社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－SE－163 (23)
2009／3／19

軽量形式手法を用いた車載電子システムの安全性分析に関する研究

押川 倫憲† 武藤 嘉伸†† 豊島 真澄††† 鈴村 延保†††† 鬼頭 正広††

† 北九州市立大学大学院 国際環境工学研究科
†† アイシン・コムクルーズ株式会社
††† 北九州市立大学
†††† アイシン精機株式会社

要 旨

本稿では，車載電子システムの信頼性や安全性を高めるために，仕様書を基に軽量形式手法Alloyで記述及び解析を行った．Alloyを用いた記述及び解析は，ドメイン分析や要求分析工程においてトップダウンに行われることが多いが，本研究では，リバースエンジニアリングの道具としてAlloyを用い，対象システムの分析を支援する．実際に，車体系製品を想定した仕様書を基にAlloyで記述及び解析を行うことによって，制約の顕在化やドメイン知識の習得，要求仕様を踏まえた仕様の安全性分析などを行うことができた．結果，開発の早期において抜け漏れを防ぎ，信頼性や安全性の向上に寄与すると考えられる．

Analyzing Safety Properties of Automotive Electronic Systems with a Lightweight Formal Method

Tomonori Oshikawa† Yoshinobu Muto†† Masumi Toyoshima†††
Nobuyasu Suzumura†††† Masahiro Kito††

†Graduate School of International and Environmental Engineering, University of Kitakyushu
††AISIN COMCRUISE Co., Ltd.
†††University of Kitakyushu
††††AISIN SEIKI Co., Ltd.

Abstract

In this article, we report our experience of applying a lightweight formal method called Alloy to Automobile Electronic Systems(AES). Usually, Alloy model-finding method is used as a top-down approach in domain analysis or requirement analysis phase. Unlike it, we regard Alloy as a reverse engineering tool and apply Alloy model-finding method onto specification and design model. Actually, we constructed Alloy models and analyzed it based on a specification of case study application to an virtual body system in AES. As a result, we could reveal constrains, acquire domain knowledge, and analyze safety properties of designs based on requirement specifications. Hence it can prevent oversights and help to construct high-safety systems.

1 はじめに

近年，車載電子システムの大規模化・高機能化が急激に進んでいる．それに伴い，従来行われてきたテスティング手法や完成品に対する試験だけでは信頼性や安全性を確保する上で不十分であるという認識が広がってきている．このような従来の手法は，開発プロセスの下流工程で行われることが多く，仕様に対する実装の誤りを検出できるとしても，仕様や設計の誤りそのものを発見することはできない．また間接的にその誤りを発見できたとしても，手戻りが発生し，信頼性や安全性の確保に必要な工数の

増大やプロジェクトの長期化などの問題が発生してしまう．

そこで本研究では，車載電子システムの信頼性や安全性を高めるために，軽量形式手法 Alloy[1][2] を開発の上流工程に適用した．一般的に Alloy による記述及び解析は，トップダウンに適用されることが多いが，本研究ではそれとは異なるアプローチを採用する．具体的には，仕様書に基づいてモデルを構築し，解析を行うことによって，ドメイン知識の習得，要求仕様のチェック，安全性分析等を行う．つまり，リバースエンジニアリングの道具としての Alloy の活用を試みる．

また，事例適用によってその有効性を確認するが，本研究では「車体系」の製品群に属する「ラゲージクローザシステム」を想定した仕様書を対象とする．一般的に車載電子システムは，パワートレイン系，シャシー系，車体系，情報通信系といった 4 つの製品群に分類されるが，それぞれの領域において，求められる性質（リアルタイム要求や信頼性要求）や制御方法，動作環境などが異なるため，ひとまとまりに扱う事は適切とは言えない．車体系は，パワートレイン系やシャシー系と比べ，リアルタイム要求が比較的緩やかだが，高い安全性や信頼性が求められる製品群である．また，仕様は状態遷移モデル中心に考えられることが多く，実際に独自の状態遷移図を用いて仕様書が作成されていた．本研究の初期段階における仕様書の解析で，エンジニアが使用している仕様書は一般的な記法とは言えないがほぼ十分な情報を含んでいることが分かった．このため本研究では，これら仕様書を既存の仕様記述の枠組みに入れるために変換して扱うのではなく，エンジニアが使い慣れている状態遷移図を，できる限りそのままの形で扱うためのフレームワークを用意するというアプローチをとることとした．

2 モデリングと分析の手順

本研究における，Alloy を用いた「車体系」の車載電子システムのモデリング及び分析の手順を以下に示す．前準備として，対象とする状態遷移図の分析やその定義，イベントフレームワークの作成に大部分の時間をかけ，「ラゲージクローザシステム」のモデリングは (3) だけであった．

(1) 「車体系」製品の仕様書に用いられる独自の状態遷移図のセマンティクスを分析し，明確な定義を与える．
(2) (1) の分析と定義に基づき，状態遷移図をシミュレートするイベントフレームワークを定義する．この中には，イベントや状態，時間，スイッチ，内部変数，アクチュエータなどの構成要素，及びスイッチの読み取り方式（レベルトリガー方式とエッジトリガー方式）等が含まれる．
(3) (2) で作成したイベントフレームワークに基づいて，実際の対象となるアプリケーションを記述する．
(4) 構築したモデルのシミュレーション実行やセーフティプロパティの解析を行う．

3 状態遷移図の定義

本研究でモデル化の対象とする状態遷移図は，対象ドメインに特化し，独自の拡張がなされていた．従って，UML などの一般的な状態遷移図と異なる部分を明らかにし，その状態遷移図のセマンティクスを定義し直す必要があった．本節では，再定義した状態遷移図の概要について述べる．

3.1 状態

まず，状態におけるアクションは，実行振る舞いのみを対象とし，かつアクチュエータの制御のみである．次に，一般的な状態遷移図では，状態を階層的に定義したコンポジット状態や一つの状態において複数の状態を内部状態として持つ直交状態などもあるが，今回分析対象とした仕様書の記法には含まれていないため，単純状態のみを扱う．よって，ある時刻においてシステムの取りうる状態は必ず一つである．

3.2 遷移

ここで述べる遷移はある状態から他の状態への変化を意味する外部遷移である．遷移には一般的にその遷移の起こるきっかけとなるイベントを表すトリガ，遷移の条件を表すガード，遷移に伴うアクションを表す作用を記述する必要がある．本研究で扱う状態遷移図では，トリガをクロック[1]に対応付ける．また，非決定的な遷移を排除するために一つの状態からの複数の遷移を記述する場合には，その優先順位を記述する．優先順位は自然数で表し，その値が小さいものほど優先度が高いことを意味する．

4 イベントフレームワーク

イベントフレームワークは，「車体系」製品の仕様書に用いられる独自の状態遷移図をシミュレートし，共通で再利用可能なライブラリとして定義する．[3]

[1]ハードウェアクロックが一定周期で起こす割り込みではなく，タスクの一定周期を意味する．

で用いられているイベント記述のアプローチを拡張し，イベントと時間に加えて，状態，スイッチ，アクチュエータ，及び内部変数などの要素，更にはこれら要素間に成り立つべき制約を定義する．イベントを状態遷移図における遷移と置き換え，イベントの発生を遷移の発火と対応づける[2]．

時間を除くイベントフレームワークの構成要素（イベント，状態，アクチュエータ，スイッチ，内部変数）は，抽象シグニチャとしてそれぞれ定義し，実際のモデルではそれらを継承して個別に定義する．このように定義する事によって，その要素が本質的に持つ関係や制約といったアプリケーションに共通な部分を，イベントフレームワークとしてライブラリ化することができる．

4.1 時間

ここでは時間を離散的な全順序関係の並びとして表現するために，既存の ordering モジュールを使用して時間のシグニチャを定義する．

```
open util/ordering[Time] as time
sig Time {}
```

4.2 イベント

イベントは状態遷移図における遷移に該当する．まず *abstract* キーワードを用いて，以下のように抽象シグニチャとしてイベントを定義する．

```
abstract sig Event { pre, post: Time }
```

pre はイベントの発生前時刻を表す，イベントと時間の二項関係，*post* はイベントの発生後時刻を表す，イベントと時間の二項関係である．また上記の宣言において，発生前時刻と発生後時刻がそれぞれ1つしかないことを制約として記述している[3]．

4.3 状態

状態はイベントと同様に抽象シグニチャを用いて定義する．また，どの時刻にシステムがその状態にあるかを表現するために *exists* という関係を導入する．以下がその記述である．

```
abstract sig State { exists : set Time}
```

exists は多重度キーワードの *set* を用いて記述されており，実行系列においてシステムがその状態にあることに関して特に制限を設けないことを意味している．つまり，システムがその状態にあることがなくてもよいし，複数回その状態にあっていてもよいということを意味している．

4.4 時間・イベント・状態間の制約

本節ではシグニチャ宣言時の制約に加えて，状態遷移図をシミュレートするために時間・イベント・状態の三要素間で満たされるべき制約について述べる．以下にその制約の意味と Alloy による記述[4]を示す．

- 全ての時刻（最後を除く）において，その時刻が発生前時刻で次の時刻が発生後時刻となるようなイベントは高々一つである．

  ```
  all t: Time - last | lone e: Event |
    e.pre = t and e.post = t.next
  ```

  イベント（遷移）は，ある時刻において1つ発生するか，もしくは発生せずにその状態に留まるかのどちらかであることから，上記の制約が満たされるべきである．
- 全てのイベントの発生後時刻は発生前時刻の次の時刻である．

  ```
  all e: Event | e.post = next[e.pre]
  ```

  イベントの発生は，常に一単位時間で起きるべきで，複数単位時間以上をかけて起きることはない．これは，適用対象が周期的な処理を行うシステムであることと，状態遷移図の遷移のトリガが常にクロックであることから明らかである．状態遷移に複数単位時間はかからない．
- ある時刻においてシステムが取る状態は只一つである．

  ```
  all t: Time | one exists.t
  ```

  単純状態のみを扱うため，ある時刻においてシステムは一つの状態にあるはずである．
- イベントが発生しない限り状態は変化しない．

  ```
  all t: Time - last | no pre.t =>
    exists.t = exists.(t.next)
  ```

  状態遷移図において，遷移が発生しない限り状態の変化は起こらないことから，上記の制約は常に満たされるべきである．

[2]但し，イベントは発生条件を満たしても必ず起きるとは限らないが，本研究で扱う状態遷移図はガードが成り立てば必ず発火する（優先順位が一番高い場合）．

[3]シグニチャ宣言時に関係を定義する場合，多重度キーワードを用いて，関係の多重度を制約できる．今回のようにそのキーワードが省略されている場合は、自動的に多重度キーワード *one* が採用されることから，関係 *pre*，*post* は一つのイベントに対して一つの時間との関連を持つ関係となる．

[4]実際にはファクトとして定義される．図1を参照．

4.5 スイッチ

スイッチはシステムの外界にあり，システムの振る舞いと関係なく，ON・OFF が変化する非同期イベントとしてモデル化する．またセンサは，そのセンサ値が用いられる条件で述語抽象を行い，スイッチの一種として扱うことにする．例えば車速センサの場合，ある一定速度以上かどうかという条件で用いられることから，その条件が真の場合と偽の場合だけを考慮する．ある時刻のスイッチの ON・OFF は，スイッチと時間との二項関係 *on* を用いて表現する．その Alloy 記述を以下に示す．

```
abstract sig Switch { on : set Time }
```

4.5.1 スイッチの読み取り方式

スイッチの読み取り方式は，レベルトリガー方式とエッジトリガー方式の二通りがある．従って，その双方をモデル中で扱える必要がある．ここではその読み取り方式を抽象化し，フレームワーク上での意味を定義する．

まず，レベルトリガー方式とは，ある瞬間のスイッチの状態を単純に調べる方法である．ある時刻 t において，スイッチが ON の場合に真となる述語 *onLevel* と，OFF の場合に真となる述語 *offLevel* を以下のようにそれぞれ定義する．

```
pred onLevel[t: Time, s: Switch]{
   one s.on & t }
pred offLevel[t: Time, s: Switch]{
   no s.on & t }
```

次に，エッジトリガー方式とは，スイッチの ON・OFF の変化を調べる方法である．ある時刻 t と，その前の時刻 $t'(t.prev)$ に注目し，OFF から ON への変化を *onEdge*，ON から OFF への変化を *offEdge* として以下のように，それぞれ述語で記述する．

```
pred onEdge[t: Time, s: Switch]{
   one t & s.on
   no t.prev & s.on }
pred offEdge[t: Time, s: Switch]{
   no t & s.on
   one t.prev & s.on }
```

これらの述語は，各イベントの制約を記述する際に用いられる．

4.6 内部変数

本研究で扱う内部変数は，真と偽を表すフラグと考える．従って，以下のように *true* という関係を定義し，ある時刻において変数が真であることを表す．

```
abstract sig Variable { true : set Time }
```

また，対象とする状態遷移図において，遷移が発火する際のアクションでのみ内部変数は更新されることから，以下の制約[5]が成り立つべきである．

```
all t: Time - last | no pre.t =>
   true.t = true.(t.next)
```

もう一つ重要なポイントとして，スイッチがシステムの外界で非同期に振る舞うのに対して，内部変数はシステムの内部にあり，意図的に更新される．そこで，意図したイベントでのみ内部変数が更新されるようにフレームコンディション [4] という概念を用いて，内部変数に関して制約を記述する．具体的には，それぞれのイベントの制約として「変数が更新される」という制約と，それに加えて，もし更新されないのであれば「変数は変化しない」という制約を明示的に記述する．従って，イベントの制約を記述する際に用いる述語として，内部変数のセットを意味する *setVariable* とクリアを意味する *clearVaribale*，そして不変を意味する *unchanged* をフレームワーク内で以下のように定義しておく．

```
pred Event.setVariable[v: Variable]{
 one v.true & this.post }
pred Event.clearVariable[v: Variable]{
 no v.true & this.post }
pred Event.unchanged[field:univ->Time]{
 field.(this.pre) = field.(this.post) }
```

更に，内部変数はイベントのガードの記述にも用いられることから，その変数が時刻 t において真か判断する述語として，*isTrue* を以下の通り定義する．

```
pred isTrue[t: Time, v: Variable]{
 one v.true & t }
```

4.7 アクチュエータ

本研究ではアクチュエータの一つとしてモーターをモデル化する．モーターの振る舞いとして考えられるのは，正転出力，逆転出力，出力なしである．また，同時に正転出力と逆転出力にはならないという制約も含めて，以下の通りモーターを記述する．

```
abstract sig Motor {
   forward, reverse : set Time
}{ no forward & reverse }
```

イベントフレームワーク全体の Alloy 記述 (*ABS.als*[6]) を図 1 に示す．また，イベントフレームワーク全体の概略図を図 2 に示す．

[5] 実際にはファクトとして記述する．図 1 を参照．
[6] Automotive Body Systems

```
module ABS
open util/ordering[Time] as time

sig Time{}
abstract sig Event{
 pre, post: Time}
abstract sig Motor{
 forward, reverse: set Time
}{ no forward & reverse }
abstract sig Switch{on: set Time}
abstract sig Variable{true: set Time}
abstract sig State{exists: set Time}

fact EventFrameworkConstraints{
 all t: Time - last {
  lone e: Event | e.pre=t
    && e.post=t.next
  no pre.t => exists.t=exists.(t.next)
    && true.t=true.(t.next)
 }
 all e: Event | e.post = next[e.pre]
 all t: Time | one exists.t
}

pred onLevel[t: Time, s: Switch]{
 one s.on & t }
pred offLevel[t: Time, s: Switch]{
 no s.on & t }
pred onEdge[t: Time, s: Switch]{
 one t & s.on
 no t.prev & s.on }
pred offEdge[t: Time, s: Switch]{
 no t & s.on
 one t.prev & s.on }

pred Event.setVariable[v: Variable]{
 one v.true & this.post }
pred Event.clearVariable[v: Variable]{
 no v.true & this.post }
pred Event.unchanged[field:univ->Time]{
 field.(this.pre) = field.(this.post) }
pred isTrue[t: Time, v: Variable]{
 one v.true & t }
```

図 1: フレームワークの Alloy 記述（ABS.als）

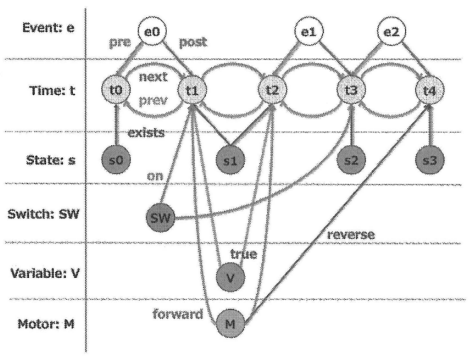

図 2: イベントフレームワーク全体の概略図

5 アプリケーションの記述

前節で述べたように，構成要素はシグニチャを継承して宣言する．その例を図 3 に示す．

```
/* モーターの宣言 */
one sig CloserMotor extends Motor {}
/* 各種スイッチの宣言 */
one sig SmartSW,... extends Switch {}
/* 変数の宣言 */
one sig OFlag extends Variable {}
/* 状態の宣言 */
one sig Init,... extends State {}
```

図 3: 構成要素の記述例

但し，イベントに関してはガードやアクションなどをそれぞれ記述する必要があり，フレームワークで定義した述語（スイッチの読み取りや内部変数の更新等）を用いて制約として記述する．図 4 に状態遷移図の例の一部を示す．この図では，システムが $Reverse\_Actuate$ 状態にあり，$PositionSW1 = OFF$ かつ $PositionSW2 = ON$ かつ $OPEN$ フラグ $= ON$ のとき，イベント $e18$ が発生する．そして次の時刻において，$OPEN$ フラグをクリアされており，かつシステムの状態が $Open\_Actuate$ となる．これをイベントフレームワークを用いて図 5 のように記述する．

今回の適用事例において，スコープ[7]に依存しないシグニチャとしては状態が 12，スイッチが 9 つ，内部変数が 1 つであった．また，イベントと時間の数は解析時に設定するスコープの数に依存するが，記述したイベントの数自体は 24 あった．

加えて，イベントフレームワークで記述していない，初期状態に関する制約やモーターの出力制御に関する制約，実時間制約に関する制約，また優先順位の制約などについても記述した．

6 モデルの解析

本研究では，複数の項目について解析を行った．大きく分けると，モデルのシミュレーション実行とセーフティプロパティの解析の 2 つに分けられる．ここでは，セーフティプロパティ「走行中にラゲージドアは開かない」について述べる．このプロパティを解析する為には「走行中」と「ラゲージドアが開

[7]解析を行う際に設定する範囲のことを指す．例えば，スコープに 15 を設定すると，モデル中の制約を踏まえてシグニチャの数を最大で 15 まで考慮して解析を行う．

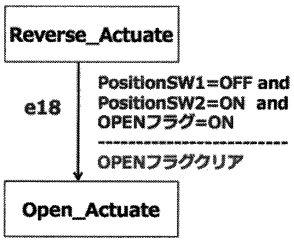

図 4: 状態遷移図の例の一部

```
sig e18 extends Event {}{
/* 事前条件 */
 pre.e18_preCond
/* 事後条件 */
 exists.post = Open_actuate
 OpenFlag.clearVariable
}
pred Time.e18_preCond []{
 exists.this = Reverse_actuate
 this.offLevel[PositionSW1]
 this.offLevel[PositionSW2]
 this.isTrue[OpenFlag] }
```

図 5: イベントの記述例

かない」をどのようにモデル中で表現すればよいかという点がポイントになってくる.

6.1 表明 1

まず表明 1 では,「走行中」を車速がある一定速度より大きい場合に該当すると考えた. 車速センサを表すスイッチ *SPD* を用いて,「走行中」を表す述語 *isDriving* を以下のように定義する.

```
pred Time.isDriving [] {
    this.onLevel[SPD] }
```

次に「ラゲージドアが開かない」について考えてみる. この状況は,「ラゲージドアが開く」という状況の否定と考える事が出来る. オープン作動に対応する状態 (*Open1, Open2, Open3*) のいずれかの状態にある場合に, ラゲージドアが開くと考えると, 述語 *isDriving* を用いて以下の通りに記述できる.

```
assert LuggageNeverOpened {
 all t: Time | t.isDriving =>
   no (Open1+Open2+Open3).exists & t
}
check LuggageNeverOpened for 10
```

この表明を解析すると, 反例が出力された. この反例を分析してみると, オープン作動の直前では停止していたが, オープン作動が始まった直後に急発進したシナリオではないかと考えられた. つまり, 走行中にオープン作動が行われる状況であり, 走行中にオープン作動が開始されるケースではない. 従って, 正しい表明を記述できていなかったことが明らかになった. そこで表明を再検討する.

6.2 表明 2

表明 2 では,「ラゲージドアが開かない」をオープン作動を開始するイベントが発生しないと解釈する. 仕様書からそれに該当する遷移をピックアップし, 走行中にそれらのイベントが発生しないことを表明として以下のように記述する.

```
assert LuggageNeverOpened2 {
 all t: Time | t.isDriving =>
  no (e2+e4+e5+e8+e10+e26+e29).pre & t
}
check LuggageNeverOpened2 for 8
```

この表明の解析を実行すると, 再び反例が出力され, クローズ作動時 (*Close1, Close2, Close3, Close4*) かつ走行中に, スマートスイッチが OFF から ON に変化することで, オープン作動が開始されるシナリオが発見された. この状況について分析してみると, まず走行中にスマートスイッチが押下される状況は考えないという外部環境の前提があるのではないかと考えられた. しかし, 議論・検討の結果,「クローズ作動時にユーザや物が挟まれた場合, ユーザがスマートスイッチを押すことを想定して, その場合は走行中でもドアをオープンさせる」という安全性を考慮しての仕様であることが明らかになった.

6.3 表明 2'

表目 2 の解析結果から, クローズ作動中状態 (*Close1, Close2, Close3, Close4*) においてスマートスイッチが押下されるケースを除くという前提を置く. 以下の通り表明 2' を定義する.

```
pred assumption [t: Time] {
 exists.t in (Close1+Close2+Close3+Close4)
   => not t.onEdge[SmartSW]
}
assert LuggageNeverOpened2' {
 all t: Time | t.isDriving && t.assumption
  => no (e2+e4+e5+e8+e10+e26+e29).pre & t
```

```
}
check LuggageNeverOpened2' for 8
```

この表明を解析すると，再度反例が発見された．この反例を分析してみると，クローズ作動時（*Close2, Close3, Close4*）かつ走行中に，何らかの要因でカーテシスイッチがOFFからONに変化し，ドアがオープンされるというシチュエーションであった．これも表明2'で発見されたシナリオと同様に，ユーザが何か挟まれた場合を考慮してドアをオープンさせるというシチュエーションだと考えられる．また，カーテシスイッチがONということは，ほぼドアが開いているとも考えることができる．従って，意図した振る舞いであり，このケースを除いて再度解析する必要がある．

6.4 表明2''

表目2'の解析結果から，クローズ作動時（*Close2, Close3, Close4*）にカーテシスイッチが押下されるケースを除くという前提を置く．以下の通り表明2''を定義する．

```
pred assumption [t: Time] {
 exists.t in (Close1+Close2+Close3+Close4)
   => not t.onEdge[SmartSW]
 exists.this in (Close2+Close3+Close4)
   => not this.onEdge[CurtesySW]
}
assert LuggageNeverOpened2'' {
 all t: Time | t.isDriving && t.assumption
  => no (e2+e4+e5+e8+e10+e26+e29).pre & t
}
check LuggageNeverOpened2'' for 8
```

この表明の解析を実行すると，再び反例が出力され，クローズ作動時（*Close4*）かつ走行中に，時間経過に関する条件が成り立ち，ドアが開くというシナリオが発見された．この状況について分析してみると，何か挟まってクローズが完了しない場合やスイッチのオフ故障が発生した場合のモーター焼き付きを防止するための意図通りの振る舞いであった．

6.5 表明2'''

表明2''の解析結果から，クローズ作動時（*Close4*）において時間経過に関する条件が成り立つケースを除くという前提を置く．以下の通り表明2'''を定義する．

```
pred assumption [t: Time] {
 exists.t in (Close1+Close2+Close3+Close4)
   => not t.onEdge[SmartSW]
 exists.this in (Close2+Close3+Close4)
   => not this.onEdge[CurtesySW]
 exists.this in Close4
   => not this.timeCondition[T2]
}
assert LuggageNeverOpened2''' {
 all t: Time | t.isDriving && t.assumption
  => no (e2+e4+e5+e8+e10+e26+e29).pre & t
}
check LuggageNeverOpened2''' for 15
```

この表明を解析した結果，反例は出力されなかった．よって，構築したモデルにおいてスコープ15以内で表明を違反するしシチュエーションがないことがわかった．

7 議論

事例適用を通して明らかになった事柄や手法の応用について議論する．また，その適用限界についても言及する．

7.1 制約の顕在化

6.2節において，スマートスイッチが走行中に押下されないという前提があるのではないかと考えられた．実際は少し異なっていたが，このように外部環境に前提を置いた上でソフトウェアの仕様が考えられている場合があったり，仕様書に登場しないドメイン固有の制約が暗黙裏となっているケースがある．そういった制約をシミュレーション実行やプロパティの解析を通して明らかにすることができると考えられる．

7.2 安全性の向上

安全性分析は仕様策定の段階で行われることが一般的だが，今回は仕様書から上流へリバースすることによって，仕様書に埋もれた要求仕様を明らかにすることができた．また，重要なポイントとして，安全性分析によって発見された状況は，ソフトウェアの設計者よりむしろシステム設計者に問い合わせるべき事柄であったことが挙げられる．従って，システム設計における各専門領域の責務が曖昧な部分をも明らかにでき，それを明確にすることによって，より安全性を高めることができると考えられる．

7.3 他分野への応用

本研究では車載電子システムの特に車体系を対象とし，実例を想定した仕様書に基づいて安全性について分析を行った．ここでいう安全性とは，システム全体の横断的な関心事であると考えられる．例えば，エンタープライズ系のシステム開発について考えてみると，セキュリティなどがその横断的な関心事であると言える．よって，セキュリティに注目してシステムのアーキテクチャを分析する際に，本研究で行った解析手法が応用できるのではないかと考えられる．

7.4 適用限界

Alloy では解析が自動で行える反面，予め設定したスコープ以内でしか解析を行うことができない．つまり，それ以上のスコープで解析を行うと、表明を違反するケースが存在する可能性があり，完全性が保証されない．また，時間を正確に扱うことが難しく，本事例においても，時間経過に関して条件が成り立つ場合と成り立たない場合をランダムに与えているだけである．よって，Alloy のような軽量形式手法では正しいことを保証することは難しい．それよりも，特定の性質に注目してシナリオを抽出し，見つかったシナリオをユーザが妥当か検討するというアプローチが向いているのではないかと考えられる．今回の事例適用では，安全性に注目してシナリオを抽出し，その妥当性の確認を行うことができた．

8 おわりに

本稿では，車載電子システムの信頼性や安全性を高めるために，Alloy をリバースエンジニアリングの道具として利用し，安全性分析の支援を行った．

実際に，実例を想定した仕様書に基づいて Alloy で記述，解析を行うことによって，表明を違反するケースを複数発見できた．安全性に関して問題はなかったが，ツールによる網羅的かつ自動的な解析によって，思いつきにくい，または漏れ易い状況の検出が可能であるという知見が得られた．同時に，要求仕様やそれに基づくシステム設計の曖昧な部分が明らかになり，それぞれの専門領域の責務を改めて明確にすることができるのではないかと考えられる．

今後は，自動変換に取り組むつもりである．具体的には，MetaEdit+[5] を用いて，再定義した状態遷移図のモデリング環境を構築し，その環境に基づいて記述されたモデルから Alloy モデルの自動生成を試みる．仕様の全てを自動変換するのは難しいが，部分的な変換は可能だと思われる．今後この手法が有効かどうか検討していきたい．

謝辞

本研究の一部は，アイシン精機株式会社様よりの資金提供を含む，財団法人北九州産業学術推進機構産学連携研究開発事業助成金 (マッチングファンド) を活用して，北九州市立大学，九州大学，九州工業大学，アイシン精機株式会社で実施した共同研究テーマ：「機能安全に対応可能な車載システムの安全設計ガイドと形式検証に関する研究」の一部として行ったものである．

参考文献

[1] Daniel Jackson. Alloy: a lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 11, No. 2, pp. 256-290, 2002.

[2] Daniel Jackson, Ilya Shlyakhter, and Manu Sridharan. A micromodularity mechanism. In ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 62-73, New York, NY, USA, 2001. ACM.

[3] Daniel Jackson. Software Abstractions: Logic, Language, Analysis. The MIT Press, 2006.

[4] Alex K. Simpson. The Proof Theory and Semantics of Intuitionistic Modal Logic. PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics., 1994

[5] MetaCase. Metaedit+ website: http://www.metacase.com/ja/mep/.