Vol.2011-SE-171 No.10
2011/3/14

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

柳 翔太[†1] 小飼 敬[†2] 上田賀一[†1]
大久保 訓[†3] 高橋勇喜[†3] 中野利彦[†3]

ソフトウェアの品質を保証するために，ソフトウェア検証を行う必要性が高まっている．そこで，振舞いを網羅的に検証するモデル検査を開発現場に導入することを考える．現在，情報制御システム記述モデルに対してモデル検査を実行する手法が提案されている．しかし，検証すべき事項の記述方法はまとまっていない．そのため，本研究では検証項目記述の定義を行い，検証項目の設計手法について検討した．また，モデル検査で検証する性質を表す LTL(Linear Temporal Logic) 式に検証項目を変換するルールを定義した．モデル検査ツールを用いてこれらの有効性を確認した．

A Verification Item Description of Information Control System Descriptive Model and Confirmation by SPIN

Shota Yanagi,[†1] Kei Kogai,[†2] Yoshikazu Ueda,[†1]
Satoshi Okubo,[†3] Yuki Takahashi[†3]
and Toshihiko Nakano[†3]

The necessity to perform software verification is growing in order to guarantee the quality of software. Therefore we think that we introduce a model checking to verify the behavior cyclopaedically into the development field. A technique to execute a model checking for information control system descriptive model is suggested now. However, the description method of the matters which should be verified is not completed. Therefore we defined the verification item description in this study and examined the design technique of the verification item. In addition, we defined a rule to convert a verification item into LTL(Linear Temporal Logic) formula which expressed a property to verify by model checking. We confirmed these effective by using a model checking tool.

1. はじめに

近年，システムの品質に対しての関心が高まっている．品質を高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，システムが大規模かつ複雑になるにつれて，従来の手法では十分に対応しきれなくなっている．設計レビューは人手による確認作業が基本なので，大規模システムにおいては人手によるレビューでは限界がある．また，テスト・デバッグについても，ツールによる自動化が進んでいるが，テストデータやテストケースの作成は基本的に人手で行っている[1)]．システムの振舞いを網羅するためには多くのコストがかかる．そこで，振舞いを網羅的に検証するモデル検査を開発現場に導入することを考える．著者らは，情報制御システムの制御規則をルールベースで表現する記述言語 RASYS を対象とし，RASYS で記述したモデルに対する検証支援手法を研究してきた．RASYS モデルに対する検証の前段階として，RASYS の記述仕様に対する検証を形式手法 Alloy[2)] を用いて行った．検証を MDA のメタ階層に対応するように分割することで，大規模なモデルでも実用的な時間での検証を可能とした[3)]．また，RASYS モデル自体の検証支援手法も提案している．モデル検査ツール SPIN[4)] で使用する言語 PROMELA に変換するルールを定義することで，RASYS モデルに対してモデル検査を行うことを可能にした[5)]．これにより，RASYS モデルの検証を機械的に行うことは可能になった．RASYS に関して，これらの先行研究が進められているが，まだ解決していない問題がある．問題の 1 つに RASYS に検証すべき事項をどのように記述するべきか定義が無いことが挙げられる．SPIN によって検証を行う際に，どの事項についてどのような性質を検証したいか指定する必要がある．しかし，RASYS には検証すべき事項を記述する記法が存在しないため，検証の実行者はそれぞれの記述方法で検証事項を記述することになる．記述方法が統一されていなければ，RASYS モデルから検証事項を機械的に抽出することはできない．

そこで，本研究では RASYS に対して，検証すべき事項を記述する記法として検証項目記述を定義し追加する．安全性や到達性等の特性検証すべき事項の記述方法を定義することで，それぞれの特性毎に検証事項を記述できるようになる．また，SPIN では検証内容を

†1 茨城大学
Ibaraki University
†2 茨城工業高等専門学校
Ibaraki National College of Technology
†3 株式会社 日立製作所
Hitachi Ltd.

LTL 式と呼ばれる線形時相論理式で表現するため，記述した検証事項はそれぞれの検証特性に応じた LTL 式へと変換する必要がある．この変換ルールに関しても定義する．これらの定義を行うことにより，RASYS モデルから機械的に検証事項を抽出することが可能になり，検証の自動化が進むことになる．

今回，我々は検証項目記述の定義をし，LTL 式への変換ルールの定義を行った．そして，有効性を SPIN を用いて検証したところ，有効性を確認することができた．

2. 基礎となる関連知識

本研究で対象とする情報制御システム言語 RASYS とモデル検査ツール SPIN を説明する．また，検証の手順についても説明する．

2.1 情報制御システム言語 RASYS

今回対象とする RASYS は，情報制御システムを対象として開発されているモデル記述言語である．RASYS はシステムの制御規則をルールベースで記述する．システムの振舞いや実行条件などのルール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システム全体を表現している．

2.2 モデルの構成要素

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
  システムを構成する設備を表す．設備の状態等の属性とその属性値を持つ．オブジェクトは設備に対応するインスタンスとして実装され，個々の設備とインスタンスは 1 対 1 に対応する．設備の状態は基本的に離散化された値として属性が持つ．
- アクタ
  オブジェクトで定義した設備が持つ属性値の参照や更新などの操作を表す．アクタはシステムの振舞い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，次のアクタを実行しない．

2.2.1 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，5 種類のモデル記法でシステムの構成要素を記述する．その概要を図 1 に示す．RASYS では，オブジェクト，アクタ，アクタシナリオが以下の 5 種類の記法を利用してモデル化される．本研究で定義する検証項目はこの 5 種類の記法に追加する形になる．

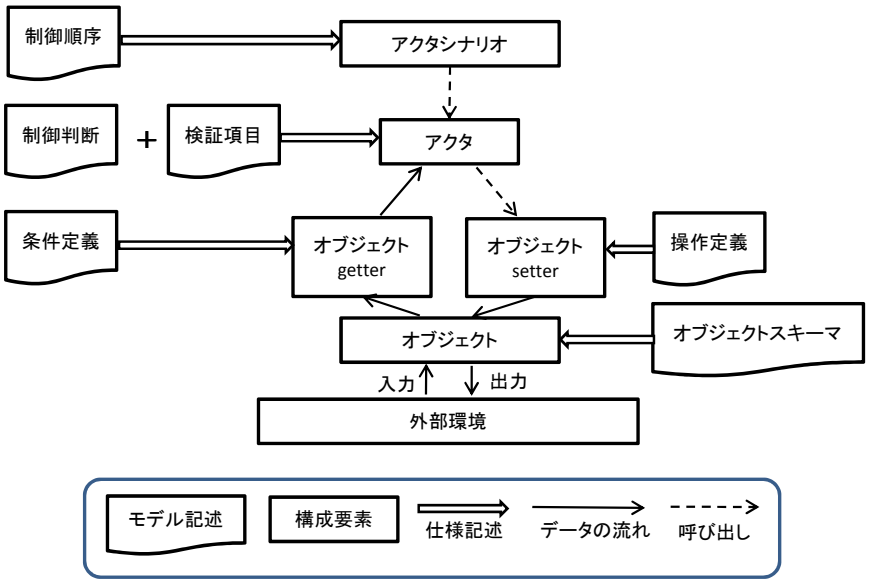

図 1 システムの構成要素とモデル記法の対応

(1) 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．

(2) 制御判断
制御判断の例を図 2 に示す．制御判断は制御対象となるオブジェクトと実行条件，実行周期を規定する部分と，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操作項目の組で構成されている．制御条件は，オブジェクトの属性値を組み合わせて表す．複数のオブジェクトの属性値を組み合わせて条件とすることができる．実行項目には，対象オブジェクトに対して実行する操作を記述する．この操作は操作定義によって定義されているものである．制御判断は実行周期に従い，表の上から順番に制御条件を確認し，条件が一致した場合は実行項目に記述されている操作を行う．

(3) 条件定義

ルール				
No	条件		実行項目	
	オブジェクト A	オブジェクト B	オブジェクト C	オブジェクト D
	属性 A	属性 B	-	-
1	属性値 A	属性値 B	実行項目 A	-
2	属性値 C	属性値 D	実行項目 B	実行項目 C

図 2 制御判断例

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．属性の属性値は 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み合わ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．

(4) 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操作項目で表される．条件はオブジェクトの属性値の組合せで表現される．実行項目は属性値を代入する操作や別の操作を指定する場合がある．

(5) オブジェクトスキーマ
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．

2.3 モデル検査ツール SPIN

本研究で使用するモデル検査ツール SPIN[4] について説明する．SPIN は G. J. Holzmann らが開発した，オートマトンをベースとしたモデル検査ツールである．SPIN による検証は次の手順で行われる．まず，対象とするシステムの設計モデルを PROMELA と呼ばれる仕様記述言語に変換する．次に，検証する性質を LTL 式で表す．最後に，LTL 式で表した性質をモデルが満たしているか，SPIN を用いて検証する．SPIN はモデルが取りうる可能性がある動作を網羅的に探索して，指定した性質が成り立つことを確認する．モデルが指定した性質を満たさない場合は，その反例が示される．

SPIN で使用する PROMELA は，C 言語に似た文法の言語であり，PROMELA のモデルはプロセスの集合で構成される．しかし，C 言語程の自由度は無い．例えば，使用できるデータ型は Int 型などの基本的な型や列挙型のみである．また，ポインタも使用することができない．そのため，対象システムによっては，PROMELA の記述能力ではモデル化できない場合がある．

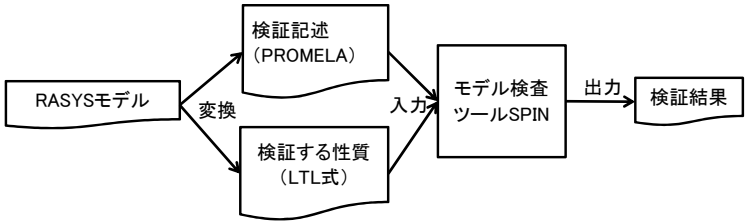

図 3 RASYS モデルの検証の流れ

2.4 検 証 手 法

RASYS に対する検証の流れを図 3 に示す．まず，RASYS によって記述されたシステムの振舞いを PROMELA 記述に変換する．次に，検証項目記述を LTL 式へ変換する．最後に PROMELA 記述と LTL 式を SPIN に入力することで検証結果が出力される．これらのプロセスの内，RASYS モデルの PROMELA への変換手法については先行研究[5] において定義している．そのため，本研究で検証項目記述記法の定義を行い，LTL 式への変換ルールを定義することで検証手順を全て確立することができる．

3. 検証項目記述

検証項目記述の定義から設計手法，LTL への変換について説明する．検証項目は，対象となる RASYS モデルに対して，検証したい項目を定義するものである．

3.1 検証種別の定義

SPIN を用いた検証を考慮した結果，以下の 4 種類の検証種別を定義することにした．検証は “状態” を基に行うため，検証種別も “状態” を基に定義している．

- 到達可能性
  初期状態から，ある特定の状態へと少なくとも一回到達することを検証できる．
- 進行性
  特定の状態に無限に訪れることを検証できる．
- 活性

表 1 検証項目表

ルール					
No	開始状態		到達状態		検証種別
	オブジェクト X	オブジェクト Y	オブジェクト W	オブジェクト Z	
	属性 A	属性 B	属性 C	属性 D	
1	-	-	属性値 c1	-	到達可能性
2	-	-	属性値 c2	属性値 d1	進行性
3	属性値 a1	属性値 b1	属性値 c3	-	活性
4	-	-	属性値 c4	属性値 d2	安全性

ある状態になれば，いつかは特定の状態に到達することを検証できる．

- 安全性
  成立しては困る状態が成立しないことを検証できる．

各種別を用いた具体的な検証項目は 3.3 節で説明する．

3.2 検証項目表の定義

定義した検証種別を全て表現できる検証項目表を定義することにする．検証種別に使われている “状態” という言葉は 2 種類に分けられる．1 つは前提となる状態であり，もう 1 つは最終的に到達する状態である．これら 2 つの状態を組み合せることで検証項目を作成することができる．そのため，検証項目表には 2 種類の状態を記述できる 2 つのカラムが必要となる．また，状態は複数の属性値の組み合わせで表現されるため，各状態カラムは複数の属性値を記述可能にする．加えて，制御判断に体裁を関連づけることにする．これは，設計者が制御判断の設計と同程度の能力や労力で検証できることを考えているためである．これらの，条件を基に定義した検証項目表を表 1 に示す．1 行が 1 つの検証項目を表している．前提となる状態を開始状態と命名し，最終的に到達する状態を到達状態と命名する．表 1 では開始状態，到達状態の両方とも 2 つの属性値で構成されているが，検証したい状態に応じて増減させることは可能である．検証種別のカラムには，開始状態と到達状態の組み合わせに対して検証する種別を記述する．この検証種別を基に 3.4 節で説明する LTL への検証項目の変換が行われる．

3.3 検証項目の設計

検証項目の設計は検証種別ごとに行われる．検証種別ごとに検証できる性質が異なっているためである．各検証種別の設計観点は以下のようになる．

- 到達可能性
  初期化等のシステム起動時から最低 1 度は到達しなければならない状態を挙げる．
- 進行性
  どのような遷移をしても必ず到達しなくてはならない状態を挙げる．例として，例外処理を行ったあとに必ず通常処理に復帰するなどが挙げられる．
- 活性
  特定の状態になるといつかは必ず到達して欲しい状態を挙げる．例として，例外が発生すると例外処理を必ず行うなどが挙げられる．
- 安全性
  成立しては困る状態 (同時に成立しては困る属性値の組合せ) を挙げる．例として，信号機が全て同時に青になるなどの状態が挙げられる．

車用信号機を 4 つ持つ交差点システムを対象に上記の観点に基づいた検証項目を設計してみる．例として，安全性について検証項目を設計する．交差点にとって成立しては困ることを信号機の動作に焦点を当てて考えることにする．全ての信号機が同時に青になってしまうことは安全上良いことではない．よって，安全性に属する検証項目とする．しかし，言葉で定義しただけなのでこのままでは検証項目表を作成することができない．そのため，“全ての信号機が同時に青になる” という状態がどのようなオブジェクトと属性値の組み合わせで構成されているか把握する必要がある．対象とする交差点には車用信号機が 4 つあるので，それぞれの信号機の出力が青になると “全ての信号機が同時に青になる” という状態になる．これを，検証項目表に当てはめると表 2 になる．

表 2 信号機の検証項目

ルール						
No	開始状態	到達状態				検証種別
	-	車用信号機A	車用信号機B	車用信号機C	車用信号機 D	
	-	出力	出力	出力	出力	
1	-	青	青	青	青	安全性

3.4 検証項目の LTL 式への変換

検証項目表を基に LTL 式へ変換する手順について説明する．

LTL 式では，命題論理式に時相演算子と呼ばれる演算子をつけて修飾する．いくつかある演算子の中で，本研究で使用する演算子は “□” と “◇” である．また，命題論理式に使用する演算子は “⇒” と “!” である．それぞれの演算子の意味は以下の通りである．

- □:「常に」という意味を表す．

命題変数 p が「常に」真となるとき□ p は真となる．命題変数 p の真偽値が途中で偽になると□ p は偽になる．

- ◇:「いつかは」という意味を表す．
  命題変数 p が「いつかは」真となるとき◇ p は真となる．命題変数 p がずっと偽であった場合◇ p は偽になる．
- ◇:「否定」という意味を表す．
  命題変数 p が偽のとき!p は真となる．
- ⇒:⇒の左側の処理が完了したとき，右側の処理を行うことを意味している．
  p ⇒ q の場合は，命題変数 p が真となるとき命題変数 q の真偽を判断する．

検証種別と LTL 式の対応関係を表 3 に示す．この対応関係に沿って変換が行われる．

表 3 検証種別と LTL 式の対応関係

検証種別	LTL 式
到達可能性	◇ (到達状態)
進行性	□ (◇ (到達状態))
活性	□ (開始状態 ⇒◇ (到達状態))
安全性	□!(到達状態)

変換手順は以下の通りである．

- 開始状態と到達状態の値に対して順番に検証名として「`l + _n + _m`」を割り当てる．
  (n は行番号であり m は左から数えて何番目の属性値であるか表している)
- LTL 式を作成する際に検証名を使用して作成する．
- 検証項目表の検証種別に従って対応する LTL 式を作成する．
- 開始状態と到達状態に複数の列がある場合は「`&&`」でつなぎ LTL 式を作成する．
- PROMELA 記述に検証名と値の対応定義を`#define` を用いて追加する．

表 1 を基にした LTL 式への変換例を以下に示す．

```
No1 到達可能性:<>(l1_3)
No2 進行性:[](<>(l2_3 && l2_4))
No3 活性:[]((l3_1 && l3_2)  -> <>l3_3)
No4 安全性:[]!(l4_3 && l4_4)
```

これらの変換手順を用いて表 2 を変換すると，次のようになる．

```
[]!(l1_2 && l1_3 && l1_4 && l1_5)
```

作成した LTL 式を用いて検証するときは LTL 式の否定をとって検証する．これは，LTL 式を用いた検証では LTL 式が成立するときにエラーが報告されるためである．

4. 適用実験と考察

定義した検証項目記述の有効性を確かめるために RASYS によってモデル化した 2 つのシステムを対象に適用実験を行った．対象とする RASYS モデルを PROMELA 記述に変換し，設計した検証項目を検証種別毎に検証できるか実験した．

4.1 信号機制御システムの検証

4.1.1 システムの概要

対象とするモデルは信号機制御システムをモデル化したものである．システムの構成を図 4 に示す．このシステムは，丁字路に設置している信号機の出力制御を行うことを目的としたシステムである．丁字路内に設置してある信号機を，歩行者ボタンや車用センサ，時間帯に応じて制御を行う．システムはオブジェクトとして車用信号機を 3 つ，歩行者用信号機を 1 つ，車用センサを 1 つ，歩行者用ボタンを 1 つ，タイマを 1 つ持っている．信号機の動作はタイマによって時間帯を判断し，深夜帯であれば深夜動作を行い，それ以外は通常動作を行う．

モデルの規模は属性が 16 個、属性値が 21 個である．また，制御判断は条件の属性が 7 個，実行項目が 8 個、行数が 18 行である．これは，制御判断の規模としては標準である．

4.1.2 検証項目の設計

信号機制御システムに対して検証すべき項目を設計した．検証する検証種別は到達可能性，活性，安全性の 3 つとした．到達可能性の観点に沿った検証項目として,「深夜動作に移行する」ことを挙げる．活性の観点に沿った検証項目としては,「深夜動作に移行したあと、通常動作に移行する」ことを挙げる．安全性に関しては,「全ての信号機が同時に青になる」ことを挙げる．設計した検証項目表を表 4 に示す．活性と安全性の検証は反例が出るように意図的にバグを埋め込んだ．バグの内容は，活性の場合は制御判断に対して深夜稼動から通常稼動に移行する条件を削除したことである．また，安全性の場合は制御判断に全ての信号機が青になる条件を追加したことである．到達可能性に反例が無く，活性と安全性に反例があれば問題なく検証ができたと判断できる．

Vol.2011-SE-171 No.10
2011/3/14

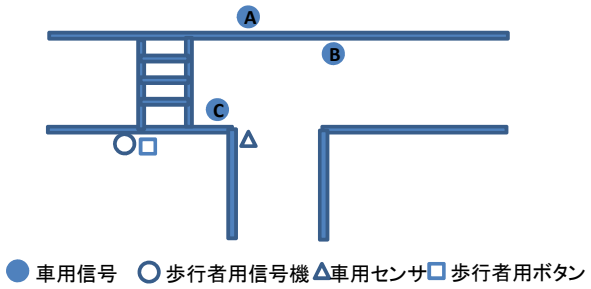

図 4 信号機制御システムの概要図

表 4 信号機制御システム検証項目

ルール							
No	開始状態	到達状態					検証種別
	T字路 稼動モード	T字路 稼動モード	車用信号機A 出力	車用信号機B 出力	車用信号機C 出力	歩行者用信号機A 出力	
1	-	深夜稼動	-	-	-	-	到達可能性
2	深夜稼動	通常稼動	-	-	-	-	活性
3	-	-	青	青	青	青	安全性

表 5 信号機制御システムの検証項目と LTL 式の対応関係

検証項目 No	検証種別	LTL 式
1	到達可能性	`<>(l1_2)`
2	活性	`[](l2_1 -> <>l2_2)`
3	安全性	`[]!(l3_3 && l3_4 && l3_5 && l3_6)`

4.1.3 検証項目の変換

検証を実行するために設計した検証項目を LTL 式へと変換する．変換ルールに従って変換した結果を表 5 に示す．作成した LTL 式に使用した検証名は RASYS モデルを変換した PROMELA 記述と対応がとれていない．そのため，検証名と PROMELA 記述内の値との対応関係を PROMELA 記述に追加する必要がある．対応関係を追加した結果を図 5 に示す．オブジェクトスキーマから変換した属性と属性値は mtype 型（列挙型）で定義している．属性の属性値と検証名の対応関係を PROMELA 記述では`#define` 用いて定義している．

4.1.4 検 証 結 果

検証結果を表 6 に示す．到達可能性の検証では反例が出力されなかった．これは，想定通りの出力結果である．活性と安全性の検証では反例が出力された．この結果もまた，想定通りの出力結果である．また，出力された反例を分析した結果，意図的にバグを埋め込んだ箇所が原因でエラーが起きていることを確認できた．

検証した全ての検証項目の検証結果が想定通りであったため，全ての検証項目に対して正しく検証できた言える．

```
#define l1_2 TShapedRoad_OperationMode == MidnightMode
#define l2_1 TShapedRoad_OperationMode == MidnightMode
#define l2_2 TShapedRoad_OperationMode == UsuallyMode
#define l3_3 TShapedRoad_CarSignalA_Output == Blue
#define l3_4 TShapedRoad_CarSignalB_Output == Blue
#define l3_5 TShapedRoad_CarSignalC_Output == Blue
#define l3_6 TShapedRoad_WalkerSignalA_Output == Blue

mtype = {Red,Yellow,Blue,UsuallyMode,MidnightMode};

mtype TShapedRoad_CarSignalA_Output = Red,
      TShapedRoad_CarSignalB_Output = Red,
      TShapedRoad_CarSignalC_Output = Red,
      TShapedRoad_WalkerSignalA_Output = Red,
      TShapedRoad_OperationMode = UsuallyMode;
```

図 5 信号機制御システムの検証名と属性値の対応関係の記述

表 6 信号機制御システム検証結果

検証項目 No	作成状態数	探索遷移数	反例	検証時間
1	667	667	なし	0.245 秒
2	867	868	あり	0.115 秒
3	447	447	あり	0.971 秒

4.2 トンネル換気制御システムの検証

4.2.1 システムの概要

本システムは，道路に設置されているトンネル内の換気制御を行うことを目的としたシス

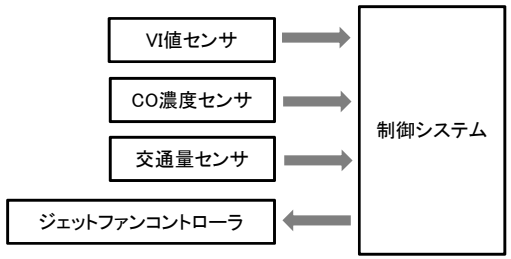

図 6 トンネル換気制御システムの概要図

表 7 トンネル換気制御システム検証項目

ルール				
No	開始状態	到達状態		検証種別
	-	トンネル	トンネル	
	-	換気ファン出力更新要求	トンネル状態	
1	-	要求あり	-	到達可能性
2	-	-	汚れている	進行性

テムである．トンネル内に設置した各種センサの値を基に換気装置を運転し，VI 値 (煤煙透過率)，CO 濃度を法令で定められた基準値を超えないようにする．システムの構成を図 6 に示す．システムはオブジェクトとしてセンサ類 (VI 値センサ，CO 濃度センサ，交通量センサ) とジェットファンコントローラを持っている．

モデルの規模は属性が 16 個、属性値が 34 個である．また，制御判断は条件の属性が 1 個，実行項目が 1 個、行数が 2 行である．これは，制御判断の規模としては比較的小規模である．

4.2.2 検証項目の設計

トンネル換気制御システムに対して検証すべき項目を設計した．到達可能性，進行性について検証項目を設計した．到達可能性の観点に沿った検証項目として,「換気量を変更する要求が出る」ことを挙げる．進行性の観点に沿った検証項目としては,「トンネルの状態が汚れているになる」ことを挙げる．設計した検証項目表を表 7 に示す．到達可能性の検証は反例が出るように意図的にバグを埋め込んだ．バグの内容は，換気ファン出力更新要求の値を定義する条件定義に対して，要求ありになる条件を削除したことである．進行性に反例が無く，到達可能性に反例があれば問題なく検証ができたと判断できる．

表 8 トンネル換気制御システムの検証項目と LTL 式の対応関係

検証項目 No	検証種別	LTL 式
1	到達可能性	<>(l1_2)
2	進行性	[](<>l2_3)

```
#define l1_2 TNL_FunUpReq == exit
#define l2_3 TNL_TNLState == Dirty

mtype = {VeryDirty,Dirty,Clean,exit,none};

mtype TNL_TNLState = Clean,
      TNL_FunUpReq = none;
```

図 7 トンネル換気制御システムの検証名と属性値の対応関係の記述

表 9 トンネル換気制御システム検証結果

検証項目 No	作成状態数	探索遷移数	反例	検証時間
1	1601	1601	あり	0.002
2	10016	13479	なし	0.35

4.2.3 検証項目の変換

検証を実行するために設計した検証項目を LTL 式へと変換する．変換ルールに従って変換した結果を表 8 に示す．PROMELA 記述に対応関係を追加した結果を図 7 に示す．

4.2.4 検 証 結 果

検証結果を表 9 に示す．進行性の検証では反例が出力されなかった．これは，想定通りの出力結果である．到達可能性の検証では反例が出力された．この結果もまた，想定通りの出力結果である．また信号機制御システムと同様に，出力された反例を分析した結果，意図的にバグを埋め込んだ箇所が原因でエラーが起きていることを確認できた．

検証した全ての検証項目の検証結果が想定通りであったため，全ての検証項目に対して正しく検証できた言える．

4.3 考　　察

今回 4 種類の検証種別を定義し，それぞれの検証種別毎に検証項目の設計を行い検証することができた．そのため，定義した検証種別を基に検証することは問題ないと考える．今

回定義した 4 種類の検証種別は RASYS モデルの検証を行う上で必要になると考えたものである．しかし，適用実験を行ったことで必要の無い検証種別があるかもしれないと考えた．その検証種別は活性である．通常のシステムではある処理を行った後に必ず期待する処理が行われるとは限らない．そのため，活性のような検証が必要になるだろう．しかし，RASYS はルールベースでシステムの制御規則を表現している．全ての処理が現在の状態が条件に一致するか否かで実行の可否を判断する．このような性質があるため，活性の検証をしなくても到達可能性や進行性で条件の状態に到達することを検証するだけで問題ないと考える．あくまで，今回対象としたモデルに関して考えたとき活性は必要ないと思われるが，モデルの中には活性が必要な場合もあるかもしれない．今後それぞれの検証種別の使い道について検討することが必要である．

今回対象とした 2 つのシステムに対する検証結果の状態数を比較する．制御判断の規模は信号機制御システムの方が大きかった．しかし，作成状態数や探索遷移数はトンネル換気制御システムの方が多い結果になった．これは，状態数は制御判断の複雑さには関係なく，モデル全体の属性と属性値の数のみに依存していることが考えられる．そのため，状態数を減らすためには制御判断を分割などして縮小するのではなく，モデル全体の属性と属性値を減らす工夫をする必要があると考える．

検証項目をどの程度の属性の組み合わせで表現するか考える．組み合わせる属性数が多くなればその分だけ検証時間が長くなる．これは，探索しなければならない状態数が増加するためである．現在想定している属性数は多くても 10 程度である．この程度であれば今回対象とした RASYS モデルに対して検証を実行することができた．しかし，あまり属性数を多くすることは検証時間増加の点から好ましくない．そのため，検証項目を表現するために多くの属性が必要になる場合は，検証項目を表現したい “状態” ごとに分割し，それぞれの状態に到達することを検証するなどの工夫を施すことが妥当であると考える．

検証自動化の例としては小池[6] の研究が挙げられる．小池は，状態遷移表と付加的な表から制約条件まで含む検査プログラムを自動生成するモデル検査支援環境を構築した．我々の研究と目指すべき方向性は似ているが異なる部分もある．小池の研究は設計段階の状態遷移表を対象とし検証を行っている．そのため，設計の品質を保証しても実装段階での品質が保証されていない．一方，我々が研究対象としている RASYS は設計したモデルから直接ソースコードが抽出される．加えて，RASYS モデルの実行系に合わせて PROMELA 言語に変換している．そのため，モデルの検証を行うことで実際に動作するプログラムの品質を保証することが可能となる．

5. おわりに

今回検証項目記述を定義したことで RASYS の記法拡張を行った．また，検証項目記述から LTL 式への変換ルールを定義したことで，LTL 式の知識を持たなくても，モデル設計者が検証を行うことができるようになった．定義した検証項目の有効性を確認するために適用実験を行った結果，問題なく検証できたため有効性を確認できた．

今後の課題は以下の通りである．

- 検証項目記述から LTL 式への自動変換ツールの開発
  現在は変換を手動で行っている．しかし，規模が大きくなると手動ではコストがかかりすぎるため自動化する必要がある．
- 検証種別の追加現在の検証種別は 4 種類である．これは，LTL 式で表現できるものの中から RASYS に適していると考えたものである．しかし，LTL 式ではより複雑な表現もできるため，他にも RASYS に適した検証種別を定義できる可能性がある．

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) A Guide to Alloy,
`http://www.doc.ic.ac.uk/project/examples/2007/271j/suprema_on_alloy/Web/.`
3) 小飼敬，上田賀一，大久保訓，高橋勇喜，中野利彦 : Alloy を利用した情報制御システム記述言語の仕様検証の実用化，ソフトウェア工学の基礎 XVI，pp.259-266，近代科学社 (2009).
4) Holzmann, G. J. : The Spin Model Checker:Primer and Reference Manual, Addison-Wesley，(2004).
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会第 27 回大会，D-2 (2010) .
6) 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，EMB，組込みシステム Vol.2008，No.116，pp.91–96(2008).