形式検証の必要性

実践的モデル検査を考えよう！

財団法人福岡県産業・科学技術振興財団　片平典幸

目次

- 形式検証ってなんだ？
- 実践的なモデル検査をしてみよう！
- Garakabu2って何だ？

形式検証ってなんだ？

- 形式手法とは？

  数学を基盤としたソフトウェア、ハードウェアシステムの
  仕様記述、開発、検証の技術の総称。

  水準0 形式仕様記述を行い、プログラム自体を非形式主義的に行う。「軽い形式手法」と呼ぶ。費用対効果が早く得ることができる選択である。

  水準1 開発と検証を行い、より形式主義的にプログラムを生成する。例えば、仕様記述からプログラム作成において詳細化と属性の証明を行う。高信頼システムに適した選択である。

  水準2 自動定理証明によって完全に機械的に証明が行う。道具を整備するのに費用がかかるか、厳密である必要がありシステムを記述するのに手間がかかる。間違いが混入することで生じる損失に見合わなければ実施しない。

  - プログラム意味論の分類（形式手法）:

    表示的意味論： システムの意味は領域理論で表現。領域理論の性質によってシステムの意味を与える。しかし、あらゆるシステムが関数に直感的に表現できるわけではないとも言われている。（CSPなどのプロセス代数表現）

    操作的意味論：操作的意味論では、より単純な計算モデルの一連の動作によってシステムの意味を表現する。この場合、モデルの単純性が表現を明確にする。しかし、これは意味論的な判断の先延ばしとも言われている。（状態遷移系・・・STM,STD、UML等）

    あと、公理的意味論なんてのもありますが、省略します・・・。

目次

- 形式検証ってなんだ？
- 実践的なモデル検査をしてみよう！
- Garakabu2って何だ？

形式検証ってなんだ？

- 形式手法って、結局なんなの？
  …しかし、開発現場では、意味論をどうのこうのが問題でなく、
  正しい設計・実装をするために
  形式手法（モデル検査）を使うのです・・・。

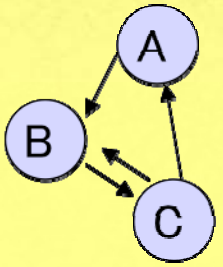
  一般にソフトウェア開発で使われるのは、
  軽量(light weight)形式手法
  です。

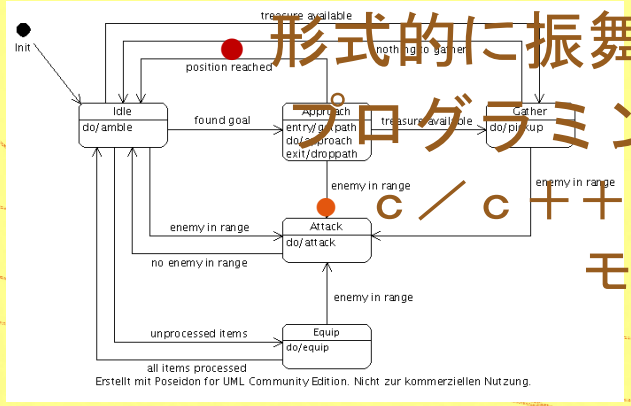
- モデル検査と形式手法の関係は？
  形式検証（モデル検査）とは？
  形式手法で記述されたモデルを数理的に検証する技法　のこと。

※形式手法＝モデル検査　ではないという意見もある。

形式検証ってなんだ？

- モデル検査のモデルって？
  - ここでいうモデルは、状態遷移（振舞い）を記述したモノを指します。
    - 例えば・・・
      - UML（Activity図、StateMachine図）
      - 状態遷移表（StateTransitionMatrix）
      - 状態遷移図（StateTransitionDiagram)
      - 等々・・・





- 形式的に振舞いが記述されている場合、
プログラミング言語も振舞いモデルと考えられます。
  - ｃ／ｃ＋＋言語を形式的言語と考え、
モデル検査するツールもあります。

STM1	Sts1	StsA
Ev1		
Ev2		

形式検証ってなんだ？

- 形式検証（モデル検査）の
  “検証・検査”って、
  今までの“テスト”と何が違うの？
  - 数理的網羅検査なので、
    全ての組合せの検査を
    行うことが可能。

※例えば、デバッグ時の
フルパステストではアルゴリズムの全分岐テストはできるが、
特定の条件（検査性質）に対して変数値の変化範囲全ての
組合せ検査は出来ていない。それを自動化して行うことができる。

★でも、実装物（ソースコード）ではなく考え方（設計）を検査出来るコトが一番素晴らしい（と我考える。　異論はあると思いますが…）

形式検証ってなんだ？

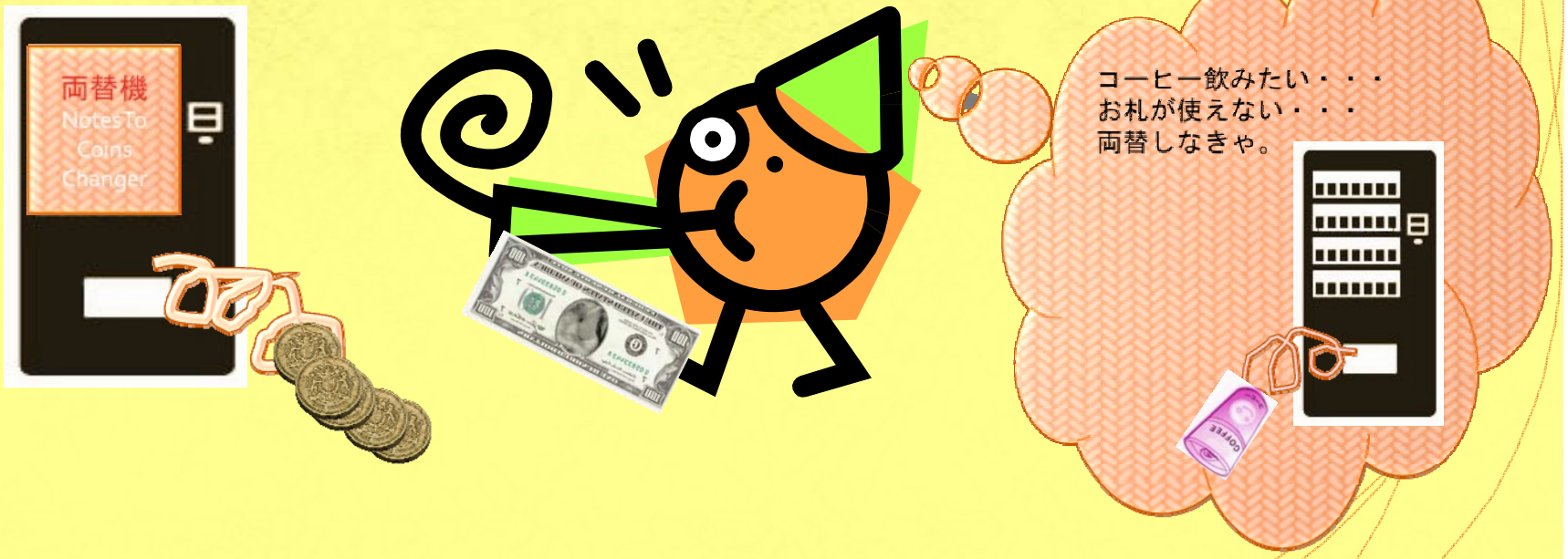
- 形式検証（モデル検査）って何が嬉しいの？
  - “こんなとき、いつかこうなる”という検査が出来る。
    （**時相論理**：時系列な振舞いの変化の検査が出来る）
  - 設計レベルでの検証（**妥当性確認**）が出来る。
  - あってはならない挙動を見つけることが出来る。
    ※モデル検査によってあったりなかったり。
    Garakabu2では“不可セル”という定義で
    ユーザーが一意に設定できる。

目次

- 形式検証ってなんだ？
- 実践的なモデル検査をしてみよう！
- Garakabu2って何だ？

実践的なモデル検査をしてみよう！

- 両替機制御のモデル検査をしてみよう！



・・・どんなことを検査（検証）するのかな？

実践的なモデル検査をしてみよう！

- 検査したいことを挙げてみよう
  - ありえない状態遷移（設計）をして行かないこと。
  - コインが足りないときは、両替しない。
  - コインがなくなるまで、両替できれば両替する。
  - 紙幣一枚ずつしか両替できない機能を確認する。

- その前に・・・
  - なんで検査したいんだっけ？
    - 両替機の制御設計が正しく出来ているか確認するため。
  - では、制御を状態遷移（≒設計）で書いてみましょう。

実践的なモデル検査をしてみよう！

状態遷移表の読み方

		Status A	Status B
		0	1
Event A	0	Transition Destination Action	
Event B	1		Transition Destination Action
Event C	2		Transition Destination Action

状態記述部

遷移先記述部

実行部記述部

イベント記述部

両替したい人の状態遷移表（例）

□ RETURNER		WAIT	RETURN	
		0	1	
payment	0	RETURN payment=false;	×	

xRecieve	1	/	payMoney == 0
			WAIT
			getMoney=true;
			xReceive=false;
				else
				WAIT
				payMoney=0;
				getMoney=true;
				xReceive=false;

★見かた
支払い=両替(イベントpayment発生)したいとき、
何もしていない状態（状態WAIT）だったら、
コインが出るのを待つ状態になり（次の状態がRETURN）、
支払いをやめる（payment=false）。

実践的なモデル検査をしてみよう！

両替機の状態遷移表

☐ CHANGER		STOP	WAIT_REQUEST		WAIT_MONEY_TAKEN
		0	1		2

xChangePrepare	0	WAIT_REQUEST
		changeMoney=changeMoney+10000;
		xChangePrepare=false;
			/		/

x10KYenRequest	1	/	changeMoney >= 10000
			WAIT_MONEY_TAKEN
			payMoney=10000; payment=true;
			changeMoney=changeMoney-payMoney;
			x10KYenRequest=false;
				else
				STOP
				payMoney=0;
				payment=true;
				x10KYenRequest=false;
					/

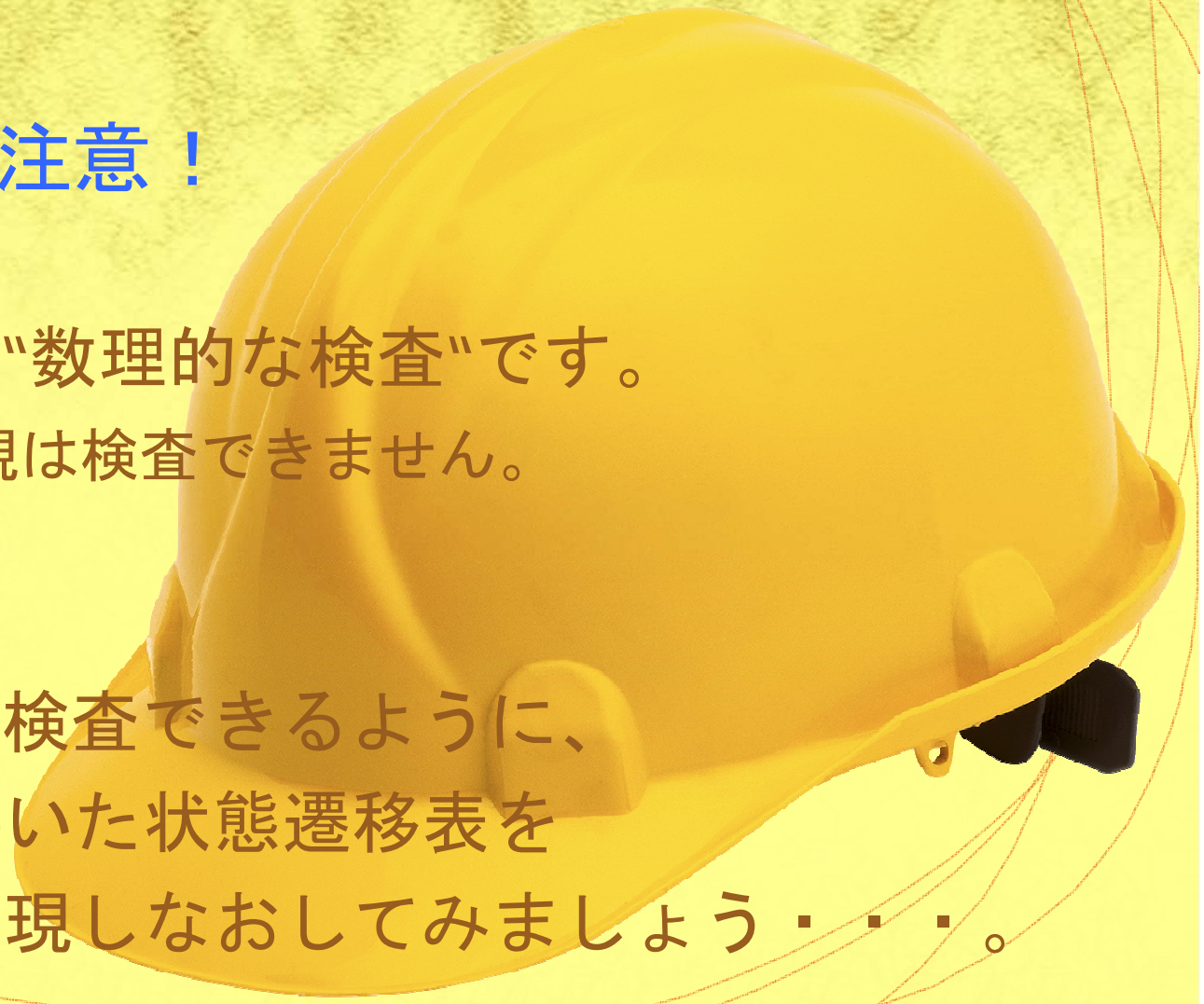
getMoney	2	/	×		WAIT_REQUEST getMoney=false;

両替したい人の状態遷移表

☐ RETURNER		WAIT	RETURN	
		0	1	
payment	0	RETURN payment=false;	×	

xRecieve	1	/	payMoney == 0
			WAIT
			getMoney=true;
			xReceive=false;
				else
				WAIT
				payMoney=0;
				getMoney=true;
				xReceive=false;

実践的なモデル検査をしてみよう！



★ちょっと、注意！

- モデル検査は"数理的な検査"です。
  - 文章的な表現は検査できません。

- では、モデル検査できるように、
  振舞いで書いた状態遷移表を
  数理的に表現しなおしてみましょう・・・。

実践的なモデル検査をしてみよう！

- 再び・・・、何が検査したかったんだっけ？

  検査したいのは、こんなコトでしたね。

  - 変な状態遷移でありえない状態に行かないこと。
  - コインが足りないときは、両替しない。
  - コインがなくなるまで、両替できれば両替する。
  - 紙幣一枚ずつしか両替できない機能を確認する。

- これらを検査するときには"検査性質"と言います。
  - 上記を検査性質式で表すと、こんな感じになります。
    - Garakabu2の場合：“不可セルフチェック”On
    - （changeMoney < 10000） => (payMoney == 0)
    - [G][F](changeMoney >= 10000 && payMoney == 10000）
    - [G][F](!getMoney)

実践的なモデル検査をしてみよう！

- んんっ？　なんだ？　この変な表現は？
  - ＝＞
    - Implies（∽のときに∽になる）
      - 例：ブレーキを踏むと減速する。
  - [F],[G]
    - 時相論理（LTL）
      - オートマトン（状態遷移表）には、時系列な変化（時間）の概念があります。
      - 時間は０．５次元（未来方向）なので、
        “現在から将来はこうなる”という検査を行うことができます。
        現在から将来を含めた検査表現を“時相論理”と言います。
        - 例：ブレーキを踏み続けると、いつか停止する。

LTL（Linear Temporal Logic）：
線形時相論理

現在＆未来

未来

実践的なモデル検査をしてみよう！

- 時相論理（LTL）で何が検査できる？

単項演算系
- [F]inally:　〇〇は将来のいずれかの時点でTrueとなる。
- [G]lobally:　〇〇は今後常にTrueである。

二項演算系
- [U]ntil:　〇〇は現在、未来の位置でTrueで、XXはそれ以前の位置まで有効でなければならない。
- [R]elease:　〇〇がTrueである最初の位置まで XX がTrueなら、〇〇は XX を開放する。

上記を組み合わせることで、現在から未来への状態遷移条件に対する色々な検査が出来ます。

例えば･･･
“商品がなくなるまで、販売できる。”
コレは・・・
[F](商品在庫数>0 => [G](販売受付窓口Flag))
のような表現ができます。

時相論理は数式表現なので
組合せは無限ですが、
“コレだけあれば足りるだろう”
というパターン集が有ります。
有名なのがDywerのパターン。
他に産総研（AIST)の図示記法など、
色々な研究成果があります。

実践的なモデル検査をしてみよう！

- では、実際に検査してみましょう！
  - モデル検査では、検査する性質に対してそぐわない条件が存在したときまでの状態遷移を
    “反例”と言います。
  - ここでは、
    - 反例が見つかった場合
    - 反例が見つからなかった場合
    の両方のパターンを見てみましょう。

では、ZIPCとGarakabu2を使って実際に検査してみましょう！

※なぜSPINでもSMVでもないか？
それは僕がGarakabu2を作っているからです…。

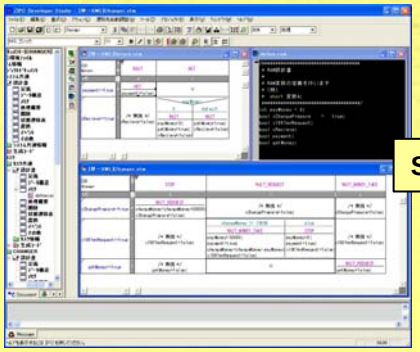
目次

- 形式検証ってなんだ？
- 実践的なモデル検査をしてみよう！
- Garakabu2ってなんだ？

Garakabu2ってなんだ？

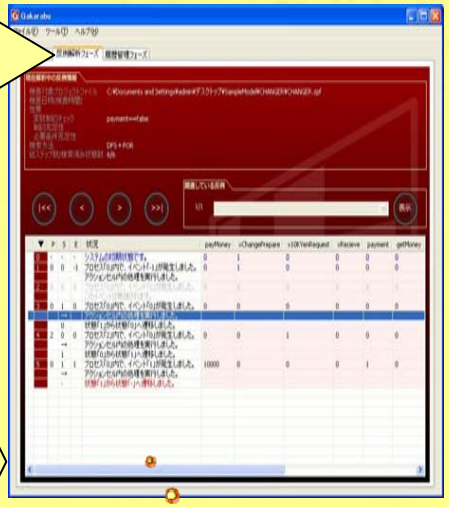
- Garakabu2 – 状態遷移表（ZIPC）を使用したモデル検査ツール

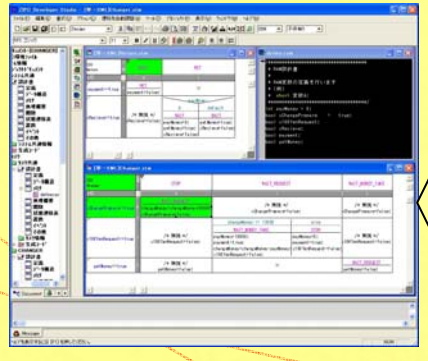
ZIPC



STMを読み込んで…

Garakabu2





反例追跡

検証して…

ZIPC ©
CATS Co. Ltd
・Embbeded CASE Tool
・STM-based MDD Tool

Garakabu2

SMT-based symbolic bounded model checking algorithms

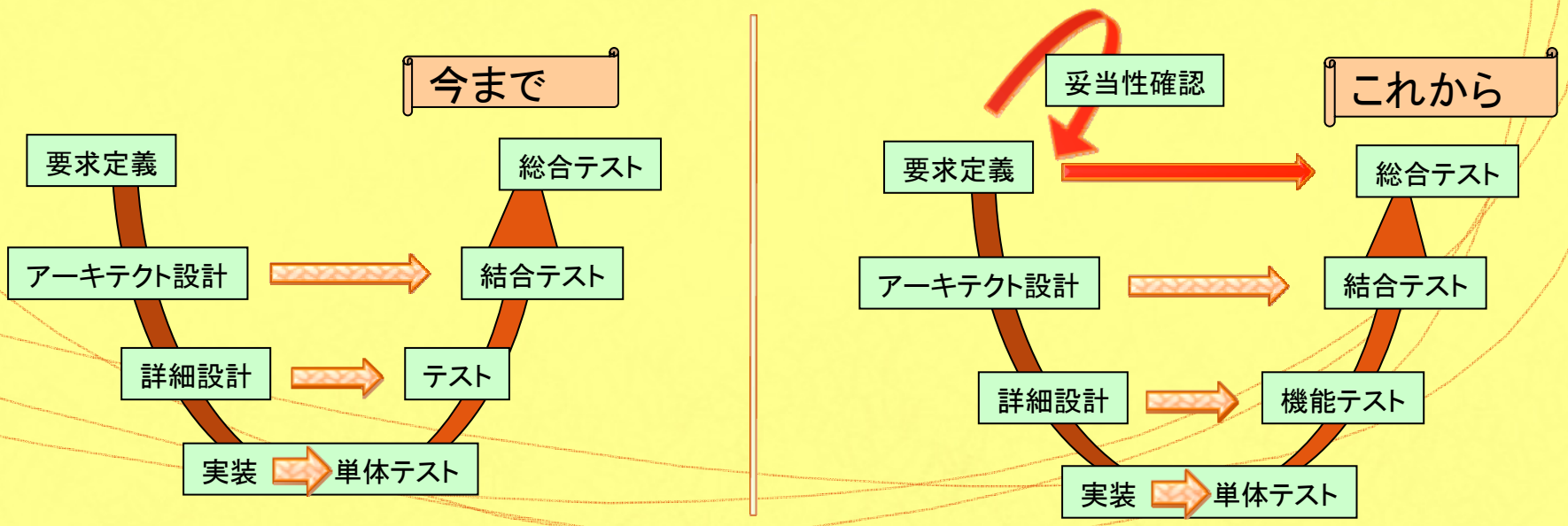
(1) Limited LTL formulas
(2) CVC3 (ver. 2.2)'s C++ API
(3) Linear problems with limited support for non-linear problems

Garakabu2 URL: http://garakabu.lab-ist.jp

Garakabu2ってなんだ？

なぜ検証（Verification)が必要なのか？

- 従来のソフトウェア開発における試験（Test)では、詳細設計仕様に基づいて作成されたロジックの正しさの確認になってしまい、元々の要求定義である"何がしたい（≒設計）"の正しさの確認と乖離している可能性がある。
- そこで、昨今、要求仕様に対する検証としてV&V（Verification&Validation=検証＆妥当性確認）が強く求められている。



Garakabu2ってなんだ？

検証技法としてのモデル検証

- まず振舞いモデル（オートマトン）として仕様を記ことにより、論理的な（数理的）検証を行うことができる。

- 組込みソフトウェアの機能安全確保手法として、各分野においてその検証が重要視されている。
（IEC61508が代表的な規約）

Garakabu2ってなんだ？

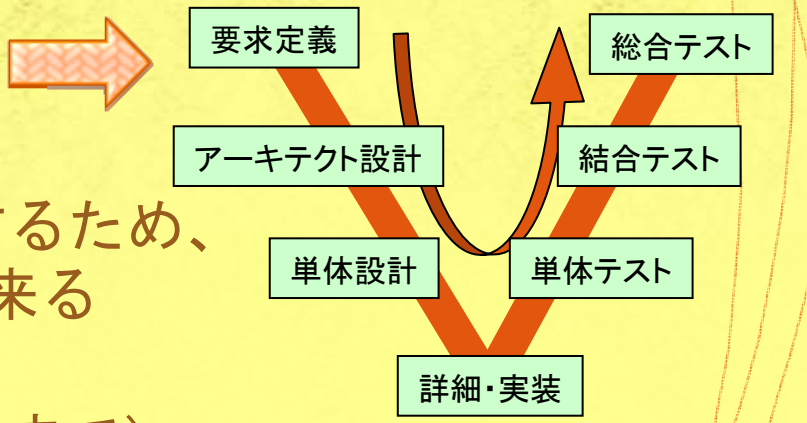
- 形式検証ツール一覧

ツール	モデル（形式）記述	特徴	出力検証結果	その他
Spin	Promela	オートマトン、LTL	標準出力（Text出力）	
muSMV	状態遷移図（SMVコード）	CTL、BMC対応 BDDによる記号モデル	標準出力（Text出力）	
UPPAAL	状態遷移図（c++風）	時間付きCTL 連続時間オートマトン	GUI表示（シーケンス図）	
Vdm	vdm-SL vdm++	形式手法ツール	VdmTools(GUI)	Vdm-SLはISO標準（15048,61508）
Garakabu2	状態遷移表（ZIPC）	LTL 有限界オートマトン	GUI表示（状態遷移表と同期）	言語的記述を行わない
CBMC	ANSI-C/c++	LTL 有限界オートマトン	GUI表示（eclipsePlugin）	

Garakabu2ってなんだ？

モデル駆動開発（MDD)との親和性

- 粒度の大きい仕様での検証手法
  - ➔上流工程での検証方法で
    V字開発全体に貢献（妥当性確認）
  - ➔上流工程でモデル化（形式化）するため、
    V字開発の左側が一貫して作業出来る
    （要求開発から、
    コード生成、シミュレーションまで）



- 要求仕様に詳細なタイミングは書けない（ことが多い）
  - ➔逆に考えれば、〝～のとき、その後～になる〟などを
    検証するには最適な作業フェーズである
  - ➔詳細なタイミングは別途下流で行うことで検証粒度を分ける

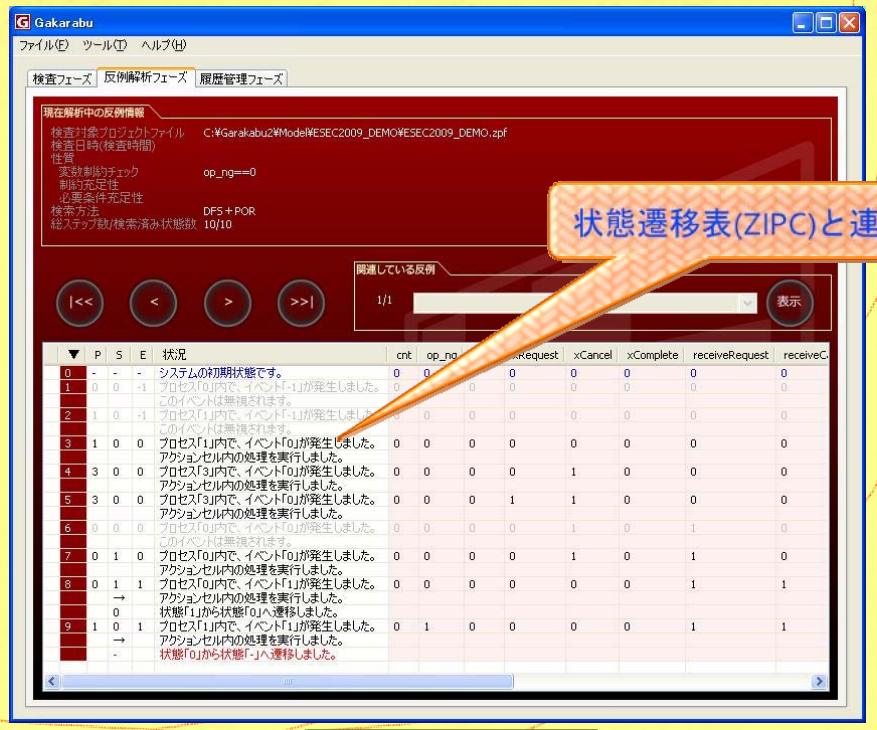
Garakabu2ってなんだ？

検証結果はどんなふうに見える？

- 不具合（反例）は、モデルに正しさとして修正しなければならない。
- フィードバックの容易さは、検証技法自体の使用頻度に関わってくる。

```
Starting Init:withpid(0)

Spin:warning" PSx.prom"  is newew than Psy.pron.aaa

Starting personA with pid 1

 1: proc0(:init) line 109  "" Psy.prom(state1) [run person1
      (person…

 2: proc0(:init) line 118  "" Psy.prom(state1) [run person2
      (p・・・・


1283: proc0(:init) line 118  "" Psy.prom(state1) [run
      user・・・・・
```

Spinの場合



Garakabu2の場合

Garakabu2ってなんだ？

フィードバックの容易さ
＝検証者の敷居の低さ

- SPINやVDMなど自然言語的な検証（形式記述）では、記述・検証結果が"見える化"していないため、言語的フィードバックとなってしまい、下流設計では有効だが、振舞いが見えづらい、解りづらい、という開発現場で適用しずらい面がある。
- Garakabu2,Uppaalなどの"振舞いモデル"をGUI化した図表化記述方式のツールでは、モデル自身、また、検証結果が視覚的に確認できるため、検証結果（反例）の修正が容易である。

まとめ

- 分かりやすいモデリング
  - ➔形式的に検証するためにはモデリングが必要
  - ➔自然言語的なものより"見える化"されているほうが仕様が分かりやすい
- 検証の重要性
  - ➔肥大化していくソフトウェアに対して、今後、上流工程での検証が重要性を増す
  - ➔ロジックの正しさではなく、要求仕様への正しさが重要
- 検証結果とモデル修正
  - ➔不具合（反例）を見つけても、モデル修正が難しくては現実的な開発手法として普及の障害になってしまう
  - ➔不具合の修正の容易さが検証・モデリングともに今後の課題