Formal Assurance Case in Agda（FACIA)

形式アシュランスケースのAgda言語による記述

2014-03-18

第5回D-Case研究会 @ 国立情報学研究所

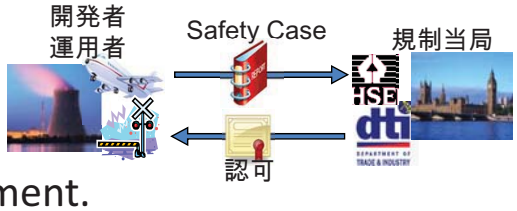
神奈川大学 武山誠

(research funded by JST CREST DEOS project)

アシュランスケース (AC)

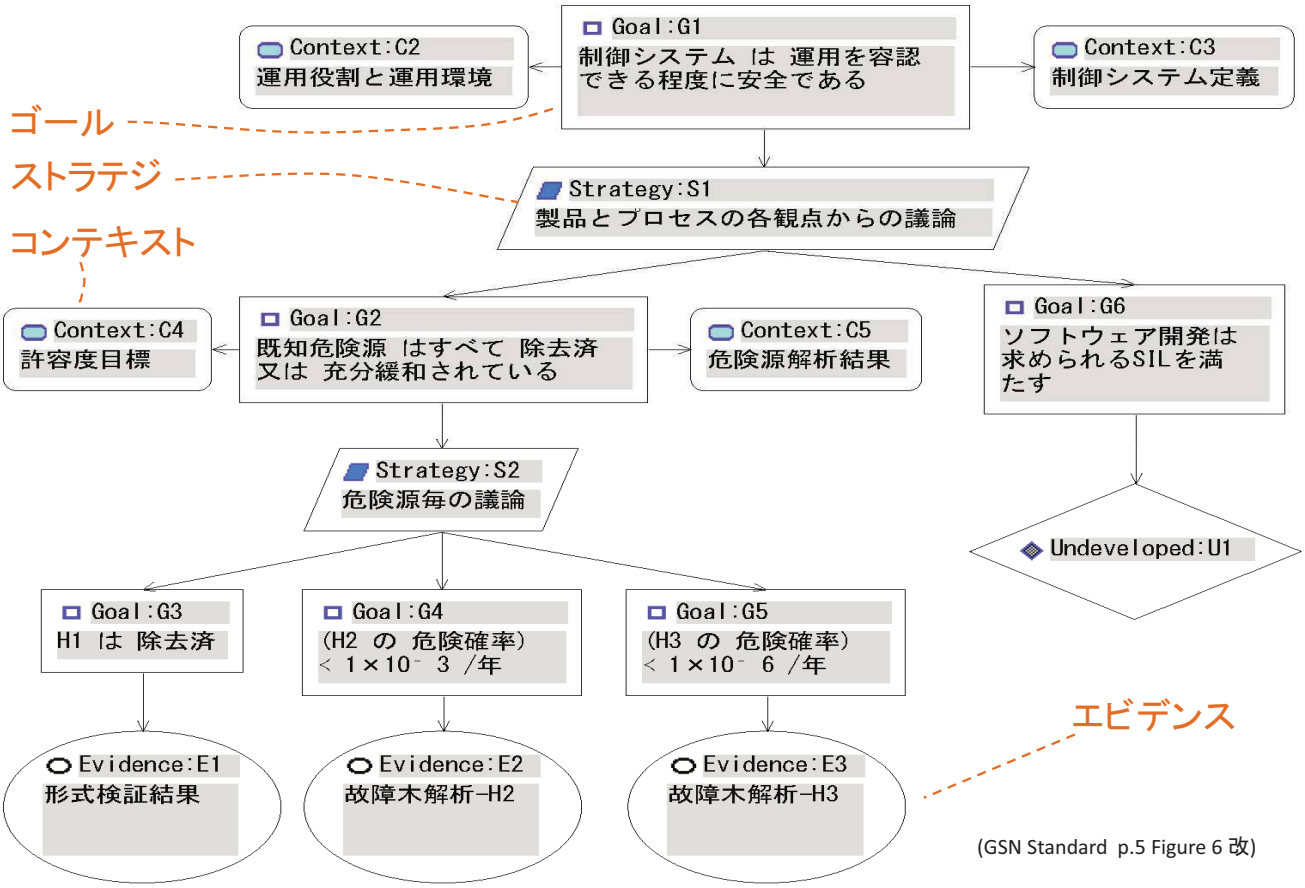
システムが要求を満たすことを
一連の主張、証拠、それらを結ぶ**明示的議論**によって
理由付き・監査可能な形で示す**文書一式**。

- Safety case が元: (欧)安全規制行政が prescriptive から goal-based にシフトする中で重要度を増す (90年代);
  安全管理活動の top-level control document.



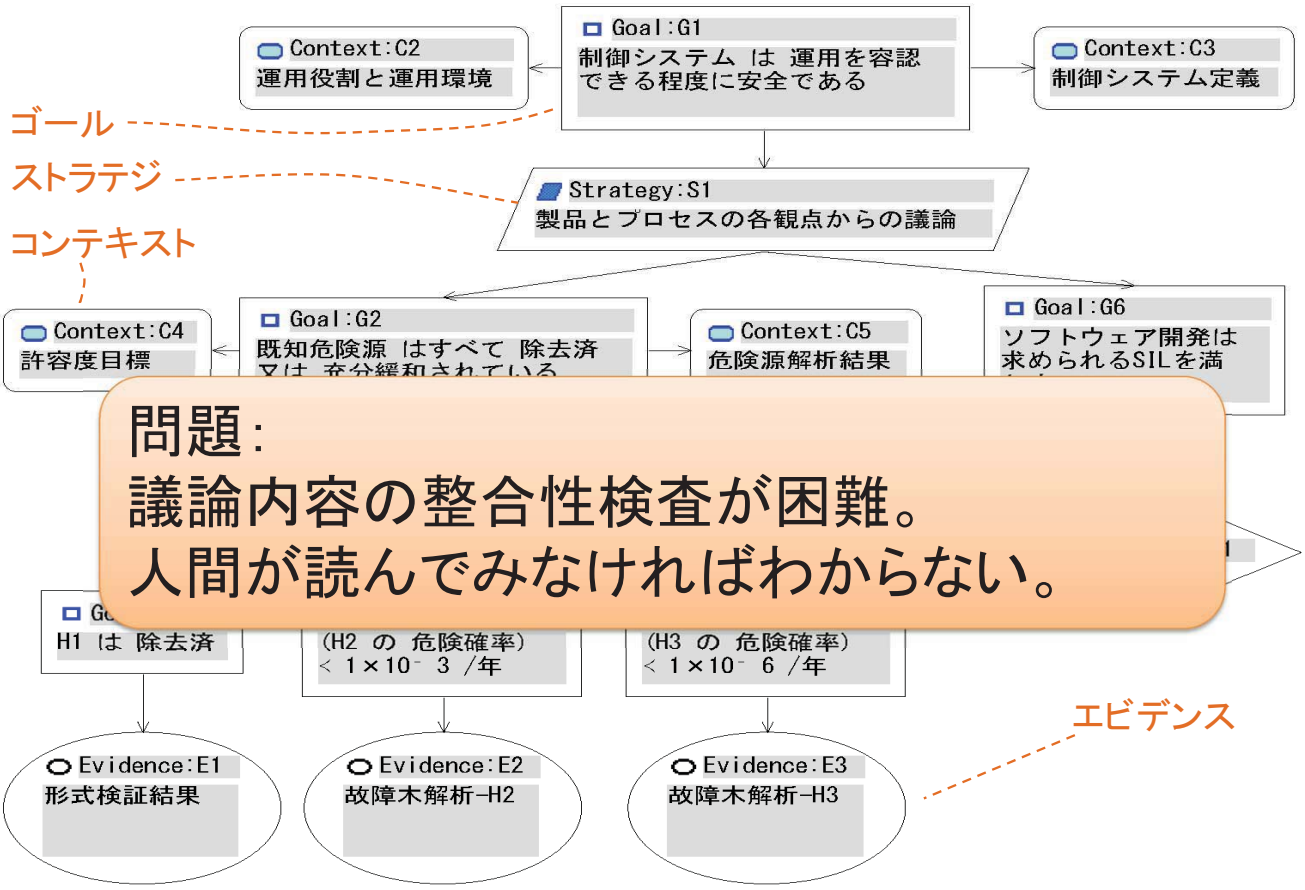
- ACは他の特質のcaseも含む総称 (reliability & maintainability-case, security-, ...)
- 規制・標準での義務化: 石油リグ、原発、防衛、航空、鉄道、自動車、医療機器、...
- **問題点: 複雑・膨大な議論の構築と整合性レビューは非常に困難**

アシュランス議論の図式表記



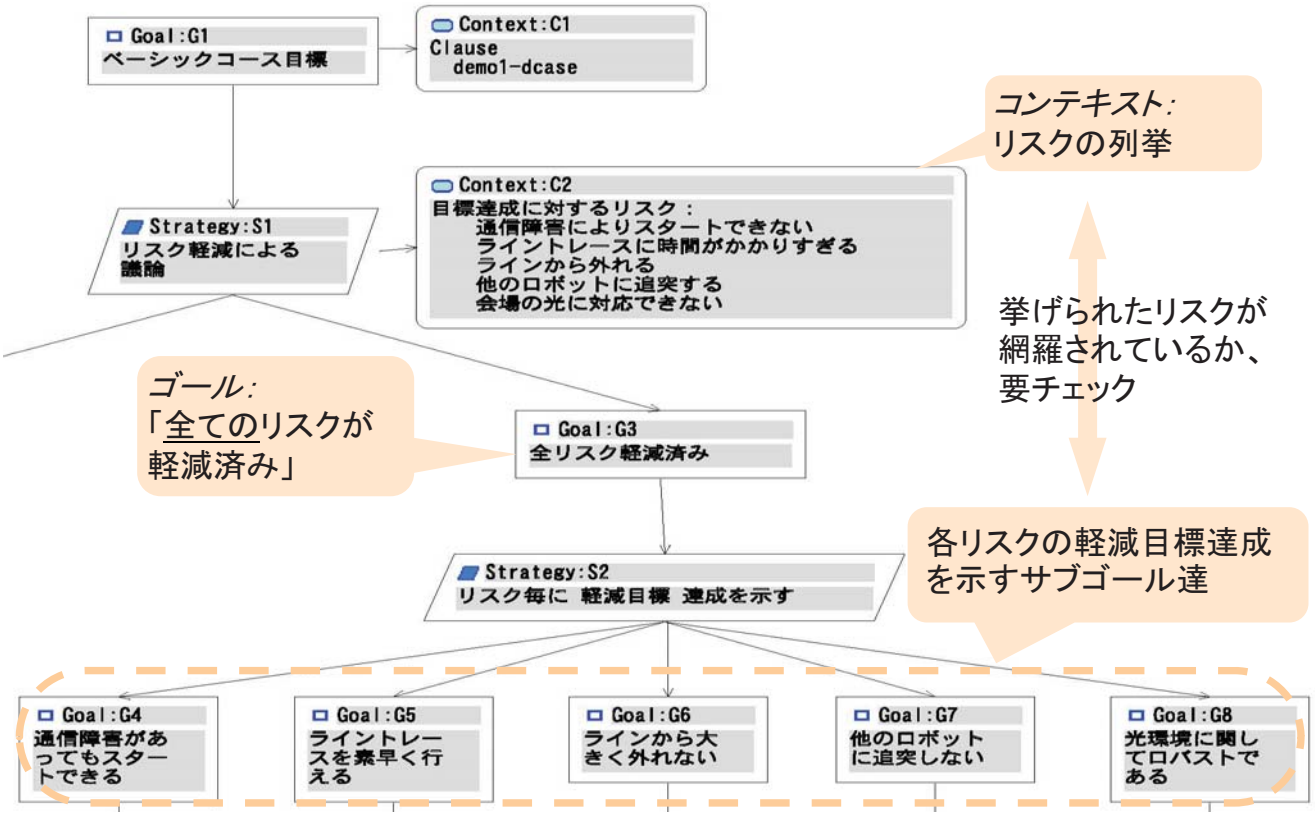
(GSN Standard p.5 Figure 6 改)

アシュランス議論の図式表記

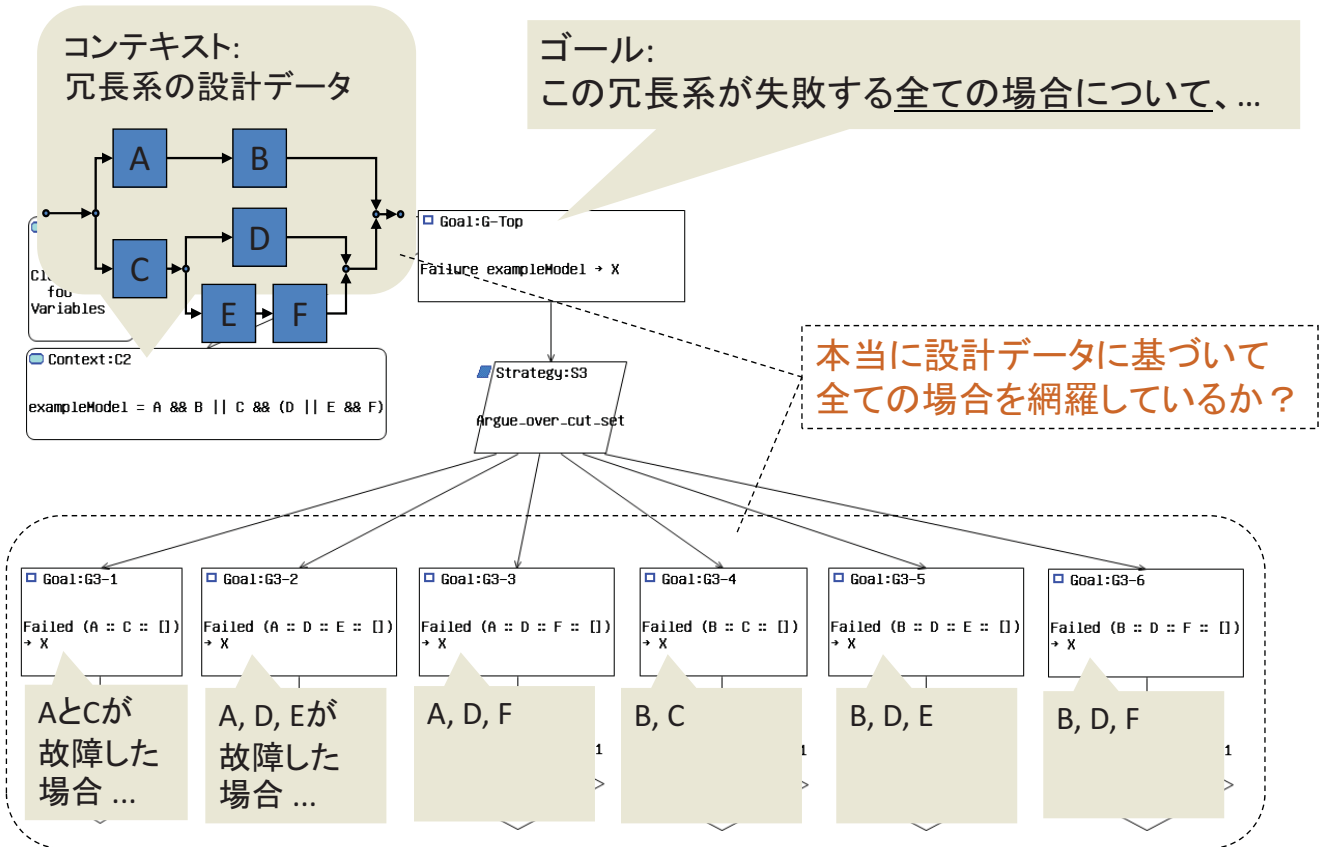


問題:
議論内容の整合性検査が困難。
人間が読んでみなければわからない。

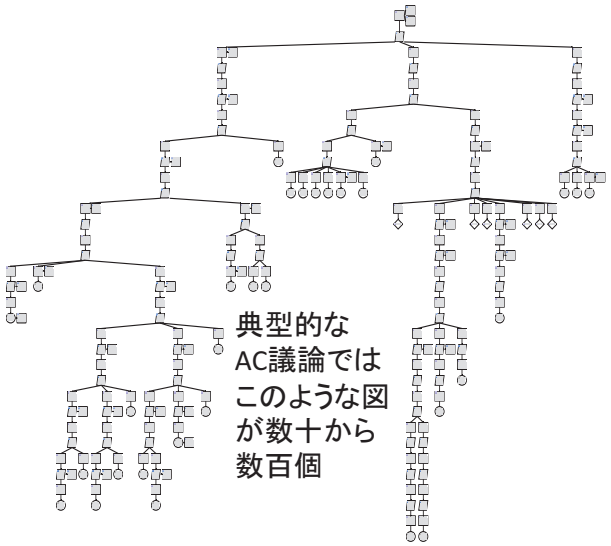
単純な整合性検査



専門的判断を要しない検査



課題



- 専門的判断を要しない検査も人力レビュー頼り。
- 不整合は局所的とは限らない
  - 遠く離れた枝間
  - 何百ファイルにも分かれた議論間
  - 議論と外部参照データとの間
- 各議論、データは多数の手で刻々変更される

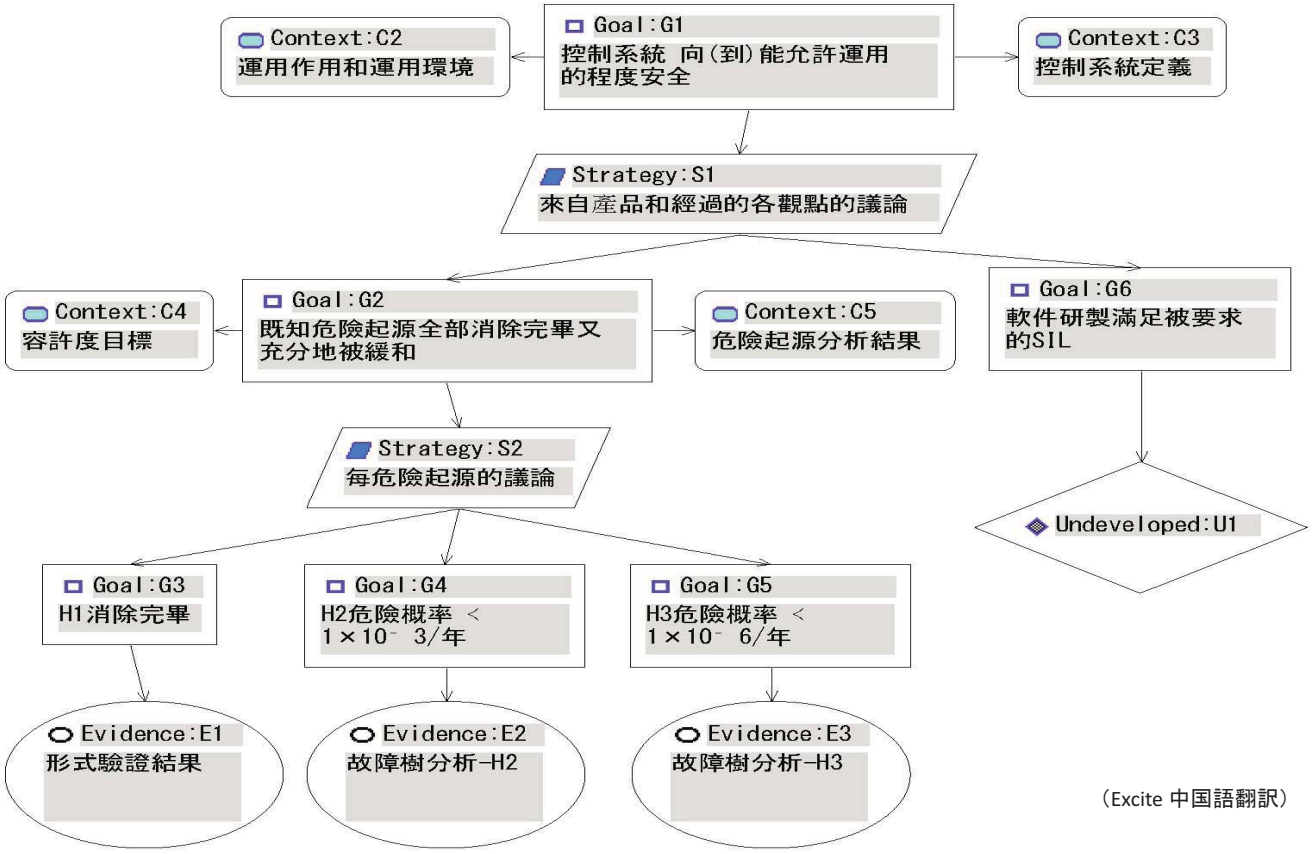
- 機械に検査できることは機械で検査
- レビュワーは専門的判断を下すことに集中できるように

アプローチ: 形式アシュランスケース

- 人力レビューでは、人間がゴール記述他を読んで、
  - *システムや環境を構成する「もの」*
  - *システムに要求される性質、環境の制約、仮定などの考慮すべき「こと」*

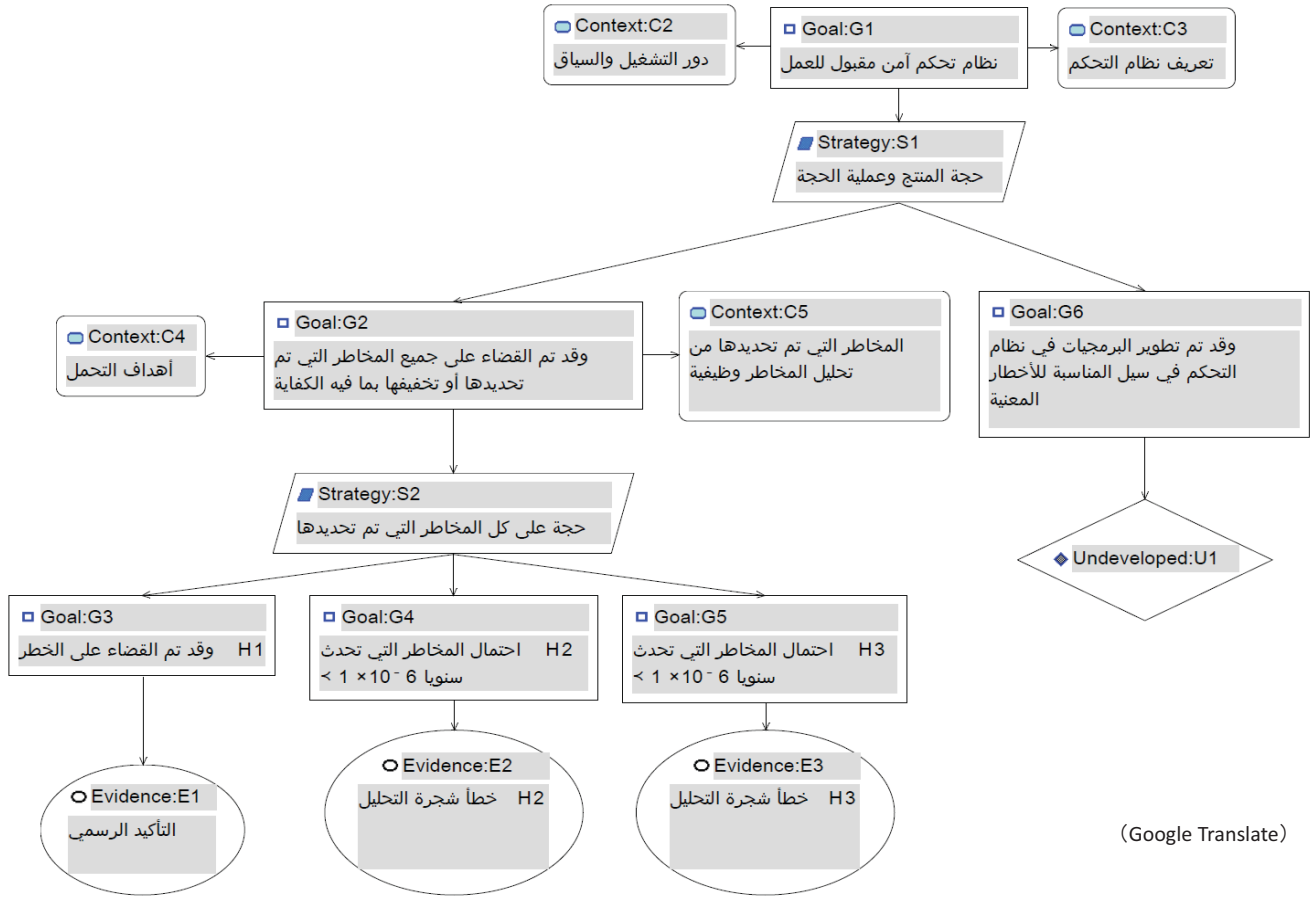
  にどんなもの・ことがあるか(**オントロジー**)を**文法・言葉の意味から理解**した上で、議論整合性を判断。
- **形式AC ＝ 機械にも解るオントロジーの定義 & それに基づく議論**
- 議論の整合性検査を「議論がきちんと定義に基づいているかどうか」の機械的検査に帰着

整合的？



（Excite 中国語翻訳）

整合的？



（Google Translate）

アプローチ: 形式アシュランスケース

- 人力レビューでは、人間がゴール記述他を読んで、
  - *システムや環境を構成する「もの」*
  - *システムに要求される性質、環境の制約、仮定などの考慮すべき「こと」*

  にどんなもの・ことがあるか（**オントロジー**）を
  文法・言葉の意味から理解した上で、議論整合性を判断。
- **形式AC ＝ 機械にも解るオントロジーの定義 & それに基づく議論**
- 議論の整合性検査を
  「この議論は定義に正しく基づいているか？」の
  機械的検査に帰着　（定義は適切か？とは別）

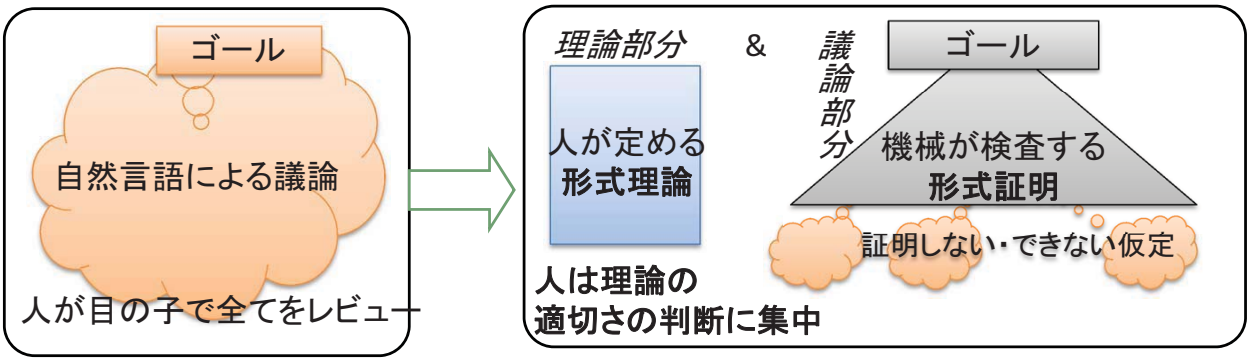
形式アシュランスケース

- **形式アシュランスケース**
  ＝機械にも解るオントロジーの定義 & それに基づく議論
  ＝形式理論の定義 & その理論の中での形式証明

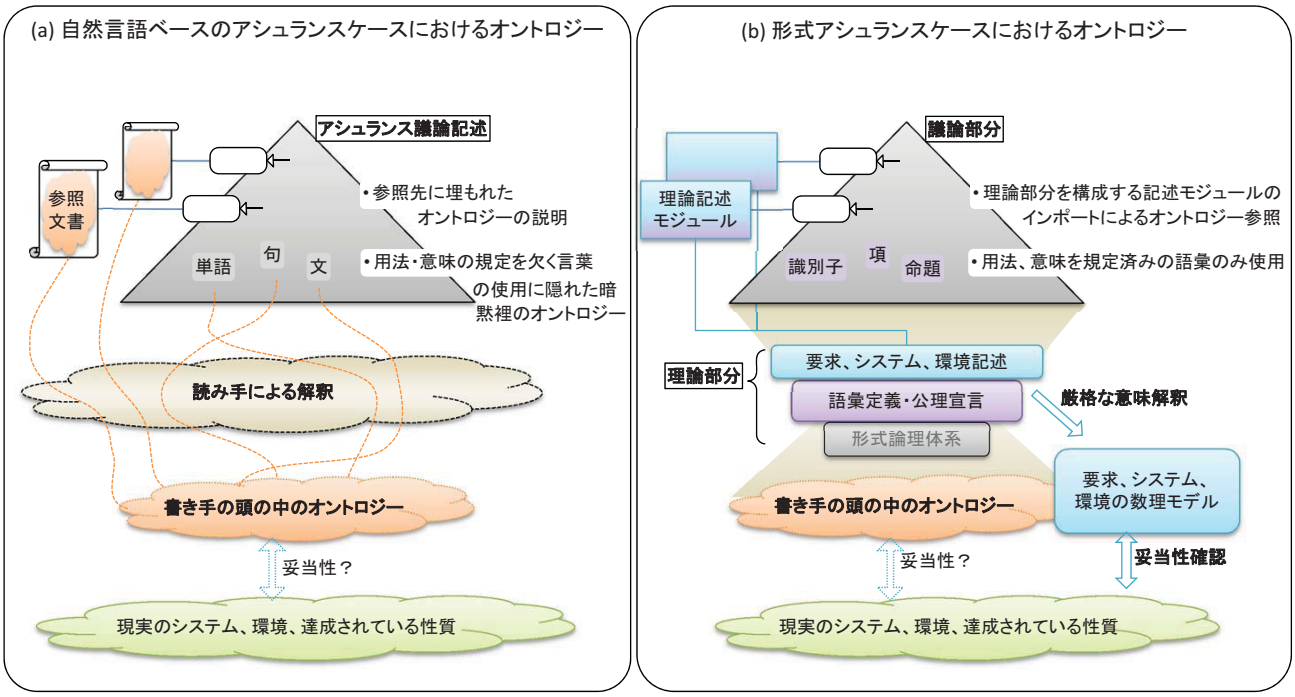
形式論理での 語彙定義, 公理の宣言（理論部分）

語彙, 公理からの推論 の正しい組み合わせ（議論部分）

- 議論の整合性検査をすべて
  「これは形式証明になっているか？」の機械的検査に帰着。



形式ACにおけるオントロジー



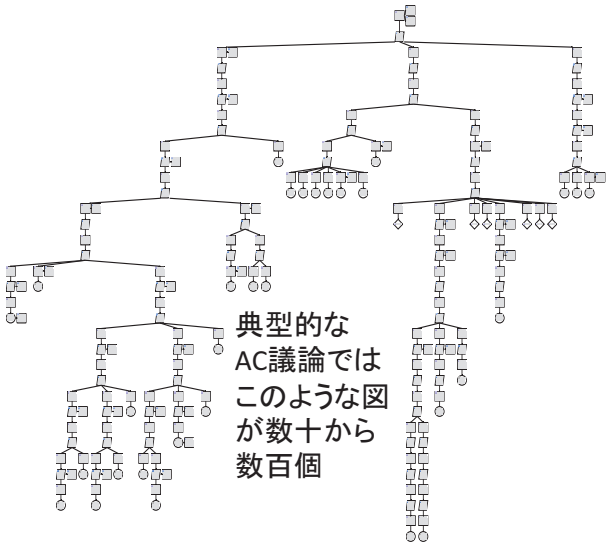
FACIA(Formal Assurance Case in Agda)

- 普通の述語論理等の記法は不便で使えない。
- 「命題は型、**証明はプログラム**」の確立された対応を適用。
  - 命題 G　= G成立の直接の証拠と認められるデータの型
  - Gの証明 = G型のデータを作るプログラム

  それに適した型付プログラミング言語 **Agda** でACを記述

- **FACIA** ＝機械にも解るオントロジーの定義 & それに基づく議論
  ＝ 型・定数・関数を宣言・定義するライブラリ
  & ライブラリを用いたプログラム
- 議論の整合性検査を、
  「このプログラムはライブラリを正しく使っているか？」の
  プログラムの型検査に帰着

課題



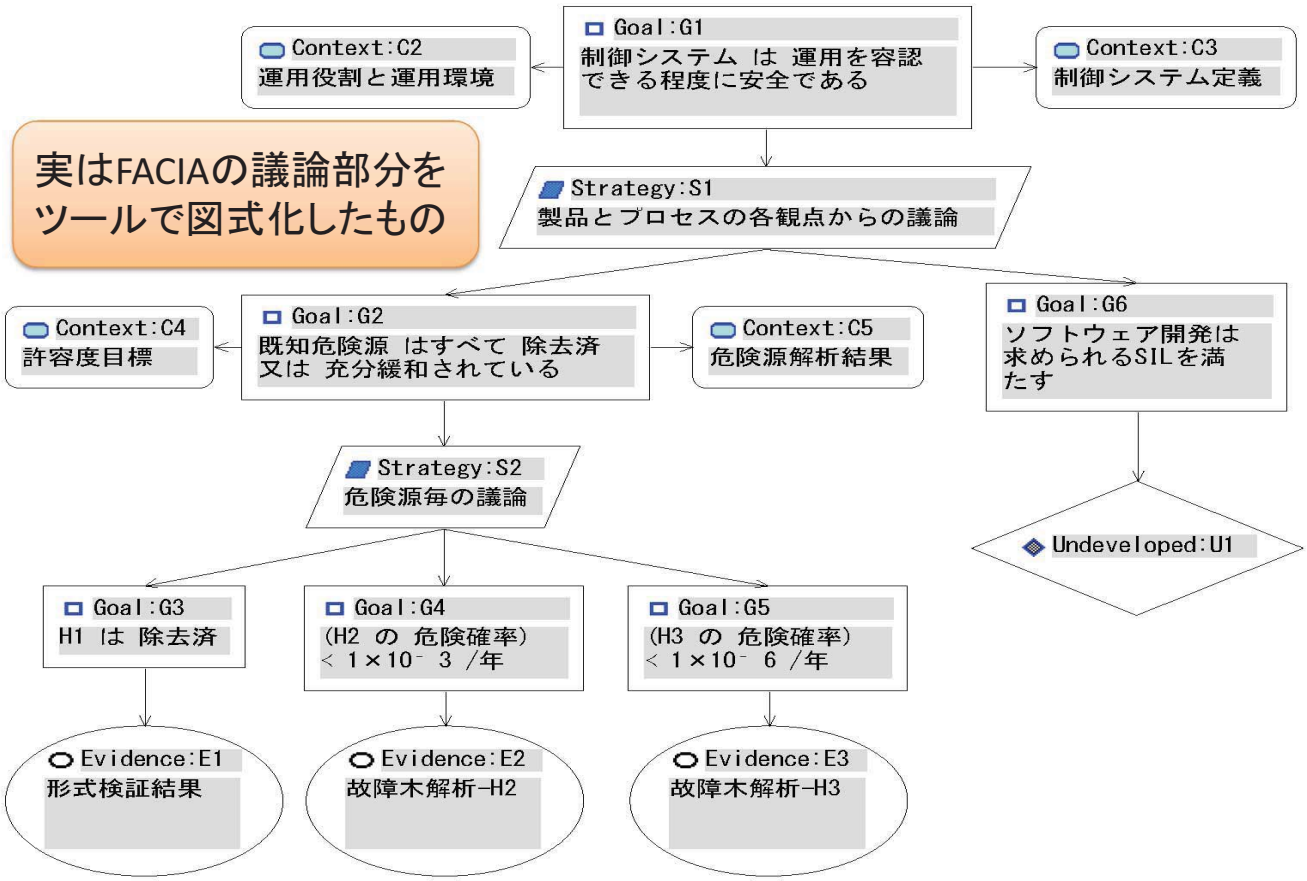
- 専門的判断を要しない検査も人力レビュー頼り。
- 不整合は局所的とは限らない
  - 遠く離れた枝間
  - 何百ファイルにも分かれた議論間
  - 議論と外部参照データとの間
- 各議論、データは多数の手で刻々変更される

- 機械に検査できることは機械で検査
- レビュワーは専門的判断を下すことに集中できるように

課題の解決

- AC ＝ ライブラリ定義＆プログラムとしてプログラミング、ソフトウェア工学の技術を適用
- **議論の整合性検査　　＝1プログラムの型検査**
  数百の関連議論の検査 ＝数百ファイルのプロジェクトのビルド
- 抽象化、モジュール化等のプログラミング技法で規模に対応
- 変更管理、構成管理等にソフトウェア工学の手法を適用
  型検査は、変更の影響範囲の判定にも有効

FACIA例



FACIA例

```
module Example1-1-tmp where
open import D-Case-in-Agda

module 語彙と定義 where
  module C3-制御システム定義 where
    postulate
      制御システム型 : Set
      制御システム : 制御システム型
  module C5-危険源解析結果 where
    data 既知危険源 : Set where
      H1 H2 H3 : 既知危険源
    postulate
      危険確率 : 既知危険源 → 確率型
  module C4-許容度目標 where
    open C5-危険源解析結果
    緩和目標 : 既知危険源 → 確率型
    緩和目標 H1 = 確率零
    緩和目標 H2 = 1×10⁻ 3 /年
    緩和目標 H3 = 1×10⁻ 6 /年
    充分緩和されている : 既知危険源 → Set
    充分緩和されている h = 危険確率 h < h の 緩和目標
    postulate
      除去済 : 既知危険源 → Set
      危険源毎の議論 :
          H1 は 除去済 →
          H2 は 充分緩和されている →
          H3 は 充分緩和されている →
          ∀ h → h は 除去済 又は 充分緩和されている
      危険源毎の議論 p1 p2 p3 H1 = または-右 p1
      危険源毎の議論 p1 p2 p3 H2 = または-左 p2
      危険源毎の議論 p1 p2 p3 H3 = または-左 p3
  module C2-運用役割と運用環境 where
    open C3-制御システム定義
    open C4-許容度目標
    open C5-危険源解析結果
    postulate
      ソフトウェア開発は求められるSILを満たす : Set
      運用を容認できる程度に安全である : 制御システム型 → Set
      製品とプロセスの各観点からの議論 :
        (既知危険源 はすべて 除去済 又は 充分緩和されている) →
        ソフトウェア開発は求められるSILを満たす →
        制御システム は 運用を容認できる程度に安全である

module 証憑 where
  open 語彙と定義
  open C5-危険源解析結果
  open C4-許容度目標
  postulate
    形式検証結果 : H1 は 除去済
    故障木解析-H2 : 危険確率 H2 < 1×10⁻ 3 /年
    故障木解析-H3 : 危険確率 H3 < 1×10⁻ 6 /年

module 議論 where
  open 語彙と定義
  open 証憑
  main =
      let open C2-運用役割と運用環境
          open C3-制御システム定義 in
      制御システム は 運用を容認できる程度に安全である
      ∵ 製品とプロセスの各観点からの議論
        • (let open C4-許容度目標
               open C5-危険源解析結果 in
           既知危険源 はすべて 除去済 又は 充分緩和されている
           ∵ 危険源毎の議論
             • (H1 は 除去済 ∵ 形式検証結果)
             • (H2 の 危険確率 < 1×10⁻ 3 /年 ∵ 故障木解析-H2)
             • (H3 の 危険確率 < 1×10⁻ 6 /年 ∵ 故障木解析-H3))
        • (ソフトウェア開発は求められるSILを満たす ∵ 未確認)
```

図式表記では暗黙裡

理論部分 = ライブラリ定義

- 基本概念の、未定義の型・関数としての宣言
  複合概念の型・関数としての定義
  それらの間の関係の天下り的公理の宣言
  宣言・定義から導かれるストラテジーとしての補題の証明
- レビュワーの専門的判断、関係者間の合意による正当化が必要。

証拠の宣言

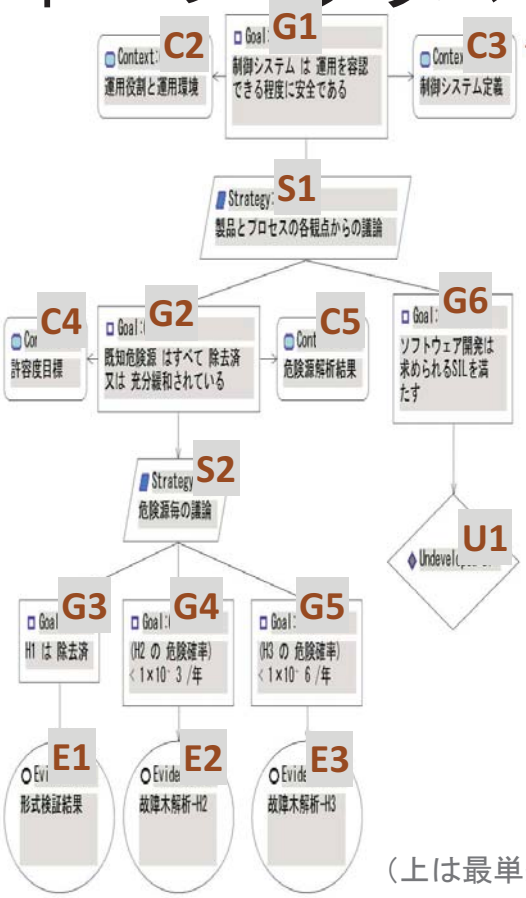
- 末端ゴール命題成立の物的証拠への参照
- 要正当化

図式表記はここの構造のみ

議論部分 = メインプログラム

- 末端証拠データにライブラリ関数を組み合わせて適用してトップゴールの証拠を作るプログラム
- 理論に照らして整合的な議論であることを型検査が保証
- トップゴール内容の適切さは要正当化

木～プログラム




```
let open C2-運用役割と運用環境
    open C3-制御システム定義 in
-- G1
制御システム は 運用を容認できる程度に安全である
-- S1
∵ 製品とプロセスの各観点からの議論

  • (let open C4-許容度目標
        open C5-危険源解析結果 in
    -- G2
       既知危険源 はすべて 除去済 又は 充分緩和されている
      -- S2
       ∵ 危険源毎の議論
          -- G3                E1
          • (H1 は 除去済 ∵ 形式検証結果)
          -- G4                                  E2
          • (H2 の 危険確率 < 1×10⁻ 3 /年 ∵ 故障木解析-H2)
          -- G5                                  E3
          • (H3 の 危険確率 < 1×10⁻ 6 /年 ∵ 故障木解析-H3))

  -- G6                                        U1
  • (ソフトウェア開発は求められるSILを満たす ∵ 未確認)
```

（上は最単純例。プログラム側は、同じ意味のより高度な表現も可能）

語彙定義～型・関数の宣言、定義

```
module C3-制御システム定義 where
  postulate
    制御システム型 : Set
    制御システム : 制御システム型
module C5-危険源解析結果 where
  data 既知危険源 : Set where
    H1 H2 H3 : 既知危険源
  postulate
    危険確率 : 既知危険源 → 確率型
module C4-許容度目標 where
  open C5-危険源解析結果
  緩和目標 : 既知危険源 → 確率型
  緩和目標 H1 = 確率零
  緩和目標 H2 = 1×10⁻ 3 /年
  緩和目標 H3 = 1×10⁻ 6 /年
  充分緩和されている : 既知危険源 → Set
  充分緩和されている h = 危険確率 h < h の 緩和目標
```

分析しない語はプリミティブな定数として宣言

分析する語は定義（ここでは型）

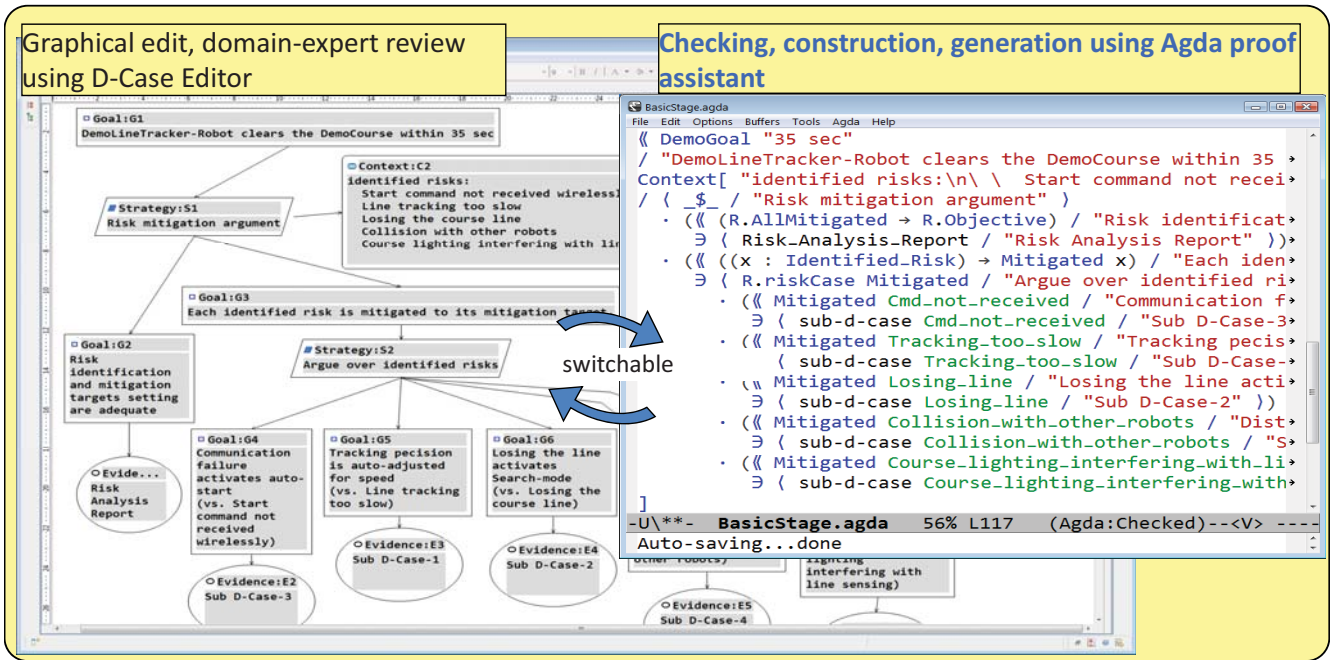
分析する語は定義（ここでは関数）

宣言・定義は積み重ねられていく

整合性検証ツール “D-Case in Agda”

- Agdaプログラムとしての議論と図式記法との間の双方向変換
- Agda証明支援系（Agda開発環境）で議論の整合性検査、議論の半自動構成

dcase-en dcase-agda code-agda



変化対応における整合性検査のシナリオ

- 簡単例
- 中規模例

まとめ：形式AC、FACIA

- 自然言語記述AC = 明示されないオントロジー & 自然言語議論
  → オントロジーの定義 & に基づく議論
  → 形式AC = 形式理論 & 形式証明
  → FACIA = 型・関数のライブラリ & プログラム
- 議論の整合性検査 = プログラムの型検査
- プログラミング、ソフトウェア工学、証明支援の技法を駆使して大規模でも理解しやすい・変更しやすい議論を構築

プログラム分野で当たり前に解決済みのこと（*でも50年掛かったこと*）は、議論記述向けに場当たり的に解こうとすべきでない。

- 関数抽象（パターン）、自由変数・束縛変数（まともな代入）、帰納型・再帰関数（止まるループ）、高階関数、抽象データ型、局所宣言、…
  パラメタ付モジュール、定義のスコープ、名前空間、情報隠蔽、…
- 分割コンパイル（分割検査）、SCC、変更管理、構成管理、…
- ライブラリ、ソフトウェア・フレームワーク、…

まとめ：形式AC、FACIA

- → オントロジーの定義 & に基づく議論
  → 形式AC = 形式理論 & 形式証明
  → FACIA = 型・関数のライブラリ & プログラム
- 議論の整合性検査 = プログラムの型検査
- プログラミング、ソフトウェア工学、証明支援の技法を駆使
  規模でも理解しやすい・変更しやすい議論を構築

今後

- FACIAの 入出力付プログラムとしての実行
  →現実の状況を取り込み、現実に効果を与える動的議論

- AC形式化の本義：AC自体を「対象」として厳格に扱うこと。
  - **対象ACの性質を論じるメタ・アシュランスケース**
  - **ACを対象とした妥当性維持のための操作**

まとめ：形式AC、FACIA

- → オントロジーの定義 & に基づく議論
  - → 形式AC = 形式理論 & 形式証明
  - → **FACIA** = 型・関数のライブラリ & プログラム
- 議論の整合性検査 = プログラムの型検査
- プログ

今後

形式ACの殻に、中身を詰めることも最重要

- ➢ **形式ACの開発フレームワーク**（DEOS & 特定領域での）

- FACIAの 入出力付プログラムとしての実行
  →現実の状況を取り込み、現実に効果を与える動的議論

- AC形式化の本義：AC自体を「対象」として厳格に扱うこと。
  - ➢ **対象ACの性質を論じるメタ・アシュランスケース**
  - ➢ **ACを対象とした妥当性維持のための操作**