

Formal assurance cases
as programs
for machine-checking
and checklisting
(meta-verification?)

for Panel discussion in ASSURE 2013

Makoto Takeyama

Kanagawa University

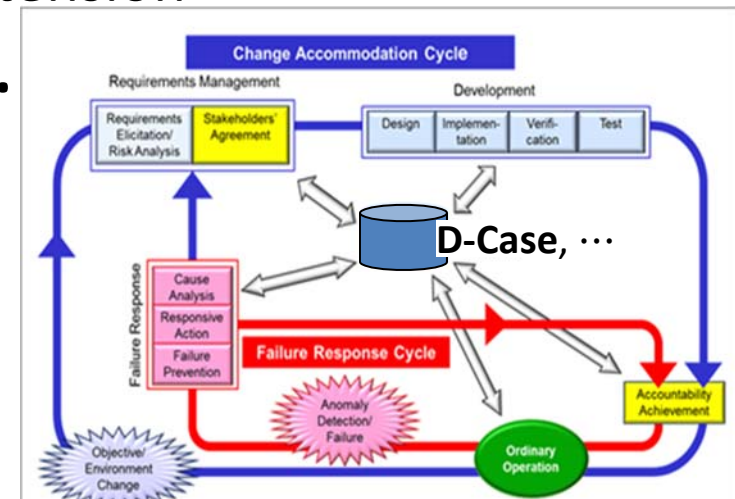
(Research funded by JST CREST DEOS project)

(DEOS project, D-Case)

- DEOS project: developing a lifecycle to achieve “Open Systems Dependability”
 - ~ “Service continuity under uncertainty and incompleteness” via
 - Consensus building (stakeholders’ Agreement)
 - Accountability achievement
 - Failure response
 - Change accommodation

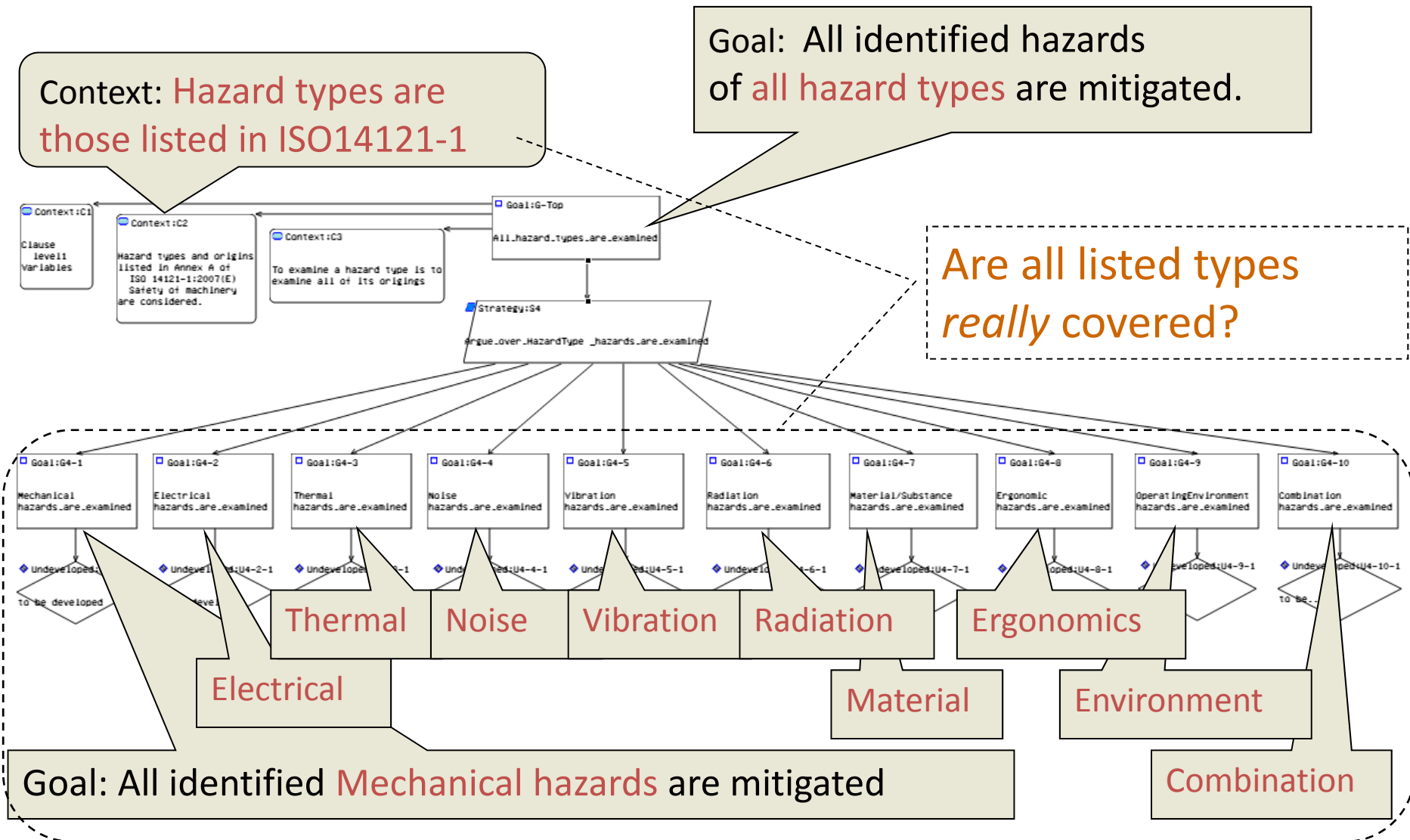
- **D-Case:** Assurance Case with DEOS extension

- **Control-doc/data for the DEOS proc.**
- **Not just for humans; Electronically Integrated with DEOS Runtime Env. (monitoring, action scripting, ...)**

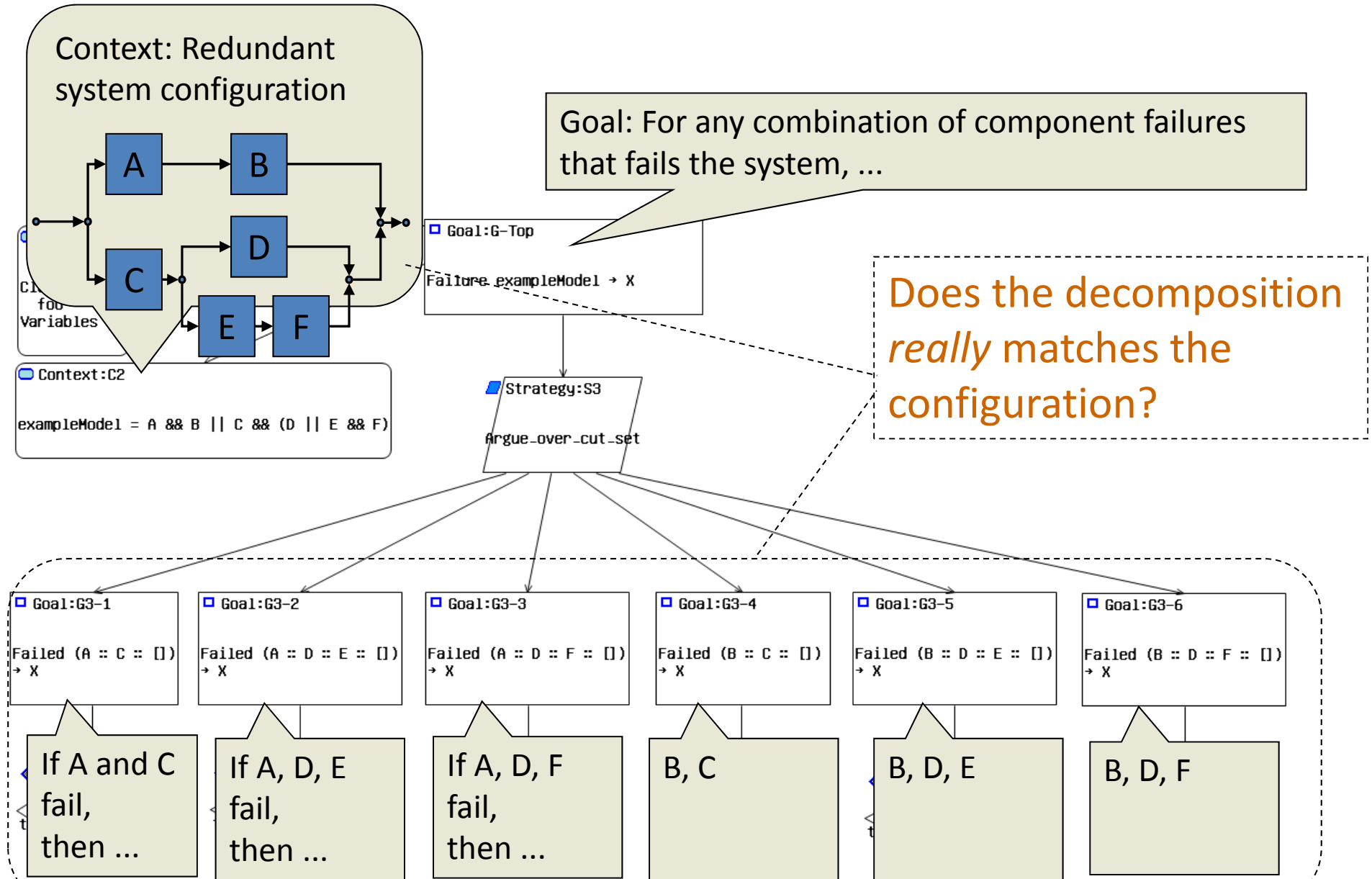


(Tokoro. DEOS project Whitepaper, 2011)

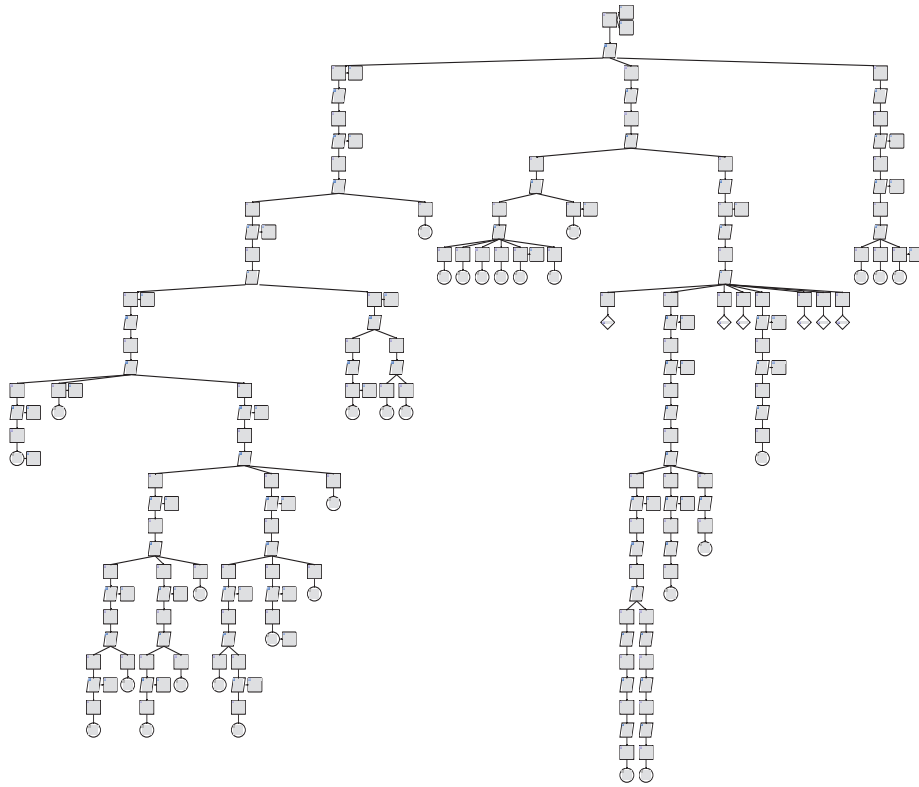
A very mundane checking



a not-so-mundane but mechanical checking



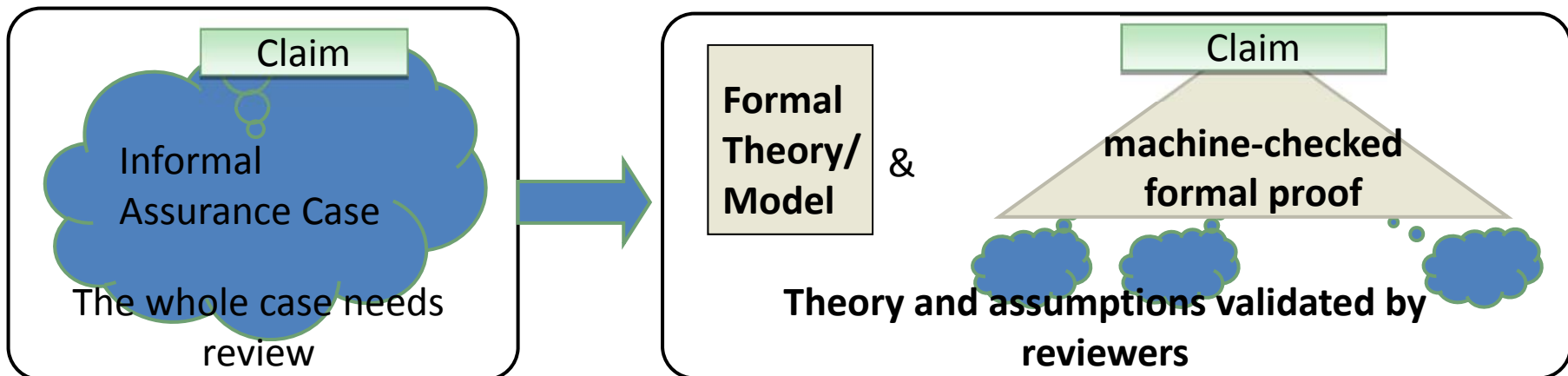
Problem



- Reviewer must do mundane checking by reading informal contents of argument elements (beside exercising judgment).
 - Points to check are interspersed across 100s of these, each of which is frequently updated.
-
- **Let machines check what they can check**
to let reviewers concentrate on exercising expert judgment.

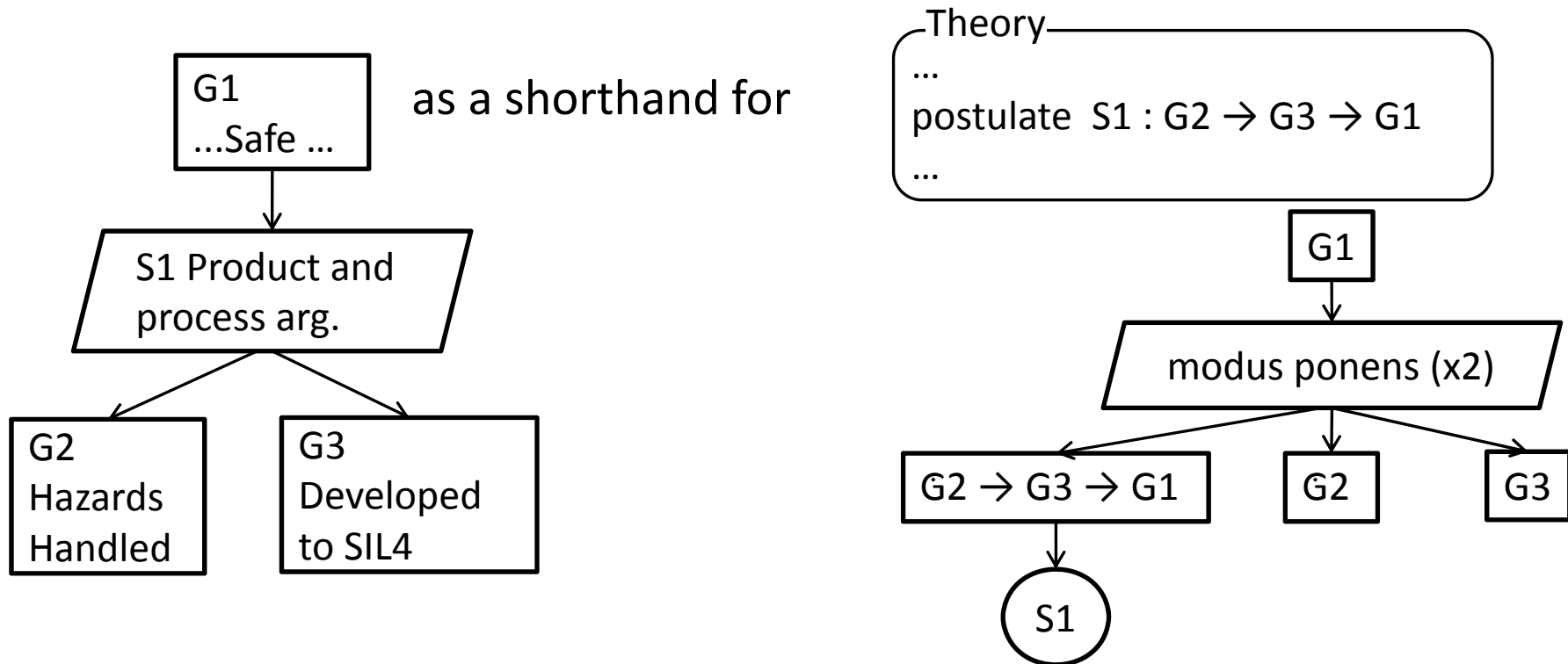
Formal AC as $\langle Theory, Proof \rangle$

- Checking can only be w.r.t. the basis of argument.
Vocabulary / ontology of concepts and things;
models of system/environment;
reasoning principles both inductive and deductive, ...
- Let each **AC be** \langle decl./def. of the basis , arg. on that basis \rangle
formulated as \langle formal theory (model) , formal proof in it \rangle
- All mundane checks are turned into
“Is this a formal proof in this theory?”
- Expert reviewer exercises judgment on
“Is this theorising valid in Real World ?”



“Arguments are not proofs” ?

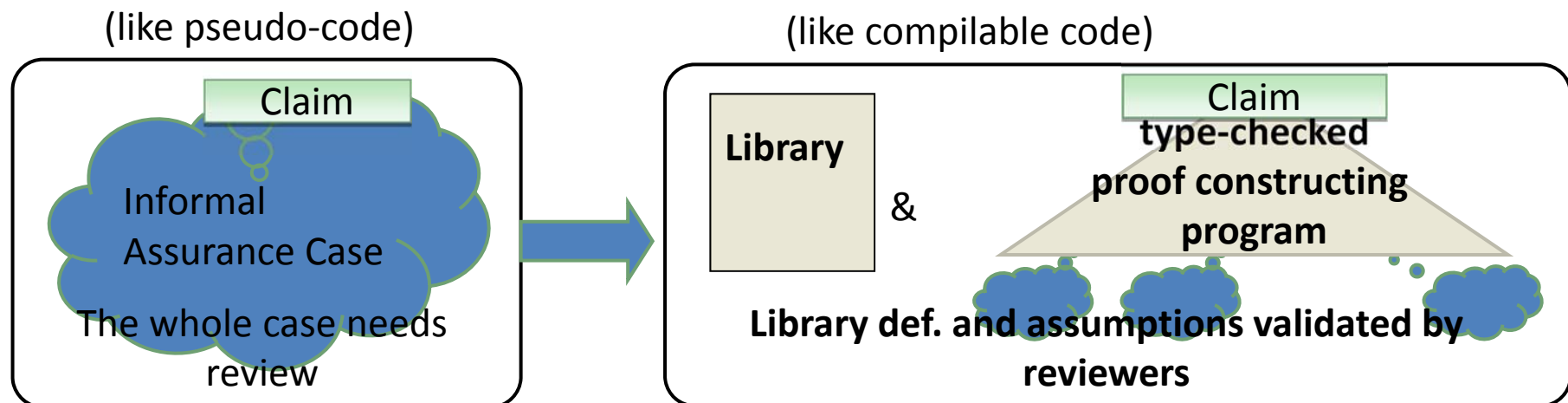
- They should be, *in an explicitly declared theory.*



- Declare inductive and other extra-logical aspects explicitly in the theory.
 - arguments need not be “from the first principles” to be a formal proof.
- Separate
 - declarations and definitions to be agreed on / validated and
 - their usage in arguments to be checked / verified.

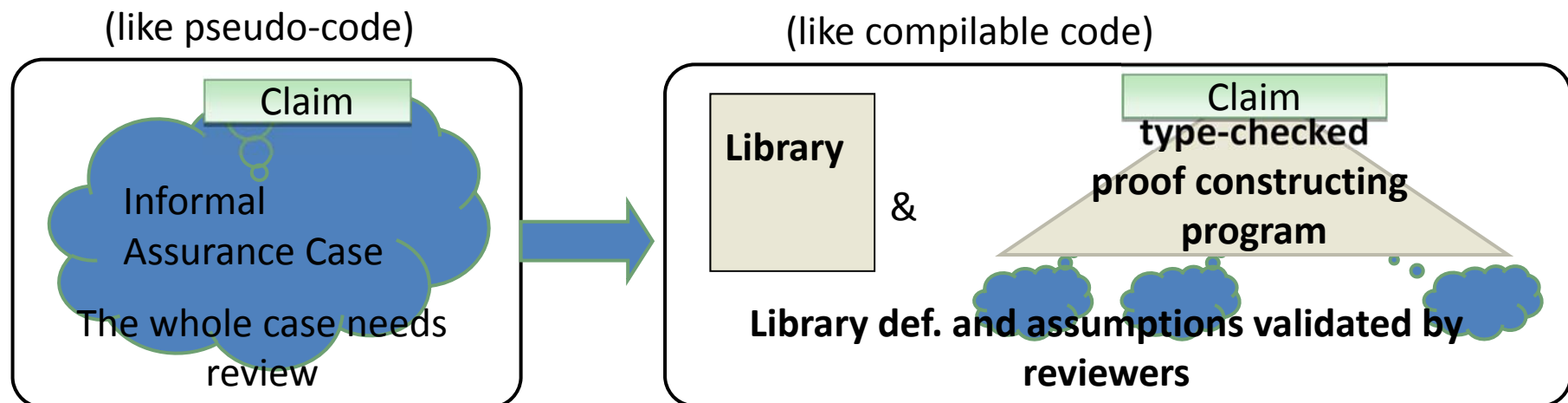
Formal AC as Programs

- Conventional formulation of formal proofs lacks mechanism for large-scale development (no definitional mech., no structuring, ...)
- Write an AC as \langle library defn., program \rangle using a prog. lang. supporting “Proofs as Programs” with rigorous semantics, e.g., Agda.
- Checking an argument = type-checking a program
Checking 100 connected args = doing a build on a project
- For constructing understandable, maintainable, large arguments, all the programming / sw-eng. techniques can be imported wholesale. (*abstraction, recursive defn, modularisation, change-management, ...*)



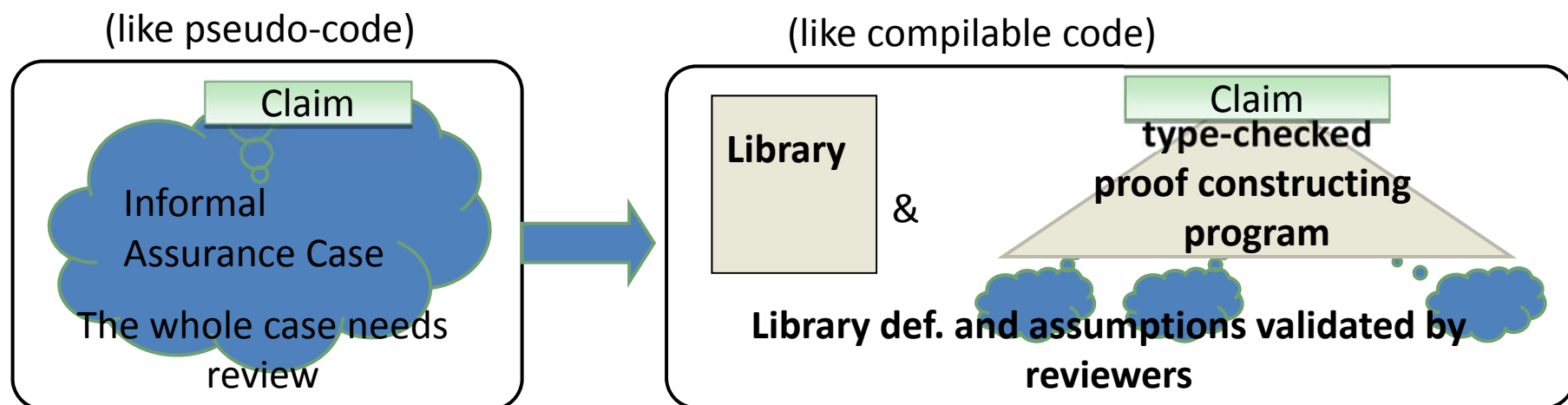
Formal AC as Programs

- Proposition = Type of data that counts as direct evidence
- Proof = Program that produces the direct-evidence data when run
- Write an AC as \langle library defn., program \rangle using a prog. lang. supporting “Proofs as Programs” with rigorous semantics, e.g., Agda.
- Checking an argument = type-checking a program
Checking 100 connected args = doing a build on a project
- For constructing understandable, maintainable, large arguments, all the programming / sw-eng. techniques can be imported wholesale. (*abstraction, recursive defn, modularisation, change-management, ...*)

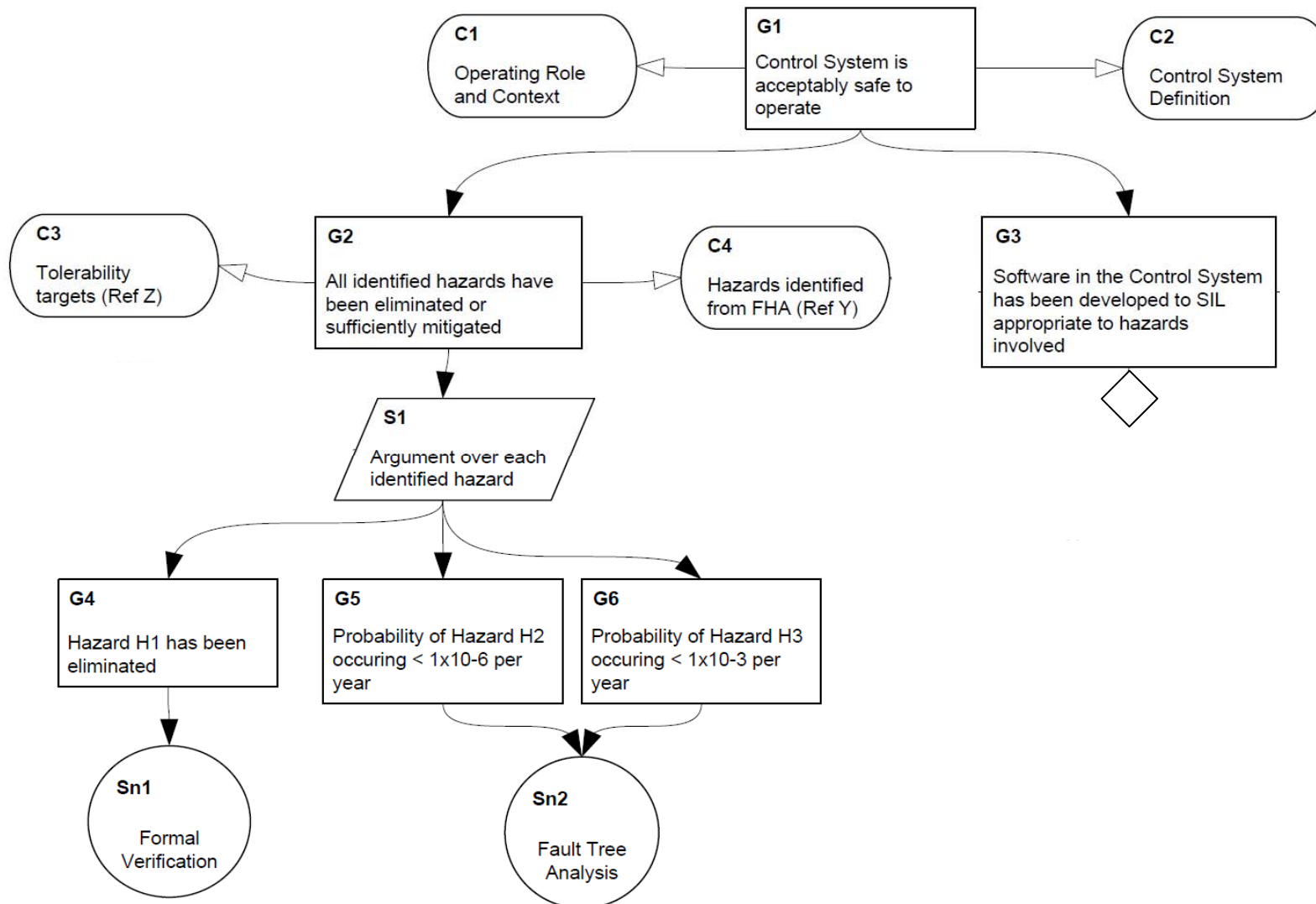


Formal AC as Programs

- Proposition = Type of data that counts as direct evidence
- Proof = Program that produces the direct-evidence data when run
- Write an AC as \langle library defn., program \rangle using a prog. lang.
 - **Issues ALL SORTED OUT:** free-/bound-variables, scoping, safe-looping,...
 - Type checked patterns / templates guarantee that any legal instantiation results in “correct” argument.
- For constructing understandable, maintainable, large arguments, all the programming / sw-eng. techniques can be imported wholesale. (*abstraction, recursive defn, modularisation, change-management, ...*)



(From GSN Standard Figure 6, simplified)



(Fig 6 in arguments-as-programs form)

```
(-# OPTIONS --allow-unsolved-metas #-)
module ExampleAssuranceCase where
open import Data.Empty
open import Data.Nat
open import Data.Sum
open import Data.Unit
open import PoorMansControlledEnglish

-----
-- Theory part
-----
module Theory where
data Probability.Type : Set where
  1x10-3_per_year : N → Probability.Type
impossible : Probability.Type
postulate
  _<_ : Probability.Type → Probability.Type → Set
infix 1 _<_

module C2-Control_System_Definition where
postulate
  Control_System.Type : Set
  Control_System : Control_System.Type

module C4-Hazards_Identified_from_FHA where
data Identified_Hazards : Set where
  H1 H2 H3 : Identified_Hazards
postulate
  Probability_of_Hazard : Identified_Hazards → Probability.Type

module C3-Tolerability_Targets where
open C4-Hazards_Identified_from_FHA
mitigation_target : Identified_Hazards → Probability.Type
mitigation_target H1 = impossible
mitigation_target H2 = 1x10-3_per_year
mitigation_target H3 = 1x10-6_per_year
Sufficiently_mitigated : Identified_Hazards → Set
Sufficiently_mitigated h =
  Probability_of_Hazard h < mitigation_target_of h
postulate
  Eliminated : Identified_Hazards → Set
argument_over_each_identified_hazard :
  H1 is Eliminated →
  H2 is Sufficiently_mitigated →
  H3 is Sufficiently_mitigated →
  ∀ h → h is Eliminated or Sufficiently_mitigated
argument_over_each_identified_hazard p1 p2 p3 H1 = inj₁ p1
argument_over_each_identified_hazard p1 p2 p3 H2 = inj₂ p2
argument_over_each_identified_hazard p1 p2 p3 H3 = inj₃ p3

module C1-Operating_Role_and_Context where
open C2-Control_System_Definition
open C3-Tolerability_Targets
open C4-Hazards_Identified_from_FHA
postulate
  Software_has_been_developed_to_appropriate_SIL : Set
  Acceptably_safe_to_operate : Control_System.Type → Set
argument_over_product_and_process_aspects :
  (For-all h of Identified_Hazards
  → h is Eliminated or Sufficiently_mitigated) →
  Software_has_been_developed_to_appropriate_SIL →
  Control_System is Acceptably_safe_to_operate

-----
-- References to evidence
-----
module Evidence where
open Theory
open C4-Hazards_Identified_from_FHA
open C3-Tolerability_Targets
postulate
  Formal_Verification : H1 is Eliminated
  Fault_Tree_Analysis_H2 : Probability_of_Hazard H2 < 1x10-3_per_year
  Fault_Tree_Analysis_H3 : Probability_of_Hazard H3 < 1x10-6_per_year

-----
-- Reasoning part
-----
module Reasoning where
open Theory
open Evidence

main =
  let open C1-Operating_Role_and_Context
      open C2-Control_System_Definition
      in
      Control_System is Acceptably_safe_to_operate
      by argument_over_product_and_process_aspects
      • (let open C3-Tolerability_Targets
          • (let open C4-Hazards_Identified_from_FHA
              in
              (For-all h of Identified_Hazards
              → h is Eliminated or Sufficiently_mitigated)
              by argument_over_each_identified_hazard
              • (H1 is Eliminated)
                by Formal_Verification)
              • (Probability_of_Hazard H2 < 1x10-3_per_year)
                by Fault_Tree_Analysis_H2)
              • (Probability_of_Hazard H3 < 1x10-6_per_year)
                by Fault_Tree_Analysis_H3)
          • (Software_has_been_developed_to_appropriate_SIL
            by undeveloped (1) / "undeveloped" ) )
      end
  end

(-# DCASE main root #-)
```

“Theory” part:

- Declares and defines the basis of the argument:
 - Primitive terms for primitive things and concepts,
 - Defined terms for defined things and concepts,
 - Presumptive relationship among legal terms.
- Gives definite meaning to any legal combination of terms.
- Must be agreed / approved through supporting process.
- Organized into modules corresponding to contexts.

“Evidence” part

- Declares presumptive existence of evidence to some claim-terms
- Must be agreed / approved through supporting process

“Reasoning” part

- Exhibits a combined term as a proof for the top claim-term
 - Whether it is a legal proof or not is machine-checkable.
- The top claim-term must be agreed / approved.

D-Case/Agda (“D-Case in Agda” Verification Tool)

- Provides translation between arg. in graphical form and in Agda program form.
- AC as an Agda program is checked in Agda dev. environment, which also is a proof-assistant for constructing args as programs.
- Google “D-Case/Agda” for download. [dcase-en](#) [dcase-agda](#) [code-agda](#)

Graphical edit, domain-expert review using D-Case Editor

Checking, construction, generation using Agda proof assistant

switchable

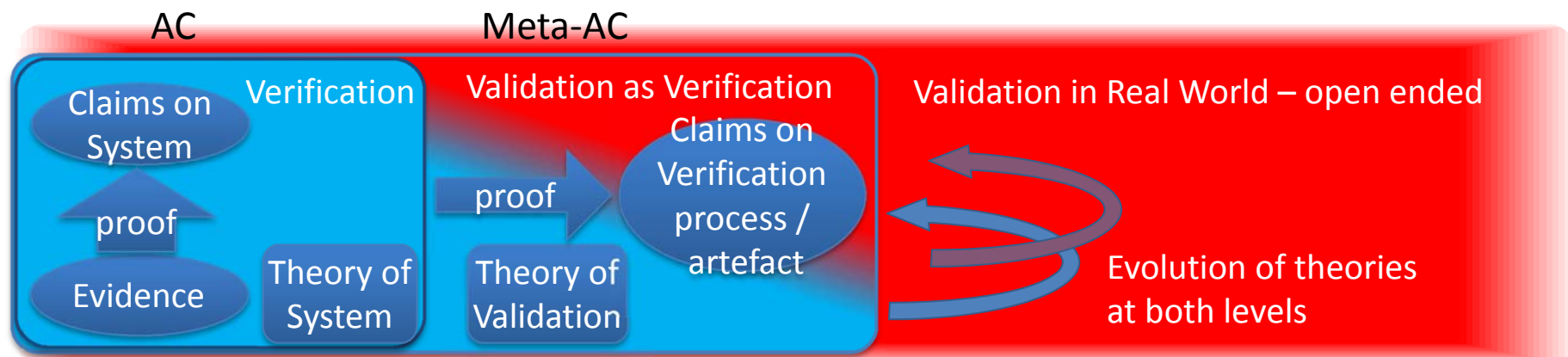
```
BasicStage.agda
File Edit Options Buffers Tools Agda Help
« DemoGoal "35 sec"
 / "DemoLineTracker-Robot clears the DemoCourse within 35
Context[ "identified risks:\n\ \ Start command not recei
 / { _$ / "Risk mitigation argument" }
 . ( (« (R.AllMitigated → R.Objective) / "Risk identifi
   ⊃ { Risk-Analysis-Report / "Risk Analysis Report" })
 . ( (« ((x : Identified-Risk) → Mitigated x) / "Each id
   ⊃ { R.riskCase Mitigated / "Argue over identified ri
 . ( (« Mitigated Cmd-not-received / "Communication f
   ⊃ { sub-d-case Cmd-not-received / "Sub D-Case-3
 . ( (« Mitigated Tracking-too-slow / "Tracking precis
   { sub-d-case Tracking-too-slow / "Sub D-Case-
 . ( (« Mitigated Losing-line / "Losing the line acti
   ⊃ { sub-d-case Losing-line / "Sub D-Case-2" })
 . ( (« Mitigated Collision-with-other-robots / "Dist
   ⊃ { sub-d-case Collision-with-other-robots / "S
 . ( (« Mitigated Course-lighting-interfering-with-li
   ⊃ { sub-d-case Course-lighting-interfering-with
 ]
-U\**- BasicStage.agda 56% L117 (Agda:Checked)--<V> ---
Auto-saving...done
```

Checklisting and meta-verification (speculative)

- Proving is primarily about
 - Verification : “Are we building the system right?”
(w.r.t. given, specified criteria : sys. spec, operational conditions, ...)
 - and not about
 - Validation : “Are we building the right system?”
(w.r.t. “Real World” : user needs, actual environment, ...)
 - No definitions for “right claims to make”, “right structure to argue”, ...
- But checklists / requirements for AC in guides and standards are there to help.
- Template libraries can be prepared to enforce required forms and contents of AC.
(But no explanation of why it’s good. Little room for adaptation.)

Checklisting and meta-verification (speculative)

- Part of validation can be verification about system-verif.:
Meta-verification: “Are we verifying the system right?”,
(w.r.t. checklists / requirements for AC in guides and standards)
- **Formal AC as data with rigorous semantics can be a target for such verification.**
- The rationale for checklists for AC could be codified in a “theory of validation” about properties of AC, and Meta-AC about AC could argue for the ‘goodness’ of AC.
- AC ~ Traceability matrix on steroids (R. Chapman, ‘08)
Meta-AC ~ Checklists on steroids



Recommendations?

- Explicitly declare / define the basis of the argument.
AC is not just an argument.
- Make the argument machine-checkable w.r.t. that basis
via “arguments as proofs.”
- Apply programming and software engineering techniques
for constructing and maintaining large AC,
directly via “arguments as programs.”
- Develop libraries of verified patterns / frameworks
 - to ease construction of formal AC,
 - to enforce recommended forms and contents of AC.
- Formulate the rationale behind requirements for AC in
standards as a theory of meta-verification about verification