ソフトウェア信頼性向上のための形式技法・開発支援ツールの研究

2012. 9. 10

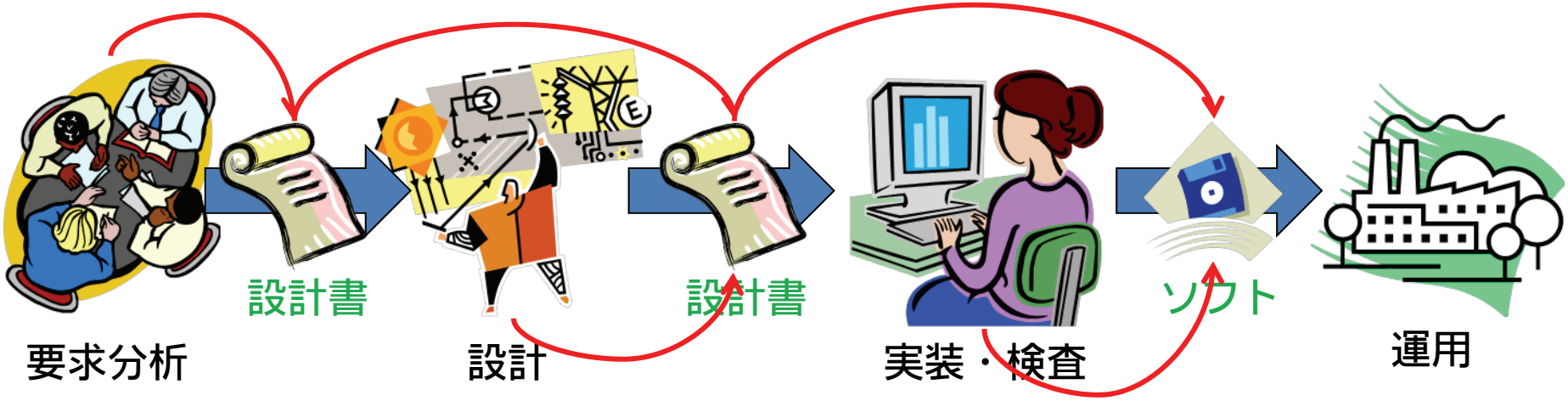
産業技術総合研究所
セキュアシステム研究部門
高信頼ソフトウェア研究グループ
大岩　寛

組み込み機器とソフトウェア

- クリティカルな用途にソフトウェア制御
  - 車両・航空機制御の伝送・電子制御化
  - 車両等の自動運転
  - 医療機器等
  - 社会インフラ
    - スマートグリッドなど

機能要求とソフトウェア

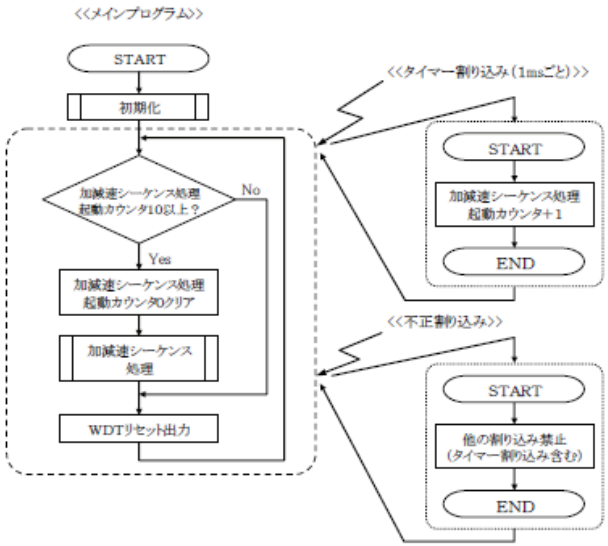
- ソフトウェアは指示されたとおりにしか動作しない
  - 動作する段階では設計仕様やその背景にある人間の意図は汲んでくれない
  - 人間が間違ったとおりに計算機は間違える



信頼できないソフトによる障害事例 (1)

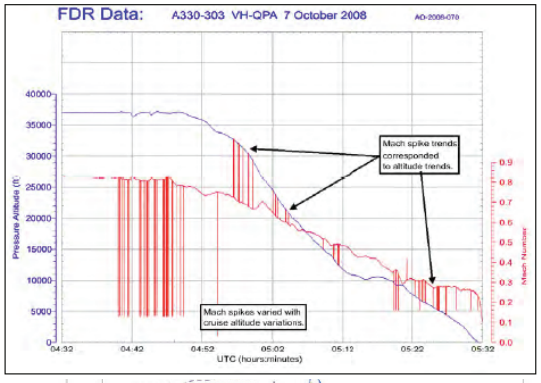
- 湘南モノレール線車両暴走事故 (2008. 2)
  - 駅で車両が停止できず信号冒進
  - 運輸安全委員会の調査で加速制御装置のソフトウェア不良が判明 (RA2009-6)
    - 異常時処理に2つの誤り
      - 入力を受け付けなくなる = 加速状態が解除できない
      - 故障検知装置 (WDT) も入力を受け付けない状態を正常動作と誤認識
    - アクセルとブレーキを両方働かせた状態で駅に進入

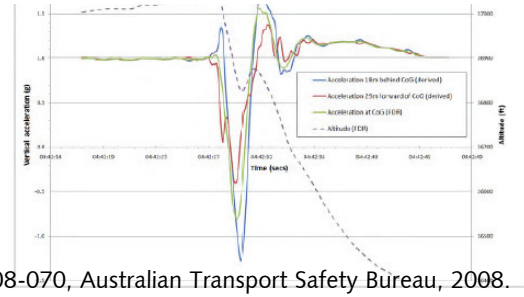
図/写真: 国土交通省運輸安全委員会 鉄道事故調査報告書RA2009-6 より引用



信頼できないソフトによる障害事例 (2)

- カンタス航空 72便 乱降下事故 (2008. 10. 7)
  - 飛行中の突然の急降下で115名重軽傷・緊急着陸
  - 豪国事故調査委員会の調査 (AO-2008-070)
    - 発端は慣性航法装置1台の故障
      - 原因不明のデータ混信
      - 過大な上昇を誤報告
    - 多重系によるバックアップが機能せず
      - 他の2台の装置の正常なデータを使わず、誤ったデータを元に自動操縦が降下指示





Figures/picture Source: ATSB Transport Safety Report AO-2008-070, Australian Transport Safety Bureau, 2008.

信頼できないソフトによる障害事例 (3)

- その他にも…
  - Therac-25 (1985-87)
    - 放射線照射治療機械のバグで最低5名死亡
      - 誤操作に対するインターロックが機能せず
    - 2000年パナマで類似の事件、8名死亡
    - 2006年にもフランスで類似の事件、1名死亡
  - Intel Pentium FDIV バグ (1993)
    - 除算表のバグによりCPUチップを全回収
    - 400億円以上のリコールに

ソフトの信頼性に関わる事例 (4)

写真: Wikimedia Commonsより引用

- 米国トヨタ車リコール騒動 (2009-2011)
  - 2009: 複数のトヨタ車の暴走事故が報告される
  - 2010.2: 電子制御システムの不具合の可能性がUSDOT に指摘される
    - リコール騒ぎ・集団訴訟等に発展
    - 大きな損失（リコール関連だけで1000億以上）
  - 2011.2: NASA, USDOT, NHTSA の調査結果でソフトウェア設計に問題が見つからなかったことの最終報告

「ソフトウェアが信頼できる」ことの証明の難しさ

ソフトウェア信頼性と標準・認証

- 絶対的に高信頼が必要な用途にソフトウェアが用いられるようになる
  - 消費者機械: ミスが他人に実害を及ぼす用途
  - セキュリティ
- 機械的な仕組みと違い、目に見えない
  → 安全性を第三者が納得する手段が必要

- 安全性の標準化・認証化
  - セキュリティ, 消費者機械, 航空機など応用毎にそれぞれの安全性規格
  → 安全性認証取得の国際的重要性

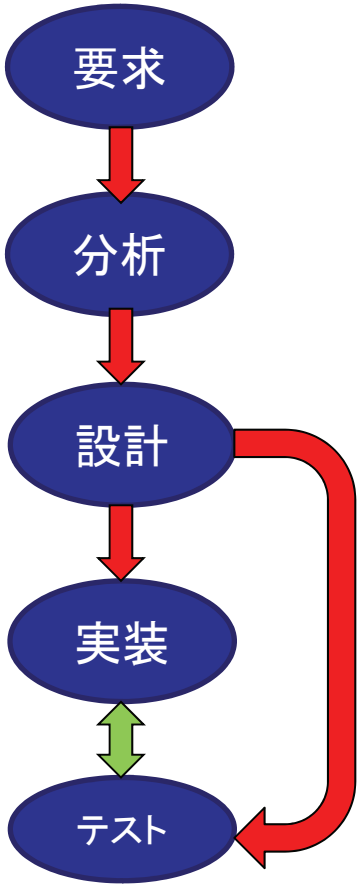
現状のソフトウェア開発の問題点

- 人手に頼った信頼性確保
  - 要求分析から実装までのプロセスの連係の問題
    - 各段階で個別にツール化 (or 紙ベース)
    - 情報化されていても、データとして連携する段階に至っていない
  - テスト項目は設計書から人が作成
    - テスト自体の正しさの保証がない

    → 人を納得させるのが困難

◆正しく作るコスト自体の増大

◆正しさを証明するコストの増大



研究の目標点

- 仕様に適合したソフトウェアを正確に実装する技術
- ソフトウェアが（上の意味で）正確で信頼できることを証明・提示できる技術
  - 安心してソフトを使えるようにする
  - 規格などの取得コストを低減する

- 要求される安全性レベルとコストに応じた適切な技術の選択肢を提示
  - 複数技術を自由に選択・併用できる環境を目指す

RISEC における研究体制

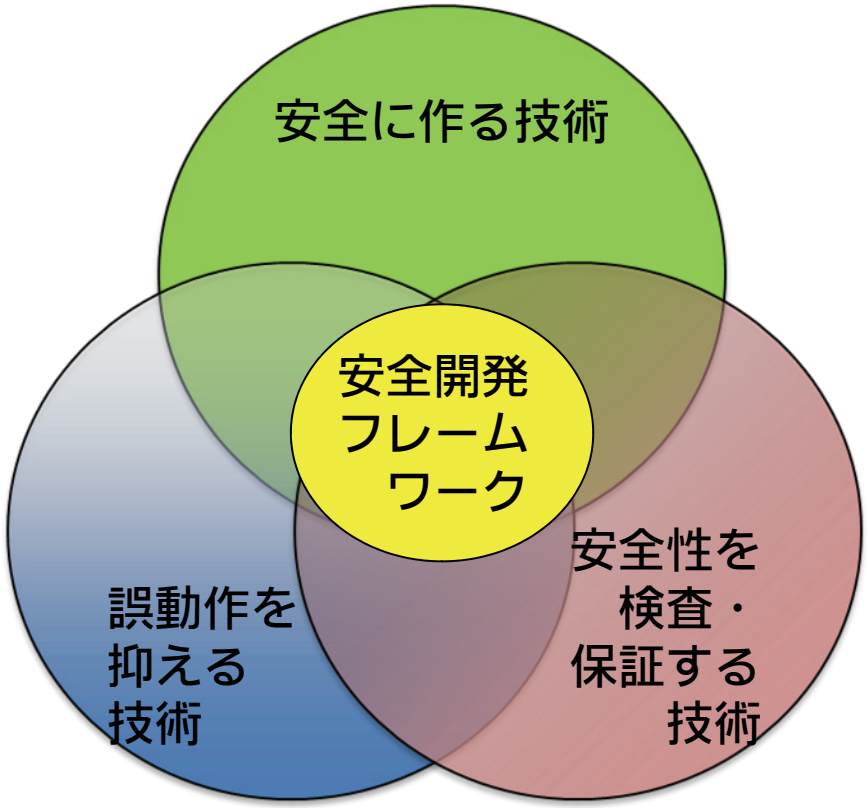
- 2グループで分担・連係

システムライフサイクル
研究グループ
- 関西・尼崎
- 主に上流工程
  - 要件分析・システム設計
  - モデル検査
  - システムテスト（ブラックボックス）
- 関西圏企業との連係

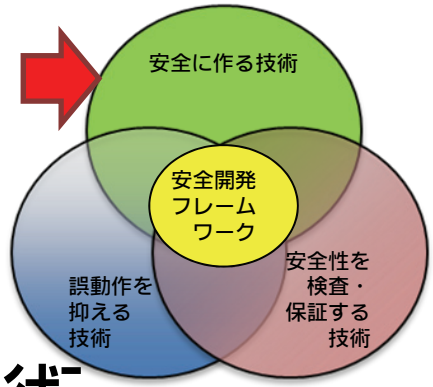
高信頼ソフトウェア
研究グループ
- つくば
- 主に下流工程
  - 詳細設計
  - 形式検証
  - ソフトウェア実装技術
  - 仮想化・隔離
  - テスト（ホワイトボックス）

研究する課題

- 大きく3種類の技術
  - 安全にソフトを設計・実装する技術
  - ソフトウェアの安全性を検査・保証する技術
  - ソフトウェアの誤動作を抑止する技術
  - これらを自由に組み合わせる技術フレームワーク



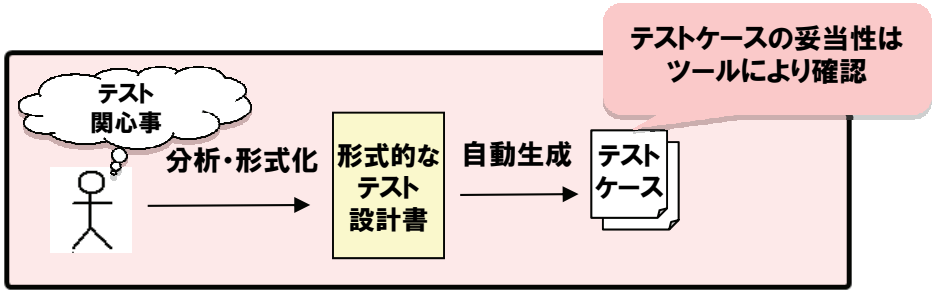
1. 安全に作る技術

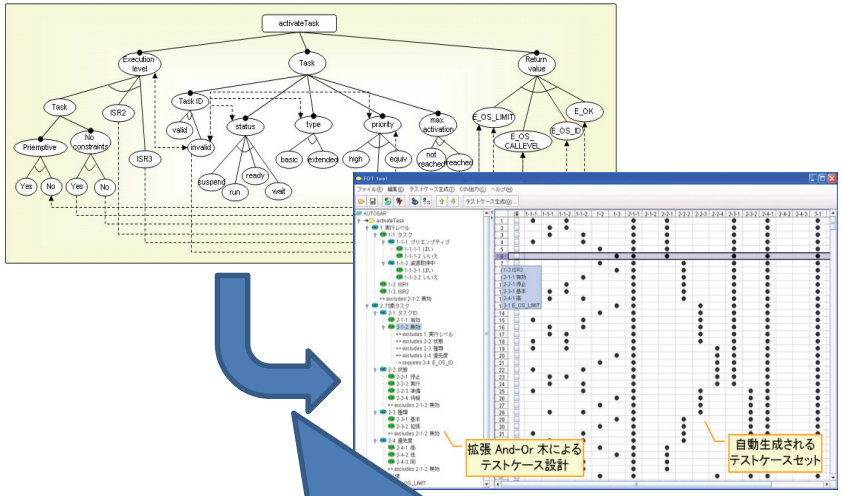


- 開発プロセス全体を通じて
  仕様と実装の一貫性を担保する技術
  - 問題解析の自動化・テスト自動生成
  - モデルを元にしたプログラム開発
  - プログラム自動生成

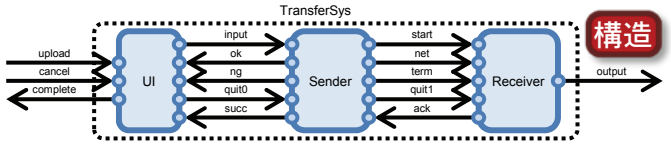
1. 安全に作る技術

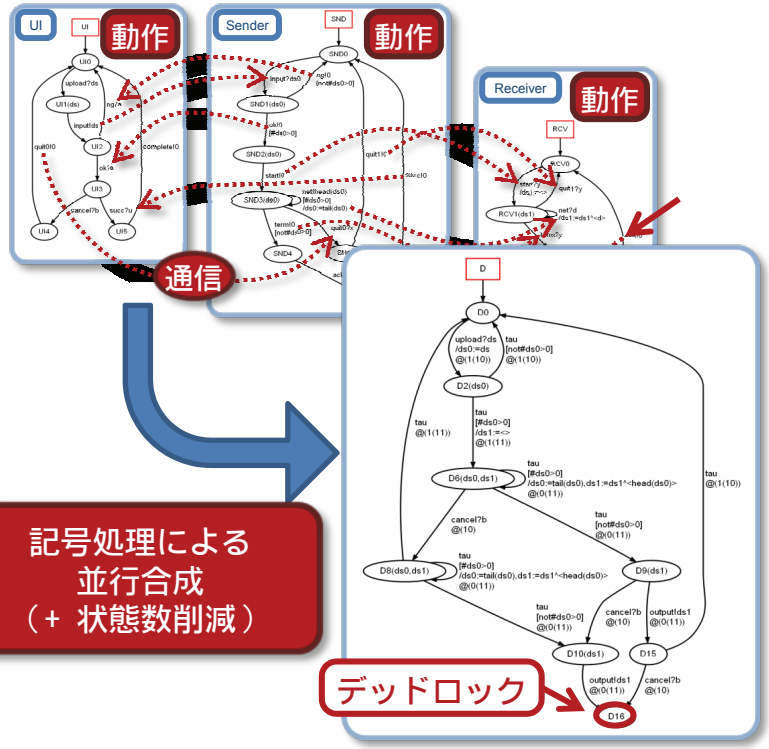
- 要求記述からテスト自動生成
- 並行プロセスの自動解析





次の講演で詳しく説明

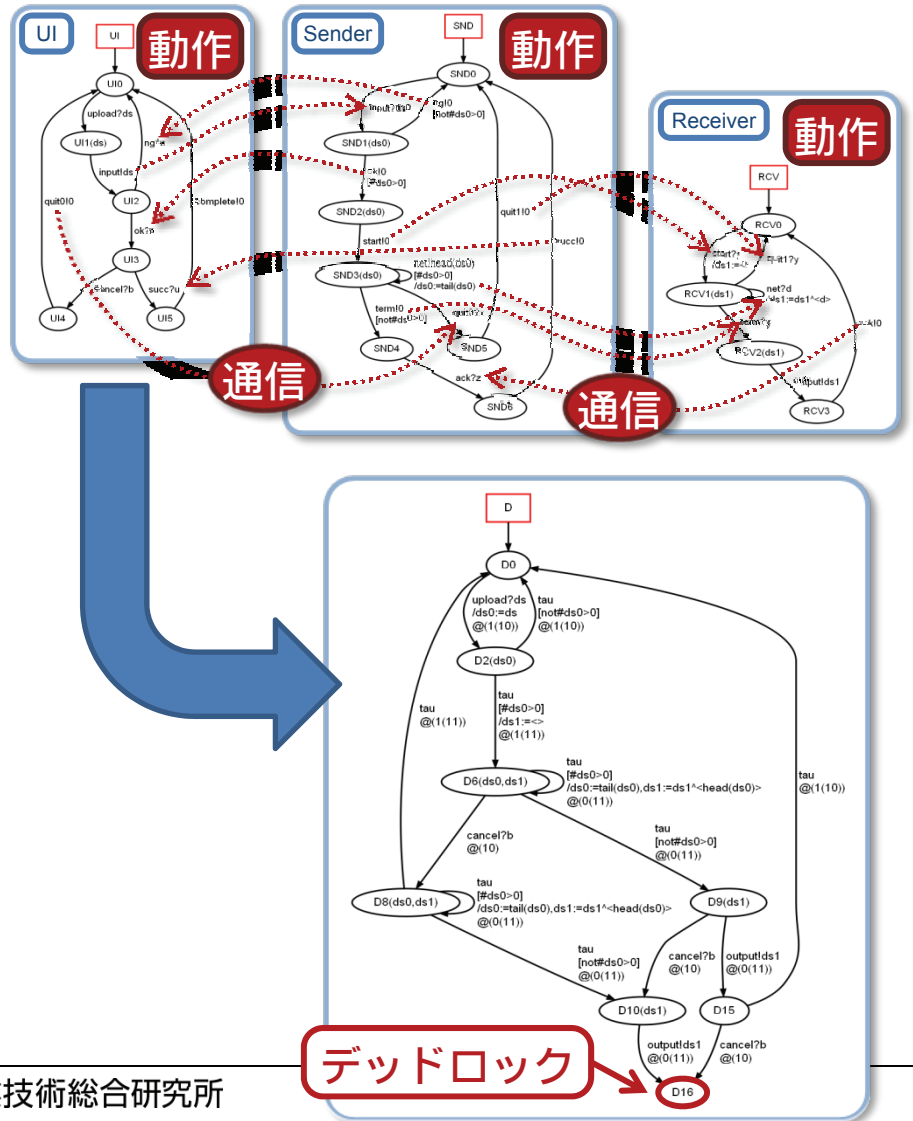




記号処理による
並行合成
（+ 状態数削減）

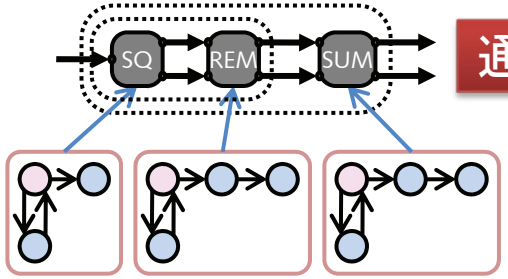
並行プログラムの自動解析

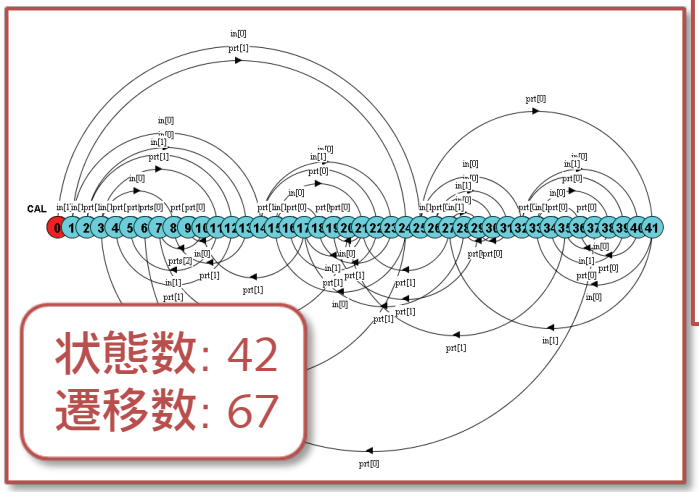
- 並行動作のある
  システムの振る舞い
  を簡単に解析できる
  システム
  - 各個別プロセスの仕様と結びつきの仕様から、システム全体の動作を自動的に解析・出力
    - 1枚の図で全体の動作を把握できる
    - 隠れていたデッドロックなどを発見可能



並行プログラムの自動解析

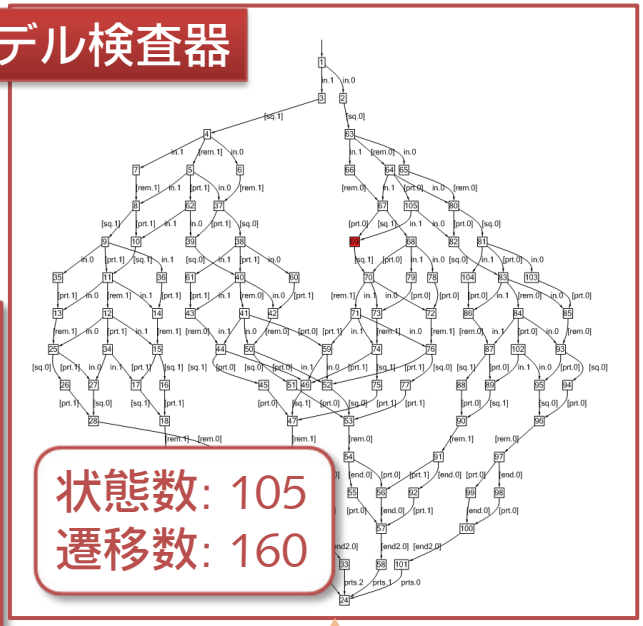
- 記号処理・定理検証技術とモデル検査を統合
  - 無駄な状態爆発を抑えてわかりやすい出力





状態数: 42
遷移数: 67

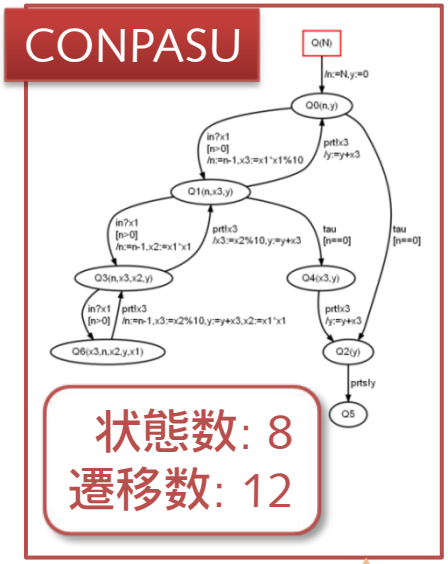
通常のモデル検査器



状態数: 105
遷移数: 160

入力値:{0,1}
入力回数:3

CONPASU

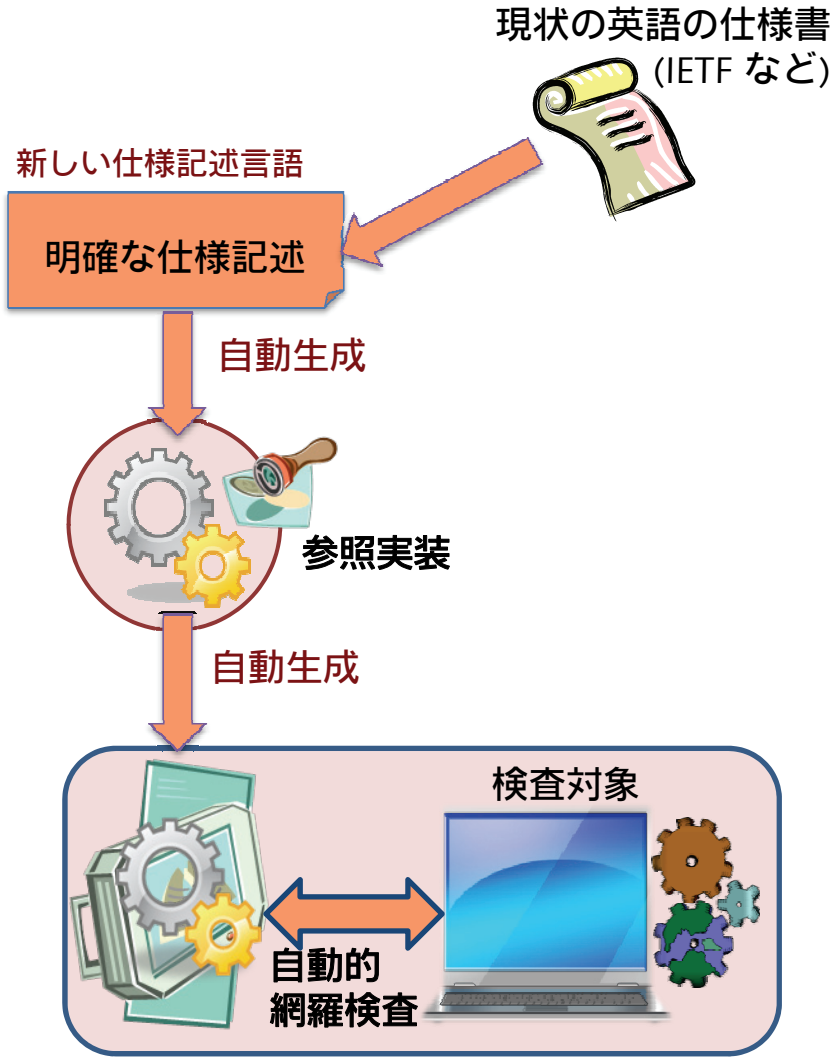


状態数: 8
遷移数: 12

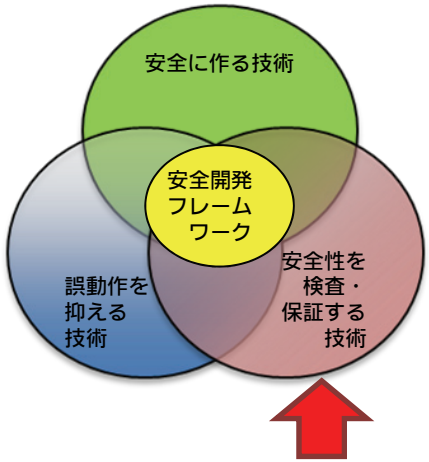
入力値:無制限
入力回数:任意

実装・検査器の自動生成

- 通信規約の仕様書から参照用の実装と動作検査器を自動生成
  - 仕様書の記述を見直し
  - 仕様書から参照実装を自動生成
    - 曖昧さに起因する相互運用性の問題を解決
  - 検査器自体の正しさを担保できる
    - テストの信頼性の向上



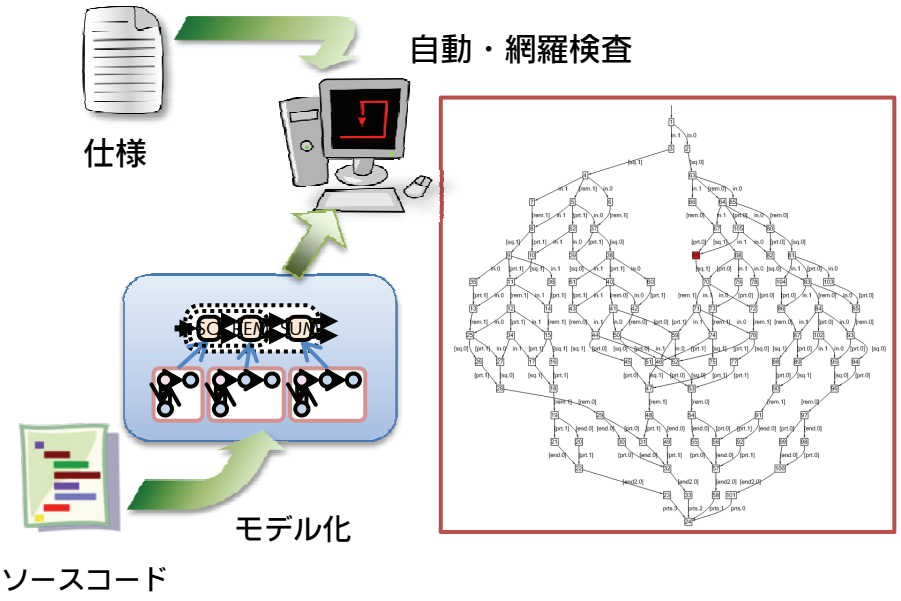
2. 安全性を検査/保証する技術



- 開発中のソフトウェアの性質や安全性を論理的に検証する技術
  - モデル検査
    - 端的に言うと、「モデル化」＋「網羅的検査」
  - 形式検証
    - 端的に言うと、「数学的証明作業」

モデル検査

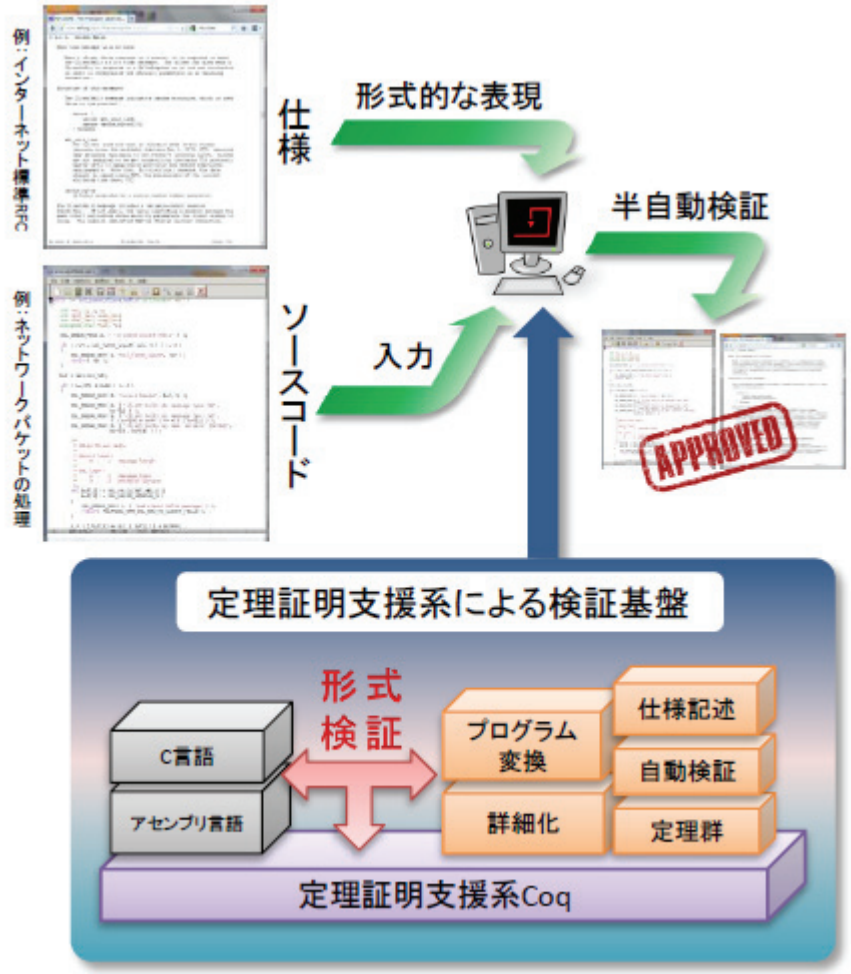
- ソフトウェアの設計や実装からモデルを作成
  - 重要な部分の振る舞いを抽出
- 網羅的に振る舞いを全検査
  - 不要な状態が発生ないことをしらみつぶしに探索
  - 見つかれば、反例として設計変更にフィードバック



- 既に実績のある技術
- 開発現場への展開と現場からの要望に基づく技術改良
  - 関西中心に共同研究多数

ソフトウェア実装の形式検証

- ソフトウェアを論理的な記述として証明の対象にする
  - 仕様を満たすことを数学定理として証明
  - 証明の機械化による効率化と誤りの排除
  - 正しいということの「証明」を生成
    - 他人に「納得させる」証拠の提示
    - 将来の認証基盤として



ソフトウェア実装の形式検証

- モデル検査との違い
  - 証明できる性質に原理的には制限がない
    - 定理として書けることなら原理的には何でも
  - （人間の知恵が及べば）
    問題サイズにも制限がない
    - 問題の数学的性質をうまく証明に反映できる
    - 規則的だが状態が爆発するような処理に有効
  - 現状では高コスト
    - 半自動的なシステムなので人手が必要
    - 数学的な理解のあるスタッフが必要
- 現在でも一部の応用で使われ始めている
  - CPUの演算装置の正当性の検証 (AMD, Intel 等)
  - 航空機のソフトウェアへの応用の模索

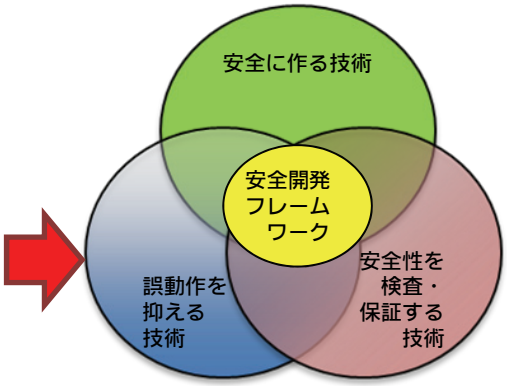
ソフトウェア実装の形式検証

- 我々の研究の方向性
  - 仕様だけでなく、実際の実装に関する形式検証
    - ソフトウェア実装が設計された仕様を満たすことを形式的に証明
    - ソフトウェア実装が要求される性質を満たすことを直接形式的に証明
- 現在はC言語とアセンブリ言語が主な研究対象
  - Coq 定理証明器上で直接的に定式化
  - 生のソースコードを扱える体系を構築中

ソフトウェア実装の形式検証

- 得られた成果（一部）
  - メモリ管理ルーチン (C) の正当性の検査
  - 暗号処理ルーチン（アセンブリ）の安全性
    - アセンブリで実装されたある暗号疑似乱数の実装が、暗号的安全性 (乱数の予測不能性) を満たすことを直接的に証明
      - 多倍長演算などの定式化や、暗号学的な論理をすべて含む
  - プロトコル実装 (C言語) の仕様適合性の検証

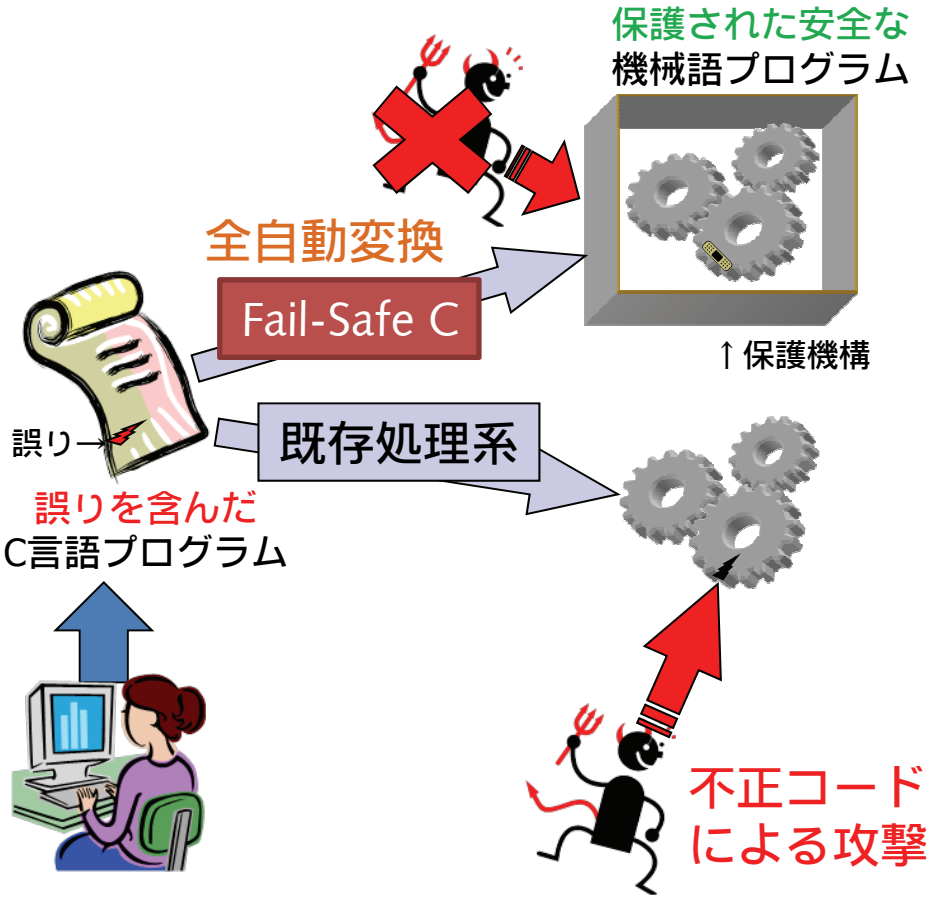
3. 誤動作を抑える技術



- 万が一開発時に誤りが残った場合も
  その影響を抑える技術
  - システムの部分的な形式検証などとの組み合わせも
- 開発している技術の一例
  - C言語プログラムの実行時安全性検査
  - 仮想化モニタを応用した実行監視技術

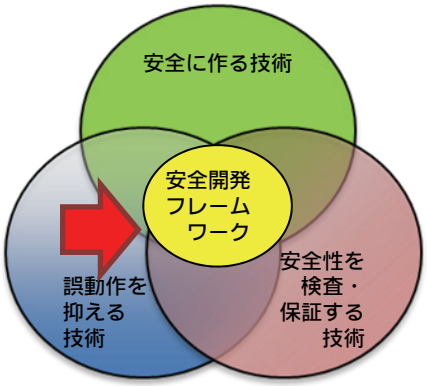
安全なC言語コンパイラ

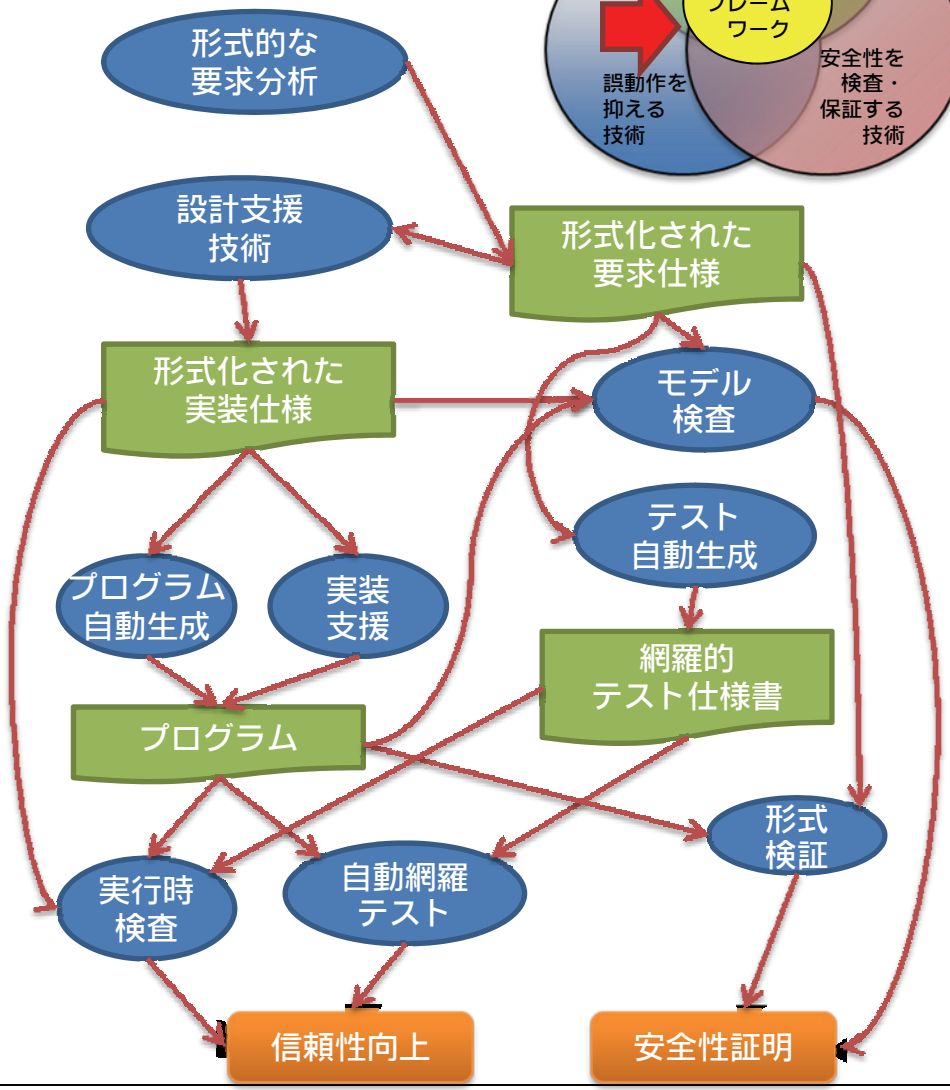
- C 言語プログラムに“自動的”に安全性保護を追加する言語処理系
  - JIS 規格に完全対応
  - 数多くの実用ソフトが動作可能
    - メールサーバなど多数
  - ウィルスなどの不正コードの実行を確実に防止する
    - ウィルスの侵入経路であるメモリ破壊が発生する直前に確実にプログラムを停止させる



4. 検査技術の統合

- これらの技術を
  自由に組み合わせられる
  統合プロセスを目指す
  - 与えられたコストに応じて
  - 必要な安全性に応じて
  - 必要な認証取得レベルを
    目指して
  - 部分的な検証
  - 部分ごとに
    異なった手法を適用
  - 開発開始後の
    適用技術の変更





主な現有技術の位置づけ