

---

---

# 組み込みソフトウェア検証への モデル検査技術の導入に関する技術動向

---

横川 智教

t-yokoga@cse.oka-pu.ac.jp

岡山県立大学

# 講師紹介

- 所属

- 岡山県立大学
  - › 情報工学部情報システム工学科
  - › 回路デザイン研究室・助教

- 最近の研究テーマ

- モデル検査による組み込みソフトウェアの設計検証
- 非同期回路のモデル化・性能評価・検証

- 連絡先

- [t-yokoga@cse.oka-pu.ac.jp](mailto:t-yokoga@cse.oka-pu.ac.jp)
- <http://circuit.cse.oka-pu.ac.jp/>

# 本チュートリアル概要

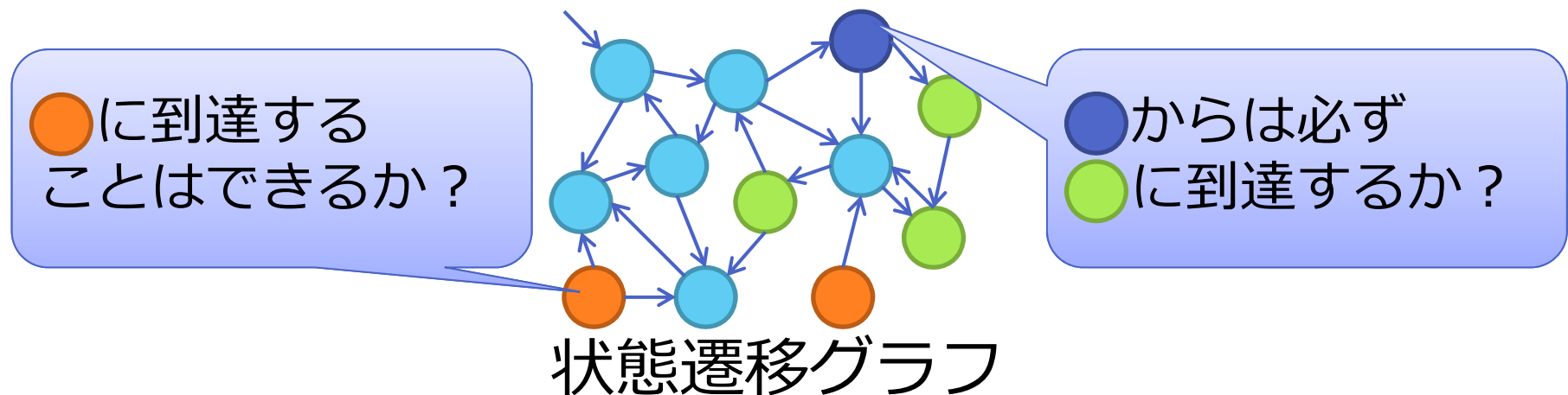
- はじめに – モデル検査とは
  - 形式手法とは？
    - › どんな手法がある？
    - › どんなツールがある？
- 組み込みソフトウェア開発への導入事例
  - モデル検査の導入事例
  - そのほかの形式手法の導入事例
- モデル検査によるUML設計検証支援ツール
  - ツールの概要
    - › ツールを利用するための準備
    - › ツールの使い方について
  - 適用事例

# 本チュートリアル概要

- はじめに – モデル検査とは
  - 形式手法とは？
    - › どんな手法がある？
    - › どんなツールがある？
- 組み込みソフトウェア開発への導入事例
  - モデル検査の導入事例
  - そのほかの形式手法の導入事例
- モデル検査によるUML設計検証支援ツール
  - ツールの概要
    - › ツールを利用するための準備
    - › ツールの使い方について
  - 適用事例

# モデル検査

- 形式手法(formal method)の一種
  - システムの自動検証技術
  - システムを状態遷移グラフでモデル化し, モデルの網羅的探索によって求める特性を満たすか否かを自動的に判定



# 形式手法

- 形式手法とは
  - システム開発に数学・論理学を導入するための技術
    - › 手続きの自動化による品質向上・コスト削減
- 形式手法の利用
  - 仕様記述
    - › 集合論に基づいてデータ構造を記述
    - › 述語論理を用いてシステムの要求を表現
  - 開発
    - › プロセス代数論により要求と設計の等価性判定
  - 検証
    - › 公理によるプログラム証明
    - › グラフ探索アルゴリズムによる特性検証

# 主な形式手法

- 形式仕様記述(formal specification)
  - システムの仕様を集合論や論理式を用いて数学的に表現するための記法
  - 抽象的かつ厳密な記述
- 形式的証明(formal reasoning)
  - 形式的な手続きに基づいてシステムの正しさを証明するための技術
  - 定理証明とモデル検査に分類される

# 形式的証明

- 定理証明(theorem proving)
  - 公理と推論規則に基づいてシステムの性質を証明
  - 完全に自動化することは困難とされる
  - 証明の自動生成を支援するツールは存在
- モデル検査(model checking)
  - 状態遷移グラフでモデル化されたシステムの網羅的探索に基づいて求める性質を満たすか検証
  - 完全に自動化された手続き
    - › 有限状態システムのみが対象
  - 状態爆発への対処が課題



# 主な形式仕様記法

- Z
  - 集合論, 述語論理,  $\lambda$ 計算の記法に基づいてプログラムの仕様と振る舞いを記述する
- B, Event-B
  - 段階的詳細化により仕様記述からプログラム生成を行う手法(Bメソッド)において用いられる
- CSP
  - プロセス代数論に基づいて, 並行動作するプロセス群の相互作用を記述する
- SCADE
  - 同期型プログラミング言語であり, 時間に同期して動作するリアルタイムシステムの記述が可能

# 主な定理証明支援ツール

- PVS

- 集合論に基づく仕様記述言語と検証ツールによって構成
- 対話的な定理証明が可能

- Coq

- 定理証明支援言語
- Coq言語で記述したプログラムに対する証明が可能

# 主なモデル検査ツール(1)

## ● SPIN

- 分散システムを対象としたモデル検査ツール
- 時相論理LTLで検証する特性を記述
- システムと特性をオートマトンでモデル化

## ● FDR

- CSPモデルを対象としたモデル検査ツール
- プロセスの模倣性検証が可能

## ● VDM-Tools

- 形式仕様記法VDM-SL,VDM++で記述された仕様の検証や実行が可能

# 主なモデル検査ツール(2)

## ● SMV

- 記号モデル検査アルゴリズムに基づいた高速な検証が可能
- 専用言語(SMV言語)で検査対象システムを記述
- 時相論理CTLで検証する特性を記述

## ● UPPAAL

- リアルタイムシステムを対象としたモデル検査ツール
- 時間オートマトンで検査対象システムを記述
- 時相論理TCTLで検証する特性を記述

# 本チュートリアル概要

- はじめに – モデル検査とは
  - 形式手法とは？
    - › どんな手法がある？
    - › どんなツールがある？
- 組み込みソフトウェア開発への導入事例
  - モデル検査の導入事例
  - そのほかの形式手法の導入事例
- モデル検査によるUML設計検証支援ツール
  - ツールの概要
    - › ツールを利用するための準備
    - › ツールの使い方について
  - 適用事例

# モデル検査の導入事例

## ● SPIN

- 日立ソリューション社, Blu-Rayディスク制御ミドルウェア
- Selex Communications社, 船舶通信システム

## ● FDR

- Philips Healthcare社, X線CTスキャンシステムの制御ソフトウェア

## ● VDM-Tools

- フェリカネットワークス社, Felicaチップ

## ● UPPAAL

- JAXA, 実行衛星の姿勢制御ソフトウェア

# 定理証明の導入事例

## ● PVS

- Logica Nederland B.V.社, 運河の水門開閉の意志決定システム
  - › 仕様記述にはZを導入
  - › 制御構造の検証にSPINを導入

# 形式仕様記述の導入事例

- Z記法
  - Praxis社, 航空管制システムiFACT
  - PMES社等, 艦載ヘリコプタ運行システム
- B, Event-B
  - STS社, 地下鉄列車制御システム
  - ClearSy社, 地下鉄プラットフォームドア制御システム
- CSP
  - Altran Praxis社, ICカード認証システム
- SCADE
  - Airbus社, 航空制御システム
  - POSCON社, 地下鉄プラットフォームドア制御システム



# 本チュートリアル概要

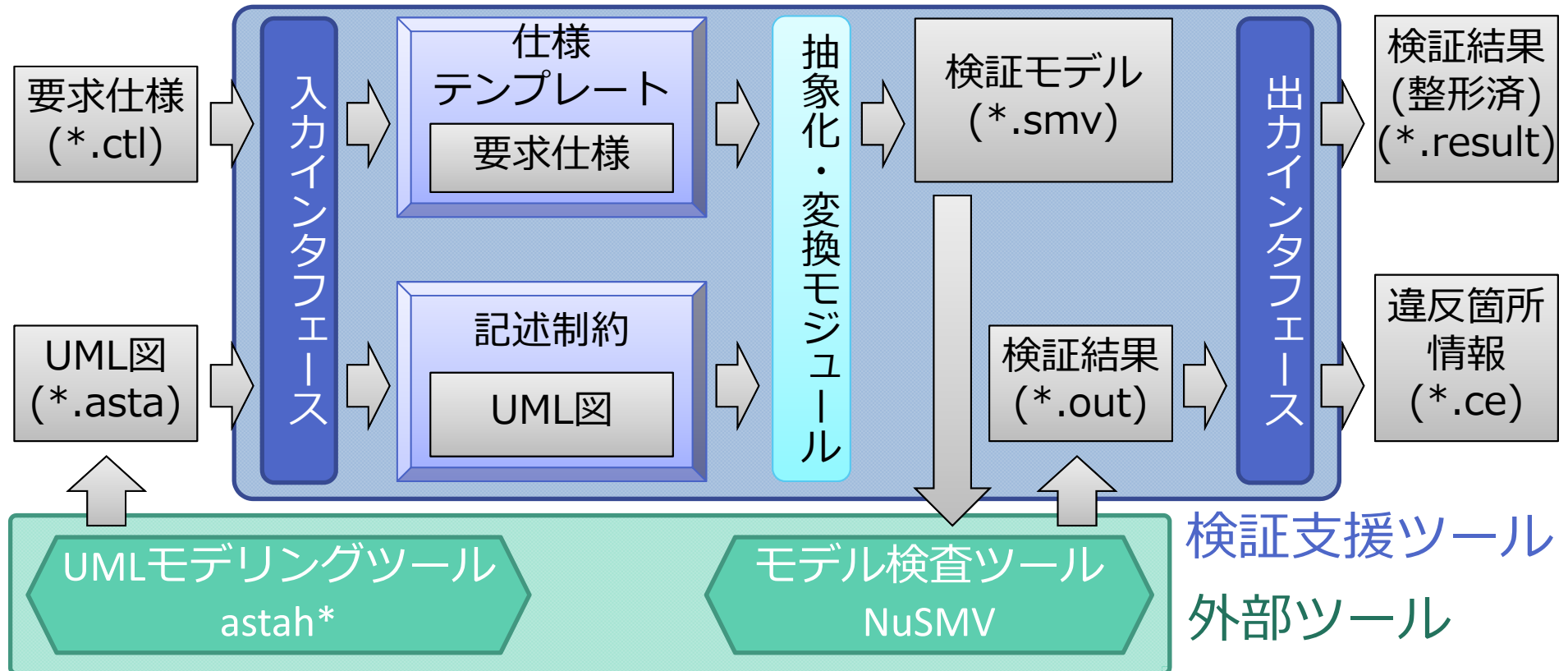
- はじめに – モデル検査とは
  - 形式手法とは？
    - › どんな手法がある？
    - › どんなツールがある？
- 組み込みソフトウェア開発への導入事例
  - モデル検査の導入事例
  - そのほかの形式手法の導入事例
- モデル検査によるUML設計検証支援ツール
  - ツールの概要
    - › ツールを利用するための準備
    - › ツールの使い方について
  - 適用事例

# 検証支援ツール

- モデル検査の導入コスト削減を目的として、IPA/SECが実施した2013年度RISE事業の支援を受けて開発
- UMLの状態マシン図・シーケンス図による設計を対象
- モデル検査ツールNuSMVで検証を行うための入力モデルを自動生成
- 現在のバージョンはコマンドラインでの動作のみをサポート
- 当研究室のWebサイトにて公開中
  - <http://circuit.cse.oka-pu.ac.jp/tool.html>

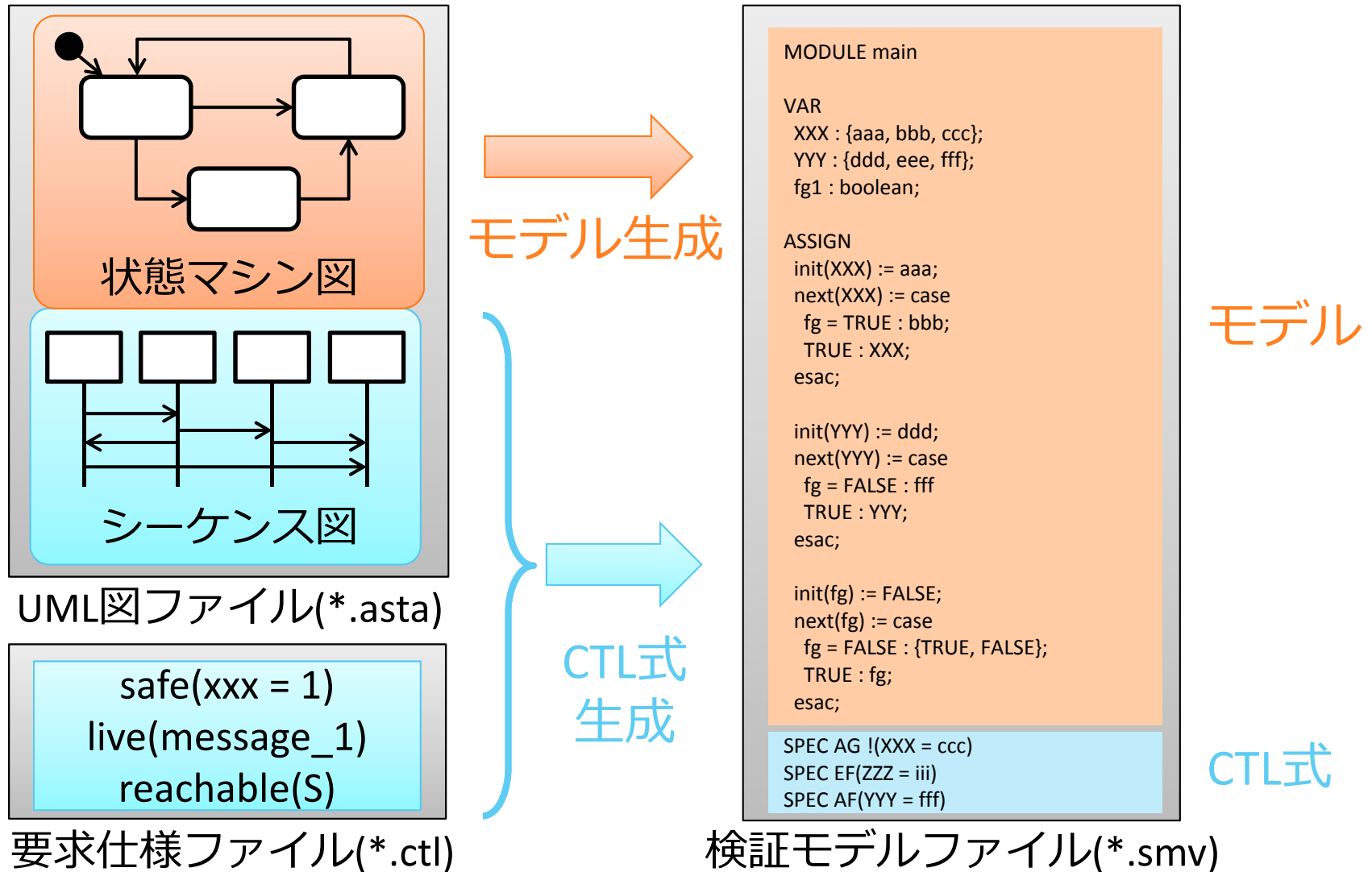


# 概要



- UML図の記述には**astah\***を，検証には**NuSMV**を用いる
- UML図が満たすべき**記述制約**と，要求仕様を記述するための**仕様テンプレート**が定められている

# 検証モデル生成の流れ



# ツールの導入準備

- 対象OS

- Windows7(64bit), Windows8(64bit), Mac OS 10.7

- 実行環境

- Java 7のインストール
- astah\* community 6.7のインストール
- NuSMV 2.5.4のインストール
- 検証支援ツールのインストール
  - › ダウンロードして解凍

# ツールの使い方

1. UMLモデルの作成(astah\*)
2. 要求仕様の記述(テキストエディタ)
3. SMVファイルの生成(検証支援ツール)
4. SMVファイルの修正(テキストエディタ)
5. モデル検査の実行(NuSMV)
6. 検査結果の整形(検証支援ツール)

# UMLモデルの作成

- astah\*を用いて作成
  - 状態マシン図・シーケンス図が対象
- 記述制約
  - 状態マシン図
    - › 単純状態およびその遷移のみを扱う
    - › 変数は整数型・ブール型のみを扱う
  - シーケンス図
    - › 結合フラグメントによる階層構造をもたない, 単純な相互作用のみを扱う

# 要求仕様の記述

## ● 仕様テンプレートを用いて記述する

No.	特性	要素	式表現	意味
1	安全性	状態	safe(s)	決して状態sはアクティブにならない
2		メッセージ	safe(m)	決してメッセージmは送信されない
3		変数	safe(x,a)	決して変数xの値がaとならない
4	活性	状態	live(s)	いつか必ず状態sがアクティブとなる
5		メッセージ	live(m)	いつか必ずメッセージmが送信される
6		変数	live(x,a)	いつか必ず変数xの値がaとなる
7	到達可能性	状態	reachable(s)	将来的に状態sがアクティブになる可能性がある
8		メッセージ	reachable(m)	将来的にメッセージmが送信される可能性がある
9		変数	reachable(x,a)	将来的に変数xの値がaとなる可能性がある

## ● 直接記述することも可能

- NuSMVにおける入力形式であるCTLで記述する
  - › CTL：時相論理の一種



# SMVファイルの生成

- UML図(\*.asta)と要求仕様(\*.ctl)を入力としてSMVファイル(\*.smv)を生成する

```
java -jar umltool.jar UML図ファイル名.asta 要求仕様ファイル名.ctl
```

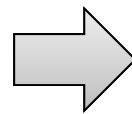
- SMVファイルの拡張子を除いたファイル名は、UML図ファイル名と同じものとなる

# SMVファイルの修正

- 本ツールで生成されたSMVファイルでは、変数の初期値および定義域は記述されていない
  - 状態マシン図において、これらを直接記述するための記法が存在しないため
- ユーザが記述する必要がある

VAR

X: 型を入力してください  
Y: 型を入力してください



VAR

X: {1, 3, 5};  
Y: 0..10;

# モデル検査の実行

- 修正したSMVファイル(\*.smv)を入力としてNuSMVによるモデル検査を行い, 結果を検証結果ファイル(\*.out)を保存する

```
NuSMV.exe -coi SMVファイル名.smv > 検証結果ファイル名.out
```

- coi(Cone of Influence)は, 要求仕様と不要なモデルを抽象化し, 検証を高速化するためのオプション

# 検証結果の整形

- 検証結果ファイル(\*.out)を入力として、整形済み検証結果ファイル(\*.result)と反例ファイル(\*.ce)を生成する

```
java -jar umltool.jar 検証結果ファイル名.out
```

- 整形済み検証結果ファイルと反例ファイルの拡張子を除いたファイル名は、検証結果ファイル名と同じものとなる
- 検証の結果として全ての要求仕様が満たされた場合、反例ファイルは生成されない

# 事例適用による評価

## ● 経緯

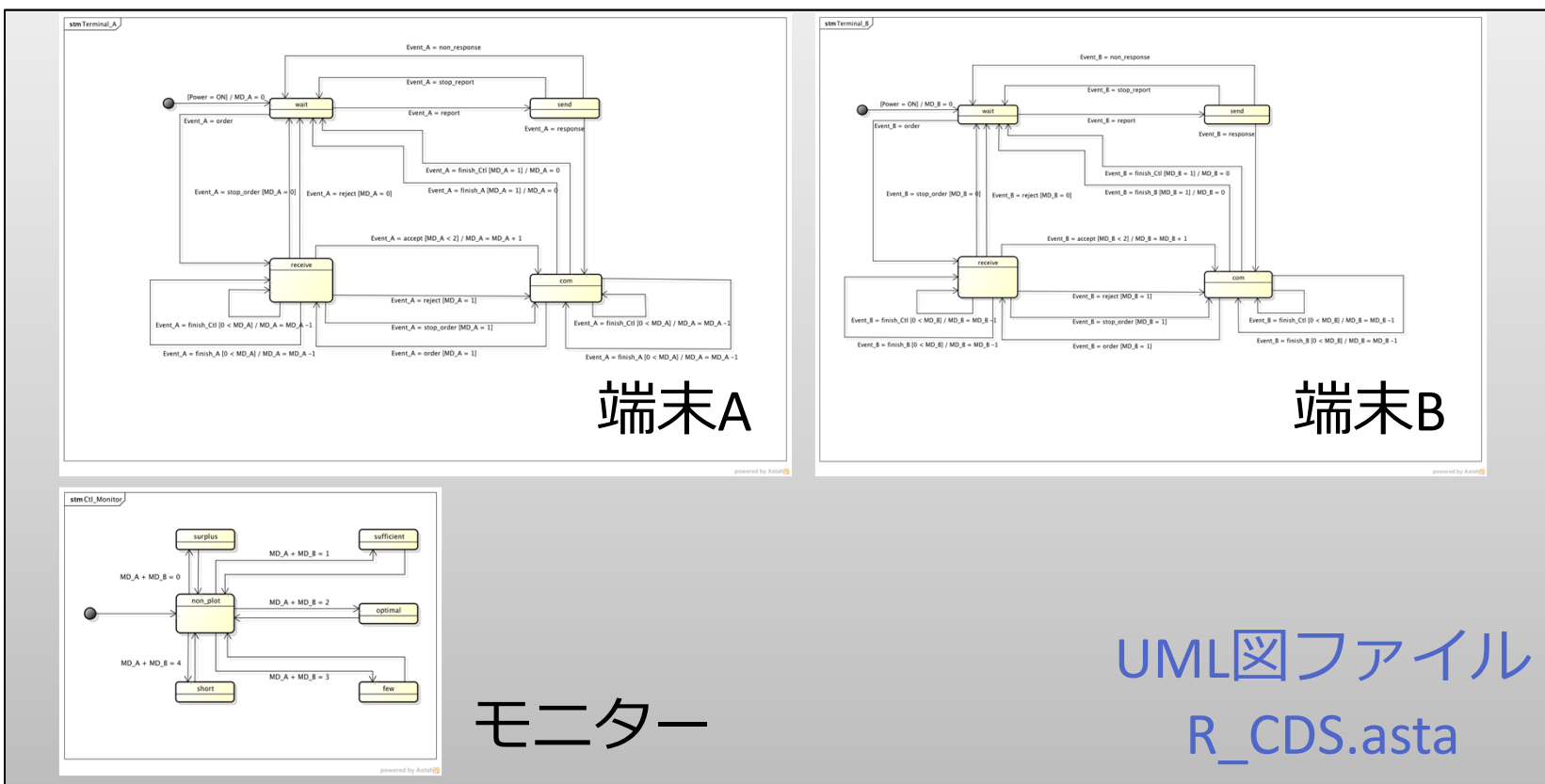
- 某ソフトウェア開発企業に事例提供を打診
- 開発に用いた状態遷移表からUML図(状態マシン図)を作成
- 対象システムは店舗従業員向けの「商品供給指示システム」
  - › 売場の端末と商品管理室のモニタの間で、商品供給のための通信を行う

## ● 検査項目

- 検査1：仕様テンプレートを用いた基本特性の検査
- 検査2：事例提供元より要望のあった特性の検査

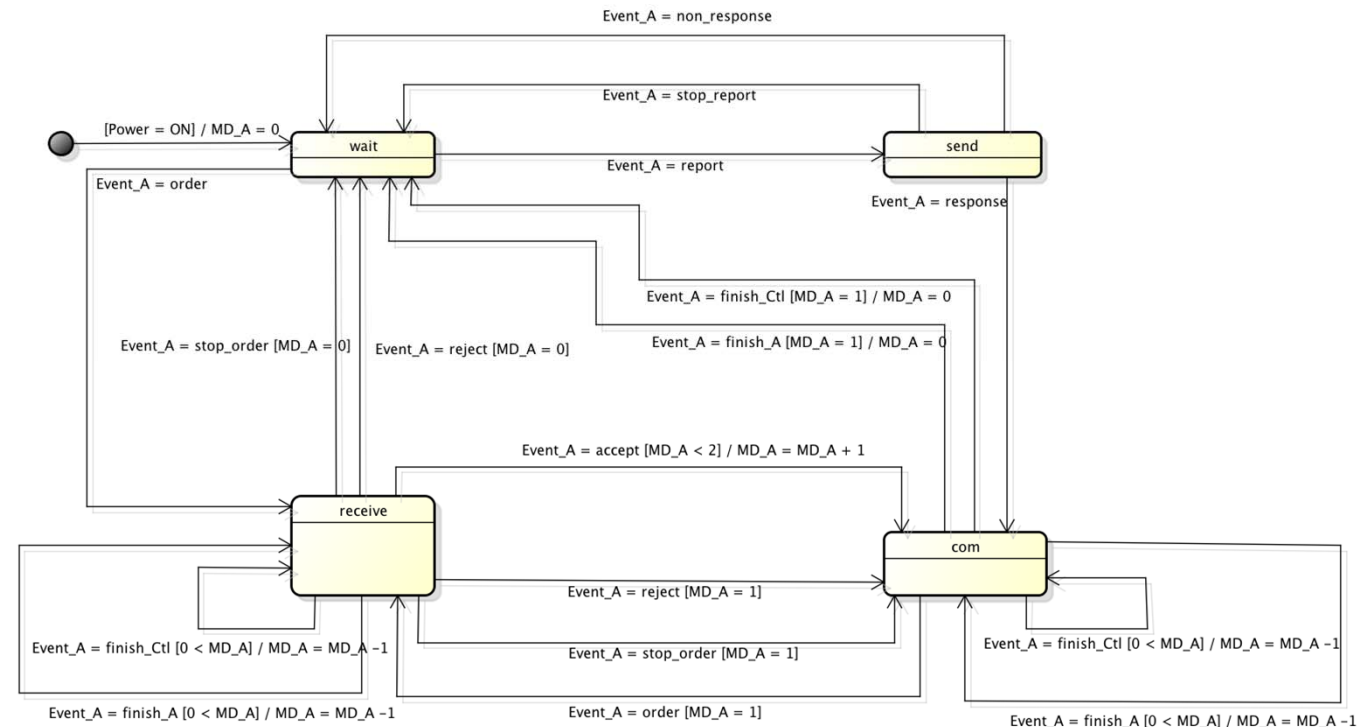
# 対象となるUML図

2台の売場端末とモニター表示の動作の状態マシン図  
(状態遷移表を元に作成)



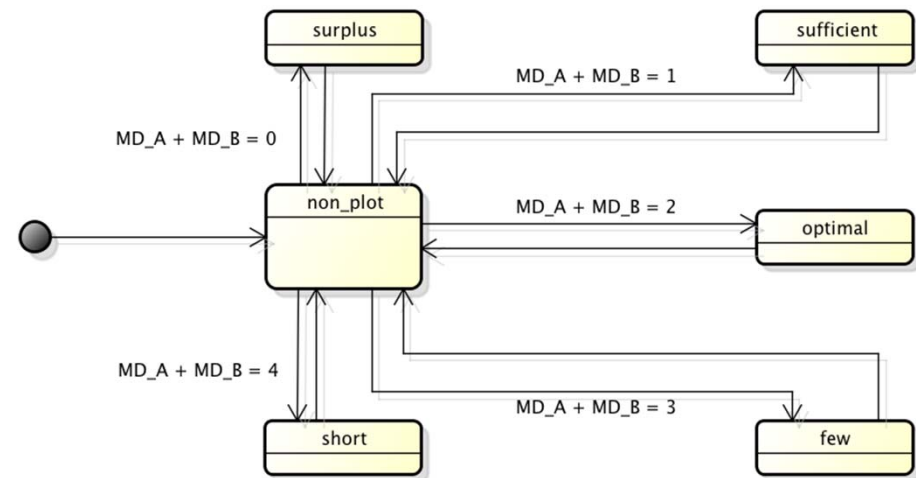
# 売場端末の振る舞い

- 各端末は4つの状態をもつ
  - 待機中(wait), 発話中(send), 着信中(receive), 通話中(com)
- 同時に2回線(MD=2)まで通話が可能



# 管理室モニタの振る舞い

- 通話回線数に応じて6つの状態をもつ
  - surplus(0), sufficient(1), optimal(2), few(3), short(4), non\_plot(不明)
- 通話回線数は端末AおよびBの通話回線数の和( $MD\_A + MD\_B$ )となる

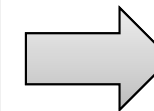




# 検証する特性（検査1）

仕様テンプレートを用いて4つの基本特性を記述

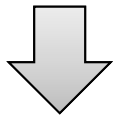
検査項目	テンプレート	ツールへの入力
状態の到達可能性	reachable(s)	reachable(Terminal_A = send) reachable(Terminal_A = receive) reachable(Terminal_A = com)  reachable(Terminal_B = send) reachable(Terminal_B = receive) reachable(Terminal_B = com)
通信モードの到達可能性	reachable(x,a)	reachable(MD_A, 1) reachable(MD_A, 2)  reachable(MD_B, 1) reachable(MD_B, 2)
通信モードの安全性	safe(x,a)	safe(MD_A, 3) safe(MD_A, -1)  safe(MD_B, 3) safe(MD_B, -1)
モニターの到達可能性	reachable(s)	reachable(Ctl_Monitor = surplus) reachable(Ctl_Monitor = sufficient) reachable(Ctl_Monitor = optimal) reachable(Ctl_Monitor = few) reachable(Ctl_Monitor = short)



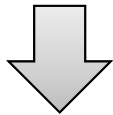
要求仕様  
ファイル  
BASIC.ctl

# 検証結果（検査1）

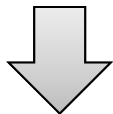
R\_CDS.asta  
BASIC.ctl



R\_CDS\_BASIC.smv



R\_CDS\_BASIC.out



R\_CDS\_BASIC.result

本ツールを用いて  
SMVファイルを生成

モデル検査器  
NuSMVで検査

検査結果の  
整形

```
(001) EF Terminal_A = send is true
(002) EF Terminal_A = receive is true
(003) EF Terminal_A = com is true
(004) EF Terminal_B = send is true
(005) EF Terminal_B = receive is true
(006) EF Terminal_B = com is true
(007) EF MD_A = 1 is true
(008) EF MD_A = 2 is true
(009) EF MD_B = 1 is true
(010) EF MD_B = 2 is true
(011) AG !(MD_A = 3) is true
(012) AG !(MD_A = -1) is true
(013) AG !(MD_B = 3) is true
(014) AG !(MD_B = -1) is true
(015) EF Ctl_Monitor = surplus is true
(016) EF Ctl_Monitor = sufficient is true
(017) EF Ctl_Monitor = optimal is true
(018) EF Ctl_Monitor = few is true
(019) EF Ctl_Monitor = short is true
```

R\_CDS\_BASIC.result

全ての結果が**TRUE** → 誤りは存在しない

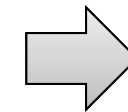
# 検証する特性（検査2）

事例提供元より要望のあった特性を直接検査式として記述

1. 端末が待機状態であり，かつ通信モードが通話無しでない状態への到達可能性
2. 端末が着信中状態であり，かつ通信モードが2回線通話である状態への到達可能性
3. 端末が通話中であり，かつ通信モードが通話無しである状態への到達可能性

```
1A: SPEC !EF(Terminal_A = wait & !(MD_A = 0))
2A: SPEC !EF(Terminal_A = receive & MD_A = 2)
3A: SPEC !EF(Terminal_A = com & MD_A = 0)

1B: SPEC !EF(Terminal_B = wait & !(MD_B = 0))
2B: SPEC !EF(Terminal_B = receive & MD_B = 2)
3B: SPEC !EF(Terminal_B = com & MD_B = 0)
```



要求仕様ファイル  
R\_CDS\_REQ.ctl

# 検証結果 (検査 2)

R\_CDS.asta  
R\_CDS\_REQ.ctl



R\_CDS\_REQ.smv



R\_CDS\_REQ.out



R\_CDS\_REQ.result  
R\_CDS\_REQ.ce

```
(001) !(EF (Terminal_A = wait & !(MD_A = 0))) is true  
(002) !(EF (Terminal_A = receive & MD_A = 2)) is true  
(003) !(EF (Terminal_A = com & MD_A = 0)) is false  
...
```

R\_CDS\_REQ.result(一部)

3A,3Bの結果がFALSE → 誤りを検出し、反例を出力

```
(003) !(EF (Terminal_A = com & MD_A = 0)) is false  
-> State: 1.1 <-  
Terminal_A = initial  
MD_A = 0  
Power = start  
Event_A = emp  
...
```

R\_CDS\_REQ.ce(一部)

# 生成された検証結果ファイル

```
1 (001) !(EF (Terminal_A = wait & !(MD_A = 0))) is true↓
2 (002) !(EF (Terminal_A = receive & MD_A = 2)) is true↓
3 (003) !(EF (Terminal_A = com & MD_A = 0)) is false↓
4 (004) !(EF (Terminal_B = wait & !(MD_B = 0))) is true↓
5 (005) !(EF (Terminal_B = receive & MD_B = 2)) is true↓
6 (006) !(EF (Terminal_B = com & MD_B = 0)) is false↓
7 [EOF]
```

R\_CDS\_REQ.result

R\_CDS\_REQ.ce

```
1 (001) !(EF (Terminal_A = wait & !(MD_A = 0))) is true↓
2 (002) !(EF (Terminal_A = receive & MD_A = 2)) is true↓
3 (003) !(EF (Terminal_A = com & MD_A = 0)) is false↓
4 -> State: 1.1 <-↓
5 Terminal_A = initial↓
6 MD_A = 0↓
7 Power = start↓
8 Event_A = emp↓
9 -> State: 1.2 <-↓
10 Power = ON↓
11 Event_A = report↓
12 -> State: 1.3 <-↓
13 Terminal_A = wait↓
14 -> State: 1.4 <-↓
15 Terminal_A = send↓
16 Event_A = response↓
17 -> State: 1.5 <-↓
18 Terminal_A = com↓
19 Event_A = report↓
20 (004) !(EF (Terminal_B = wait & !(MD_B = 0))) is true↓
21 (005) !(EF (Terminal_B = receive & MD_B = 2)) is true↓
22 (006) !(EF (Terminal_B = com & MD_B = 0)) is false↓
23 -> State: 2.1 <-↓
24 Terminal_B = initial↓
25 MD_B = 0↓
26 Power = start↓
27 Event_B = emp↓
28 -> State: 2.2 <-↓
29 Power = ON↓
30 Event_B = report↓
31 -> State: 2.3 <-↓
32 Terminal_B = wait↓
33 -> State: 2.4 <-↓
34 Terminal_B = send↓
35 Event_B = response↓
36 -> State: 2.5 <-↓
37 Terminal_B = com↓
38 Event_B = report↓
39 [EOF]
```

# 反例の解析

## 特性3Aに対する反例の解析(3Bも同様)

	1	2	3	4	5
Event_A	emp	emp	report	response	emp
MD_A	0	0	0	0	0
Terminal_A	initial	initial	wait	send	com

端末が通信中(Terminal\_A=com)であるにもかかわらず、  
通信モードが通話無し(MD\_A=0)となる状態に到達している

原因：状態遷移表のアクションの1つが  
UML図に正しく転記されていなかった

- UML図による設計の誤りを正しく検出できた
- 反例を元に誤り箇所を特定することができた

# まとめ

- モデル検査とは？
  - 形式手法の一種
  - 自動的かつ網羅的検証が可能な技術
- 形式手法とは？
  - 形式仕様記述と形式的証明
- 形式手法の適用事例
  - 交通・医療に関して多くの事例
- 検証支援ツールの紹介

# SESワークショップのお知らせ

SES2014のワークショップにて、  
形式手法に関するセッションを開催します

## 形式手法 - 産学連携における問題とその解決 -

討論リーダー：横川智教（岡山県立大）  
早水公二（フォーマルテック）

日時・場所：9月3日(水) 10:20～  
3F 302室

興味のある方は是非お越しく下さい



# 謝辞

本チュートリアルの一部は,  
**独立行政法人情報処理推進機構・技術本部**  
**ソフトウェア高信頼化センター(IPA/SEC)**が実施した,  
「**2013年度ソフトウェア工学分野の**  
**先導的研究支援事業(RISE)**」の支援を受けています。