symbolic evaluation. . .
- Instead of evaluating, say, $(5 - 3) \times (5 + 3)$ and observing that this equals $5^2 - 3^2$
- We evaluate $(x - y) \times (x + y)$
- And get some big symbolic expression
  $x \times x - y \times x + x \times y - y \times y$
- And we use automated deduction
  - The laws of (some) logic
  - And of various theories, e.g., arithmetic, arrays, datatypes

  To establish some properties of that expression
  - Like it always equals $x^2 - y^2$
- The symbolic evaluation can be over computational systems expressed as hardware, programs, specifications, etc.

Formal Analysis: Relation to Engineering Calculations

- This is just like the calculations regular engineers do to examine properties of their designs
  - Computational fluid dynamics
  - Finite element analysis
  - And so on

- In each case, build models of the artifacts of interest in some appropriate mathematical domain

- And do calculations over that domain

- Useful only when mechanized

Formal Analysis: The Difficulty

- For calculations about computational systems, the appropriate mathematical domain is logic
- Where every problem is at least NP Hard
- And many are exponential, superexponential (2^{2^n}), nonelementary ($2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}\}^n$), or undecidable
- Hence, the worst case computational complexity of formal analysis is extremely high
- So we need clever algorithms that are fast much of the time
    - Or human guidance (interactive theorem proving). . . ugh!
- But we also need to find ways to simplify the problems
    - e.g., abstraction, another kind of human guidance
- The need for (skilled) human guidance makes FM hard
- But new technologies (e.g., SMT solvers) improve things

Formal Analysis: The Benefit

- Can examine all possible cases
  - Relative to the simplifications we made
- Because finite formulas can represent infinite sets of states
  - e.g., $x < y$ represents $\{(0,1),\ (0,2),\ \ldots (1,2),\ (1,3)\ldots\}$
- Massive benefit: computational systems are (at least partially) discrete and hence discontinuous, so no justification for extrapolating from examined to unexamined cases
- In addition to providing strong assurance
- Also provides effective ways to find bugs, generate tests
- And to synthesize guaranteed designs

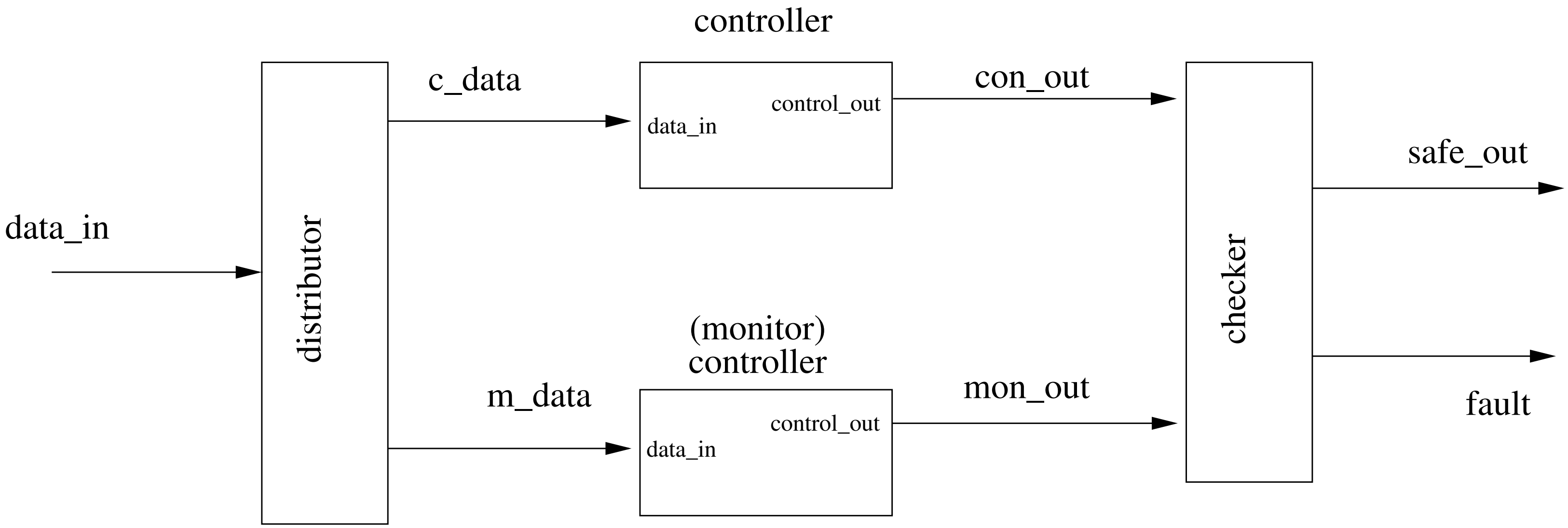
Completeness Example: Self-Checking Pair (1)

- If they are truly random, faults in separate components should be independent
    - Provided they are designed as fault containment units
        - ⋆ Independent power supplies, locations etc.
    - And ignoring high intensity radiated fields (HIRF)
        - ⋆ And other initiators of correlated faults

- So we can duplicate the component and compare the outputs
    - Pass on the output when both agree
    - Signal failure on disagreement

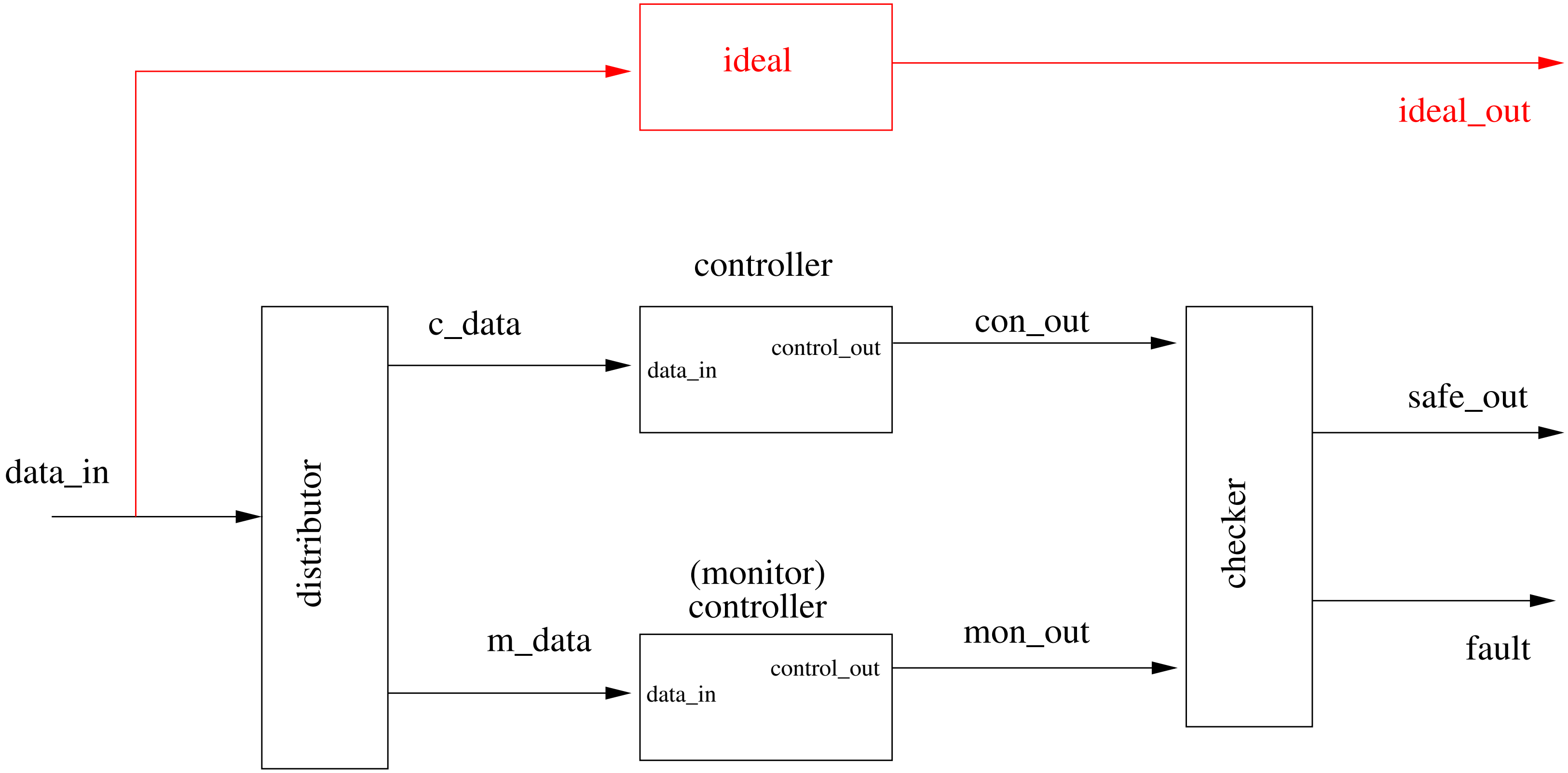
- Under what assumptions does this work?

Completeness Example: Self-Checking Pair (2)



- Controllers apply some control law to their input
- Controllers and distributor can fail
  - For simplicity, checker is assumed not to fail
  - Can be eliminated by having the controllers cross-compare
- Need some way to specify requirements and assumptions
- Aha! correctness requirement can be an idealized controller

Completeness Example: Self-Checking Pair (3)

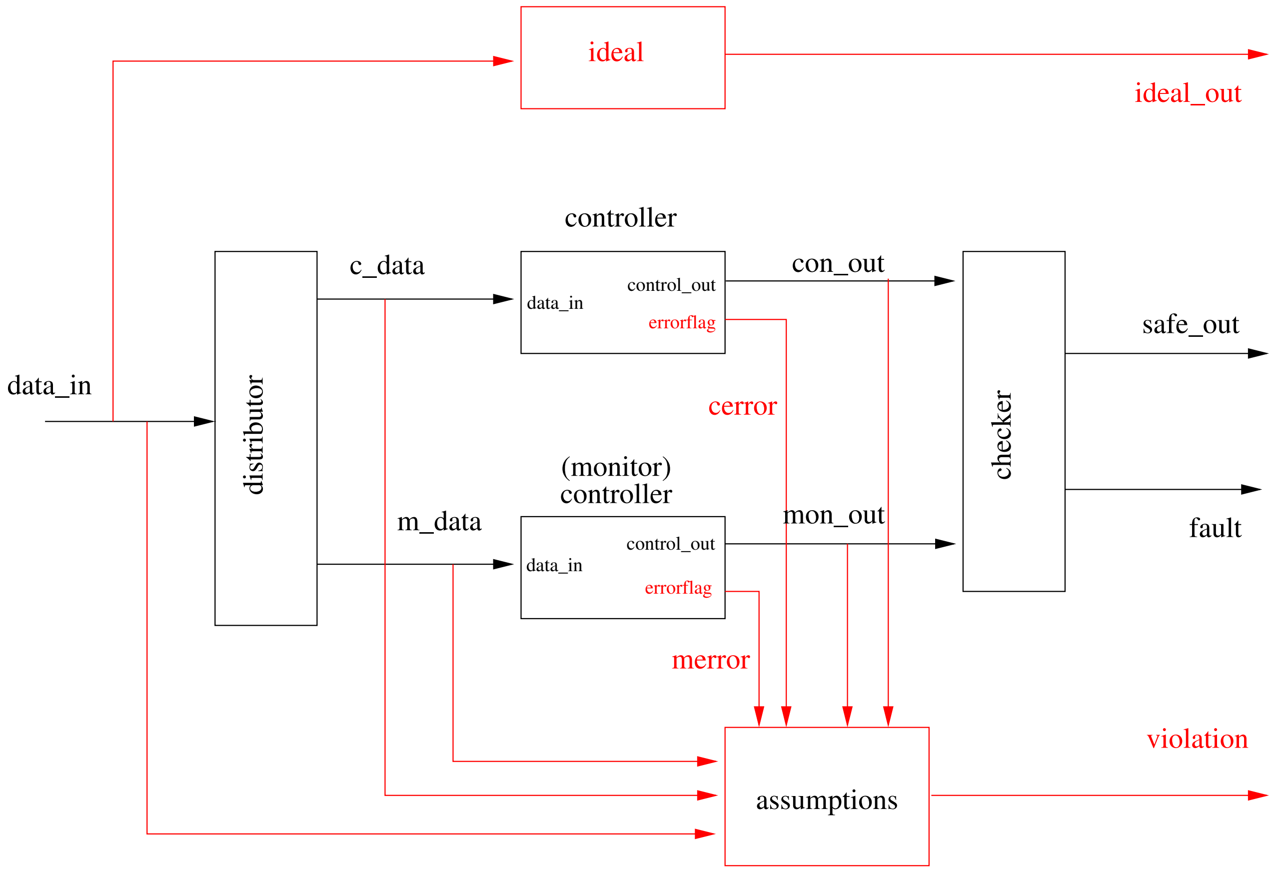


The controllers can fail, the ideal cannot

If no `fault` indicated `safe_out` and `ideal_out` should be the same

Model check for `G((NOT fault => safe_out = ideal_out))`

Completeness Example: Self-Checking Pair (4)



We need assumptions about the types of fault that can be tolerated: encode these in assumptions synchronous observer

`G(NOT violation => (NOT fault => safe_out = ideal_out))`

Completeness Example: Self-Checking Pair (5)

- Find four assumptions for the self-checking pair
  - When both members of pair are faulty, their outputs differ
  - When the members of the pair receive different inputs, their outputs should differ
    - When neither is faulty: can be eliminated
      - con gets 4 and mon 5 and it happens that $f(4) = f(5)$
      - Doubt you would find this with Simulink: you'd have to have something explicit for $f(x)$, say $x + 1$
    - When one or more is faulty
  - When both members of the pair receive the same input, it is the correct input
- Can prove by 1-induction that these are sufficient
- One assumption can be eliminated by redesign
- Two require double faults (hence, improbable)
- Attention is directed to the most significant case

Observations

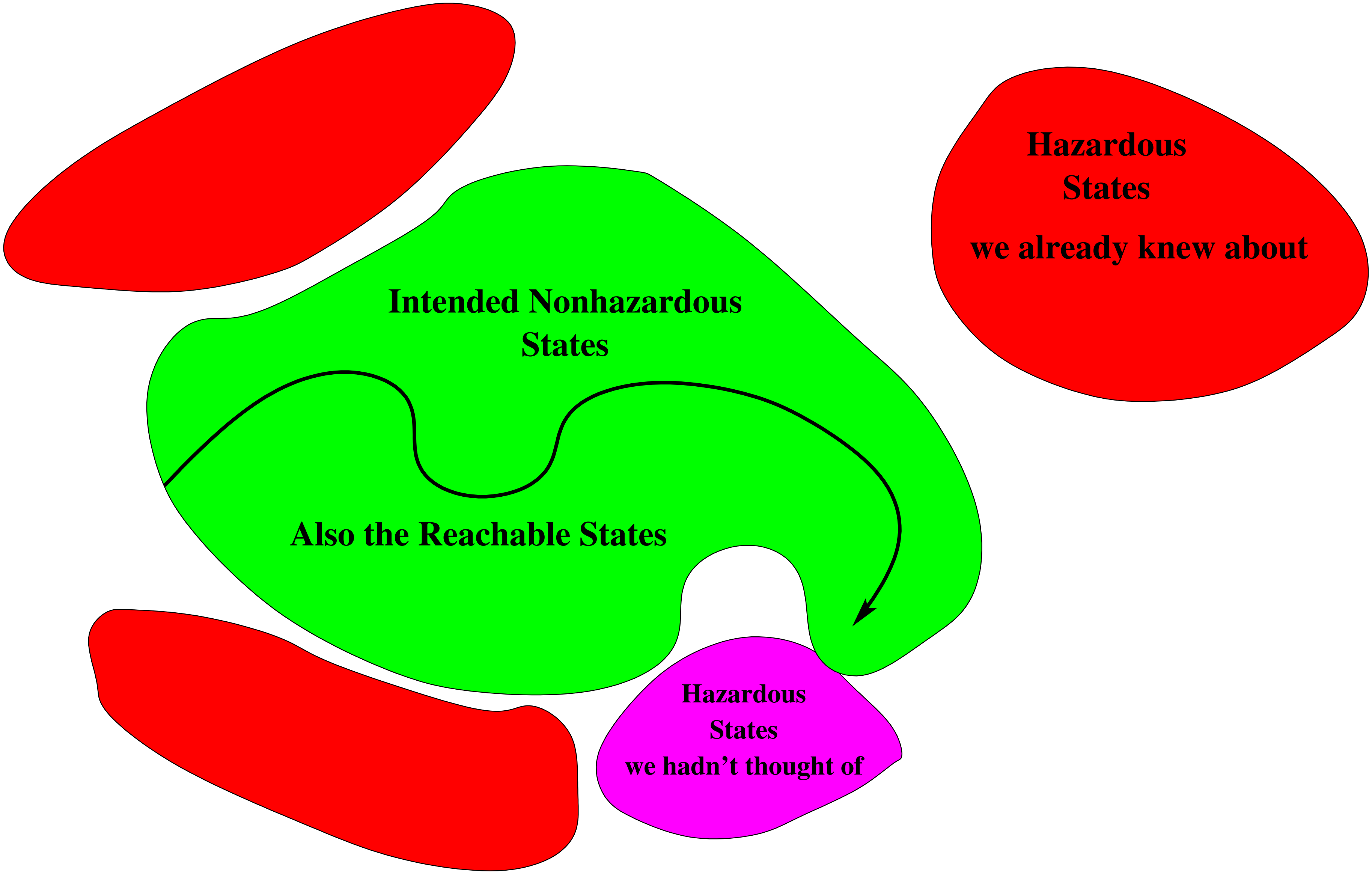
- Instead of constructing behavior as in a system specification
- A synchronous observer recognizes it
- And the model checker synthesizes the behavior for us
- May be costly with an explicit state model checker
  - Has to generate many behaviors, then throw them away

  But OK for symbolic ones

Reducing Epistemic Doubt: Completeness (ctd 2).

- Robin Bloomfield argues that we should not reduce assurance cases to an enumeration over hazards
    - Even though some authorities implicitly require this
    - i.e., show each hazard is unreachable
- We should also examine intended behavior
- Obviously, we do need to show that the thing works
- But I think this is also a way to discover new hazards
    - By exploring intended behaviors, we may be led to contemplate some nearby unintended behaviors
    - That had not been revealed by a pure search for the unintended
- cf. Derived Requirements in DO-178C

Nearby Unintended Behaviors, In Pictures



Compositional Assurance

- This is the Holy Grail
  - It's difficult for system properties due to the possibility of emergent misbehavior
  - The subject of tomorrow's talk
- Use a MILS- or IMA-like architecture
  - Partitioning constrains possible interaction paths
- Then (epistemic) assumptions of one component
  - Become requirements on its environment
  - i.e., on the components it interacts with
- Can discover weak(est) environment assumptions by formal analysis/machine learning

Tying It All Together

- Modern formal methods use quite complex, often ad-hoc toolchains
- And the assurance case sits above and around all the separate verifications
- We want to reduce logic and epistemic doubt in the case by additional analysis, some formal, some informal
- So we have complex workflows
- Need to be able to assemble components to provide these toolchains and workflows
  - Possibly working over different logics and theories
  - And need to keep track of changes
- That's why we have developed an Evidential Tool Bus (ETB)
  - Paper in VMCAI, January 2013

Monitoring

- Certification can be supported by runtime monitoring
  - Sanctioned in ARP 4754A
  - E.g., can construct a Level A component from a Level C operational part and a Level A monitor
- Monitors can be small and simple, have very credible formal verification or synthesis
  - Obtain high probability of perfection
- Perfection of the monitor is conditionally independent of failures of the operational channel
- Hence, reliability of system is product of probability of perfection of monitor and reliability of operational channel
- Details in IEEE TSE paper (with Bev Littlewood), Nov 2012
  - Complications if monitor activates when it should not
- Application to “Just In Time Certification” ICECCS Jul 2007, and “Runtime Certification” RV Apr 2008

Summary

- Exactly two kinds of doubt: logic and epistemic
- Can eliminate logic doubt by automated formal verification
- So should focus on reducing epistemic doubt
- Often best accomplished by minimizing epistemic assumptions
- Hence should prefer models described by constraints not simulation models
- Can use automated formal verification to explore these
- SMT solvers are an enabling technology