

## 企業におけるフォーマルメソッドの実践

関西電力株式会社 電力技術研究所

篠崎 孝一

メルコ・パワー・システムズ

早水 公二

## 前半 ～実践のためのガイド～

1. 背景（実践研究会の活動背景）
2. なぜモデル検査を選択？
3. 導入ステップと教育の現状
4. 実務導入の課題
5. 実用的な時間で成果を出すために
  - GUIソフトの開発／活用
6. 無駄、失敗を防ぐための実践ノウハウ
  - 対象選定から検査作業まで

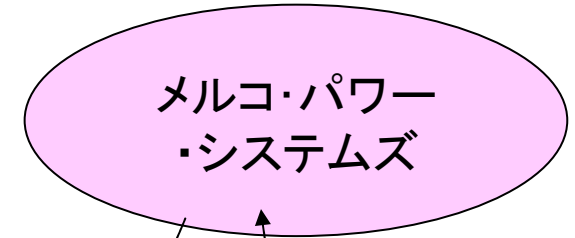
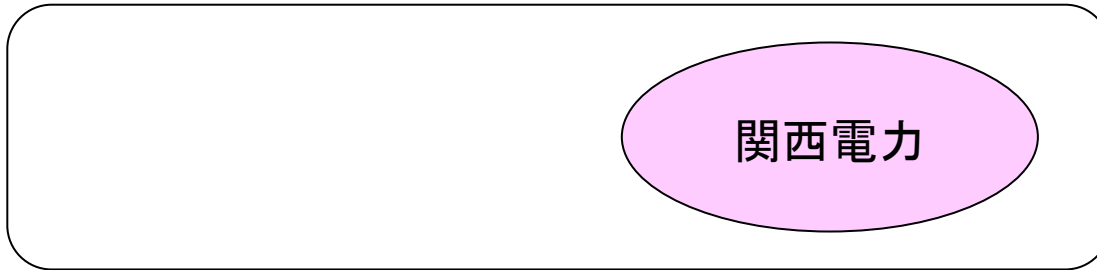
後半 ～始めよう！ 広げよう！ モデル検査～

## 1. 活動の背景

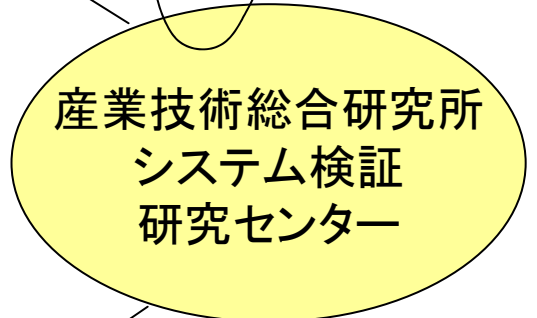
1. シーズ先行で、モデル検査を試した結果が良かった。
2. 当時(2001)、国内で紹介されていなかった。
3. 使い勝手が悪く、改善が必要だった。
4. 実践に必要なノウハウが不足していた。

(2002～2004年)

### データ収集装置の発注仕様書をモデル検査



(研究員派遣)

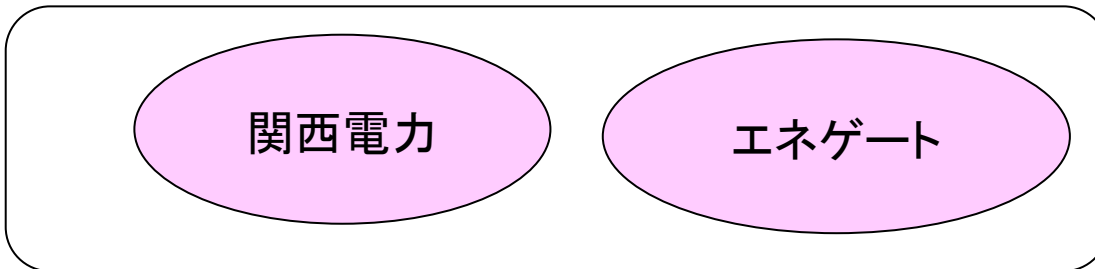


技術指導  
(でもテキストは英語)

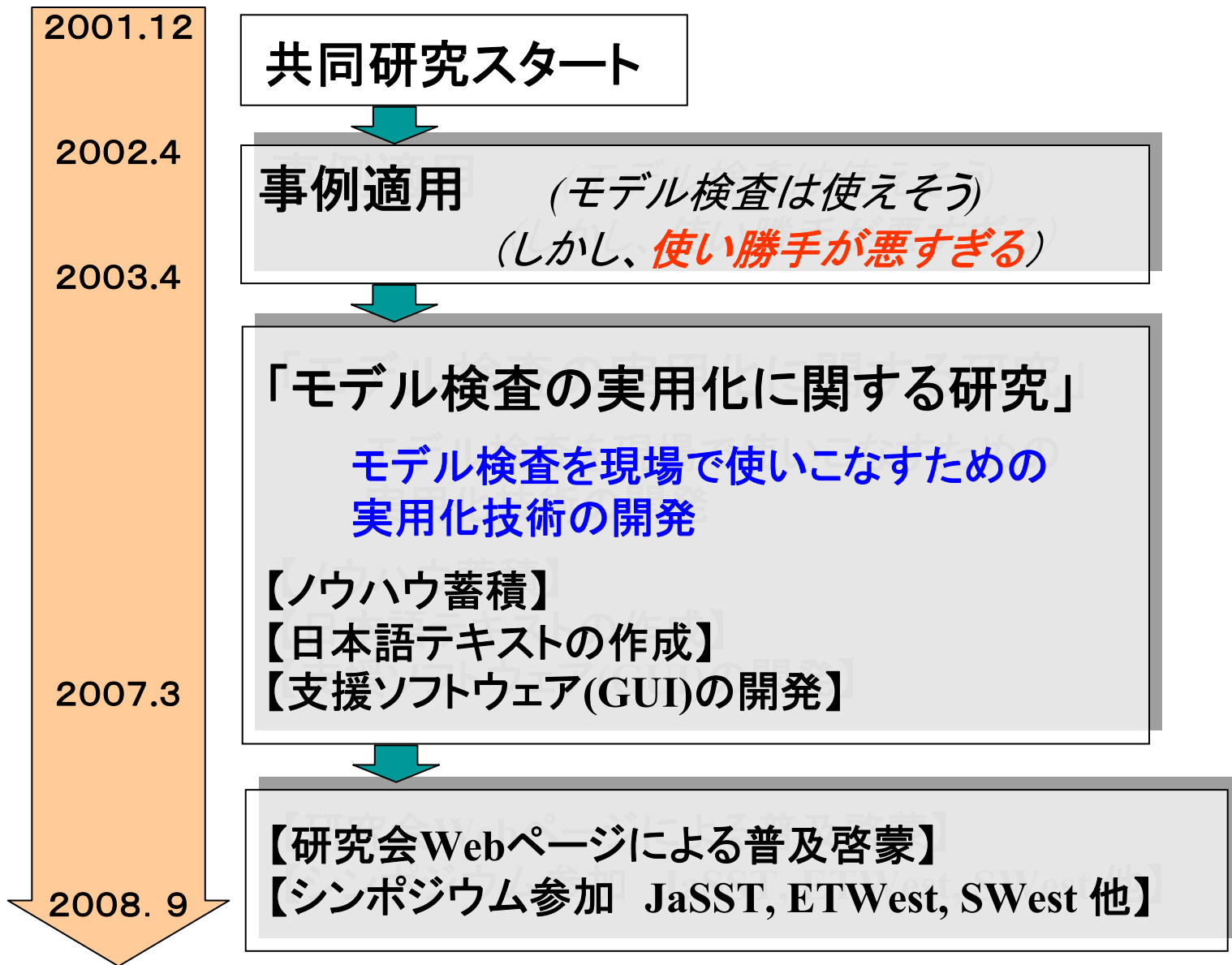
実績のある仕様書を元に実験用の仕様を作成・検査  
記述漏れ・不整合を発見 (何だ、この技術は?)

(2003～2004年)

### 組込み機器のプログラム仕様書をモデル検査



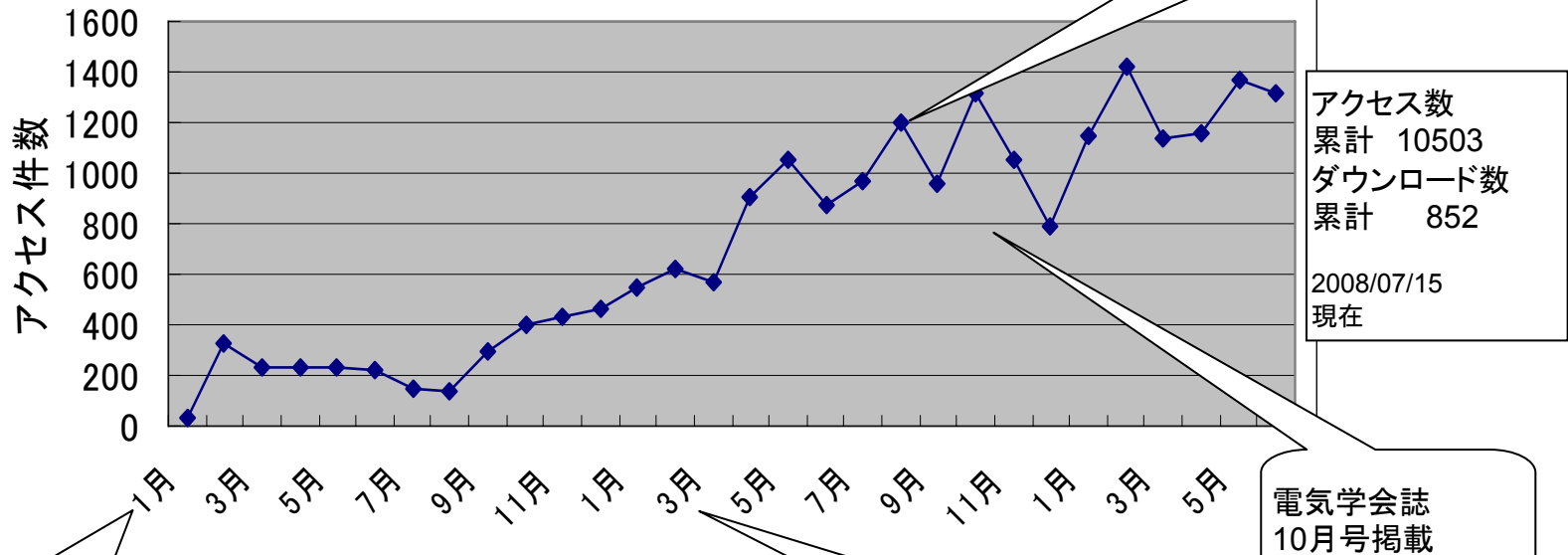
新規開発品のプログラム仕様を、開発と併行して検査  
多重割り込みのタイミング問題を発見 (すごい技術だ)(使い難い問題点もあるぞ)



# 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、メルコ・パワー・システムズ、エネゲート)

## モデル検査研究会Webページの月間アクセス件数



2006/1/26 日経エレクトロニクス誌に掲載

モデル検査の活用事例を紹介

4/3 試供版ダウンロード提供開始

(<http://www.modelcheck.jp/>)

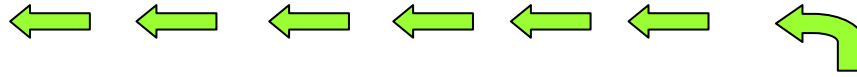
## 2. なぜモデル検査を選択？

### 【実務での使いやすさ】

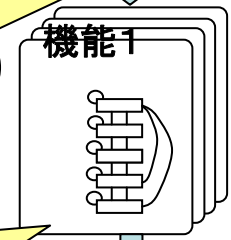
1. 前提知識が少ない。
2. 自動化割合が高い(モデル検査器が使える)。
3. 反例が出力される。

# モデル検査の作業

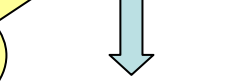
(検査対象)



モデル化範囲  
絞込み



状態遷移図  
の作成



検査プログラムの  
作成

```
MODULE main
VAR
Home:boolean;
Bus:{on, off}
ASSIGN
init(PC) := ON
next(PC) := case
```

検査項目

時相論理式の作成

```
SPEC
EF(vend=COIN & pow = ON & can = OF
SPEC
AG(vend=COIN & pow = ON & can =OFF
```

検査プログラムに  
時相論理式を追加

検査結果の  
フィードバック

(出力結果)

```
時相論理式1
true
時相論理式2
true
時相論理式3
true
時相論理式4
false
(falseの事例)
```

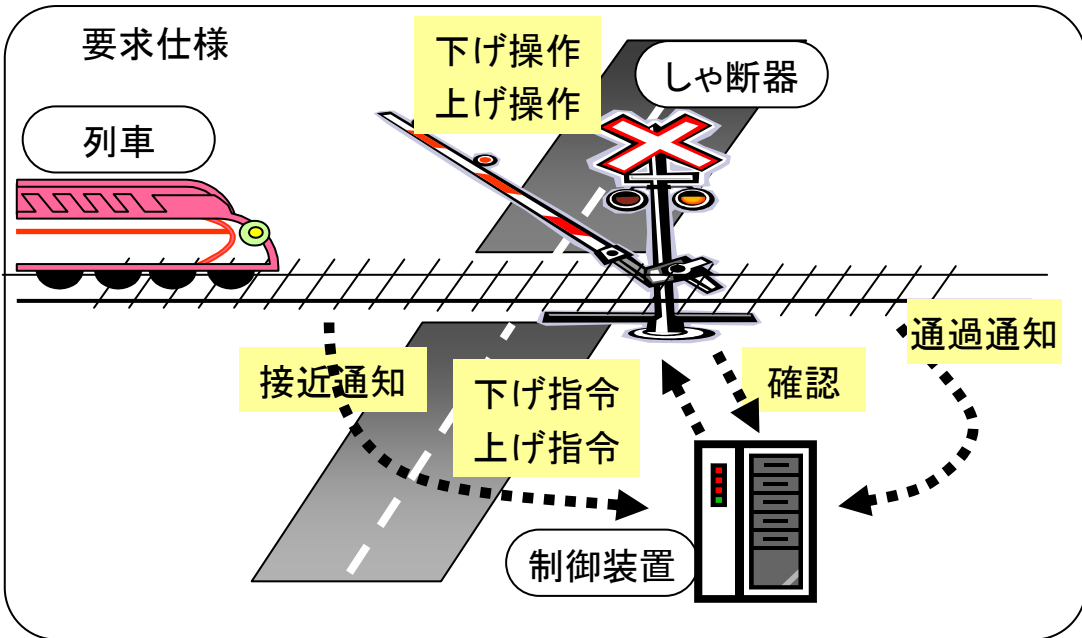
モデル検査器による自動検査

自動検査  
プログラムで書ける  
(数式が最低限度)  
結果 (反例) が読める

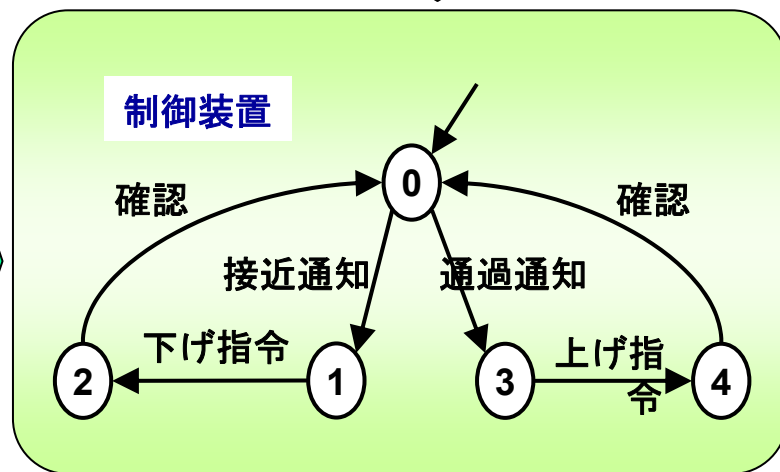
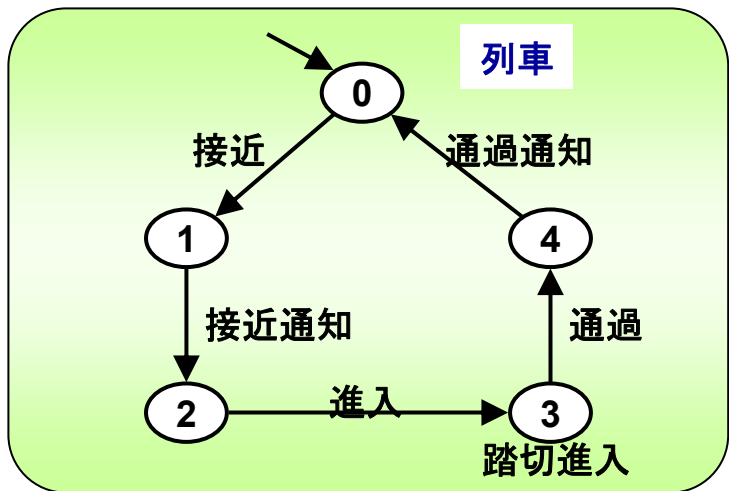
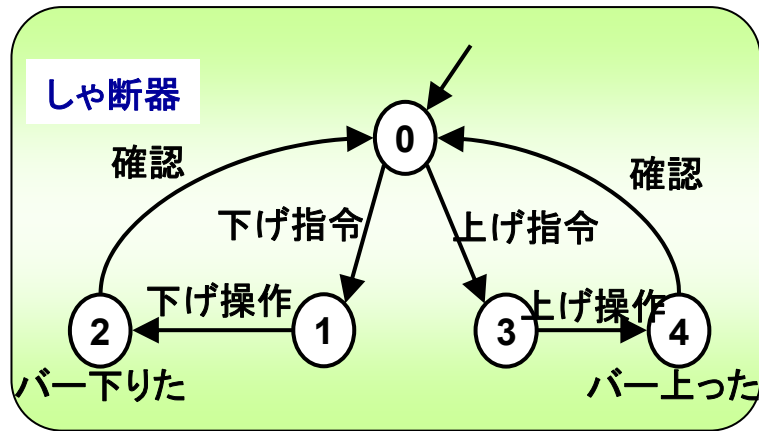




# 事例に見るモデル検査のメリット



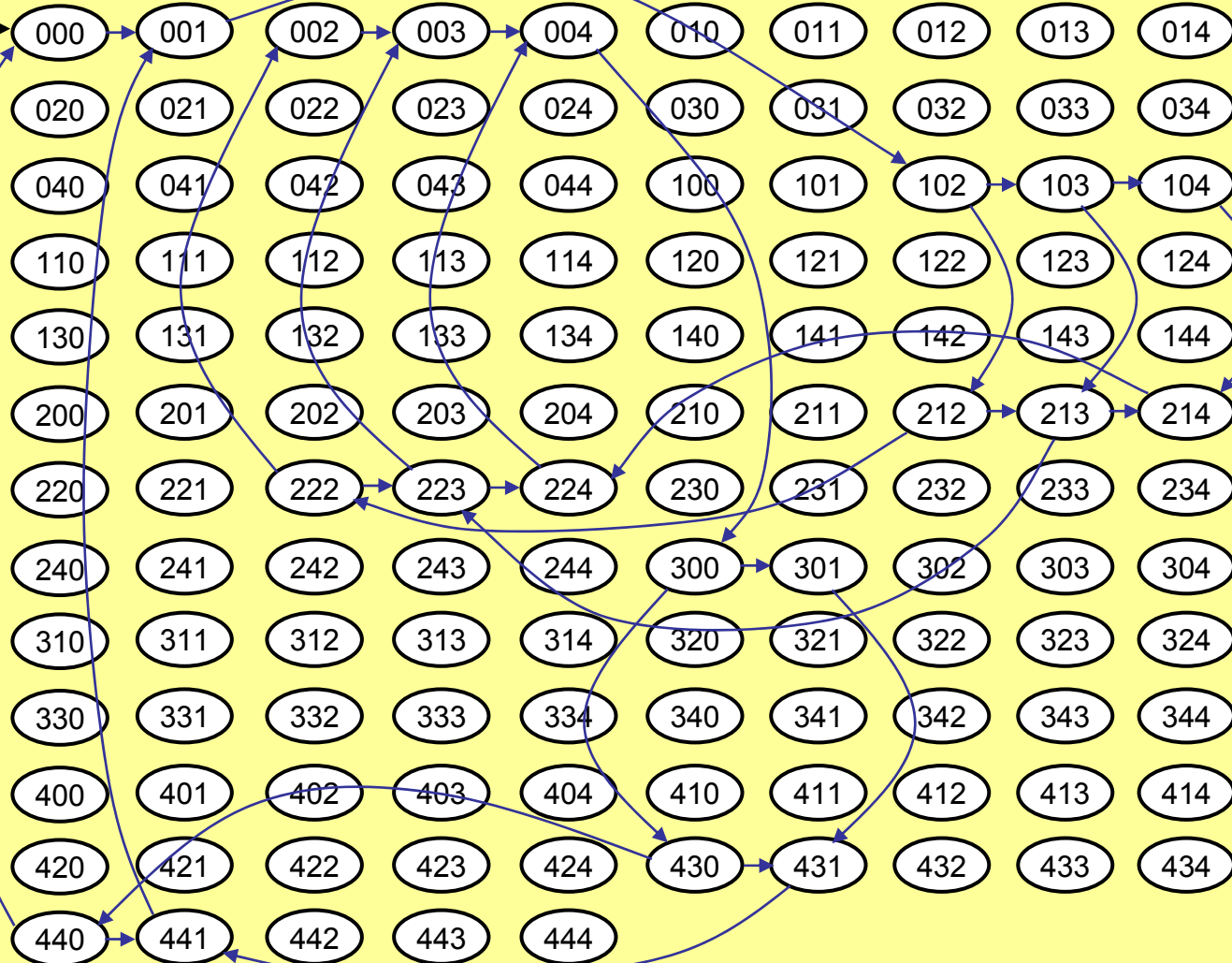
## 状態遷移図



# システム全体の状態と遷移

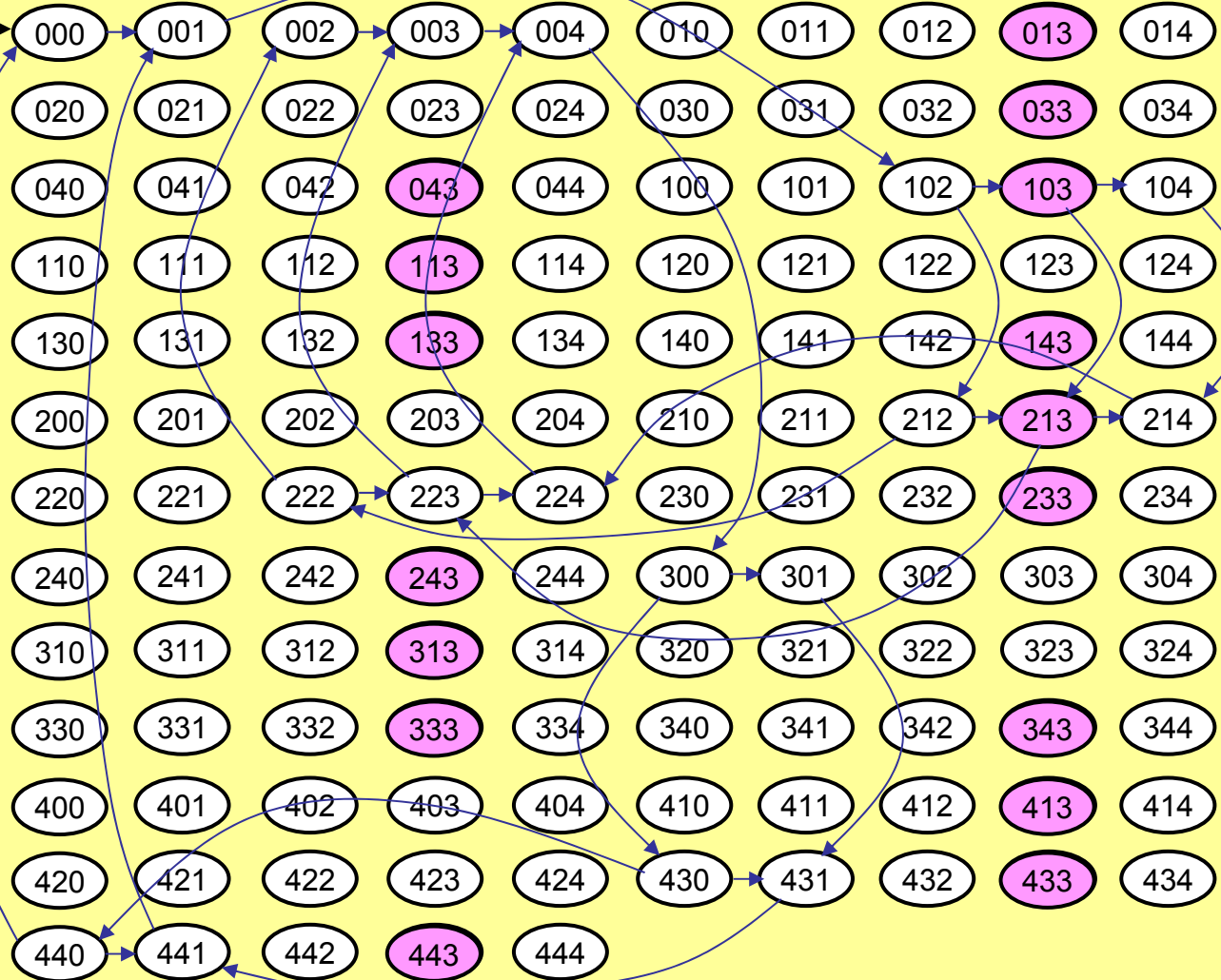
(制御装置：しゃ断器：列車)

初期状態



# 不具合状態は全状態の中に存在

初期状態

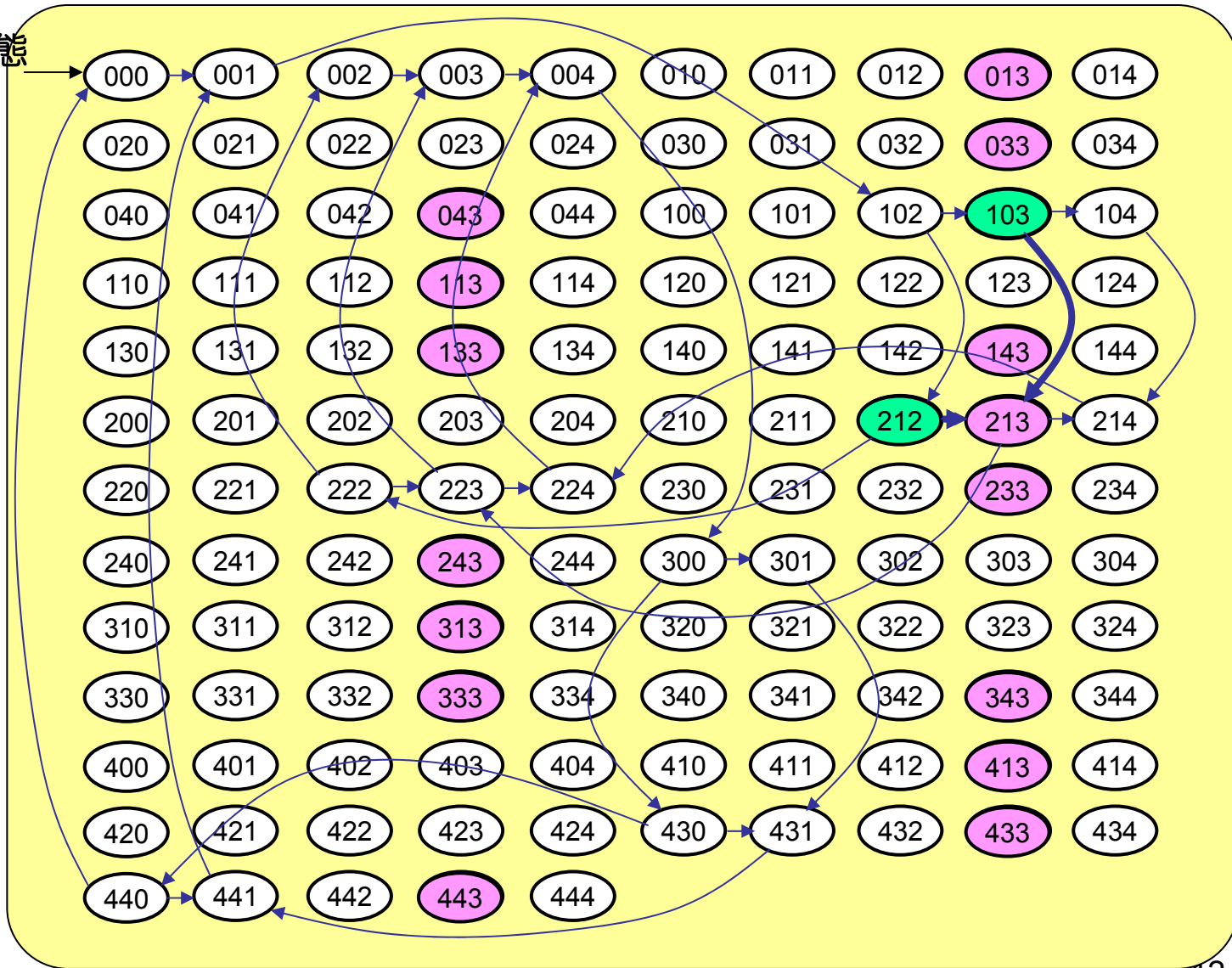


しゃ断器が  
下りずに、  
列車が踏切  
に進入する  
状態集合  $F_T$   
は、全状態中  
に、存在する

# 不具合状態から遷移関係を遡る

初期状態

状態集合  $F_T$  から遷移関係をひとつ遡って前状態集合  $F_{T-1}$  を求めることができる

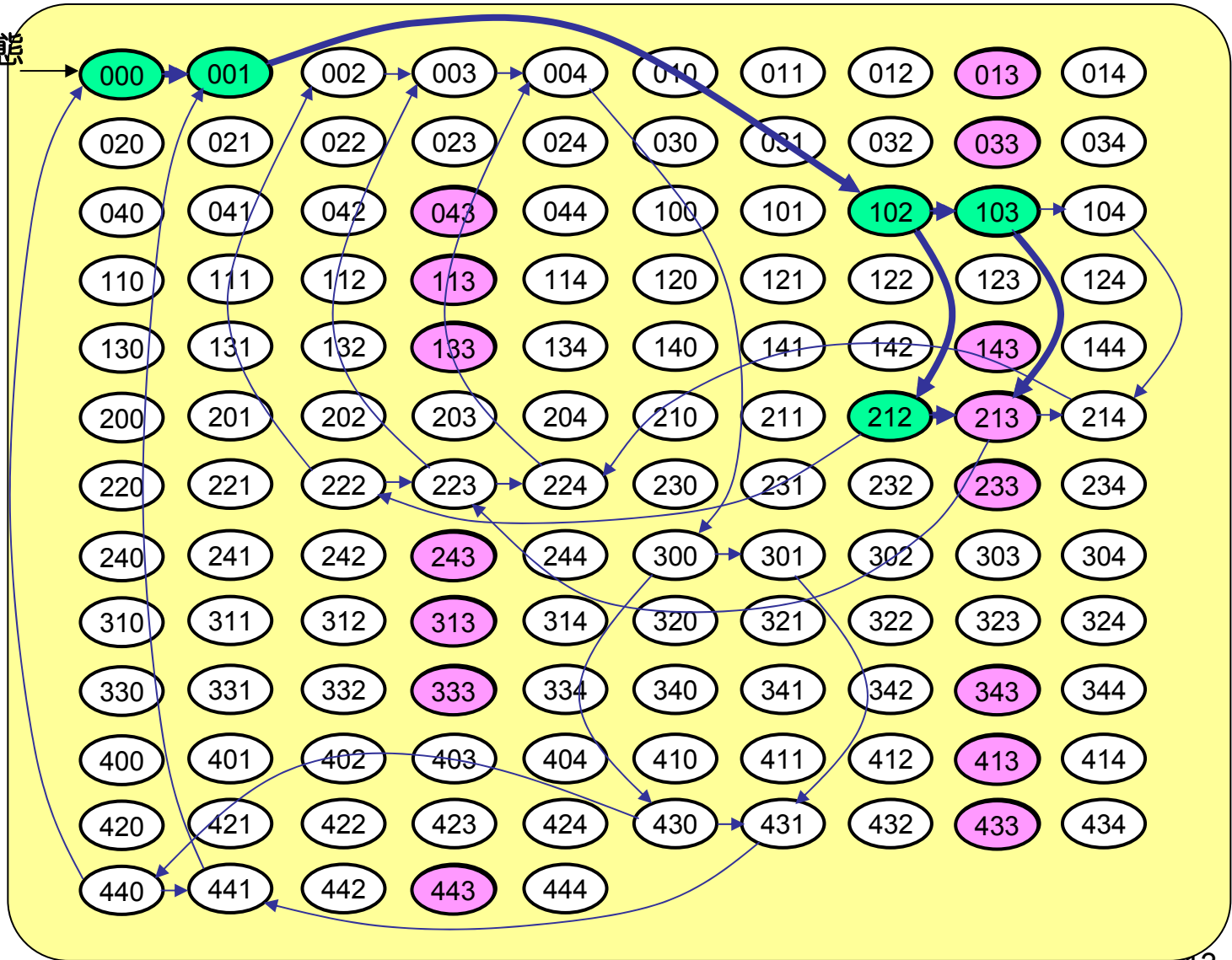


# 遡って初期状態へ至れば、不具合は発生する

初期状態

状態集合  $F_T$  の  
前状態集合  $F_{T-1}$   
に初期状態が  
含まれていれば  
システムは、  
起動後に不具合  
事象に到達する  
場合がある

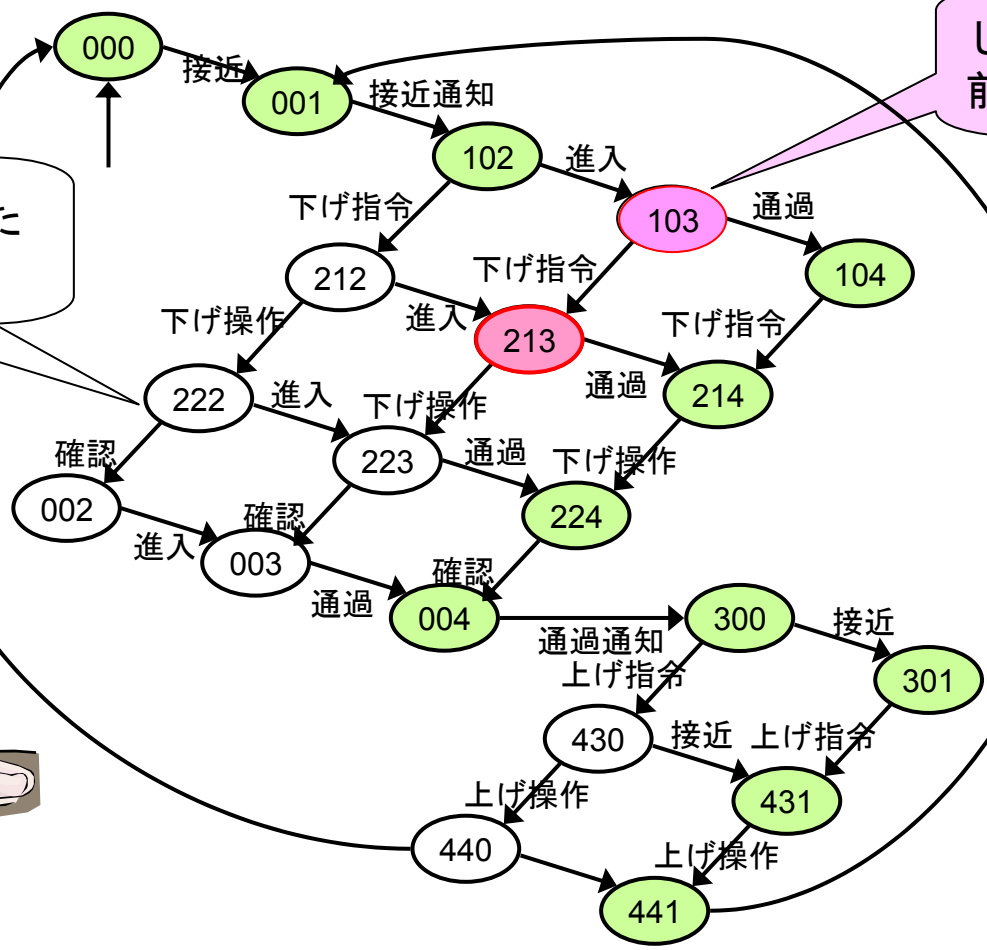
論理的な操作  
手順により、  
不具合事象発  
生の可能性を  
確実に検出  
できる。



# 反例で出力される状態遷移

しゃ断器バーが下りる前に列車進入

設計者が考えていた安全な経路



全数探索により見逃しを確実に検出

# モデル検査の考え方

## 検査項目(例)

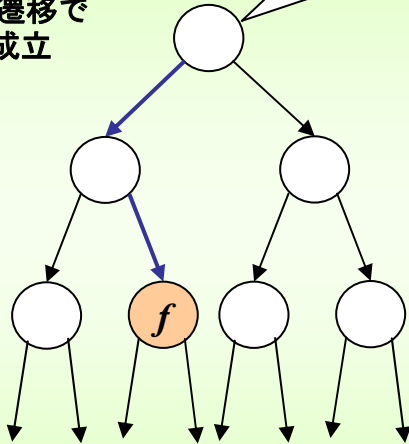
このシステムは  $f$  になる場合がある

初期状態から出発し、  
何れかの状態遷移で  
いつかは  $f$  が成立

初期状態

$EF f$

時相論理式



## 検査の手順

システム動作の全状態を展開する

$f$  が成り立つ状態の集合を求める

その集合に1ステップで遷移  
できる前状態の集合を求める

スムージング  
演算  
(論理式)

全ての前状態  
集合?

NO

YES

前状態集合に  
初期状態が含まれる?

YES

NO

初期状態から  
 $f$  に  
たどり着ける

初期状態から  
 $f$  には  
たどり着けない

初期状態が含まれる=  
【検査項目は満足される】

初期状態が含まれない=  
【検査項目は満足されない】

## モデル検査のメリット

### 【実務での使いやすさ】

1. 前提知識が少ない。
2. 自動化割合が高い(モデル検査器が使える)。
3. 反例が出力される。

### 【事例でわかること】

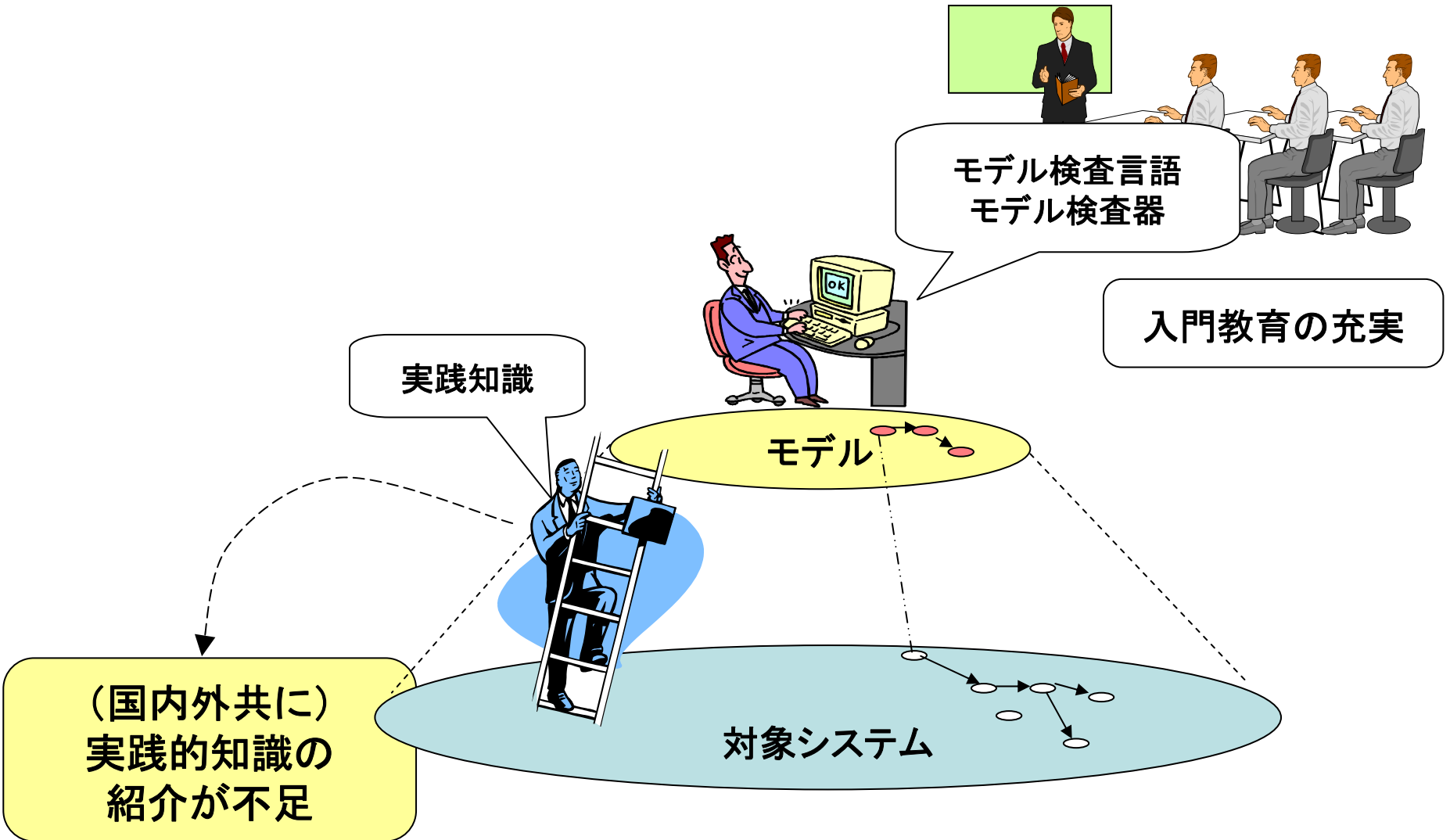
1. 状態遷移モデルを作る (部分モデルを作る)。
2. 検証項目を作る (テストデータは不要)。
3. モデルの全状態遷移を探索 (怪しい要素は、モデル化要)。
4. 反例を読み取り、不具合を理解する。



### 3. 導入ステップと教育の現状

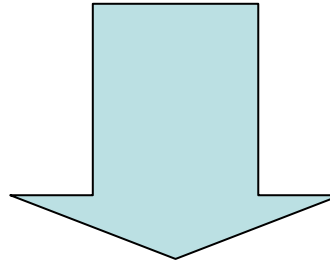
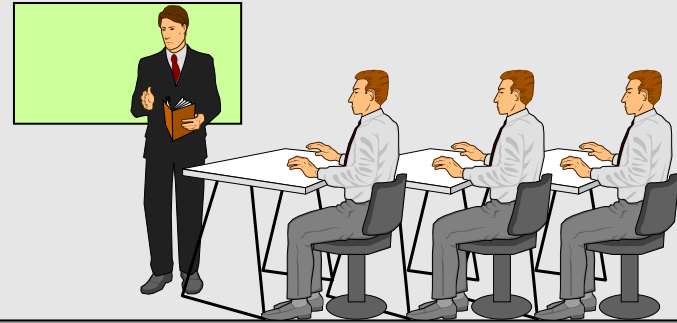
1. 技術者向け入門教育が整備され始めた。
2. 実践部分の教育は、これからの段階。
3. モデル検査の使い方も、枝分かれし始めた。

# モデル検査の技術者向け教育の現状



## 【基本的な知識を得る】

1. 基本的な勉強をする。
2. モデル検査器の使い方を知る。
3. 演習問題を解く。

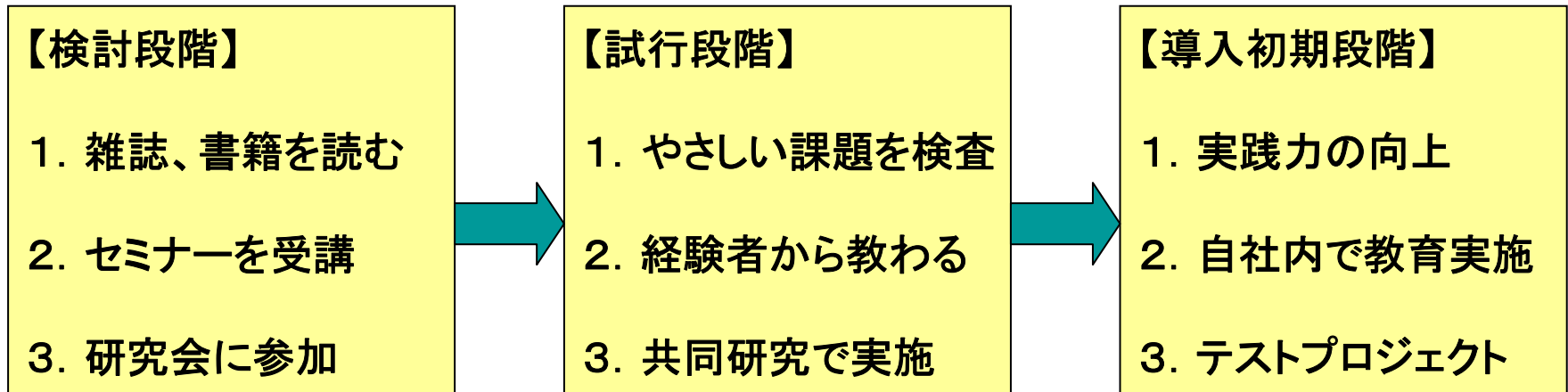


## 【実務に使える実力をつける】

1. 実務に使えて、初めて効果がある。
2. 特有のノウハウがたくさんある。
3. 演習問題だけでは不十分で、経験が必要。
4. 実務導入の経験者に教わるのが早道。



## 実践力をつけながら導入拡大を目指す



初心者向けの解説書  
がほとんどない。

教育セミナーの試行  
が始まっている。

費用対効果のデータ  
がほとんどない。

自社で試行し、実感  
してデータ化する。

作業時間が掛かりすぎる

使い易いツールを選択  
する

実践力（ノウハウ）が  
不十分

先行企業から情報を得る

# モデル検査の使い方の枝分かれ

ねらい

ソフトウェア一般の不具合

システム固有の問題に  
入り込んだ不具合

特徴

- ・誰でも使える
- ・自動的にモデル化
- ・モデルを意識しない

- ・システムの知識が必要
- ・人手によりモデル化
- ・モデルをしっかり考える

- ・分野、対象を限れば、  
自動化は可能

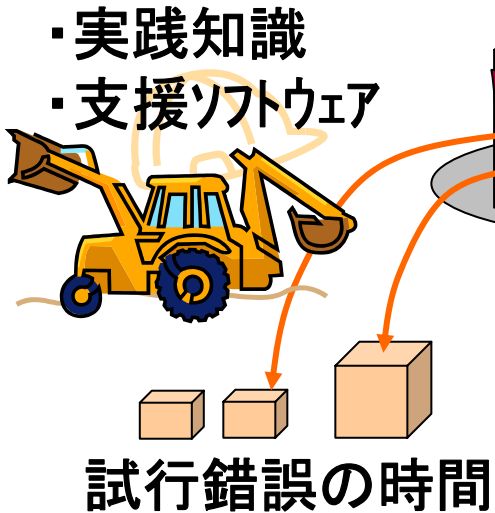
## 4. 実務導入の課題

1. 作業時間の短縮
2. 関係者の説得
3. 実践経験の不足

# 作業時間の短縮

## モデル検査に要した時間

開発費に  
計上される



## モデル検査の時間効果

バグの影響は  
「スケール・フリー」  
算定不能？

- ・テスト工数の削減？
- ・回収費用の未然防止？

様々な実践知識を活用して、時間短縮／効率化が必要

# 関係者の説得

仲間を増やす。実施を認めてもらう。



技術者対象の  
2時間コース

- ・ 社内外の説明会で困ったこと

## 質問

Q1

「なぜ正しいのか、簡単に説明せよ」

Q2

「モデル検査器の動作は誰が保証？」

Q3

「直感的に、理解できない」

Q4

「導入効果のデータを示せ」



社内トップ  
への説明

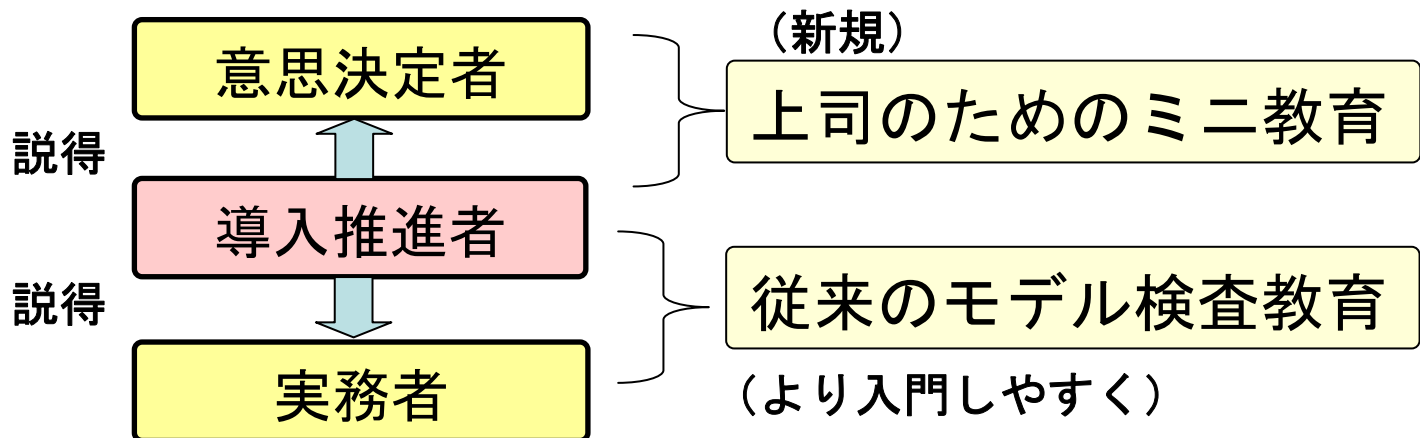




# 実務者向けの説明だけではダメ？

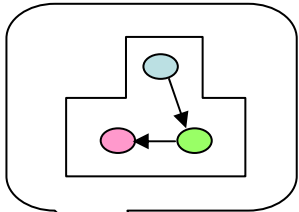
モデル検査を知らない人にも、理解を得なければ導入が難しい

これまでは、モデル検査を習得したい人を対象にして、教えることを考えていた。



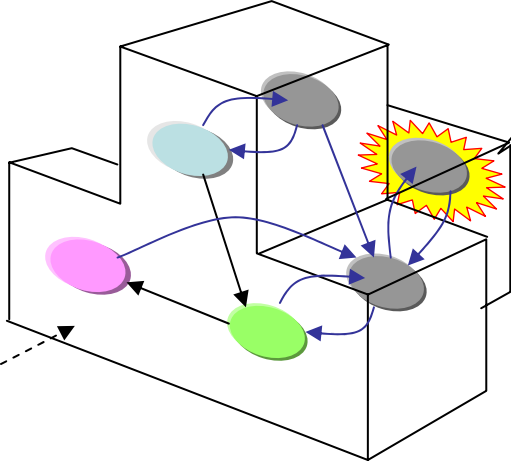
# 「直感的なイメージを掴んでもらう」

設計者の視点からの認識



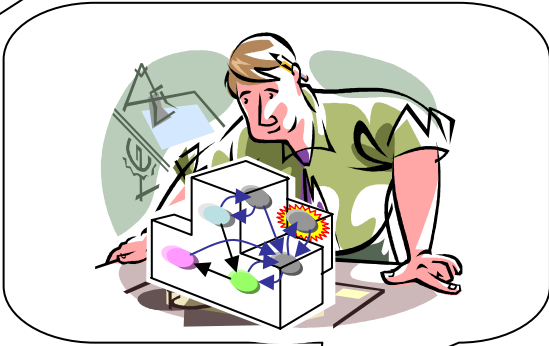
システムが複雑になると意識できていない動作がある

ソフトウェア設計【起こり得る動き】



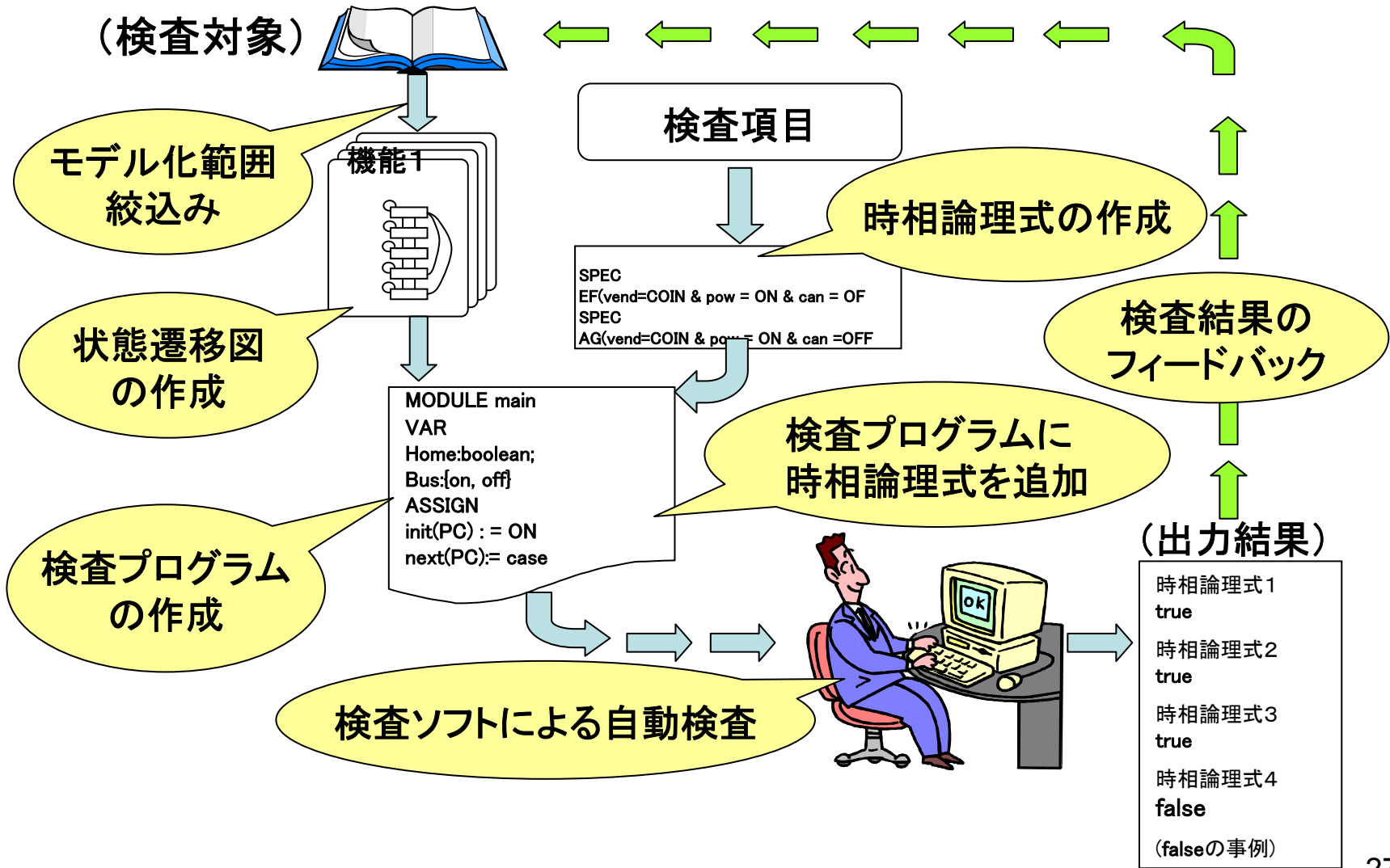
● : 見落としている状態

見落としてる危険状態



```
MODULE main
VAR
  Home : (On, Off);
  Ticket_Mony : (OK, EMP, NG);
  Time : #;
  Ticket_N : 0..2;
  PC : #;
ASSIGN
  Init(Home) => On;
  Init(Ticket_Mony) => EMP;
```

# 「作業の流れを掴んでもらう」



# 実戦経験の不足

実践の経験／ノウハウが不足

(導入へのハードル)

- ・検査項目の決定
- ・難解なモデル記述
- ・状態爆発による無回答
- ・検査項目の記述
- ・反例解析の手間

先行企業／経験者から情報を得るのが、効率的！

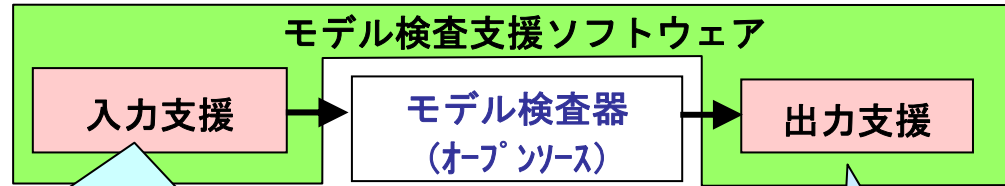
(自力で頑張っても、無駄が多い)



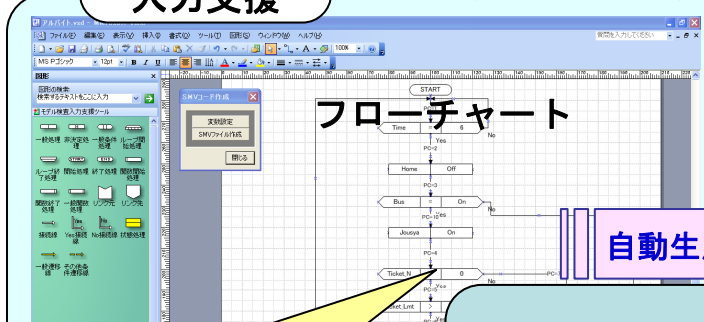
## 5. 実用的な時間で成果を出すために

1. GUIソフトの開発／活用
2. モデル検査の見える化
3. 見える化によるコミュニケーションの改善

# GUIソフト（モデル検査支援ソフト）の開発／活用



入力支援



フローチャート

自動生成

モデル検査専用言語  
のプログラム

```

MODULE main
VAR
  Home : {Off, On};
  Ticket_Mmy : {OK, EMP, NG};
  Time : 6..18;
  Ticket_N : 0..2;
  PC : 1..15;
ASSIGN
  
```

画面上でフロー図を作成すれば、検査プログラムに自動変換する

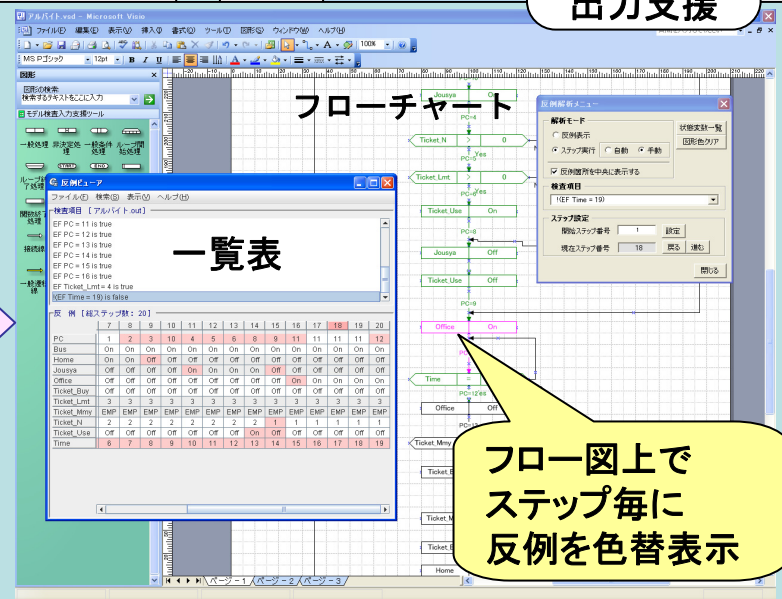
```

-- specification AG ((year mod 400 = 0 & PC = 9) -> output = LEAP) is false
-- as described by the following execution sequence
Trace Description: CTL Counterexample
Trace type: Counterexample
-> State: 1.1 <-
  year = 1800
  PC = 1
  output = REGULAR
-> State: 1.2 <-
  PC = 2
-> State: 1.3 <-
  PC = 4
-> State: 1.4 <-
  PC = 9
-> State: 1.5 <-
  year = 1801
  PC = 2
-> State: 1.6 <-
  PC = 3
-> State: 1.7 <-
  PC = 8
  
```

自動生成

検査結果  
データリスト

出力支援



フロー図上でステップ毎に反例を色替表示

# フロー図の作成

# 状態遷移図の作成

INST.vsd - Microsoft Visio

ファイル(F) 編集(E) 表示(V) 挿入(I) 書式(O) ツール(T) 図形(S) ウィンドウ(W) ヘルプ(H)

検索するテキストをここに入力

MS Pゴシック 12pt B I U

図形の検索: 検索するテキストをここに入力

モデル検査

START END

開始処理 終了処理 一般処理 非決定処理

一般条件処理 ループ開始処理 ループ終了処理 関数開始処理

関数終了処理 一般関数処理 リンク元 リンク先

接続線 Yes接続線 No接続線 状態処理

一般遷移線 その他条件遷移線

メニュー

表示形式

変数名 日本語変数名

プロジェクト名

INST

SMVコード作成

検査項目作成

モデル検査実施

反例解析

測定回路

センサ

制御部

停電



# 検査項目の入力（時相論理式作成）

検査項目 (CTL式) 生成

検査項目: 条件Pが成立すれば、必ず条件Qが成立する。

条件Pが成立すれば、必ず条件Qが成立する。  
条件Pが成立すれば、必ず次に条件Qが成立する。  
条件Pが成立すれば、条件Qが成立することはない。  
条件Pが成立することがある。  
条件Pが成立することはない。

AccountG_Entry_1	boolean		
AccountG_Entry_2	boolean		
AccountG_Entry_3	boolean	QueRemainNum	0..10
AccountG_Name_A	Area1,Area2,Area3	SecCheck_1	boolean
AccountG_Name_C	CaseLevel_H,CaseLevel_M,CaseLevel_L	SecCheck_2	boolean
AccountG_Name_S	Sec_1,Sec_3,Sec_5	SecCheck_3	boolean
Group	1..3	SecurityLevel	AA,AB,BA
HostNum	1..2	SecurityLevel_RT	1..4
		kFg_1	boolean
		kFg_2	boolean
		kFg_3	boolean
		File_S,Program_S,Update_S	
		UPS_SecCounter	1..7
		USB_SecCounter	1..7
		UserCounter	1..4
		UserGetAA	boolean
		UserGetAB	boolean
		UserGetBA	boolean
		UserGetPB	boolean

変数一覧

変数値の選択

検査パターン (日本語)

作成された CTL

選択項目: 0

選択事項: 0

OK キャンセル

クリア

クリア

CTL式:

AG ( ( AccountG\_Entry\_1 = 1 & LmtCheck\_1 = 1 ) -> AF ( ServerCheckFg\_1 = 1 ) )

開 < 保 存 CTL式生成 終了

# 反例 (counter-example) のシミュレーション表示

INST.vsd - Microsoft Visio

反例ビューア

検索項目 [ INST.out ]

```

(AG (INST = Order_1 -> AF (SNR1 = OK | SNR1 = NG)) & AG (INST = (
AG ((INST = Stop & PF = 0) -> AF INST = Init) is true
AG (AF SNR2 = OK) & AG (AF SNR1 = OK)) is false
AG (AF (CTRL = Req_1 | CTRL = Req_2)) is false
    
```

反例【総ステップ数: 7 (ループ開始: 5)】

	2	3	4	5	6	7
CTRL	Req_1	Wait_1	Wait_1	Wait_1	Wait_1	Wait_1
INST	Init	Order_1	Order_1	Stop	Init	Stop
PF	0	0	1	0	1	0
SNR1	Init	Init	NG	Init	Init	Init

接続線 Yes接続 No接続線 状態処理

反例解析メニュー

解析モード

- 反例表示
- ステップ実行
  - 自動
  - 手動

反例箇所を中央に表示する

状態変数一覧  
図形色クリア

検索項目

AG (AF (CTRL = Req\_1 | CTRL = Req\_2))

ステップ設定

開始ステップ番号: 1 [設定]

現在ステップ番号: 7 [戻る] [進む]

[閉じる]

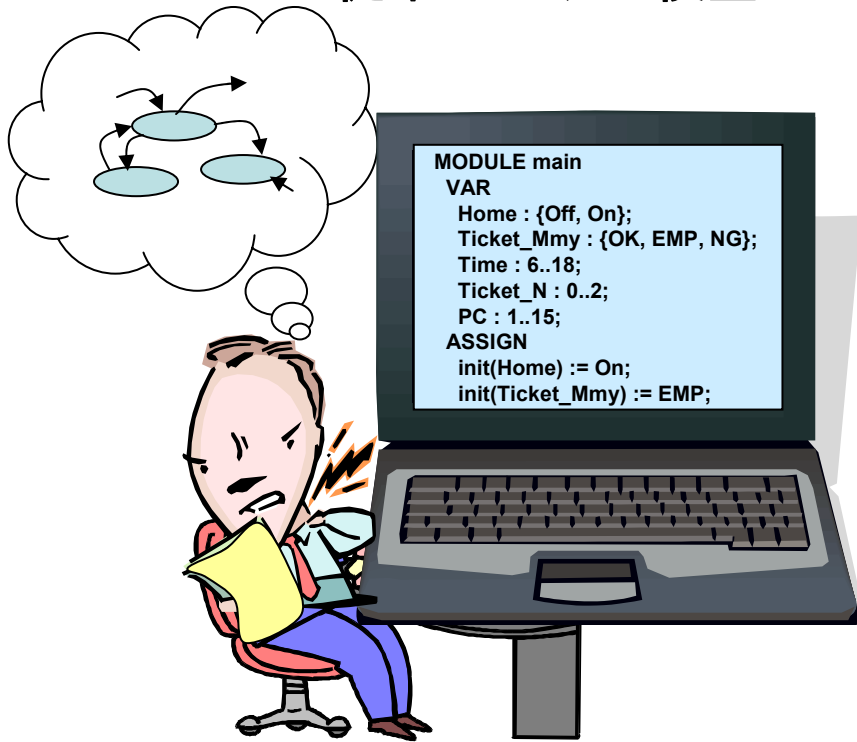
測定回路

センサ

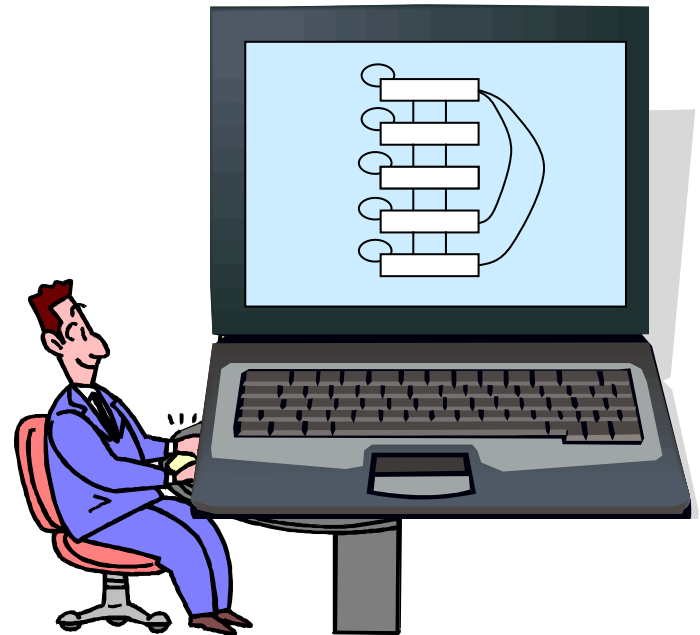
制御部

停電

## 従来のモデル検査



## モデル検査支援ソフトウェア



- ・記述力は高いが学習、経験が必要
- ・作業時間が長い

- ・学習、経験が短時間で済む
- ・作業時間を1/5に短縮

さらに、「見える化」の効果が

従来のモデル検査＝専用言語を読める人だけ使える

このモデル化  
合ってますか？

この検査式  
合ってますか？

専用言語  
読めません

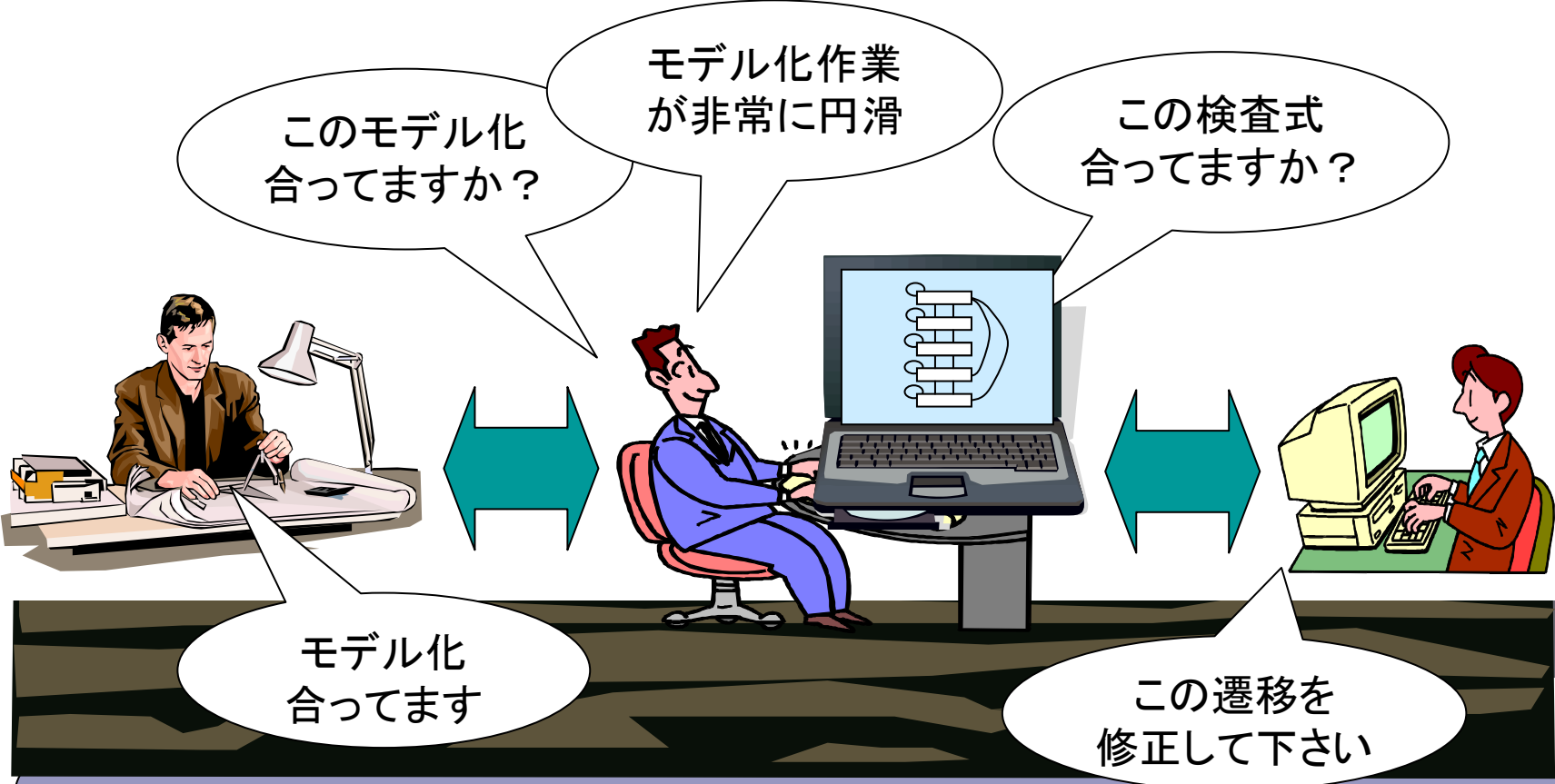
言っていること  
が意味不明

```
MODULE main
VAR
  Home : {Off, On};
  Ticket_Mmy : {OK, EMP, NG};
  Time : 6..18;
  Ticket_N : 0..2;
  PC : 1..15;
ASSIGN
  Init(Home) := On;
  Init(Ticket_Mmy) := EMP;
```

全員で情報共有できない

間違い探しできる人が限定される

支援ソフトウェアを使うと＝専用言語を読めない人も理解できる



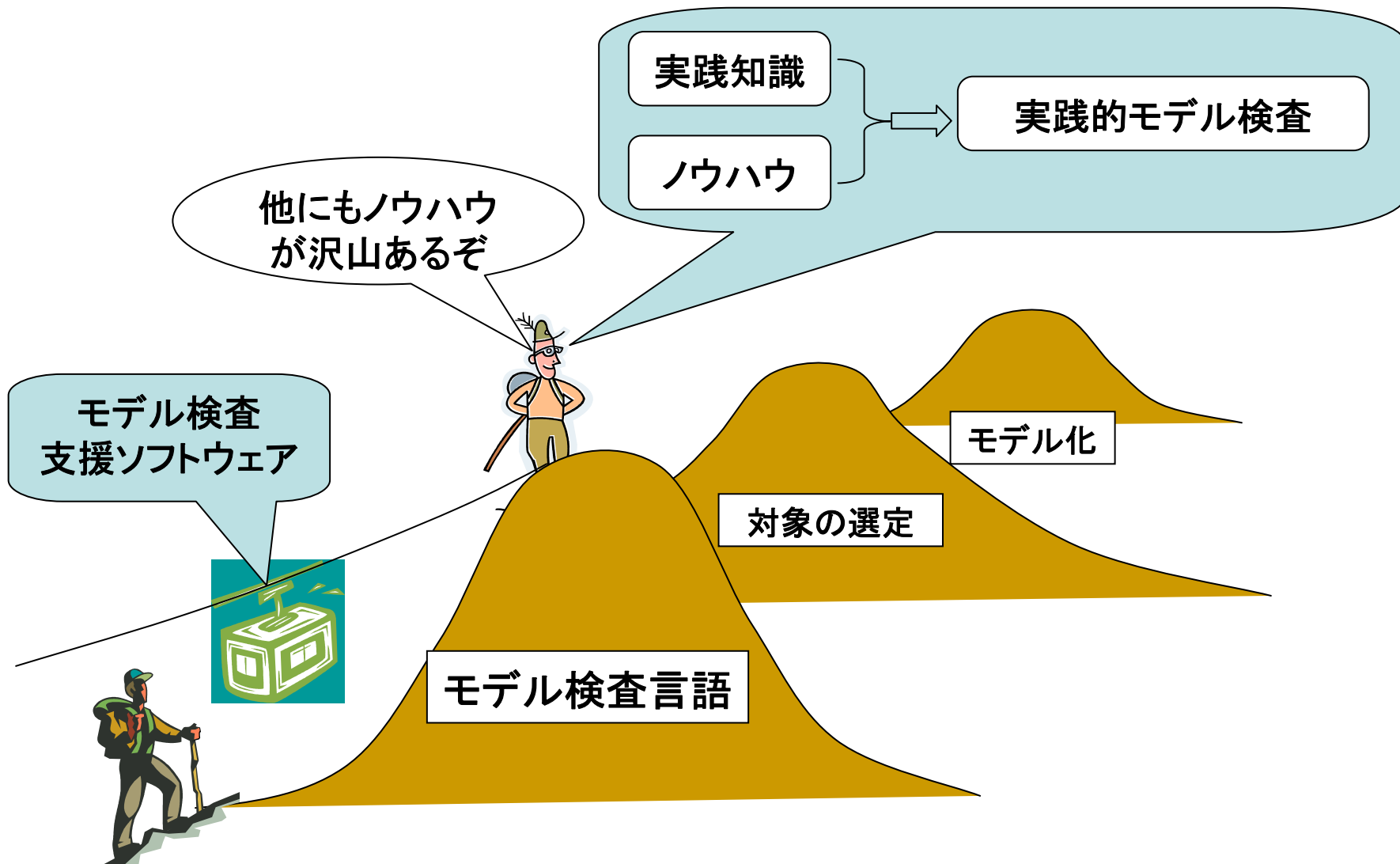
全員で情報共有できる

関係者でチェックできる

コミュニケーションの改善

## 6. 無駄、失敗を防ぐための実践ノウハウ

1. 対象の選定
2. 検査項目の設定
3. モデル化の留意点
4. モデル検査作業のノウハウ



# 検査対象の選定

実践では、できるだけ短時間で期待する成果が求められる

開発システムのどの部分の  
何をモデル検査するのか？



対象システム／検査内容は  
モデル検査に適しているか？

検査すべき項目は何か？

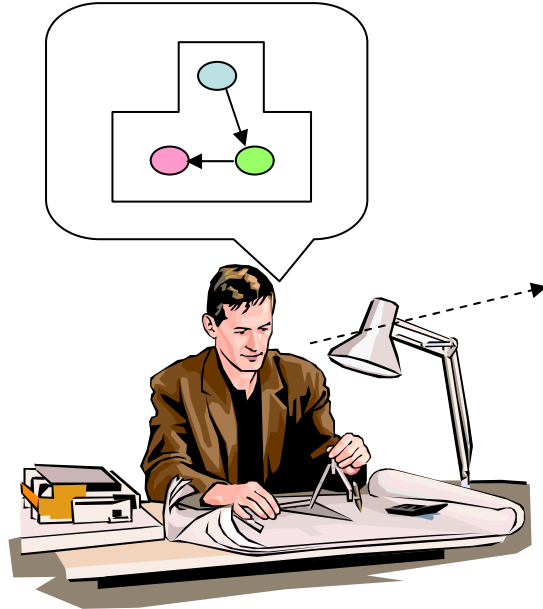
モデル検査の結果は、価値が  
共有できるか？



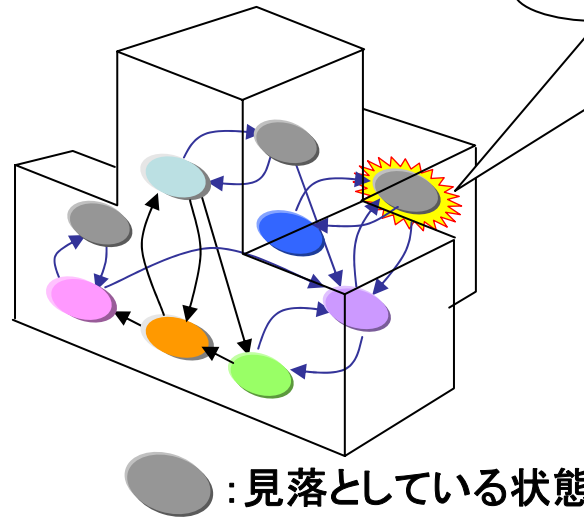
# モデル検査の向き不向き

適性： 状態遷移モデルの全パス検査が有効なもの

設計者の視点からの認識



ソフトウェア設計  
【起こり得る動き】



人間がトレースしきれず  
見落としてる危険状態

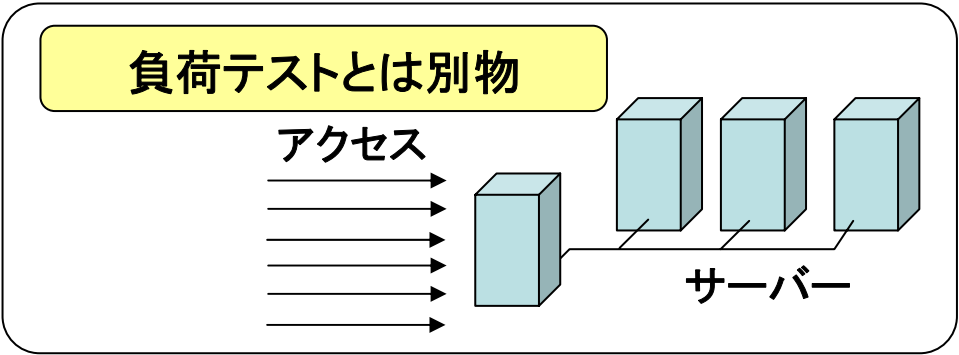
発見



```
MODULE main
VAR
  Home : (Off, On);
  Ticket_Mmy : (OK, EMP, NG);
  Time : 0..15;
  Ticket_N : 0..2;
  PC : 1..15;
ASSIGN
  Init(Home) := On;
  Init(Ticket_Mmy) := EMP;
```

● : 見落としている状態

- ・組み込みソフト
  - ・監視制御ソフト
  - ・Webアプリの画面遷移
- 他で実績



## 工程別モデル検査適用の特徴

対象	モデル規模	所用時間	特長
仕様書	中	中	検査の目的、外部環境に注意してモデル化要
設計書	大	長	規模によっては、モデルの分割、組合せを行う
コード (デバッグ)	小	短	検査目的に応じて大幅にモデルを縮小

評価が得られる(価値が共有されている)

何が大事か立場によって違う⇒価値の共有できるもの

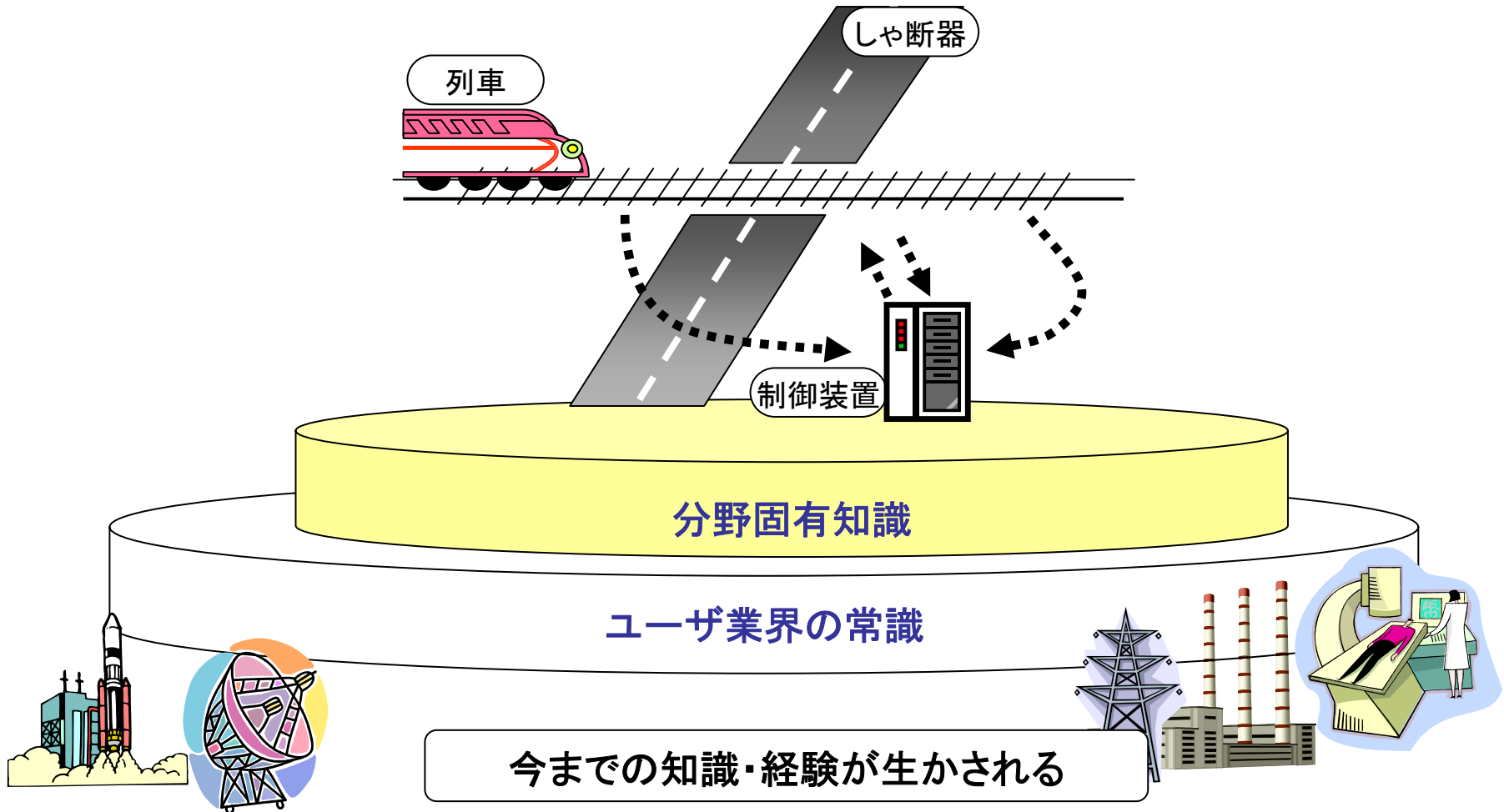


# 検査項目の設定

【 検査すべきこと⇒固有知識が不可欠 】

起きてはいけないことは何？

出来ていなければならないことは、何？



## 固有知識＋アルファ



個々の動作よりも全体として  
満足／保証すべきことは何？



従来とは視点を変えてみる



見落としに気付かない

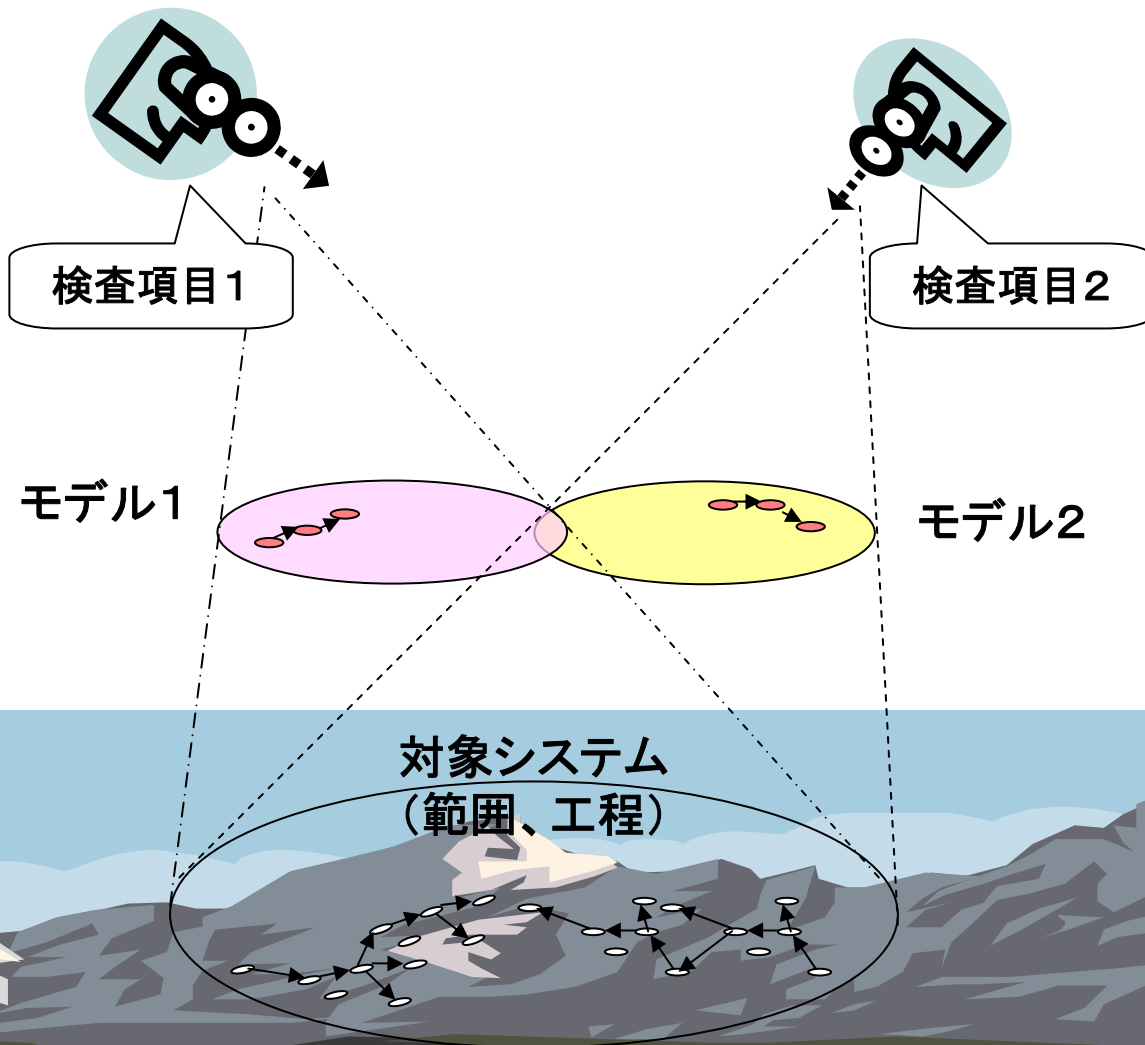
個別の要求は、間違いにくい

## 道具(モデル検査器)の選択

種類	入力言語	特徴
NuSMV	SMV	CTL式の検査 (時制と分岐を扱う)
NuSMV + (支援ソフトウェア)	フロー図 状態遷移図	モデル化／反例解析 が容易
Spin	Promela	LTL式の検査 (時制を扱う)
UPPAAL	状態遷移図	時間制約が扱える

どのモデル検査器も、モデル作成が必要⇔モデルの世界を検査する

# モデル化の留意点



# 検査項目がモデルを決める

起きてはいけない動作

出来ていなければならない動作

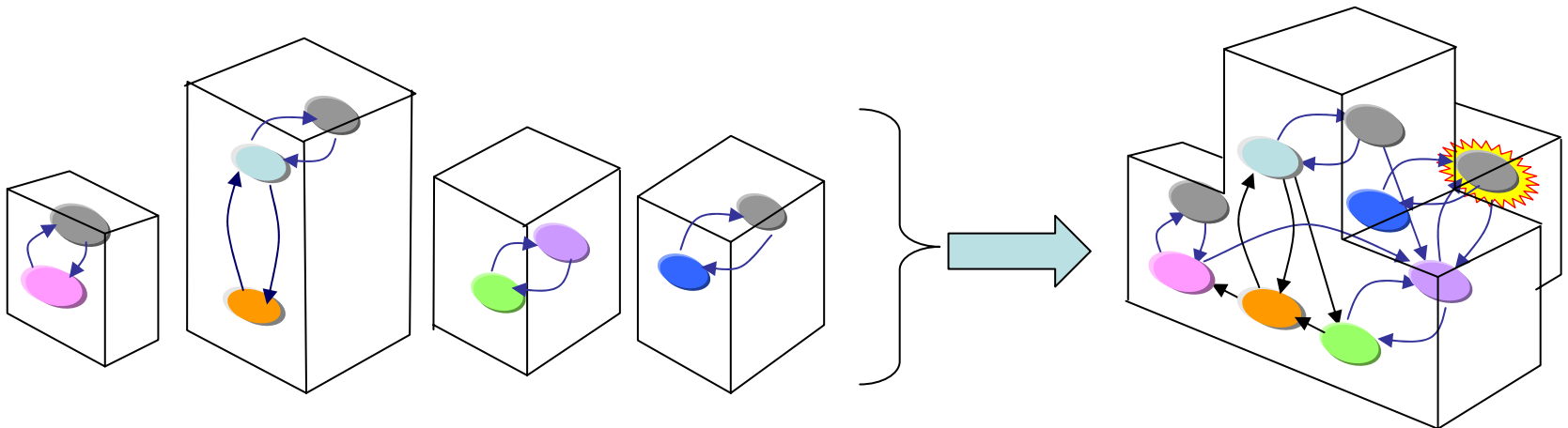
動作に必要な要素をモデルに取り込み

1. 検査したい**要素**がモデルに書かれていること（状態にないものは検査できない）



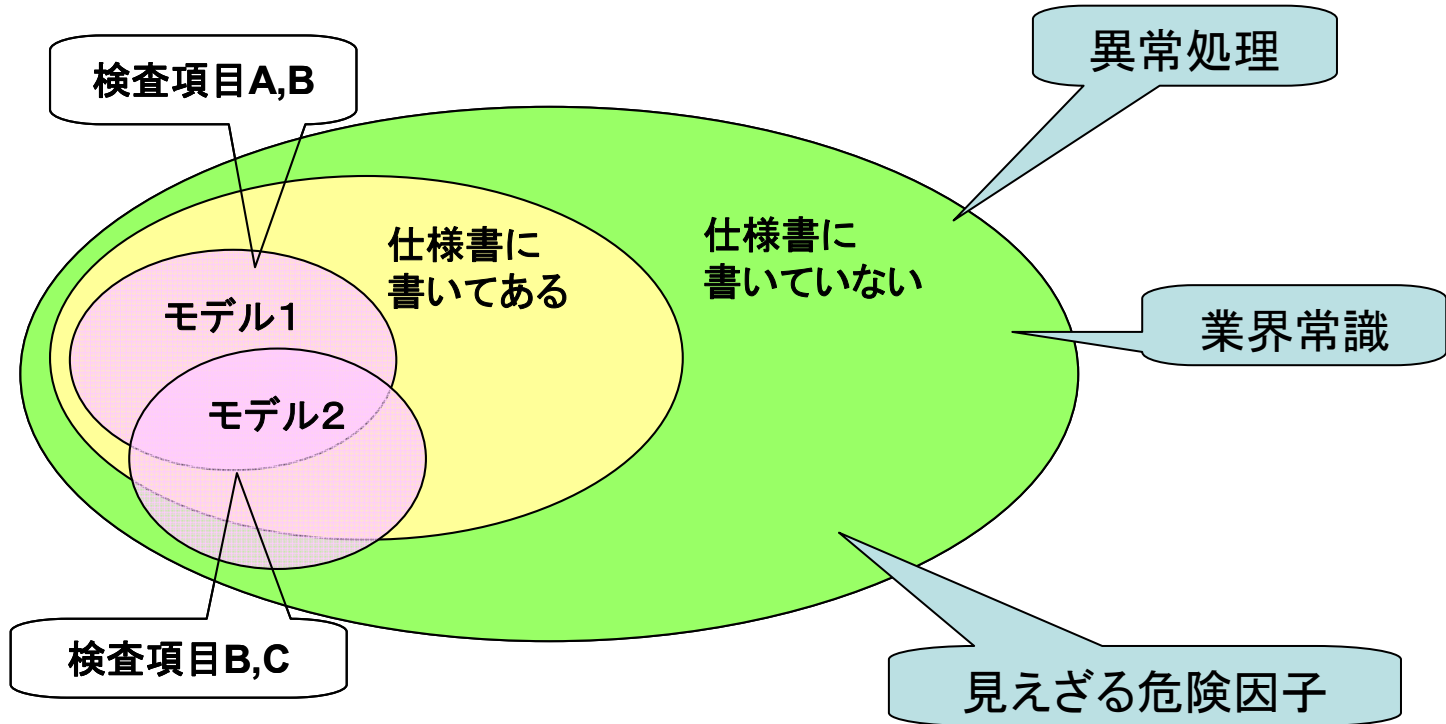
「どうなるか」は、分からなくて良い  
「この要素が怪しい」を入れておく

2. システムの部分モデルを作れば良い（全体モデルは自動生成される）



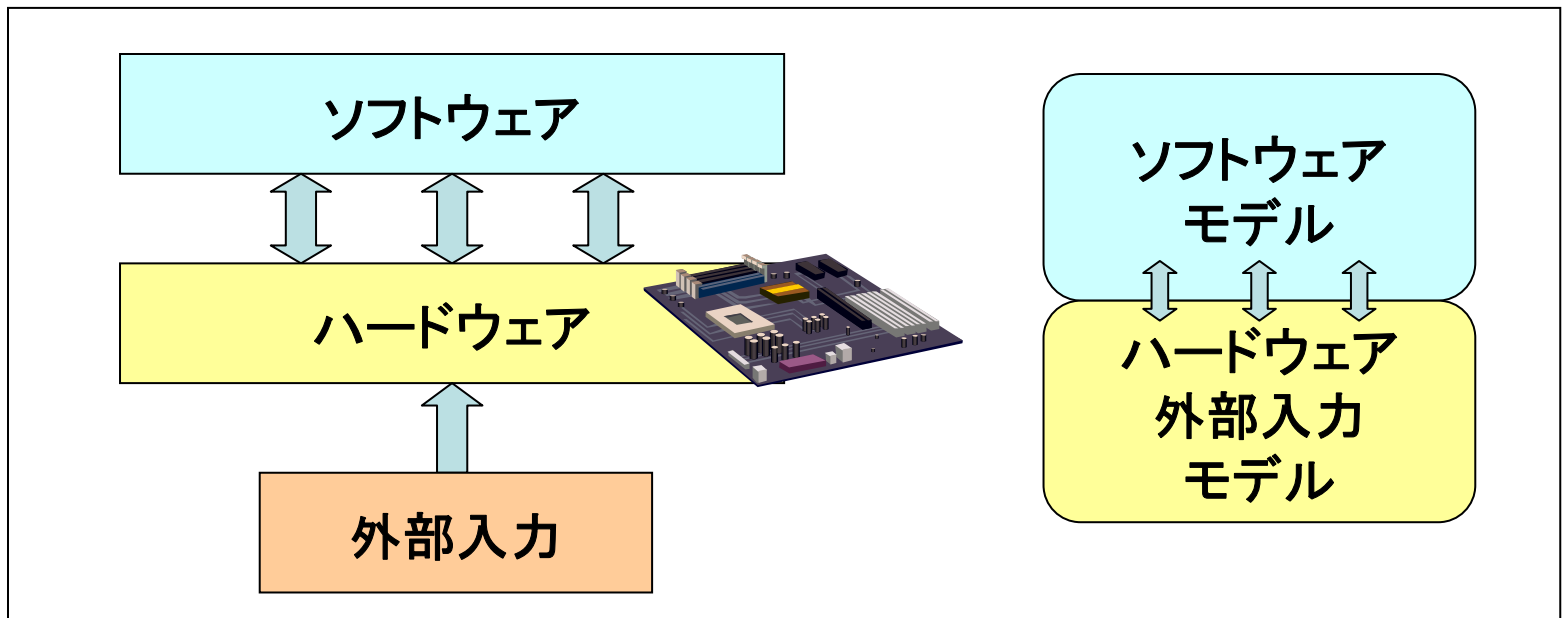
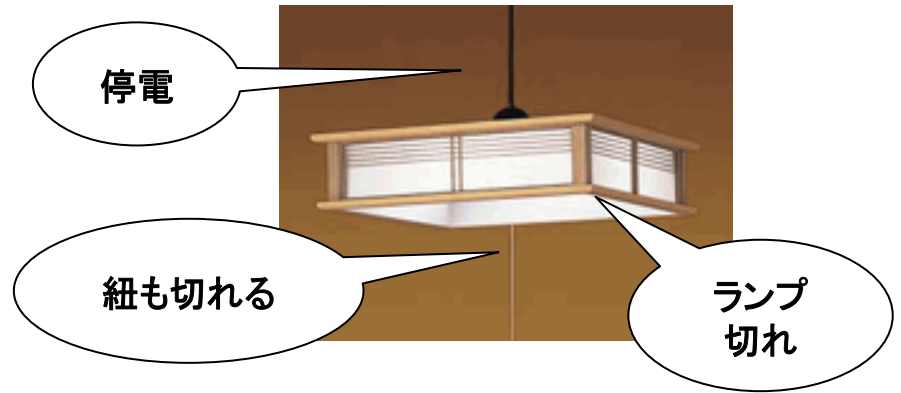


# 検査項目がモデルを決める

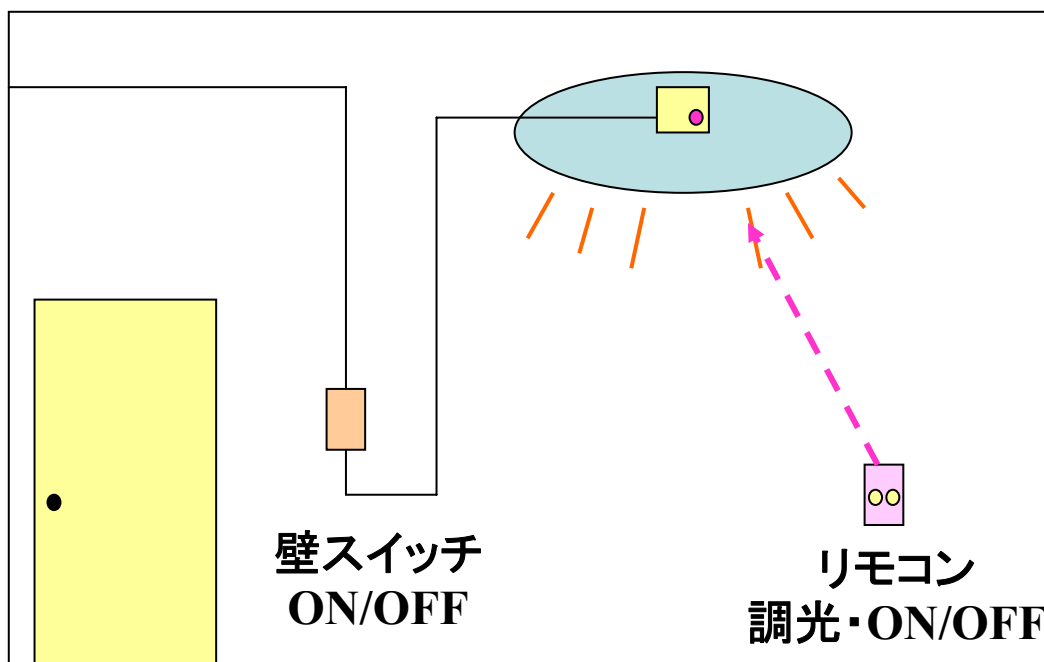
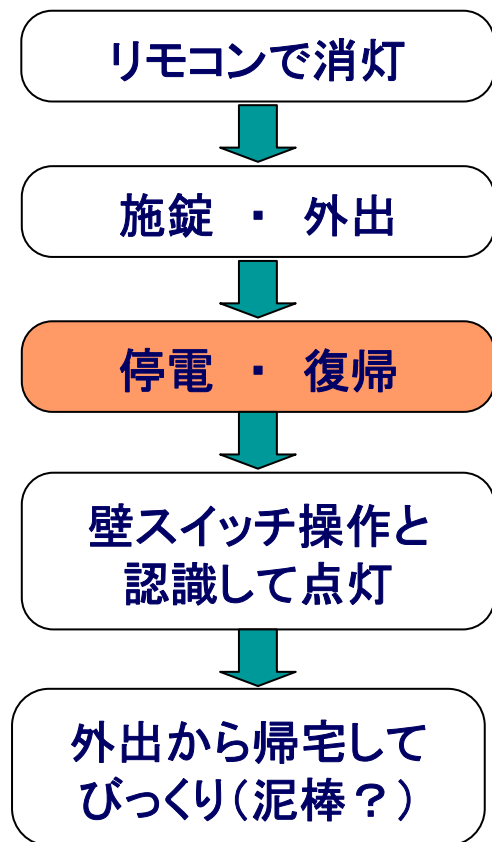


モデルがひとつとは限らないし、仕様書に書いてあるとも限らない

# 外部環境のモデル

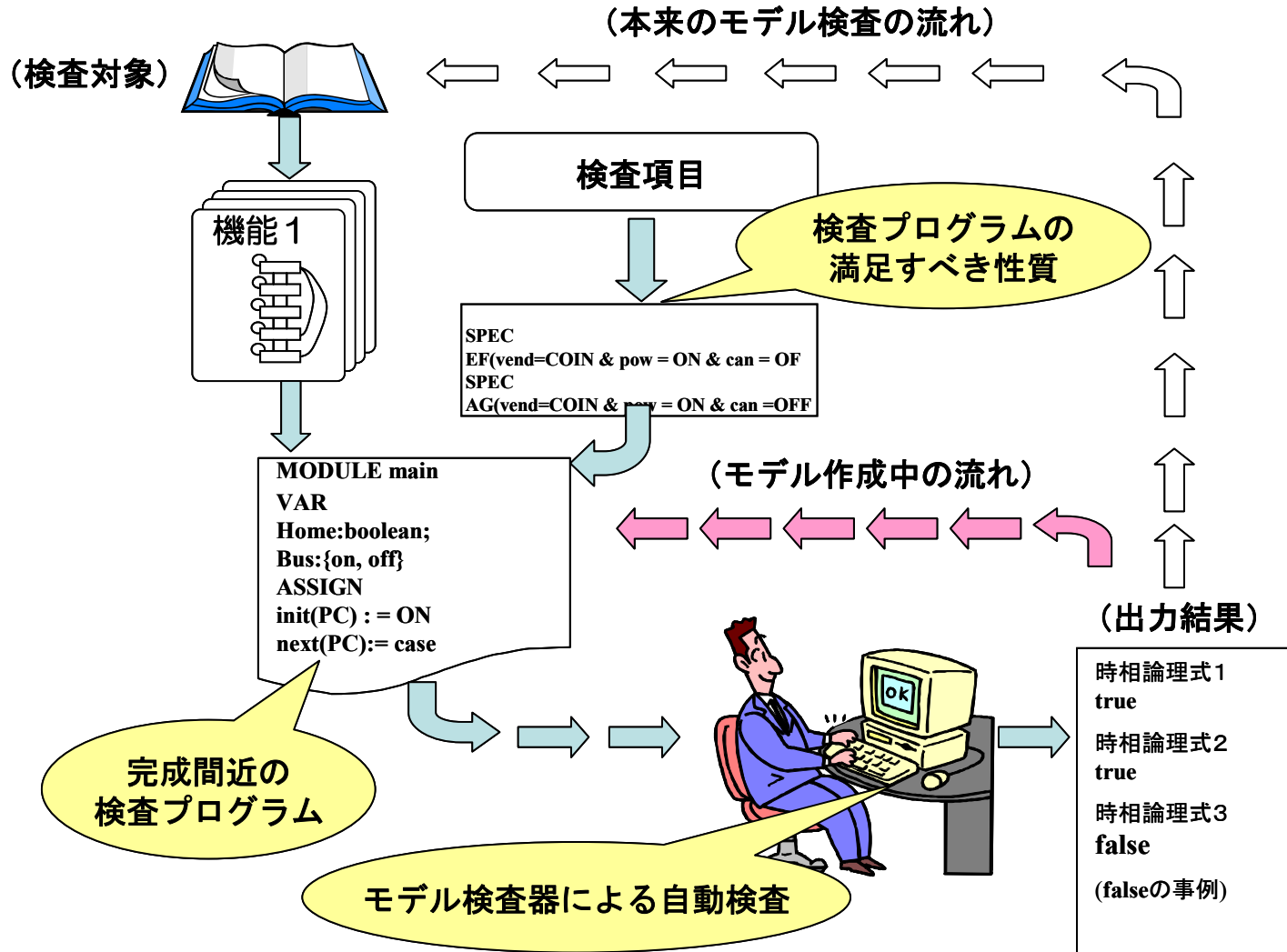


## 外部環境のモデル例



# モデル検査作業のノウハウ

## モデル作成途中での検査

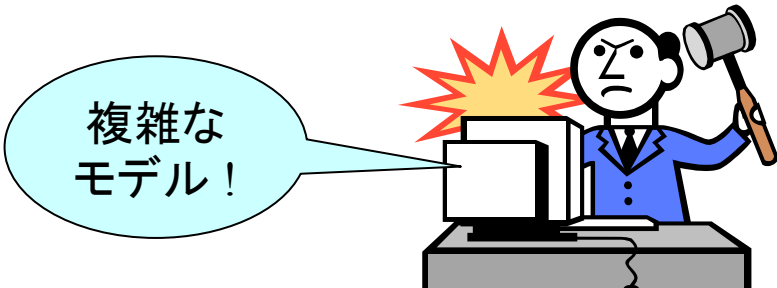


# 小さなモデルから検査開始

## 課題

状態爆発による無回答

## 対策

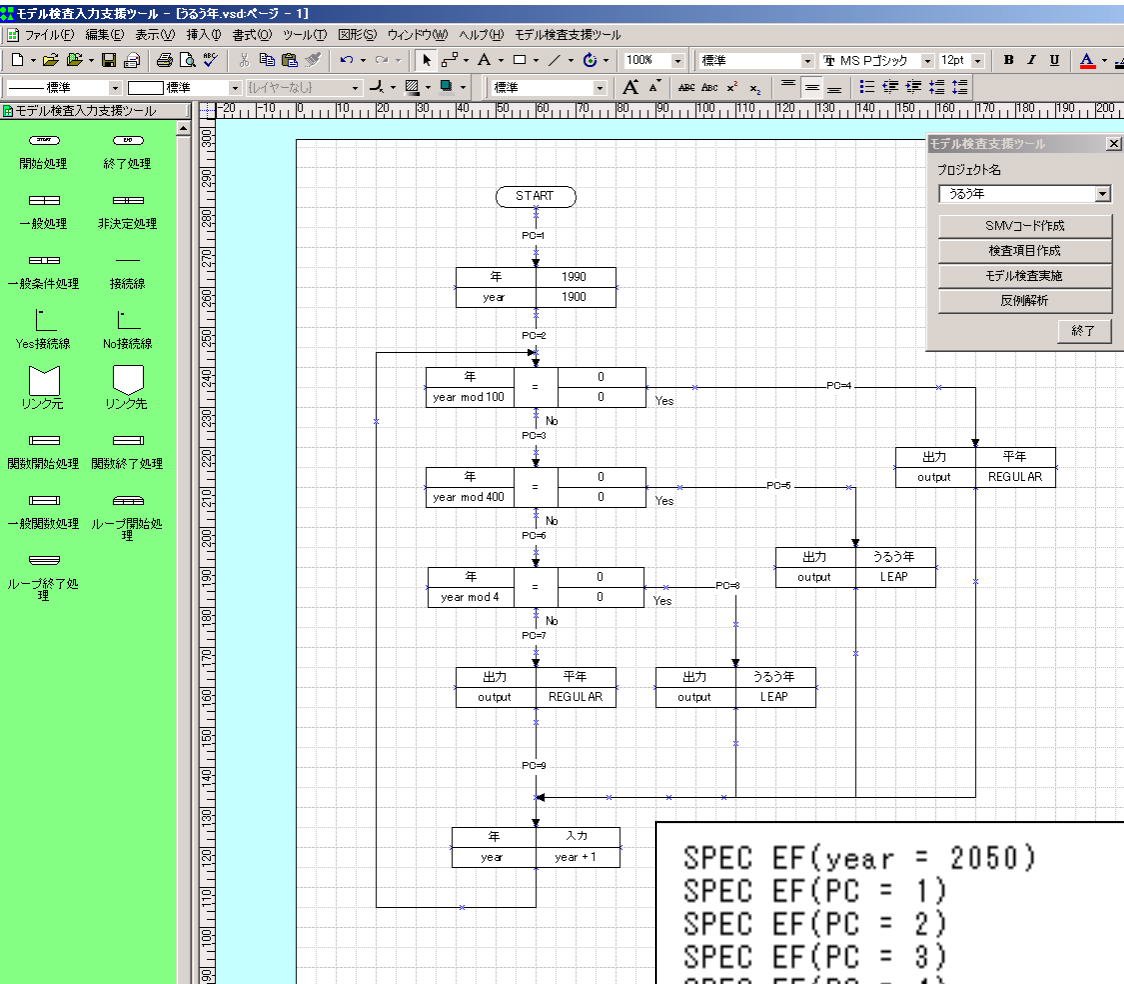


複雑な  
モデル!

実行継続・中止を判定するための  
閾値を求める

単純なモデル・検査項目から出来具合も見ながら複雑に

# モデルの事前確認



MODULE main

VAR

```

year : 1900..2050;
PC : 1..9;
output : {LEAP, REGULAR};
    
```

ASSIGN

```

init(year) := 1900;
next(year) :=
  case
    PC = 1 : 1900;
    PC = 9 : {1900..2050};
    1 : year;
  esac;
    
```

init(PC) := 1;

next(PC) :=

case

```

PC = 1 : 2;
PC = 2 & year mod 100 = 0 : 4;
PC = 2 & !(year mod 100 = 0) : 3;
PC = 3 & year mod 400 = 0 : 5;
PC = 3 & !(year mod 400 = 0) : 6;
PC = 4 : 9;
PC = 5 : 9;
PC = 6 & year mod 4 = 0 : 8;
PC = 6 & !(year mod 4 = 0) : 7;
PC = 7 : 9;
PC = 8 : 9;
PC = 9 : 2;
1 : PC;
esac;
    
```

- ・モデルは、まず疑う
- ・基本的な性質を検査

# 検査項目の事前確認

$AG(FRE = 1 \rightarrow AX(FDL=1))$

日本語の「～ならば」に近い意味(厳密には違う)

前提条件が成立しないと「True」になる

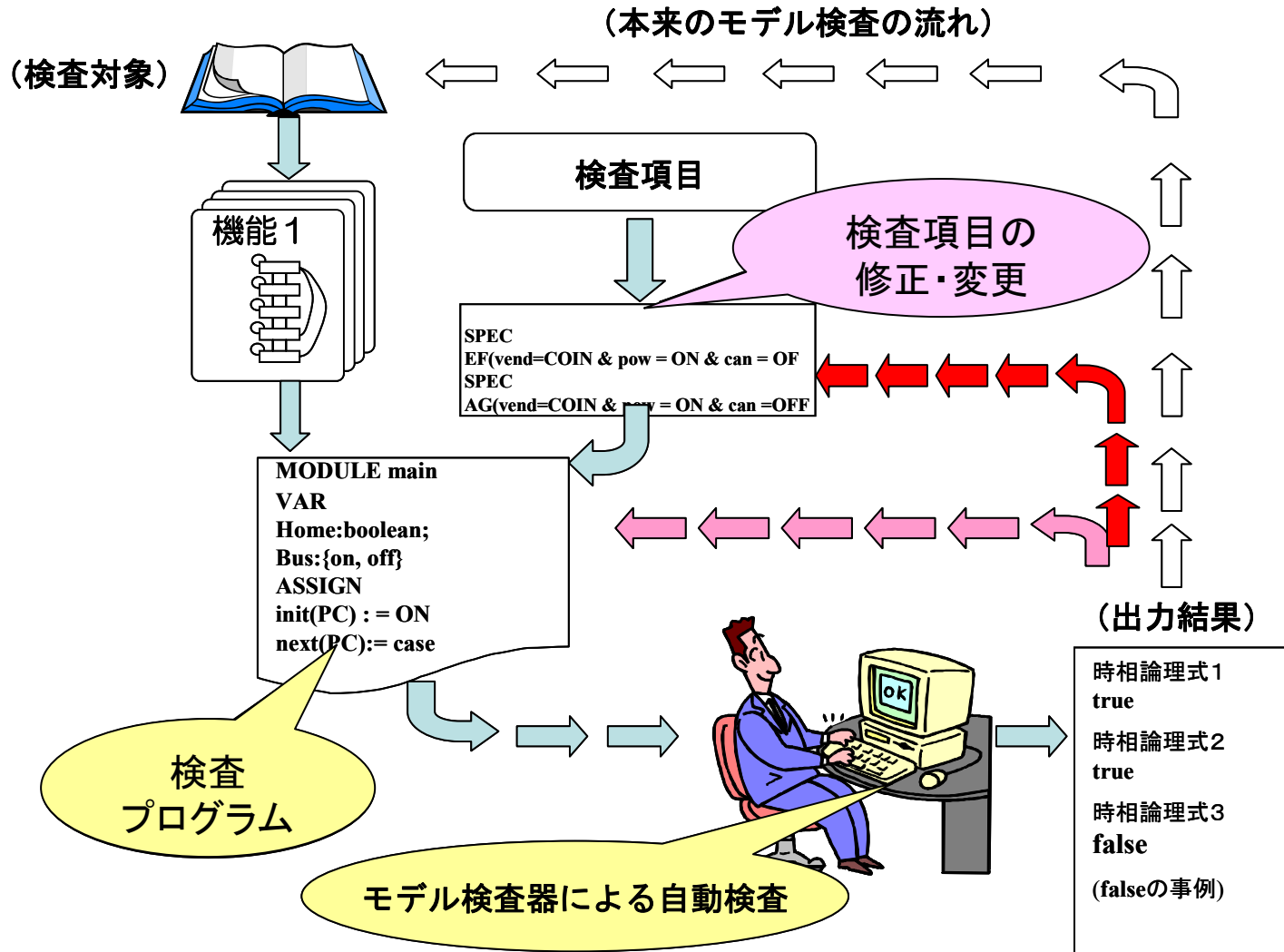
$EF(FRE = 1)$

前提条件の成立を先に検査しておく



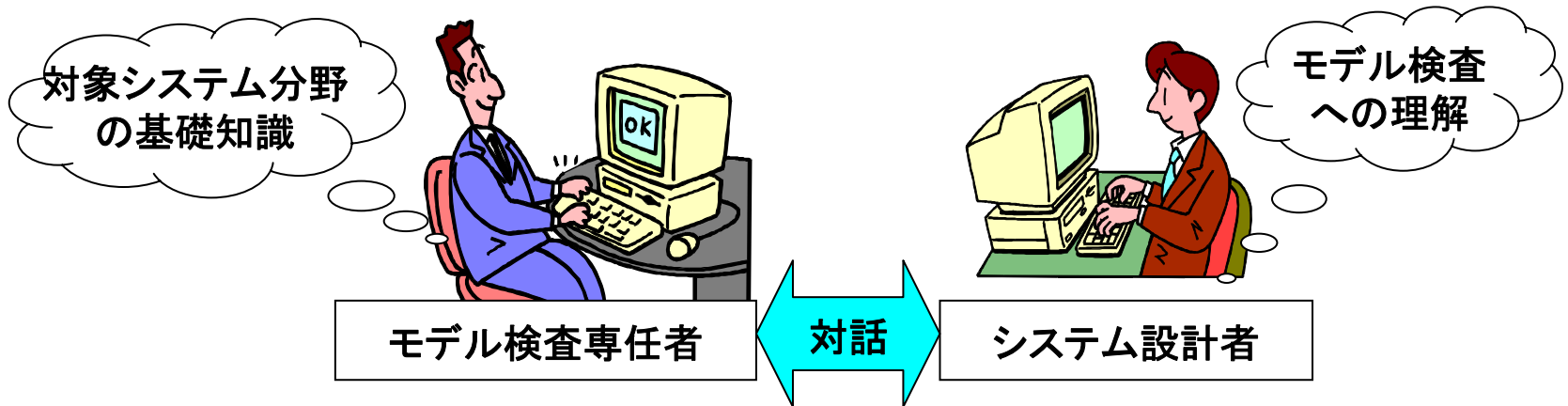
「おっと、しまった！」  
があります

# 何度でも気楽にモデルを回す



検査項目を変更しながらシステムの不具合動作を理解する





**【効率的な使い方をする】**

1. よく知っているものをモデル検査する。  
(対象をよく知っている人に協力してもらう)
2. 効果の大きいものをモデル検査する。
3. 最初から大きなモデルを作らない。

# モデル検査の特徴まとめ

## ・モデルをつくる作業が必要

- 検査対象の理解が深まる。
- 検査対象をよく知っている早い。
- モデル作成中に不具合を見つけることも多い。

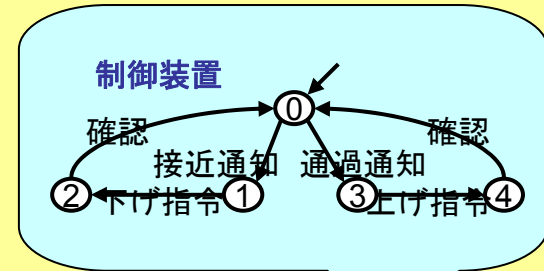
## ・必要な要素をモデル化できていれば、 不具合のメカニズムは考えなくて良い。

- とりあえず検査して、反例を見ることができる。
- 必要な要素が抜けると、モデルの作り直しが必要。

## ・反例を読み解く作業が必要

- 検査対象をよく知っている早い。
- 変数が増えると、読み取りも難しくなる。

## ・検査する内容は、人間が考える。



ご清聴ありがとうございました

後半【**始めよう！** **広げよう！** **モデル検査**】へ続く。