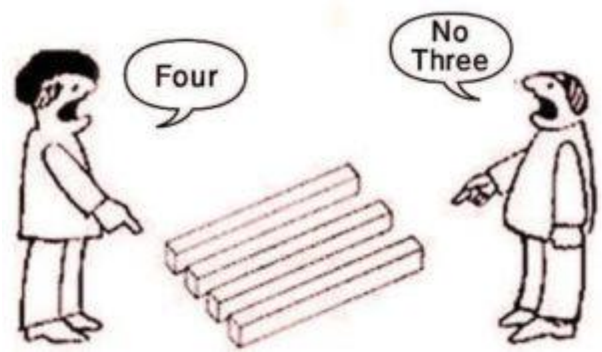
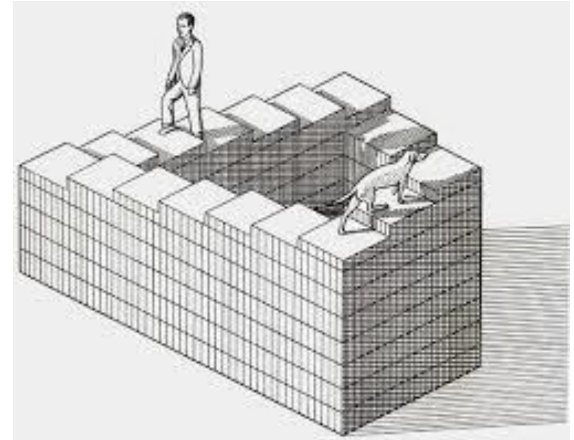




Safety Assurance (Safety Case): Is it Possible? Feasible?

Prof. Nancy G. Leveson

Aeronautics and Astronautics
MIT
(http://psas.scripts.mit.edu/...)

STAMP Virtual Workshop 2020

The Question

- Lots of talk about assuring safety (after system designed) and safety cases for certification and approval.
  - Goal is to argue that design will be safe (after detailed design complete)
  - May use various notations to assist in argument

- Systems today can be enormously complex
  - Examples usually done on trivially small systems or small parts of systems
  - Do the techniques scale up? Anything works on small problems.
  - Do notations such as “Goal Structure Notation” help?

- What is the alternative? Is it more practical?

My Argument

- Safety must be built into a system.
- If it is not already safe,
  - No amount of argument will make it be safe.
  - Only an incorrect argument will make it <u>appear</u> to be safe
  - If find not adequately safe, then what?
    - Start all over from scratch (impractical)
    - Add expensive retrofits that are costly and unlikely to be effective
    - Make an argument that design <u>*is*</u> safe and try to convince yourself and others that your incorrect argument is actually correct
    - Put incorrect argument in a graphic notation that is unreadable for realistic complex systems. Then rely on confirmation bias and inability to read and parse such a complex notation to make your argument
- After system completed, **HUGE** pressure for argument to be successful

Emphasis needs to be on designing system to be safe from the start.

An Alternative

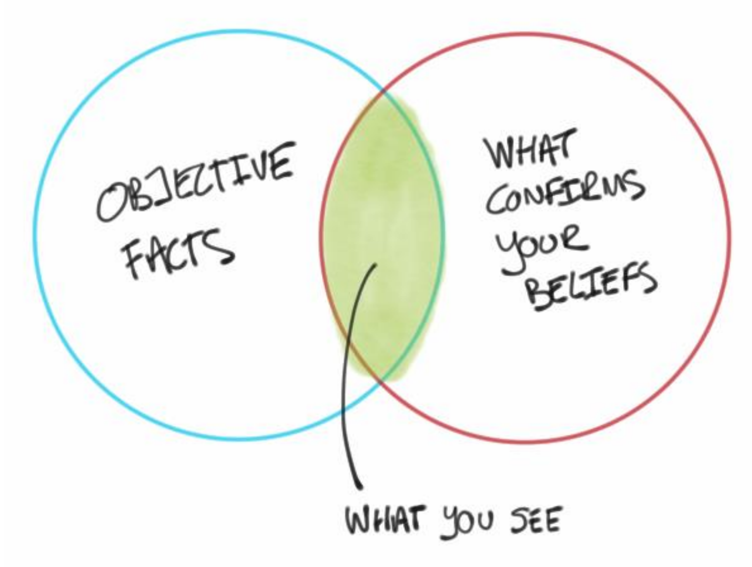
- "Proof" by construction
    1. Use analysis to identify system safety requirements and constraints during concept development
    2. Trace safety requirements and constraints to components before components designed
    3. Ensure safety requirements satisfied as design created. Usually involves more analysis to assist in making detailed design decisions.

- Assurance process is spread throughout development process.
    - Argument made while design is being created
    - At end, assurance is relatively trivial and requires only a review of documentation of what has been done during development
    - Almost always cheaper and more effective
    - Will work for even the most complex systems

Confirmation Bias

- Tendency to interpret evidence (or look for evidence) as confirmation of one's existing beliefs or theories.

"Still a man hears what he wants to hear and disregards the rest"
(*The Boxer* by Simon and Garfunkle)


OBJECTIVE FACTS
WHAT CONFIRMS YOUR BELIEFS
WHAT YOU SEE


AHH, NOT A BEAR IN SIGHT.
THE BEAR PATROL MUST BE
WORKING LIKE A CHARM.

Limits of Assurance (1)

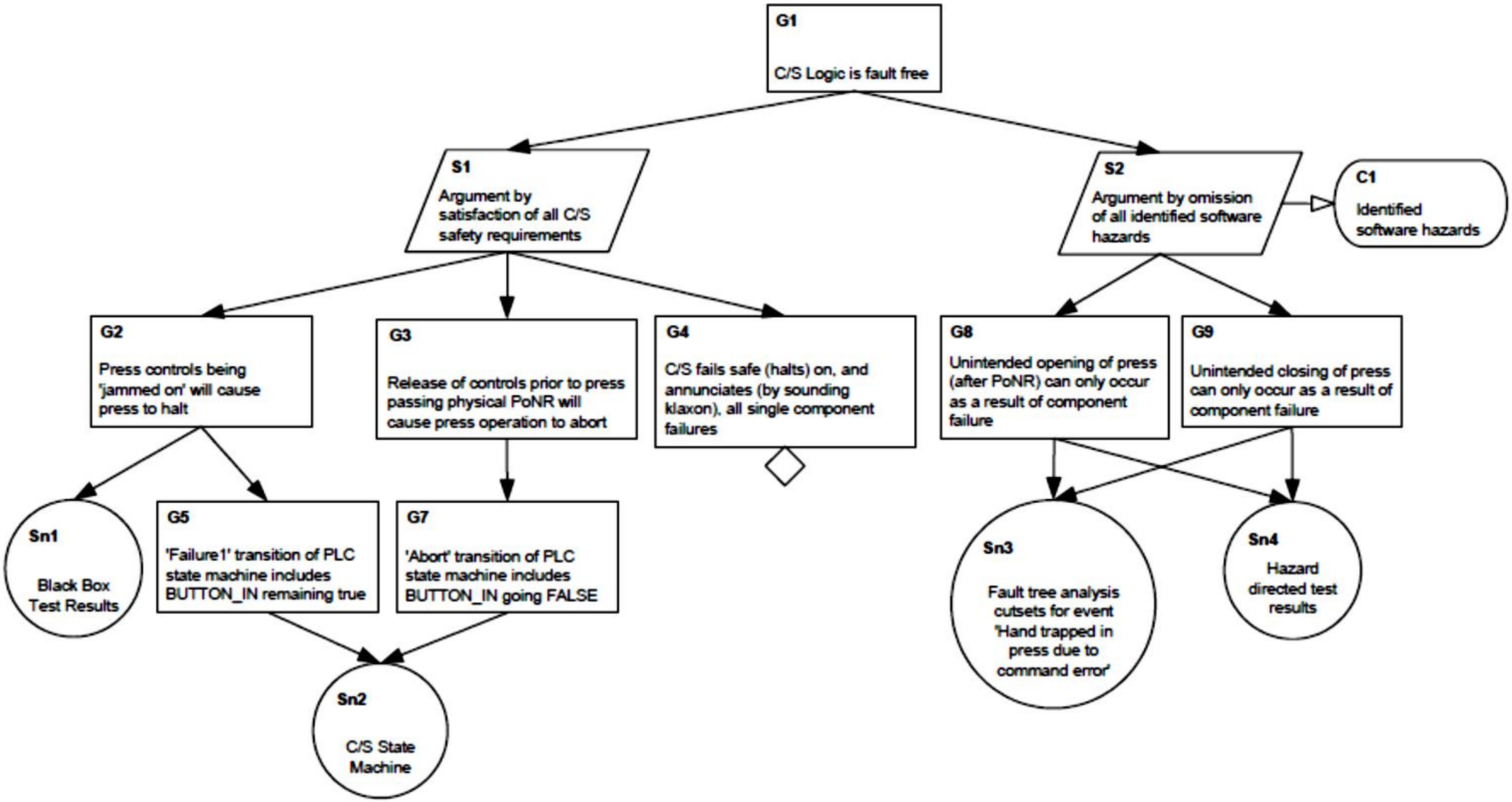
1. Testing and Simulation
   - Exhaustive testing not possible on complex systems today
   - Testing can only show presence of errors, not their absence
   - Safety problems almost always arise from design flaws and flawed assumptions (requirements)
     - Test and simulation based on same flawed assumptions
     - Cannot show requirements are correct or system has a desired property
     - Changes will occur in the future

2. Formal mathematical/logical arguments
   - OK when arguments based on physical principles (e.g., aerodynamics)
   - Not so good for software:
     - Shows consistency between requirements and code, but almost all accidents caused by requirements errors
     - Omits humans, not practical for systems today even if did work

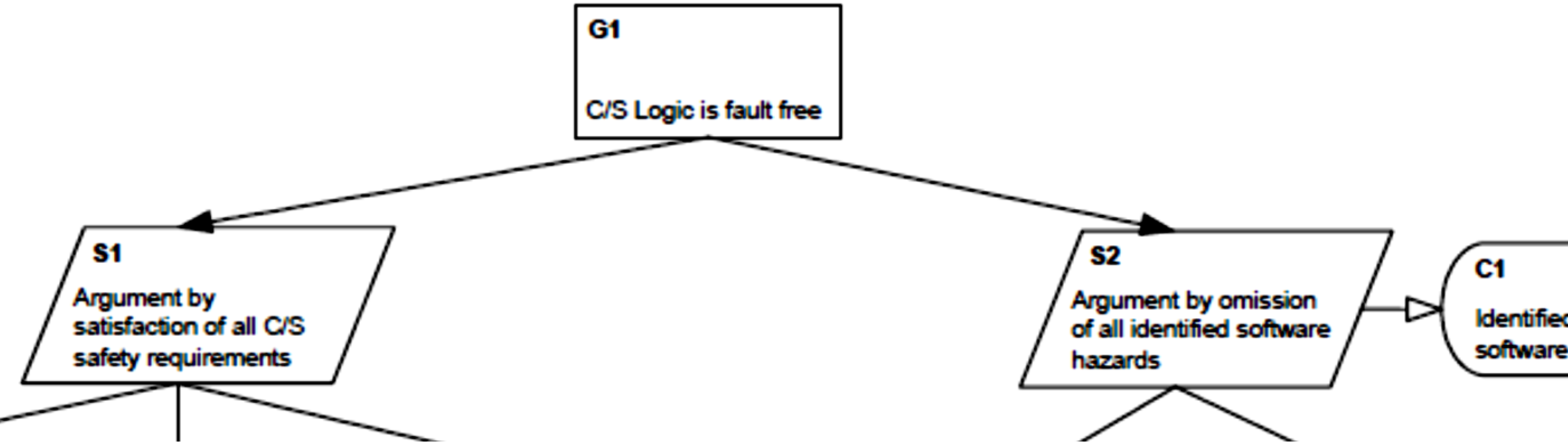
Limits of Assurance (2)

3. Informal arguments (such as Goal Structuring Notation– GSN)
   - Subject to confirmation bias
   - Questions phrased so that affirmative answer supports hypothesis
   - Omitted information or questions are ignored or not considered
   - Graphical formats do not eliminate bias and may make it worse
     - Assume there is “structure” to the argument and therefore must be correct
     - Everyone GSN argument I have seen has been logically incorrect
   - Arguments often based on omission: if hazard cause not identified, then assumed to not exist
     - Like testing only finding faults, cannot prove they do not exist
     - Easy to provide evidence to support any argument if only evidence considered is that which supports it.
   - Several examples from published literature are shown in paper

Example: GSN for a Press Hazard



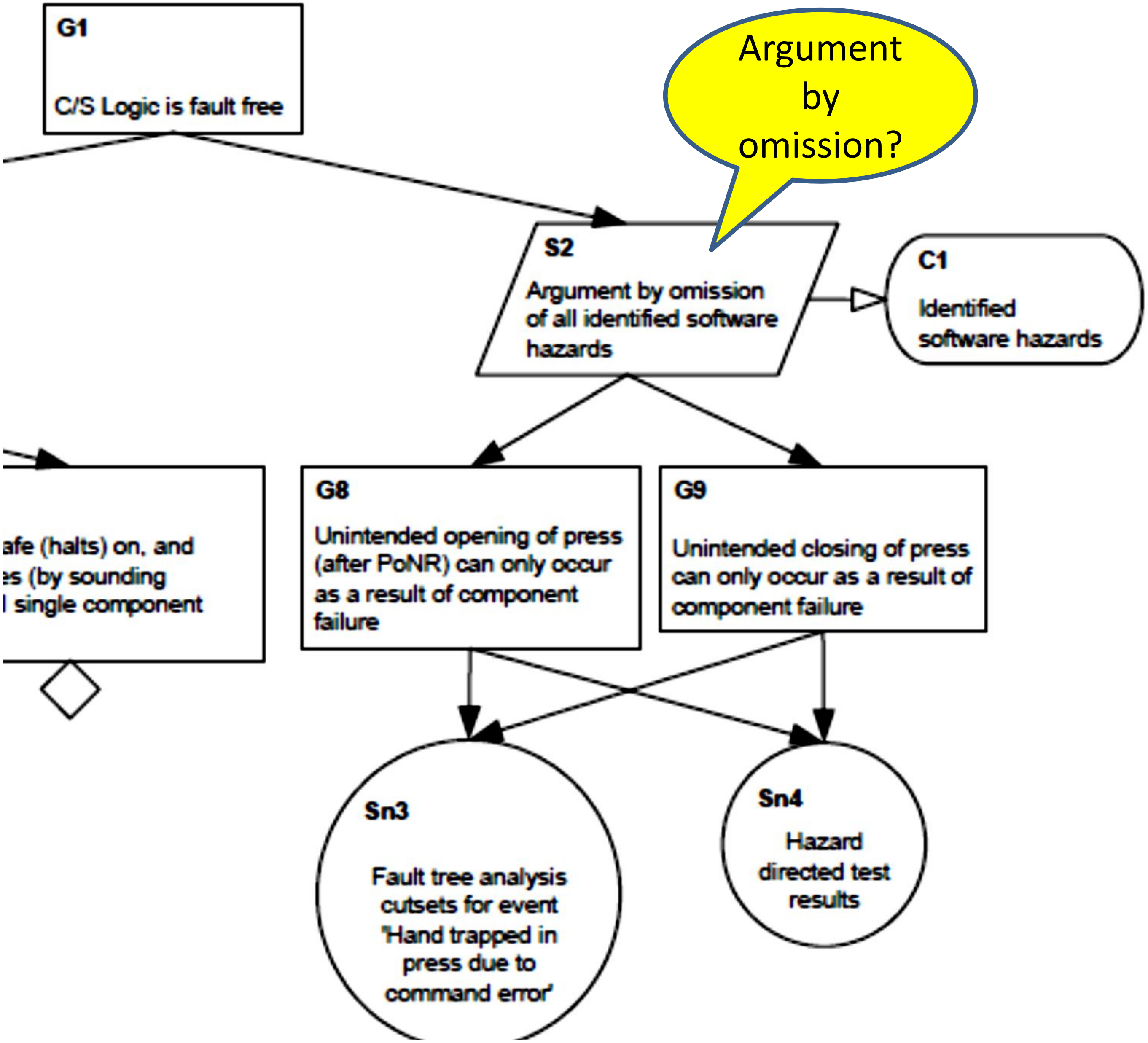
A highly cited and reused example



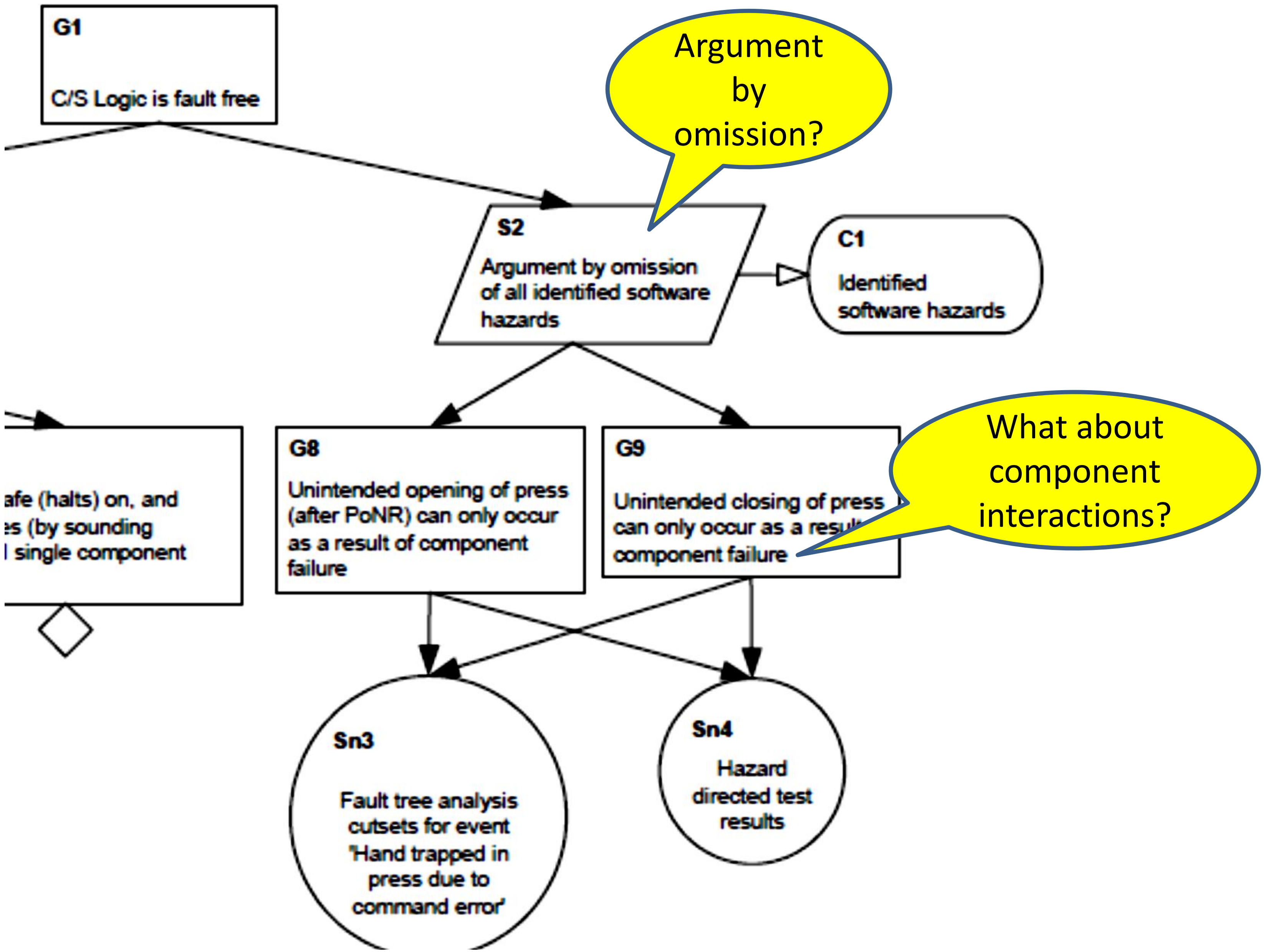
“C/S Logic is fault free”: What does that have to do with safety?

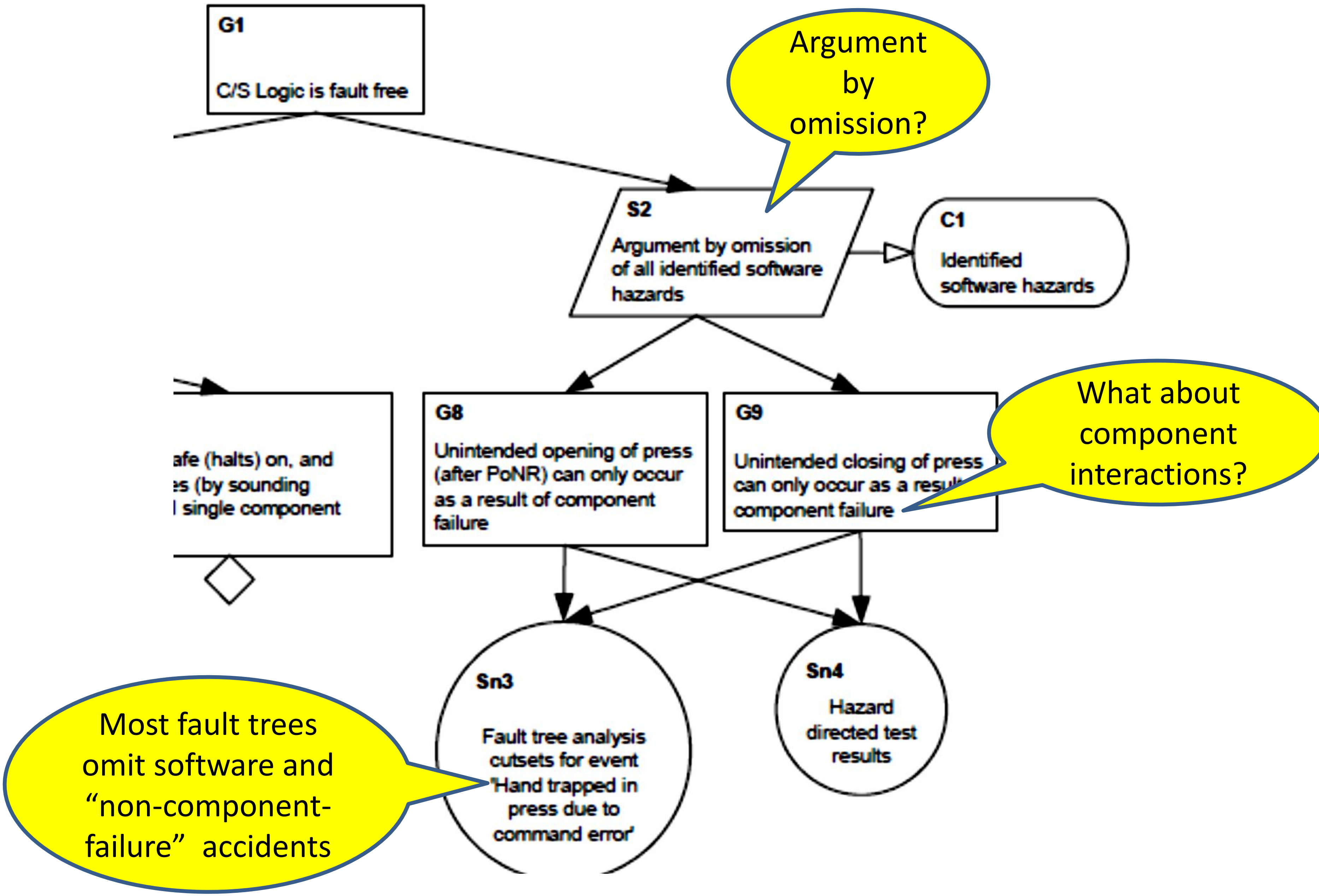
S1: Argument by satisfaction of all C/S safety requirements”:

- How do you know all have been identified?
- Problems below box left as an exercise for you

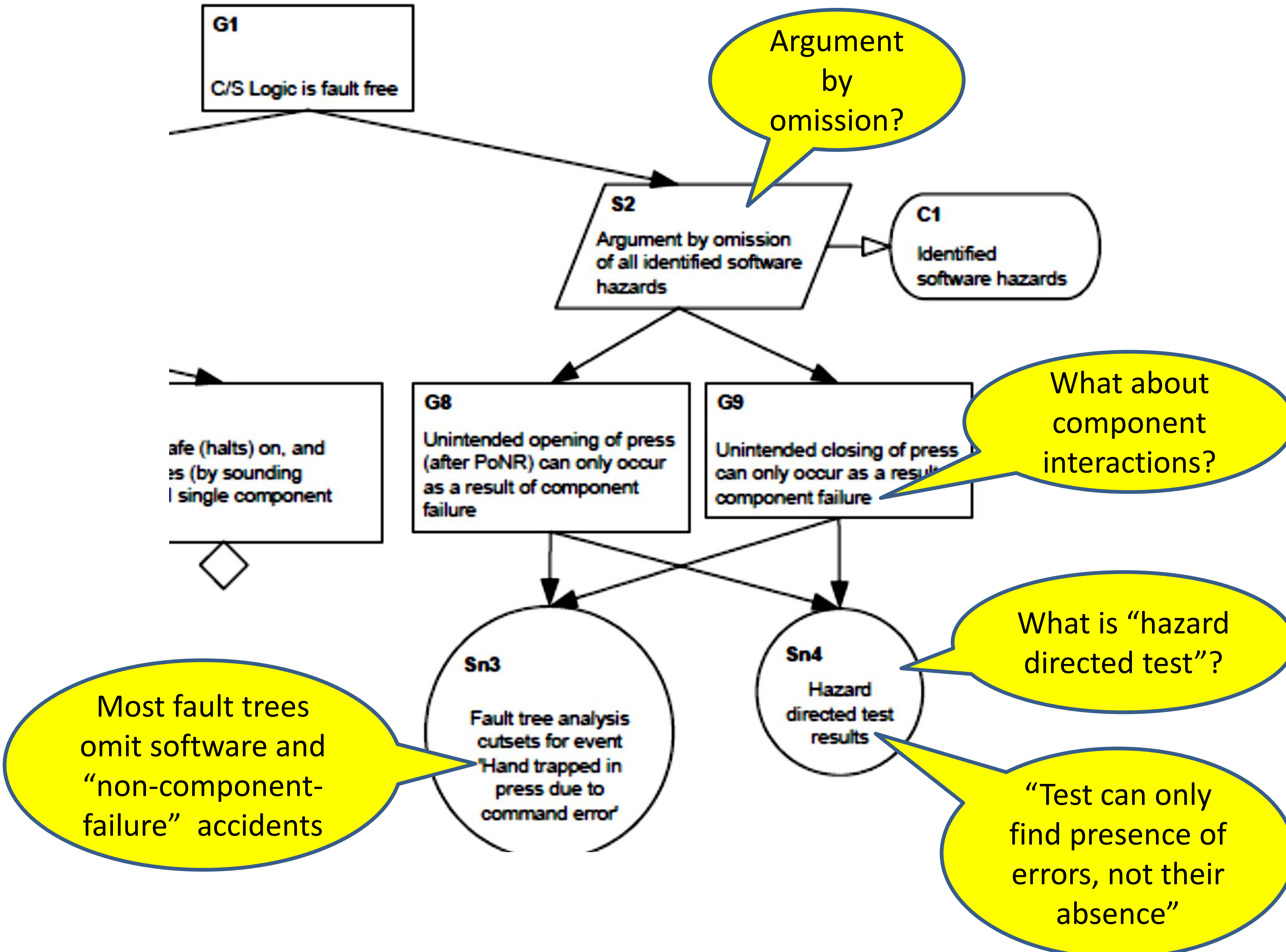


G1
C/S Logic is fault free
Argument by omission?
S2
Argument by omission of all identified software hazards
C1
Identified software hazards
afe (halts) on, and
es (by sounding
l single component
G8
Unintended opening of press (after PoNR) can only occur as a result of component failure
G9
Unintended closing of press can only occur as a result of component failure
Sn3
Fault tree analysis cutsets for event 'Hand trapped in press due to command error'
Sn4
Hazard directed test results


G1
C/S Logic is fault free
Argument by omission?
S2
Argument by omission of all identified software hazards
C1
Identified software hazards
afe (halts) on, and
es (by sounding
l single component
G8
Unintended opening of press (after PoNR) can only occur as a result of component failure
G9
Unintended closing of press can only occur as a resul
component failure
What about component interactions?
Sn3
Fault tree analysis cutsets for event 'Hand trapped in press due to command error'
Sn4
Hazard directed test results

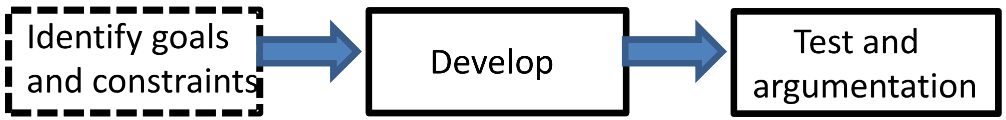


G1
C/S Logic is fault free
Argument by omission?
S2
Argument by omission of all identified software hazards
C1
Identified software hazards
afe (halts) on, and
es (by sounding
l single component
G8
Unintended opening of press (after PoNR) can only occur as a result of component failure
G9
Unintended closing of press can only occur as a resul
component failure
What about component interactions?
Sn3
Fault tree analysis cutsets for event 'Hand trapped in press due to command error'
Sn4
Hazard directed test results
Most fault trees omit software and "non-component-failure" accidents

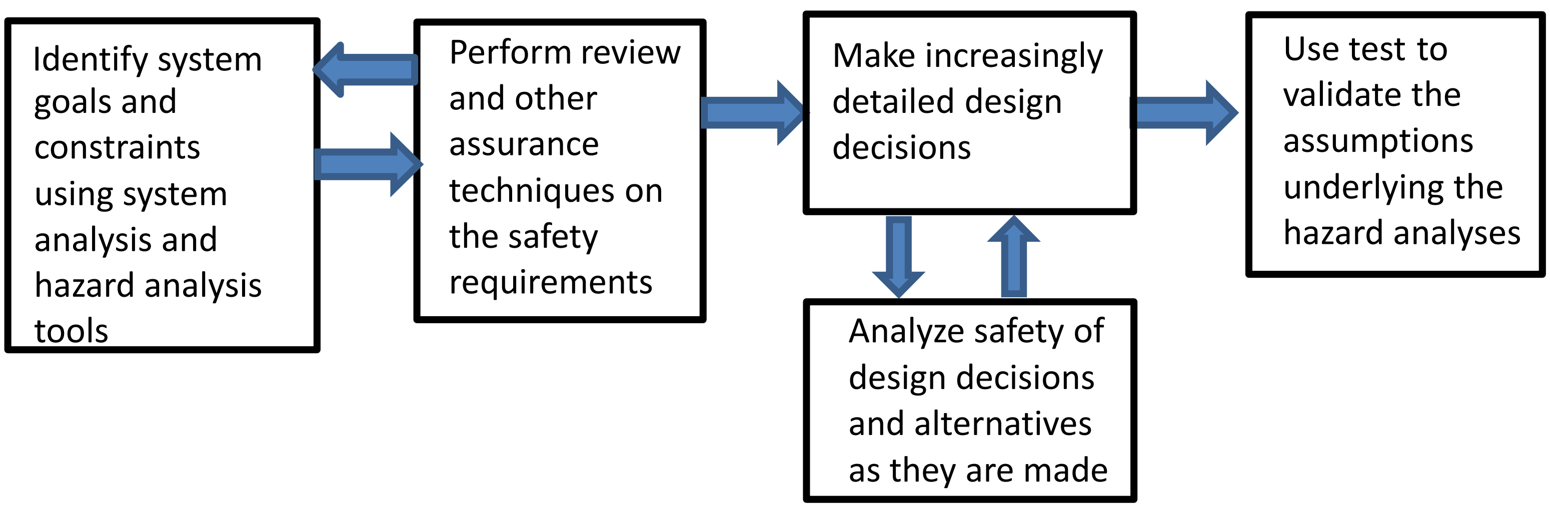

G1
C/S Logic is fault free
Argument by omission?
S2
Argument by omission of all identified software hazards
C1
Identified software hazards
afe (halts) on, and
es (by sounding
l single component
G8
Unintended opening of press (after PoNR) can only occur as a result of component failure
G9
Unintended closing of press can only occur as a resul
component failure
What about component interactions?
Sn3
Fault tree analysis cutsets for event 'Hand trapped in press due to command error'
Sn4
Hazard directed test results
What is "hazard directed test"?
Most fault trees omit software and "non-component-failure" accidents
"Test can only find presence of errors, not their absence"

The Alternatives

1. Rely on after-the-fact assurance



2. Build safety in as you design the system



Conclusions

- Need to build safety in, cannot assure it after the fact
- If find errors in complex system in your assurance process, then impractical to fix in any truly effective way
- Instead **construct the system to be safe**, using analysis along the way to identify how to do that then and assure as you build
- Note: the conclusion should not be about one particular notation. Instead:
  - If you are using hazard analysis to prove your system is safe, then you are using it wrong and your goal is futile
  - Hazard analysis (using any method) can only help you find problems, it cannot prove that no problems exist
  - The general problem is in setting the right psychological goal. It should not be “confirmation,” but exploration