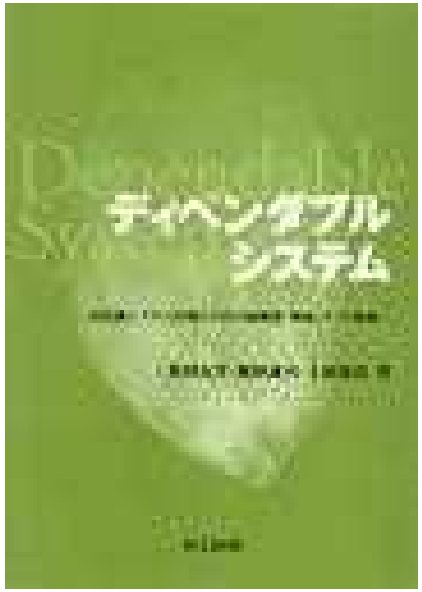
ソフトウェアテストの技術：
現場から研究へ，
研究から現場へ

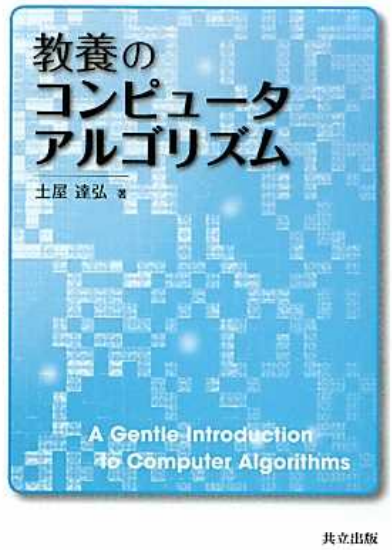
土屋達弘

大阪大学

自己紹介

- 専門
  - ディペンダブルシステム
    - 耐故障分散アルゴリズム
    - モデル検査 （自動検証）
    - ソフトウェアテスト

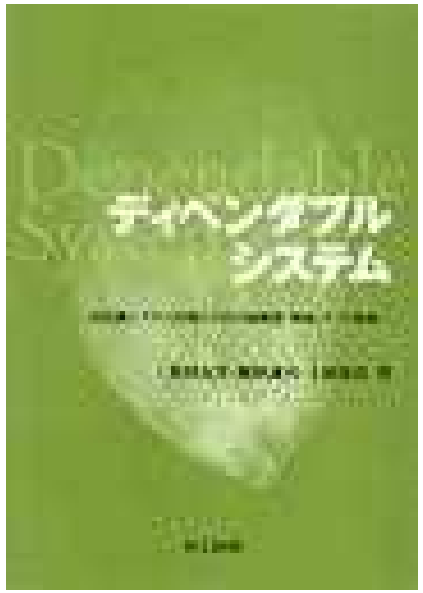




- 著書
  - 米田，梶原，土屋，ディペンダブルシステム，共立出版，2005.
  - 土屋，教養のコンピュータアルゴリズム，共立出版，2009.

ディペンダブルシステム

- ディペンダビリティ (Dependability)
  - 広い意味で，信頼性を表す用語
- ディペンダブルシステム
  - 信頼を置くことのできるシステム



JaSST '05 in OSAKA

Japan Symposium on Software Testing 2005 in OSAKA

ソフトウェアテストシンポジウム2005 in OSAKA

予稿集

http://www.swtest.jp/symposium.html

2005年7月15日（金）

会場：大阪国際会議場 グランキューブ大阪（大阪・福島）

主　催
ソフトウェアテストシンポジウム実行委員会
（ＴＥＦ：ソフトウェアテスト技術者交流会）

目次

フォールト，エラー，障害
Fault, Error, Failure

- Fault (故障，フォールト)
  - 構成要素の異常．障害，誤りの原因．

- Error (誤り，エラー)
  - システムの構成要素の異常状態．フォールトが顕在化したもの．障害の原因．

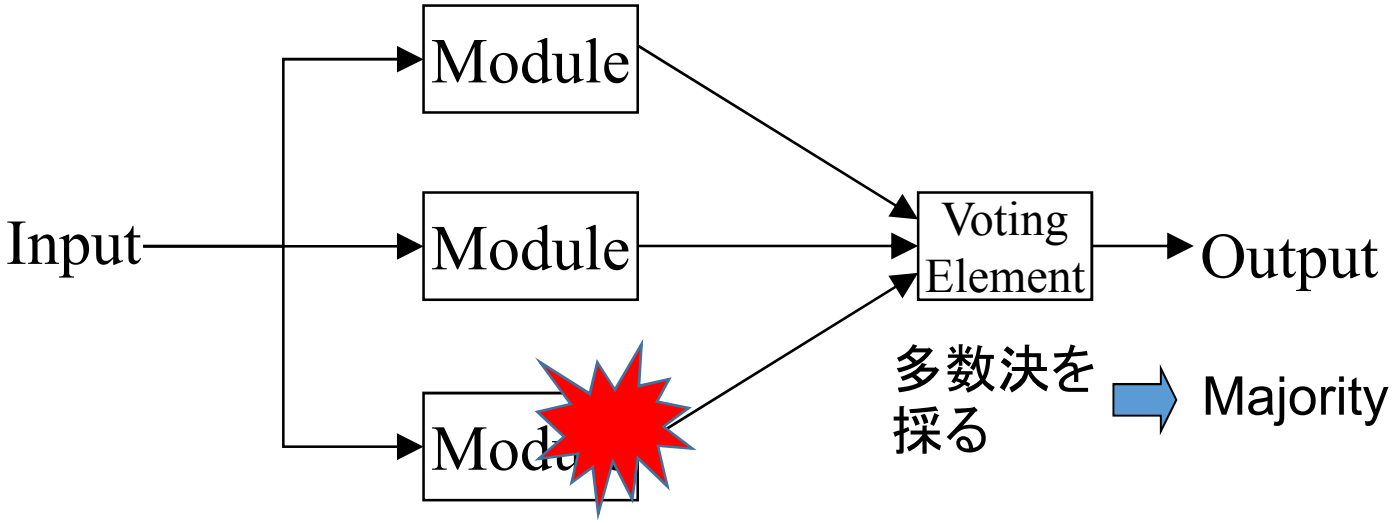
- Failure (障害)
  - システムが期待されるサービスを提供しなくなること．

ディペンダブルシステム実現のアプローチ

- フォールトアボイダンス (Fault Avoidance)
  - 障害の原因となるフォールトが発生しないようにするアプローチ
- フォールトトレランス (Fault Tolerance, 耐故障性)
  - フォールトが発生しても障害に至らないようにするアプローチ
- テスト，検証は？

フォールトトレランスの例
ハードウェア故障の場合

- 3重系多数決システム
  Triple Modular Redundancy (TMR)



- ソフトウェアに同じ考えは適用できるか？

取り上げるテスト・検証手法

- 組み合わせテスト
  （全ペアテスト，ペアワイズテスト）

- モデル検査/記号実行/コンコリックテスト

- Back-to-Backテスト

全ペアテスト（All-pairs testing, pair-wise testing)

例. コピー機のテスト

- テストしたい機能
  a. 印刷紙のサイズ
     - (1) B4, (2) A4, (3) B5
  b. カラー
     - (1) 白黒, (2) 赤黒, (3) カラー
  c. トレイ
     - (1)トレイ1, (2)トレイ2, (3)手差し
  d. 通信方法
     - (1) LAN, (2) USB, (3)Wifi

全ペアテスト用テストスイート

	サイズ	カラー	トレイ	通信
テスト1	B4	白黒	トレイ1	LAN
テスト2	A4	白黒	トレイ2	USB
テスト3	B5	白黒	手差し	Wifi
テスト4	B4	赤黒	手差し	USB
テスト5	A4	赤黒	トレイ1	Wifi
テスト6	B5	赤黒	トレイ2	LAN
テスト7	B4	カラー	トレイ2	Wifi
テスト8	A4	カラー	手差し	LAN
テスト9	B5	カラー	トレイ1	USB

テストケース9個で，すべてのペアをテスト!

背景 (JaSST'05 in Osaka)

- フォールト(不具合)の多くは，少数のパラメータ値の組み合わせによって顕在化できる
  - FTFI (failure-triggering fault interaction)
    - フォールトに関係しているパラメータ数

FTFI	エキスパートシステム				OS	組込み	Browser	Apache	DB
1	61	72	48	39	82	66	29	42	68
2	36	10	6	8	*	31	47	28	25
3	*na	*	*	*	*	2	19	19	5
4	*	*	*	*	*	1	2	7	2
5	*	*	*	*	*		2	0	
6	*	*	*	*	*		1	4	

R.D.Kuhn et al. "Software Fault Interactions and Implications for Software Testing," *IEEE Transactions on Software Engineering*, 30(6), June 2004.

全ペアテストの誕生

- 参考
  - Keizo Tatsumi, Combinatorial Testing in Japan, Workshop on Combinatorial Testing for Complex Computer Systems, 2013.
- 誕生, 黎明期
  - R. Mandl, 1985. Analogic Corporation
    - コンパイラのテスト
  - 佐藤，下川, 1984.
  - K. Tatsumi, 1987.

富士通

  - R. Brownlie et al. 1992. AT&T Bell laboratories

グリーディー手法

- AT＆Tによるグリーディー手法の研究
  - D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, The AETG System: An Approach to Testing Based on Combinatorial Design, IEEE Transactions On Software Engineering, 23(7), 1997.
- グリーディーアルゴリズム（貪欲法）
  - 良い部分解から選んでいくヒューティスティックなアルゴリズム

…

サイズ	カラー	トレイ	通信
B4	白黒	トレイ1	LAN
B4	赤黒	手差し	USB
A4	カラー	トレイ2	Wifi

サイズ	カラー	トレイ	通信
B4	白黒	トレイ1	LAN
B4	赤黒	手差し	USB
A4	カラー	トレイ2	Wifi
B5	白黒	トレイ2	USB

…

新たにカバーされたペア：(B5,白黒), (B5, トレイ2), (B5, USB), (白黒, トレイ2), (白黒, USB), (トレイ2, USB)

グリーディー手法

- 全ペアテストへの応用
  - まだカバーされていないペアを多くカバーするテストケースをその都度選択

```
while (カバーされていないペアが存在) {
        テストケースを一つ生成し追加
}
```

  - テストケース数は，パラメータに対して対数的(log)にしか増えない
- PICTでも採用
  - PICT: Microsoft社で開発された全ペアテスト用のツール
    - Pairwise.org
  - PictMasterのバックエンド

制約条件（禁則）

- テストケースが満たすべき条件
- 例.「B4 ⇒手差し」 AND 「手差し⇒USB」
- 無効なテストケースの例

B4	白黒	トレイ1	LAN

B5	白黒	手差し	Wifi

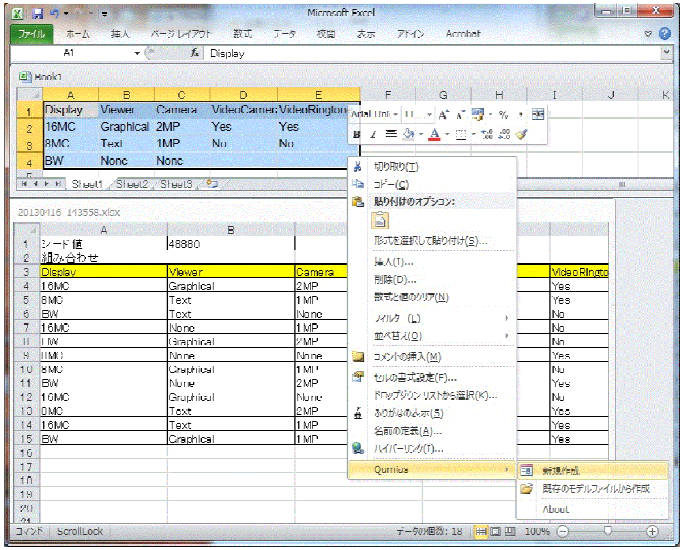
制約（禁則）処理の難しさ

- 例.「B4 ⇒手差し」 AND 「手差し⇒USB」
- テストできないペア！？
  - (A4，手差し)？
  - (B4，トレイ1)？
  - (B4，Wifi)？

- 個々の制約条件からは判断できない
  - システマティックな処理手法が必要
- PICTではどう処理しているか不明
  - 現場の知見：極端に処理が遅くなることがある

CIT-BACH

- 組み合わせテスト用テストケース生成ツール
- Qumiasのバックエンド
  - Qumias: Excelベースのテストケース生成ツール
- 名前の意味
  - Combinatorial Interaction Testing tool with a ***BDD***-Assisted Constraint Handler
- ウェブページ
  - http://www-ise4.ist.osaka-u.ac.jp/~t-tutiya/CIT/
  - http://sourceforge.jp/users/t-tutiya/

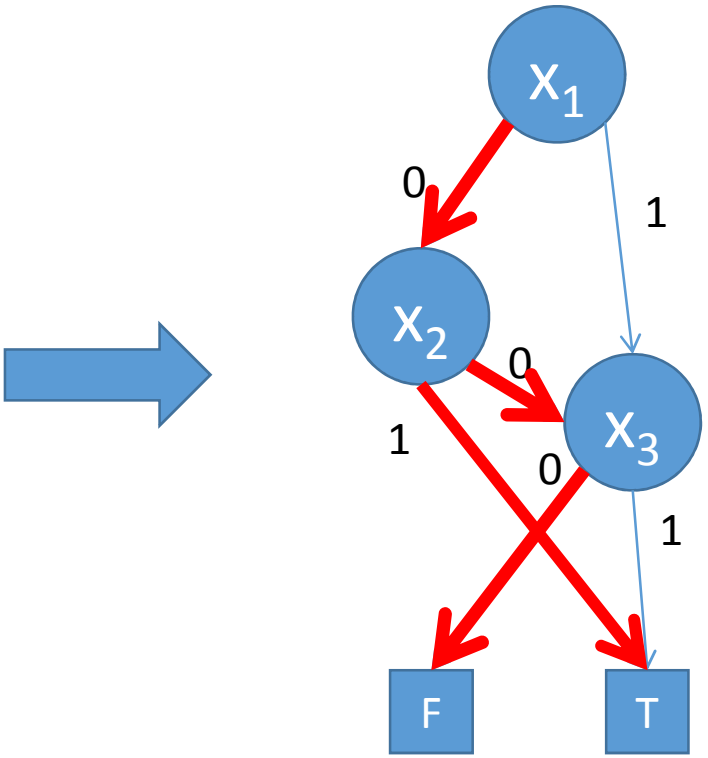


BDD (2分決定図)

- ブール関数を表すデータ構造
  - R. Bryant @CMU (1986)

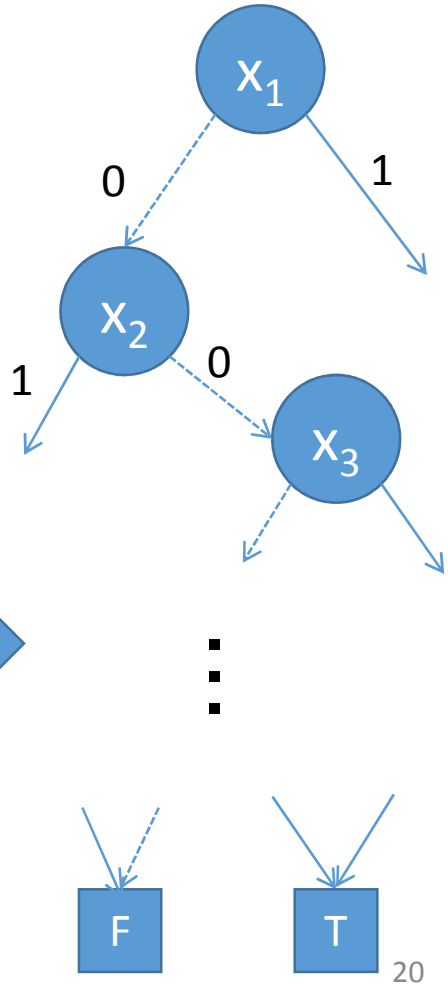
$f = (\neg x_1 \wedge x_2) \vee x_3$の真理値表

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



CIT-BACHでのBDDの利用

- 制約全体を最初にBDDで表現



- 8回枝をたどれば，有効かどうか判定できる

PICT vs. CIT-BACH

```
A: 1, 2, 3, 4
B: 1, 2, 3, 4
C: 1, 2, 3, 4
D: 1, 2, 3, 4
E: 1, 2, 3, 4
F: 1, 2, 3, 4
G: 1, 2, 3, 4
H: 1, 2, 3, 4

([A] = [B]) OR ([C] = [D])
OR ([E] = [F]) OR ([G] = [H]);
```

パラメータ

制約

PICT

```
A ( 1 2 3 4 )
B ( 1 2 3 4 )
C ( 1 2 3 4 )
D ( 1 2 3 4 )
E ( 1 2 3 4 )
F ( 1 2 3 4 )
G ( 1 2 3 4 )
H ( 1 2 3 4 )

(or (== [A] [B]) (== [C] [D])
  (== [E] [F]) (== [G] [H]))
```

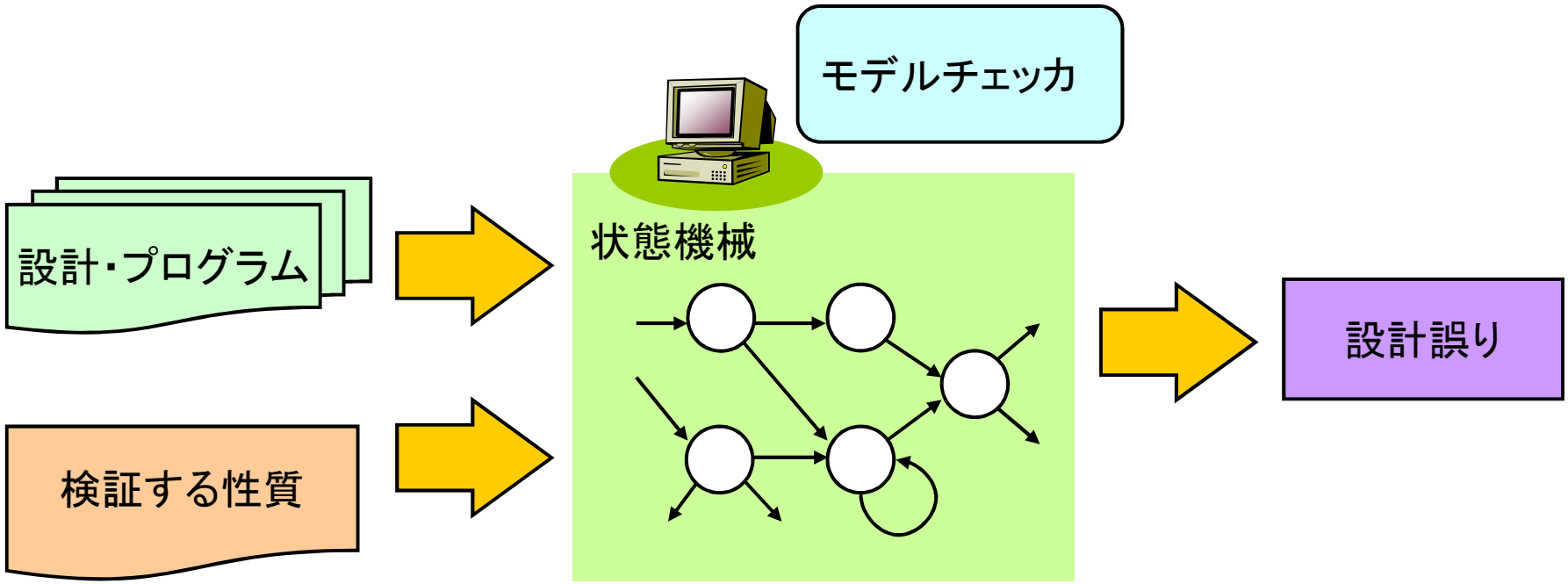
CIT-BACH

取り上げるテスト・検証手法

- 組み合わせテスト
  （全ペアテスト，ペアワイズテスト）
- モデル検査/コンコリックテスト
- Back-to-Backテスト

モデル検査

- 状態探索によってプログラムやシステムが，与えられた性質を満たすかを検証する手法



歴史と分類

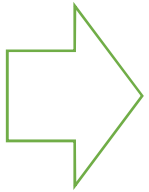
- 80年代初頭に登場
- 以降，様々な手法により劇的に扱える状態数が増加
  - 例．BDDの利用
  - 普通のPCでも億単位
- 実用レベルに到達

- 明示状態モデル検査：ツール例 SPIN
- 記号モデル検査: ツール例 SMV
- 充足可能性判定の利用
  - 有界モデル検査：ツール例 CBMC
    http://www.cprover.org/cbmc/

充足可能性判定を利用したモデル検査 (有界モデル検査)

- 充足可能性判定
  - 方程式に解があるかを判定
- プログラムは方程式で表現できる(こともある)

```
assume(x + y > 2);
if (b)
   x = x +1;
else
   x = x + 2;
y = y + x;
x = y + 1;
assert(x > 5);
```



$\wedge$ y0 + x0 > 2
$\wedge$
  $\vee$ b0 $\wedge$ x1 = x0 +1
  $\vee$ $\neg$b0 $\wedge$ x1 = x0
$\wedge$
  $\vee$ $\neg$b0 $\wedge$ x2 = x1 + 2
  $\vee$ b0 $\wedge$ x2 = x1
$\wedge$ y1 = y0 + x2
$\wedge$ x3 = y1 + 1
$\wedge$ $\neg$(x3 > 5)

解があれば,
assert違反の
動作が存在

SMTソルバ(Yices)への入力

$$
\begin{aligned}
&\wedge\ y0 + x0 > 2 \\
&\wedge \\
&\quad \vee\ b0 \wedge x1 = x0 + 1 \\
&\quad \vee\ \neg b0 \wedge x1 = x0 \\
&\wedge \\
&\quad \vee\ \neg b0 \wedge x2 = x1 + 2 \\
&\quad \vee\ b0 \wedge x2 = x1 \\
&\wedge\ y1 = y0 + x2 \\
&\wedge\ x3 = y1 + 1 \\
&\wedge\ \neg(x3 > 5)
\end{aligned}
$$

記号実行(symbolic execution)を実現

```
% cat sample.txt
(define b0::bool)
(define x0::int)
(define x1::int)
(define x2::int)
(define x3::int)
(define y0::int)
(define y1::int)
(assert
(and
  (> (+ y0 x0) 2)
  (or
     (and b0 (= x1 (+ x0 1)))
     (and (not b0) (= x1 x0))
  )
  (or
     (and (not b0) (= x2 (+ x1 2)))
     (and b0 (= x2 x1))
  )
  (= y1 (+ y0 x2))
  (= x3 (+ y1 1))
  (not (> x3 5))
)
)
(check)
% yices -e sample.txt
```

モデル検査の問題点

- 適用可能な規模が小さい
- 方程式で表現できないプログラムの存在
  - 外部関数の利用
  - 解析不可能な関数（例. ハッシュ）

```
int foo(int x, int y) {
  if (x == hash(y))
     return -1;
  return 0;
}
```

出典：P. Godefroid et al., IEEE Software, 25(5), 30-37, 2008.

ファジング

- 予測不可能な入力データを自動的に生成し，入力することで，脆弱性などのソフトウェアの不具合を検出するテスト手法
  - 商用・フリーとも多数のファザー（ツール）が存在
    - 一例．https://code.google.com/p/american-fuzzy-lop
    - IPA, ファジング活用の手引き, 2013.

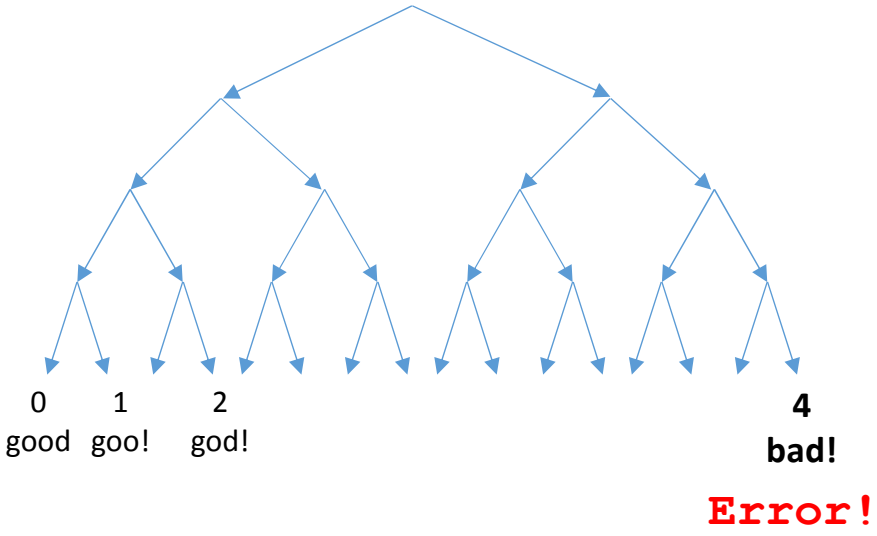
```
GET / HTTP/1.1
Host: www.ipa.go.jp
User-Agent: Mozilla/5.0 (Windows …
Accept: text/html, …
```

```
GET AAAAAAAAAAAAAAAA… HTTP/1.1
Host: www.ipa.go.jp
```

ファジング

- パスによってはほとんど実行することが不可能な場合がある

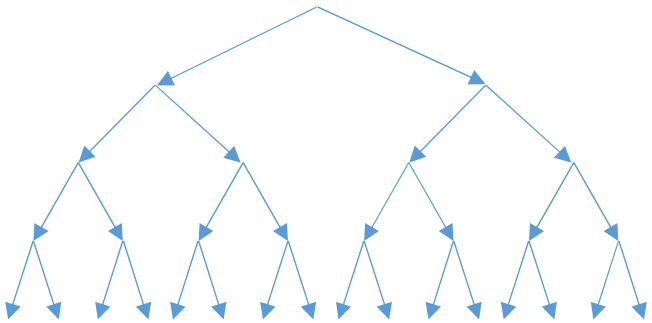
```
void top(char input[4]) {
  int cnt = 0;
  if (input[0]=='b') cnt++;
  if (input[1]=='a') cnt++;
  if (input[2]=='d') cnt++;
  if (input[3]=='!') cnt++;
  if (cnt>=3) abort(); //error
}
```



出典：P. Godefroid et al., IEEE Software, 25(5), 30-37, 2008.

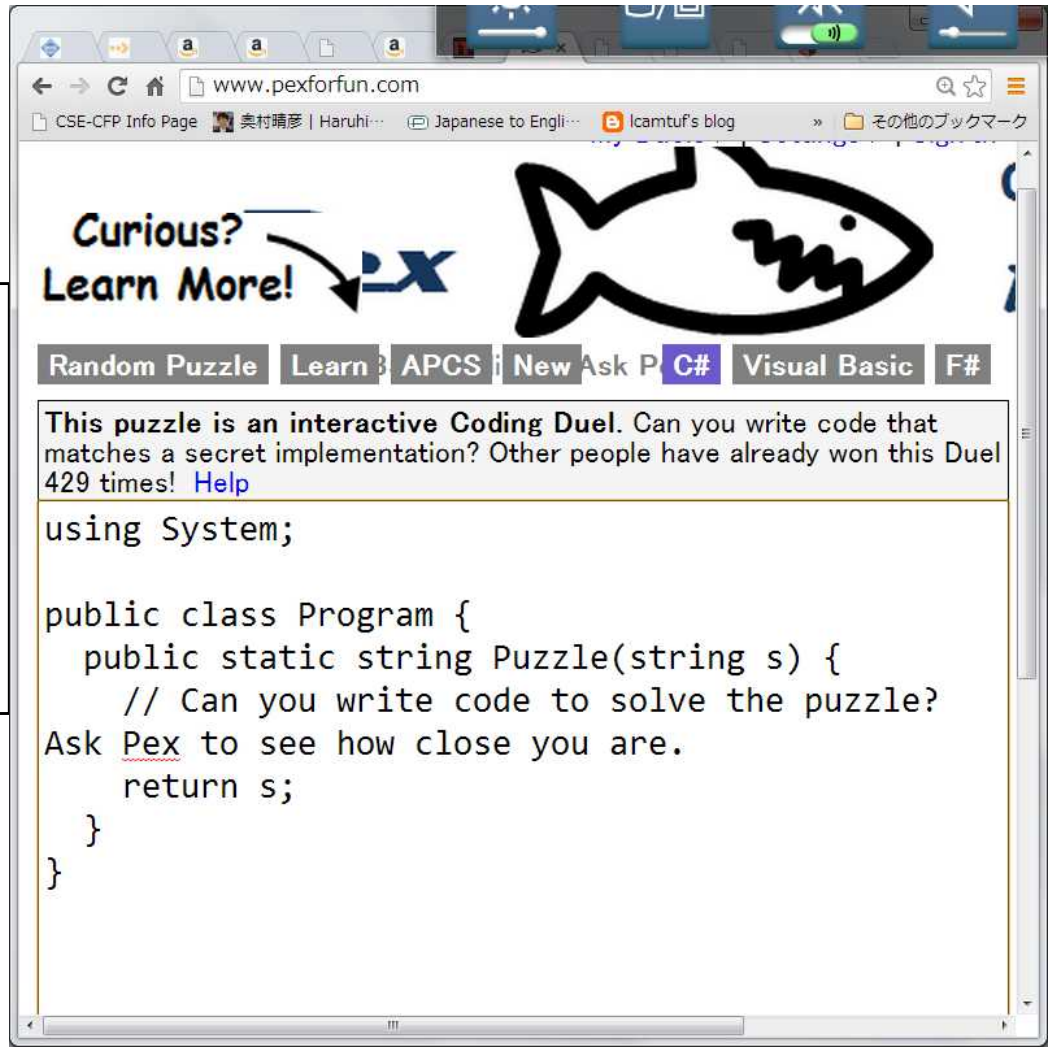
コンコリックテスティング

- コンコリック＝コンクリート + シンボリック
  - 具体的な入力によるプログラムの実行
  - 記号的なプログラム解析 (充足可能性判定)
- 分岐条件の真偽のすべての組合せの網羅を目指す
- ツール例
  - CUTE, jCUTE
  - CREST
  - PEX
    - Pex for fun www.pexforfun.com



Pex for fun

```
void check(string s) {
  string s1 = Program.Puzzle(s);
  string s2 = ProgramAns.Answer(s);
  assert (s1.CompareTo(s2)==0);
}
```



動作の様子

1. 適当な入力値で実行
2. 分岐条件に関する制約を収集
3. 制約の一部の否定をとる
4. すべての制約を満たす入力をSMTソルバで求め実行

```
int foo(int x, int y)
{
  if (x == hash(y))
     return -1;
  return 0;
}
```

1
```
foo(10, 20);
int foo(int x, int y)
{
  if (x == hash(y))
     return -1;
  return 0;
}
```

2
x = 10, y = 20, x != 327

3
x = 10, y = 20, x = 327

→解なし

3’
x != 10, y = 20, x = 327

→ x = 327, y = 20

4
```
foo(327, 20);
int foo(int x, int y)
{
  if (x == hash(y))
     return -1;
  return 0;
}
```

取り上げるテスト・検証手法

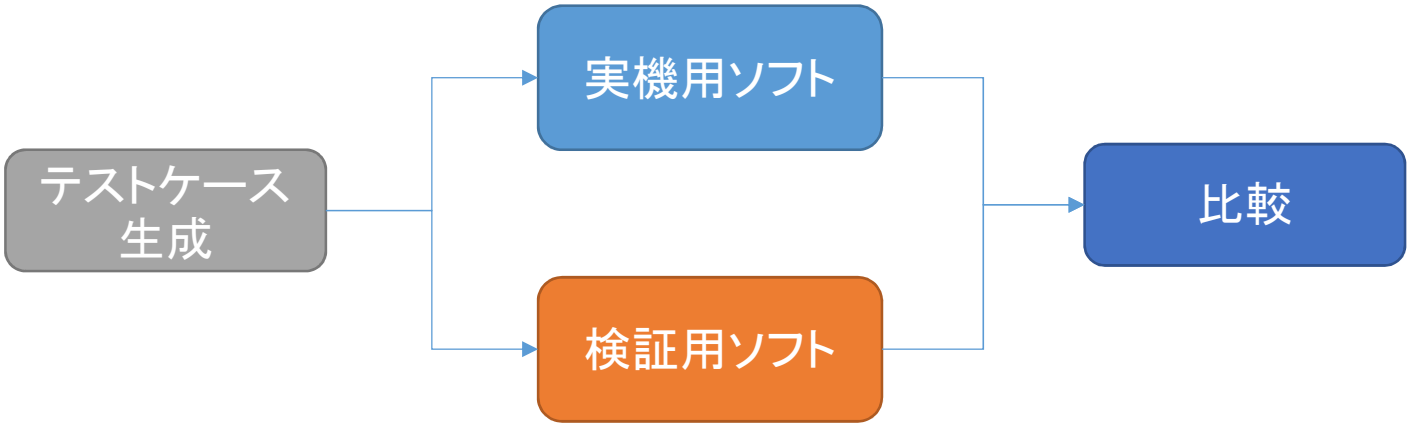
- 組み合わせテスト
  （全ペアテスト，ペアワイズテスト）

- モデル検査/記号実行/コンコリックテスト

- Back-to-Backテスト

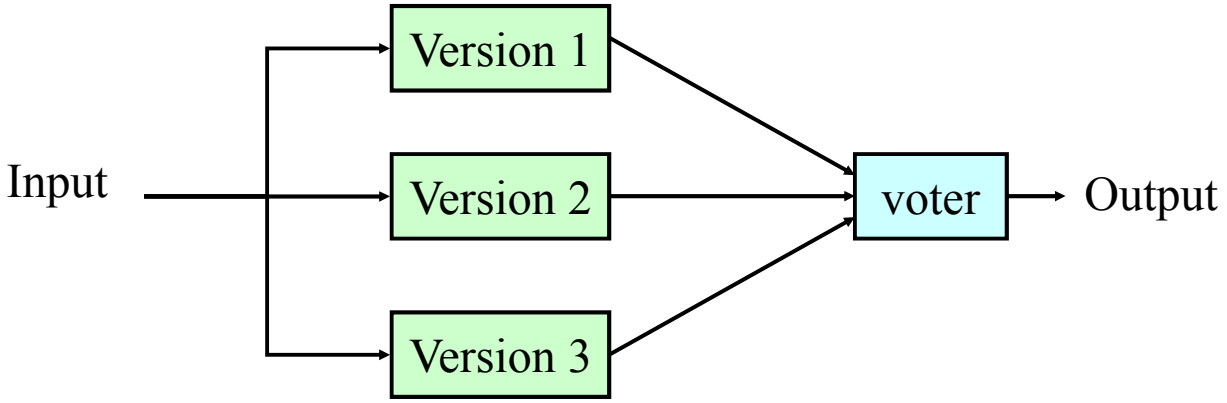
Back-to-Back テスト

- 同一目的のソフトウエアを二つ作り，同一処理の結果を比較することでテストを行う手法
- 応用例：自動改札機のテスト
  - 幡山五郎，JaSST'11 Kansai 基調講演
  - 時井，高木，上田，Omron Technics 46(2), 2005



Nバージョンプログラミング

- 同一仕様で異なる設計のソフトウェアを複数用意．処理の結果を比較，もしくは，多数決をとることで，設計フォールトに耐える方法
  - Chen and Avizienis @ UCLA (1977)



- 応用例：Airbus A320～A380

ISSTA2004(つづき)

- 4. 経験研究:
  - 開発中の継続テストの経験的評価
  - バグはどこに
- 5. テスト生成:
  - Java PathFinder を用
  - ループ代入フラグの
  - クラスの発展的テス
  - ツール:モデルに基
- 6. テスト2:
  - 限定された徹底的テ
  - 関数推定を用いた関
    テスト
  - テスト技法の解析的

2005/7/15

ISSTA2004(つづき)

- 7. モデルチェック1:
  - 設計レベルでの自動化された交換可能性解析
  - 演算の航空機交通量のモデルと検証
  - 時相論理式の一括最適化コンパイラ
- 8. プログラム解析2:
  - ソフトウェア動作の自動分類のための能動的学習
  - プログラム舵取り(?) によるマルチモードシステムの適用可能性改善
  - ツール:同期反応型プログラムのスライシングツール
- 9. プログラム解析3:
  - 原子性に対する純粋性拡大
  - 部分型を用いた高速制約解決
  - SABER: エラー減衰に基づくスマート解析

まとめ

- 理論, 基盤技術は「ある」
  問題は使い方
- 必要は発明の母