偽反例の出力を抑制するモデル検査支援環境

佐藤直人[†1] 芹沢一[†1] 前田浩光[†2] 高橋伸明[†2] 前川義之[†2] 四野見秀明[†2]
村尾直哉[†2] 大島豊[†2] 田代敦[†2] 黒川勇[†2] 八木沢育哉[†2] 高田豊[†2] 清永章彦[†2]
大橋正己[†2]

概要：システムの仕様不具合を網羅的に検査する手段として，モデル検査がある．モデル検査とは，仕様を有限状態機械としてモデル記述し，その状態空間をモデル検査器と呼ばれるツールで機械的に網羅検査する技術である．検査により仕様不具合が検出された場合，モデル検査器はその不具合に至るまでの実行系列を反例として出力する．上流段階の設計では，仕様の簡略化のため，システムの振る舞いを抽象化して表現する場合がある．その場合，実際のシステムの振る舞いと一致しない反例が出力される可能性がある．これは偽反例と呼ばれ，不具合解析の障害になっている．偽反例を除去するためには，仕様が実システムの振る舞いと一致するように詳細化すればよい．しかし，上記のような目的で意図的に仕様を抽象化している場合は，その目的を達成できなくなる．そこで本稿では，仕様を詳細化せずに偽反例の出力を抑制する手法を提案する．さらに，提案手法に基づく作業を，形式言語を習得していない設計者でも実施可能にするモデル検査支援環境を提案する．本支援環境は，偽反例を入力として受け付け，その偽反例が出力されないように形式言語モデルを自動変更する．また本稿では，提案手法の実現性を確認するために行った適用実験結果も示す．

キーワード：モデル検査，偽反例，検証支援環境

Model Checking Assistance Environment to Prevent Spurious Counterexamples

NAOTO SATO[†1] KAZUYOSHI SERIZAWA[†1] HIROMITSU MAEDA[†2]
NOBUAKI TAKAHASHI[†2] YOSHIYUKI MAEKAWA[†2] HIDEAKI SHINOMI[†2]
NAOYA MURAO[†2] YUTAKA OHSHIMA[†2] ATSUSHI TASHIRO[†2]
ISAMU KUROKAWA[†2] IKUYA YAGISAWA[†2] YUTAKA TAKATA[†2]
AKIHIKO KIYONAGA[†2] MASAMI OHASHI[†2]

1. はじめに

産業機器などに搭載される組込みシステムの設計では，図や表形式でその状態遷移仕様を定義する場合がある．従来，状態遷移仕様の誤りは，設計レビューにて検出，修正してきた．しかし近年の組込み機器の多機能化に伴い，状態遷移仕様は膨大かつ複雑になってきている．そのため，設計レビューにて不具合を網羅的に検出，修正するのは困難になりつつある．

上記問題を解決する手段として，モデル検査がある．モデル検査とは，検査対象の仕様を有限状態機械としてモデル化し，その状態空間を機械的に網羅探索する技術である．モデル検査技術による状態遷移仕様の検証では，対象の仕様とその仕様が満たすべき要件を，形式言語で記述する．本稿では，形式言語による仕様記述結果を形式言語モデル，あるいは単純にモデルと呼ぶ．記述した形式言語モデルをモデル検査器と呼ばれるツールに入力すると，モデル検査器は網羅検査を実行し，仕様が要件を違反するかを判定した結果を出力する．さらに仕様が要件を違反する場合は，上記判定結果に加えて，要件を違反する状態に至るまでの実行系列を出力する．この実行系列は反例と呼ばれる．設計者は，反例を解読することで仕様の誤りを検知し，仕様を任意に修正する．上記形式言語の一つに Promela がある．そしてそのモデル検査器として，SPIN[1]が知られている．

ところで，状態遷移仕様の設計では，実際のシステムの振る舞いを抽象的に表現する場合がある．例えば，基本設計や機能設計などと呼ばれる上流段階の設計では，状態遷移仕様に表れる状態数を抑える目的で，いくつかの状態をまとめて一つの状態に集約する場合がある．そのように抽象化した状態遷移仕様は，実際のシステムの振る舞いと完全には一致しない．そして上記抽象化した状態遷移仕様をモデル検査器で検証すると，実際のシステムの振る舞いとは異なる実行系列が，反例として出力される可能性がある．このような反例を偽反例と呼ぶ．

モデル検査器は，反例を一つ検出した時点で網羅探索を中止する．つまり形式言語モデルが偽反例を含む場合は，モデル検査器は状態空間の一部を探索せずに検査を終了してしまう．よって状態遷移仕様の網羅検査のためには，偽反例の出力を抑制する必要がある．以降，偽反例ではない反例を正反例と呼ぶ．そして，偽反例および正反例を合わ

†1 (株)日立製作所　研究開発グループ
Hitachi, Ltd., Research & Development Group
†2 (株)日立製作所　ICT 事業統括本部
Hitachi, Ltd., Information & Communication Technology Business Division

せて反例と総称する．

上述のとおり，偽反例の原因は，抽象化した状態遷移仕様と実際のシステムの振る舞いとの不一致にある．よって，偽反例を除去するためには，実際のシステムの振る舞いと一致するように，状態遷移仕様を詳細化すればよい．しかし，上記のように状態数を抑える等の目的で意図的に状態遷移仕様を抽象化している場合は，その目的を達成できなくなるという問題がある．

上記問題に対して本稿では，状態遷移仕様を詳細化せずに偽反例の出力を抑制する手法を提案する．さらに，提案手法に基づく作業を，形式言語を習得していない設計者でも実施可能にするモデル検査支援環境を提案する．本支援環境は，偽反例を入力として受け付け，その偽反例が出力されないように形式言語モデルを自動変更する．本稿の貢献を以下にまとめる．

- 状態遷移仕様を詳細化せずに，偽反例の出力を抑制する手法を提案する
- 提案手法に基づく作業を，形式言語を未習得の設計者でも実施可能にする支援環境を提案する
- 提案手法の実現性を示す実験結果を提供する

本稿の構成は次のとおりである．まず 2 章で本研究の前提となる技術およびツールについて説明する．次に 3 章にて本稿で扱う課題を述べ，4 章ではその解決方法として偽反例の出力を抑制する手法を提案する．さらに 5 章では，4 章で提案する手法の支援環境を提案する．6 章では提案手法とその支援方式の評価実験の内容を示し，7 章で結果を評価する．8 章では関連研究を示し，9 章で結論をまとめる．

2. 前提技術・ツール

2.1 モデル検査

モデル検査とは，検査対象の仕様を有限状態機械としてモデル化し，その状態空間を機械的に網羅探索する技術である．モデル検査の概念図を図 1 に示す．

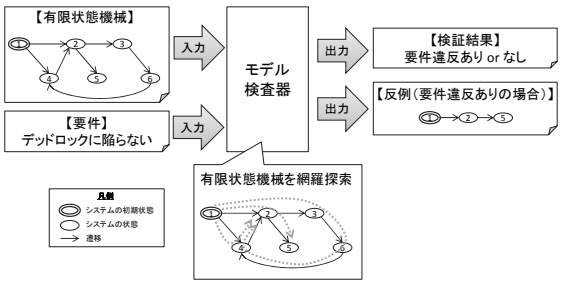

図 1　モデル検査の概念図

Figure 1　Concept of Model Checking

モデル検査では，対象システムの仕様を表す有限状態機械と，その仕様が満たすべき性質を表す要件が入力となる．上記有限状態機械および要件の記述に用いる形式言語は，使用するモデル検査器に依存する．モデル検査器は受け付けた有限状態機械を網羅探索し，要件を満たさない実行系列が存在するかを検査する．そして要件を満たさない実行系列を検出した場合には，「要件違反あり」を意味する検証結果と，検出した上記実行系列を反例として出力する．本研究では，モデル検査器 SPIN と，その対応する形式言語 Promela を使用する[1]．以降，Promela にて記述した形式言語モデルを，Promela モデルと呼ぶ．

2.2 状態遷移仕様の設計支援環境

状態遷移仕様設計においてモデル検査を適用するためには，設計者は状態遷移仕様とその要件を，形式言語で記述する必要がある．しかし，上記形式言語で記述した仕様，即ち形式言語モデルは，形式言語を習得していない他の設計者や顧客には解読できない．よって上記形式言語モデルとは別に，自然言語で記述した仕様書も従来どおり作成する必要がある．つまり，モデル検査を適用する場合，形式言語モデルの記述作業が追加的に発生する．また，そもそもモデル検査を担当する設計者が形式言語を未習得の場合は，その習得にかかるコストも上乗せされる．

この問題に対して本研究では，状態遷移設計におけるモデル検査の適用を効率化することを主目的に，以下の機能を備える状態遷移設計支援環境を開発している．

【状態遷移仕様設計支援環境の主機能】
(1) 状態遷移仕様の記述支援機能
(2) モデル検査支援機能

以降の節では，これらの機能の概略を示す．

2.2.1 状態遷移仕様の記述支援機能

本支援環境では，システムの状態を構成する状態変数ごとにその遷移仕様を記述する方法を採用している．本支援環境における状態遷移仕様の記述アプローチを図 2 に示す．

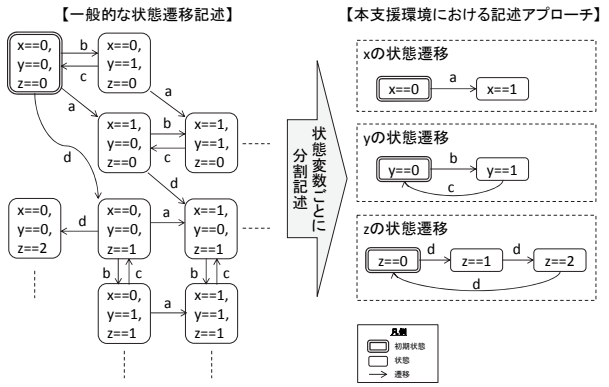

図 2　状態遷移仕様の記述アプローチ

Figure 2　Our Approach to Describe State Transition Spec

例えば，対象のシステムの状態が状態変数 x，y および z で構成される場合，一般的な状態遷移記述では，それらの状態変数の組合せ状態に対して遷移仕様を記述する．しかし，仕様が複雑で多くの状態変数が存在する場合，上記方法では記述が煩雑化するという欠点がある．そこで本支援環境では，状態変数ごとに遷移仕様を分割記述するアプロ

ーチを採用している．これにより，状態遷移仕様の記述量を削減できるという効果が見込める．状態変数ごとの遷移仕様は，表 1 に示す形式で記述する．

表 1　状態遷移仕様の表記述形式

Table 1　Table Form of State Transition Specification

#	状態変数	遷移前状態値	遷移条件			遷移後状態値	アクション	
			イベント	他状態	グローバル変数		イベント出力	グローバル変数更新
1	x	0	ev_a	y==0	var1==true	1	ev_b	−
2	y	0	ev_b	x==1	var1==true	1	ev_c	var1=false
3	y	1	ev_c	*	*	0	ev_d	−
4	z	0	ev_d	y==1	*	1	−	−
5	z	1	ev_d	*	*	2	−	−
6	z	2	ev_d	*	var1==false	0	ev_a	var1=true

「状態変数」の列には，x などの状態変数を記述する．「遷移前状態値」および「遷移後状態値」の列には，それぞれ遷移前と後の状態変数の値を記述する．例えば#1 の遷移は，状態変数 x の値が 0 から 1 に遷移する場合の遷移仕様を表している．「イベント」の列には遷移発生の条件となるイベント事象を記述する．例えば#1 の遷移では，イベント ev_a の発生が遷移の条件となる．その他にも，「他状態」の列には x 以外の他の状態変数に関する遷移条件を記述できる．同様に「グローバル変数」の列には，状態とは別に定義可能なグローバル変数に関わる遷移条件を記述できる．「イベント出力」の列には，当該遷移が発生した場合に，伴って発生するイベント事象を記述する．同様に「グローバル変数更新」の列には，当該遷移が発生した場合に実行するグローバル変数への代入手続きを記述する．

表 1 に示した遷移仕様の記述結果は，以降，遷移決定表と呼ぶ．この遷移決定表の記述を支援するため，本支援環境では以下の機能を提供する．

【遷移決定表の記述支援機能】
- プルダウンによる選択式の入力支援
- 文法チェック
- 状態遷移図の表示

状態変数や遷移前／遷移後状態値およびイベントなどの名称は，用語辞書にて一元管理する．この用語辞書を利用することで，遷移決定表の一部をプルダウンによる選択式で記入できる．また，上記イベントなどの名称の誤りや記入の漏れなどは，文法チェック機能にて検出できる．さらに，遷移決定表で定義した遷移仕様の俯瞰的理解を促進するため，図 2 に示したような図形式に変換して表示する機能を備える．

2.2.2 モデル検査支援機能

本支援環境は，記述した遷移決定表を Promela に変換することで，SPIN によるモデル検査の実施を支援する．モデル検査支援のために本支援環境が提供する機能を図 3 に示す．

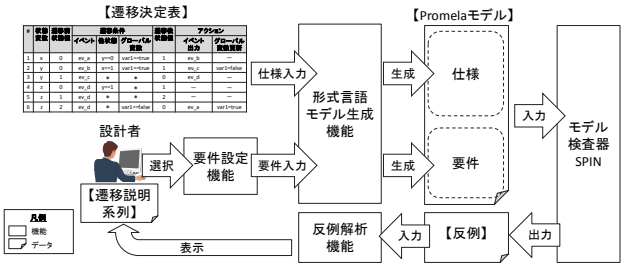

図 3　モデル検査支援機能の構成

Figure 3　Components of Model Checking Assistance Feature

本支援環境は，要件設定機能，形式言語モデル生成機能，，反例解析機能，およびモデル検査器 SPIN から構成される．要件設定機能は，検証する要件のリストを予め保持している．例えば，「デッドロックに陥らない」などの要件を保持している．設計者は，上記リストの中から検証したい要件を選択する．要件設定機能は，選択された要件を形式言語モデル生成機能に受け渡す．形式言語モデル生成機能は，記述した遷移決定表と，上記要件を入力として受け付け，Promela モデルを生成する．遷移決定表の各行に記述した遷移仕様は，一定の規則に従いそれぞれ Promela のコードブロック（以降，遷移コードブロックと呼ぶ）に変換する．また遷移コードブロックの直後には，標準出力を行うコード（以降，説明出力コードと呼ぶ）を挿入する．説明出力コードは，当該遷移コードブロックに対応する遷移仕様を，設計者向けに説明する文字列（以降，遷移説明文と呼ぶ）を標準出力する．例えば表 1 に示した遷移決定表の場合，図 4 に示す Promela モデルを生成する．

```
active proctype transition() {
  do
  /*遷移#1*/                                          遷移#1の
  :: x==0 && ev_a==true && y==0 && var1==true ->      遷移コードブロック
     x=1; ev_b=true; ev_a=false;
     printf("[遷移説明] 遷移#1, 遷移前:x==0, イベント:ev_a,
             条件:y==0, var1==true,  遷移後: x=1, ev_b=true");

  /*遷移#2*/                                          遷移#1の
  :: y==0 && ev_b==true && x==1 && var1==true ->      説明出力コード
     y=1; ev_c=true; var1=true; ev_b=false;
     printf("[遷移説明] 遷移#2, 遷移前:y==0, イベント:ev_b,
             条件:x==1, var1==true, 遷移後:y=1, ev_c=true, var1=false");

  /*遷移#3～#6は省略*/
  od
}                                                     要件コード
ltl prop { [] (<> (x==0 && y==0 && z==0)) }
```

図 4　生成される Promela モデルの例

Figure 4　An Example of Generated Promela Model

図 4 では，変数の型を宣言するコードは省略している．図 4 の点線部は，表 1 の遷移#1 に対応する遷移コードブロックを示している．そして，この遷移コードブロックに続く printf 文（下線部）が，遷移#1 の説明出力コードに相当する．printf 文は，括弧内の文字列を標準出力する．説明出力コードが出力する遷移説明文は，後に示す反例解析機能が使用する．最下部の ltl で始まる文は，要件を表すコードである（以降，要件コードと呼ぶ）．prop は，要件の名称を表しており，それに続く{}節の中に要件を表す式を記

述している．

SPIN の出力する反例は要件違反に至るまでの実行系列であり，具体的には，遷移コードブロックの実行ログの系列である．そして上記遷移コードブロックが実行される場合，必ずその後に続く説明出力コードも実行される．よって反例には，遷移コードブロックの実行ログと，説明出力コードが出力する遷移説明文が連結して出力される．上記遷移説明文は対応するコードブロックの遷移仕様を説明する文であるため，遷移説明文の系列（以降，遷移説明系列と呼ぶ）は反例を表す．そこで反例解析機能は，解読が困難な遷移コードブロックの実行ログは除外し，遷移説明文のみを抽出して設計者に表示する．そのため，検索のキーワードとなる固有の接頭辞を遷移説明文に付与しておく．図 4 の例では接頭辞として[遷移説明]を使用している．設計者は，抽出された遷移説明文を参照することで，容易に反例を解読できるようになる．

このように，状態遷移仕様の設計支援環境を適用することで，状態遷移仕様の設計および検証を効率化できる可能性がある．また本支援環境を適用することで，設計者は Promela モデルやその実行ログを直接編集，参照せずに済むため，Promela を未習得の設計者でも SPIN による検証とその反例解読を行えるようになる．

3. 課題

2 章に示した設計支援環境を適用した場合でも，モデル検査の適用には課題がある．状態遷移仕様の設計では，実際のシステムの振る舞いを抽象的に表現する場合がある．例えば，基本設計や機能設計などと呼ばれる上流段階の設計では，状態遷移仕様に表れる状態数を抑える目的で，いくつかの状態をまとめて一つの状態に集約する場合がある．または仕様を簡略化し俯瞰的な理解を促す目的で，一部のデータや状態，遷移の実行条件などを省略する場合もある．

上記のように抽象的に記述した状態遷移仕様は，実際のシステムの振る舞いと完全には一致しない．よって，上記抽象的な状態遷移仕様から作成した形式言語モデルも，実際の振る舞いと一致しない．その場合，モデル検査器は，実際のシステムが実行することのない実行系列，即ち偽反例を出力する可能性がある．

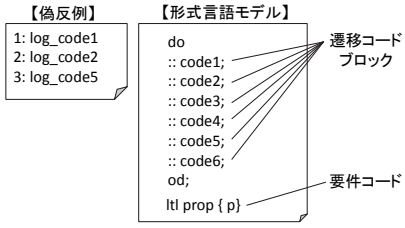

図 5　抽象的な仕様を表す形式言語モデルと偽反例の例
Figure 5　An Example of Abstract Formal Model and its Counterexample

ここで例として，遷移#1～#6 からなる遷移決定表から生成した形式言語モデルと，その偽反例の例を与える．図 5 に示す形式言語モデルでは，遷移#1～#6 に対応する遷移コードブロック code1～code6 の何れかを，非決定的にループ実行する，実システムにおいては，code2 と code3 の実行によってメモリ上のテーブルが更新され，その更新結果に基づいて実行可能な遷移コードブロックが制御される．しかしこの例では仕様の簡略化のため，上記テーブルに関わる振る舞いを省略している．この抽象化された状態遷移仕様を要件 p についてモデル検査した結果，同じく図 5 に示す「log_code1，log_code2，log_code5」という反例が検出されたとする．log_code1，log_code2 および log_code5 は，それぞれ code1，code2 および code5 の実行ログを表す．しかし実システムにおいては，上記テーブルに関わる制御によって，code2 が上記テーブルを更新した後は，code5 は実行不可になるとする．その場合，上記反例は偽反例となる．

モデル検査器は，反例を一つ検出した時点で網羅探索を中止する．つまりモデル検査器が偽反例を検出した場合は，残りの状態空間は未検査のまま終了するため，正反例の有無を判定できない．よって仕様を網羅検査するためには，偽反例の出力を抑制する必要がある．またモデル検査器によっては，反例を検出しても探索を継続するように設定できる場合もある．その場合モデル検査器は，探索によって検出した全ての反例を出力する．その反例の集合には，同一の抽象化に起因する複数の偽反例が含まれることがある．よって，仕様が要件を満たすかあるいは違反するかを結論付けるためには，出力された反例の中に正反例が含まれるかを確認する必要がある．そのため設計者は，出力された反例を一つ一つ解読し，偽反例か正反例かを判定することになる．上記判定作業のコストは，偽反例の数に依存する．よってこの方法を採用する場合でも，作業コスト削減のため，偽反例の出力を抑制できることが望ましい．

上述のとおり，偽反例の原因は，記述した状態遷移仕様と実際のシステムの振る舞いとの不一致である．よって偽反例を除去するためには，実際のシステムの振る舞いと一致するように，状態遷移仕様を詳細化すればよい．しかし，本章の冒頭に示したような目的で意図的に状態遷移仕様を抽象化している場合は，仕様を詳細化することでその目的を達成できなくなるという問題がある．

例えば図 5 の偽反例を除去するためには，上記テーブルの振る舞いを状態遷移仕様に導入し，code2 の実行後は code5 を実行不可にすればよい．しかしそのように仕様を詳細化することで，上記テーブルに関わる振る舞いを省略するという当初の目的を達成できなくなる．さらに，詳細化の結果が別の偽反例の原因となることで詳細化のスパイラルが発生し，最終的にソースコードと同等の詳細度で仕様を記述することになる可能性もある．以上より，状態遷移仕様を詳細化せずに偽反例の出力を抑制することが課題となる．

4. 偽反例の出力抑制手法

4.1 手法

3 章に示した課題を解決するため，状態遷移仕様を詳細化せずに，形式言語モデルの一部を変更することで偽反例の出力を抑制する手法を提案する．形式言語モデルの変更アプローチを図 6 に示す．

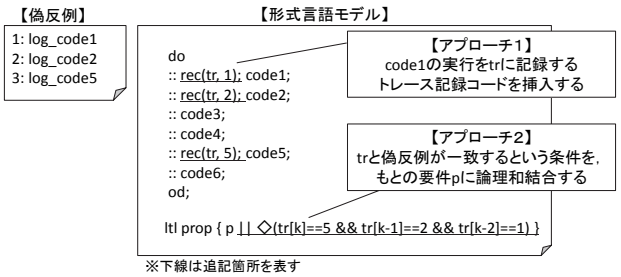

図 6　形式言語モデルの変更アプローチ

Figure 6　Approach to Change a Formal Model

提案手法では，偽反例に基づいて形式言語モデルを変更する．まずモデル検査器が出力する偽反例を$S = \{l_1, l_2, \dots, l_n\}$と定義する．$l_i\ (1 \le i \le n)$は実行ログを表す．設計者は，偽反例$S$を編集することで，以下の汎化偽反例$S'$を作成してもよい．

$$S' = \{l'_1, l'_2, \dots, l'_n\}\ such\ that\ \forall i \exists j \cdot l'_i = l_j$$

汎化偽反例S'を作成する狙いは 4.2 節に示す．以降の手続きは，偽反例Sあるいは汎化偽反例S'のいずれかを入力に行う．本稿では偽反例Sを入力とした場合について説明する．汎化偽反例S'を入力にする場合でも，Sと同様の手続きを実施する．

まず，偽反例Sを構成する実行ログ$l_1, l_2, \dots, l_n$に対応する遷移コードブロックを特定する．ここでは，$c_1, c_2, \dots, c_n$とする．そして，遷移コードブロック$c_i\ (1 \le i \le n)$の直前に，当該コードブロックの実行を記録するためのコード$rec(tr, i)\ (1 \le i \le n)$を挿入する．つまり，$c_i$を以下のとおり更新する．

$$c_i := rec(tr, i);\ c_i$$

記号";"は逐次結合を表す．trはトレース情報を蓄積する配列型の変数で，トレース変数と呼ぶ．トレース変数trにおいて，トレース情報を格納済みの末尾番地を$k\ (-1 \le k)$とする．$k = -1$は未格納を意味する．$rec(tr, i)$はtrの$k+1$番地に，遷移コードブロックc_iの識別子$id(i)$を記録する．

$$rec(tr, i) \stackrel{\text{def}}{=} tr[k+1] := id(i)$$

このrecをトレース記録コードと呼ぶ．idはiを引数にとり，c_iの識別子を返す関数である．recの実行とともに，trの末尾番地kは$k := k + 1$と更新される．よって，$s\ (0 \le s \le k)$回前に記録したtrの値は，$tr[k-s]$にて参照できる．図 6 の例では，偽反例のログ log_code1 に対応する遷移コードブロック code1 に対して，トレース記録コード rec(tr, 1)を挿入している．

次に要件コードを変更する．提案手法では，「トレース変数に記録したトレース情報の示す実行系列が，偽反例に含まれる」という条件を，もとの要件に論理和結合する．要件をpとすると，この変更は以下のとおり表せる．

$$p := p \vee \Diamond\left(\bigwedge\nolimits_{0 \le x \le n-1} tr[k-x] = id(n-x)\right)$$

"◇"は将来のある時点を表す時相演算子であり，$\Diamond a$は，実行系列のいずれかの時点で論理式aが成立することを意味する．図 6 の例では，もとの要件を表す式 p に対して，上記条件を表す式◇(tr[k]==5 && tr[k-1]==2 && tr[k-2]==1)を論理和結合している．トレース記録コードは対応する遷移コードブロックの直前に挿入しているため，遷移コードブロックの実行により要件 p の違反が発生する時には，既に上記条件は成立している．このように要件コードを変更することで，偽反例の実行系列では，p の成立に関わらず要件を満たすと判定されるようになる．これにより，当該偽反例は検証対象から除外され，以降，反例として出力されなくなる．

4.2 汎化偽反例の狙い

提案手法では，モデル検査器が出力する偽反例に着目し，その偽反例が再度出力されないように形式言語モデルを変更する．しかし 3 章でも触れたが，ある一箇所の仕様抽象化に起因して，類似の偽反例が複数通り出力される場合がある．その場合に提案手法を適用すると，同一の原因によって発生する N 通りの偽反例の対処のために，N 回のモデル変更を行うことになる．このモデル変更回数 N は，抑制対象の偽反例Sを，汎化偽反例S'に変更することで削減できる可能性がある．

例えば，モデル検査器が偽反例$S1 = \{log1, log2, log5\}$を出力した場合を考える．そして上記$S1$と同一の原因により，$S2 = \{log2, log4, log5\}$および$S3 = \{log2, log3, log5, log4\}$という偽反例も存在すると仮定する．これらの偽反例$S1$，$S2$および$S3$の共通点は，$log2$の後に$log5$を実行している点であり，そのような条件を満たす実行系列は全て偽反例であるとする．この場合，設計者は汎化偽反例$S' = \{log2, log5\}$を作成し，S'に対して 4.1 節に示した手続きを実行することで，偽反例$S1$，$S2$および$S3$の出力を一度に抑制できるようになる．このように，偽反例からその本質的な原因を表す実行系列を抽出することで，モデル変更の回数を削減できる．

ただし，偽反例の共通点を見誤って汎化偽反例を作成すると，偽反例だけでなく正反例の出力まで抑制してしまう可能性がある．よって汎化偽反例を作成する際には，その汎化偽反例の表す反例集合が，実際の偽反例の部分集合になるように慎重に検討する必要がある．

5. 偽反例出力抑制の支援環境

2 章に示した支援環を前提に，4 章に示した提案手法の

実施を支援する方式を提案する．図 3 に示したモデル検査支援機能の構成図に，提案手法支援のための追加機能を加えた結果を図 7 に示す．

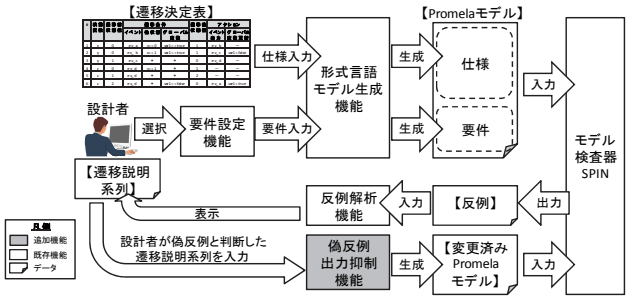

図 7　提案手法支援のための追加機能

Figure 7　Additional Function for Our Proposed Method

図 3 と比較して，図 7 では偽反例出力抑制機能を追加している．設計者は，本支援環境から取得した遷移説明系列が偽反例であると判断した場合，取得した遷移説明系列，あるいはその編集結果である汎化遷移説明系列を偽反例出力抑制機能に入力する．汎化遷移説明系列は，4.2 節に示した汎化偽反例に対応する遷移説明系列である．

偽反例出力抑制機能は，遷移説明系列あるいは汎化遷移説明系列を入力として受け付け，その系列を構成する遷移説明文ごとに，対応する遷移コードブロックを特定する．そして，特定したコードブロックの直前に，4 章に示したトレース記録コードを挿入する．そして，上記トレース記録コードが使用したトレース変数を用いて，要件コードを変更する．図 4 に示した Promela モデルの場合，例えば図 8 のように変更する．

```
active proctype transition() {
  int tr[100];                          トレース変数
  int k = -1;                           トレース変数の配列
                                        インデックス
  do
  /*遷移#1*/
  ::  if :: k<100 -> k++; tr[k]=1;      トレース記録コード
      :: else -> skip;
      fi;
      x==0 && ev_a==true && y==0 && var1==true ->
      x=1; ev_b=true; ev_a=false;
      printf("[遷移説明] 遷移#1, 遷移前:x==0, イベント:ev_a,
              条件:y==0, var1==true,  遷移後: x=1, ev_b=true");

  /*遷移#2*/
  ::  if :: k<100 -> k++; tr[k]=2;      トレース記録コード
      :: else -> skip;
      fi;
      y==0 && ev_b==true && x==1 && var1==true ->
      y=1; ev_c=true; var1=true; ev_b=false;
      printf("[遷移説明] 遷移#2, 遷移前:y==0, イベント:ev_b,
              条件:x==1, var1==true, 遷移後:y=1, ev_c=true, var1=false");

  /*遷移#5に対しても同様にトレース記録コードを挿入する*/
  od
}

ltl prop { ([] (<> (x==0 && y==0 && z==0)))   要件コード
          || (<> (k>=2 && tr[k]==5 && tr[k-1]==2 && tr[k-2]==1)) }
```

※下線は追記箇所を表す

図 8　変更後の Promela モデルの例

Figure 8　An Example of Updated Promela Model

図 8 に示す Promela モデルの冒頭では，トレース変数 tr と，その配列インデックス k を宣言している．遷移#1 の遷移コードブロックの直前には，トレース記録コードとして，配列インデックス k をインクリメントする文 k++と，代入文 tr[k]=1 を挿入している．トレース変数 tr のサイズを 100 に設定しているため，k<100 の場合のみトレース記録コードを実行する．この方式の問題点については，7 章で考察する．tr[k]に代入している定数 1 は遷移#1 を表す．同様に，遷移#2 および遷移#5 に対しても，トレース記録コードを挿入する．さらに要件コードでは，論理式<> (k>=2 && tr[k]==5 && tr[k-1]==2 && tr[k-2]==1)をもとの要件に論理和結合している．tr[k]==5 && tr[k-1]==2 && tr[k-2]==1 は，遷移#1，遷移#2，遷移#5 の順番で遷移が発生することを表す．k>=2 は少なくとも 3 回以上，tr にトレース情報を格納したことを意味する．"<>"は，時相演算子◇を表す．このように Promela モデルを変更することで，遷移#1，遷移#2，遷移#5 の順番で遷移を実行した場合*は，要件を満たすと判定されるようになる．

以上のとおり支援環境を構築することで，偽反例の出力を抑制するためのモデル変更作業を自動化できる．これにより設計者は，Promela モデルおよびその実行ログを直接編集，参照せずに済むため，Promela を未習得の設計者でも，提案手法による偽反例の抑制を実現できる．

6. 実験

4 章に示した偽反例の出力抑制手法を，5 章に示した支援方式に則って適用することにより，偽反例の出力を抑制できることを確認する．そのため，実製品の状態遷移仕様を対象とする実験を行った．ただし，提案する支援環境は未実装のため，支援環境が提供する予定の機能は手作業でシミュレーションした．題材に使用した状態遷移仕様は，8 つの状態変数と 134 の遷移から構成される．本実験にて作成した Promela モデルの説明出力コード，トレース記録コード，および要件コードの一部を，図 9 に例示する．

```
【説明出力コード】
 printf("----¥n TRACE #%d :外部イベント発生#2：イベントAが発生¥n", trCounter); trCounter++;

【トレース記録コード】
 index++; tr[index] = BL;

【要件コード】
 ltl prop { (
 ( [] ( <> ( p == 0 )))
 || ( <> ( index>=2 && tr[index] == BL && tr[index-1] == CR && tr[index-2] == BL ) )
 || ( <> ( index>=2 && tr[index] == BL && tr[index-1] == BL && tr[index-2] == CR ) )
 || ( <> ( index>=2 && tr[index] == BL && tr[index-1] == CR && tr[index-2] == CR ) )
 )}
```

図 9　Promela モデルの一部

Figure 9　A Part of the Promela Model

本実験では，5 章に示した支援方式を適用する上で，設計者が実施する作業と，支援環境の機能をシミュレーションする作業を明確に区別して行った．特に後者の作業は機械的に実現可能な手続きになるよう定義した．

まず設計者の作業として，題材の状態遷移仕様を遷移決定表に記述した．次に支援環境が実現する処理として，上記仕様記述から 2916 ステップの Promela モデルを作成した．その際，図 9 に示すような説明出力コードを挿入した．図

* 遷移#1，遷移#2，遷移#5 の間に，それ以外の遷移が入り込んでも良い．

9 の説明出力コードに現れる変数 trCounter は，遷移説明文に通番を付与する目的で導入している．そして，支援環境が実現する処理として上記 Promela モデルを SPIN に入力した結果，約 2 千万ステップからなる反例が出力された．そこで出力された反例に対して，支援環境が実現する処理として, 先頭から 10 万ステップまでを対象に遷移説明文を抽出した. その結果, 27363 行からなる遷移説明系列を得た．設計者の作業として，この遷移説明系列を先頭から 947 行解読したところ，当該反例では BL および CR という 2 つの遷移が，実際には起こり得ない{BL，CR，BL}という順序で実行されていることが判明した．よって，汎化遷移説明系列として{BL，CR，BL}を作成した．上記汎化遷移説明系列を受け，支援環境の処理として，BL および CR 対してトレース記録コードを挿入した．さらにもとの要件コード([] (<> (p == 0)))に対して，<> (index>=2 && tr[index] == BL && tr[index-1] == CR && tr[index-2] == BL)を論理和結合した. 図 9 には, BL に挿入したトレース記録コード，および変更後の要件コードを例示している．

上記のとおり変更した Promela モデルを再度 SPIN で検証した結果，また別の反例が出力されたため，同様に解読を行った．その結果，上記 BL および CR が，{CR，BL，BL}の順序で実行されていることが判明した．この実行順序も実際には起こり得ないため，同様の論理式を要件コードに論理和結合した. その後もう一度同様の手続きを経て，最終的に図 9 に示す要件コードを作成した. そしてこの要件コードにて再検証を実施した結果，要件違反なしとの結果を得た．

7. 評価

6 章に示した実験では，提案手法に従うことで偽反例の出力を抑制できた．その結果，仕様を詳細化することなく検査を完了できた．これにより，提案手法を適用することで課題を解決できること確認した．3 章で述べたように意図的に状態遷移仕様を抽象化している場合，提案手法によって仕様を詳細化せずにモデル検査を実施できるため，有用と考える．また意図的な抽象化でない場合でも，提案手法を適用することで仕様の詳細化にかかるコストを削減できるという利点がある．

また本実験では，設計者が実施する作業と，支援環境の機能をシミュレーションする作業を明確に区別し，特に後者の作業は機械的に実現可能な手続きになるよう定義した. その結果，上記機械的手続きにて偽反例の出力抑制を達成できたため，5 章に示した支援環境の実現見込みを得た．また設計者が実施する作業においては，Promela モデルやその実行ログなどを直接編集，参照せずに済むことを確認した．よって，5 章に示した支援環境を実現することで，形式言語 Promela を未習得の設計者でも，偽反例の出力を抑制できるようになる可能性が高い．

ただし，Promela を対象に提案手法を適用する場合，以下の問題がある．Promela では配列を定義する際に，そのサイズを有限に定義する必要がある．そのため図 8 では，トレース変数 tr のサイズを 100 に定義している．よって現在の方式では，遷移コードブロックの実行が 101 回を超える場合は，以降のトレース情報は tr に記録されない．偽反例出力抑制機能の入力として偽反例（実際には遷移説明系列）を与える場合は，その偽反例の長さよりも tr のサイズを大きく設定すれば問題ない．一方，汎化偽反例（実際には汎化遷移説明系列）を入力として与える場合は，出力を抑制したい偽反例の最大長は明らかでない．そこで，仮に tr のサイズを 100 と定義すると, 遷移コードブロックを 100 回実行した後にその汎化偽反例の示す実行系列が出現するような偽反例は抑制できなくなる．6 章に示した実験ではそのような偽反例が存在しなかったため問題にはならなかったが，今後対策が必要である．例えば，配列サイズを超える場合は，0 番地に戻って上書きすることでトレース情報の記録を続けるようにする．

本稿では対象を状態遷移仕様に限定したが，4 章に示した提案手法は状態遷移仕様の特徴を利用していない．よって, 状態遷移仕様以外の仕様に対しても応用可能と考える．また提案手法には，モデル検査器のアルゴリズムを変更することなく適用できるという利点もある．5 章に示した支援方式についても，基本的な機能構成は他の種類の仕様に応用できる．ただし，仕様の記述形式やその意味の変更に応じて，形式言語モデル生成機能を変更する必要がある．それに伴い，偽反例出力抑制機能にも変更が生じる．

また，提案手法のもとでモデル検査を適用し，要件を満たすと判定された状態遷移仕様は,「抑制した偽反例を除けば」要件を満たす仕様である. よって, 後の開発工程では，抑制した偽反例が除去されるように仕様を詳細化すればよい．つまり，偽反例の出力抑制に使用した遷移説明系列や汎化遷移説明系列は，後工程の設計において有益な情報となる．

8. 関連研究

モデル検査における偽反例の除去は従来から研究対象になっている．文献[2]の著者らは，検出された反例が正反例か偽反例かを判定するアルゴリズムと，上記反例が偽反例だった場合に，その除去のための詳細化方法を多項式時間で発見するアルゴリズムを提案している．さらに文献[8]では上記判定アルゴリズムを，並列実行可能なアルゴリズムに改良している．これらの技術では，偽反例の除去のために仕様を詳細化することが前提になっている. 本稿では，上流段階の設計において仕様を抽象的に記述する場合があることを示した上で，仕様を詳細化せずに偽反例の出力を抑制する手法を提案した．

また 2.2 節では，モデル検査の実施を支援する機能とし

て，状態遷移仕様から形式言語モデルを生成する機能と，モデル検査器が出力する反例の解析機能を提示した．同様の機能を備える支援ツールは既に提案されており，文献[11][12][13][14][15][16]に見ることができる．本稿では，これらの機能に加えて，偽反例の出力抑制機能を備えるモデル検査支援環境を提案した．

本稿で提案した手法では，複数の偽反例に共通的な実行系列として汎化偽反例を作成することで，それら複数の偽反例の出力を一度に抑制できるようにしている．上記汎化偽反例を作成するためには，偽反例の共通点，即ち原因を正しく理解する必要がある．そのための支援技術として，以下に示す反例解析技術を応用できる可能性がある．文献[7]ではプログラムを対象に，モデル検査器が出力した反例と類似する実行系列を生成する方法を示している．上記類似性は，要件違反に到達する際の遷移と，その直前の実行位置（control location）が一致するという性質によって定義される．上記反例と，その類似する実行系列との共通点や差異を分析して設計者に提示することで，反例の原因理解を支援する．また文献[3]では，反例に類似する実行系列の生成に SAT ソルバ[9][10]を用いたアプローチを導入するとともに，文献[7]とは異なる類似性の概念として，実行系列における状態変数の値に着目した *distance metric* を導入している．さらに文献[4]では，より直感的な原因理解を促すため，反例とその類似する実行系列との差異を述語抽象化[5]する技術を提案している．文献[6]では，反例とその他の実行系列とを比較することで，要件を違反する場合とそうでない場合の分岐点を特定する方法を示している．これらの技術は，偽反例の原因分析にも活用できる可能性がある．

また文献[6]では，上記分岐点から派生する経路のうち，反例となる経路を実行不可にする方法も提案している．その後再度モデル検査を実行すると，当該反例の出力が抑制されるため，別の反例が得られるようになる．このように，一部の経路を実行不可にするというアプローチは，偽反例の出力抑制にも応用できる可能性がある．

9. まとめ

本稿では，状態遷移仕様の抽象度を維持したままモデル検査を適用する場合，偽反例の出力抑制が課題になることを提起した．そして上記課題を解決する方法として，状態遷移仕様を詳細化せずに偽反例の出力を抑制する手法を提案した．提案手法では，トレース記録コードの挿入や要件コードの変更によって偽反例の出力を抑制する．さらに，提案手法に基づく作業を，形式言語を未習得の設計者でも実施可能にする支援環境を提案した．そして，実製品の仕様を題材に提案手法とその支援方式の評価実験を行い，その実現性を確認した．

最後に今後の課題を述べる．まず 5 章で提案した支援環境を実装し，その支援方式の実現性を実証する．さらに，Promela を未習得の設計者が実際に使用することで，形式言語を未習得の設計者でも提案手法を適用できることを実証する．また現在の手法では，遷移の実行系列を与えることで抑制対象の偽反例を特定しているが，今後は他の特定方法も検討する．例えば，「ある状態において特定の遷移が実行されること」を条件として与え，その条件に合致する遷移を含む実行系列を，偽反例として抑制する．

参考文献

[1] Holzmann, G.J.: SPIN Model Checker, The: Primer and Reference Manual (2003).
[2] Clarke, E., Grumberg, O., Jha, S., Lu, Y. and Veith, H.: Counterexample-guided abstraction refinement, In Computer Aided Verification, pages 154-169 (2000).
[3] Groce, A.: Error Explanation with Distance Metrics, In Tools and Algorithms for the Construction and Analysis of Systems, pages 108-122 (2004).
[4] Chaki, S., Groce, A. and Strichman, O.: Explaining Abstract Counterexamples, ACM SIGSOFT Software Engineering Notes, Volume 29 Issue 6, Pages 73-82 (2004).
[5] Graf, S. and Saidi, H.: Construction of abstract state graphs with PVS, Conference on Computer Aided Verification CAV'97, LNCS 1254, pp. 72-83, (1997).
[6] Ball, T., Naik, M. and Rajamani, S.: From symptom to cause: Localizing errors in counterexample traces, In Principles of Programming Languages, pages 97-105 (2003).
[7] Groce, A. and Visser, W.: What went wrong: Explaining counterexamples, In SPIN Workshop on Model Checking of Software, pages 121-135 (2003).
[8] Tian, C. and Duan, Z.: Detecting Spurious Counterexamples Efficiently in Abstract Model Checking, Software Engineering (ICSE), 2013 35th International Conference, pp.202-211 (2013).
[9] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L. and Malik, S.: Chaff: Engineering an Effcient SAT Solver, In Proceedings of the 38th Design Automation Conference (DAC'01), pages 530-535 (2001).
[10] Aloul, F., Ramani, A., Markov, I. and Sakallah, K.: PBS: A backtrack search pseudo Boolean solver, In Symposium on the theory and applications of satisfiability testing (SAT), pages 346-353 (2002).
[11] 小池隆：モデル検査支援ツールと ZIPC との連携による設計検証の導入事例．ZIPC WATCHERS Vol.13.
[12] 小池隆：状態遷移表に基づくモデル検査の支援環境，情報処理学会研究報告 Vol.2008 No.116 （EMB-10） (2008).
[13] 矢野恭平，小池隆：状態遷移表のモデル検査における LTL 検証パターン，情報処理学会研究報告 Vol.2009 No.31（SE-163） (2009).
[14] 通信制御ソフトウェア開発における状態遷移設計の品質向上への取組み，情報処理推進機構，先進的な設計・検証技術の適用事例報告書 2015 年度版，PARTⅢ (2015).
[15] 高田沙都子，森奈美子，村田由香里：モデル検査自動化ツールの開発～検査自動化と反例解析効率化～，情報処理学会第 74 回全国大会講演論文集 (2012).
[16] 森奈美子，高田沙都子，長谷川保，村田由香里，進博正：モデル検査自動化ツールの開発～入力支援機能と状態遷移表縮約機能～, 情報処理学会大 74 回全国大会講演論文集 (2012).