段階的詳細化に基づく 鉄道信号へのフォーマルメソッド適用法

寺田　夏樹*

Application of Formal Methods to Signalling Systems using Techniques based on Stepwise Refinement

Natsuki TERADA

It is anticipated that Formal methods are beneficial to increase reliability of software. In preparing formal specifications of complicated systems from a preliminary draft text, stepwise revisions are very effective to prepare the specifications to understand readily and to realize verified codes that satisfy the specifications. In this report, we first explain the concept of gradual revisions. In writing an abstract specification with minimal conditions initially, then refine the specifications more precisely until they are ready to be serviceable. Next, we present an example, in which allowable speed of train as calculated when the distance to stop is given.

キーワード：フォーマルメソッド，VDM，段階的詳細化，B メソッド，ブレーキ曲線

１．はじめに

鉄道信号は安全性が要求される分野である。電子化機器のハードウェアの安全性技術に関しては技術的にはかなり確立されたものと考えられている。しかし一方でソフトウェアはシステムの中の比重も規模も年々大きくなっているのにも関わらず，その安全性を向上させる技術については以前とは大きく変わっていない。

我々は以前からフォーマルメソッド（formal methods, 形式的手法）によるソフトウェアの高信頼化手法の研究を行ってきた。特に証明による仕様の整合性の保障に強い関心を持っており，ATCの線区データベースに関する整合性の検証等を実施した[1, 2]。ただデータベースはシステムの信頼性に大きく関わっているものの，静的なデータであって列車を実時間で直接制御していない。

そこで実時間で動作する信号装置への適用を目指して研究を実施してきた。まずは実時間システムについてフォーマルメソッドで仕様を記述していく試みを行ったが，複雑な制御システムを記述しようとする場合には，単純に形式的仕様記述言語で記述しようとしても仕様の理解が難しいことが判明してきた。

そこで，段階的詳細化（stepwise refinement）と呼ばれる技法に着目した。これはモデルをいきなり書くのではなく，段階を追って仕様を実装に近づけていく作業である。段階的な記述を行うことにより，仕様の理解がしやすくなる。それぞれの詳細化過程が前の段階からの条件を満たすことを証明することによってその作業の正当性も得ることができる。

段階的詳細化手法に基づき複雑な制御システムを記述する試みを行い，その有効性を検討した。本報告ではその適用例とともに詳細化手法の有効性について述べる。

２．フォーマルメソッドとは

フォーマルメソッドといっても様々な定義があるが，基本的には数学的論理学的体系を持った仕様記述言語を用いて形式的に（数学的に）仕様を記述し，その形式的仕様に対して形式的・数学的に検証を行うことによりシステムの品質を上げるための技術の総称である。図１はフォーマルメソッドによるシステム開発の一例である。

形式的に仕様を記述することにより，仕様のあいまいさを排除できる。この事は仕様の理解の共有のためには重要である。仕様を記述する際にしばしば「暗黙の了解」に基づき，細かい仕様を省略することがありうる。ところがこの「暗黙の了解」が仕様策定者やプログラマの間

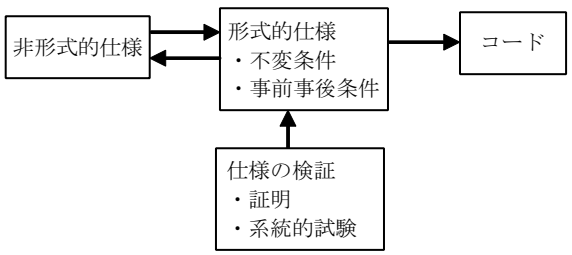

図１　フォーマルメソッドによるシステム開発

＊　信号通信技術研究部（信号）

で共有されていないと，間違いが発生しやすい。また後でメンテナンスする際にも問題が生じる。フォーマルメソッドでは，仕様に記述されていないことは，定義されていないと同義となるため,「暗黙の了解」であっても記述すべきこととなる。こうしてあいまいさを極力排除することが，品質向上の第一歩と考える。

また形式的仕様記述はコンピュータにも理解できる形であり，自動テストや自動証明などといった様々な形でコンピュータの支援により仕様の検証を効率よく行うことができる。仕様の段階での検証を早期段階で十分に行うことにより信頼性やソフトウェアの生産性をあげるのはフォーマルメソッドの狙いである。

なおフォーマルメソッドをサポートするツールによっては仕様からコードを直接生成できるものもあるが，ハードウェアやユーザとのインターフェース部分はある程度手作業でのコーディングが必要である。

フォーマルメソッドの中にもZ[3], VDM[4], B[5], CSP, OBJなど実際には様々な手法が存在する。手法ごとの得手不得手があるため，必要に応じて手法を選択し，場合によっては組み合わせて使うのがよい。VDMやZはシステム仕様の記述言語としての側面が強い手法であり，モデル指向手法と呼ばれるが，証明等による検証は難しい面がある。一方CSPやOBJのような手法ではシステムの検証に重点が置かれているが数学に慣れていない人にとってシステムの記述が難しくなる面があり，記述・理解のしやすさと検証のしやすさは相反する傾向がある。

2.1 VDMとは

フォーマルメソッドの導入として比較的容易な手法として，VDMが挙げられる。この手法はVDM-SLあるいはVDM++と呼ばれる言語で仕様をモデル化し，それをテストなどの検証を通じて，最終的にシステムの高信頼化を図ろうとする手法である。元々は仕様の矛盾のないことを証明によって検証することを目指していたが，現在ではモデル化のツールとしての側面を強調している。

VDMによる開発ツールについては以前はデンマークの企業が提供しており，文法チェック機能や仕様を実行するインタプリタ機能やテスト支援機能が提供されていた。現在は日本の企業が権利を買い取ってサポートしている。そのこともあり，現在は日本語を用いることが可能であるが，日本語を使った記述ではより文書に近い仕様の記述が可能になり，よりVDMを導入しやすい環境が整えられつつあると考えている。

2.2 不変条件

VDMをはじめとするフォーマルメソッドでよく出てくる言葉に不変条件（invariant）という言葉がある。これはシステムを記述する変数，あるいは変数同士の関係に課せられた制約を指している。例えば，交通信号機を考えたときに赤信号のランプの状態を示す変数aと青信号のランプを示す変数bがあったとしてaとbは常に同時に点灯状態にはならない，といった条件である。

不変条件が課せられた場合，それが常に守られるように仕様を書くことになる。もしその不変条件が守れなければ，それは仕様自体が矛盾を持っていることを表す。そのような矛盾がないことを検証するのが，フォーマルメソッドの一つの側面である。

2.3 事前・事後条件

VDMではシステムの状態変数を変更する操作や値を演算する関数を記述する際に，操作や関数を適用する際のパラメータや状態変数の間に要求される事前条件や事後条件を記述することができる。特に操作や関数の具体的な中身を書かずに記述することができるのがVDMの特徴である。例えば正の平方根を求める関数sqrtは以下のように書くことができる。

```
sqrt(x : real) r : real
pre x >= 0 // 入力は正である。
post x = r * r & r >= 0
```

ここでxは入力パラメータ，rは出力を表す。そしてpreに続く文が事前条件，postに続く文が事後条件を表すが，ここでは入力と出力の関係だけが記述され，具体的な計算アルゴリズムについては省略されている。本来はパラメータや出力の事前事後での関係だけが必要な要件であり，その計算手法についてはあくまでもそれを実現する1つの手段でしかないというのがこの記法の意味する所である。このように事前事後の振る舞いで規定するのが仕様の本来の書き方であるというのが，ZやVDMの主張である。とはいえ直接的に演算内容を記述した方が記述が容易な場合も多く，テストをする際にも演算内容が記載されていないと難しいことから直接的な演算内容を仕様として記述することも可能である。

2.4 証明責務

仕様の整合性を証明すると述べたが，証明という言葉の意味するところは，証明責務（proof obligation）と呼ばれる定理を証明することである。この証明責務は仕様が整合性を持っていることを保障するために証明しなければならない条件のことである。例えば

$$f(a,b) = a/b \tag{1}$$

という関数を考える。もしbの値が0ならば結果が不定となる。その結果コンピュータはどうなるかは保証されない。従って例えば別の箇所で

$$g(a,b) = f(a,b) + c \tag{2}$$

のようにこの関数を別の場所で適用することを考えると，引数bが0でないことを確認しなければならない。この

ような条件をチェックするために生成されるのが証明責務である。そのほかにも

(a) 不変条件が保たれていることを確認する条件

(b) 操作・関数において事前条件が満たされた際に事後条件を満たす結果が存在することを確認する条件

など様々な証明責務がある。VDM の仕様記述言語に対して証明責務はどう生成されるかということは既に研究されており，この生成ルールを用いて，形式化された仕様から証明責務を自動生成することが可能である。VDM でもこの証明責務の生成機能が提供されている。

実際に証明責務を生成させると，その大部分は変数がある条件を満たしている時に，関数の事前条件が満たされているかどうかといった簡単な条件である。記述を少しでも誤れば，このような条件は満たされなくなる可能性がある。不変条件が満たされることを確認する条件は一般に証明が難しいのであるが，事前条件に関する証明責務については比較的簡単であり，記述間違いをチェックするためのリストとして捉えることもできる。

2.5 仕様の証明の意義

仕様の証明とは証明責務の証明であることは既に述べた。証明責務の証明により仕様の高信頼化が図れるが，証明責務は仕様が大きくなれば，量がそれに応じて増大するため，できれば自動で証明されるのが望ましい。このような要求は多く，実装されているものも存在する。

線区データベースの検証ではVDM-SLで仕様を記述しHOL[6]と呼ばれる証明用の言語に変換して証明による検証を実施した。ただし試作段階で開発が止まってしまい，現在のところ商用化される見通しは立っていない。

フォーマルメソッドの草分け的存在であるZやその派生であるBにも証明ツールが存在する（Bについては後述）。先ほど述べたHOLもそれ自身を仕様記述に使えば証明ツールを使用できるがHOLで書いた仕様から証明責務を生成する作業は別途必要となる。

なお全ての証明責務を完全に自動で証明することは事実上不可能であり，特に証明責務を否定することは難しい。しかし証明責務がある程度自動で証明されることにより，次の利点が得られる。

(a) 手作業で証明しなければいけない証明責務の数はある程度範囲が絞られる。

(b) 証明されずに残った証明責務を調べることにより，仕様の誤りに気づくことがある。簡単な記述違いでも証明責務は証明できないので，記述違いに気がつくことができる。

従って将来，自動証明が実用的になれば，仕様等の誤りをより発見しやすくなると考えられる。

3．段階的詳細化とB-Method

VDM 等の手法を用いて，仕様を記述することの重要性は既に述べた。しかし繰り返すがシステムの仕様は複数の人々に共有されるものである。従って仕様が単に仕様記述言語で書かれればよいのではなく，いかに理解しやすい仕様を作成するかを考えなければならない。

形式的な仕様と言ってもその扱う対象の抽象度には様々なレベルがある。これまでの研究の議論では，比較的抽象的なレベルでの話が多かったと考えられる。具体的なシステムの実装となると抽象レベルでは考えていなかった様々な条件を考慮しなければならない。そうした条件を実装では満たしているのか，条件の妥当性は検討されているのか，一方で最初の抽象的なレベルでの制約を実装がきちんと満たしているか，といった議論についてはあまり深く行われてこなかった。

このように考慮すべき要素がたくさん存在する時にそれらを一度に記述しようとしても，概して理解しにくい仕様ができやすい。実際に実時間システムの記述を試みたが，仕様が複雑になり分かりにくい結果に終わった。従って最初は簡単に記述し，次第に拡張していき，複雑にしていく手法が考えられる。これが段階的詳細化（stepwise refinement）という考え方である。しかしVDMの開発ツールそのものではそのような開発スタイルはサポートしていないため別のツールの使用を考えた。

段階的詳細化という考え方自体は決して最新のものではなく，既にこのような考え方に従って構築された手法も存在する。B-Method と呼ばれる手法はまさにこの詳細化に則ったものであるため，その検討を行った。

3.1 B-Method とは

B-Method はVDMと同様のモデル化指向手法に分類されるが，その中身は段階的詳細化と証明という構成でVDM とは様相がかなり異なる（図2）。現在ではフランスの企業が開発ツールを提供しているが，VDM のようなインタプリタはなく，証明を行う機能がその重要な位置を占める。また途中の段階の仕様でテストを行うということも難しい。

B では最初に抽象機械（abstract machine）と呼ばれる抽象的な制約を持った仕様として記述する。それを段階的に詳細化する。それぞれの段階の仕様はそれ自身が矛盾のないことを示す必要があるが，この時，詳細化した

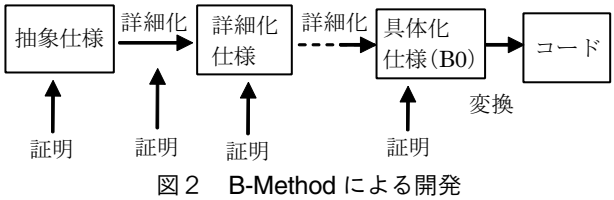

図2 B-Method による開発

仕様が詳細化する前の仕様を満たす必要がある。最後にはプログラムに近い形（B0）に記述する。これはCやADAといった言語に翻訳可能である。ここで各ステップの詳細化では分かりやすい範囲の違いだけを考えることにする。従ってシステムが大規模になれば詳細化の段数は増えるが，何が詳細化によって変わったのかをはっきりさせることができる。

このB-Methodを用いて信号システムの形式的記述を試みることとした。対象としたのはATC（Automatic Train Control）と呼ばれるシステムである。このうちの一部について紹介する。

3.2 例題 - ATCブレーキパターン計算への応用

列車の保安システムの1つであるATC（Automatic Train Control）システムを考える。従来は許容速度を示す信号をレール等を使って車上に伝送し，車上装置がその信号を解読して，許容速度より列車の速度が高ければブレーキをかけるという制御を行ってきた。この許容速度はあらかじめ机上で計算された速度現示図を元に提示されるが，このような地上の信号設備が速度を現示するシステムを地上主体型システムとも呼んでいる。

もう一歩進むと，許容速度ではなく進行可能な位置までの距離（あるいは閉そく数等）を車上に伝送し，車上装置がその情報を元に許容速度を計算してその計算結果に応じてブレーキ制御を行う。これを車上が許容速度を計算するという意味で車上主体型制御と呼ぶ。このようなシステムは既に一部の新幹線等に導入されているが，こうしたシステムにおいて受信したデータを元に許容速度を車上で算出する機能について考えてみる。

車両の位置と許容速度の関係をブレーキ曲線（ブレーキパターン）と呼んでいるが，これを算出するプログラムをこの詳細化によって作成してみる。なお，許容速度には2種類ある。1つは先行列車に衝突しないために停止位置までに停止できる速度を示す停止パターン，もう1つは曲線等で制限速度がある場合にその制限がある位置までに減速できる速度を示す制限パターンであるが，ここでは単純化して停止パターンのみを扱うこととする。

3.2.1 抽象機械

ある停止目標までの距離が与えられた時に，一定の減速度で減速して安全に停止できる速度を計算して返す機能を考える。このように最も抽象的なレベルの仕様を抽象機械と呼ぶことは前述したが，*distance*（*sp*）を速度*sp*からブレーキをかけながら停止するまでに走行する距離とすると，この抽象機械は次のように書くことができる。

SPEED = {0,..,255} ∧ /* 速度（8bit）*/
ta ∈ ℕ ∧ *lo* ∈ ℕ ∧ *brakespeed* ∈ *SPEED* ∧
/* *ta* は停止位置，*lo* は現在位置，*brakespeed* は 許容速度である */
（*ta* ≦ *lo* ⇒ *brakespeed* = 0）∧
/* 停止位置が現在位置より手前であれば許容速度は0である。*/
（*ta* > *lo* ⇒ *distance*（*brakespeed*）≦ *ta* – *lo*）
/* 停止すべき位置が現在位置より先であれば，許容速度から停止するまでの走行距離は停止すべき位置と現在位置の差に等しいか小さい */

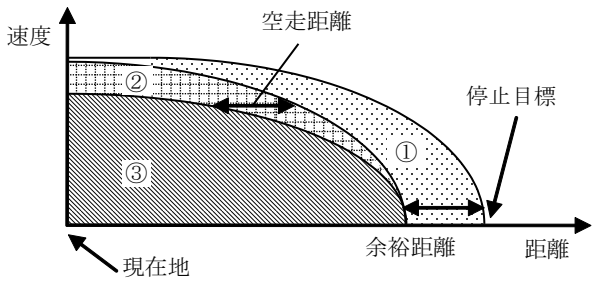

図3 ブレーキ曲線の詳細化

これによりブレーキ曲線は図3中の①～③部に描かれればよいという仕様を書いたこととなる。この仕様についてその中で矛盾がないことを証明する必要がある。

3.2.2 最初の詳細化

通常はぎりぎりの位置に止まるように制御することはまれで停止目標に対して数十m手前に止まるように制御させる。そのため，先ほどの制約に余裕（*margin*）を加える。この場合ブレーキ曲線の制約は次のようになる

SPEED = {0,..,255} ∧ /* 速度（8bit）*/
ta ∈ ℕ ∧ *lo* ∈ ℕ ∧ *brakespeed* ∈ *SPEED* ∧
（*ta* ≦ *lo* ⇒ *brakespeed* = 0）∧
（*ta* > *lo* ⇒ *distance*（*brakespeed*）+ *margin* ≦ *ta* – *lo*）
/* 停止すべき位置が現在位置より先であれば，許容速度から停止するまでの走行距離に余裕距離を加えた距離が停止すべき位置と現在位置の差に等しいか小さい */

前の記述と比べると変わった部分は下線の部分が増えた点である。その結果ブレーキ曲線が取りうる領域が前の段階を満たしていることを証明する必要がある。厳密な証明は可能であるが，簡単に図3で見てみる。新しいブレーキ曲線の取りうる領域は図3で示すと①の部分が除かれた②～③の部分となる。図を見ると分かるようにこの部分は網掛け部全体の一部である。余裕距離を含めて計算されたブレーキ曲線は最初の抽象機械で計算されたブレーキ曲線の制約を満たしていることが分かる。しかも領域が狭くなっていてより具体的になっていることが分かる。

3.2.3 2回目の詳細化

さらに実際の車両を考えるとブレーキをかけ始めてもブレーキが完全に効くまでに時間がかかるため，その間に走る距離（空走距離）を差し引かなければならない。空走時間（ブレーキが効くまでの時間）を *ti* とすると空走距離は空走時間と速度の積になるから

SPEED = {0,..,255} ∧ /* 速度（8bit） */
ta ∈ ℕ ∧ *lo* ∈ ℕ ∧ *brakespeed* ∈ *SPEED* ∧
(*ta* ≦ *lo* ⇒ *brakespeed* = 0) ∧
(*ta* > *lo* ⇒ *distance*(*brakespeed*) + *brakespeed*・*ti* + *margin* ≦ *ta* - *lo*)

この結果さらに②の部分が除かれてブレーキ曲線の範囲としては図中③となっており，より具体的になっている。

3.2.4 3回目以降の詳細化

これ以降さらに制約を厳しくしていく。まずは下り勾配により減速度が小さくなる場合を考慮する。減速度が小さくなることにより，ブレーキカーブに許容される範囲は小さくなる。また信号を受信してから解読するまでの伝送遅れを考慮に入れる。そして，目標位置と現在位置の差を停止目標までの距離とする。

さらに実装上の制約を加えていく。例えば目標距離の値の範囲は理想的には0から∞であるが実装上は許されない。16 bit の値を取るとすれば，0～65535 となる。それを超える場合は目標距離を最大値に丸める処理をする。

最終的に許容されるカーブの領域は一定の領域となる。安全という観点ではその領域中にカーブが納まれば十分ではあるが，実際にはなるべく大きい速度が好ましいので，領域の外周に沿って許容速度を算出すればよい。領域の外周とはその最大値であるから

brakespeed = max（ブレーキカーブに許容される範囲）

最終段の仕様を実装と呼ぶ。ここではそれまでの段階で要求されていることを満たすアルゴリズムを記載することが要求される。今回は上の最大値を求めるのに，2分探索で求めるアルゴリズムを導入した。例えば値の範囲が2進数で0～255とする。まず真ん中である128の時にブレーキカーブが許容されるかどうかを調べる。もし許容されるなら128～255の間，許容されないなら0～127の間と範囲を小さくしてさらに探索してゆくものである。

最終段はそのままコードに変換することが可能で，それに入出力のコードを加えて最終プログラムに仕上げる。

3.2.5 証明

この例での詳細化のステップは12段になった。各詳細化毎に生成された証明課題を表1にまとめた。最終段を除き100～300の証明課題が生成されるが，そのうち大部分は自明な証明課題で，実際の証明作業が必要とされたのは1割程度である。それらのうち自動で証明された（表の自動証明の欄）のが半分程度であった。従って本当に手作業で証明しなければならないもの（手動証明の欄）は全体からすれば非常に少ないため，それに集中すればよいということがわかる。また，最後の詳細化（実装）だけは証明責務の数が他と比べて多いことがわかる。最終段だけはプログラミング言語に直接訳せる関係上，表現に若干の違いがあることが影響している。

表1 証明責務の数

モジュール	証明責務			
	全体	自明	自動証明	手動証明
抽象機械	7	135	2	5
詳細化1	121	117	7	8
詳細化2	129	124	3	2
詳細化3	160	144	4	12
詳細化4	244	223	8	13
詳細化5	223	182	12	29
詳細化6	320	288	15	17
詳細化7	154	131	12	11
詳細化8	155	143	8	4
詳細化9	229	166	2	61
詳細化10	70	56	3	11
詳細化11	55	47	4	4
実装	348	111	90	147
計	2215	1867	170	324

3.2.6 計算結果

ブレーキ曲線の計算例を図4に示す。ここでは現在地点から停止点までの距離をx軸にしているため，図2とグラフの向きが異なるが，きれいな放物線状の曲線が描かれている。

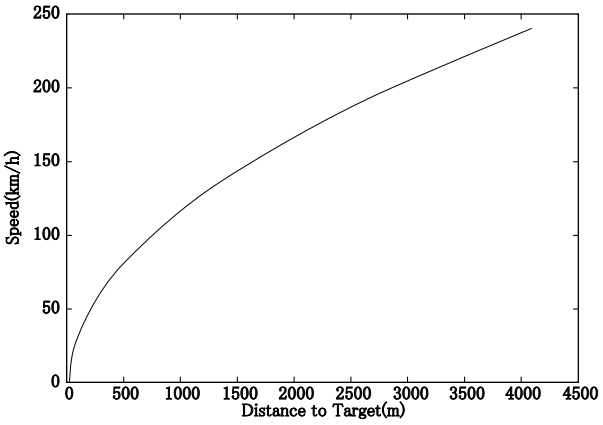

図4 ブレーキカーブの計算例

3.3 B-Method の評価

B-Method を用いた段階的詳細化手法は非常に強力であり，仕様の整合性を検証しつつ，仕様を忠実に満たすコードを生成することができた。また，証明は非常に難しい作業でありながらも，余裕を次第に導入していくという考えがとりやすいブレーキカーブの計算は段階的詳細化の考え方に適合している。しかしながら，この手法を他に広げていくには以下の難点がある。

(a) B-Methodの記述に柔軟性がない。証明を意識して記述する必要がある。仕様を読みやすくすると言う点でかなり問題があるし，一般に利用されるとはいいがたい。

(b) 証明そのものが難しく時間のかかる作業である。証明という特殊な技能を持った人を多数必要とする。信号システムのためにそのような技能集団を必要とすべきかどうかについては現時点では議論の余地がある。

（c）B-Method は抽象的な仕様というよりは比較的低レベルの実装寄りの仕様を記述するのに向いていると考えられる。インタプリタ機能やテスト機能もないため，仕様が考えていることを示しているかどうかについてはあまり検証ができない。

そこで，段階的詳細化と証明の有効性を認めつつも，もう少し導入しやすい方法を検討した。そこで提案するのはやはりVDMのようなモデル化に向く言語を中心に考えることである。

４．VDM を用いた段階的詳細化

VDM を使った場合の段階的詳細化については，特に証明を意識しないで使用していくこととする。つまり，前のモジュールの条件を後ろのモジュールが満たすか満たさないかについては，任意と考える。これはあくまでも仕様の書きやすさからの提言であり，もし安全上重要な部分であれば，B-Method 等を使用して整合性を証明していくことが必要であると考えられる。

また考慮する条件については見落とすことがないように最初にリストアップするというプロセスが入る。特に安全性に関する条件は FTA や FMEA といった分析を行って抽出を行う必要があると考えられる。

こうした検討を実際のモデル化を通じて行った結果，我々が提案するVDMを中心とした信号システムのモデル化手法は以下の通りである。

a) システムを分析し，必要な条件や基本的な要求を洗い出す
b) 最初の段階を VDM にてモデル化する。この際なるべく「何をしてほしいのか」を記載する。モデル化しつつ，そのモデルを検証する。
c) a) の中から必要な条件をピックアップしながら，さらに詳しいモデルをVDMで記述し，検証する。次第に「どうやって実現するか」を記述していくことになる。その際にa) の条件にチェックを入れて，どの条件を考慮に入れたかを把握できるようにする。
d) 仕様がある程度具体化した段階で安全性上，証明が必要な部分を抽出し，その部分については B-Method 等を使用して証明されたコードを生成
e) その他の部分については，VDM で詳細な仕様まで記述する。VDM のコードジェネレータあるいは手作業でコード化する。

現時点では証明に関してはかなりの妥協をしている。将来的には d）で証明してコード化という部分が大きな比重を占めることも考えられる。VDM 仕様に対する証明ツールが提供される可能性があるが，このような手法が大きく取り入れられるには相当の時間がかかると思われる。しかしながらそれ以外の部分については比較的導入が容易と思われる。この提案した手法の有効性について今後検討していく予定である。

５．まとめ

プログラムの信頼性を向上する技術として段階的詳細化について紹介し，これをATCのブレーキパターン計算の例に適用してみた。段階的詳細化を行うことにより，仕様が複雑となっても各段階の違いはわずかであるため，理解がしやすい。そしてその結果得られた曲線は複雑な制約を満たしたものとなった。適用してみると，ブレーキパターンの計算には段階的詳細化という考え方を取りやすいことが判明した。

今後その他の信号システムについても同様の段階的詳細化の手法の適用を検討したいと考えている。この手法が同じようにきれいに適用できるかどうかについては分からない部分も多い。しかし，様々な信号システムに適用できることが判明すれば，その適用の経験を元に鉄道信号へのフォーマルメソッドの適用の指針へとつなげられるのではないかと考えている。そのことにより信号システムの信頼性向上につながることを期待している。

謝　辞

本研究を進めるにあたり，日本信号㈱に例題の提供，モデル化の検討をはじめとする多大なご協力を頂いた。この場を借りて感謝を申し上げる。

文　献

1）寺田, 福田: ディジタルATCデータベースの証明による検証 , 信学技報 , Vol. 101, No. 505, FTS2001-73, 2001
2）寺田，福田：鉄道信号システムへのフォーマルメソッドの適用，鉄道総研報告 , Vol.16, No.7, 2002
3）B, Potter, J. Sinclair, D. Till: An Introduction to Formal Specification and Z, International Series in Computer Science, Prentice-Hall, 1991
4）J. Fitzgerald, P. G. Larsen: Modelling Systems -- Practical Tools and Techniques in Software Development, Cambridge University Press, 1998
5）Abrial J-R: The B-Book: Assigning programs to meaning
6）M. J. C. Gordon and T. F. Melham: Introduction to HOL: A theorem proving environment for highter order logic, Cambridge University Press, 1993
7）寺田，高橋 : 段階的詳細化に基づく信学技報 , 信学技報 , Vol.97, No.98, DC2005-65, 2005