モデル検査における複雑な検査式に対する反例解析手法の提案

大池　勇太郎[1,a)]　小形　真平[1,b)]　青木　善貴[3,c)]　中川　博之[2,d)]　小林　一樹[1,e)]
岡野　浩三[1,f)]

概要：NuSMV によるモデル検査では，多数の部分式に分解できる長大な検査式に対して，大規模な反例が出力された場合，反例で満たされていない部分式（以下，不満足な部分式）を手動で識別することに時間がかかる問題がある．そこで本研究では，不満足な部分式を自動識別するための解析手法と支援ツールを提案する．提案手法では，構文解析により複雑な検査式を部分式に分解し，部分式が反例で満たされるかどうかを調べて不満足な部分式を識別する．提案手法の有効性評価のために交通制御システムのモデルに適用した結果，不満足な部分式を正しく識別できたため，提案手法は有効である見込みを得た．

キーワード：NuSMV, 反例，モデル検査，構文解析

A Method to Analyze Counterexample for Complex Specification in Model Checking

1. はじめに

モデル検査 [7], [14] はソフトウェア開発の上流工程を支援する形式手法の一種であり，システムのモデルが検査式として定義した仕様を満たしているか否かを網羅的に検査する手法である．この手法では，開発者がシステムのモデルを有限ステートマシン，仕様を検査式として時相論理式で定義し，モデル検査ツールを用いて検査を行う．モデル検査ツールの一種に NuSMV[6], [16] がある．NuSMV は大規模・複雑なシステムのモデルを検証するのに適しているという特徴がある．実際に IoT の設計に NuSMV を適用した論文がいくつか存在している [5], [15].

NuSMV に SMV 形式で定義された，モデルと検査式を入力すると，検査式を満たしていた場合は “TRUE” が出力され，満たさない場合は “FALSE” と反例が得られる．しかし，NuSMV の反例では変数の状態遷移が文字列で出力されるだけである．そのため，大規模・複雑なシステムのモデルに対し，多数の部分式に分解できる長大な検査式を適用して大規模な反例が出力された場合，反例で満たされていない部分式（以下，不満足な部分式）を手動で識別することに時間がかかる問題がある．また，NuSMV は一つの反例に対して検査式が満たしているか識別する機能は存在しない．満足できない仕様を反例から識別することは検証ひいては不具合修正の作業上で不可欠であるが，NuSMV に関する従来の方法 [2], [3], [13], [20] では，一つの反例と複数の検査式を元に不満足な検査式を識別するために対比を行い反例解析を効率化支援する手法は提案されていない．

そこで，本研究はこの問題を解決するための NuSMV の解析手法と支援ツールを提案する．提案手法では複雑な検査式を用いたモデル検査を対象とし，検査式を部分式に分解し，反例と対比して不満足な部分式を特定する方法を新たに実現する．提案ツールはこの特定方法を自動化するためのものである．

本稿では，ケーススタディとして，交通制御システムの

[1] 信州大学
Shinshu University
[2] 大阪大学
Osaka University
[3] 日本ユニシス株式会社
Nihon Unisys, Ltd.
[a)] 19w2020g@shinshu-u.ac.jp
[b)] ogata@cs.shinshu-u.ac.jp
[c)] yoshitaka.aoki@unisys.co.jp
[d)] nakagawa@ist.osaka-u.ac.jp
[e)] kby@shinshu-u.ac.jp
[f)] okano@cs.shinshu-u.ac.jp

モデルのコントロールループを検証する検査式に対して，提案手法を適用した結果を報告する．本交通制御システムに見られる CPS/IoT 形態のシステムは，複数のモジュールにおいて制御プロセスと被制御プロセスの相互作用によって動作するコントロールループが形成される [12], [21]．多数のモジュールの相互作用を検査する場合，たとえば，コントロールループが停止するかどうかを検査する検査式 [1], [18] では，一つのモジュールの一つの動作が一つの部分式となることで多数の部分式に分解できる程に検査式が長大化する．

提案手法を適用・評価した結果として，提案手法は不満足な部分式を正しく識別することができたため，有効であるという見込みを得た．本研究の貢献を以下に示す．

- 反例に基づき不満足な部分式を自動特定することで複雑な検査式から容易にモデルの欠陥を見つけるための新しい手法を提案する．
- 提案手法で正確に不満足な部分式を正しく特定できることをケーススタディを通して実証する．

第 2 章では要素技術と研究課題の説明を行う．第 3 章では提案手法の説明を行う．第 4 章ではケーススタディの説明を行い，第 5 章でその考察を行う．第 6 章では関連研究の説明を行う．第 7 章では本稿の内容や今後の課題についてまとめを行う．

2. 要素技術と研究課題

2.1 NuSMV を用いたモデル検査

NuSMV[17] はモデル検査ツールの一種であり，SPIN[10] や UPPAAL[11] などの他のモデル検査ツールに比べて大規模・複雑なシステムのモデル検査に適しているという特徴がある．NuSMV を使用するためには smv 形式で記述したモデルと線形時相論理（LTL）または計算木論理（CTL）で記述した検査式が必要である．図 1 は，NuSMV を用いて検査できるモデルと検査式の簡単な例を示している．

このモデルは変数 “a”，“b”，“switch” が定義されており，それぞれの初期値は “0”，“0”，“off”，である．変数 “a” の値は “b = 1” の時は遷移せず，それ以外の場合には “(a + 1) mod 3” に遷移する．変数 “b” の値は “a = b” の時は遷移せず，それ以外の場合には “(b + 1) mod 3” に遷移する．変数 “switch” の値は “a = b = 1” の時は “off” に，“a = b = 2” の時は “on” に遷移し，それ以外の場合には遷移しない．

“ G(F(switch = off & F(switch = on)))” は LTL で定義した検査式の例である．この検査式は常に “switch” が “on” と “off” を切り替わることを検査するための式である．表 1 に LTL の時相論理記号とその意味を示す．このモデルと検査式を用いて NuSMV を用いて検査を行った場合 False と図 2 に示す反例が出力される．この反例では “State1.1” から “State1.4” の状態にそれぞれ一度遷移した

```
MODULE main
--変数宣言 変数a,b,switchがある
--{}内のそれぞれの値どれかを持つ
VAR
    a : {0,1,2};
    b : {0,1,2};
    switch : {on, off};

ASSIGN
--変数の初期値
    init(a) := 0;
    init(b) := 0;
    init(switch) := off;
--変数の状態遷移
    next(a):=case
        b = 1 : a;
        a = 0: 1;
        a = 1: 2;
        a = 2: 0;
        TRUE: a;
        esac;

    next(b):=case
        b = a : b;
        b = 0: 1;
        b = 1: 2;
        b = 2: 0;
        TRUE: b;
        esac;

    next(switch):=case
        a = b & b = 2 : on;
        a = b & b = 1 : off;
        TRUE : switch;
    esac;

--検査式
LTLSPEC G (F switch = on & F (switch = off))
```

図 1 簡単なシステムのモデルと LTL で記述した検査式

表 1 LTL の時相論理記号 [16]

LTL 時相論理記号	説明
$X\varphi$	φ が state t' $(t' = t+1)$ において真である時，$X\varphi$ は真である．
$F\varphi$	φ が state t' $(t' \geq t)$ においていつか真である時，$F\varphi$ は真である．
$G\varphi$	φ が state t' $(t' \geq t)$ において常に真である時，$G\varphi$ は真である．
$\psi U \varphi$	φ が state t' $(t' \geq t)$ においていつか真であるかつ，ψ が $t'', (t' > t'' \geq t)$ において常に真である時，$\psi U \varphi$ は真である．

```
-> State: 1.1 <-
  a = 0
  b = 0
  switch = off
-> State: 1.2 <-
  a = 1
-> State: 1.3 <-
  a = 2
  b = 1
-> State: 1.4 <-
  b = 2
-- Loop starts here
-> State: 1.5 <-
  a = 0
  switch = on
-> State: 1.6 <-
  a = 1
  b = 0
-> State: 1.7 <-
  a = 2
  b = 1
-> State: 1.8 <-
  b = 2
-> State: 1.9 <-
  a = 0
```

図 2 NuSMV をから出力された反例

後に，“Loop starts here” という記述後の “State1.5” から “State1.9” の状態をループすることを示している．

今回の検査式では全ての状態において部分式 “F switch = off” と “F switch = on” を満たすことが定義されている．ループ開始後に “switch = on” は常に満たすが，“switch = off” を満たすことはない．よって部分式 “F switch = off” を満たさないが故に反例が出力されたということができる．

2.2 ケーススタディ

2.2.1 交通制御システムモデル

Filho らは本稿のケーススタディとして用いる交通制御システムを例題としてあげている [8]．このシステムはスマート信号制御アプリケーションを用いて車，信号，デバイス間で通信を行い交通制御を行うシステムである．

図 3 は，交通制御システムにおける各コンポーネントの役割と，コンポーネント間の制御プロセスと被制御プロセスの関係を表すコントロールストラクチャ図である．なお，コントロールストラクチャ図は安全解析手法の一種である STAMP/STPA[24] において用いられる図の一種である．コントロールストラクチャ図では制御プロセスから被制御プロセスへの制御指示を赤の矢印で示す．また，被制御プロセスから制御プロセスへのフィードバックを青の矢印で示す．

図 4 は，図 3 のモデルに基づく小規模な構成例をオブジェクト図として示す．本構成例は本稿でケーススタディにおける適用事例ともなっており，適用事例に関連する各コンポーネントインスタンスの役割と定義された変数の数，また，検査式に登場する変数とその概要も示している．

NuSMV では構造化された型に対する複数のインスタンスを扱うこともあり，変数の総数は約 110 個に及ぶ．また，モデルは約 600 行で定義されている．

今回は，このモデル内の以下のコントロールループを検証するための検査式を用いて検証を行う．このコントロールループでは，本システムが正常な場合の基本的な動作を示す．

Step 1. デバイスは車をモニタリングする．

Step 2. デバイスは全車両のデータをスマート信号制御アプリケーションに送信する．

Step 3. スマート信号制御アプリケーションは受信したデータに基づき，信号機の制御を行う．

Step 4. 車両は信号機をモニタリングし，運転を行う．

各 step において制御プロセスと被制御プロセスを検査式として定義する．

本モデルは先行研究 [23] において，一人の企業研究者が当該システムのモデルと検査式を作成し，洗練を行った．洗練の過程において，ソフトウェア設計を専門とする学術研究者が二人と CPS アプリケーションを専門とする学術研究者一人がモデルと検査式のレビューを行なった．

図 5 に検査式で部分式として定義するシステムモデルの各動作を示す．図 6 は図 5 で定義された動作を NuSMV で検査をするために用いる LTL で示された検査式である．各行は図 5 で示した各プロセスに対応している．例として，部分式 “CDV.Monitor_A_For_V1 = Monitoring” は “デバイスが車をモニタリングする” というプロセスに対応している．上述したシステムのモデルと図 6 に示した検査式を用いて検査を行った結果，反例が出力された．この反例は約 500 行で 49 の状態から成り立つ．

2.2.2 長大な反例分析のコスト削減

NuSMV は大規模・複雑なシステムを検証することができる特徴がある．しかし，約 500 行，49 状態，109 種類の変数からなる長大な反例から，不満足な部分式を特定するのは時間がかかる．

しかし，NuSMV の反例から不満足な部分式を自動で特定するなどして開発者負担を低減する手法はない．そこで，この課題解決のために NuSMV から出力された反例と複雑な検査式から不満足な部分式を自動で出力するツールを提案する．最終的には，開発者がモデルの欠陥を発見する作業時間の低減を目的としている．

3. 提案手法

本研究では，NuSMV から得られた反例に対して，複雑

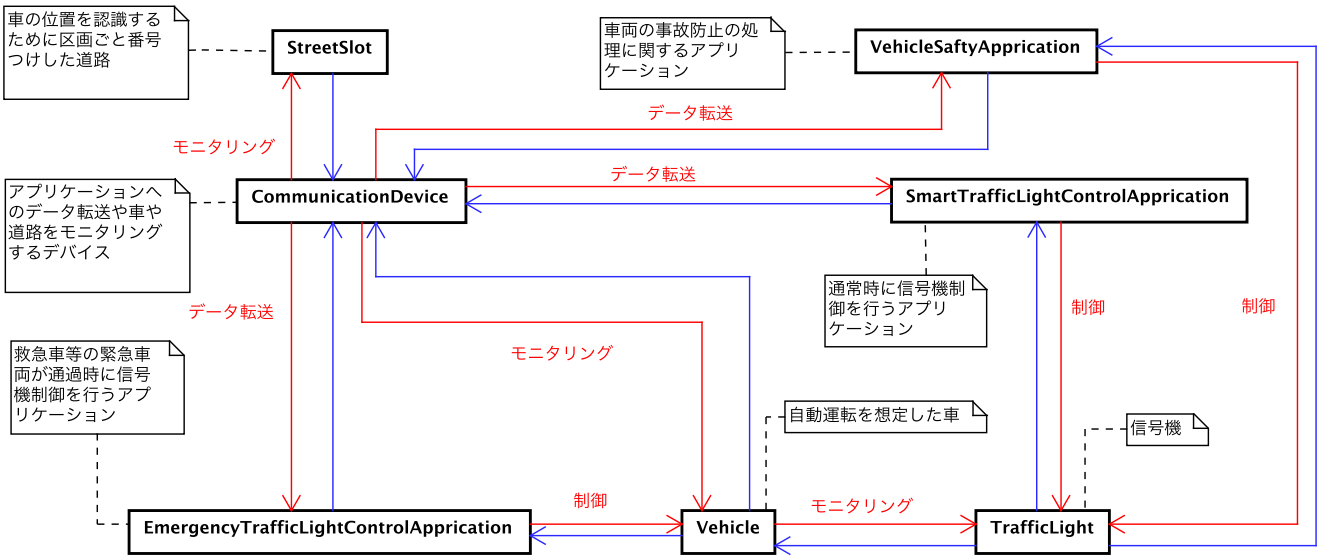

図 3　ケーススタディモデルのコントロールストラクチャ図

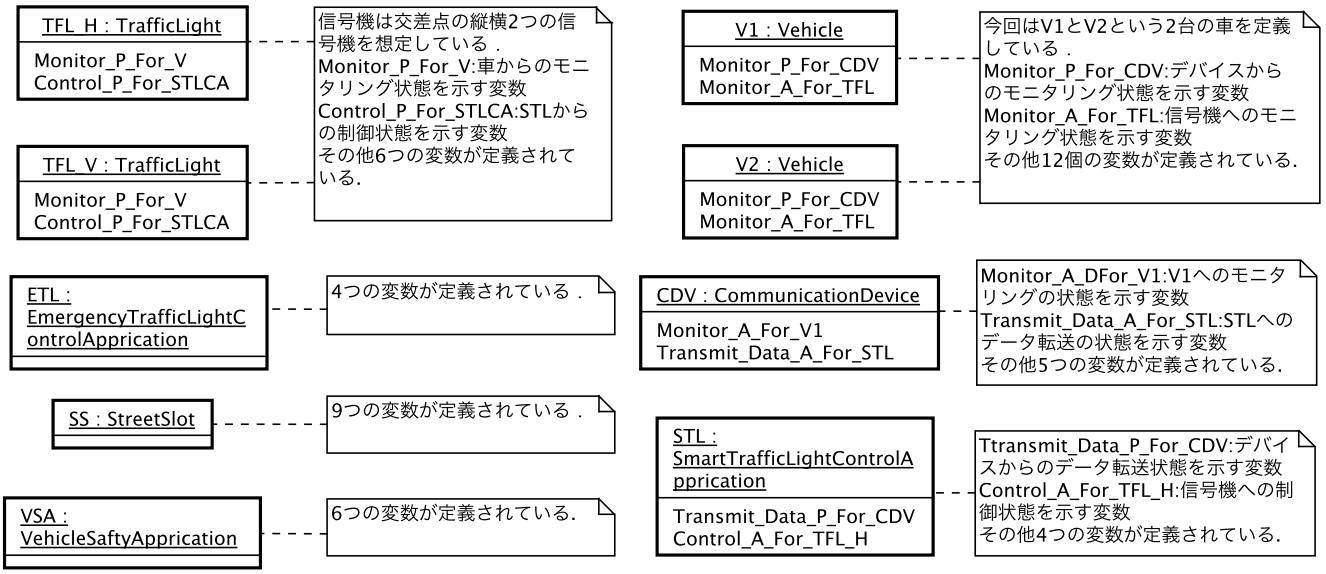

図 4　ケーススタディモデルのオブジェクト図

な検査式から不満足な部分式を自動的に識別する手法を提案する．また，提案手法を実現するためのツールを提案する．提案手法では検査式を部分式に分解し，部分式と反例の対比を行い，不満足な部分式の特定を行う．

3.1　概要

図 7 に提案手法の概要を示す．提案ツールのプロトタイプを Python3 で作成した．以下に提案手法の手順を説明する．

Step 1. 開発者は提案ツールに smv 形式のモデルと LTL 形式の検査式を入力する．

Step 2. 提案ツールは NuSMV を用いてモデル検査を行う．

Step 3. 反例が得られた場合は提案ツールは Step4 の処理を行う．それ以外の場合は処理を終了する．

Step 4. 提案ツールは与えられた検査式をレキサとパーサを用いて分解を行い，部分式を得る．詳細は 3.2 節で説明を行う．

Step 5. 提案ツールでは得られた反例を用いて検査式内の部分式の中で満たされていないものを特定する．反例解析の詳細は 3.3 節で説明を行う．

Step 6. 開発者は，反例解析の結果からシステムモデルの欠陥を探す．

3.2　検査式分解

反例に基づいて検査式内の不満足な部分式を特定するた

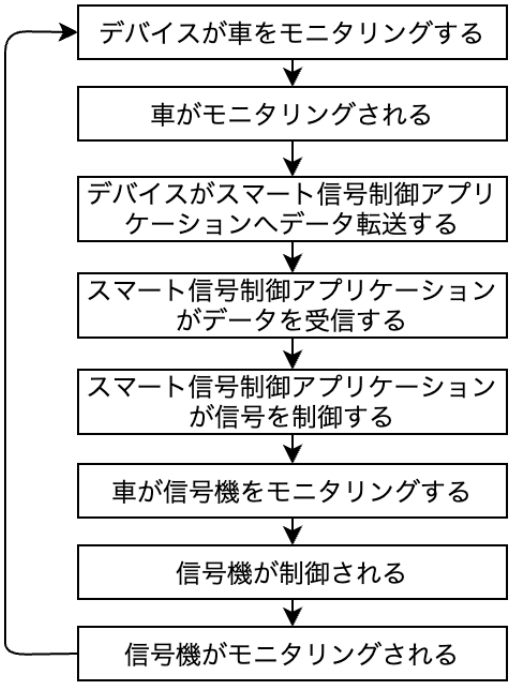

図 5 交通制御システムの検査式の概要

めに LTL 検査式を分解する方法を説明する．LTL 検査式の分解を行うためには語彙規則と構文規則に沿って分解を行うレクサとパーサが必要になる．そこで，LTL 検査式のパーサを作成するためにパーサジェネレータの一種である antlr4 [19] を使用した．語彙規則と構文規則は “NuSMV 2.6 User Manual” [16] に即して，antlr4 の入力形式の規則を手動で作成した．antlr4 では作成した規則に対応するパーサプログラムを自動で生成することができ，言語として python3 を選択することができる．図 8 は，図 1 の検査式を分解した結果を示す．本稿では部分式への分解として，論理演算子 “&” に基づいて分解できる最小の単位への分解をする．論理演算子 “|” で接続された部分式が反例の出力原因となる場合，接続された部分式全てが満たさないが故に反例が出力される．そのため，論理演算子 “|” に基づいて分解を行う意味はない．論理演算子 “!” や時相論理記号はそれぞれ，分析するために必要な要求を示しているため，時相論理記号に基づいて分解することはできない．論理演算子 “->” は時相論理記号 “U” と同様に前後の式に基づいて満たしているか確認する必要があるため，“->” に基づいて分解は行わない．以上の理由より，本稿で部分式は “&” に基づいて分解できるものを最小の単位として分解を行う．

3.3 反例解析

反例から検査式内の不満足な部分式を特定する方法を説明する．不満足な部分式を特定するためには，部分式と反例の各状態を対比する必要がある．また，対比の手順としては，LTL の時相論理記号のセマンティクスに依存する．

表 2 に提案ツールにおける反例解析の LTL 演算子ごとの処理を示す．反例解析の流れを図 1 のモデルと検査式を用いて説明を行う．

図 8 の二行目より，時相論理記号 “G” が “F(switch = off & F(switch = on))” に対応している．そこで，提案ツールでは “F(switch = off & F(switch = on)” が満たされているか否かを “State1.1” から各状態において満たすか否かを確認する．その中で，図 8 の四行目より，時相論理記号 “F” が “ switch = off & F(switch = on) ” に対応している．そのため，提案ツールでは “ switch = off & F(switch = on) ” がいつか満たすか否かを “State1.1” から確認する．部分式 “ switch = off & F(switch = on)” を満たすためには，部分式 “ switch = off” と部分式 “ F(switch = on)” の両方を満たす必要がある．

提案ツールではまず，“state1.1” から “switch = off” が満たされるか否かを確認する．その後，いつか “switch=on” を満たすか否かを確認する．結果として，“switch = off” は “state1.1” で満たす．その後 “state1.5” において “switch = on” を満たす．部分式 “F(switch = off & F(switch = on)) ” は一度満たした．しかし，繰り返し満たすか否かを確認すると “state1.5” 以降で “switch = off” を満たさない．故にこの反例は部分式 “F (switch = off)” が満たさないことが原因で出力された考えられる．提案手法ではこのような反例の出力原因となる部分式を特定することを目的としている．

図 2 の反例の解析手順を図 9 に示す．提案ツールでは時相論理記号に即した処理をそれぞれの部分式について行う．次に，部分式をそれぞれの “state” で満たすか否かを順次対比する．部分式 “switch = off” と “F (switch = on)” は “&” を用いて結合されている．そのため，提案ツールでは最初に “switch = off” を，満たすか否かを確認し, 満たした場合, 次に，“F (switch = on)” を満たすか否か確認する．これが “G” に対応しているため，それを繰り返す．

提案ツールによる反例解析の結果を図 10 に示す．“expression” は部分式を示す．“result” は対応する部分式が満たしたか否かを示す．“startState” は対応する部分式の確認を開始した状態番号を示す．“fillState” は対応する部分式が満たされた状態番号を示す．図 10 より，満たされない部分式として “F (switch = off)” を出力し，部分式の確認を始めた状態として “state1. 5” を出力することが分かる．

4. ケーススタディ

4.1 概要

本評価では，提案手法が不満足な部分式を正しく識別できるかどうかを調査するために，2.2 節で示した複雑な検査式と長大な反例に対して提案手法を適用した．

```
LTLSPEC G F (CDV.Monitor_A_For_V1 = Monitoring
& F (CDV.Monitor_A_For_V1 = Monitoring U V1.Monitor_P_For_CDV = Monitored
& F (V1.Monitor_P_For_CDV = Monitored U CDV.Transmit_Data_A_For_STL = Transmitting
& F (CDV.Transmit_Data_A_For_STL = Transmitting U STL.Transmit_Data_P_For_CDV = Transmitted
& F (STL.Transmit_Data_P_For_CDV = Transmitted U STL.Control_A_For_TFL_H = Controlling
& F (STL.Control_A_For_TFL_H = Controlling U TFL_H.Control_P_For_STLCA = Controlled
& F (TFL_H.Control_P_For_STLCA = Controlled U V1.Monitor_A_For_TFL= Monitoring
& F (V1.Monitor_A_For_TFL= Monitoring U TFL_H.Monitor_P_For_V = Monitored)
)))))))) ;
```

図 6　交通制御システムの検査式

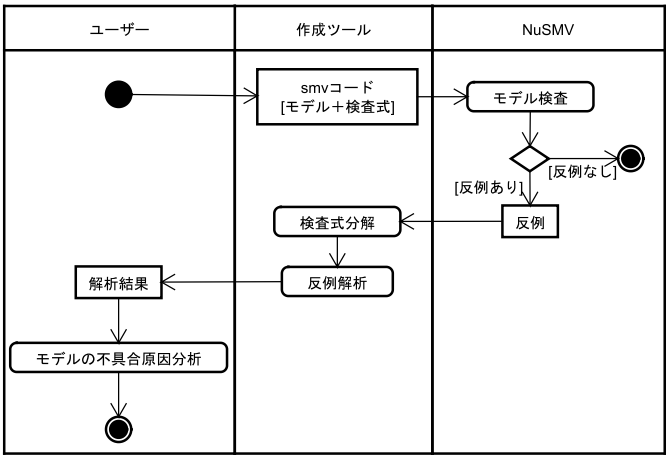

図 7　提案手法の概要

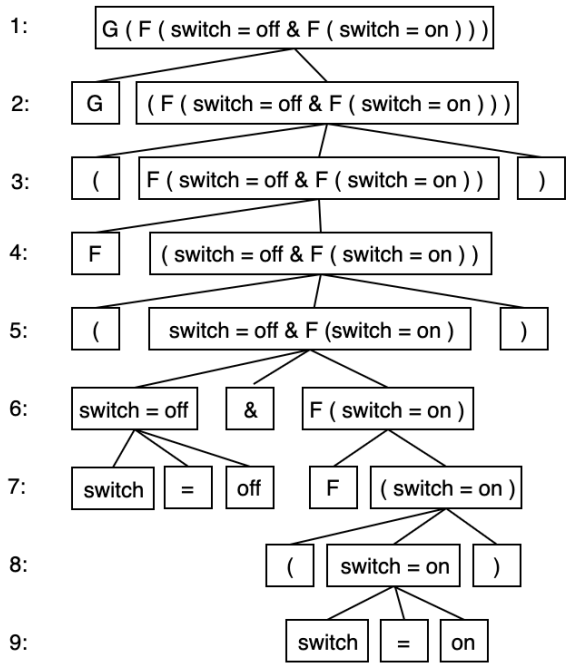

図 8　検査式分解

4.2　手順

ケーススタディにおける提案手法の適用手順を以下に示す．本ケーススタディでは，不具合原因が特定できている交通制御システムの smv ファイル（モデルと検査式）を適用事例とする．なお，本事例は先行研究 [23] で作成されたものを応用したものである．

(1) 実験者が smv ファイルを提案ツールに入力する．
(2) 提案ツールがモデル検査で得られる反例を解析し，結果を出力する．
(3) 実験者は提案ツールの出力が不満足な部分式を正しく指摘しているかを手動で確認する．
(4) 実験者は出力された不満足な部分式をもとに，モデル上で不具合原因箇所に辿り着くかどうかを手動で確認する．

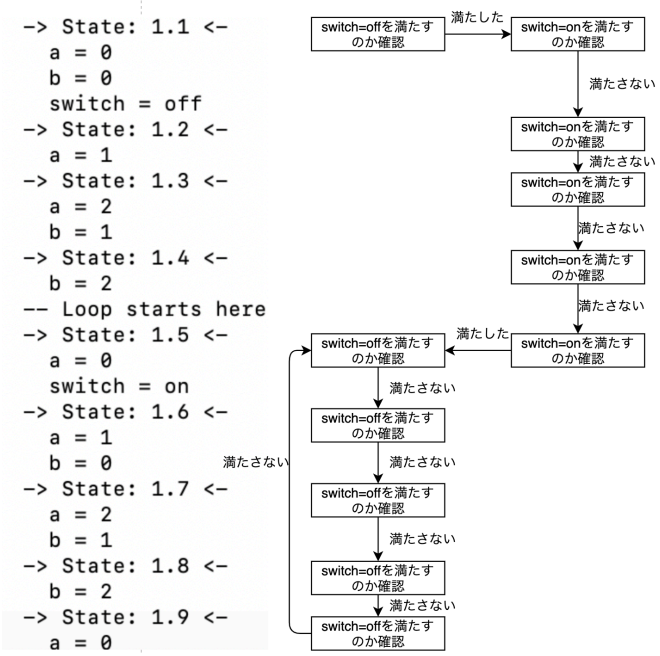

図 9　反例解析

```
expression : F ( switch = off ) , result : True
startState : 1 fillState : 1
↓
expression : F ( switch = on ) , result : True
startState : 1 fillState : 5
↓
expression : F ( switch = off  ) , result : False
startState : 5
```

図 10　反例解析における解析結果

4.3　結果

提案ツールの出力結果を図 11 に示す．図 5 で示した “車が信号機をモニタリングする” ことを定義した部分式が満

```
expression : F ( CDV.Monitor_A_For_V1 = Monitoring ) , result : True
startState : 1 fillState : 2
↓
expression : F ( CDV.Monitor_A_For_V1 = Monitoring U V1.Monitor_P_For_CDV = Monitored ) , result : True
startState : 2 fillState : 4
↓
expression : F ( V1.Monitor_P_For_CDV = Monitored U CDV.Transmit_Data_A_For_STL = Transmitting ) , result : True
startState : 4 fillState : 4
↓
expression : F ( CDV.Transmit_Data_A_For_STL = Transmitting U STL.Transmit_Data_P_For_CDV = Transmitted ) , result : True
startState : 4 fillState : 13
↓
                                                    .
                                                    .    途中省略
                                                    .
expression : F ( CDV.Transmit_Data_A_For_STL = Transmitting U STL.Transmit_Data_P_For_CDV = Transmitted ) , result : True
startState : 19 fillState : 22
↓
expression : F ( STL.Transmit_Data_P_For_CDV = Transmitted U STL.Control_A_For_TFL_H = Controlling ) , result : True
startState : 22 fillState : 22
↓
expression : F ( STL.Control_A_For_TFL_H = Controlling U TFL_H.Control_P_For_STLCA = Controlled ) , result : True
startState : 22 fillState : 24
↓
expression : F ( TFL_H.Control_P_For_STLCA = Controlled U V1.Monitor_A_For_TFL= Monitoring  ) , result : False
startState : 24
```

図 11 ケーススタディに対する提案ツールの出力

表 2 反例解析における LTL 時相論理記号ごとの処理プロセス

LTL 時相論理記号	処理プロセス
$X\varphi$	現在の状態から次の状態で φ が成り立つか確認する．この部分式が “false” になった際に，$X\varphi$ を不満足な部分式として出力する．
$F\varphi$	現在の状態以降で φ が成り立つか確認する．この部分式が “false” になった際に，$F\varphi$ を不満足な部分式として出力する．
$G\varphi$	現在の状態以降の各状態全てにおいて φ が成り立つか確認する．この部分式が “false” になった際に，$G\varphi$ を不満足な部分式として出力する．
$\psi U\varphi$	現在の状態から，φ の直前の状態までの各状態で，ψ が満たされているかどうかを確認する．以下の二つのケースの場合に $\psi U\varphi$ を不満足な部分式として出力する． (1) φ を満たす状態がない場合． (2)φ を満たす直前の状態までの各状態において ψ を満たさない状態がある場合．

たされていないことを示している．このように，検査式で複数定義された部分式のうち，反例において不満足な部分式の特定を自動で行うことができた．

この部分式が満たされない原因をモデルの状態遷移の条件から手動で分析を行った．結果として，二台の車両が同じ道路に同じ方向から侵入しようとしたために予期せず停止していることがわかった．二台の車両は停止を続けているため信号機をモニタリングできず，コントロールループが成り立たなかった．これは，このモデルにおいて，既知の不具合原因であり，正しく不満足な部分式の特定ができたと言える．

5. 考察

ケーススタディの結果より，提案手法は反例が不満足な部分式を自動的に識別できたと言える．今回のケーススタディでは，提案手法は “U” を含む 8 つの部分式の中から一つの部分式を特定することができた．特定した部分式はモデルの欠陥を発見する補助として適切なものであった．そのため，本事例では実験者の反例解析のコストを削減することができたと言える．したがって，提案手法はモデルの欠陥検出の効率化に寄与できる見込みを得た．

今回のケーススタディでは全ての部分式に “G” が対応している．提案手法では図 11 の最後の結果のみではモデルが一度も満たさないのか，あるいは何度か満たしたのちに満たさなくなるのかわかりずらい．そのため，解析結果を可視化するなど，より開発者が利用しやすいように改善する必要がある．提案手法を適用したのは本ケーススタディのみである．その一因として，大規模・複雑なシステムの NuSMV のコードと要求仕様を明確にした例が少ないことがある．例えば，NuSMV の公式サイト [17] では，完全なコードが複数公開されているが，いずれの例も要求仕様が明確に定義されていない．そのため，今後は要求仕様を明確にすることも含めて，長期的に複数の事例に対し，本提案手法を適用し，その汎用性を評価する．また，今回は LTL に焦点を当てたが，CTL についても効果的である可能性が高いと考えられるため今後調査を行う．

6. 関連研究

関連研究として反例分析を支援するための手法がいくつか提案されている．NuSeen [2] は NuSMV を用いたモデリ

ングや検証を支援するフレームワークである．このツールは，構文の静的解析，変数の依存関係や反例の可視化などの機能がある．しかし，反例に基づいて検査式内の不満足な部分式を特定する機能はない．

Barbon ら [4] は反例解析におけるラベル付き遷移システム（LTS）のデバッグ手法を提案している．この手法ではモデルと反例を対比することにより，解析支援を行なっている．提案手法では反例と検査式を対比するため，本手法とは異なる．加えて，我々の先行研究 [1][18] では NuSMV を用いた大規模・複雑なモデルの検証を行なっているが，本手法では NuSMV に焦点を当てていないため，適用が難しい．

他のモデリング言語のモデルから NuSMV のコードを生成し，NuSMV のモデル検査を利用する方法はいくつかある [3], [13], [15], [22] が，これらの方法で反例解析に焦点を当てたものは存在しない．そのため，提案手法はこれらの手法と連携することにより，他のモデリング言語のモデルの解析補助に役立つ可能性がある．

Gastin ら [9] は SPIN[10] を用いて計算時間とメモリ容量が足りなくなる複雑な検査式の検証を行うために LTL 検査式の Büchi オートマトンへの変換アルゴリズムを提案している．本手法は反例解析は行なっていないが，提案手法の処理の一部を本手法に置き換えることで汎用性を高められる可能性がある．

7. おわりに

本稿では，大規模・複雑なシステムの検証を支援するために，LTL 検査式内の部分式と反例に基づき不満足な部分式を自動的に特定する手法を提案した．提案手法では，LTL 検査式を分解しその部分式と反例を対比することにより，不満足な部分式を導出する．ケーススタディでは，提案手法を用いて，複雑な検査式と長大な反例から，不満足な部分式を自動的に識別することができた．よって，モデルの欠陥検出の効率化に寄与することが期待できる．今後の課題として，得られた解析結果の可視化や，多くの事例に適用してその汎用性を評価することがあげられる．

参考文献

[1] Yoshitaka Aoki, Shinpei Ogata, Kazuki Kobayashi, and Hiroyuki Nakagawa. Verification of cps based on control loop using model checking. In *Proc. of the 25th Asia-Pacific Software Engineering Conference (APSEC) 2018*, pp. 678–682. IEEE, 2018.

[2] Paolo Arcaini, Angelo Gargantini, and Elvinia Riccobene. NuSeen: a tool framework for the NuSMV model checker. In *Proc. of the IEEE International Conference on Software Testing, Verification and Validation (ICST) 2017*, pp. 476–483. IEEE, 2017.

[3] Salma Ayari., Yousra Bendaly Hlaoui., and Leila Jemni Ben Ayed. A refinement based verification approach of bpmn models using nusmv. In *Proc. of the 13th International Conference on Software Technologies (ICSOFT)*, Vol. 1, pp. 529–540. INSTICC, SciTePress, 2018.

[4] Gianluca Barbon, Vincent Leroy, and Gwen Salaun. Debugging of behavioural models using counterexample analysis. *IEEE Trans. of Soft. Eng.*, 2019. early access.

[5] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated iot safety and security analysis. In *Proc. of the 2018 USENIX Conference on Usenix Annual Technical Conference*, pp. 147–158. USENIX Association, 2018.

[6] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proc. of the 14th International Conference on Computer Aided Verification (CAV) 2002*, pp. 359–364, Berlin, Heidelberg, 2002. Springer-Verlag.

[7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, Vol. 8, No. 2, pp. 244–263, 1986.

[8] J. G. Filho, N. Przigoda, R. Wille, and R. Drechsler. Towards a model-based verification methodology for complex swarm systems. In *Proc. of the Sixth International Symposium on Embedded Computing and System Design*, pp. 18–23, 2016.

[9] Paul Gastin and Denis Oddoux. Fast ltl to büchi automata translation. In *International Conference on Computer Aided Verification*, pp. 53–65. Springer, 2001.

[10] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, Vol. 23, No. 5, pp. 279–295, 1997.

[11] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *Int. J. Softw. Tools Technol. Transf.*, Vol. 1, No. 1–2, pp. 134–152, 1997.

[12] Nancy Leveson. *Engineering a safer world: Systems thinking applied to safety*. MIT press, 2011.

[13] Diego Marmsoler and Silvio Degenhardt. Verifying patterns of dynamic architectures using model checking. In Jan Kofron and Jana Tumova, editors, *Proc. of the International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA@ETAPS) 2017*, Vol. 245, pp. 16–30, 2017.

[14] Kenneth L. McMillan. *Symbolic Model Checking*, pp. 25–60. Springer US, 1993.

[15] Kazuya Nakahori and Shingo Yamaguchi. A support tool to design iot services with nusmv. In *Proc. of the IEEE International Conference on Consumer Electronics (ICCE) 2017*, pp. 80–83. IEEE, 2017.

[16] NuSMV. Nusmv 2.6 user manual. http://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf (accessed 28 Feb. 2020).

[17] NuSMV. Nusmv examples: the collection. http://nusmv.fbk.eu/examples/examples.html (accessed 28 Feb. 2020).

[18] Shinpei Ogata, Yoshitaka Aoki, Hiroyuki Nakagawa, and Kazuki Kobayashi. A template system for modeling and verifying agent behaviors. In *Proc. of the International Conference on Principles and Practice of Multi-Agent Systems (PRIMA) 2018*, pp. 576–584. Springer, 2018.

[19] Terence Parr. Antlr 4. http://www.antlr.org/ (accessed 28 Feb. 2020).

[20] S Sheerazuddin, S Anand, and RS Badhri. A scheme to verify services with unboundedly many clients using nusmv. *arXiv preprint arXiv:1812.00183*, 2018.

[21] John A Stankovic. Research directions for the internet of things. *IEEE Internet Things J.*, Vol. 1, No. 1, pp. 3–9, 2014.
[22] Hongli Wang, Deming Zhong, Tingdi Zhao, and Fuchun Ren. Integrating model checking with sysml in complex system safety analysis. *IEEE Access*, Vol. 7, pp. 16561–16571, 2019.
[23] 青木善貴, 小形真平, 中川博之. STAM-P/STPA を用いた Cyber-Physical System の検証. 第 2 回 STAMP ワークショップ, 2017. https://www.ipa.go.jp/files/000063285.pdf.
[24] 福澤寧子. Stamp/stpa によるシステム安全・セキュリティ解析. システム/制御/情報, Vol. 62, No. 4, pp. 130–133, 2018.