複雑さと信頼
ソフトウェアと数学の機械による検証

ジャック・ガリグ
Jacques Garrigue
garrigue@math.nagoya-u.ac.jp

名古屋大学多元数理科学研究科

2018 年 10 月 4 日

複雑さと信頼

ソフトウェアと数学の複雑化

- 機械の論理を司るソフトウェアがどんどん複雑になっていく
  - 飛行機や銀行のソフトウェアは数百万行におよぶ
  - スマートフォンの計算能力はパソコンと変わらない, 少し前のスパコンなみ
- 数学でも, 証明が数百ページにおよぶような定理が珍しくない
  - 群論における奇数位数定理 (Feit-Thompson 定理)
  - 望月の ABC 予想の証明
  - Hales のケプラー予想の証明

ソフトウェアと数学の信頼性

- 社会基盤となっている多くのソフトウェアには致命的なバグがある
  - 昨年の航空予約管理システム障害で世界中に影響
  - スマートフォンやパソコンの勝手な再起動
  - 下手すると, 命を危険にさらすこともある
- 数学の証明もバグから免れない
  - 今までのリーマン予想の証明が全て間違っていた
    先月の Michael Atiyah の証明は大丈夫なのか
  - ABC 予想の証明が中々信頼されない
  - 証明した本人が信頼しない場合もある (Hales, Voevodsky)

機械が信頼性を取り戻せるか

- 正しさは論理規則で表現される
  - 解釈にあいまいさを残さない
- 機械は論理規則を間違えない[1]
  - 規則はそれほど大くない
  - 迷うことはない
  - 人間はよく間違える
- しかし, 示せることに限度がある
  - 停止判定問題の判定不能性 (Turing, 1936)
  - 不完全性定理 (Gödel, 1931)

[1]正しくプログラムされたら

信頼性のための技術

強い型システム　プログラムの書き方を制限することで, ソフトウェアの整合性の問題の多くを解決できる.

静的解析　プログラムを自動的に解釈し, 様々な性質を保証する. 簡単な場合では停止性も証明できる.

プログラムの証明と定理証明支援系　原理的にはどんな性質でも検証できる[2]. ただし, 証明を構築するのに人間の助けが必要.

[2]不完全性定理により, 自分自身の無矛盾性は証明できない

インテル Pentium FDIV バグ

- 1994 年に発生
- 演算装置 (パソコンの CPU) のバグ
- 障害：特定の値に対して割り算の結果を間違える
- 原因：演算装置内のデータの間違い
- 損失額：500 億円 (バグのある CPU の多くを交換した)

FDIV バグを防ぐ

- このバグを機に, ハードウェアの検証の研究が増えた
- その後に出た AMD 社の K5 CPU[3] では浮動小数点演算を定理証明支援系 ACL2 で証明した
- 証明は数学の公理に基いているので, 不備は必ず発見される
- それ以降も多くの CPU 機能が証明されている

[3] Pentium の競合品

定理証明支援系

- 定理の証明の確認を行うためのツール
- プログラムの性質も証明の対象として扱える
- 定理の文を書き, それを証明する
- 一定の自動化があるが, 判定不能性より任意の定理が証明できるわけではない
- ACL2 の場合, 定理や補題が自動で証明されるが, そのために推論に必要な補題の文を与えなければならない
- Coq や Isabelle の場合, 証明を書くための言語が別に用意されている
- HOL Light などでは実装言語を証明にも使う

アリアン5ロケットの打ち上げ失敗

- 1996年6月4日に, 打ち上げの37秒後に発生
- 障害：軌道から外れ, 自動爆発した
- 原因：大きな整数値を範囲の狭い変数に入れようして, 回復できないエラーを起こした.
- 損失額：400億円

アリアン5障害の予防

- 本来, 範囲の狭い変数に入れるときは溢れないような処理を入れないといけない
- アリアンに使われたプログラミング言語 ADA では, 実行時にエラーを起こし, 忘れられることを防ぐつもりだった
- しかし, プログラムのテストはアリアン 4 のときに行われたため, アリアン 5 が飛んだときに初めて溢れた
- もしも**強い型システム**を使っていれば, プログラム作成時に不整合が発見されるはずだった

強い型システム

- プログラムに表われる全ての値と変数を**型**で分類
- エラーや不整合を起こすような自動変換を禁止
- プログラムの実行中に型の不整合が起こらないことが理論的に証明されている：**型健全性**
- 瞬時の自動検証が可能
- 言語処理系の一部なので, プログラム修正時にも適用される

マーズ・クライメイト探査機

- 1999 年 9 月 23 日
  火星の軌道に乗ろうとして発生
- 障害：噴射の時間を間違えて落下した
- 原因：事情があって, 探査機のデータを地上局のコンピュータで処理する必要が出たとき, 急遽修正されたプログラムの中でメートル法とヤード・ポンド法が混在し, 計算の間違い
- 損失額：150 億円
- 予防法：これも**強い型システム**で予防できる

ウィンドウズのブルースクリーン

- 1993 年からお馴染
- 障害：基本ソフトウェア・ウィンドウズの使用中に表われる青い画面. 対応は再起動のみ.
- 原因：様々な原因が存在するが, 多くの場合ではウィンドウズの機能を拡張するドライバーのバグが原因. 無限ループや不正メモリアクセスなど.
- 損失額：評価されたことはないと思われるが, 多大な時間とデータが長年失われて来た

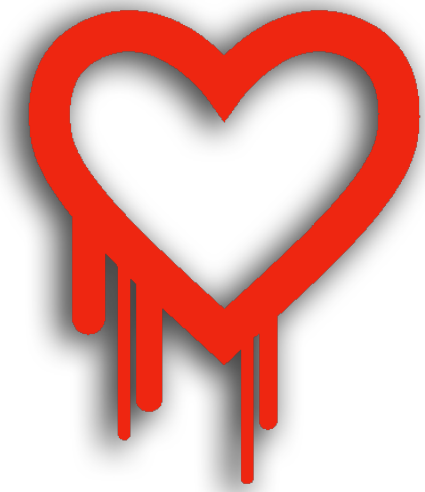
ブルースクリーンは何故減った

- ドライバーはマイクロソフト社の外で開発されることが多いので, 社内だけでは対応しきれない
- 多くの対応策が試みられた
- 決定打は**静的解析**によるドライバーのコードの検証
- SLAM というツールが開発者に与えられ, ドライバーの様々な性質を自動で検証
- SLAM で検証できなかったものはウィンドウズでの使用を禁止した

静的解析

- プログラムの様々な自動検証方法の総称
- 時間がかかるので, 定期的にチェックする
- 検証空間を有限分割し, 網羅的な検証を行う
- 停止性が証明できたりするが, 当然ながら完全な方法ではない
- **抽象解釈**はその一例
  - 各データ型を有限集合におきかえ, 忠実な計算規則を定義する
  - その計算規則で危険な状態になることがなければ, プログラムの安全性が証明される
  - 解釈の性質より, 停止しないプログラムも有限時間で検証できる

ハートブリード

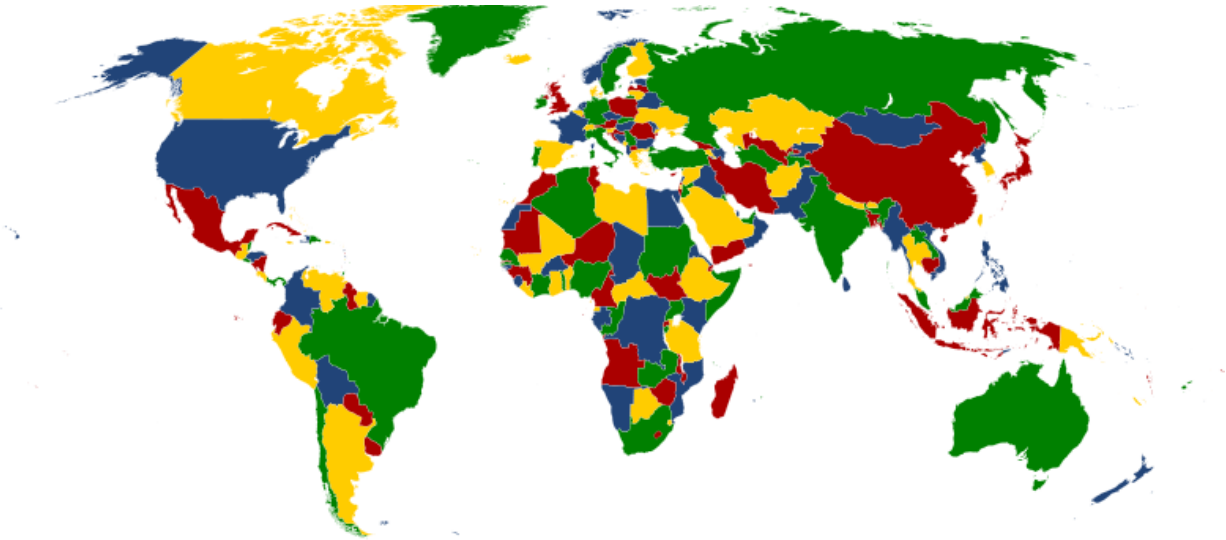


- 1994 年に発生
- 障害：多くのウェブブラウザやサーバーで使われる OpenSSL というソフトウェアの TLS 機能のバグによって, ユーザーのデータが漏れる
- 原因：メモリアクセスを十分に防御していなかったため, 受け取った要請によって本来返すべきでないデータを返していた
- 損失額：フリーソフトウェアのため, 損害賠償が行われなかったので, 分からない

ハートブリードの予防

- 通信の暗号化の安全性は今やインターネットに必要不可欠
- しかし, OpenSSL はプログラミング言語 C で書かれ, 絶対な安全性は望めない
- そのために, 新しいプログラミング言語での TLS の実装が試みられた
- マイクロソフト社での F*による実装
  - 暗号化を含めて完全に検証された TLS の実装
  - プログラミング言語 F*は強い型システムと定理証明支援系の要素を組み合わせている

四色定理



「任意の平面地図が四色のみで描ける」

- 1852 年に予想され, 1976 年に証明された.
- 約 2000 個の場合をチェックするプログラムが証明に含まれる
- 2004 年に Gonthier が Coq で定理と各場合をチェックするプログラムを証明した

奇数位数定理

「すべての奇数位数の有限群が可解群である」

- 1962 年に Feit と Thompson によって証明
- 代数学の大部分を利用して数百ページに及ぶ
- 純粋に数学的な証明でありながら, 全ての詳細を一人の数学者が確認するのが困難
- 2007〜2012 年に Gonthier をリーダーとしたグループが Coq で証明した
- そのために新しい証明言語と代数学のライブラリを開発している

ケプラー予想

同じ大きさの球を容器に詰め込む場合，面心立方配置および六方最密充填配置が最高密度に達している

- 17 世紀に Kepler が予想した
- 1998 年に Hales によって証明されたが、四色定理と同様、証明の一部がプログラムによる各場合のチェックを含んでいる
- Hales 自身が機械での証明をリードし、2014 年に HOL Light と Isabelle を使って証明した

判定不能問題

- Turing の計算可能性論では, 人間が計算するという過程が形式化された
- 性質 p について, 各対象で成り立つかどうかを計算するプログラムがあれば, p が**判定可能**

$$\begin{array}{rcl} p : \mathbf{N} & \to & \{\text{真}, \text{偽}\} \\ x \in \mathbf{N} & \mapsto & p(x) = \text{真 または } p(x) = \text{偽} \end{array}$$

- しかし, 多くの性質が判定不能である
- 最も有名な例は停止問題：十分な表現力を持つプログラミング言語において, 各プログラムが停止するかどうかを判定するプログラムが存在しない

停止問題が判定不能

$H(\overline{P}, X)$ がプログラム P の文と入力 X を受け，
$P(X)$ が停止するかどうかを判定できるとする

- $H'(\overline{P}) = H(\overline{P}, \overline{P})$ は $P(\overline{P})$ が停止するかどうか

$$H^*(\overline{P}) = \begin{cases} YES & H'(\overline{P}) = NO \text{ のとき} \\ \text{無限ループ} & H'(\overline{P}) = YES \text{ のとき} \end{cases}$$

- $H^*(\overline{H^*}) = YES$ ならば，$H'(\overline{H^*}) = NO$，すなわち $H^*(\overline{H^*})$ が止まらないので矛盾
- $H^*(\overline{H^*})$ が止まらなければ，$H'(\overline{H^*}) = YES$，すなわち $H^*(\overline{H^*})$ が止まるので矛盾

どちらまり得ないので，H が元々存在しない

不完全性定理

ゲーデルは二つの不完全性定理を証明している

1. (ある程度大きい) 形式体系では,
   「肯定も否定もできない命題が存在する」
   コンピュータでの証明にはそれほど影響するものではない. そもそも, 背理法を認めない直観主義論理を使うなら, そういう命題の存在はむしろ自然である.
2. (もう少し大きい) 形式体系では,
   「その体系自身の無矛盾性が証明できない」
   本当の限界になる. できれば定理証明支援系自身の正しさを証明したいが, この定理よりそれは不可能である.

論理学と経験

論理学が教えてくれること

- 論理的に動く機械だけで, 証明したいこと全てが証明できるわけではない

経験が教えてくれること

- いくら丁寧に書いたプログラム/証明でも, 大きくなれば人間が間違える
- 機械は大きさと関係なく, 正しさを調べることができる
- 安全なプログラム/証明を作るには, やはり人間と機械の協力が必要