モデル検査とは

株式会社フォーマルテック

1. 概要（形式手法のなかのモデル検査）

モデル検査は形式手法（数理的技法）の１つです。
形式検証／フォーマル検証とも呼ばれています。

形式手法とは → 特定の手法を指すわけではなく、
→数学や論理学を基盤としたシステムの記述方法や検証手法の総称です。

形式手法は上記３つの手法に分類されます（VDM、SMV等はSWツールの名前です）。
→モデル検査はツールによる全自動化が最も進んだ手法です。

1. 概要（モデル検査の特徴）

モデル検査の最大の特徴（メリット）は？

システムが取り得る「全ての状態」と「全ての実行パス」を

- 全自動で
- 網羅的に検査して
- 不具合に至るパスを出力する

ことです。

2. 形式手法と国際規格（IEC61508）

機能安全に関する 国際規格IEC61508 で推奨されています。

■ＩＥＣ(国際電気標準会議)が２０００年に制定した国際規格
　国内ではＪＩＳ Ｃ 0508が対応

■対象：コンピュータ技術を用いた安全関連系を使用する全ての産業分野
　極めて広範囲：プロセス産業、機械、医療機器、鉄道、自動車、航空宇宙、原子力

■電気・電子・プログラマブル電子技術を用いた安全関連系の安全性能を４つの「安全度水準」(Safety Integrity Level：SIL) で区分
　→リスク解析、設計、実装、運用、保守、廃棄に至る規範的手順を提示

ＳＩＬ４では形式手法を強く推奨

2. 形式手法と国際規格（IEC61508）

IEC61508（機能安全）では、

□Software safety requirements specification

□Software design and development：detailed design

において、安全度水準SIL2以上でFormal methodの適用が推奨されています。

Software safety requirements specification				
Technique/Measure	SIL1	SIL2	SIL3	SIL4
1 Computer-aided specification tools	R	R	HR	HR
2a Semi-formal methods	R	R	HR	HR
2b Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	—	R	R	HR
Software design and development: detailed design				
Technique/Measure	SIL1	SIL2	SIL3	SIL4
1a Structured methods including for example, JSD, MASCOT, SADT and Yourdon	HR	HR	HR	HR
1b Semi-formal methods	R	HR	HR	HR
1c Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	—	R	R	HR
2 Computer-aided design tools	R	R		HR
3 Defensive programming	—			HR
4 Modular approac				
5 Design and codin				
6 Structured programming	HR	HR	HR	HR
7 Use of trusted/verified software modules and components (if available)	R	HR	HR	HR

SIL4ではHR(High Recommended)

2. 形式手法と国際規格（ISO26262）

ISO 26262（機能安全の自動車版）では、
□Product development at the software level
において、自動車安全整合性ASIL C以上で形式検証が推奨されています。

No.	ソフトウェアユニットの設計及び実装の検証手段	ASIL			
		A	B	C	D
1a	ウォークスルー	++	+	o	o
1b	インスペクション	+	++	++	++
1c	準形式検証	+	+	++	++
1d	形式検証	o	o	+	+
1e	制御フロー解析	+	+	++	++
1f	データフロー解析	+	+	++	++
1g	静的コード解析	+	++	++	++
1h	セマンティクスコード解析	+	+	+	+

3. モデル検査のワークフロー

以下の手順で検査します。

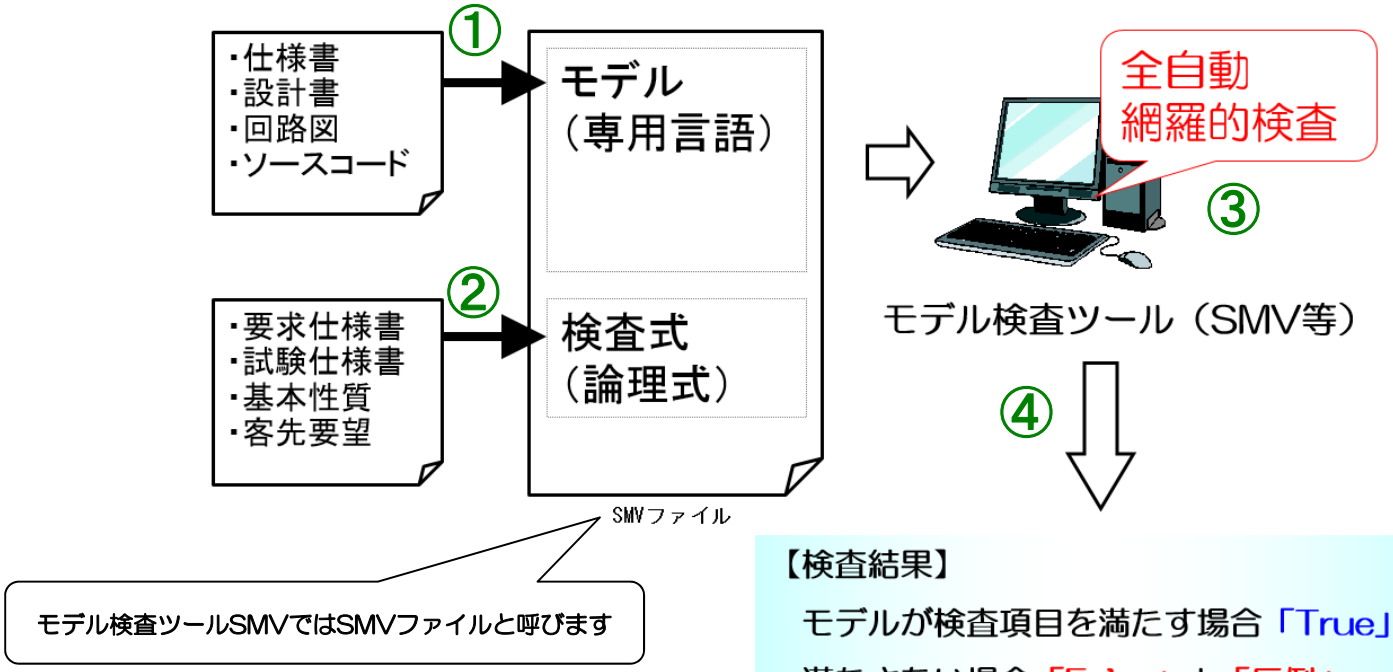
① 検査したいもの（仕様書やソースコード等）から専用言語でモデルを作成します。
② 仕様書やソースコードが満たすべき性質から検査式を作成します。

} 1つのファイルに記述します。

人手で作業するのはここまでです。

③ 作成したファイルを入力すると、モデル検査ツールが全自動で網羅的検査を行ってくれます。

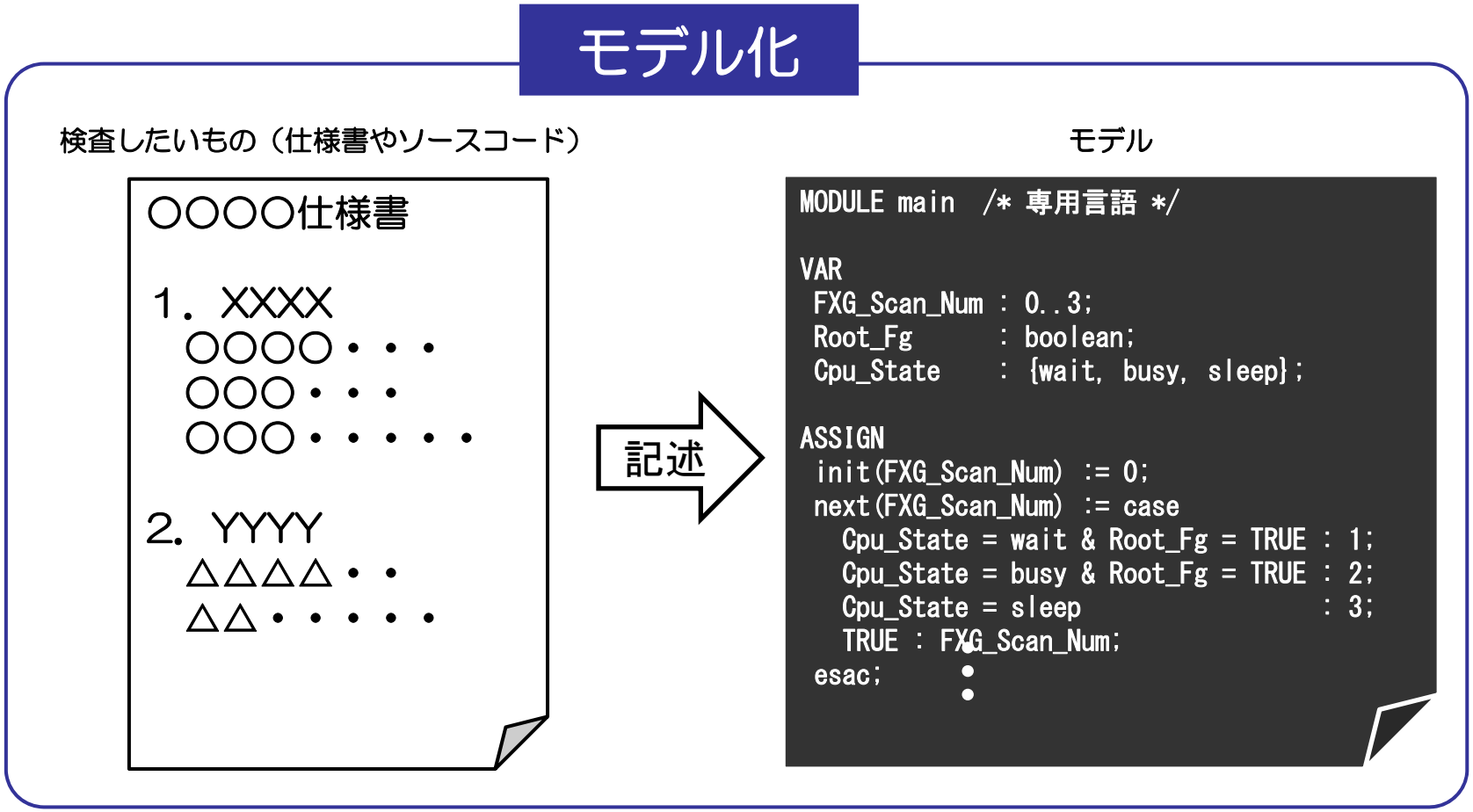
④ モデル検査ツールから検査結果が出力されます。



4. モデルについて

モデルの形式

モデル検査ツールの専用言語で記述したテキスト形式のものです。



5. 検査式について

検査式は

モデルが満たすべき性質を論理式で記述したものです。

モデル検査ではAGやEFなどの特殊な記号を使います。

検査式の記述例

「セマフォAとセマフォBが同時に1となることはない」

→ !EF(Semaphore_A = 1 & Semaphore_B = 1)

「割り込みが発生した場合、必ずスリープ状態を抜ける」

→ AG(Interrupt = ON -> AF(Sleep = Off))

「Qが発生すればRが発生する前に、Pが発生してSが発生する」

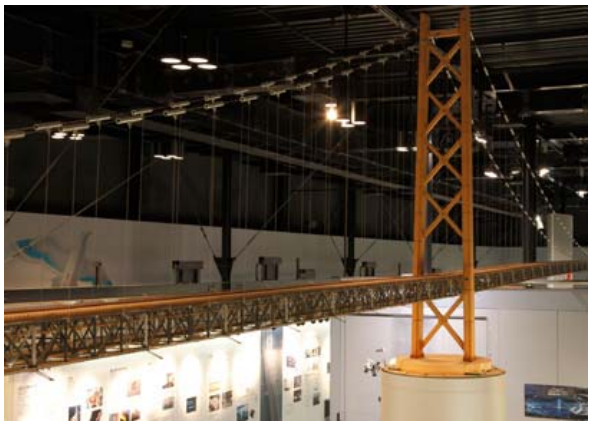
```
→ AG(Q & !R ->
       !E[!R U (!(P -> A[!R U (S & !R)]) & !R)])
```

EF : Exist Future　AG : Always Globally・・・

6. モデル検査で不具合（バグ）が見つかる原理

不具合が見つかる原理

モデル



実物

モデルを検査する

→モデルで不具合を発見

モデルの元(ソースコード／仕様書等)
にも不具合があるはず！

7. 反例について

反例とは？

不具合に至るまでの経路（パス：変数の時系列の変化）です。

モデル検査ツール（SMV等）

⇩

【検査結果】

モデルが検査項目を満たす場合「True」

満たさない場合「False」＋「反例」

```
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  SELECT = emp
  SEMAPHORE = free
  TASK_A = wait
  TASK_B = wait
  TASK_C = wait
-> State: 1.2 <-
  SELECT = task_c
-> State: 1.3 <-
  SELECT = task_b
  TASK_C = load
-> State: 1.4 <-
  SELECT = task_a
  SEMAPHORE = task_c
  TASK_B = load
  TASK_C = exe
-- Loop starts here
-> State: 1.5 <-
  SELECT = task_b
  TASK_A = load
  TASK_B = exe
```

8. モデル検査の作業（おさらい）

作業は２つだけ → モデルの作成と検査式の作成です。

仕様書
設計書
ソースコード

「モデル化」

検査したいこと

「検査式作成」

SMVファイル

```
MODULE main  /* 専用言語 */

VAR
 FXG_Scan_Num : 0..3;
 Root_Fg      : boolean;
 Cpu_State    : {wait, busy, sleep};

ASSIGN
 init(FXG_Scan_Num) := 0;
 next(FXG_Scan_Num) := case
      ●
      ●
      ●



SPEC
 !EF(EG(Kasoku = 0 & Gensoku = 1 & Hyoji = 1))
```

9. モデル検査で発見しやすい不具合

モデル検査の網羅的検査で以下のような不具合を発見できます。

□微妙なタイミングで発生する不具合

- 割込み -> 想定外のタイミングで発生する・・・
- 機器の故障 -> いつ発生するか分からない・・・
- 人間の操作 -> ユーザは何をするか分からない・・・
- JavaVMのスレッド切替 -> いつ発生するか分からない・・・

□複数の条件が偶然一致して発生する不具合

- 複数の設定値 -> 設定項目の組合せは膨大・・・
- 複雑なアルゴリズム（If文とLoop文のネスト） -> ちゃんと設計したつもりだが・・・

□設計時の漏れor抜けが原因で発生する不具合

- 記述漏れ -> 場合分けの不足など・・・

□非同期処理の不具合

- 二重系 -> 相互に関与して振る舞いが複雑・・・
- プロセス -> 人間の頭では制御しきれない・・・

10. モデル検査が苦手な問題

モデル検査には苦手な問題もあります。

□アナログ（連続）値の変化や誤差に起因する問題

問題自体は検出できないがモデル化は工夫（抽象化と絞込）すれば可能です。

- モータの回転数 → 段階値（停止／低速／中速／高速）にします。
- パネルの開度　→ こちらも10段階などに抽象化します。

□巨大な状態空間を持つ大規模なシステムの問題

全てをモデル検査すると状態爆発が発生してしまいます。

→ 中枢部分や重要なサブシステムだけ検査します。

11. モデル検査の課題と対策

技術的な課題と運用面での課題と対策を紹介します。

■状態爆発

網羅的検証を行うが故にモデルの規模が大きくなると、
現実的な時間でツールの実行が終了しない、あるいは
メモリを消費し尽してしまう問題です。
→ 対策：抽象化／絞り込みを行います。

■モデル化のコスト

手作業によるモデル化はコストが増大します。
→ 対策：ツールによる自動化が王道です。

■独学での導入が困難

他の技術に比べて独学が難しいと言われています。
→ [汝、形式手法の師匠を身近に持つべし]（形式手法十戒の一節）
→ 対策：専門家からしっかり学びましょう。

12. 従来手法との比較

従来手法（テスト・シミュレーション）との比較表です。

項目	モデル検査	テスト・SIM
自動化	ツールにより実現	可能だが人手による設定が必要
網羅性	全状態網羅	部分的
厳密生	厳密	不確実
検査対象の規模	大規模は困難	大規模でも可能
コスト	モデル化のコスト大	一般的に小
事例	少	多

製品の品質を重視すると、
メリット ≫ デメリットではないでしょうか！？