Vol.2014-SE-183 No.11
2014/3/19

情報制御システムのモデル検査における反例分析支援ツールの開発

大森 祐貴[1]　小山 恭平[1]　小飼 敬[2]　上田 賀一[1]

概要：情報制御システムを対象としたモデル検査は，安全性を確保する上で意味あるアプローチといえる．この際，任意の性質が満たされない実行系列（反例）を得ることで不具合の原因特定に役立てることができる．しかし，反例を理解するには専門的な知識が必要となり，反例から振舞い仕様の問題箇所を特定するのは困難である．そこで本研究ではモデル検査ツールにおける反例分析支援のための可視化ツールを開発した．具体的な事例を適用しその結果，本ツールが問題箇所の特定において，有責な情報を提供できることがわかった．

An Analysis Support Tool of Counterexamples for Model-Checking of ICS

Yuki Omorii[1]　Kyohei Oyama[1]　Kei Kogai[2]　Yoshikazu Ueda[1]

Abstract: Model checking for Information Control System is an effective approach in order to ensure safety. In this case, obtaining execution sequences not to satisfy properties(counterexample) is useful to find out cause of errors. However, we require expertise to understand the counterexample and it is difficult to identify error locations of behavior specification by the counterexample. Therefore, this study report about a visualizing tool which was developed to support analyzing counterexample on model checking tool SPIN. It found that developed tool can provide valuable information in identifying error locations.

Keywords: Information Control SystemModel checkingCounterexample analysisSPIN

1. はじめに

社会インフラを支える情報制御システムでは安全性・信頼性等の品質に対して重要な要求となっている．品質を確かめるためには設計レビューやテスト/デバッグを行う必要がある．しかし，これらの手法は基本的に人手で行われるため，設計レビューやテスト/デバッグではシステムの大規模化及び複雑かに対応しきれなくなっている [1].

このような問題を解決するため、形式手法の一種であるモデル検査が注目されている．モデル検査とは，システムの振舞い仕様をモデル化し，網羅シミュレーションを用いて，任意の性質を満たすか検査する手法である．モデル検査は，数学的理論がベースとなっており，検査が自動的に行われることから，大規模システムであっても品質を保証することが可能である，

モデル検査において，性質違反の実行系列が発見された場合，モデル検査ツールはその実行系列を反例として出力する．検査者はこの反例を用いて，振舞い仕様の問題箇所を修正する．しかし，反例の内容を理解するにはモデル検査ツールに関する専門的な知識が必要である．また，専門的な知識があっても，反例から振舞い仕様の問題箇所を特定することは困難である．

そこで本論文では，モデル検査ツール SPIN における反例分析を支援するため，SPIN における反例可視化ツールを開発した．

[1] 茨城大学
Ibaraki University
[2] 茨城工業高等専門学校
Ibaraki National College of Technology

2. 関連知識

2.1 モデル検査ツール SPIN

SPIN とはオートマトンをベースとしたモデル検査ツールである [2]．SPIN では，対象システムの動作仕様を SPIN 用仕様記述言語 Promela を用いてモデル化する．このように仕様記述言語でモデル化された動作仕様を一般的に振舞いモデルと呼ぶ．特に，Promela を用いて記述された振舞いモデルを Promela 記述と呼ぶ．

SPIN におけるモデル検査ではこの振舞いモデルから状態遷移モデルを作成する．次に，この状態遷移モデルにおいて任意の性質を満たさない状態遷移パスが存在するかを網羅的に調べる．

検査する形質は LTL 式 (線形時相論理式) か assert 文によって定式化される．LTL 式で定式化された性質は状態遷移モデルの探索中，常に評価される．一方で，assert 文で定式化された性質は検査者が意図したタイミングで評価される．SPIN は性質違反の状態遷移パスが見つかった時に，反例として trail ファイルを出力する．この trail ファイルを利用することで性質違反の状態遷移パスを再現することができる．

2.2 情報制御システム

情報制御システムは作業員が手作業で行っていた設備による制御を自動で行うため，作業員のノウハウをルールとして体系化し，そのルールをもとに自動または手動で制御を行うシステムのことである [3] ．主に電力や列車運行制御などのインフラ系の制御システムとして使用される．

情報制御システムは制御プログラムと物理環境から構築されている (**図 1**)，制御プログラムは定期的に，センサから制御対象の情報を取得し，仮想環境を更新する．次に，仮想環境の情報と作業員のノウハウを基に，設備へ与える操作指示を決定する．最後に物理環境の設備へ操作指示を出力する．

物理環境では，設備が制御プログラムから操作指示を受け取り，それに合わせて，自身の状態を更新する．制御対象は，設備の状態を基に，任意のタイミングで自身の状態を更新する．

制御プログラムの振舞いと物理環境の振舞いから，情報制御システムは**図 2** に示す振舞いを有限回実行するとみなす．ここで，設備の状態更新は制御プログラムの操作指示出力と同時に行われている．また，図 2 に示す振舞いを情報制御システムの動作単位と定義する．

2.3 情報制御システム記述言語 RASYS

RASYS とは，図 1 の制御プログラムの振舞いを記述するルールベースの仕様記述言語である [4]．システムの振舞いや実行条件はそれぞれ専用の記法によってモデル化され，複数のモデルを組み合せることで制御プログラムの振舞いを表現している [5]．本論文において，RASYS モデルとは，RASYS を用いて記述された動作仕様である．

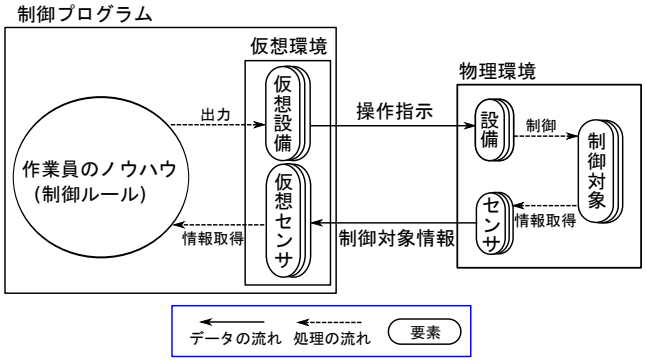

図 1 情報制御システムの概念図
Fig. 1 Outline of ICS

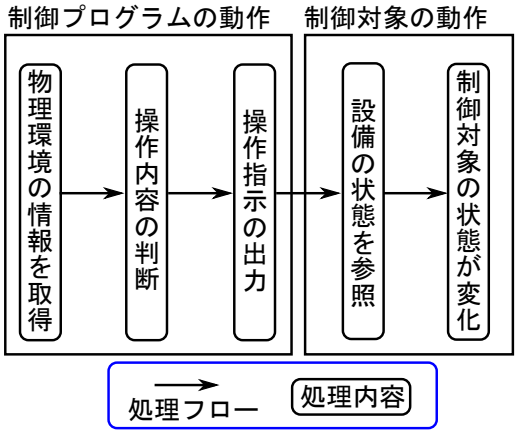

図 2 情報制御システムの動作
Fig. 2 process flow of ICS

モデルは以下のような要素から構成されている．

オブジェクト
仮想環境を構成する仮想設備と仮想センサに対応する．オブジェクトは状態や他オブジェクトとの関連を表す属性とその属性値を持つ．オブジェクトの状態を表す属性値は基本的に抽象化された離散値である．

アクタ
オブジェクトが持つ属性の参照や更新などの操作を表す．アクタは制御プログラムの制御ロジックや制約条件をルールベースのモデルとして表現している．

アクタシナリオ
アクタを順次呼び出して実行する関数手続きにて実装する．前のアクタの実行が終了しない限り，次のアクタは実行しない．

2.4 モデルの記法

実行可能な情報制御システムとなるために，6 種類のモデル記法でシステムの構成要素を記述する．RASYS では，

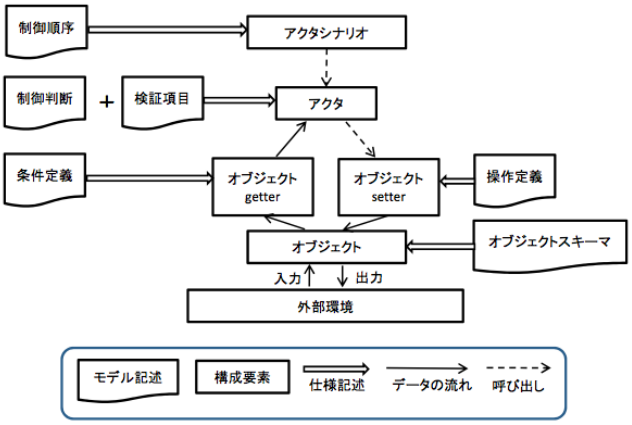

図 3 RASYS の構成要素とモデル記述の対応
Fig. 3 Contribution elements and model discriptions

表 1 条件定義例
Table 1 Example of conditional definition

ルール				
No	属性定義		条件	
	属性名称	属性値	オブジェクト A 属性 B	オブジェクト A 属性 C
1	属性 A	属性値 a1	属性値 b1	属性値 c1
2		属性値 a2	属性値 b2	属性値 c2

オブジェクト，アクタ，アクタシナリオが以下の 6 種類の記法を利用してモデル化される．システムの構成要素とモデル記法の対応を**図 3** に示す．

- 制御順序定義
  制御判断が表す制御内容の実行順序を表す．
- 制御判断
  制御判断は制御対象となるオブジェクトと実行条件 を規定する部分と，制御ルールを規定する部分に分かれている．制御ルールは，制御条件と条件が成立した際に実行する項目の組で構成されている．
- 条件定義
  条件定義の例を**表 1** に示す．条件定義は制御判断や操作定義で条件として使用する属性値を定義している．属性値は 2 種類の方法で値が決定される．1 つはシステムの外部から離散化された値を受け取る方法である．もう 1 つは複数の属性値の組合せによって値を決定する方法である．条件定義で定義する属性値は後者である．条件である属性値の組合せと，対応する属性値の組を定義する．
- 操作定義
  制御判断において制御条件が成立するときに実行する操作を定義する．操作定義は条件と条件成立時に実行する項目で表す．
- オブジェクトスキーマ
  オブジェクトスキーマの例を**表 2** に示す．オブジェクトが持つ属性の一覧と他のオブジェクトとの依存関係を表す．属性毎に構造や型，初期値を定義している．オブジェクトスキーマで定義されていない属性は，制御判断など他の記法で使用することはできない．使用する属性は，必ずオブジェクトスキーマで定義する必要がある．

表 2 オブジェクトスキーマ例
Table 2 Example of object schema

No	種別	属性	構造	型	属性値	初期値
1	状態	属性 A	単一	選択	属性値 X 属性値 Y	○
2	関連	関連 B	単一	[スキーマ B]		

RASYS 記述を SPIN で検査するため，RASYS 記述から Promela 記述への変換方法が既に存在している [3]．これにより，RASYS に関する知識があれば，SPIN を用いて RASYS 記述にモデル検査を適用し，制御ロジック内のルールを検査することが可能である．

このとき，性質違反の状態遷移パスを発見すると，SPIN は Promela 記述の状態遷移パスを反例として出力する．よって，反例を分析するには，Promela の知識が必要となる．また，Promela の知識がある場合でも，Promela 記述の状態遷移パスから RASYS 記述内で定義した制御ロジックのうち，問題のあるルールを特定することは困難である．そのため，検査者が反例を分析しやすい形で可視化するツールを開発した．

3. アプローチ

3.1 概念

検査者が反例分析をしやすい形で可視化するため，次に 2 つのアプローチを用いる．

- 反例中の各遷移を RASYS 記述のルールに対応させる (**図 4**).
- 複数の反例を用いて，反例のパスにおける状態遷移モデルを作成する (**図 5**).

1 つ目のアプローチに関して, SPIN で生成される反例は, Promela 記述における状態遷移パスである．また, Promela 記述における各状態遷移と RASYS 記述のルールの関係に関する情報が存在しない．そのため，反例から，RASYS 記述において，どのルールが適用されたのか把握するのが困難である．よって，Promela 記述に RASYS 記述との対応関係情報を埋め込み，Promela 記述の状態遷移と制御ルールの対応関係取得を可能にする．

2 つ目のアプローチに関して，単一の状態遷移パス (反例) だけでは，どの状態に問題があるのか特定するのが困難である．よって複数の状態遷移パスを用いて，状態遷移モデルを構築することで，複数の反例で共通して出現する

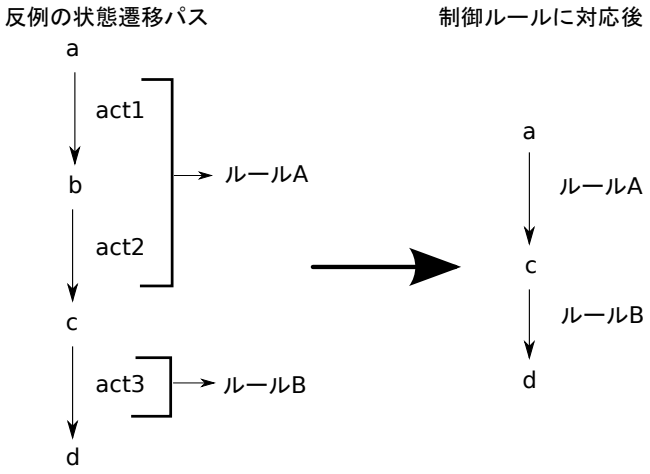

図 4 アプローチ 1 の概念図
Fig. 4 Image of approach1

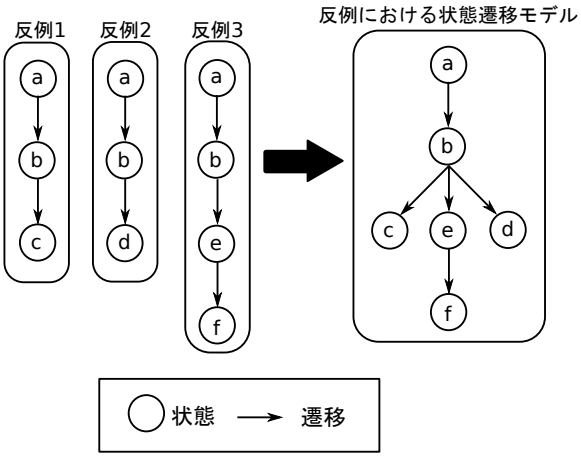

図 5 アプローチ 2 の概念図
Fig. 5 Image of approach2

状態を特定することが可能である．複数の反例に出現する状態は不具合の原因の可能性が高いため，このアプローチで原因の特定が容易になると考えられる．

3.2 実現方法

前述のアプローチを実現するため，反例の可視化ツールを開発した．本ツールの概要図を**図 6** に示す．ツールを用いたモデル検査と反例分析の手順を示す．

(1) Promela 記述内に RASYS 記述内ルールとの対応関係を埋め込む．
(2) SPIN を用いてモデル検査を行う．このとき，全ての反例を出力するように SPIN のオプションを指定する．
(3) 反例として出力された各 trail ファイルを分割する．
(4) 分割後の各 trail ファイルを用いて，SPIN による状態遷移パスの再現を行い，使用されたルールと，状態遷移パスにおける最終状態を取得する．
(5) 取得したルールと状態から，状態遷移モデルを作成する．

ここで，手順 1 と手順 2 は人手で行い，それ以外の手順は

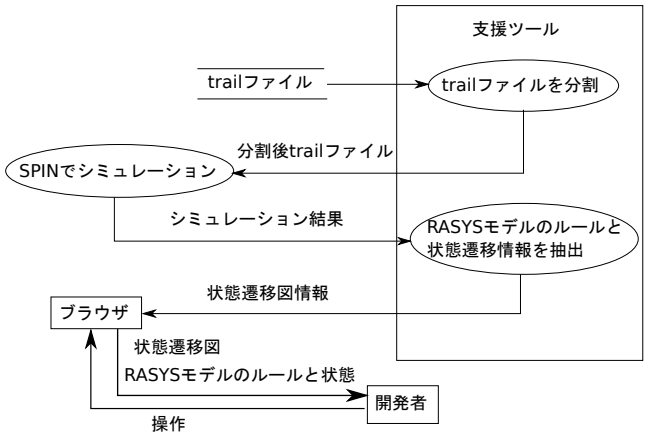

図 6 支援ツールの概要図
Fig. 6 Outline of support tool

ツールが自動的に行う．

3.2.1 Promela と RASYS の対応関係埋め込み

SPIN がシミュレーションによって Promela 記述における状態遷移パスを出力するが，それだけでは，各遷移と RASYS 記述内の各ルールとの関係を知ることはできない．そのため Promela 記述内に，RASYS 記述内でどのルールが使用されたかを示すフラグ変数を埋め込む．

これにより状態遷移パスの再現をするときにそのフラグ変数を確認することで，状態遷移時に使用されたルールを知ることができる．フラグ変数は配列で定義し，それを d[i-1] としたとき，d は RASYS のルール名，i は RASYS の表の行番号と対応する

3.2.2 状態遷移パスの抽象化

RASYS 記述と Promela 記述では抽象度が異なっているため，Promela 記述における複数の遷移が RASYS 記述内の 1 つのルールに対応する．よって，使用されたルールを調べるには複数の状態遷移を調べる必要があるため，使用されたルールを調べるのは困難である．そこで，状態遷移パスを抽象化し，1 つの遷移に対して，使用されたルールを対応させる．これにより，1 つの遷移を調べることで使用されるルールを調べることができる．

そのため，図 2 に示す振舞いモデルの動作に着目する．情報制御システムの 1 動作単位では，仮想設備の状態から，条件を満たすルールが必ず実行される．よって，状態遷移パスを情報制御システムの動作単位レベルで抽象化する．つまり，状態遷移パスにおいて，情報制御システムの 1 動作単位に対応する状態遷移を 1 つにまとめる．

状態遷移パスを 1 つにまとめるため，シミュレーションに使う trail ファイルを分割する．trail ファイルは図 7 のようになっている．一行目の-4:-4:-4 は SPIN にどのようなシミュレーションを行うかを指定するものである．2 行目以降のコロンで区切られた 3 列目の値は，SPIN が網羅的に検査を行うために付けられた Promela 記述の状態 ID である．2 列目値は Promela 記述の状態 ID が属するプロセ

```
-4 : -4 : -4
1  : 0  : 2
2  : 1  : 14
3  : 2  : 23
4  : 2  : 25
5  : 3  : 21
```

図 7 trail ファイルの例
Fig. 7 Example of trail file

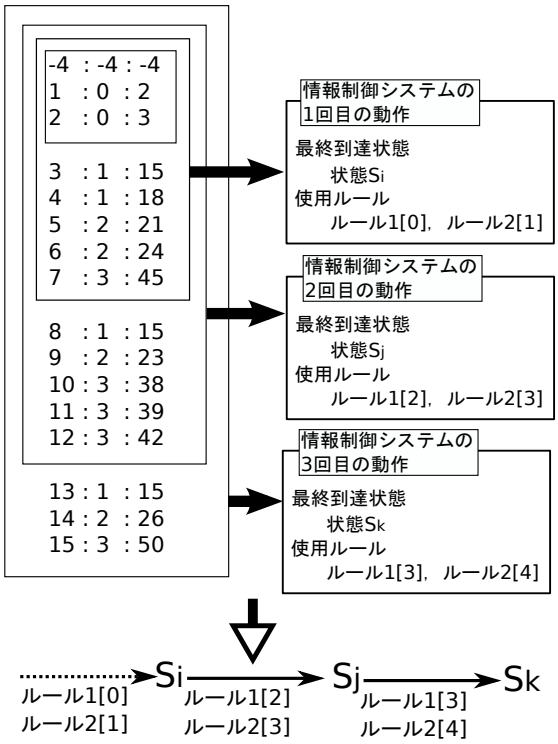

図 8 trail ファイルの分割方法と状態遷移情報の対応
Fig. 8 Relation of divided trail file and state transitions

スの番号である．プロセス番号 0 は init プロセスで始めの 1 回のみ実行される．1 列目の値は，検査のときに Promela 記述における状態 ID が実行された順番である．

状態遷移パスの再現では trail ファイルを読み込み，最終行を読み込んだ時点の状態 (ルールの使用状況を含む) を出力する．そこで，trail ファイルの任意の行以下を削除する．この処理を行うことで任意の状態を出力させることができる．本ツールでは，情報制御システムの各動作単位ごとの到達状態を取得するため，**図 8** のように，プロセス番号 1 から 3 番目まで実行され，再びにプロセス番号 1 に戻る前までを一単位として trail ファイルを分割した．

4. 適用事例

本ツールの適用実験のために，RASYS によって記述された 4 階建てエレベータモデルを用意した．

4.1 エレベータモデル

システムの動作仕様を以下に示す．

- 前提条件
  - (1) 目的階層に関してのみドアの開閉を行い，その他階については移動に関してのみ行う．
  - (2) 目的階層のボタンは 2 つ以上同時に押されることはない．
  - (3) タイマがアクティブの時はボタンの入力をうけつけない．
  - (4) ボタンの入力は常にエレベータが止まっている場合のみである．
- (1) タイマについて
  - (a) ドアを開く時にアクティブとなり，一定時間が経過するとウェイトになる．
- (2) ドアの動きについて
  - (a) エレベータが停止しているときにタイマがウェイトになると閉じる．
  - (b) エレベータが停止しているときに閉ボタンが押下されると閉じる．
  - (c) エレベータが停止しているときに開ボタンが押下されると開く．
  - (d) 目的階層に到着したときにドアは開く．
- (3) エレベータの動きについて
  - (a) ドアが閉じているときのみ移動する．
  - (b) 目的階層はボタンが押下されることで決定する．
- (4) ボタンの動きについて
  - (a) ドアが開閉するたびに，開閉ボタンと停止している階に対応するボタンは押下なしになる．
  - (b) 目的階層に到着すると，目的階層に対応するボタンが押下なしになる．
  - (c) 進行方向がない時に，ボタンが押下されると現在階から目的階層の方向に進行方向が設定される．

エレベータのオブジェクトスキーマを**表 3** に示す．

4.2 エレベータの制御ルール

エレベータの制御ルールを示す．制御ルールはルール名 (フラグ変数名)，の次に条件を列挙している．これらを条件定義の記法にしたがって記述した．

- (1) 現在状態判定 (CheckCurrentState)
  - (a) いずれかのボタンが押されているとき，「現在状態」を「移動状態」とする．
  - (b) いずれのボタンも押されていないとき，「現在状態」を「待機状態」とする．
- (2) 移動要求判定 (CheckMovementRequest)
  - (a) 現在階と異なる階のボタンが押されていないとき，「移動要求」を「なし」とする．
  - (b) 「現在状態」が「移動状態」のとき，「移動要求」を「あり」とする．
- (3) 進行方向判定 (CheckDirection)
  - (a) 現在階よりも上の階のボタンが押されたとき，「進行方向」を「上方」とする．

表 3 エレベータのオブジェクトスキーマ
Table 3 Object schema of elevator

No	属性	属性値
1	1 階指定ボタン	押下あり 押下なし
2	2 階指定ボタン	押下あり 押下なし
3	3 階指定ボタン	押下あり 押下なし
4	4 階指定ボタン	押下あり 押下なし
5	開ボタン	押下あり 押下なし
6	閉ボタン	押下あり 押下なし
7	ドア状態	開 閉
8	タイマ状態	アクティブ ウェイト
9	進行方向	上 下 なし
10	現在階	1 階 2 階 3 階 4 階
11	現在状態	到着状態 待機状態
12	移動要求	あり なし
13	ドア開閉要求	開ける 閉じる なし
14	目的階層	1 階 2 階 3 階 4 階 なし

(b) 現在階よりも下の階のボタンが押されたとき,
「進行方向」を「下方」とする.

(4) 目的階層判定 (CheckDistnation)
(a) ボタンが押された階を「目的階層」とする.
(b) ボタンが押されていないとき,
「目的階層」を「なし」とする.

(5) ドア開閉要求判定 (CheckDoorState)
(a) 目的階に到着したとき,
「ドア開閉要求」を「開ける」とする,
「タイマ」を「アクティブ」とする.
(b) ドアが開いていて,
かつ「タイマ」が「ウェイト」のとき,
「開閉要求」を「閉じる」とする.
(c) どちらにも該当しないとき,
「ドア開閉要求」を「なし」とする.

4.3 検査性質

検査性質は以下のものを assert を用いて検査する.

(1) 現在階が 1 階のとき，進行方向が下方になることはない.
(2) 現在階が 4 階のとき，進行方向が上方になることはない.

また，この検査性質を満たさないようにするため，4.2 節で定義したルールの進行方向判定を 3 ヶ所書き換える．書き換え箇所を以下に示す.

- 7 行目のルール「現在階が 1 階のとき，3 階指定ボタンが押されたら，進行方向を上方にする.」を「現在階が 1 階のとき，3 階指定ボタンが押されたら，進行方向を下方にする.」とする.
- 33 行目のルール「現在階が 4 階のとき，1 階指定ボタンが押されたら，進行方向を上方にする.」を「現在階が 4 階のとき，1 階指定ボタンが押されたら，進行方向を上方にする.」とする.
- 34 行目のルール「現在階が 4 階のとき，2 階指定ボタンが押されたら，進行方向を下方にする.」を「現在階が 4 階のとき，2 階指定ボタンが押されたら，進行方向を上方にする.」

このルールに対応する埋め込んだフラグは CheckDistnation[8], CheckDistnation[34], CheckDistnation[35] である.

4.4 適用結果

バグを埋め込み検査を行った結果，15 個の反例が見つかった．出力された trail ファイルを用いて支援ツールを使うと**図 9** の状態遷移図が出力された．末端のノードが反例となった状態を表している．本ツールはノードをクリックすると，対象オブジェクトの状態の情報を得られる．また，エッジをクリックすると，その遷移に利用された RASYS 記述のルールを得られる．反例となる遷移を持つノードをそれぞれ A，B，C とする．また，A，B，C それぞれの状態と root の状態を**表 4** に示す．A から反例となる遷移と，そうではない遷移に使用されたルールは共に CheckCurrentState[3]，CheckMovementRequest[13]，CheckDirection[8]，CheckDoorState[18] の 4 個である．どちらの遷移にも同じルールが使用されているので反例の原因はわからない．B も同様に使われたルールは CheckCurrentState[3]，CheckMovementRequest[13]，CheckDirection[8]，CheckDoorState[18] の 4 個で反例の原因はわからない．C から反例となる遷移は 10 個ある．それぞれの遷移で使用されたルールを**図 5** に示す．適用されたルール，C の状態，4.2 節で定義したルールをもとに手動で属性の変化を確認する．例として，表 5 の 1 行目の

表 5 C から反例への遷移に使用されたルール
Table 5 Rule adopted in transion from nodeC to counterexample

No	使用されたルール
1	CheckCurrentState[3],CheckMovementRequest[13],CheckDirection[8],CheckDoorState[18]
2	CheckCurrentState[2],CheckMovementRequest[13],CheckDirection[35],CheckDoorState[18]
3	CheckCurrentState[3],CheckMovementRequest[13],CheckDirection[8],CheckDoorState[18]
4	CheckCurrentState[1],CheckMovementRequest[13],CheckDirection[34],CheckDoorState[17]
5	CheckCurrentState[2],CheckMovementRequest[13],CheckDirection[35],CheckDoorState[17]
6	CheckCurrentState[1],CheckMovementRequest[13],CheckDirection[34],CheckDoorState[18]
7	CheckCurrentState[1],CheckMovementRequest[13],CheckDirection[34],CheckDoorState[18]
8	CheckCurrentState[2],CheckMovementRequest[13],CheckDirection[35],CheckDoorState[18]
9	CheckCurrentState[3],CheckMovementRequest[13],CheckDirection[8],CheckDoorState[18]
10	CheckCurrentState[1],CheckMovementRequest[13],CheckDirection[34],CheckDoorState[18]

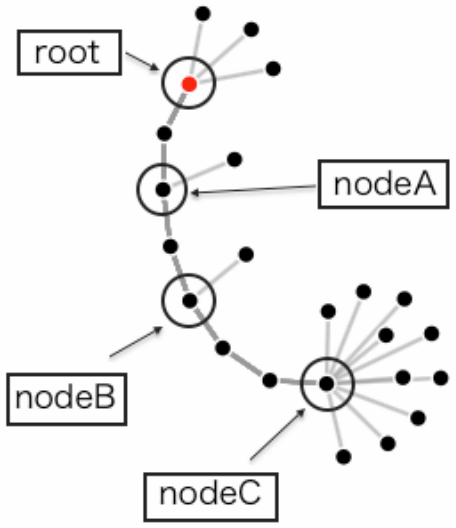

図 9 支援ツールによる状態遷移図
Fig. 9 State transition diagram by support tool

表 4 nodeA，B，C と root の各状態
Table 4 States of nodeA,B,C and root

属性値	nodeA	nodeB	nodeC	root
1 階指定ボタン	押下なし	押下あり	押下なし	押下なし
2 階指定ボタン	押下あり	押下なし	押下あり	押下なし
3 階指定ボタン	押下なし	押下なし	押下なし	押下なし
4 階指定ボタン	押下なし	押下なし	押下なし	押下なし
開ボタン	押下あり	押下なし	押下あり	押下なし
閉ボタン	押下なし	押下あり	押下なし	押下なし
ドア状態	開	開	閉	閉
タイマ状態	ウェイト	ウェイト	ウェイト	ウェイト
進行方向	上	なし	上	なし
現在階	1 階	1 階	1 階	1 階
現在状態	到着状態	到着状態	到着状態	待機状態
移動要求	あり	あり	あり	なし
ドア開閉要求	なし	閉じる	なし	なし
目的階層	なし	1 階	なし	なし

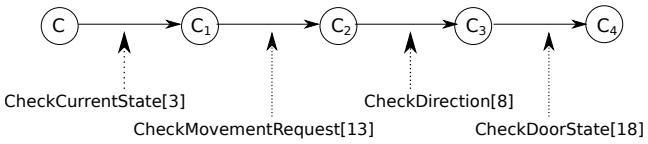

図 10 nodeC に対し表 5 のルール 1 を使用する遷移
Fig. 10 Transition of nodeC by adopting rule-1 in fig.5

場合の手順を示す．**図 10** のようにルールを使用したときの属性値の変化を**表 6** に表す．C に CheckCurrent[3] を使用すると，属性値は表 6 の C_1 になる．次に C_1 に対して CheckMovementRewuest[13] を使用すると，属性値は表 6 の C_2 になる．次に C_2 に対して CheckDirection[8] を使用すると，属性値は表 6 の C_3 になる．この状態は性質違反の状態だが，今回は assert を用いた検証を行っているので，この段階では反例は出力されない．次に C_3 に対して CheckDoorState[18] を使用すると，属性値は表 6 の C_4 になる．ルールが全て使われたのでこの時反例が出力される．このようにして表 5 で現れたルールの組合せを確認することで，CheckDirection[8]，[34]，[35] が問題のあるルールだとわかる．

4.5 考察

今回の例は反例から RASYS 記述のルールが取得できるということがわかった．また，反例で共通の状態を取得できるということがわかった．今回の適用事例では，本ツールによって原因となるルールを特定することはできたが，支援の効果については，適用事例が少なく現段階では評価することができない．今後，本論文でのモデルよりも大きいモデルに適用する，また RASYS の知識を持った人間に利用してもらうことで効果を評価する必要がある．

5. まとめ

今回の適用実験では assert に影響するルールが 1 つしか

表 6 図 10 における属性値の遷移
Table 6 changes of attribute values in fig.10

属性値	nodeC	C_1	C_2	C_3	C_4
1 階指定ボタン	押下なし	押下なし	押下なし	押下なし	押下なし
2 階指定ボタン	押下あり	押下あり	押下あり	押下あり	押下あり
3 階指定ボタン	押下なし	押下なし	押下なし	押下なし	押下なし
4 階指定ボタン	押下なし	押下なし	押下なし	押下なし	押下なし
開ボタン	押下あり	押下あり	押下あり	押下あり	押下あり
閉ボタン	押下なし	押下なし	押下なし	押下なし	押下なし
ドア状態	閉	閉	閉	閉	開
タイマ状態	ウェイト	ウェイト	ウェイト	ウェイト	ウェイト
進行方向	上	上	上	下	下
現在階	1 階	1 階	1 階	1 階	1 階
現在状態	到着状態	到着状態	到着状態	到着状態	到着状態
移動要求	あり	あり	なし	なし	なし
ドア開閉要求	なし	なし	なし	なし	なし
目的階層	なし	なし	なし	なし	なし

なかったため，反例の原因が見つかりやすかった．しかし，複数のルールが関連しあっている時はどちらが原因かわからないだけで はなく，それらの適用順序によって結果が変わってくる可能性がある．よって，反例とルールの関係だけではなく，ルール同士の関係も考慮する必要がある．本研究では，情報制御システムのモデル検査において，反例が出力された際の trail ファイルを活用することにより，その反例分析を支援することができた． 本ツールを使わずに反例分析を行おうとすると Promela の知識が必要になる．状態遷移図から RASYS 記述の誤りを分析するというアプローチをすることで，Promela のような専門的な知識を持たない開発者でもモデルの修正作業を行うことができる．

謝辞
本研究を進めるにあたり，適切な助言をくださいました株式会社日立製作所　武澤隆之氏，山形知行氏に感謝致します．本研究は JSPS 科研費 25330075 の助成を受けたものである．

参考文献

[1] 荻谷昌己 監修:SPIN による設計モデル検証, 近代科学社 (2008).
[2] Mordechai Ben-Ari 著 中島 震監訳 : SPIN モデル検査入門, オーム社 (2010).
[3] 柳翔太，小飼 敬，上田 賀一，大久保 訓，高橋 勇喜，中野 利彦：情報制御システム記述モデルの 検証項目記述と SPIN による確認, 情報処理学会研究報告 Vol.2011-SE-171 No.10(2011 年 3 月)
[4] 検証項目を持つ情報制御システム記述言語のための分析・設計手法:小飼 敬, 柳 翔太, 上田 賀一, 大久保 訓, 高橋 勇喜, 中野 利彦, 日本ソフトウェア科学会 第 17 回 ソフトウェア工学の基礎ワークショップ (FOSE 2010), pp.101-106 (2010 年 11 月)
[5] 柳翔太，小飼 敬，上田 賀一，大久保 訓，高橋 勇喜，中野利彦：情報制御システム記述モデルの検証用記述への変換と効率的検証，日本ソフトウェア科学界第 27 回大会，D-2(2010 年 9 月)
[6] 小飼 敬, 柳 翔太, 上田 賀一, 大久保 訓, 高橋 勇喜, 中野利彦:検証項目を持つ情報制御システム記述言語のための分析・設計手法, 日本ソフトウェア科学会 第 17 回 ソフトウェア工学の基礎ワークショップ (FOSE 2010) (2010 年 11 月)
[7] SPIN Language Refarence:http://www.spinroot.com/spin/Man/pron