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

2008－ＳＥ－160 (6)
2008－EMB－ 9 (6)
2008／6／19

制御系ＥＣＵ調停器の検証における演繹的アプローチについて

湯浅 能史[1,2]　　足立 正和[3]　　佐野 範佳[3]

[1] 東京工業大学 大学院情報理工学研究科
[2] 産業技術総合研究所 システム検証研究センター
[3](株) 豊田中央研究所

組み込みソフトウェアの検証技術としては、従来のソフトウェアテストに加えて、近年ではモデル検査法も注目されている。これらは、入力例に対する振る舞いを調べて確認する、いわば実証的な検証法である。これと双璧をなす古典的な形式検証技術として、定理証明による演繹的手法が知られているが、事例報告は現状では少い。本研究* では、車載システム検証の一事例として、電子制御ユニット (ECU) の調停器の幾つかの性質を、対話的定理証明器 Agda を用いて演繹的に検証したので、それについて報告する。

Verifications of an arbiter for electronic control units with an interactive theorem prover

Yoshifumi Yuasa[1,2]　　Masakazu Adachi[3]　　Noriyoshi Sano[3]

[1]Tokyo Institute of Technology
[2]Institute of Advanced Industrial Science and Technology
[3]Toyota Central R&D Labs., Inc.

As a method for verifying embedded softwares, various software tests are widely used. Some model checking methods also have become popular in recent years. All these are inductive methods in the sense that we verify a program by tracing and examining many (or all) executions with it. On the other hand, not many cases of verifications of embedded softwares by deductive methods are reported. In this paper, we report a case study of verifications of automotive softwares by deductive methods. We show some basic properties of an arbiter for electronic control units with an interactive theorem prover Agda.

1 序

ソフトウェア検証の方法論は大きく２種類に分られる。一つは物理実験のように、数多くの（場合によっては全ての）具体的実行例が求められる性質を持つことを確認する方法である。ここでは、これを**実証的検証法**と呼ぶことにする。現在もっとも広く行われている検証法であるソフトウェアテストはこの代表である。また、近年注目されているモデル検査法も実行パスの検証を行うものであるのでここに分類する。

もう一つは、数学で行うような証明によって、ソフトウェアの性質を保証するという方法論である。これを**演繹的検証法**と呼ぶ。演繹的手法によるソフトウェ

*本研究の主要部分は産業技術総合研究所と豊田中央研究所の共同研究として行われた。

ア検証は、歴史的にはモデル検査よりも古く、軍事/宇宙開発、原子力、また民生用では航空管制など、主にSafty-criticalなシステム開発において用いられてきた。しかし、一般的な組み込みソフトウェアの開発でこれを用いた事例は、実験的なものまで含めても、かなり少いと思われる。

よく指摘されるように、ソフトウェアテストは網羅的でなく、従って検証したい性質を完全に保証するものではない。この点モデル検査法は網羅的であり、完全な信頼性を保証できる。ただし検証できる性質には制限があり、時相論理式や簡単な算術等の範囲で記述される必要がある。この範囲を越えるものについては、個別に抽象化等の工夫を行い、問題を変換することになるが、これは変換の正当性を保証するといった別の問題を生み出す。また、中規模以上のソフトウェアに適用する場合には、状態数爆発によって現実的な時間内に検証が終了しない事もある。

これに対して演繹手法では、基本的に任意の性質を完全に保証することができる。また検証過程をユーザが細かく制御できるため、適切な場合わけの設定などにより状態爆発の問題を避けることができる。但し、この点は問題点でもある。現在普及している証明支援系では、なんらかの意味で自動証明器による証明サポート機構が組み込まれてはいるが、モデル検査器等に比べると、はるかに人手の介在を必要としており、またユーザに要求する技能レベルも高い。現在、演繹手法が産業界で余りポピュラーでないのは、上記のような人的コストに主な原因があると思われる。

しかし、任意の性質が示せる点は代替の効かない利点であり、実証的手法の隙間を埋めるかたちで、組み込みソフトウェアの検証法としても、近い将来、重要な位置を占めるのではないかと思われる。本稿6節では、演繹的検証法の有用な利用法として、相対的正当性検証の話題を取り上げたが、これはそのような利用法の一例でもある。

本稿では、演繹手法による車載ソフトウェア検証の一事例として、制御系ECUの調停器を検証した。対象はデータフロー図で記述されており、更に状態遷移機械も含んでいる。まず、このようなシステムを数学的に表現する一般論を提示する。続いて具体的なECUと調停器の実装を提示し、これが「調停器の公理」とでもいうべき3つの性質を満すことを演繹的に証明する。

形式証明の作成は対話的証明支援系Agdaを用いて行ったが、他の証明支援系でも実行可能である。最後の節では、相対的正当性保証の概念を導入し、Agdaの汎用プラグイン機構を利用した統合検証環境について述べる。

2 演繹的検証と証明支援系

まずソフトウェアの演繹的検証法の一般論を簡単に解説する。演繹的検証法では、ソフトウェアを関数やグラフ（状態遷移グラフ）といった数学的な対象物として捉え直す。これに伴い、検証項目は関数やグラフの持つ性質となり、対象物がその性質を満たすことを証明することが、行うべき検証作業となる。

自然言語で表現した性質を手書きで証明しても、本質的には演繹的な検証だとは言える。しかし、自然言語による記述は曖昧性を含むため誤解を生じることもあり、また手書きの証明では思わぬミスをする可能性がある。記号論理学の枠組みを用いて、検証すべき性質を論理式により記述し、形式化された証明図を作成することで、間違いを完全に排除できる。

形式証明の作成は人手で行うには膨大で、かつストレスの大きい作業である。このため計算機による支援が不可欠となる。それを行うツールが**対話的証明支援系**である。広く知られた証明支援系としては、Isabelle/HOL[1], Coq[2], PVS[3]等が挙げられる。

本稿で行った検証を形式化するにあたっては、対話型証明支援系Agdaを用いた。これはChalmers工科大で開発されたMartin-Lof型理論をベースとする証明支援系である。Curry-Howard対応により関数プログラミング風の証明をサポートする。即ち、証明すべき論理式を型と考え、その型を持つ関数を定義することが証明となる。場合分けはケース文に、数学的帰納法は再帰関数定義に対応させる。以下に簡単な証明の例（自然数は偶数または奇数である）を掲げる。

```
lem0 :: Even zero = ...
lem1 :: (n::N) -> Even n -> Odd  (suc n)
  = ...
lem2 :: (n::N) -> Odd n  -> Even (suc n)
  = ...
thrm :: (n::N) -> Even n \/ Odd n
  = \(n::N) ->
    case n of
```

```
zero  -> inl lem0
suc m -> case (thrm m) of
           (inl p) -> inr (lem1 m p)
           (inr p) -> inl (lem2 m p)
```

証明作成には Emacs によるユーザインターフェースが利用できる。証明支援系 Agda についての詳細は文献 [4] を参照されたい。

3 DFD の高階関数モデル

本稿で対象とするシステムは、文献 [5] によるものであり、データフロー図 (DFD) として表現されている。[1]また内部に状態遷移機械も含んでいる。演繹的検証の土俵に載せるためにまず、このようなシステムに数学的な表現を与えなくてはならない。

DFD の処理対象はデータストリームである。これを自然数（離散的な時刻を表す）にデータを対応させる関数とみなす。データの型を T として：

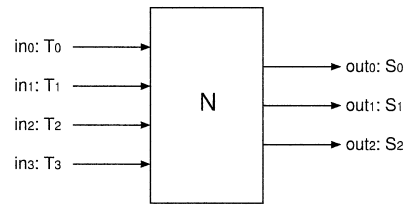
$$Strm\,T = \mathbb{N} \longrightarrow T$$

と定義する。ストリーム処理を行う演算器を**ノード**と呼ぶ。これは入力ストリームを出力ストリームに変換する連続な高階関数と考えることができる。

一般にノードは複数ストリームを入力としてを得て、複数のストリームを出力する。しかしそのような場合でも、それぞれを「束ねた」形で扱い、１入力１出力のノードとみなすことは可能である。本稿ではこの形でノードを表現する事にする。入力型 T, 出力型 S のノードの型を以下の様に定義する。

$$Node\,T\,S = Strm\,T \longrightarrow Strm\,S$$

と定める。以下に具体例を挙げる。



$$N : Node\,T\,S \quad ,$$
$$where \quad T = T_0 \times T_1 \times T_2 \times T_3$$
$$S = S_0 \times S_1 \times S_2.$$

[1]SCADE モデルとして与えられている。

以降の議論で用いるため、まず幾つかの記法を導入する。直積型 $T_0 \times \cdots \times T_{k-1}$ に対して、その第 $i_0, \cdots, i_{l-1}$ 成分を取り出す射影を $\pi_{i_0 \cdots i_{l-1}}$ とする。即ち：

$$\pi_{i_0 \cdots i_{l-1}}(d_0, \cdots, d_{k-1}) = (d_{i_0}, \cdots, d_{i_{l-1}})$$

と定める。射影写像を用いて「束ねられた」ストリームから指定の要素を取り出すことができる。

$$(str)_{i_0 \cdots i_{l-1}} = \pi_{i_0 \cdots i_{l-1}} \circ str : Strm\,T_{i_0} \times \cdots \times T_{i_{l-1}}$$
$$(str : Strm\,T_0 \times \cdots \times T_{k-1})$$

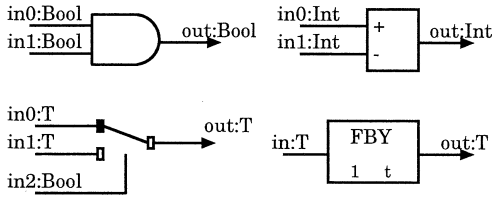
逆にストリーム達を「束ねる」演算も導入しておく。

$$\langle str_0, \cdots, str_{k-1} \rangle = \lambda n.\,(str_0\,n, \cdots, str_{k-1}\,n)$$
$$: Strm\,T_0 \times \cdots \times T_{k-1}$$
$$(str_i : Strm\,S_i \ (i = 0, \cdots, k-1))$$

と定義する。

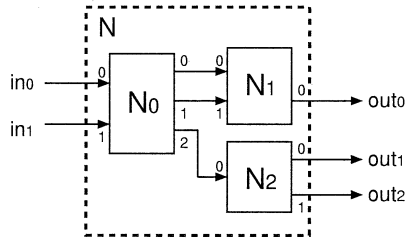
一般に DFD が与えられた時、これはより小さな幾つかの DFD から構成さられたものと見ることができる。（それ以上分割できないもある。これを基本ノードと呼ぶ。）このようなノードの構成に沿って、対応する高階関数を構成する方法を述べる。

基本ノード ノードの最小単位である。データの（各時刻毎の）数値/ブール演算や条件スイッチ等が代表例である。また遅延回路 (FBY) も基本ノードの一種と考えることにする。



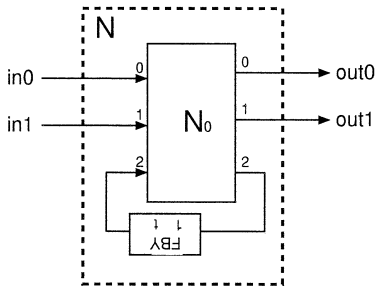
$$And\,\langle in_0, in_1 \rangle \quad = \lambda n.\,(in_0\,n \wedge in_1\,n)$$
$$Sub\,\langle in_0, in_1 \rangle \quad = \lambda n.\,(in_0\,n - in_1\,n)$$
$$Switch\,\langle in_0, in_1, in_2 \rangle$$
$$= \lambda n.\,\text{if}\ (in_2\,n)\ \text{then}\ (in_1\,n)\ \text{else}\ (in_0\,n)$$
$$FBY_{1\,t}\,in$$
$$= \lambda n.\,\text{if}\ (in\,n = 0)\ \text{then}\ t\ \text{else}\ (in\,(n-1))$$

ノードの連結 既に定義されたノードの出力を別のノードに継ぎ、これらをまとめて一つのノードを構成する（ただしストリームのループが生じないようにする）。これを「連結」と呼ぶことにする。連結は高階関数の合成に対応付けることができる。



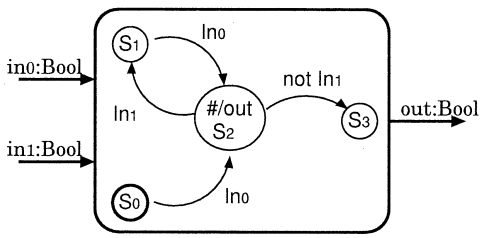
$$N\,\langle in_0, in_1\rangle = \langle N_1\,(s)_{01}, N_2\,(s)_2\rangle,$$
$$where \quad s = N_0\langle in_0, in_1\rangle.$$

フィードバック すでに定義されたノードに対して、その出力を自身にフィードバックして新しいノードを構成する。必然的にストリームのループが生じるが、その場合には、途中に必ず遅延回路 (FBY) が置かれなくてはならない。我々のモデル化では、これを（高階の）再帰的関数定義により表現する。



$$N\,\langle in_0, in_1\rangle = N_0\,\langle in_0, in_1, str\rangle,$$
$$where \quad str = N_0\,\langle in_0, in_1, FBY_{1\,t}\,str\rangle.$$

遷移機械 まず機械の状態を表すデータ型を導入し、遷移系の振舞いをこの型のストリームとして表現する。フィードバックのときと同様、再帰的関数定義を用いる。



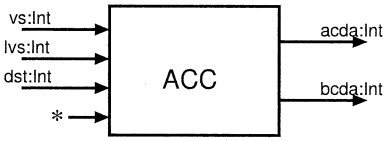
$$D = \{S_0, S_1, S_2, S_3\}$$
$$Step_{(i_0,i_1)}\,S = \text{ case } S \text{ of}$$
$$S_0 \Rightarrow \text{ if } i_0 \text{ then } S_2 \text{ else } S_0$$
$$S_1 \Rightarrow \text{ if } i_0 \text{ then } S_2 \text{ else } S_1$$
$$S_2 \Rightarrow \text{ if } i_1 \text{ then } S_1 \text{ else } S_3$$
$$S_3 \Rightarrow S_3$$

$$s_{\langle in_0,in_1\rangle}0 = S_0$$
$$s_{\langle in_0,in_1\rangle}m' = Step_{(in_0\,m, in_1\,m)}\,(s_{\langle in_0,in_1\rangle}\,m)$$
$$N\,\langle in_0, in_1\rangle = \lambda n.\,(s_{\langle in_0,in_1\rangle}\,n == S_2)$$

4 3つのECUと調停器

本稿で扱うのは、実験用に設計された3つの簡易的なECU（ACC, BKC, TC）と、それらの調停器 Arbiter3 である。文献 [5] にあるものを元にしている。以下にECUのインタフェースを示す。入力の内で本研究と関係無いものはまとめて“*”とした。次節以降これらは省略する。

車間距離制御 (ACC) 自動的に車間距離と走行速度を調整する。

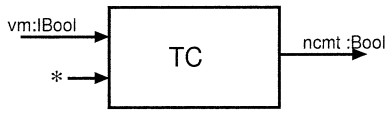


vs	VehicleSpeed	(自車速度)
lvs	LeadingVehicleSpeed	(先行車速度)
dst	Distance	(車間距離)
acda	AccelControlData	(アクセル印加量)
bcda	BrakeControlData	(ブレーキ印加量)

制動力維持制御 (BKC) パーキングブレーキ操作時や、停止時に制動力を印加する。

vm	VehicleMoving	(車速は正か？)
bcdb	BrakeControlData	(ブレーキ印加量)

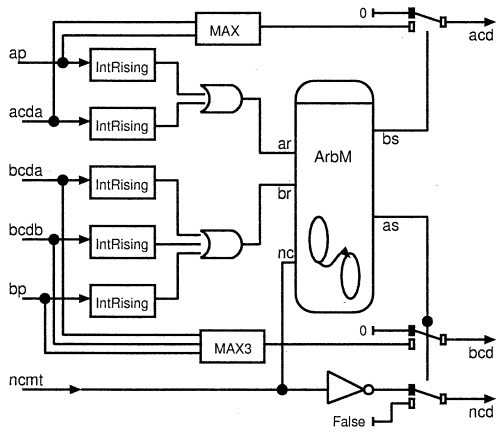
駆動力伝達制御 (TC) 長時間停止時にニュートラル状態にする。



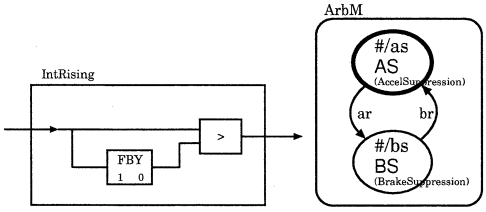
vm	VehicleMoving	(車速は正か？)
ncmt	NeutralControlMode	(ニュートラル)

文献 [5] では3つの ECU は非同期的に動作するとしているが、本稿では（十分に細かい間隔で）同期的に動作すると考える。

さて、3つの ECU は別々の設計者によって独立に設計されると想定して見よう。このような場合、例えば ACC がアクセル印加を行うと同時に、BKC がブレーキ印加を行うといったことが、生じる危険性がある。こういった矛盾した命令を避けるよう、命令の統合を司るのが**調停器**と呼ばれるユニットである。調停器 Arbiter3 の具体的な実装を以下に図示する。



ここで、ノード IntRising および遷移系 ArbSMM は以下のものとして与える。



5 調停器の公理とその検証

前節で示した調停器の入力型を In とし、出力型を Out とする。即ち：

$$\begin{aligned} In &= Int^2 \times Int^3 \times Bool, \\ Out &= Int \times Int \times Bool \end{aligned}$$

と定める。[2]よって調停器の型は $Node\ In\ Out$ である。入力ストリーム $in : Strm\ In$ に対して：

$$(in)_{acd} = (in)_0,\ (in)_{bcd} = (in)_1,\ (in)_{ncm} = (in)_2$$

と定める。出力ストリームについても同様の記法を用いる。また述語 E を次のように定める。

$$\begin{aligned} E\ in = \forall n.\ (\quad & (\max \circ (in)_{acd})\ n = 0 \quad \vee \\ & (\max \circ (in)_{bcd})\ n = 0 \quad) \end{aligned}$$

これはアクセル入力とブレーキ入力が競合しない（同時に印加されない）という条件である。

以下に、調停器 $(A : Node\ In\ Out)$ に必要な最低限の条件をリストアップした。ここに $in : Strm\ In$ は任意のストリームで条件：

$$(*)\ \ (in)_{acd}\ 0 = \vec{0}$$

を満たすものとする。[3]これら三つを調停器の公理と呼ぶことにする。

公理1 アクセル、ブレーキともに、出力が入力の最大を超えることはない。

$$\begin{aligned} \forall n.\ (\quad & (A\ in)_{acd}\ n \leq (\max \circ (in)_{acd})\ n \quad \wedge \\ & (A\ in)_{bcd}\ n \leq (\max \circ (in)_{bcd})\ n \quad) \end{aligned}$$

公理2 アクセル出力とブレーキ出力が同時に印加されることはない。

$$\forall n.\ (\quad (A\ in)_{acd}\ n = 0 \vee (A\ in)_{bcd}\ n = 0 \quad)$$

公理3 アクセル入力とブレーキ入力に競合がない時は、それぞれの最大値がそのまま出力される。

$$\begin{aligned} & E\ in \Rightarrow \\ \forall n.\ (\quad & (A\ in)_{acd}\ n = (\max \circ (in)_{acd})\ n \quad \wedge \\ & (A\ in)_{bcd}\ n = (\max \circ (in)_{bcd})\ n \quad) \end{aligned}$$

[2]本稿では Int は非負の整数を表すことにする

[3]この条件が必要なのは本実装に固有の事情による。実装を変更してこの条件をはずせればなお良い。

定理 5.1 前節で提示した調停器の実装を Arb とする。これは公理１～３を満たす。

以下、この定理の証明の方針を説明する。公理１は定義から容易に導ける。条件スイッチの性質：

$$\forall n.\quad (Switch\langle in_0, in_1, in_2\rangle\, n = in_1\, n \\ \vee\ Switch\langle in_0, in_1, in_2\rangle\, n = in_2\, n)$$

によって場合分けして、定義から導くことができる。ちなみに、こういった基本ノードの性質は、予めライブラリ等で証明しておくとよい。

公理２は、遷移機械 $ArbM$ に関する以下の補題から直ちに導かれる。

補題 5.2 任意の $in : Strm\, Bool^2$ について：

$$\forall n.\, ((ArbM\ in)_{bs}\, n = \bot \vee (ArbM\ in)_{as}\, n = \bot)$$

証明： $ArbM$ は各時刻 n で、AS または BS の何れかの状態をとる。前者なら $(ArbM\ in)_{bs}\, n$ が、後者なら $(ArbM\ in)_{as}\, n$ が、値 $\bot$ をとる。 (証明終)

公理３の証明は、他の二つに比べて若干難しい。ポイントは以下の補題を示すことである。ストリーム ar, br, as, bs は前節で図示したものである。

補題 5.3 任意のストリーム $in : Strm\, In$ で条件 (*) を満すものをとり、更にこれが条件 "$E\, in$" を満すと仮定する。このとき以下が成り立つ。

(i) $\forall n.\ (bs\, n = \top \vee (in)_{acd}\, n = \vec{0})$

(ii) $\forall n.\ (as\, n = \top \vee (in)_{bcd}\, n = \vec{0})$

証明： (i). 時刻 n についての数学的帰納法で示す。時刻 0 では (*) により $(in)_{acd}$ が $\vec{0}$ である。次に時刻 $m+1$ の時を示す。アクセル入力 $(in)_{acd}\, (m+1)$ が $\vec{0}$ なら自明。そうでない時は、時刻 m での $ArbM$ の状態で場合分けし、以下のように証明する。条件 $E\, in$ から $(in)_{bcd}\, (m+1)$ が $\vec{0}$ である事に注意せよ。

時刻 m での状態が AS なら $bs\, m$ は $\bot$ である。帰納法の仮定から $(in)_{acd}\, m$ は $\vec{0}$ であり、よって $ar\, (m+1)$ は $\top$ となる。従って BS への遷移が生じ、$bs\, (m+1)$ は $\top$ となる。時刻 m での状態が BS の場合。上で注意したことより $br\, (m+1)$ は $\bot$ であるから、状態遷移は起らず $bs\, (m+1)$ は $\top$ となる。

(ii) も同様に示せる。ただし、時刻 0 では $as\, 0 = \top$ の方が成り立つ。これは $ArbM$ の初期状態の定義から分る。 (証明終)

補題から公理３を示すことは容易である。(i) に基いて場合分けすれば、定義から：

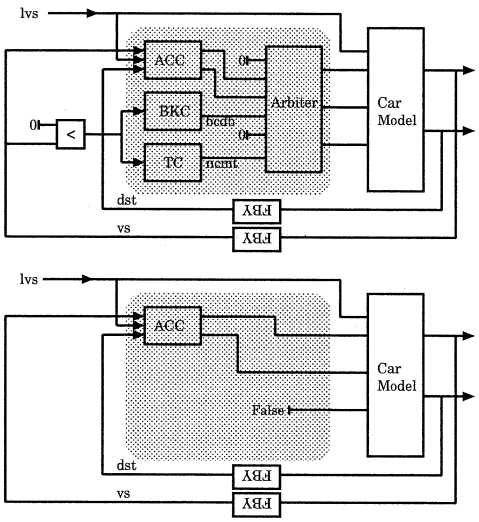
$$(A\ in)_{acd}\, n = (\max \circ (in)_{acd})\, n$$

を確認できる。ブレーキ出力の方も (ii) による場合わけで同様に示すことができる。

6 相対的正当性証明と統合的検証

本節では、実証的手法と演繹的手法を統合的に用いた検証のひとつのスタイルを提案し、その中での演繹的検証の役割として相対的正当性証明の概念を導入する。

以下に示す二つのモデルを考える。前者は、これまで本稿で扱ってきた３つの ECU の出力を、調停器を通して最終的な駆動/制動出力としたものである。運転者はブレーキ、アクセルペダルを操作しないと仮定する。また、適当な車体-走行環境モデルを考え、得られた車速 vs と先行車との車間距離 dst を ECU にフィードバックさせた（lvs は先行車速度)。このモデルを W とする。一方、後者では簡易 ACC のみを用いて、その出力を直接、車体-走行環境モデルに与えてた。こちらを W_0 と呼ぶことにする。



更に ACC と BKC が次の条件を満すとする。それぞれ、これらの ECU に対する条件として十分順当なものである。

条件 A ACC 自体のアクセル出力とブレーキ出力は競合しない。

$$\forall n.\ (ACC\,vm)_{acd}\,n = 0 \vee (ACC\,vm)_{bcd}\,n = 0$$

条件 B 走行中、BKC は制動力印加を行わない。

$$D\,vm \Rightarrow \forall n.\ (BKC\,vm)\,n = 0$$

ここに述語 D は以下のものとする。

$$D(vm : Strm\,Bool) \equiv \forall n.\ vm\,n = \top$$

これは車が走行中（Driving）の状態を意味する。条件 $D\,vm$ が満される時には、条件 A と条件 B により Arbiter 入力についての非競合条件 $E\,in$ が成り立つことに注意せよ。この事実と公理 3 にから、二つのモデルの ECU 部分（網掛け箇所）が同じ振舞いをすることを演繹的に証明できる。従って、モデル全体に関しては以下が成り立つ。

定理 6.1 条件「$\forall n.\ 0 < (W_0\,lvs)_{vs}\,n$」の下で、二つの高階関数 $W_0\,lvs$ と $W\,lvs$ は等しい。

この定理により、もしモデル W_0 での走行パターンが某かの性質 P（例えば「車間距離が常に一定以上に保たれる」といったもの）を満すことが検証されたならば、モデル W も性質 P を満すと帰結してよい、ということが保証される。このようにある対象の性質を別の対象の性質に帰着させる検証方式を**相対的正当性証明**と呼ぶ。[4]これは演繹的検証でなくては難しい検証スタイルである。

仮定の「モデル W_0 が性質 P を持つ」ことを示すのは、演繹的手法でも良いが、それが困難なら他の方法で示すこともできる。モデル検査を用いても良いし、最悪の場合、十分なテストによって「正しいと信じる」という方法もありうる。また本件のように物理系を含むモデルでは、実機テストによる検証も現実的な選択である。何れの方法によるにせよ、定理 6.1 と組合せることで、もとの（W_0 の）性質と「同等の確からしさ」で W の性質を保証できる。

[4]二つの性質は同じ出ある必要はない。「W_0 が Q を満せば W が P を満す」という形でもよい。

定理証明系 Agda には汎用プラグイン機構 [6] が備わっており、簡単な設定により任意の自動検証装置を接続することが出来る。上記のような、総合的な検証を Agda ベースで行うには、このプラグイン機構を用いて、各種検証ツール接続するのが良いと思われる。このようなシステムを Agda 統合検証環境 (Agda-IVE) と呼ぶ。その構築は今後の課題としたい。

7 まとめ

演繹的手法による車載ソフトウェアの検証の一例として、制御系 ECU の調停器の検証を行った。DFD で記述された制御プログラムを、高階関数として捉える方法を提示し、これに基いて 3 つの性質を証明した。また、実証的検証法と演繹的検証法を統合的に利用する枠組みに関連して、相対的正当性検証の概念を導入した。

参考文献

[1] Isabelle Home Page.
`http://www.cl.cam.ac.uk/research/hvg/Isabelle/`

[2] The Coq proof assistant.
`http://coq.inria.fr/`

[3] PVS Specification and Verification System.
`http://pvs.csl.sri.com/`

[4] Agda Official Web Site.
`http://unit.aist.go.jp/cvs/Agda/`

[5] 小田 哲也, 潮 俊光, 富永 孝, 佐野 範佳, 同期型言語を用いたリアルタイムシステムにおける調停機構の検証第 51 回システム制御情報学会研究発表講演会, pp.103-104(2007)

[6] 池上大介: Agda プラグイン機構. 日本ソフトウェア科学会 第 22 回大会 一般講演（2005）