

動的解析

NASA から学ぶ 超高信頼ソフトウェア技術

YUTAKA YK KATOH

2ND SECTION 1ST DEPARTMENT

ADVANCED PLATFORM DEVELOPMENT CENTER

RICOH COMPANY LIMITED

突然ですがここで・・・

歴史的に有名なバグを振り返ってみましょう

Source:

History's Worst Software Bugs – Wired News (Nov. 08, 2005)

<http://wired.com/news/technology/bugs/0,2924,69355,00.html>

<http://wired.com/news/technology/bugs/0,2924,69355-2,00.html>

(現在は消失、Internet Archive 及び翻訳記事参照)

火星探査機「マリナー1号」(1962年)

マリナー1号は打ち上げ時に予定のコースを外れたが、これは飛行ソフトウェアのバグが原因だった。地上の管制センターは大西洋上でロケットを破壊した。事後調査により、鉛筆で紙に書かれた数式をコンピューターのコードに置き換えるときにミスが起き、これが原因でコンピューターが飛行コースの計算を誤ったことが判明した。

- 被害：ロケット指令破壊
- 原因：プログラム記述誤り

放射線治療装置「セラック25」(1985-87年)

複数の医療施設で放射線治療装置が誤作動し、過大な放射線を浴びた患者に死傷者が出た。セラック25は2種類の放射線—低エネルギーの電子ビーム(ベータ粒子)とX線—を照射できるよう、既存の設計に「改良」を加えた治療装置だった。セラック25では電子銃と患者の間に置かれた金属製のターゲットに高エネルギーの電子を打ち込み、X線を発生させていた。セラック25のもう1つの「改良」点は、旧モデル『セラック20』の電気機械式の安全保護装置をソフトウェア制御に置き換えたことだった。ソフトウェアの方が信頼性が高いとの考えに基づく判断だった。しかし、技術者たちも知らなかった事実があった—セラック20およびセラック25に使われたOSは、正式な訓練も受けていないプログラマーが1人で作成したもので、バグが非常にわかりにくい構成になっていたのだ。「競合状態」と呼ばれる判明しにくいバグが原因で、操作コマンドを素早く打ち込んだ場合、セラック25ではX線用の金属製ターゲットをきちんと配置しないまま高エネルギーの放射線を照射する設定が可能になっていた。これにより少なくとも5人が死亡し、他にも重傷者が出了た。

- 被害：死者5名
- 原因：データ競合

CPU 浮動小数点バグ (1993年)

米インテル社が大々的に売り出したPentiumチップが、特定の浮動小数点数の除算で誤りを引き起こした。たとえば、 $4195835.0 / 3145727.0$ を計算させると、正しい答えの1.33382ではなく1.33374となる。0.006%の違いだ。実際にこの問題の影響を受けるユーザーはごくわずかだったが、ユーザーへの対応から、同社にとって悪夢のような事態につながった。概算で300万～500万個の欠陥チップが流通していた状況で、インテル社は当初、高精度のチップが必要だと証明できる顧客のみをPentiumチップの交換対象とした。しかし、最終的にインテル社は態度を改め、不満を訴えるすべてのユーザーのチップ交換に応じた。この欠陥は結局、インテル社に約4億7500万ドルの損害を与えた。

- 被害：約4億7500万ドル
- 原因：ルックアップテーブル定義ミス

アリアン5 フライト501 (1996年)

欧州宇宙機関の開発したロケット、アリアン5には、『アリアン4』で使われていたコードが再利用されていた。しかし、アリアン5ではより強力なロケットエンジンを採用したことが引き金となり、ロケットに搭載された飛行コンピューター内の計算ルーチンにあったバグが問題を起こした。エラーは64ビットの浮動小数点数を16ビットの符号付き整数に変換するコードの中で起こった。アリアン5では加速度が大きいため、64ビット浮動小数点で表現される数がアリアン4のときよりも大きくなつてオーバーフローが起つり、最終的には飛行コンピューターがクラッシュしてしまつた。フライト501では、最初にバックアップ・コンピューターがクラッシュし、それから0.05秒後にメイン・コンピューターがクラッシュした。その結果、エンジンの出力が過剰になり、ロケットは打ち上げ40秒後に空中分解してしまつた。

- 被害：ロケット爆発
- 原因：浮動小数点オーバーフロー

パナマ国立ガン研究所 (2000年)

米マルチデータ・システムズ・インターナショナル社が製作した治療計画作成用ソフトウェアを使っていたパナマの国立ガン研究所で、放射線治療で照射する放射線量の計算を誤る一連の事故が起きた。マルチデータ社のソフトウェアでは、健康な組織を放射線から守るための「ブロック」と呼ばれる金属製のシールドの配置を、コンピューターの画面上に描いて決めるようになっていた。しかし、同社のソフトウェアではシールドが4個しか使えなかつたにもかかわらず、パナマ人の技師たちはこれを5個使いたいと考えた。技師たちは、真ん中に穴を持つ1個の大きなシールドとして、5個のシールドをまとめて表示させれば、ソフトウェアをだますことができることを発見した。だが、そうした配置にした場合、穴の描き方によってこのソフトウェアが返す計算結果が違ってくることにはまったく気づいていなかった。ある方向に向けて描くと正しい照射量が計算されるが、違う方向に描くと必要な照射量の最大2倍の量を推奨してきたのだ。少なくとも8人の患者が死亡し、さらに20人が過剰照射によって深刻な健康被害を受けたとみられている。技師たちは、コンピュータによる計算結果を手作業で再チェックする法的義務を負っていたため、殺人罪で起訴されることになった。

- 被害：死者8名
- 原因：安全設計漏れ

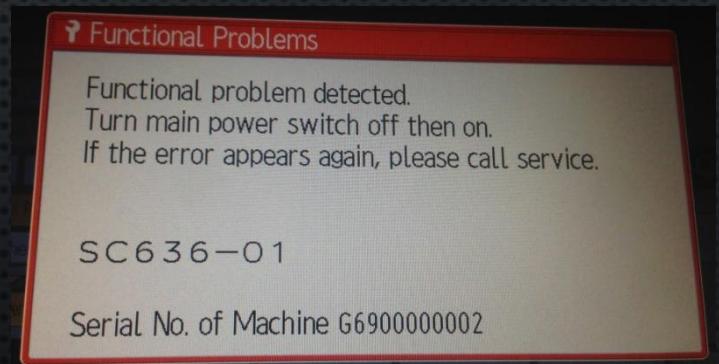
なにが言いたいかというと・・・

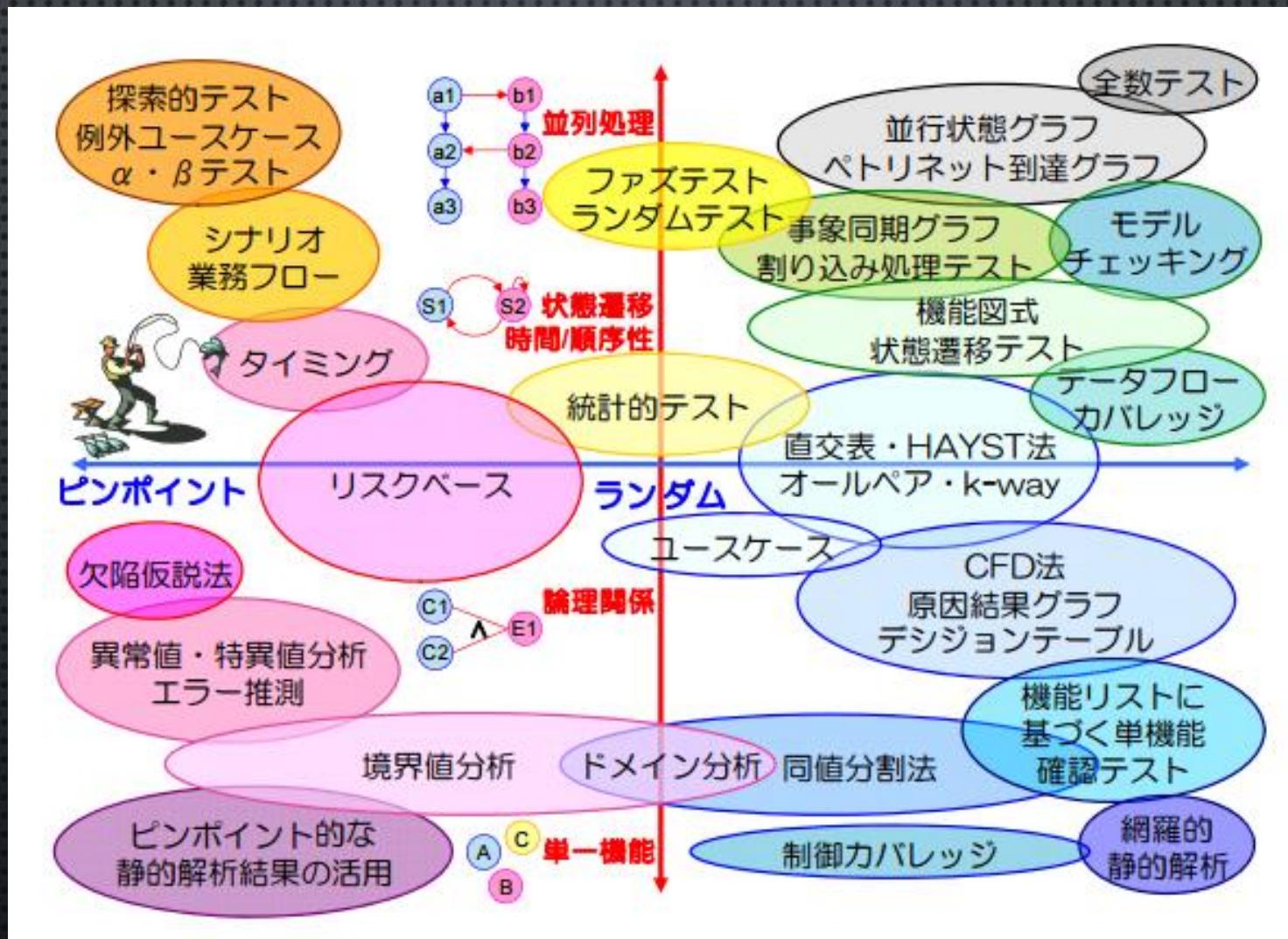
MFPのソフトエンジニアで良かった...



ではなくて・・・

バグをゼロにするにはどうすれば良いか
考えよう





テスト技法がいっぱいある理由

いかに少ない工数でバグを沢山見つけるか

逆に言うと、**テスト** (Testing) でバグが100%ないことを示すのは困難

バグが100%ないことを示すには

形式手法 (Formal Methods)

※かとうの勝手な分類

アプローチ

定理証明
(Theorem Proving)

モデル検査
(Model Checking)

実現形態

軽量形式記述

形式仕様記述

対話的証明システム

自動コード生成

コード(静的)解析

実行(動的)解析

先に言っておくと・・・

形式手法 (Formal Methods)

※かとうの勝手な分類

アプローチ

定理証明
(Theorem Proving)

モデル検査
(Model Checking)

今日の
DEMOは
ココ！

実現形態

軽量形式記述

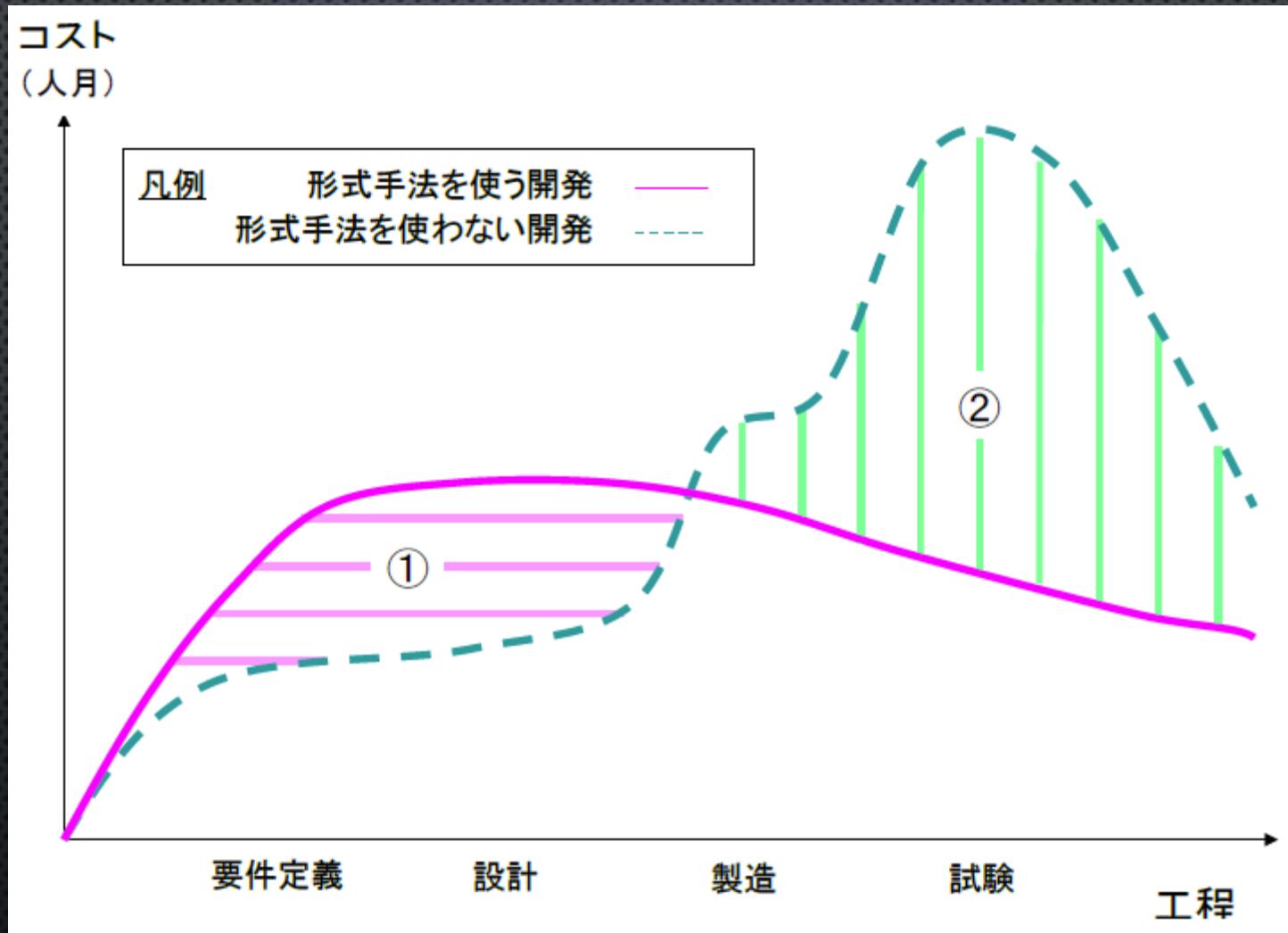
形式仕様記述

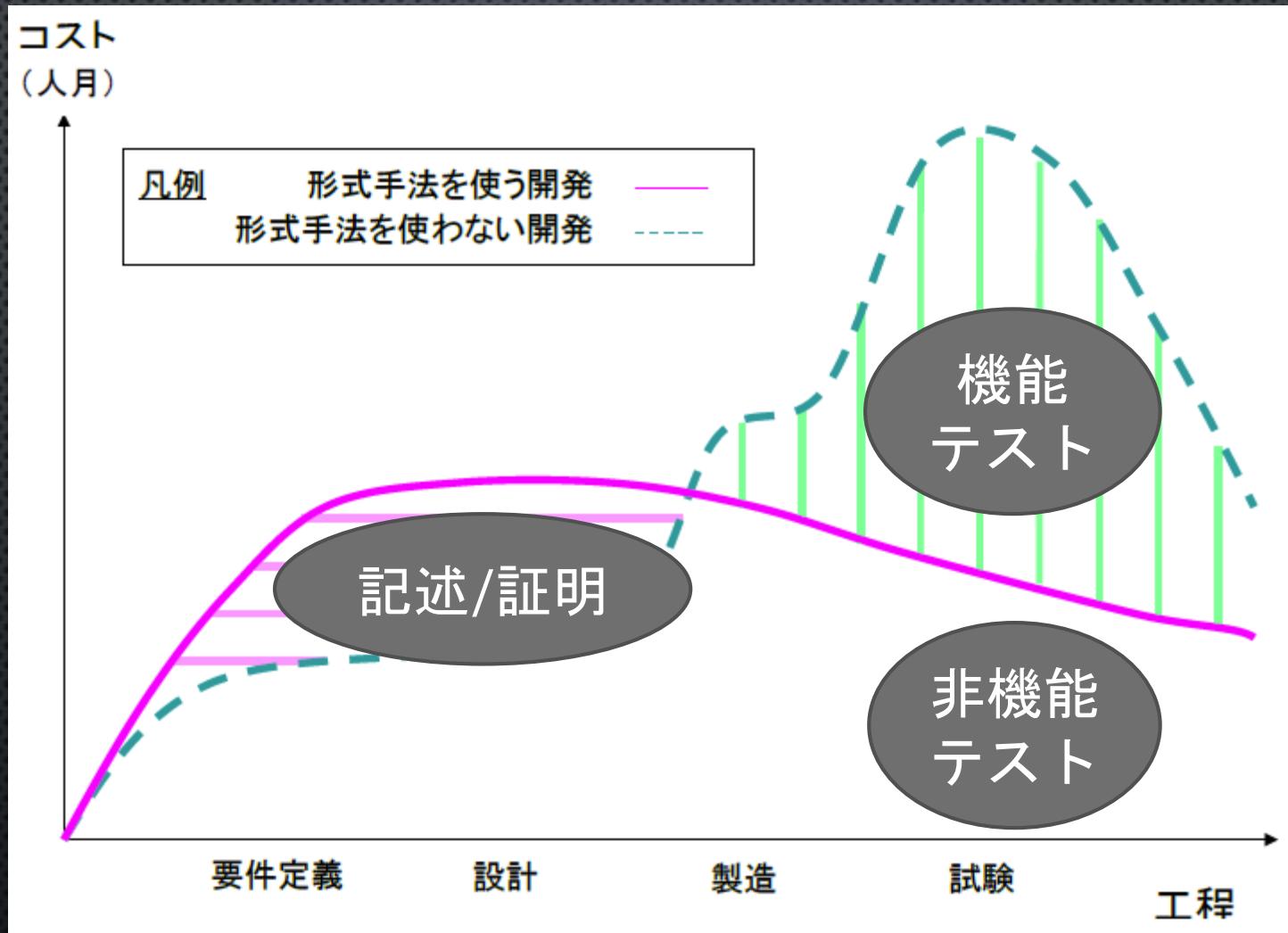
対話的証明システム

自動コード生成

コード(静的)解析

実行(動的)解析





定理証明とは

理論

- カリー・ハワード同型対応: プログラム構造と証明論が直接対応するという理論
- 相性の良い概念: **関数プログラミング** (純粋関数型)

実現形態

- COQ: 型付きラムダ計算による OCAML ベースの定理証明支援システム 
- AGDA: 構成的型理論に基づく定理証明支援システム、HASKELL ライクな文法
- CAFEOBJ: 代数的な仕様記述言語とその対話型証明システム、LISPで実装 




強みと辛み (ザックリ)

強いところ

- ・曖昧さや副作用を完全排除して設計できる
(証明駆動開発)
- ・コードが生成されている時点で定理は証明されているので、理論的には入力が無限にあっても(非決定的でも)構わない
→暗号アルゴリズムの開発では必要不可欠

辛いところ

- ・純粋関数型言語で記述する必要がある
- ・問題を数学的に抽象化して定理にするスキルがいる
→VDMを始めとした軽量手法の発展も
- ・自動定理証明できない場合は数学的帰納法などで証明ステップを書けるスキルがいる

モデル検査とは

理論

- プログラムの全状態モデルを抽出、網羅することで仕様を満たすことを証明
- 相性の良い概念: **契約による設計** (DESIGN BY CONTRACT)

実現形態

- SPIN: 独自言語 PROMELA でプログラミングし検査コードを自動生成するスタイル
- JPF: 特殊な VM 上で JAVA プログラムを動作させ状態抽出、全探索 (**動的解析**)

イマココ



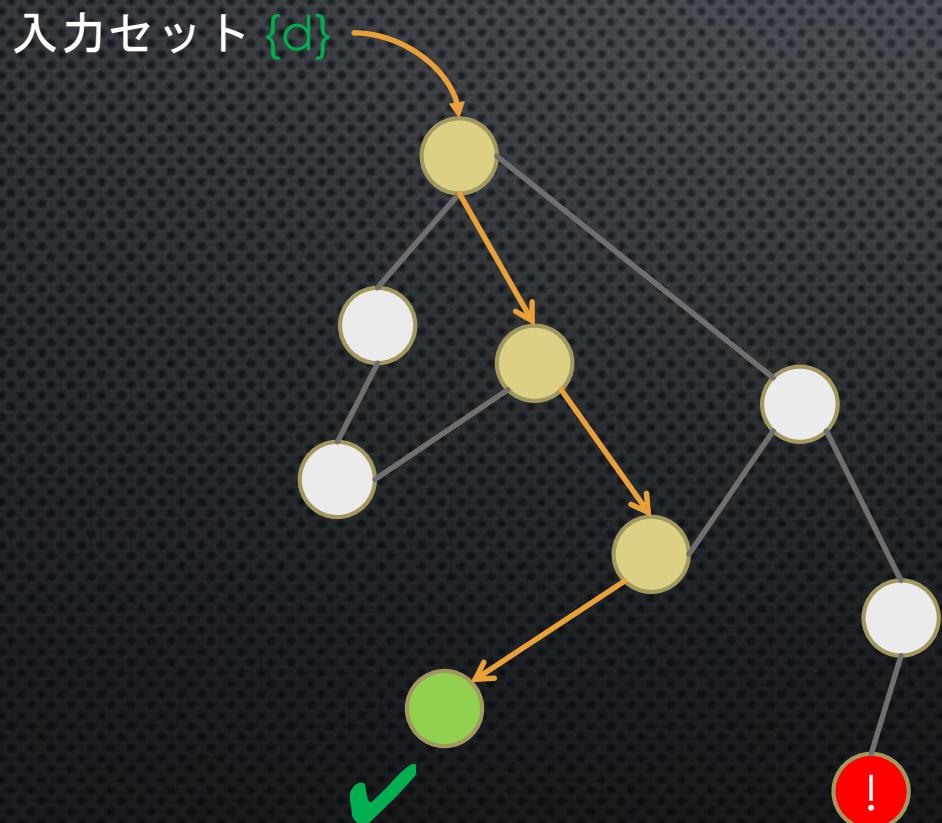
アプローチ

定理証明
(Theorem
Proving)

モデル検査
(Model
Checking)

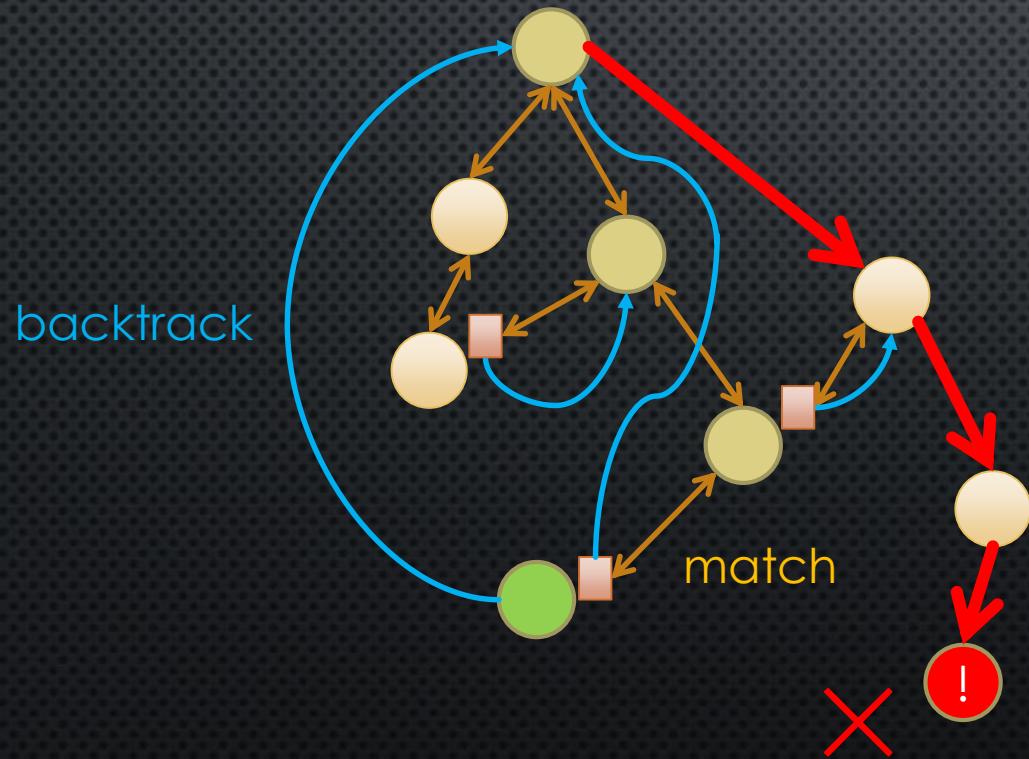


テスト (TESTING) を状態モデルで考えると・・・



- テスト (TESTING) では、1回の入力に対して**1度に1つのパス**しか検査されない
- バグを発見させるまでに多くの入力・試行が必要

状態モデルの探索とは



- 新しい状態が見つからなくなるまで全状態を探索する
- ある状態から一つ前の状態にバックトラックし、そこから別ルートの探索を進められる
- 同じ状態を複数作らないようにマッチングを行う

DEMO

- 用意するもの
 - JPF-CORE モジュール (今回は追加モジュールなし)
 - ECLIPSE (+ JPF プラグイン)
 - 検査対象のコード
- デモ内容
 - ハンドルされない例外の検出
 - データ競合の検出
 - デッドロックの検出

※追加モジュールの一例

- jpf.awt (GUI対応)
- net-iocache (TCP/IP 対応)
- jpf-statechart (UML 対応)

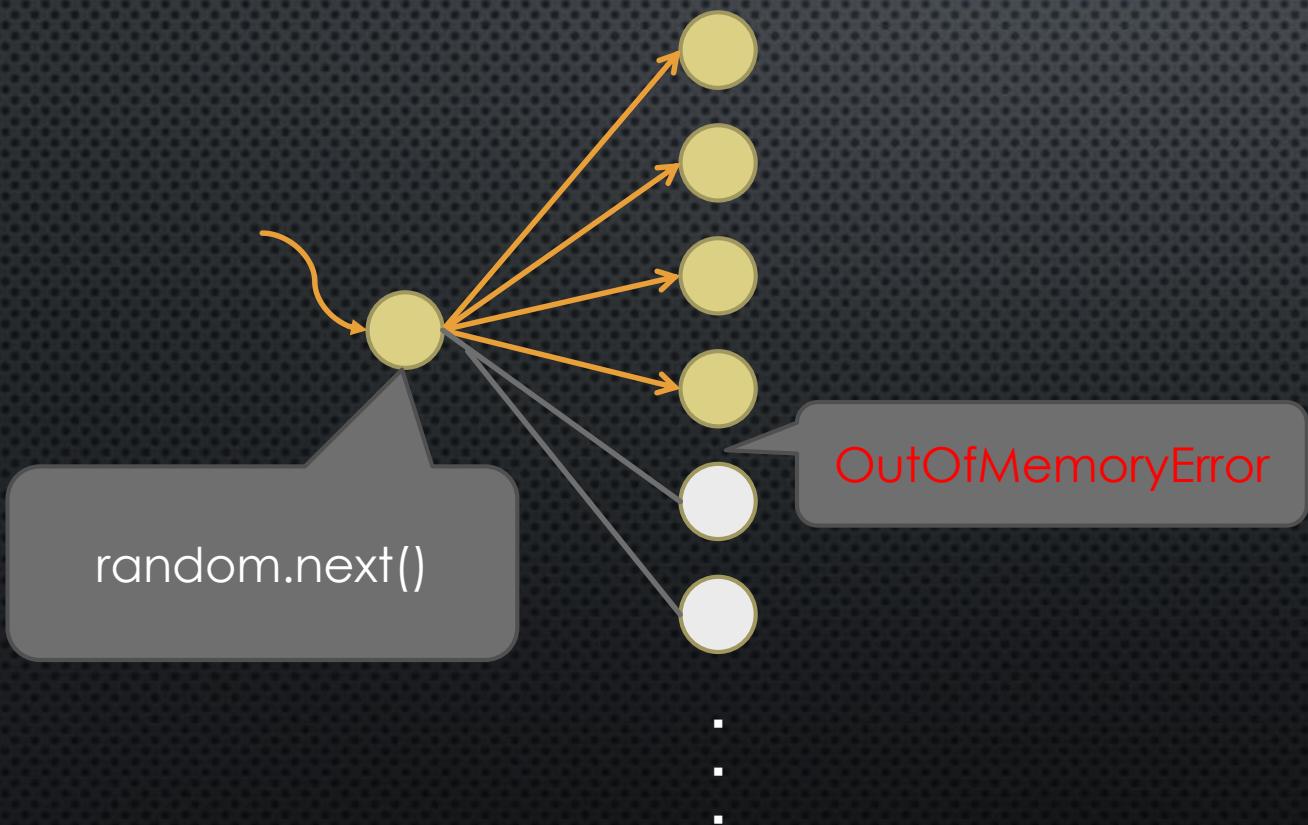
モデル検査の限界



状態爆発

(State Explosion)

状態爆発の例



非決定的状態遷移で
状態が量産される
(モデル検査の弱点)

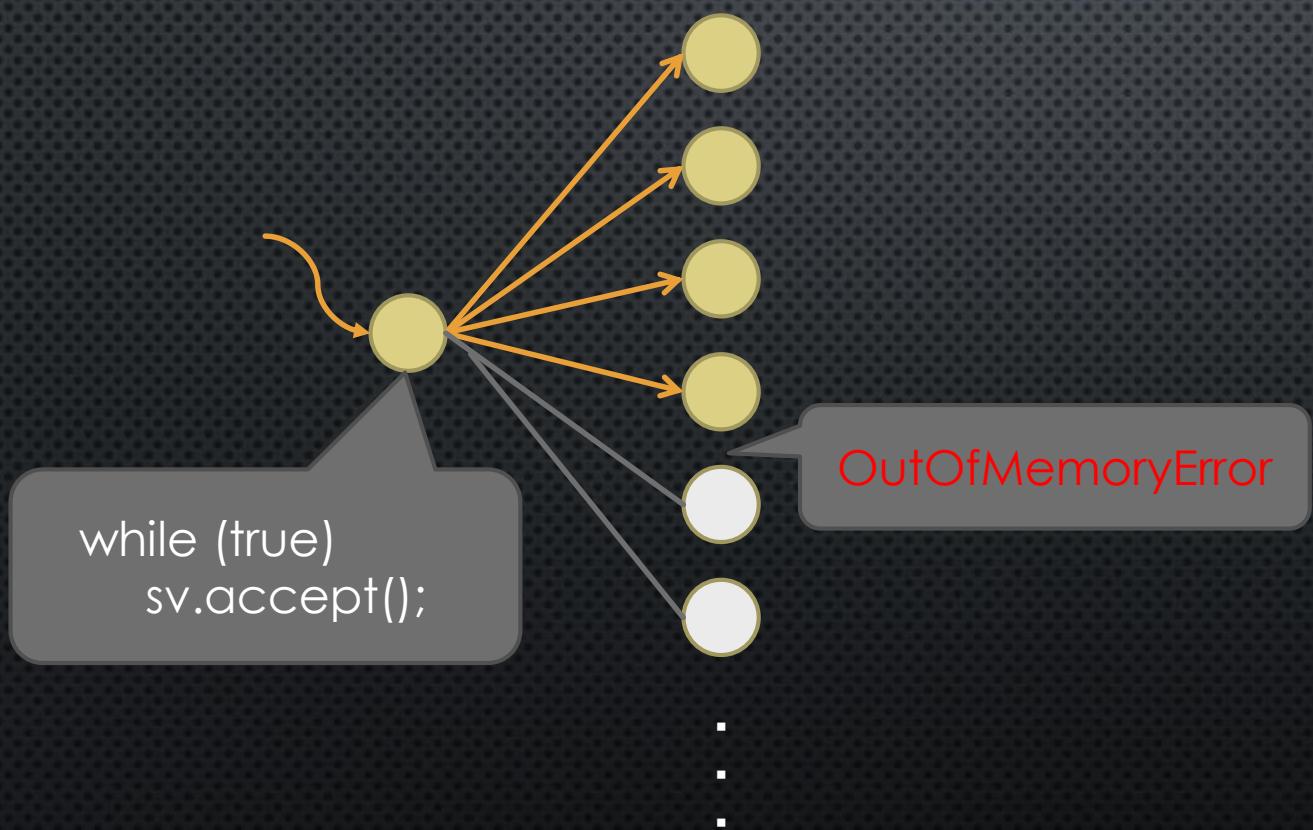


ランダムなら...

状態関係に関係する範囲に生
成値をコントロールする機構
で回避可能

ランダム以外は Choice Generator 化が必要
(つまりこれってモデル検査から外れる (Testing を持ち込む) 事を意味するよね)

状態爆発の例



そもそも無制約の1対多
関係を検査できない
(同時接続数 = 無限とか再現できない)

そこで制約を設けても...
並列度を上げると並行状態の組み合わせが膨大になる

まとめ

- ・バグが100%ないことを示す方法論は確立されている
- ・最適な手法はシステムの性質や要求によって様々
- ・適用したい手法の特性に合わせた設計・実装が求められる

形式手法には情報科学・情報工学の夢が詰まっている！

参考 (国内の主な成功事例)

- VDM++ (軽量アプローチ)
 - フェリカネットワークス - ICチップのファームウェア <HTTP://WWW.IPA.GO.JP/FILES/000005473.PDF>
- SPIN (モデル検査アプローチ)
 - トヨタ自動車 - 自動車制御ソフトウェア <HTTP://ID.NII.AC.JP/1001/00086009/>
 - 富士ゼロックス - 複合機ファームウェア <HTTP://JASST.JP/ARCHIVES/JASST08E/PDF/B4-2.PDF>
- JPF (モデル検査アプローチ)
 - 富士通 - 在庫管理ソフト <HTTP://WWW.FUJITSU.COM/DOWNLOADS/JP/ARCHIVE/IMGJP/JMAG/VOL60-6/PAPER04.PDF>

FeliCa Networks



FUJITSU