National Institute of
Advanced Industrial Science
and Technology
AIST

AIST-PS-2008-004

LTL モデル検査のための図示記法

小池憲史 *　吉田聡**　大崎人士**

*矢崎総業株式会社 技術研究所

**独立行政法人 産業技術総合研究所

システム検証研究センター

算譜科学研究速報

Programming Science
Technical Report

Research Center
for Verification and Semantics

独立行政法人 産業技術総合研究所 システム検証研究センター 発行
Published by Research Center for Verification and Semantics
National Institute of Advanced Industrial Science and Technology

LTL モデル検査の為の図示記法

Diagramic Notation for LTL Model Checking

小池 憲史[*] **吉田 聡**[†] **大崎 人士**[‡]

あらまし 有限モデル検査理論に基く既存のモデル検査ツールを用いてシステムの検証を行う場合，モデルの記述言語の習得以上に，検査式を表現する為の様相表現の習得が困難な事が多い．モデル検査技術を，ソフトウェアの開発現場に導入しようとする場合，特にこの現象は顕著になる．ソフトウェアの開発現場へ効果的にモデル検査技術を導入する為に，時相論理による様相表現の代わりに，独自に考案した図的な表現形式を用いる検査の試みを本論文では紹介する．我々が提案する 図示記法 (diagramic notation) は，特別な知識を必要としない様相表現，誰もが受け入れられる直感的な表現，行いたい検査内容を表現する表現能力，をそなえる事を目標として考案された検査式表現である．図示記法では，視点，実行列，分岐という図的な部品の組合せと，命題の配置具合により，検査したい性質の視覚的なイメージを表現する．また，与えられた検査項目から従来の様相表現で翻訳した検査式を，さらに図示記法で翻訳する事により，検査式の正しさを視覚的に認識できる利点もある．

Summary. When using a finite model checker, it is sometimes harder to acquire the skill to represent system properties in a certain modal logic, such as LTL, rather than to learn the modeling language. In case that the model checking technologies are introduced in *real* software development scenes, this problem becomes remarkable. In order for software engineers to describe accurate properties even not having enough mathematical background, more intuitive and reasonable modal representation is required in practice. In the paper we propose an pictorial representation for LTL formulas, called *diagramic notation*. This notation is designed for aiming at: easy to use, easy for others to understand what is described, and expressive to represent properties relative to LTL. Each representation in the diagramic notation consists of a *viewpoint*, *sequences of events*, *divergences*, and *propositional formulas*. Each propositional formula located along a sequence is either a logical assumption or a consequence, depending on whether it is located above or below the sequence. Combination of the above graphical components with the design of this notation enables us to easily capture the visual understanding of system properties. Furthermore, for those who obtain an LTL formula by interpreting manually from a system property, they can translate the formula in diagramic notation, so that they convince that their visual understanding of the property and the visual meaning of the formula coincide.

1 はじめに

組込みシステムの開発工程では，ソフトウェアの品質を維持する為に，様々なプロダクト評価の工程を設けている．他方，製品の多機能化や大規模化に伴い，組込まれるソースコードの規模は増加し続けている．その為，職人的技術や人的資源に依存するだけでは品質の確保が難しくなっている．特に，タイミングに起因する不具合は，開発設計時のレビューやテストだけでは，不具合を発見する為に十分な効果を上げているとは言い難い．また，ようやく発見した不具合候補も再現できる保証はない．ますます大規模化するソフトウェアにおいて，テストやデバッグ工程の大

[*]Satoshi KOIKE, 矢崎総業株式会社 技術研究所.

[†]Satoru YOSHIDA, 独立行政法人産業技術総合研究所 システム検証研究センター.

[‡]Hitoshi OHSAKI, 独立行政法人産業技術総合研究所 システム検証研究センター.

部分を人手に頼る従来工程では，その品質が今後も維持されていくかは未知である．

こうした不安を抱える組込みソフトウェアの開発現場は，特殊な例ではないだろう．我々は現在，矢崎総業と産業技術総合研究所との共同研究プロジェクトにより，車載ソフトウェアの開発現場へ数理的手法を導入する為の研究に取り組んでいる．発見困難な不具合を検出する技術の研究開発，再現性の極めて難しい不具合の原因を究明する技術の研究開発に，我々が取り組む過程で生まれた一つの検査支援技術について本論文では解説する．

我々はこれまでに，要求分析，ソフトウェア設計，プロダクト評価など，ソフトウェアの開発プロセスが，現場でどのように実施されているかという調査を行い，モデル検査法を効果的に開発現場へ導入する為の具体的な方法を検討してきた．また，様々な実践的モデル検査の適用実験を通じて，モデル検査技術を現場導入する際の障害となる問題の整理を行ってきた．その結果，経験と観察に基く考察ではあるが，モデル検査技術の現場導入の大きな障害の一つになりうると考えられるのが，検査を行う際に投入する「検査式」の作成である．

モデルを作成するには，対象となるシステムの理解があれば，かなりの助けになる．システム開発の延長線上にあると考えることも難しくない為，多少のトレーニングを必要とするものの，大きな障害なく，現場の技術者はモデル作成を受け入れる事ができる．しかし，システムの要求分析や詳細設計の知識では，モデル検査に必要な検査式を作る事は難しい．実際に，適切な検査式を作成する為には，対象となるシステムの理解以上に，初歩的な時相論理の理解が必要になるからである．時相論理は，システム開発の現場技術者が通常必要としている知識とは性格が異なる為，検査式の作成とそのデバッグを会得するにはかなりの労力を要する．

我々のモデル検査実験では，モデル検査ツール *SPIN* [1] を使用した．*SPIN* では，検査式を線形時相論理 (linear temporal logic，LTL と略す) の論理式で記述する．しかし現場技術者たちとの実験から，検査式作成の上達に予想以上の時を要するばかりか，検査全体のうち検査式作成とその修正が占める時間的な割合が大きいことが判明した．そこで我々は，LTL 論理式 (LTL 式と略す) の代わりとなる表現とは何かを模索した．特別な知識を必要としない様相表現，誰もが受け入れられる直感的な表現，行いたい検査内容を表現する表現能力，これらを目標として考案されたのが，本論文で提案する**図示記法** (diagramic notation) である．

2　図示記法

図示記法は，手書き計算の発想に基づいて考案された，LTL 式を表現する為の記法である．時系列を直線で，状態を黒点で表す単純な表現方法である．視点は，初期状態もしくは任意の状態を表す時点に置くことができる．特定の時点で，ある命題が成立する場合，もしくは成立すると仮定する場合を，それぞれ描き分けることができる．連続した時系列を指定して，命題が成立することを表すこともできる．LTL の Kripke 意味論の説明で用いられる図的な表現を，親しみやすく把握しやすい形にまとめようとしたのが，図示記法の出発点とも言える．

図示記法には実行列，視点，命題の成立 · 仮定の 3 つの基本構成要素，分岐とメタ記号の 2 つの拡張構成要素がある．以降の節で，各構成要素について解説する．

2.1　実行列

初期状態から始まる状態遷移の，逐次列の性質を表現する手段として LTL は一般に用いられている．しかし，LTL 式自体からは，表現しようとする実行列の直感的なイメージを掴みにくい場合が多い．時系列ごとの状態の移り変わりを視覚的に表現する為には，注目したい状態を時系列ごとに列挙して直線で繋ぐ方法が自然である．

図示記法では，各状態遷移系列を実行列と呼び，各状態の時点を表す黒点 (●) と時系列を表す直線で記述する．直線には実線と点線があり，黒点間を繋ぐ実線は 1

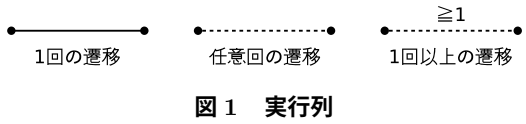

図 1　実行列

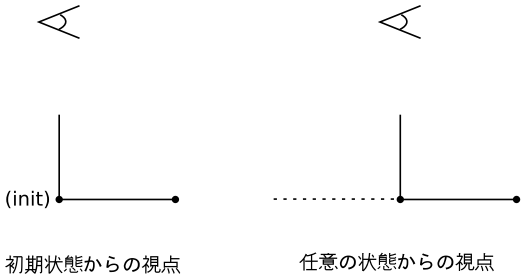

図 2　視点

回の遷移を意味する．黒点と黒点の間に任意回の遷移がある場合は点線で表す．補助的に，点線の上に不等号を用いた遷移回数の範囲を記述できる．1 回の遷移，任意回の遷移，1 回以上の遷移の表記を，図 1 に示す．

2.2　視点

次に，視点を定義する．視点は，図的に記述された実行列がどの状態から始まった列であるかを明示する．視点には,「初期状態からの視点」と「任意の状態からの視点」の 2 種類がある (図 2)．これらは，LTL 式における ◇ と □ に対応する．初期状態からの視点の場合，初期状態の時点の左側に `(init)` と記述する (図 2 左)．任意の状態からの視点の場合は，左側に任意回の遷移を表す点線を記述する (図 2 右)．さらに，視点の位置を明示する為に，視点の位置の時点を表す点 (●) から上方に縦直線を引き，その上に視点記号を記述する．

我々は，LTL 式に対する理解の困難さが，視点が複数ある状況に起因すると考えた．その為図示記法では，1 つの図につき 1 視点のみを配置可能，と定義する．

また現状では，仕様作成の段階でモデル検査を前提とすることはなく，要求仕様がどの状態から始まった実行列に対するものなのか，明確になっていない場合が多い．視点の記述を必須とし，かつ，1 つに限定する事で，満たされねばならない性質の，明確な理解を促進できると考える．

2.3　命題の成立 · 仮定

ある命題が特定の状態で成立している事を表現するには，該当する状態 (●) から上方に縦直線を引き，その上に成立する命題を配置する．この命題は，前後の状態とは無関係に成立する命題のみを対象とし，命題論理で記述する．つまり，∨(or)，∧(and)，¬(not)，→(imply) の論理結合子と原子命題からなる論理式のみを命題として配置する事ができる．

次に，ある命題が特定の状態で成立すると仮定する場合，つまり，命題論理における含意の前提を記述する場合には，成立とは逆に，該当する状態 (●) から下方に縦直線を引き，その下に命題論理式を配置する．原子命題 p の成立および仮定の例を，次頁の図 4 に示す．

ある命題が連続して成立する，または連続して成立することを仮定する場合は，図 3 に示すように，成立する状態の範囲を明示する．この時，範囲の右側にある白丸は，右側開区間を意味する．つまり，図 3 左は,「p がある状態から有限遠方のある状態を含まない状態まで成立する」ことを意味する．仮定の場合も同様である (図 3 右)．この記法は，*SPIN* での検査式 (LTL 検査式と呼ぶ) で用いる U 演算子との対応を取る事を考慮している．LTL 検査式では p U q は,「q が成立するまで (ただし，q が成立する時点を含まない) p が成立し，q はいずれ成立する」ことを表す．

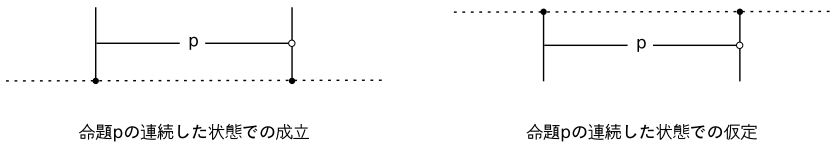

図 3　連続した命題の成立 · 仮定

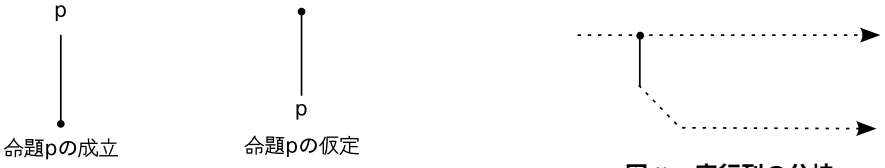

図 4　命題の成立 · 仮定

図 5　実行列の分岐

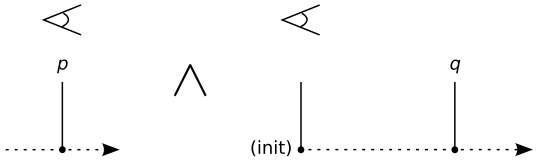

図 6　メタ記号を用いた図の接続

2.4　実行列の分岐

検査式が，2 つ以上の異なる実行列について言及している場合がある．例えば,「初期状態から見て，いずれ p が成立する．または，いずれ q が成立する」という性質等である．異なる実行列について，それぞれの性質を表す場合には，数直線を分岐させて表現する．図 5 は，実行列を分岐させる場合に使用する表記法を表している．上方，下方の実行列は，それぞれ前節までに説明した実行列と同様に扱う．

2.5　メタ記号

2.4 節までに説明した記法を用いると，最外演算子のみが様相演算子である LTL 式や，応答性 $\Box(p \rightarrow \Diamond q)$ などの LTL 式を図示記法で記述する事ができる．しかし，最外演算子のみが様相演算子である複数の LTL 式を，論理結合子でつないだ LTL 式では，1 視点のみからなる図示記法で記述する事ができない場合がある．下の式 (1) は，1 視点のみからなる図示記法で記述できない例に相当する:

$$\Box p \ \wedge \ \Diamond q \tag{1}$$

このような LTL 式を表現する為には，複数の図に対する**メタ記号**として，論理結合子を使用することを許す．例えば，式 (1) に対する図示記法は，図 6 に示す表記となり，2 つの図をメタ記号としての $\wedge$ で接続している．

最外演算子が $\vee$ の場合も同様である．しかし，この場合は，2.4 節の分岐を用いたり，記述方法を工夫する事により，1 つの図で記述できる場合もある．例えば，式 (1) の $\wedge$ を $\vee$ に置き換えて得られる以下の式 (2) の場合，図 6 中の $\wedge$ を $\vee$ に置き換えて記述する事もできるが，図 7 のような記述も可能である．

$$\Box p \ \vee \ \Diamond q \tag{2}$$

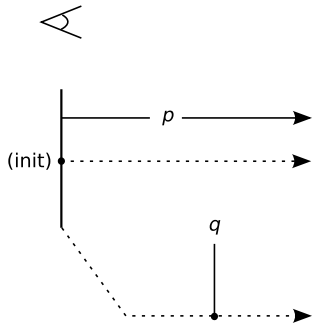

図 7　メタ記号を用いない，$\Box p \vee \Diamond q$

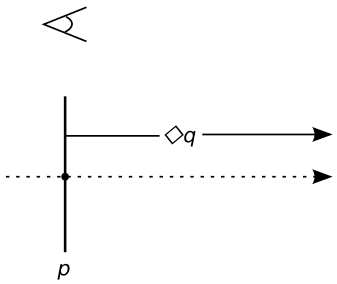

図 8　命題記号に様相演算子を含む場合

2.6　図示記法の表現能力

複雑な LTL 式は，複数のより単純な LTL 式に分割できる場合が多い．モデル検査で投入する検査式が複雑になると，検査式の修正に多くの時間を取られてしまうことがある．検査式が単純であれば，投入する検査式の正しさの信頼性が増す為，モデル検査工程の手戻りが減り，モデル検査の作業効率が上がる．また，検査式の正しさに確信が持てるならば，検査式の反例を示す実行列が得られた場合に，対象システムの不具合のなのか，モデル化に誤りがあったのか，のみを検討すればよい．こうした考察をふまえてモデル検査を行うと，投入しようとする LTL 検査式を表現する図示記法の図が得られやすくなる．実際に，我々が実施した共同研究の中では，一部の例外を除くすべての LTL 検査式が図示記法で表現可能であった．しかし，図示記法の表現可能なクラスは，LTL のクラスと一致している訳ではない．例えば式 (3) は，図示記法では記述できない．

$$\Box(\, p \;\rightarrow\; (\,\Box \Diamond q\,)) \tag{3}$$

応答性の式に似たこの式は，p が成立した時点を含む以降では，q が成立する状態が無数あることを意味する．現在の図示記法では，一つの数直線上には，有限回もしくは連続して成立する命題を表現することができる．もし，さらに式 (3) を図示記法で表現可能にする為，例えば図 8 のように，図中に記述可能な論理式に様相演算子を含む記法を許せば，この問題は解消する．

しかし図示記法では,「図から得られる直感的理解を阻害する表現は取り入れない」ことを方針としており，図 8 の表現は，この場合に相当する．そこで本論文で提案する図示記法の定義では，図中に記述可能な論理式は命題論理式のみとしている．

一方，式 (3) の部分項 $\Box \Diamond q$ は，現在の図示記法で記述可能である．図 6 の右に現れる $\Diamond\, q$ に相当する表現を基本に，視点の表記を図 2 右の表現にすればよい．

3　非正規記法と使用例

実際のシステムに対するモデル検査では，自然言語で記述された要求仕様から LTL 検査式を検討する．しかし，現状の要求仕様は LTL 式で記述するには曖昧であり，モデル検査に精通した研究者であっても，非常に困難な作業となる．さらに，反例の導出を含む場合など，一つの LTL 式では表現出来ない場合もある．以降で示す例題もその一つである．そのような複雑な検査を行う場合等，常に厳密な記法を要求する事は，かえって，検査式検討の妨げとなる事もある．

非正規記法は，図示記法に曖昧さを許す事により，図示記法と自然言語の要求仕様との対応付けの理解を促進する．その上で，命題への対応や命題間の様相的関係等を検討しながら，徐々に正規な図示記法へと置換え明確化し，LTL 検査式を作成する方法を提案する．非正規な記法には，例えば，以下のものが挙げられる．

変数名	定数	意味	変数名	定数	意味
in	SUCC	入力成功	out	ALRM_ON	警報発生
	FAIL	入力失敗		ALRM_OFF	警報停止

表 1　入出力変数

- 命題および命題論理記号のみ記述できる箇所に，自然言語の文章を記述する
- 状態遷移を表す直線に，遷移の内容を併記する
- 状態を明示する為に，状態にラベルを付ける
- メタ記号を用いて図示記法間の関係を示す箇所に，論理記号以外の記述を許す
- 「少なくとも 1 個の状態遷移パスが存在する」など，LTL では表現出来ない性質を検査する為の「反例の導出」のような，コメントを許す

正規な図示記法と非正規記法を使い分ける事で，検査内容の正確かつ直感的な記述，段階的な検査内容の検討など，多くの検査段階で一貫した記述を行う事が出来る．

この節では例題を用いて，要求仕様から LTL 検査式を作成する例を解説する．まず非正規な図示記法の図を作成，それを厳密化し，LTL 検査式を作成する．また，この例題は，組込みシステムのソースコードを対象としたモデル検査を抽象化したものである．C 言語で記述されたソースコードをできる限り忠実に Promela モデルへ変換し，*SPIN* によるモデル検査を行った．

3.1　対象システム

このシステムは，周期駆動型の組込みシステムである．メインタスク `main()` から，検査対象となる処理タスク `task()` を，無限ループを用いて一定周期ごとに呼出す．

`task()` の入出力にはグローバル変数を用いる．データ入力変数 `in` は入力成功を表す `SUCC` と，入力失敗を表す `FAIL` の 2 値を取る．また，出力には，変数 `out` に警報発生 (`ALRM_ON`) か警報停止 (`ALRM_OFF`) を代入する．`out` の値は，`in` と内部変数 `timer` によって定まる．`timer` は，初期値を `TIMER_MAX` とし，タスク開始時に `in==FAIL` の場合，`task()` 内で 1 減される．入力が `SUCC` であった場合は `timer` の値は初期化される．さらに，タスク開始時に，`in==FAIL` かつ `timer==0` であった場合，`out` に `ALRM_ON` が設定される．それ以外の場合には `ALRM_OFF` となる．入出力用変数の取り得る値を，表 1 に示す．

このシステムの特徴として，周期タスクごとに，入力変数 `in` がリセットされる点がある．実例を基にした例題の為詳細については割愛するが，今回のモデルでは，その動作を反映させ，`task()` 終了後に `in=SUCC` を実行するモデルとした．本仕様の Promela モデルの一例を，次頁の図 9 に示す．

3.2　モデル検査

実際の実験で行った検査を，図 9 のモデルで考察する．検査内容は以下である．

　　信号 `FAIL` が入力され続けたら，出力信号が `ALRM_ON` となる

この検査は，意味的に，以下の前提と結論の，2 つの検査に分ける事ができる．

前提　入力信号が `FAIL` であり続けることがある
結論　出力信号が `ALRM_ON` となる

まず前提についてだが，一見,「常に，入力信号が `FAIL` 以外である」の反例の導出により検査できる思われる．しかしこの例では，周期毎に入力信号が初期化されるので，反例導出では元の検査内容を反映していない．1 周期の動作を表した非正規な図示記法は，図 10 となる．

そこで，このシステムの特徴である周期駆動型タスクと，入力信号が FAIL である場合のカウンタの役割を果たす変数 `timer` の性質に着目する．すると，この前提

```
 1 /*** Sample Model ***/
 2
 3 /* input pattern */
 4 #define FAIL  1
 5 #define SUCC  0
 6
 7 /* output pattern */
 8 #define ALRM_ON 1
 9 #define ALRM_OFF 0
10
11 #define TIMER_MAX 10
12
13 byte in;
14 byte out;
15 unsigned timer:4 = TIMER_MAX;
16 unsigned timer_pre:4;
17 byte in_task;
18
19 proctype task()
20 {
21   if
22     :: (in == FAIL) ->
23        if
24          :: (timer == 0) ->
25             out = ALRM_ON
26          :: else ->
27             timer_pre = timer;
28             timer--
29        fi
30     :: else -> /*in==SUCC*/
31        out = ALRM_OFF;
32        timer = TIMER_MAX
33   fi;
34 }
35
36 proctype env_drv()
37 {
38   if
39     :: in = SUCC
40     :: in = FAIL
41   fi
42 }
43
44 active proctype main()
45 {
46   int n;
47
48   n = _nr_pr;
49   do
50     :: true ->
51        run env_drv(); (n == _nr_pr);
52        in_task = true;
53 ENV:
54        run task(); (n == _nr_pr);
55 VERIFY:
56        skip;
57        in_task = false;
58        in = SUCC
59        /* データ入力リセット */
60   od
61 }
```

図 9　Promela モデル

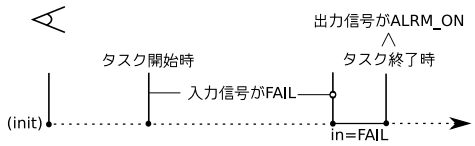

図 10　1 周期の動作

は，以下の性質と同じであると言える．

> タスク開始時の入力信号が FAIL ならば，常に，そのタスク終了後に，timer が 1 減される

さらに,「タスク終了時に timer が 1 減されていれば，常に，入力信号は FAIL である」という検査も行うと，このモデル上では，以下の性質が満たされると言える．

> タスク終了時に timer が 1 減するのは，常に，入力信号が FAIL であるときで，かつその時に限る

結論も同様に考える．出力信号が ALRM_ON になるのは，厳密にはタスク終了時に timer が 0 である場合である．従って結論は，以下の性質と同じである．

> 入力信号が FAIL の時 l に timer が 0 ならば，常に，そのタスク周期後に，出力信号が ALRM_ON となる

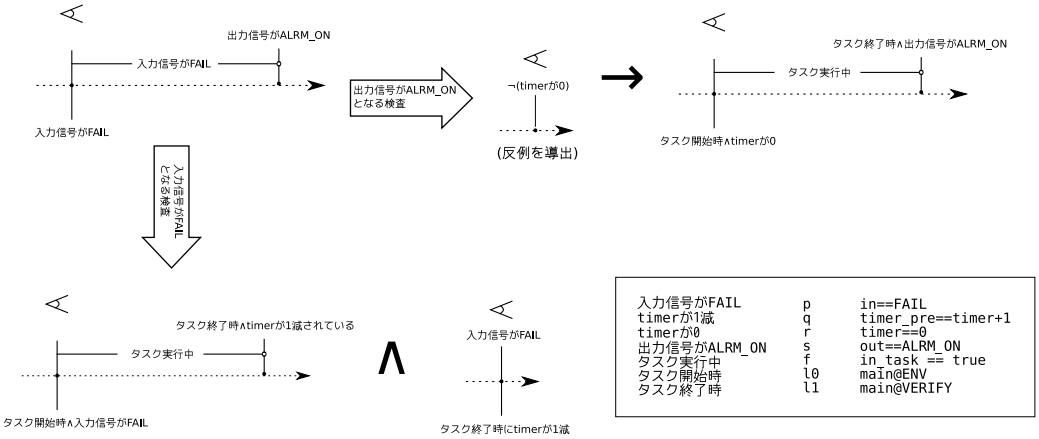

図 11　例題の検査内容

最後に，前提と結論を関連付ける性質として「timer が 0 になることがある」を検査する．これは「timer が，常に，0 以外である」ことの反例として導出できる．これらの検査内容とそれぞれの関係を，非正規な図示記法を用いて図 11 に記述する．

以上をまとめると，検査項目を以下のように 4 つに分割して検査を行えばよい。

1. 入力信号が FAIL ならば，常に，そのタスク周期後に，timer が 1 減される．
2. timer が 1 減されていれば，常に，入力信号は FAIL である．
3. timer が 0 の値を取ることがある (「常に timer は 0 とは異なる値を取る」の否定)
4. 入力信号が FAIL の時に timer が 0 ならば，そのタスク周期後に，出力信号が ALRM_ON となる．

これらの項目に対して，図 11 を参考にして適当な原子命題を選ぶことにより，図示記法の正規な記法による図を与えることができる．さらに，正規な図示記法と LTL 検査式との対応により，以下に示す LTL 検査式を与えることができる．

$$\Box((p \wedge l_0) \rightarrow (fU_s(l_1 \wedge q))) \tag{4}$$
$$\Box((q \wedge l_1) \rightarrow p) \tag{5}$$
$$\Box\neg r \quad \text{(反例の導出)} \tag{6}$$
$$\Box((p \wedge r \wedge l_0) \rightarrow (fU_s(s \wedge l_1))) \tag{7}$$

3.3　検査作業への図示記法の適用

我々の実験では，この事例の様に一見すると単純な LTL 検査式で検査可能だと思われるが，実際には幾つかの検査を組合わせなければならない検査項目が，多く存在した．この様な検査では，モデル検査に精通した技術者・研究者によって，様々な検査式を試行錯誤しながら適用する事により，必要な LTL 検査式の作成を行ってきた．しかし，これには多大な時間がかかっており，システム開発の現場技術者にこれらの作業を行わせる事は非常に困難である．図示記法を用いる事により，LTL 検査式の意味を直感的に理解でき，それにより検査式作成の時間を大幅に短縮できる．

4 関連研究

Dillon らは，Graphical Interval Logic(GIL) と呼ばれる図表現による論理を提案している [4]．GIL は，主に並行システムを対象とし，状態遷移系列の特定の区間を明示する点が特徴である．併わせて提案されている文字表現の Future Interval Logic(FIL) との対応付けにより，GIL と FIL は相互に変換可能な表現形式である．Dillon らが GIL を提案した背景は，我々と同様，開発技術者にとって，論理式の作成が困難であるとの認識からである．だが，我々が想定する組み込みソフトウェア開発現場におけるモデル検査工程では，既存の論理である LTL の部分クラスでも，検査に十分な表現能力があると考えている．したがって，論理体系そのものを提案している Dillon らとは，この点が大きく異なる．GIL は，モデルで満たされるべき検査内容だけでなく，モデルそのものも GIL で記述し，その為のツールを提供している．また，GIL は LTL と同様な様相表現を持つが，区間を表現する特徴などを踏まえた他の時相論理との関連は明確ではない．既存の研究成果をなるべく生かそうという我々の研究方針では，こうした問題は起きにくい．また，我々は，プログラミングに精通した技術者に対し，ソフトウェア開発現場に馴染みやすいモデル化技法を提供することにより，モデル化作業の負担を軽減しようと考えている．実践的なモデル化技法の開発に向けた我々の取り組みについては，[5] を参照されたい．

SPIN および Promela を提案した Holzmann らは，そのスペックパターン集を用いた検査の補助ツールとして，FeaVer を作成している [3]．FeaVer は，ANSI-C で記述されたソースコードを Promela へ変換するツールと，検査内容を図で記述し LTL 検査式へ変換する Timeline Editor，およびそれらの GUI を提供している．Timeline Editor は，時系列に沿ったイベントと，イベント列中で満されるべき制約を記述し，*SPIN* で Never claim として使用される ω-オートマトンへ変換する．また，記述されたイベントと制約は，Promela 中のブール値と対応付けられる．その為，簡便な性質の記述が可能であるが，3 節で与えたようなやや複雑な性質の記述は難しい．また，Timeline Editor での記述は，図示記法においては，

- 視点が任意の状態
- 図中の命題記号は，Promela のブール値と対応
- 制約は命題の成立，イベントは命題の仮定を用いて記述

とする事により，同等の記述が可能であると考えられる．つまり，我々の提案した図示記法の方が記述性が高く，Timeline Editor の記述性では不十分であると考える．図から LTL 検査式への自動変換については，我々も検討する必要がある．

5 まとめと課題

我々は，車載組込みシステムを対象としたモデル検査の共同研究を通じて，さまざまなモデル検査実験を行ってきた．多くの実験を通じて，経験的に獲得した LTL 検査式作成に関する考察は，次の 2 点である:

1. ソフトウェア開発の現場技術者にとって，検査項目から LTL 式への翻訳工程は，心理的な敷居が非常に高い，
2. LTL の部分クラスで，十分な検査が実施可能である．

データベース，ソフトウェアデザイン分野などで，心理的な敷居を下げる工夫に積極的な事からも分かるようにソフトウェア開発技術の現場導入を加速的に進めていくためには，1 の問題は本質的である．ソフトウェアの開発技術者にとって，LTL などの時相論理による表現は，まだ馴染みが薄く，その状況は，当面大きく変わらないだろう．本研究で提案する図示記法は，現場技術者の心理的な敷居を下げるという，モデル検査のためのニッチ (niche) 技術になり得ると考えている．

さらに，図的な表現を組み合わせて得られる視覚的な理解を，検査項目と検査式の中間に置くことで，モデル検査工程のコストを下げる効果を期待している．自然

言語で与えられた要求仕様から検査項目を作り出し，それを LTL 検査式に翻訳する従来のモデル検査工程は，技術者らにとって，まだコストが大きすぎる．

次に，検査項目そのものの考察として，複雑すぎる検査項目は検査に適さない，という考え方について述べる．問題箇所を発見後，原因をすぐに特定可能にするには，検査項目は単純な方が良く，複雑な検査項目は，複数のより単純な検査項目に分割される．実際に，ソフトウェア評価工程の検査項目は，単純な検査内容である．開発段階のレビューでも，過去の経験に基づいて，単純でかつ細かな項目を多数検討する場合が多い．ソフトウェア評価工程の一部をモデル検査に置き換えるならば，複雑すぎる検査項目を扱う機会はあまりない．我々の手がけたモデル検査実験では，これまでに多くの LTL 検査式を投入したが，これらはいくつかのパターンに分類することができる．式のパターンを表現するために必要な構成要素を図的な部品とし，部品を組み合わせる事で柔軟な表現力を発揮するという考え方を基に，図示記法は設計されている．2 点目の考察は，図示記法でその具体例を示した事になる．

本論文では紹介しなかったが，LTL 検査式の分類作業を生かして，LTL 検査式集 (スペックパターン集と呼ぶ) も作成している．Dwyer らは，我々にさきがけてスペックパターン集を作成し，一般に公開している [2]．我々の作成するスペックパターン集は，車載ソフトウェアに対するモデル検査で頻繁に使用されている検査式を収集するとともに，検査結果が別のどの検査に関連するかを解説しており，用例集に近い．Dwyer らは，構文的な分類のみにより LTL 式のパターンを整理したが，我々は,「パターン同士の幾つかは，互いに関連している」という観察のもとにスペックパターンの分類を行っている．その結果，検査の用途に応じて関連するパターンを次々と当てはめていく，という利用が可能である．また，我々のスペックパターン集では，本稿で提案した図示記法を，見出しとして用いている．これにより，LTL に十分な理解がない開発技術者であっても，図示記法による直感的な理解で，必要な LTL 検査式を選択する事ができる．このスペックパターン集には企業の機密情報を多く含む為，公開にはしばらく時間を要すると思われる．

今後は，図示記法の文法的特徴付けについて検討する予定である．同値な図示表現を自動判定する方法や，記述可能な LTL 式のクラスの切り出しが，考察の中心となる．図示記法の意味論についても，現在の定義を踏まえて，厳密に検討する必要がある．先行研究として，Dillon らによる GIL の研究が参考になる．LTL 式と図示記法との詳細な対応を検討することにより，図示記法と LTL 式との間の自動変換が可能になると考える．また，理論的な考察をもとに，図示記法の自然な拡張を行い，さらに記述性を高める事を目指したい．

謝辞 本研究は，矢崎総業株式会社およびグループ各社と，独立行政法人産業技術総合研究所システム検証研究センターによる共同研究「車載ソフトウェアのモデル検査に関する研究」(2005 年 4 月から 2007 年 3 月まで) にて実施された．共同研究に携った各位に感謝する．

参考文献

[1] G.J. Holzmann: *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley Publishing Company, 2003.

[2] M.B. Dwyer, G.S. Avrunin and J.C. Corbett: *Property Specification Patterns for Finite-state Verification*, Proc. of 2nd Workshop on Formal Methods in Software Practice (FMSP), Clearwater Beach (Florida), pp. 7–15, ACM Press, 1998.

[3] G.J. Holzmann and M.H. Smith: *Automating Software Feature Verification*, Bell Labs Technical Journal, vol. 5(2), pp. 72–87, 2000.

[4] L.K. Dillon, G. Kutty, L.E. Moser, P.M. Melliar-Smith and Y.S. Ramakrishna: *A Graphical Interval Logic for Specifying Concurrent Systems*, ACM Transactions on Software Engineering and Methodology, vol. 3(2), pp. 131–165, 1994.

[5] 河本孝久, 小池憲史, 古橋隆宏, 鈴木伸一: 周期タスク型モデル検査法と車載組込みシステムへの適用事例, 組込みシステムシンポジウム (ESS2007), 情報処理学会, 2007(予定).

LTL モデル検査のための図示記法
(算譜科学研究速報)
発行日 2008 年 2 月 18 日
編集・発行：独立行政法人産業技術総合研究所システム検証研究センター
同連絡先：〒563-8577 大阪府池田市緑丘 1-8-31
e-mail：informatics-inquiry@m.aist.go.jp

Diagramic Notation for LTL Model Checking (in Japanese)
Feb 18cb, 2008
Research Center for Verification and Semantics (CVS)
Ikeda Site
National Institute of Advanced Industrial Science and Technology (AIST)
1-8-31 Midorigaoka, Ikeda, Osaka, 563-8577, Japan
e-mail: informatics-inquiry@m.aist.go.jp

AIST01-J00022-77