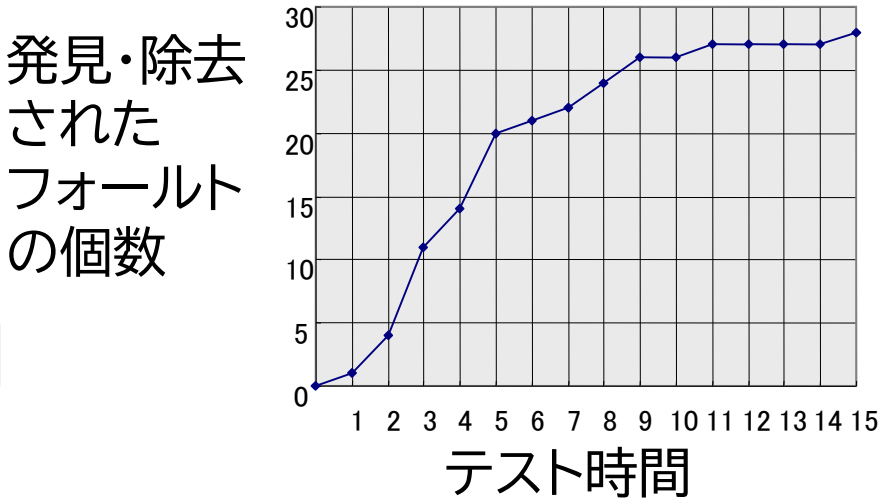
ソフトウェアの信頼性

1. 信頼度成長モデルを学習する
2. テスト設計を学習する
3. 検証手法を学習する
4. ソフトウェアフォールトトレランスを学習する

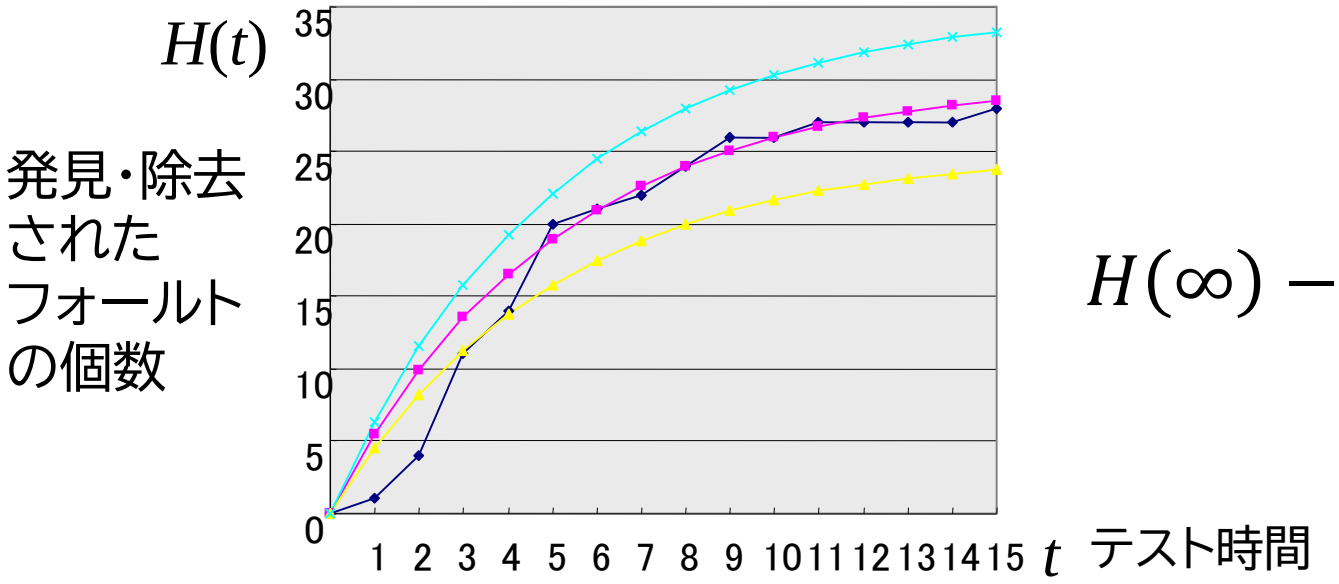
ソフトウェア信頼度成長モデル (Software reliability growth model, SRGM)

- テストの過程における，ソフトウェアの信頼度変化のモデル
  - 残存フォールト数の推定に用いられる
- 適用手順
  - モデルを選ぶ
  - 観測データによりパラメータを推定
  - 結果，残存フォールト数が推定できる
- 観測データ
  - 発見・除去されたフォールトとその時刻



例

- モデルを選ぶ
  - $H(t) = a(1 - e^{-bt})$
    - ◆時刻tまでに除去されたフォールトを表す
- パラメータを推定する
  - ツールの例. SRATS2017: https://swreliab.github.io/SRATS2017



$H(\infty) - H(t) =$ 残存フォールト数

Goel-Okumotoモデル (指数形SRGM)

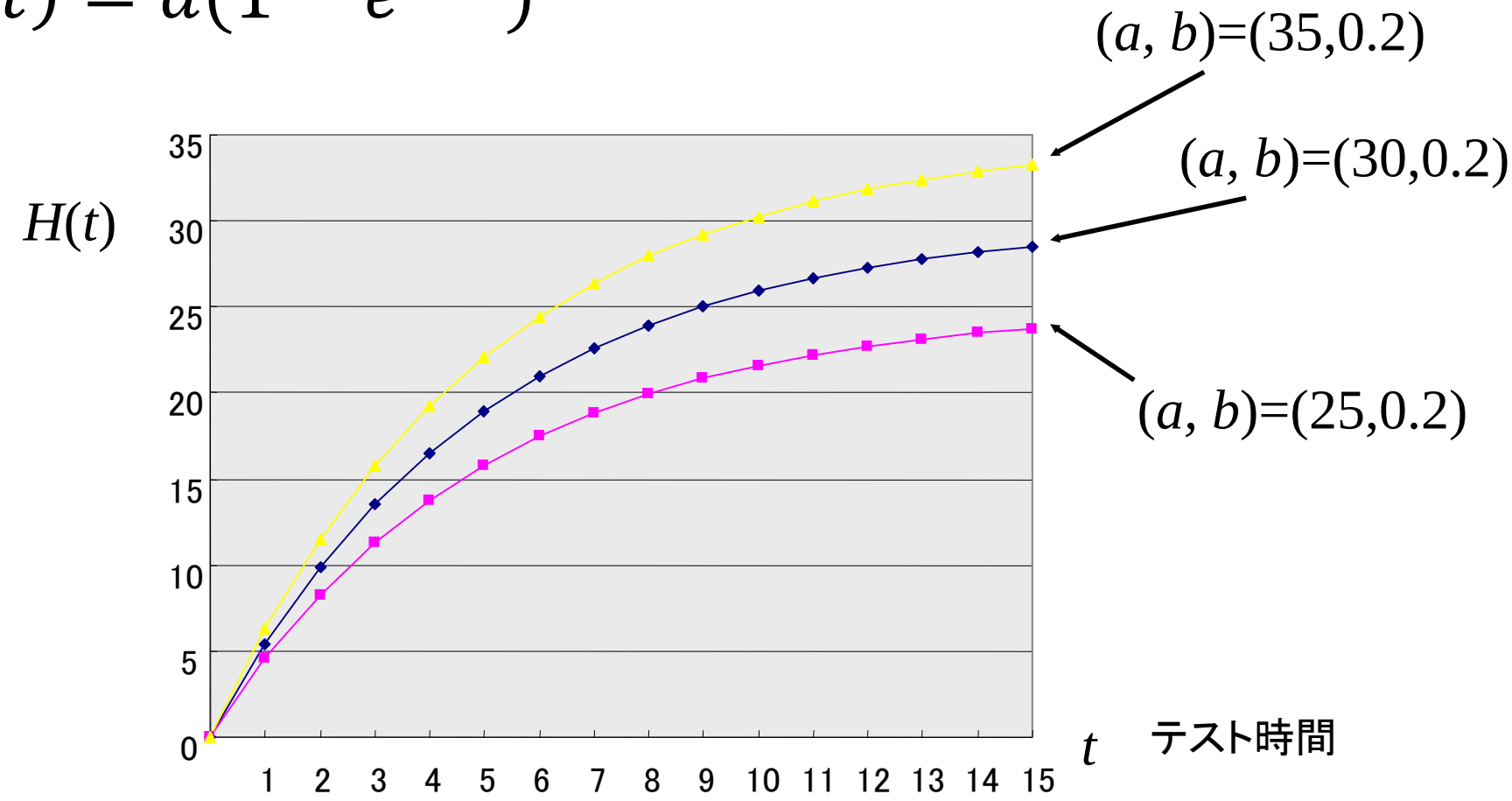
- もっとも単純なSRGM

$H(t) = a(1 - e^{-bt})$

- 考え方
  微小時間Δtの間に発見・除去されるフォールトの数は，残存フォールトの数とΔtに比例すると仮定
  - $H(t + \Delta t) - H(t) = \Delta t\ b\left(a - H(t)\right)$
    - a: 初期フォールトの数
    - b: 比例定数
  - $\frac{dH(t)}{dt} = b\left(a - H(t)\right)$
  - $H(t) = a\left(1 - e^{-bt}\right)$　(初期条件 $H(0) = 0$)

指数形SRGMのグラフ

- $H(t) = a(1 - e^{-bt})$



遅延S字型SRGM

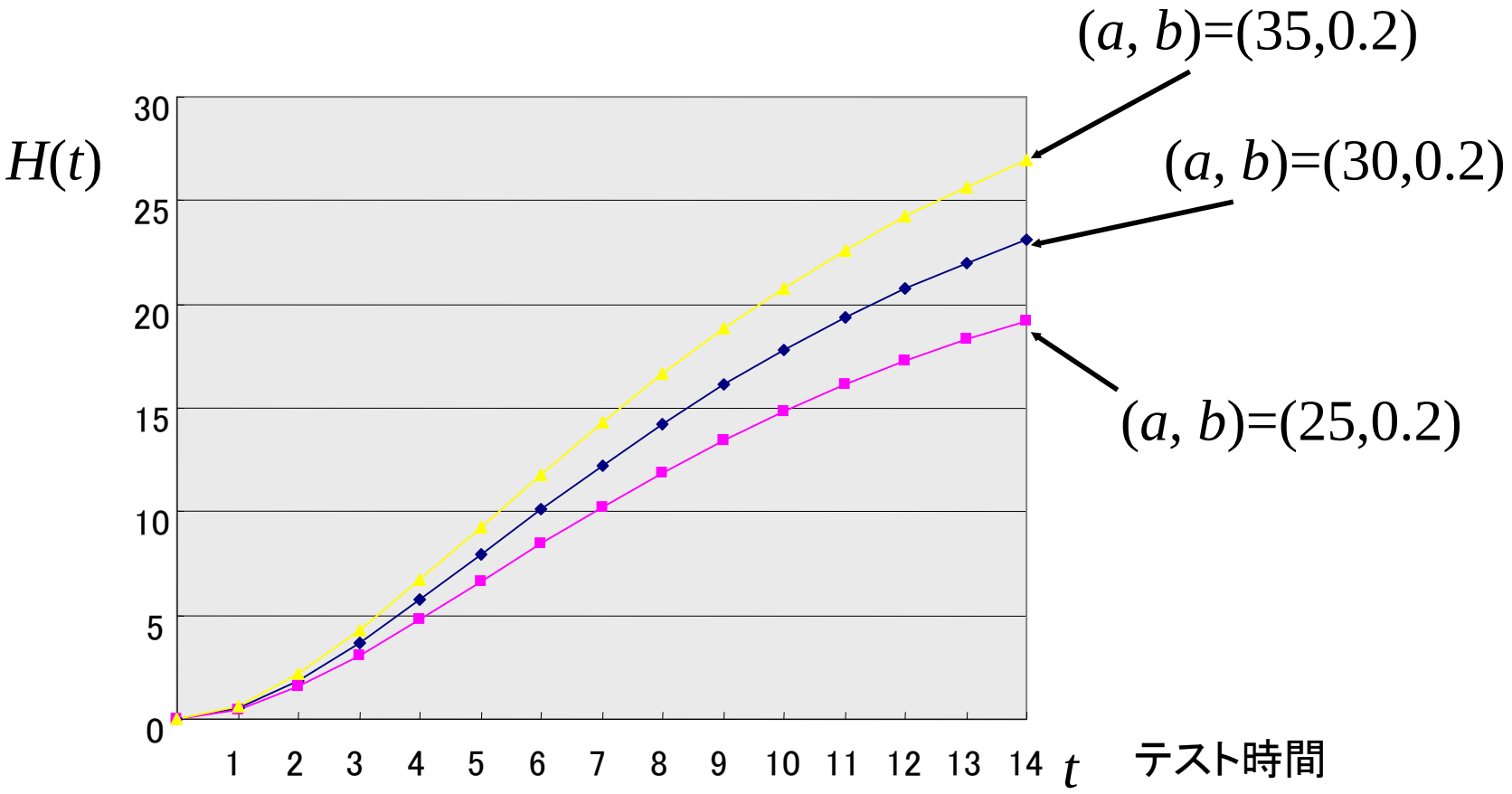
$$H(t) = a(1 - (1 + bt)e^{-bt})$$

- フォールトの発見と除去に分けて考える
  - $\frac{dm(t)}{dt} = b\big(a - m(t)\big)$, $\frac{dH(t)}{dt} = b(m(t) - H(t))$
    - ◆$m(t)$: 時刻tまでに発見されたフォールトの総数
  - $m(t) = a(1 - e^{-bt})$より

    $H(t) = a(1 - (1 + bt)e^{-bt})$

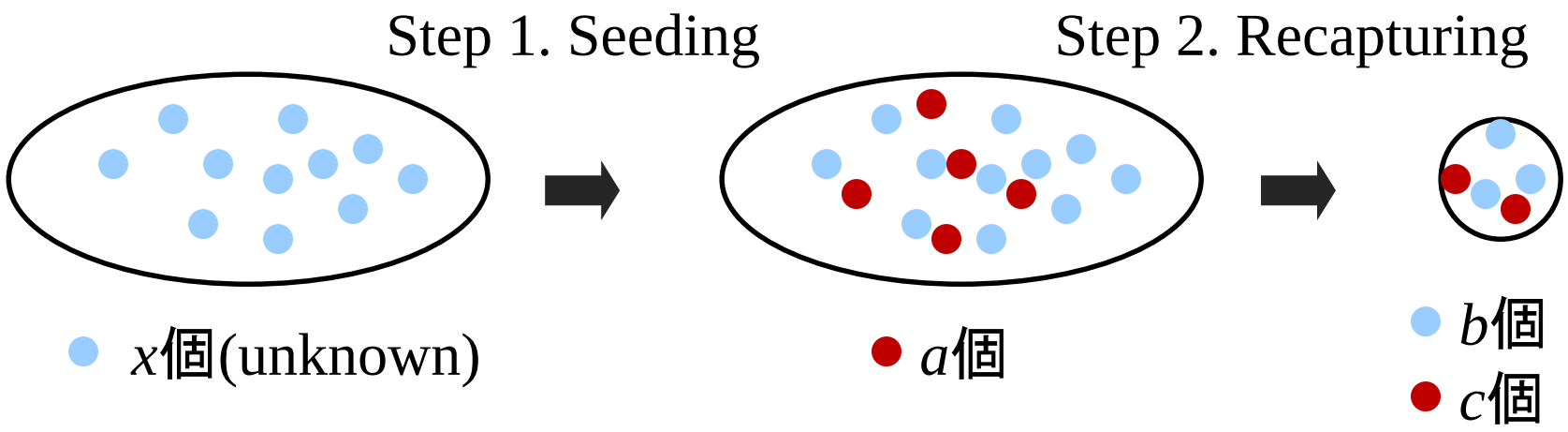
遅延S字型SRGMのグラフ

- $H(t) = a(1 - (1 + bt)e^{-bt})$



フォールト数を推定するSRGM以外の方法

- キャプチャー・リキャプチャー法
  - Step 1. フォールトを人為的にソフトウェアに入れ込む
  - Step 2. Step 1で入れたフォールトを知らない人間がテストを行う



*x*は？

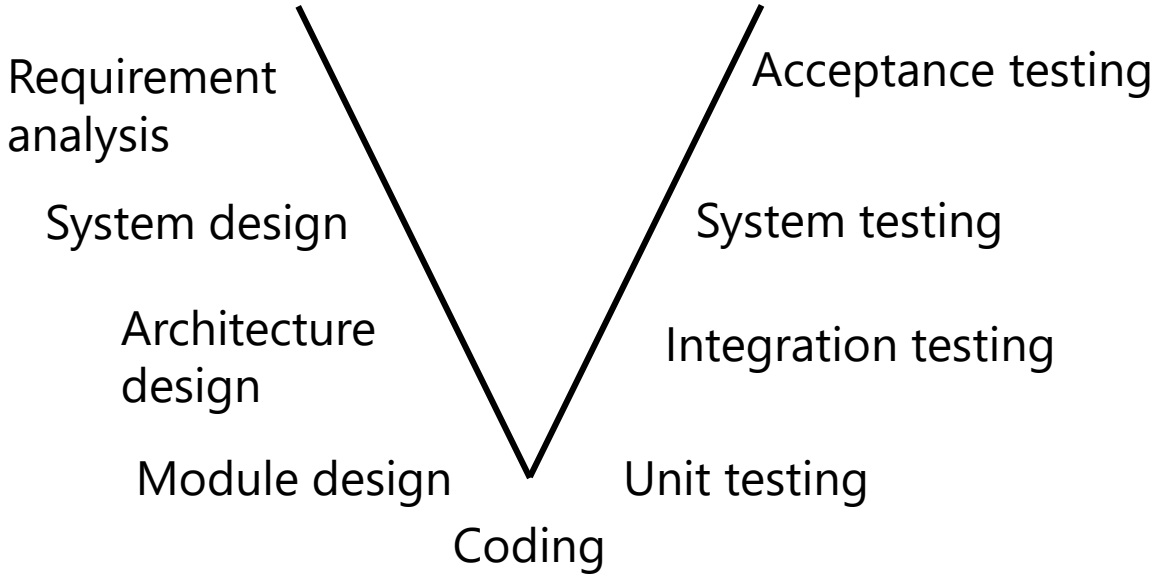
ソフトウェアテスト (Software testing)

- テストとは？○×？（by G.Myers)
  1. テストとは, エラーがないことを示していく過程?
  2. テストの目的は, プログラムが意図された通りに正しく動くことを示すこと?
  3. テストとは, プログラムが思い通りに動作するという確信を堅固にする過程?
  4. テストとは, エラーを見つけるつもりでプログラムを実行する過程?

G.Myers 他, ソフトウェア・テストの技報, 近代科学社

様々なテスト

- テスト対象による区別
  - ユニットテスト
    - ◆メソッド, クラス
  - 統合テスト
    - ◆モジュール間
  - システムテスト
  - 受理テスト
- 方法による区別
  - ホワイトボックステスト
    - ◆ソースコードに基づく
  - ブラックボックステスト
    - ◆外部から観測できる機能, 動作に基づく



Vモデル：システム開発の過程を表すモデル

テスト設計 Test design

- 主な課題: テストケースをどのようにつくるか
  - テストケース (test case)
    入力, 実行条件, 実行手順, 期待される出力, を定めたもの
    - ◆ 講義では主に入力について考える
- カバレッジ基準 (Coverage criterion)
  - テストが達成した程度を表す基準
  - 2通りの使い方
    - ◆満たす, 満たさない （100%達成か, そうでないか）
    - ◆テストできた割合. Coverage level*

P. Ammann and J. Offutt, Introduction to Software Testing, Cambridge University Press, 2008

コードカバレッジ基準

- どの程度プログラムが実行されるかを表す基準
- 代表的なコードカバレッジ基準
  - 命令カバレッジ (Statement Coverage, C0)
    - ◆各命令をカバー
  - 分岐カバレッジ (Decision (branch) Coverage, C1)
    - ◆各分岐をカバー
  - 条件カバレッジ (Condition Coverage, C2)
    - ◆各条件(condition)の真偽をカバー
    - ◆条件とは: `if (a > 1 && b == 0)` のとき `a > 1` と `b == 0`

例

```
…
if (a > 1 && b == 0) {
   x = x / a;
}
…
```

テストケースを，a, b の値とする

(a,b): (2, 0)
SC：○, DC:×, CC: ×

(a,b): (2, 0), (1, 0)
SC：○, DC: ○, CC: ×

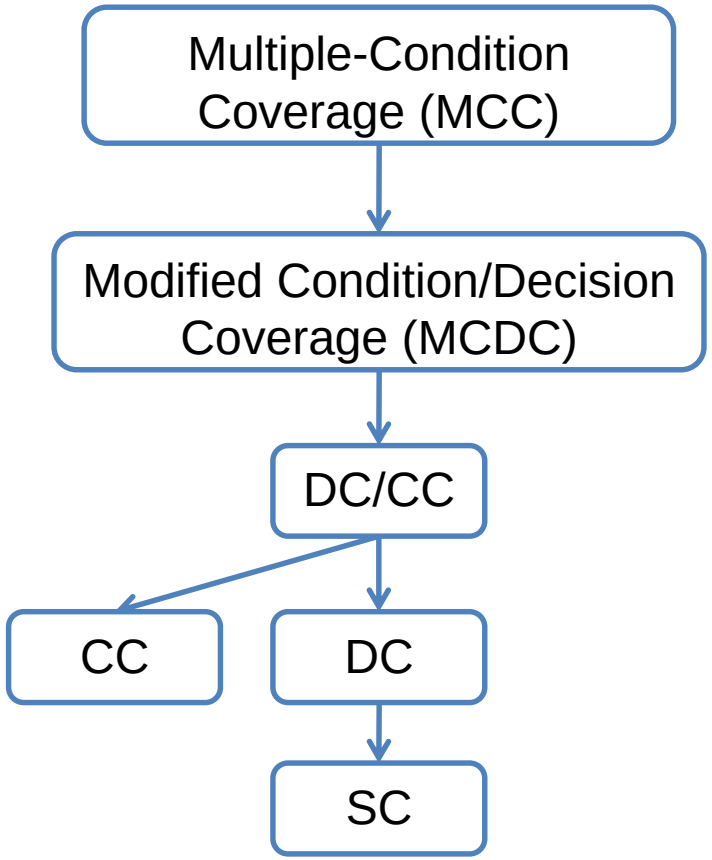
(a,b): (2, 1), (1, 0)
SC：×, DC: ×, CC: ○

```
function foo(a, b, x) {
  // a, b, and x are of integer type.

  if (a > 1 && b == 0) {
    x = x / a;
  }
  if (a == 2 || x > 1) {
    x += 1;
  }
}
```

C0, C1, C2 それぞれを
満たすテストケース集合
(テストスイート)は？

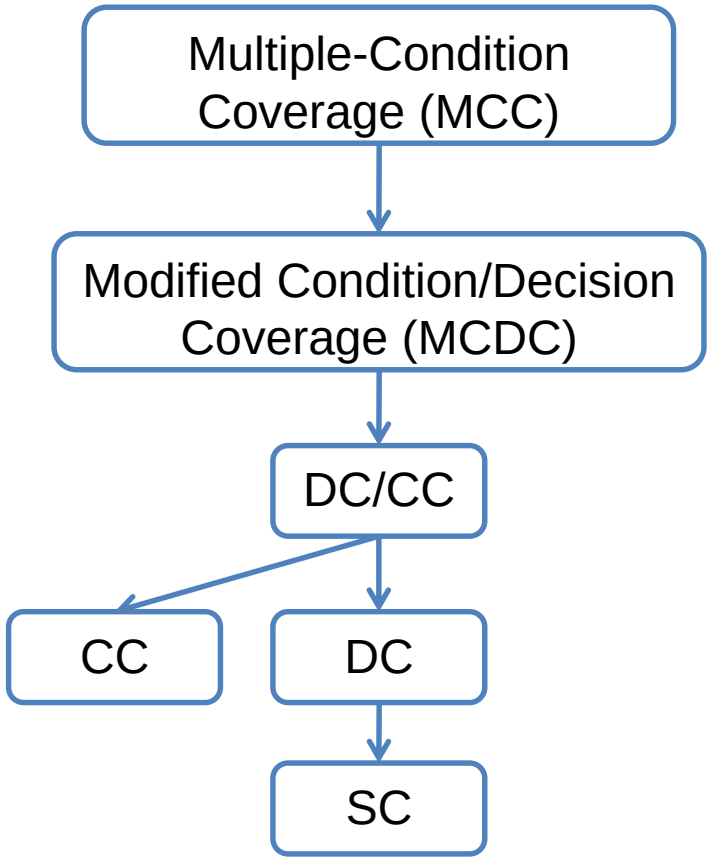
カバレッジ基準の包摂関係 (Subsumption relation)

- カバレッジ基準CがC'を包摂 (subsume)
  ⇔ Cが満たされるときC'も満たされる



より強いコードカバレッジ基準

- MCC
  - 条件の真偽の組み合わせ全て
- MCDC
  - 各条件について，他の条件の真偽を固定した上で，判定全体が真と偽となるようにテスト
  - 航空機のソフトウェアに関するガイドラインDO-178Bに導入されて普及

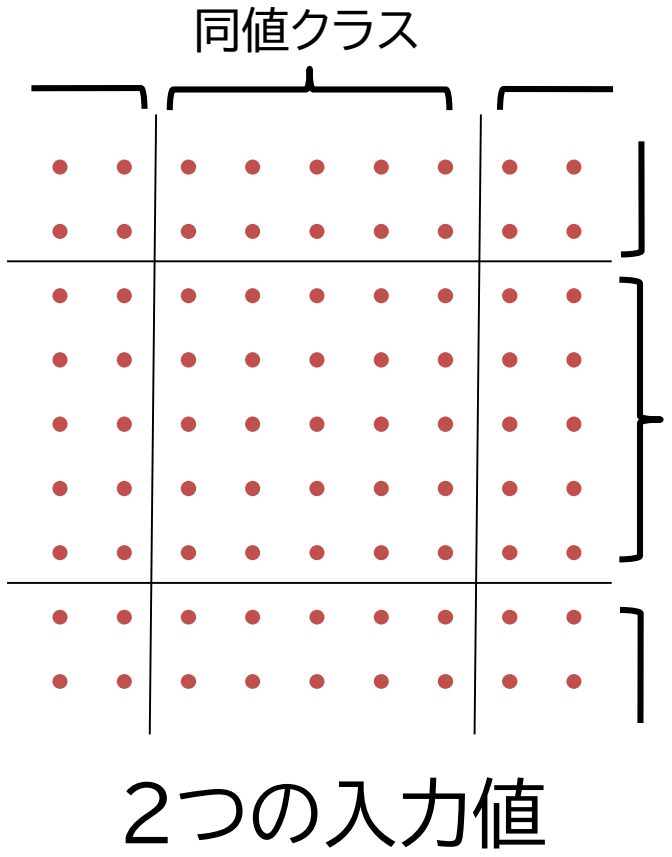


ブラックボックステストにおけるテスト技術

- 同値分割法
  Equivalent partitioning testing
- 境界値分析
  Boundary value analysis
- 組合せテスト
  Combinatorial testing
- その他
  - 状態遷移テスト
  - Etc.

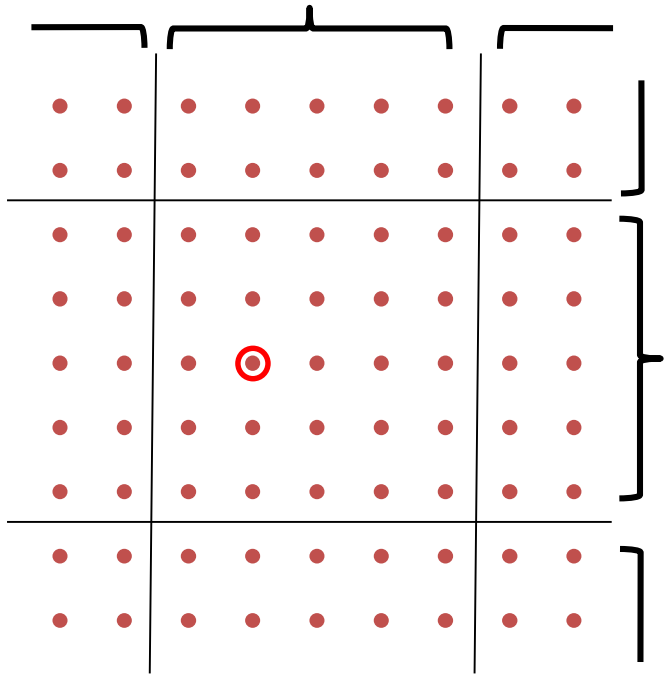
同値分割法

- 同値クラス
  - 実質的に「同じ値」と見なせる入力値のクラス
  - 例. 1～5までが有効な入力の場合,
    {0, -1, -2, ...}, {1, 2, 3, 4, 5} , {6, 7, ...}
    - ◆有効値：想定される正しい値
- 有効同値クラス
  - 有効な値からなる同値クラス
- 無効同値クラス
  - 無効な値からなる同値クラス



同値分割法

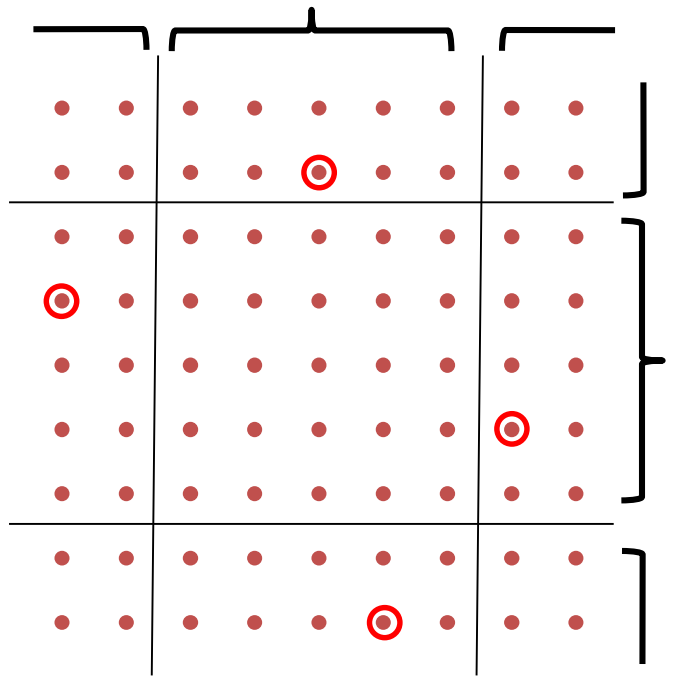
1. 同値クラスへ入力を分割
2. 有効値テスト (positive testing)
   - 有効同値クラスをカバーするように有効値のみのテストケースを設計



2つの入力値

同値分割法

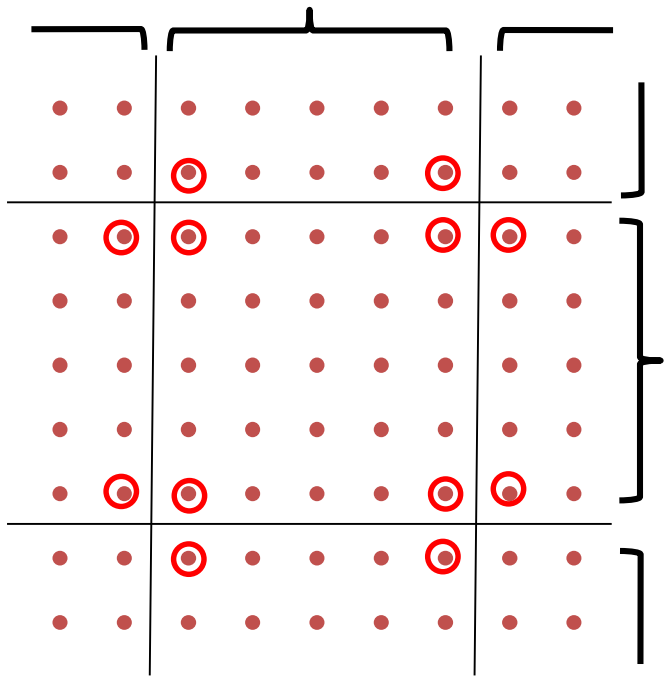
3. 無効値テスト(否定テスト, negative testing)
   - 無効同値クラスを1つだけカバーするテストケースによって, 無効同値クラスをカバー
     - 複数の無効値を入力すると, 1つを除く無効値の影響がマスクされてしまうため



2つの入力値

境界値分析

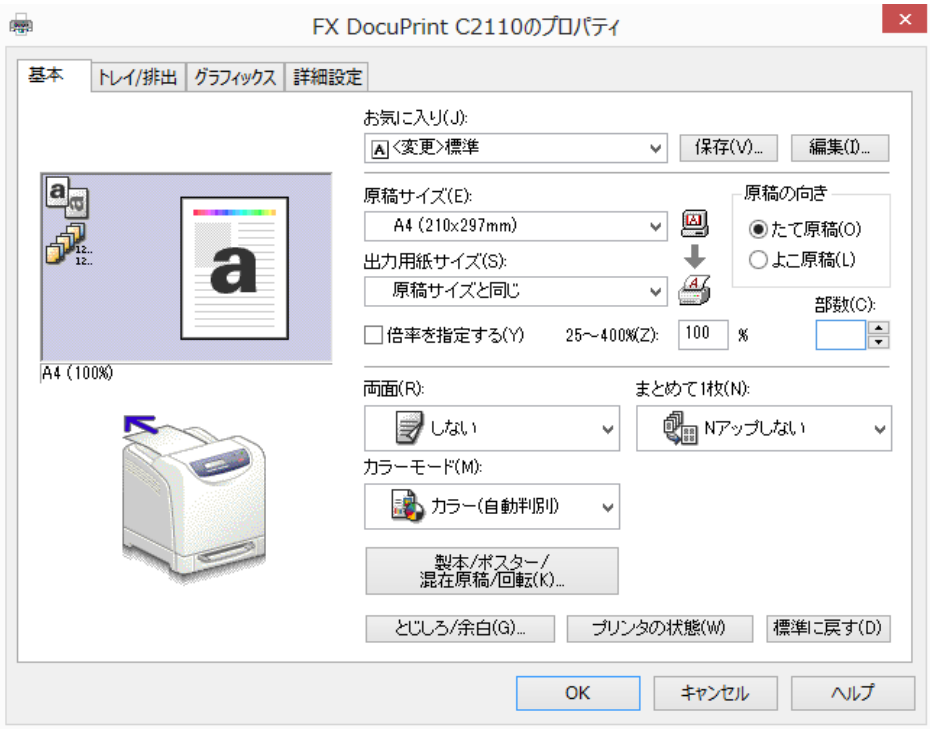
- 境界値
  - 入出力等の性質が変化する境界の値
    - ◆同値クラス間の境界の値
      - – 前の例. 0, 1, 5, 6
    - ◆出力が変化する点
    - ◆データ列の最初と最後
  - エラーが発生しやすいという経験的知見に依拠
- 境界値におけるエラーの例
  - ゲームソフト『ぷよぷよ！』で，セーブ回数が255回を超えた場合，以降のプレイ内容がセーブできなくなる (2006)



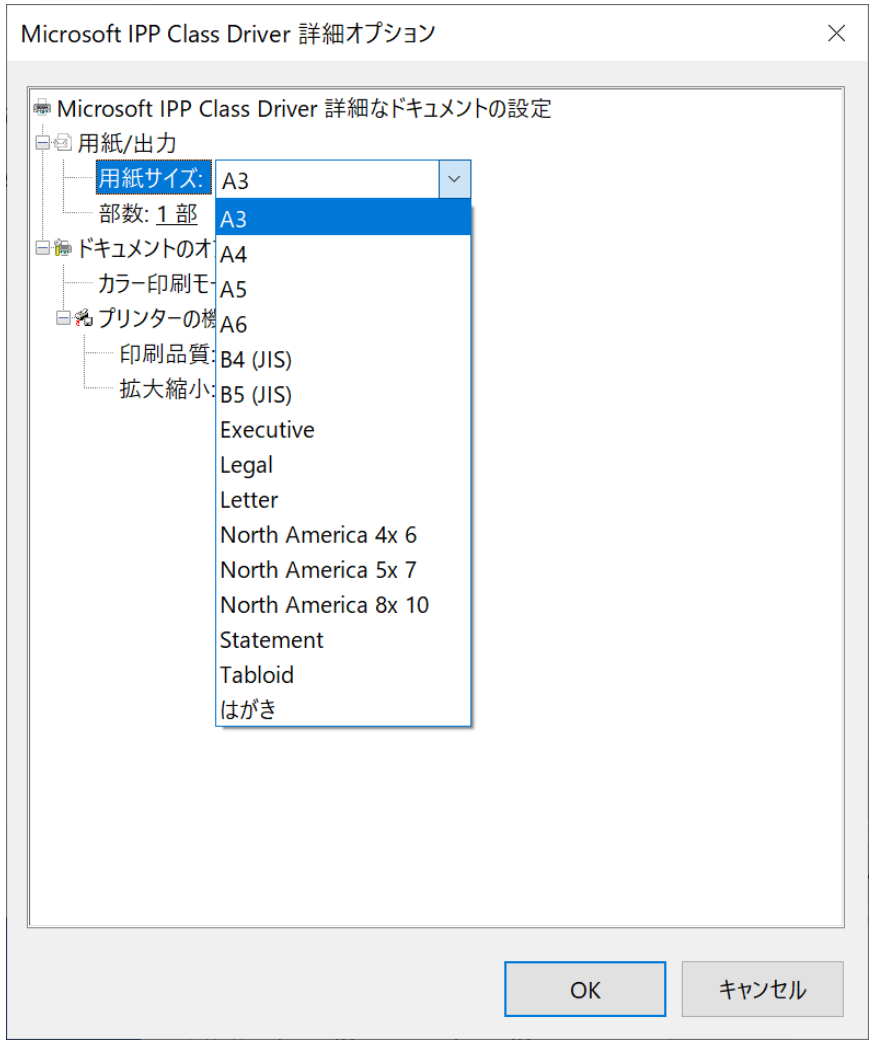
2つの入力値

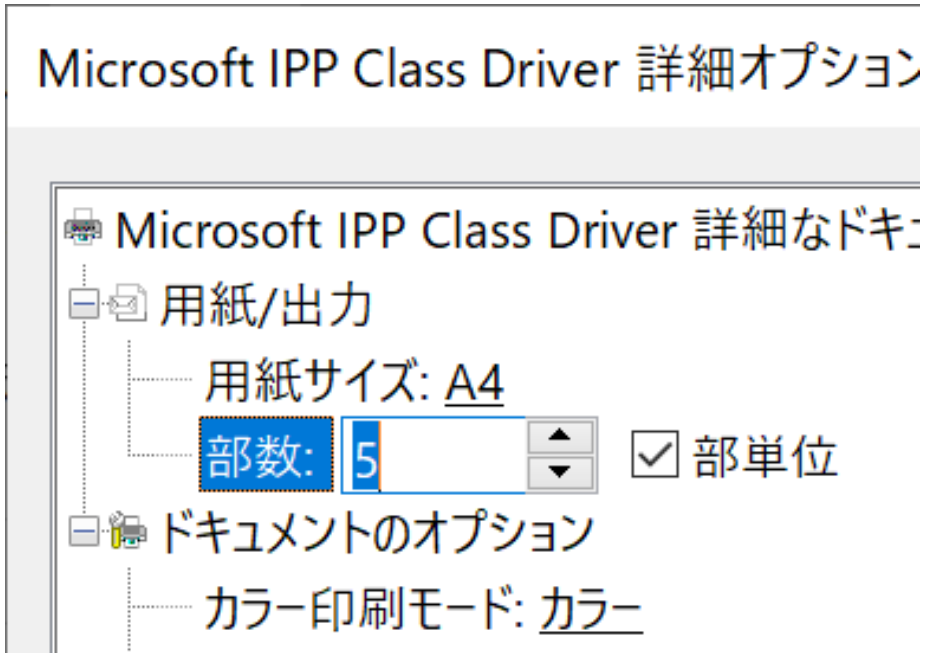
テストパラメータ分析

- テスト対象のシステム(SUT; System Under Test)から，テストするパラメータを抽出し，テストする値を決定
  - 入力値
  - SUTの状態
  - SUTの設定(configuration)
  - 実行環境の状態
  - Etc.
- 例. プリンタユーティリティ



テストパラメータ分析：プリンタユーティリティの例





この例では，無効値は設定できない

➡ 有効値テストだけを考える

分析結果

パラメータ	値
印刷の向き	横, 縦
両面印刷	なし, 長辺を閉じる, 短辺を閉じる
ページの順序	順, 逆
給紙方法	自動設定, トレイ1, トレイ2, トレイ3, トレイ4, 手差し
メディア	自動設定, 普通紙, 透明フィルム, 簡易用紙, 厚手用紙, ラベル
色	白黒, カラー
用紙サイズ	A3, A4, A5, A6, B4, B5, Executive, Legal, Letter, 4x6, 5x7, 8x10, Statement, Tabloid, はがき
部数	1, 2, 9998, 9999
部単位	無効, ✓なし, ✓あり
カラー印刷モード	カラー, モノクロ, グレースケール
印刷品質	標準, 高画質
拡大縮小	なし, ページの横幅に合わせる, 画面のサイズに合わせる

奇数, 偶数, 境界値

組合せテスト

- パラメータ値の組み合わせをカバーするテスト
  - カバー：1回以上実行すること
  - 例. プリンタユーティリティのパラメータと値（簡易版）

パラメータ	値
印刷の向き	横, 縦
両面印刷	なし, 長辺を閉じる, 短辺を閉じる
ページの順序	順, 逆
給紙方法	トレイ1, トレイ2, 手差し
メディア	普通紙, 透明フィルム, 簡易用紙

全ペアテスト
All-pairs testing, Pair-wise testing

- 全ペアテスト
  - 組合せテストの1種
  - 値のペアをカバー

テストケース11個で，すべてのペアをテスト

☛ すべての組合わせをカバーする場合は，何個必要？

印刷の向き	両面印刷	ページの順序	給紙方法	メディア
横	なし	順	トレイ1	普通紙
縦	長辺を閉じる	順	トレイ2	透明フィルム
横	短辺を閉じる	逆	手差し	透明フィルム
縦	なし	逆	トレイ2	簡易用紙
横	長辺を閉じる	逆	トレイ1	簡易用紙
縦	短辺を閉じる	順	手差し	簡易用紙
横	短辺を閉じる	逆	トレイ2	普通紙
縦	長辺を閉じる	逆	手差し	普通紙
縦	なし	順	トレイ1	透明フィルム
縦	なし	順	手差し	透明フィルム
横	短辺を閉じる	順	トレイ1	簡易用紙

組合せテストのカバレッジ基準

- *t*-way カバレッジ
  - *t*個のパラメータについて，値の組合せをカバー（1回以上テスト）
    - ◆*t*を強度(strength)とよぶことがある
- ペアワイズカバレッジ (pairwise coverage)
  - $t = 2$

組合せテストの背景

- 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

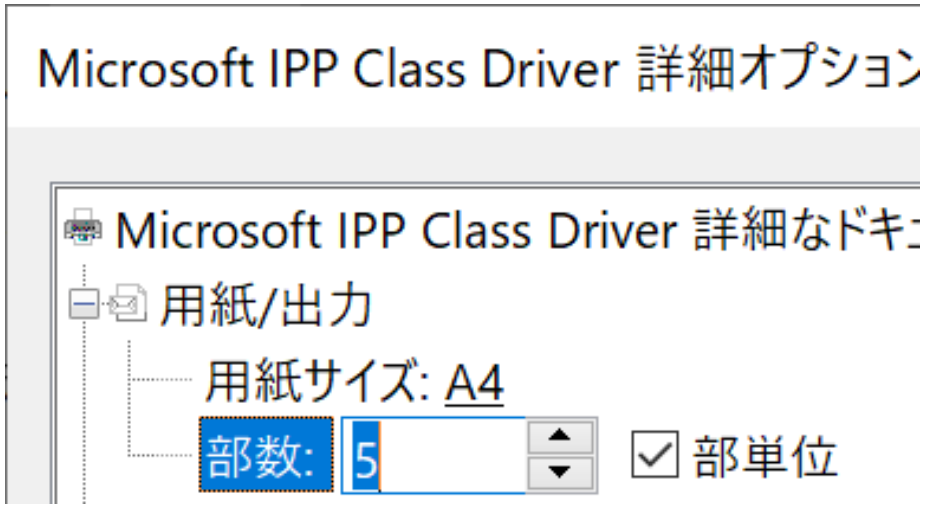
FTFI	エキスパートシステム				OS	組込み	Browser	Apache	DB
1	61	72	48	39	82	66	29	42	68
2	36	10	6	8	*	31	47	28	25
3	*na	*	*	*	*	2	19	19	5
4	*	*	*	*	*	1	2	7	2
5	*	*	*	*	*		2	0	
6	*	*	*	*	*		1	4	

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

設定不能な値の組合せ，制約

- 設定不能な値の組合せ
  - 部数:1 ＋
    部単位:✓あり，✓なし
  - 部数: 2, 9998, 9999 ＋
    部単位: 無効
- 制約 constraint
  - テストケースが満たさないといけない条件
  - 設定不能な値の組合せが出現しないように設定
    - ◆部数:1 ⇒ 部単位:無効
    - ◆¬部数:1 ⇒ (部単位:あり ∨ 部単位:なし)

      ⇒は，含意 (implication)



組合せテスト用テストケース生成ツール

```
部数: 1, 2, 9998, 9999
部単位: 無効, なし, あり
カラー印刷モード: カラー, モノクロ, グレースケール
印刷品質: 標準, 高品質

IF [部数] = 1 THEN [部単位] = "無効";
IF [部数] > 1 THEN [部単位] = "あり" OR [部単位] = "なし";
```

PICT, YACCT

https://github.com/Microsoft/pict

https://github.com/ikomamik/yactt

```
部数 (1 2 9998 9999)
部単位 (無効 なし あり)
カラー印刷モード (カラー モノクロ グレースケール)
印刷品質 (標準 高品質)

(if (== [部数] 1) (== [部単位] 無効))
(if (> [部数] 1)
  (or (== [部単位] なし)
      (== [部単位] あり)
  )
)
```

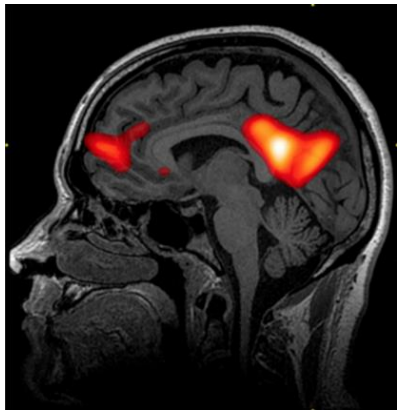
CIT-BACH

https://osdn.net/users/t-tutiya/pf/cit_bach/

AI時代のソフトウェアテスティング

- AIシステムがもたらす問題
  - AI技術を用いたソフトウェアをどうテストするか？
- 事例紹介：fMRI解析プログラムのバグ [Eklunda et al., PNAS 2016]
  - fMRI: 脳活動に関連した血流動態反応を視覚化
  - 3つの普及している解析パッケージすべてにバグ

➡多くの実験結果が無駄に



Source: John Graner, Walter Reed National Military Medical Center, public domain

テストにおける問題点

- テストオラクル (test oracle)
  - テストの成否(pass/fail)を判定する機構
  - AIシステムでは, 正しい解が分からないことが多い
    → 実現が困難
- コードカバレッジ
  - 誤った解を生成するフォールトは, コードカバレッジが高くても(実行したコードの範囲が多くても), 検出できない
    → コードカバレッジの有効性が極めて限定的

解決への取り組み
メタモルフィックテスティング

- メタモルフィックテスティング (metamorphic testing)
  - メタモルフィック関係を利用したテスト
- 複数の入力と対応する出力に関する関係
  - メタモルフィック関係の例. 経路案内

$$time(route(大阪, 新潟)) \leq time(route(大阪, 東京)) + time(route(東京, 新潟))$$

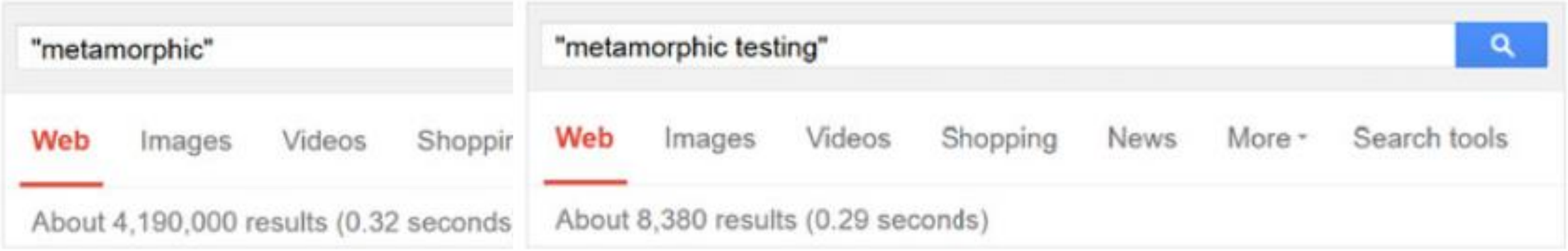
メタモルフィックテスティング

- 複数の入力に対してSUTを実行し，対応する出力を得る
  - 入力：*in*1: (大阪，新潟), *in*2: (大阪，東京)，*in*3: (東京，新潟)
  - 出力：*out*1: route(大阪，新潟),
    *out*2: route(大阪，東京), *out*3: route(東京，新潟)
- 複数の入力とそれらに対する出力に関して，メタモルフィック関係が成り立っているか調べる
  - ？ $time(out1) \leq time(out2) + time(out3)$

➡ 正しい解が分からなくてもテスト可能

メタモルフィックテスティング適用事例

- 例1. 検索サービスのテスト*
  - メタモルフィック関係:
    “word1”のヒット数 ≥ “word1 word2”のヒット数



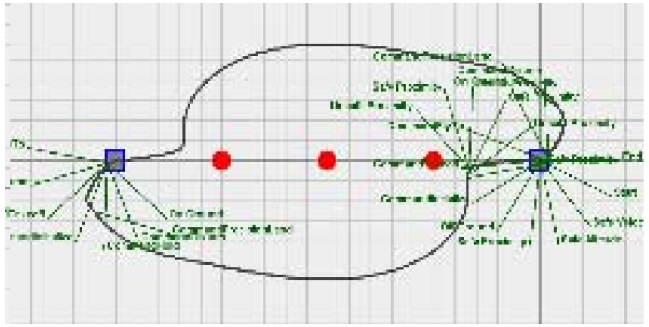
関係が満たされないなら, バグ, 動作の不整合

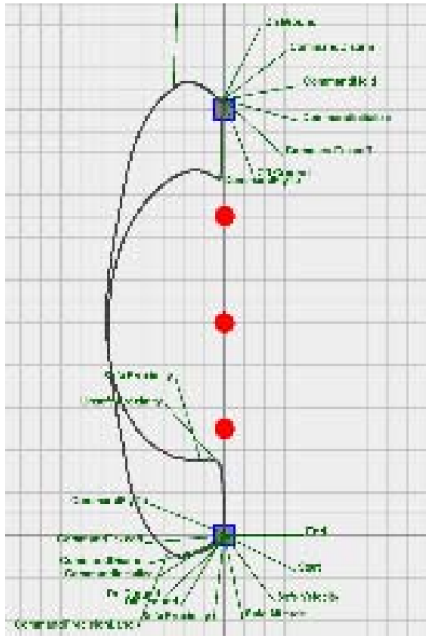
*Z.Q. Zhou, S. Xiang, T.Y. Chen, Metamorphic testing for software quality assessment: A study of search engines, IEEE Transactions on Software Engineering 42 (3), 2015

メタモルフィックテスティング適用事例

- 例2. 無人ドローンの航路
  - メタモルフィック関係
    配置を回転させても航路は変わらない

回転により航路が変化



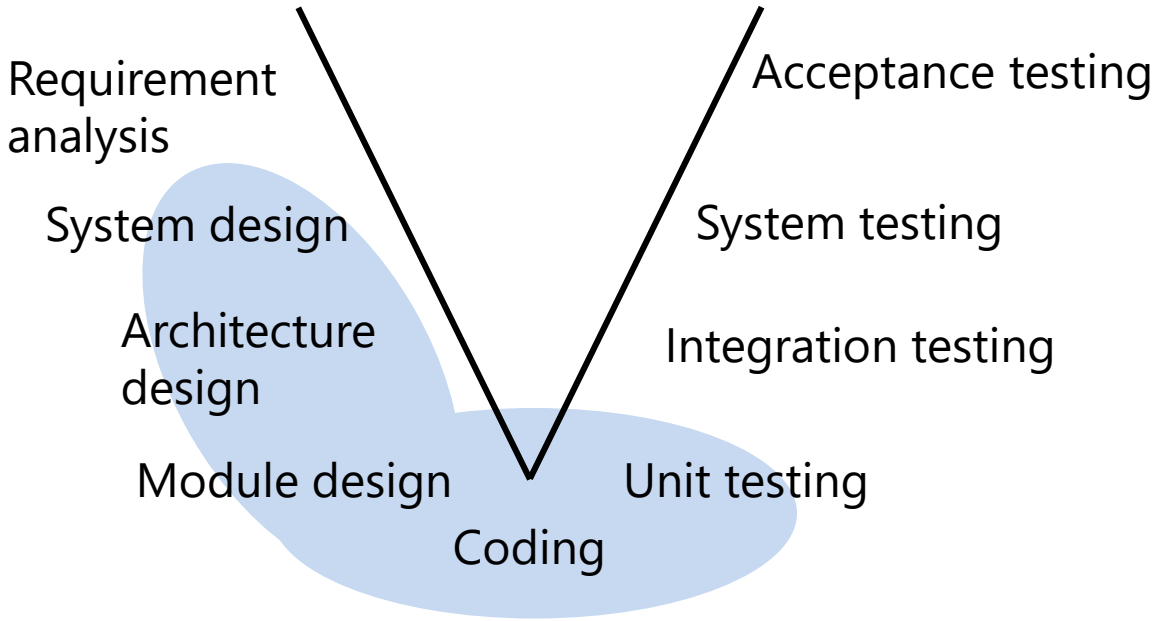


引用. Lindvall et al., Metamorphic Model-based Testing of Autonomous Systems, MET 2017.

個々の出力の正しさが判定できない場合でもテストが可能

形式手法 (Formal methods)

- 厳密な方法でシステムを設計，検証する手法
  - 形式仕様記述 (formal specification)
  - モデル検査 (model checking)



形式仕様記述 (Formal specification)

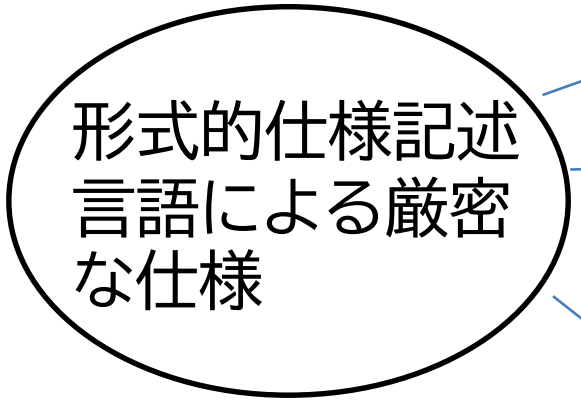
- システム仕様を数学的に厳密に記述
  - 言語の例
    - VDM
    - Z
    - Alloy
    - TLA+,...

ハノイの塔：
TLA+による表現

```
EXTENDS Sequences, Integers
VARIABLE A, B, C


CanMove(x,y) == /\ Len(x) > 0
                /\ IF Len(y) > 0 THEN Head(y) > Head(x) ELSE TRUE

Move(x,y,z) == /\ CanMove(x,y)
               /\ x' = Tail(x)
               /\ y' = <<Head(x)>> \o y
               /\ z' = z

Invariant == C /= <<1,2,3>>   \* When we win!

Init == /\ A = <<1,2,3>>
        /\ B = <<>>
        /\ C = <<>>

Next == \/ Move(A,B,C) \* Move A to B
        \/ Move(A,C,B) \* Move A to C
        \/ Move(B,A,C) \* Move B to A
        \/ Move(B,C,A) \* Move B to C
        \/ Move(C,A,B) \* Move C to A
        \/ Move(C,B,A) \* Move C to B
```

形式仕様記述 (Formal specification)

- 形式仕様記述の用途
  - 言語の例
    - VDM
    - Z
    - Alloy
    - TLA+,...



テストケースの自動生成

定理証明

プログラムの正しさを証明

使用例. CompCert

モデル検査

状態探索による検証

あいまい性,
矛盾の削除

使用例. AWS

Chris Newcombe et al.: How Amazon web services uses formal methods. Commun. ACM 58(4): 66-73 (2015)

モデル検査 (Model checking)

- モデル検査
  - コンピュータによる状態探索で，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
- ロジックモデル検査 (logic model checking)
  - 狭義のモデル検査
  - 時相論理(temporal logic)の式が成り立つ解釈=モデルを探索
    - 時相論理式の例: $\Box\Diamond p$「どの状態でも，いずれpが成立」

有界モデル検査 (Bounded model checking)

- モデル検査の一例として，有界モデル検査を紹介
- 有界モデル検査
  - 検査するシステムの要素の数や動作の長さを限定して，検証を行うモデル検査手法
- システムのモデルが検証対象
  - nuXmv
  - Alloy
- コードが検証対象
  - CBMC, ESBMC

コードに対する有界モデル検査

- コードを制約に変換し，充足解を探索

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

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

解があれば，assert違反の動作が存在

コードに対する有界モデル検査

- 制約の充足解の探索
  - SATソルバ, SMTソルバを利用

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

➡

Z3への入力例

```
(declare-const b0 Bool)
(declare-const x0 Int)
(declare-const x1 Int)
(declare-const x2 Int)
(declare-const x3 Int)
(declare-const y0 Int)
(declare-const y1 Int)

(assert (> (+ y0 x0) 2))
(assert (or
  (and b0 (= x1 (+ x0 1)))
  (and (not b0) (= x1 x0))
))
(assert (or
  (and (not b0) (= x2 (+ x1 2)))
  (and b0 (= x2 x1))
))
(assert (= y1 (+ y0 x2)))
(assert (= x3 (+ y1 1)))
(assert (not (> x3 5)))
(check-sat)
(get-model)
```

ソルバの実行結果

https://jfmc.github.io/z3-play/

```
sat
(model
  (define-fun b0 () Bool
    true)
  (define-fun y0 () Int
    0)
  (define-fun x2 () Int
    4)
  (define-fun x1 () Int
    4)
  (define-fun x0 () Int
    3)
  (define-fun x3 () Int
    5)
  (define-fun y1 () Int
    4)
)
```

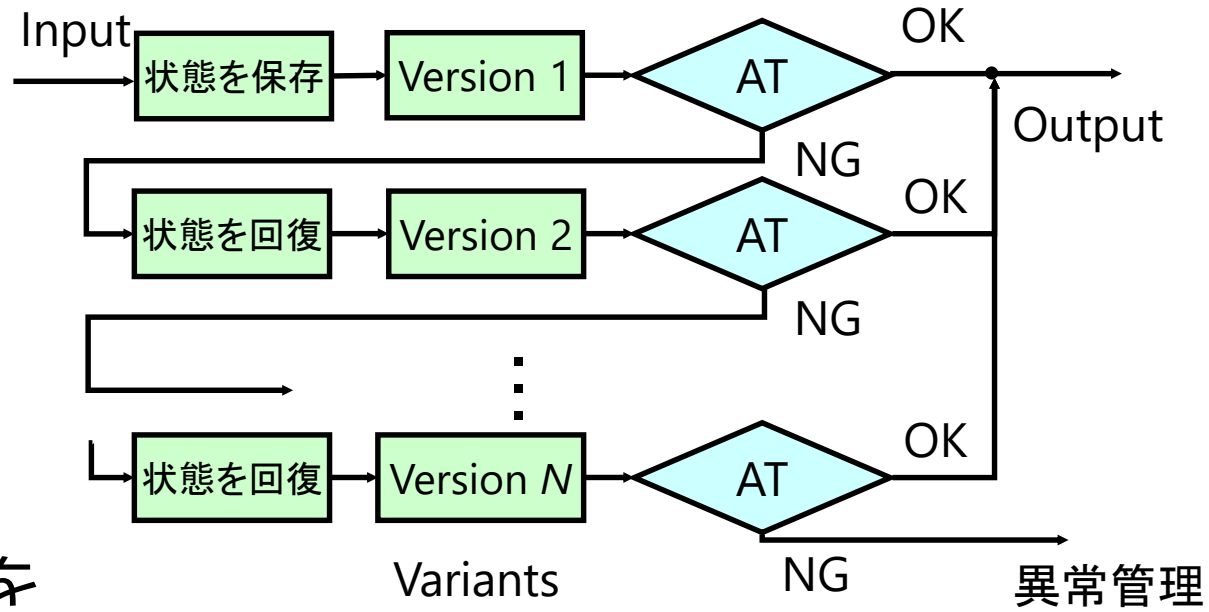
```
assume(x + y > 2);
if (b)
  x = x +1;
else
  x = x + 2;
y = y + x;
x = y + 1;
assert(x > 5);
```

フォールトトレラントソフトウェア

- 設計フォールト(design fault)に耐えられるソフトウェア
- 設計分散 (design diversity)
  - 同じ仕様だが設計の異なる複数のソフトウェアを使用するという考え方
    - バリアント (variant): 個々のソフトウェア
  - フォールトトレラントソフトウェアの基本
- 具体的な方法
  - リカバリブロック (recovery blocks)
  - Nバージョンプログラミング (N-version programming)

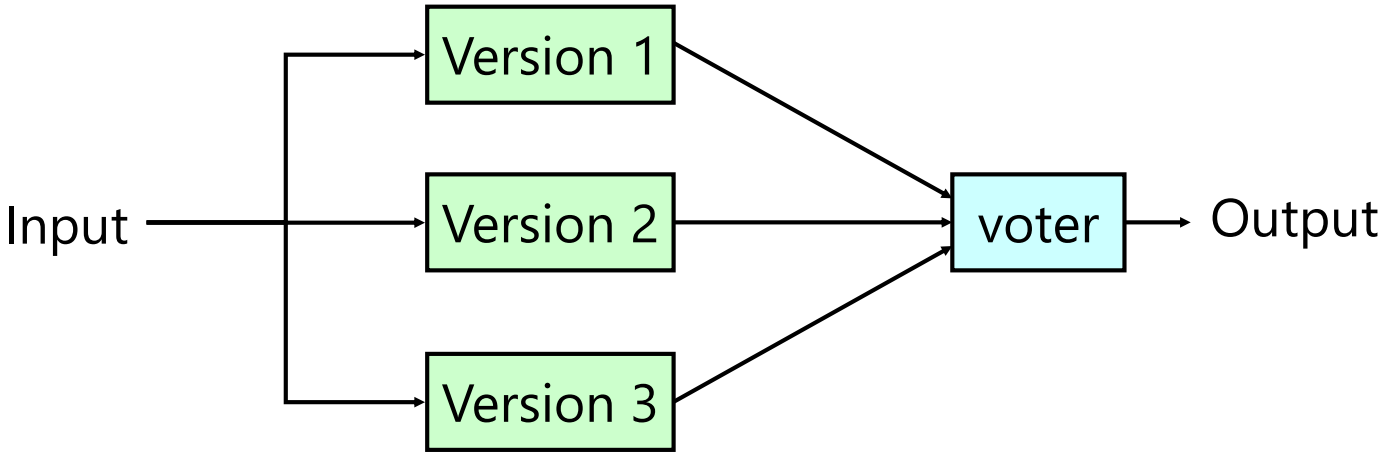
リカバリブロック (recovery blocks)

- バリアントを1つずつ実行する方法
- ステップ
  1. 1つのバリアントを実行
  2. 出力を受理テスト(acceptance test)で判定
     - ◆成功 (pass)：終了
     - ◆失敗 (fail): 別のバリアントを用いて1から繰り返す



Nバージョンプログラミング (N-version programming)

- 複数のバリアントを同時に実行する方法
- 応用例
  - Swedish State Railway
  - Airbus A320/A330/A340



Nバージョンプログラミングの問題点

- 相関する障害 (correlated failures)
  - 異なるバリアントが同じ入力に対し障害を起こす場合がある
  - Knight, Levesonにより, 実験的に確認 (1985)
    - ◆ 27バージョンのプログラムを検査
- 開発費用
  - 単一バージョンの場合よりも大幅に増加
  - ただし, N倍以下の場合もあり
    - ◆例. N＝2のとき, 第2バージョンの開発コスト 25% ～ 134%*

K. Kanoun, "Cost of software design diversity an empirical evaluation," Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443), 1999, pp. 242-247, doi: 10.1109/ISSRE.1999.809329.

データ分散

- 設計分散を用いないアプローチ
- 入力データを少し変更して(あるいは表現を変えて)処理を再実行
  - エラーの発生は入力値に依存するという考え方
- 例. sin(x)を様々な入力を用いて計算

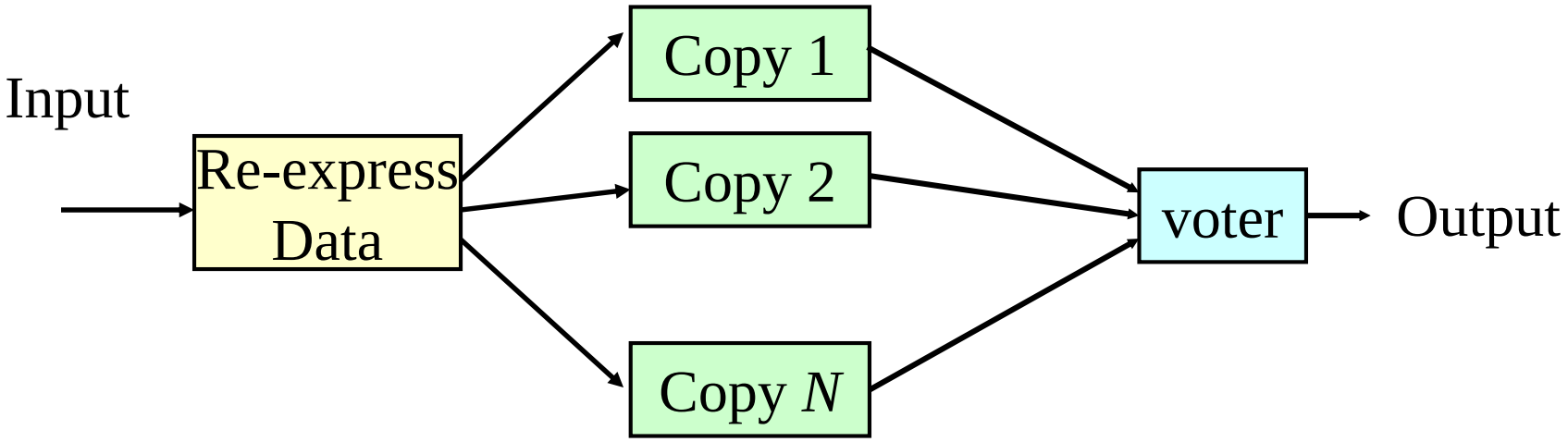
  $\sin(x) = \sin(a)\sin(\pi/2-x+a) + \sin(\pi/2-a)\sin(x-a)$

  - $\sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b)$
  - $\cos(a) = \sin(\pi/2 - a)$

P. E. Amman and J. C. Knight. Data Diversity: An Approach to Software Fault Tolerance.
In Proc. of 17th Intl. Symposium on Fault Tolerant Computing, pages 122-126, June 1987.

N-Copy programming

- N通りの入力に対し，処理を実行
  - N-バージョンプログラミングのデータ分散版



$\sin(a_N)\sin(\pi/2–x+a_N) + \sin(\pi/2–a_N)\sin(x–a_N)$

$$\sin(x) = \sin(a)\sin(\pi/2–x+a) + \sin(\pi/2–a)\sin(x–a)$$