SWEST13 セッション(S5) 2011年9月2日（金） 於　下呂温泉 水明館

目でみてわかるモデル検査

株式会社フォーマルテック
早水 公二

／ 第１部：【概要】

1. モデル検査とは？

形式手法（数理的技法）の１つ

形式手法とは → 特定の手法を指すわけではない
→数学・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

代表的な3つの手法

形式仕様記述 VDM，Z，B等　／　モデル検査 SMV，SPIN等　／　定理証明 Agda等

1. モデル検査とは？

形式手法 → システムをフォーマルに記述あるいは検証する

日本語(自然言語)による仕様・設計 → 理解し易いが「曖昧」

⇨ 論理式／数式による仕様・設計 → 厳密に規定できる

形式仕様記述

仕様の証明は 定理証明 で定理／公理を使って証明する。

テストパターン抽出による動作試験 → 簡単だが「抜け・漏れ」

⇨ 全状態を探索して性質を確認 → しらみつぶしの検査が可能

モデル検査

1. モデル検査とは？

機能安全に関する 国際規格ＩＥＣ６１５０８ で推奨

■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ＪＩＳＣ0508が対応

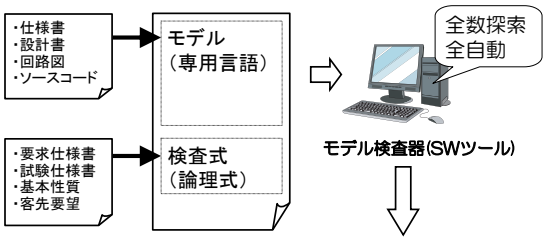
■対象：コンピュータ技術を用いた安全関連系を使用する全ての産業分野
極めて広範囲：プロセス産業、機械、医療機器、鉄道、自動車、航空宇宙、原子力

■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

1. モデル検査とは？

「専用言語」で記述されたシステムを全自動で全数探索する



【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」

1. モデル検査とは？

モデルとは？

仕様書／ソースコードをモデル検査器毎の専用言語で記述したもの

モデル化

○○○○仕様書

1. XXXX
   ○○○○・・・
   ○○○・・・
   ○○○・・・・・

2. YYYY
   △△△△・・
   △△・・・・・

記述

```
MODULE main  /* モデル検査器毎の専用言語 */

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
   Cpu_State = wait & Root_Fg = TRUE : 1;
   Cpu_State = busy & Root_Fg = TRUE : 2;
   Cpu_State = sleep                 : 3;
   TRUE : FXG_Scan_Num;
 esac;
```


1. モデル検査とは？

検査式とは？

検査したい性質を論理式（CTL式）で記述したもの

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)])

1. モデル検査とは？

不具合が見つかる原理

モデルを検査する
→モデルで不具合を発見

モデルの基(ソースコード／仕様書)にも不具合がある

1. モデル検査とは？

反例とは？

不具合に至るまでの経路（パス：時系列な変数の値の変化）

モデル検査器(SWツール)

【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
```


1. モデル検査とは？

作業は２つ → モデルの作成と検査(CTL)式の作成

仕様書
設計書
ソースコード

「モデル化」

検査したいこと
不具合事象

「CTL化」

SMVファイル

```
MODULE main  /* モデル検査器毎の専用言語 */

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))
```


1. モデル検査とは？

デモ

1. モデル検査とは？

モデル検査に関する書籍


「モデル検査：基礎から実践まで4日で学べる（初級編）」
産業技術総合研究所システム検証研究センター 著


「モデル検査：実践のための三つの技法（上級編）」
産業技術総合研究所システム検証研究センター 著


「SPINモデル検査 ― 検証モデリング技法」
中島 震 著

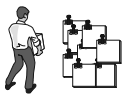
モデル検査に関する講義・セミナー

- ☑ 組込み適塾 http://cfv.jp/cvs/learning/learning01.html
- ☑ トップエスイープロジェクト http://www.topse.jp/
- ☑ 日本科学技術連盟主催（不定期）

2. 企業での普及活動

開発現場では～

数百頁規模の仕様書
数万行のソースコード



検査したい！

32bit対応のモデル検査器では限界！

- ☑ メモリ上限 → 4GByte
- ☑ 動作保証無し
- ☑ 障害対応 → 開発元にお任せ
- ☑ 配布／改良の中断も・・・

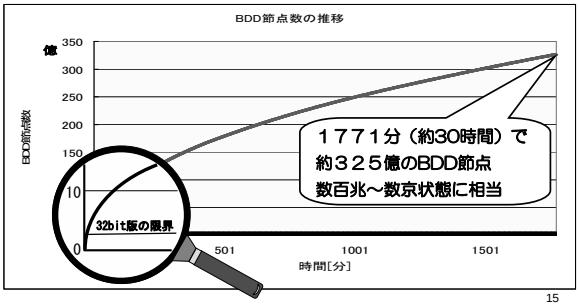
64bit版モデル検査器※
新規開発
＋
独自の効率化手法

※関西電力株式会社からの委託研究開発

2. 企業での普及活動

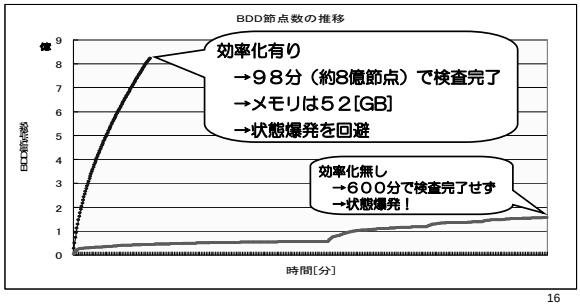
64bit版モデル検査器の動作確認

産業技術総合研究所検証クラスター「さつき」(メモリ：1TB) を利用



2. 企業での普及活動

64bit版モデル検査器を
実製品のソフトウェアのモデル検査に適用 （「さつき」利用）



3. モデル検査Webサービスの紹介

６４ｂｉｔ版
モデル検査器

＋

大容量メモリ
搭載の計算機

⇩

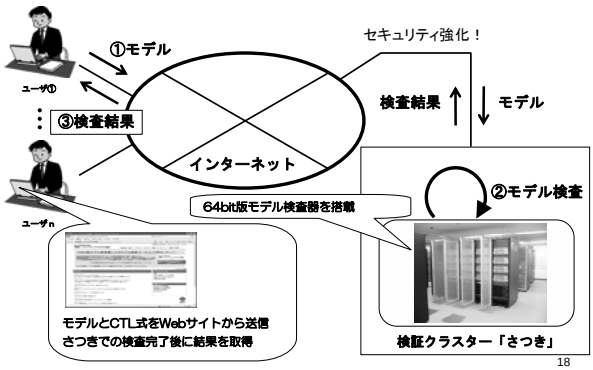
モデル検査によるソフトウェアテストの実践研究会
関西電力電力技術研究所
株式会社フォーマルテック
他　企業１社　大学１

AIST 独立行政法人 産業技術総合研究所
検証クラスター「さつき」
関西産学官連携センター
組込みシステム技術連携研究体
Collaborative Facilities for Verification and Specification

いつでも誰でも利用できるWebサービスとして公開

※関西電力株式会社からの委託研究開発

3. モデル検査Webサービスの紹介



3. モデル検査Webサービスの紹介

デモ

3. モデル検査Webサービスの紹介

本システムのメリット

■モデル検査器のインストール不要
→ PCへのSWのインストールを制限している企業には◎

■ライセンス問題を考える必要がない
→ 公開されていても「商用利用禁止」の
モデル検査器もあるので注意要

■大容量メモリ（さつき）を利用できる

■いつでもどこからでも手軽にモデル検査が利用できる

■64bit版モデル検査器を利用できる（本システムで初公開）
→「 64bit版＋大容量メモリ」で状態爆発の回避に有効

■無料

3. モデル検査Webサービスの紹介

セキュリティについて

■匿名で利用可能
→ お名前・会社名・連絡先・業種等は入力不要

~~検査太郎~~
~~メールアドレス~~
~~TEL番号~~

■通信はSSLで暗号化

■処理は全自動なので人手は介さない
→ モデルや検査結果の閲覧・チェックは一切無し

■モデルと検査結果はご利用後に全て削除
→ 「Webサーバ」「さつき」にはファイルは一切残らない

／ 第２部：【例題】

1. 例題1

タスクの優先度と排他制御のシステム

１．タスクAとタスクBとタスクCが存在する
２．タスクはランダムに１つずつロードされる
３．同じタスクが同時にロードされることはない
（例　タスクAが２つロードされることはない）
４．タスクの優先順位は高順に「タスクA→タスクB→タスクC」とする。
順位の高いタスクが実行を開始すると、より順位の低いタスクの実行を中断し、高いタスクが実行完了するまで実行される。
５．タスクAとタスクCは１つの資源を共有し互いに排他制御を行う。
両タスクともロード後に資源獲得できれば実行され、
実行完了後に資源を解放する。

1. 例題1　検査式①

①タスクBが実行中にタスクAがロードされると、タスクAは必ず完了すること

→
```
SPEC
  AG(TASK_B = exe & TASK_A = load -> AF(TASK_A = end))
```

AG（P　->　AF(Q)）
常に　ならば　将来必ず

「常に、Pが成立するならば必ずQが成立する」

2. 例題2

状態遷移表の検査

	STATE_A		
EVENT	sa_1	sa_2	sa_3
1	sa_2	sa_1	sa_1
	-	fg1 = TRUE	fg2 = TRUE
3	sa_3	-	-
	-	fg2 = TRUE	fg1 = FALSE
5	-	sa_3	sa_2
	fg2 = FALSE	-	fg1 = FALSE

	STATE_B		
EVENT	sb_1	sb_2	sb_3
2	sb_1	sb_3	sb_2
	-	fg1 = FALSE	fg2 = FALSE
4	sb_2	-	-
	-	fg2 = FALSE	fg1 = TRUE
6	-	sb_1	sb_1
	fg2 = TRUE	-	fg1 = TRUE

1．状態の初期値は　→　STATE_A = sa_1　STATE_B = sb_1
2．イベント(EVENT)は初期値は0　以後はランダムな値となる
3．フラグの初期値は　→　fg1 = FALSE　fg2 = FALSE
→【検査】fg1とfg2が共にTRUEとなった後は、共にFALSEにはならない？

2. 例題2の検査式

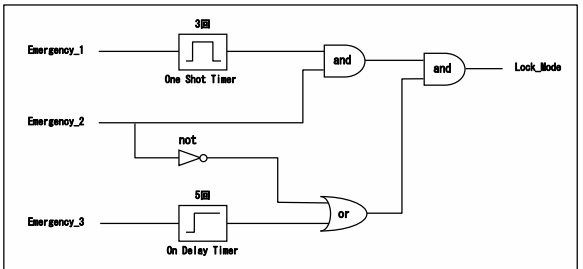
fg1とfg2が共にTRUEとなった後は、共にFALSEにはならない？
→

```
SPEC
  SPEC AG((fg1 = TRUE & fg2 = TRUE) -> !EF(fg1 = FALSE & fg2 = FALSE))
```

AG (P -> !EF(Q))
常に　ならば　将来～することがある
「常に、Pが成立するならばQが成立することはない」

3. 例題3

回路図の検査



★ロックモードになるまでの入力列が知りたい。
→【検査】ロックモード（Lock_Mode）になることがあるか？

3. 例題3の検査式

ロックモード（Lock_Mode）になることはない
→

```
SPEC
  !EF(LM)
```


Ref. 検査項目（検査式）

SMV系のモデル検査器ではCTL(Computation Tree Logic)式

基本パターン	意味
AG(P)	全てのパスにおいて、全ての状態でPが成立する
AX(P)	全てのパスにおいて、次の状態でPが成立する
AF(P)	全てのパスにおいて、将来のある状態でPが成立する
A[P U Q]	全てのパスにおいて、 Qが成立する状態より前の状態までPが成立する状態が続く
EG(P)	あるパスにおいて、全ての状態でPが成立する
EX(P)	あるパスにおいて、次の状態でPが成立する
EF(P)	あるパスにおいて、将来のある状態でPが成立する
E[P U Q]	あるパスにおいて、 Qが成立する状態より前の状態までPが成立する状態が続く

入れ子構造もOK
★式を作るのはモデル検査技術　何を検査するかはドメイン技術

Ref. 検査項目（検査式）

よく使う検査式

AG (AF(P)) 常に 将来必ず	「常に、将来必ずPが成立する」
AG (P -> AF(Q)) ならば	「常に、Pが成立するならば将来必ずQが成立する」
!EF(P) 将来～であることがある	「将来Pが成立することはない」
!EF(EG(P)) ～であり続けることがある	「将来、Pであり続ける状況が成立することはない」
AG (P -> A[Q U R]) 将来必ずRまでQが成立	「常に、Pが成立するならば将来必ずRまでQが成立する」

上記の検査式である程度の検査が可能