情報処理学会論文誌 Vol.56 No.4 1278–1291 (Apr. 2015)

自動列車制御装置における数値計算へのBメソッド適用の検討

寺田 夏樹[1,a)]

受付日 2014年7月31日, 採録日 2014年12月3日

概要：鉄道の運行を支えている信号保安装置には高い安全性が要求される．その中でソフトウェアの比重は以前に比べて大きくなり，ソフトウェアの安全性を保証するための技術の確立がより重要となっている．数値計算のようなプログラムであってもその正当性を保証する仕組みが必要である．仕様の形式記述を通じてその正当性を定理証明や自動検査によって検証するフォーマルメソッドはその解決策の 1 つと考えられる．本論文では，フォーマルメソッドの 1 手法で，段階的詳細化と定理証明を通じたコード生成に特徴を持つ B メソッドを，ATC（Automatic Train Control，自動列車制御）システムの許容速度の計算に対して適用した事例について述べる．本来ならば平方根などを含む計算が必要である事例であるが，B メソッドで許容される整数演算の範囲で実用的な精度で値の計算が可能であり，プログラムが余裕距離や勾配などの制約を満たすことの証明を行い，プログラムの高信頼化を図ることが可能であることを示す．一方で，while ループでのループ不変条件の使用や，計算途中の値を含めて 32 bit 整数の範囲で計算を行う実装上の制約から，多数の証明責務が発生する状況を示し，数値計算に B メソッドを適用する際に生じる問題点を明らかにする．そのうえで数値計算への B メソッドの適用範囲についても議論を行う．

キーワード：形式手法，B メソッド，数値計算，自動列車制御装置，ブレーキ曲線

Study on Application of B-method to the Numerical Calculation of Automatic Train Protection Systems

Natsuki Terada[1,a)]

Received: July 31, 2014, Accepted: December 3, 2014

Abstract: Signalling equipment requires high reliability. It is required to increase the reliability of software part, even if it is for numerical calculation. It is considered that formal methods are effective to solve the problem, in which the specifications are proved and/or checked for its integrity through the formal description. We applied B-Method, one of the formal methods and characterized by code generation with stepwise refinement and theorem proving, to the calculation of allowable speed of trains concerned with automatic train protection systems. We show that B-Method, in which floating point values are not used, is applicable to numerical calculation with practial precision that usually requires square root, and that the code is proved to satisfy the constraints such as margins and gradients, and gains more reliability. On the other hand, a lot of proof obligations are generated at the implementation phase, related to use of while loop with invariants, and restriction of calculation range within 32 bit integer including intermediate values. We show the problem when B-method is applied to numerical calculations, and discuss the applicable range of B-Method to numerical calculations.

Keywords: formal methods, B-Method, numerical calculation, automatic train protection, brake pattern

1. はじめに

鉄道の運行を支えている信号保安装置には高い安全性が要求される．その信号保安装置においても電子化された機

[1] 公益財団法人鉄道総合技術研究所
Railway Technical Research Institute, Kokubunji, Tokyo 185–8540, Japan
[a)] terada.natsuki.08@rtri.or.jp

器が多く導入されてきている．ハードウェアの観点から見た安全性技術に関しては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にもかかわらず，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発表を行ってきた．特に証明による仕様の整合性の保証に強い関心があり，ATC（Automatic Train Control，自動列車制御，海外では Automatic Train Protection と書かれる）の線区データベースに関する整合性の検証 [1] や単線自動閉そくなどの検証 [2] を実施した．ただし，これらの事例では数値の使用が限定的である．鉄道信号において，速度計算などで数値計算が必要な場合があるが，そのような場合にも生成されたプログラムが仕様を満たすことを保証する仕組みが望まれる．論理演算として記述可能な場合は，モデル検査による自動検査手法の適用も考えられるが，数値計算を主とする場合は，自動検査手法にはなじまない．

そこで本論文では，単線自動閉そくの検証に使用した B メソッド [3] を，ATC システムの許容速度の計算に対して適用した事例について述べる．その結果，B メソッドが整数演算しか使用できないという制限を持つものの，実用的な精度を持ち，要求仕様を満たすことが計算機上で証明されたブレーキ曲線計算プログラムを生成できたことを述べる．このことは，数値計算を主とする場合であっても，B メソッドを用いて仕様を満たすことが保証されたプログラムを生成可能であり，プログラムの高信頼化を図れることを意味する．ここで計算結果が制約を満たすというのは，余裕距離や空走時間，勾配といった各種要素が考慮されているか，計算結果の間に要求される関係が成立するか，あるいは計算結果が一定範囲に収まるかといったことを満たすことを意味する．この証明は，通常の浮動小数点計算に対して計算機上で実施する場合，厳密さを欠いてしまう．

また，この例を通じて，数値計算に B メソッドを適用する際の知見を得た．この検討を通じて数値計算に B メソッドを適用する際の問題点を明らかにし，数値計算の適用範囲についても議論を行う．なお，本論文は文献 [4] の題材に関して，検討を深度化し発展させたものである．

本論文の構成は以下のとおりである．まず，2 章では B メソッドについて，本論文に関連する範囲での解説を行う．続いて，3 章においてモデル化の対象となるブレーキ曲線について説明する．本論文では 2 段階のモデル化を行っているが，4 章で第 1 段階，5 章で第 2 段階のモデル化について説明する．4，5 章のそれぞれでモデルより生成される証明責務の証明に関する検討を行ったが，5 章において，数値計算が可能であるものの，その実装アルゴリズムに関連して証明責務が多く生成されることを示す．6 章ではモデル化で補助的に定義する関数について説明する．またその証明責務の検討から，数値演算そのものに関する証明責務が多く生成される様子を考察する．7 章では実装されたコードにより計算を実施した結果を図示する．8 章ではそれまでの検討を受けて数値計算の適用範囲について考察し，9 章の関連研究を挟んで 10 章で知見を総括した．

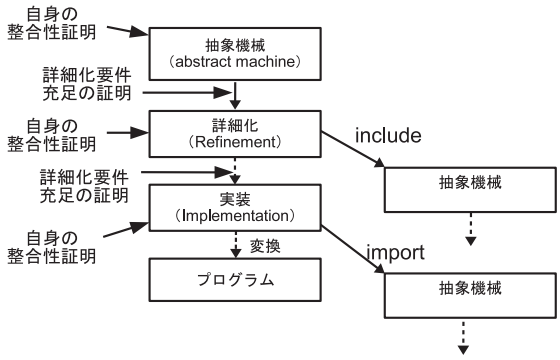

図 1 B メソッドによるプログラム開発
Fig. 1 System development with B.

2. B メソッドの解説

B メソッドとは Abrial が提唱したソフトウェア開発手法であり，仕様の段階的詳細化と証明により，プログラムを生成する手法である．フランスを中心に適用事例が多数報告されている [5], [6].

B メソッドでのシステム開発イメージを**図 1** に示す．ここでは最初に抽象機械（abstract machine）と呼ばれる仕様を記述する．記述に使用する言語を B 言語，あるいは単に B と呼ぶが，Z 記法 [7] と強く関連しており，集合に関する記法が細かく定義されているという特徴がある．抽象という言葉が示すとおり，ここでの仕様記述は，演算結果やアルゴリズムを非決定的に記述することができる．また変数および変数間の関係についてつねに成立する条件を不変条件（invariant）として記述する．この不変条件は operation などを実行した後であっても成立が求められるが，不変条件が保持されるための条件を証明責務（proof obligation）と呼ぶ．この証明責務は仕様から自動生成されるが，証明器などにより証明する必要がある．これはモジュール自身の整合性を保証するための作業であり，図 1 において抽象機械に対して「自身の整合性証明」と呼ばれる部分にあたる．

その後，この仕様を詳細化（refinement）していく．この詳細化とは，演算がより決定的，つまり演算結果の値域がより狭くなるということを意味する．詳細化段階においては，自分自身の整合性の証明だけでなく，変数や演算結果が詳細化前の条件を満たすことを証明する必要がある．この部分は図 1 における「詳細化要件充足の証明」である．最終段階は実装（implementation）段階と呼ばれ，プログラムに直接変換可能な形としての記述となる．ここでの演算は決定的なアルゴリズムとする必要があり，B のサ

```
MACHINE
  MA
INCLUDES
  SS.Minc
SETS
  AA; XX = {x1, x2}
VARIABLES
  aa, xx
INVARIANT
  aa : AA & xx : XX
INITIALISATION
  aa :: AA || xx :: XX
OPERATIONS
 Op_A = BEGIN aa :: AA || xx :: XX END;
 Op_B = SS.Op_C
END
```

図 2 B メソッドの抽象機械モジュールの概要
Fig. 2 Outline of abstract model in B.

ブセット（B0 言語）を使用して記述する．この実装段階からコード生成器によりコードを得ることができる．

B の抽象機械モジュールの概略を**図 2** に示す．先頭の MACHINE が抽象機械の宣言であり，名前が MA であることを宣言している．SETS は集合の宣言である．図 2 の XX のような列挙型集合のほか，deferred set と呼ばれる集合も定義可能である．これは集合名だけを宣言し，その実装を詳細化以降に委ねるもので，図 2 の AA がそれに相当する．これは実装段階で整数の有限区間として実装される．

VARIABLES は抽象変数（abstract variable）の宣言である．その型については，INVARIANT の中で定義する．型は通常の整数型や集合の要素を示す定義だけでなく，集合の積や関数型なども定義できる．ここで「抽象」といっているのは，仕様の定義のために使用するということであり，実装される形態とは異なっていてよい．実装される具象変数（concrete variable）も定義できるが，これについては列挙型，整数型，bool 型あるいはその配列のみが許される．

INVARIANT は不変条件を示す．変数の型の宣言のほか，変数の間につねに成り立つ関係を記述する．ここで:は左辺が右辺の集合の要素であることを示す．ここで宣言された不変条件は，INITIALISATION に示されるインスタンス生成時の初期化実行後，また OPERATIONS の中に記述される operation 実行後にはつねに満たされている必要がある．

初期化や operation の中で記述できるのは一般化代入（generalized substitution）と呼ばれ，代入の概念を拡張したものである．**表 1** に主なものを示す．そのほかに IF 条件（IF..THEN..ELSE..END）など，複数の一般化代入を組み合わせて大きな一般化代入にするものがある．図 2 の aa :: AA では aa に AA の 1 要素を非決定的に代入する．||は 2 つの代入を同時に行うことを意味するが，同時に同じ変数に代入を行うことはできないので，その場合は IF 条件などを併用する．また抽象機械においては逐次代入が使用できず，実装段階では逆に同時代入が使用できない．

表 1 B における一般化代入の例
Table 1 Part of generalized substitutions in B.

種類	記述例	概要
恒等代入	skip	何もしない
等価代入	a := b	単純な値の代入
要素代入	a :: A	集合の 1 要素を非決定的に代入
充足代入	a :(X(a))	X(a) を満たす値を非決定的に代入
逐次代入	A; B	A を実行した後，B を実行する
同時代入	A || B	複数の代入を同時に並行して行う
ブロック代入	BEGIN…END	逐次代入や同時代入をくくる

不変条件を I，初期化や operation の一般化代入を S とすると，抽象機械における証明責務は $I \Rightarrow [S]I$ と記述される [3]．ここで $[S]I$ は I を S によって変換するという意味であり，たとえば I が $x = y$ の場合に S が $x := a$ であれば $[S]I$ は x を a で置き換えた $a = y$ ということになる．

一方，詳細化に関する証明責務は 2 つある [3]．1 つは初期化に関するもので，詳細化前後の変数の関係を J，詳細化前後の初期化を B, C とすると $[C]\neg[B]\neg J$ で与えられる．もう 1 つは operation に関するもので，不変条件を I，詳細化前の operation が PRE P THEN K END（事前条件付き代入：P 成立時に K を実行する），詳細化後が PRE Q THEN L END の場合に $(I \wedge J \wedge P) \Rightarrow (Q \wedge [L]\neg[K]\neg J)$ で与えられる．

また，構造化の仕組みとしては include リンクや import リンクと呼ばれるものがある．抽象機械や詳細化段階では図 2 で示したように外部モジュールの抽象機械を INCLUDE 文で宣言することにより，外部モジュールの operation や変数を部品として利用することが可能となる．include するモジュール Minc の前に prefix をつけないで宣言すると，operation や変数の名前が自分自身の変数や operation と衝突した場合にエラーとなるが，たとえば図 2 のように SS と prefix を付けると，この operation や変数も図 2 の SS.Op_C のように SS を付けて参照することになり，名前の衝突をさけることができる．同様に実装段階においても IMPORT 文での宣言により利用可能である．

include されるモジュールの変数の値の変更は，同じモジュール内の operation を通じてのみ行える．また include される operation の振舞いは抽象機械の定義によるため，include する側からはその operation の詳細化段階で加えられたアルゴリズムには立ち入ることができない．

なお，B メソッドの開発ツール Atelier B [8] は 32 bit バイナリで配布されている．よって，実装段階で使用できる整数値は 32 bit 整数の範囲となっている．このことが数値計算上の制約になることについては 6.3 節で後述する．

3. ATC ブレーキ曲線について

今回の検討の対象である ATC システムのブレーキ曲線について紹介する．ATC とは列車の在線状況などに応じ

て決まる許容速度以下に列車の速度を制御する保安システムである．従来は許容速度を示す信号をレールなど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の計算をもとに提示されるが，このように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，または区間の数などを車上に伝送し，車上装置がその情報をもとに許容速度を計算し，結果に応じてブレーキ制御を行う．このようなシステムを車上が許容速度を計算するという意味で車上主体型システムと呼び，新幹線などに導入されているが，ここでは，そのうち受信したデータをもとに許容速度を車上で算出する機能について考える．

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作成してみる．通常はブレーキの種類に 2 種類あり，1 つは通常使用する常用ブレーキ，もう 1 つは減速度がより大きく，急停止する必要がある場合に使用する非常ブレーキである．さらに常用ブレーキが作用する前に運転士にブレーキ操作を促すために予告をする警報パターンについても考える．すなわち 3 種類のブレーキ曲線を考える．これらの関係は警報パターンより常用パターン，常用パターンよりも非常パターンの速度がつねに高いが，その関係が維持されていることも検証する．

なお，許容速度には別の分類があり，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 つは分岐など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．

ここで証明すべき要件を，以下のように整理した．

要件 1 現在位置，停止すべき位置が与えられた場合に安全に停止できる常用速度，警報速度を提示する．停止すべき位置を超えている場合は速度 0 とする．

要件 2 停止位置の前に余裕距離を設けること．

要件 3 計算には空走時間（ブレーキをかけ始めてから停止するまでの時間）を考慮すること．

要件 4 非常速度が常用速度以上であり，常用速度が警報速度以上であること．

一方，以下の制約を設けた．

制約 1 最高速度を MaxSpeed とする．

制約 2 空走時間は，非常空走時間 ≤ 常用空走時間 ≤ 警報空走時間とする．

制約 3 余裕距離は，非常余裕距離 ≤ 常用余裕距離 ≤ 警報余裕距離とする．

制約 4 現在位置と目標位置の間は MaxDist 以下とする．

制約 5 減速度は一定．常用減速度は非常減速度以下とする．警報パターンでは常用減速度を使用する．

```
MACHINE
  BrakeSpeed
VARIABLES
  ta, lo, normalspeed, emergencyspeed,
  warningspeed, distance, ...
INVARIANT
  ta : NAT & lo : NAT &
  distance : NAT & normalspeed : SPEED &
  (ta <= lo => normalspeed  = 0) &
  (ta > lo => distance <= ta - lo) &
  emergencyspeed >= normalspeed &
  normalspeed >= warningspeed ...
OPERATIONS
  SetSpeed(tt, ll) =
  PRE tt : NAT & ll : NAT
  THEN
    ta := tt || lo := ll ||
    IF tt <= ll THEN
      normalspeed := 0 || distance := 0 ||...
    ELSE normalspeed, distance, ... :(
      distance : NAT & normalspeed : SPEED &
       (tt <= ll => normalspeed = 0) &
       (tt > ll => distance <= tt - ll) ...)
    END
  END;
  sp <-- Normalspeed =
    sp := normalspeed;
```

図 3 ブレーキ曲線の抽象機械
Fig. 3 Abstract machine of brake pattern.

制約 6 下り勾配では勾配に応じて減速度を小さくする．

制約 1〜6 を満たしつつ要件 1〜4 を満たすプログラムをこれから生成する．

4. 最初のモデル化（関係式の導出）

ブレーキ曲線計算のモデル化は 2 段階に分けて実施した．最初に詳細化によりブレーキ曲線に求められる式を導出した．続いてその式に基づき，実際に計算を行う実装につなげた．本章では，最初の関係式の導出部分を示す．

4.1 抽象機械

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，**図 3** に常用ブレーキ速度 `normalspeed` に関する箇所を中心にその一部を示した．`distance` は，常用ブレーキをかけてから停止するまでの距離，`ta` は停止すべき位置，`lo` は現在位置を何らかの座標系で記述するものであり，有限の自然数である `NAT` 型とした．その差 `ta - lo` は停止位置までの距離となるが，これが負になる場合にはそもそも走ることが認められないため

```
ta <= lo => normalspeed  = 0
```

となる．これが正となれば

```
ta > lo => distance <= ta - lo
```

これらは要件 1 を示している．また不変条件の下 2 行は要件 4 に対応する．さらに `normalspeed : SPEED` の `SPEED`

は 0 から MaxSpeed までの数値の集合であり，これにより最高速度が MaxSpeed となり，制約 1 に対応する．まだこの段階では distance と normalspeed の関係は示されていないが，後の段階で関係を決めていくこととなる．

ここで変数 normalspeed に対して，その値を得る Normalspeed という operation を定義している．これは，オブジェクト指向プログラミングで値を得るメソッドを定義するのと同様で，実装段階で operation を通じて値が取得でき，実装しやすくなる．これは，実装段階では抽象変数は 5.2 節で説明するループ処理の不変条件としてしかアクセスできないために行うものである．ただし変数に変更がないので証明責務 $I \Rightarrow [S]I$ はつねに真となる．

また，入力を受ける部分は SetSpeed とした．引数の型は事前条件付き代入 PRE .. を用いて事前条件として記述する．ここでは充足代入を用い，変数が不変条件を満たすように代入していることから，証明責務は成立する．

4.2 最初の詳細化と詳細化要件

次に要件や制約を詳細化の過程で盛り込んでいく．最初に要件 2 を考慮する．通常は停止目標に対して余裕を持たせ，数十 m 手前に止まるように制御させる．そこで前節の制約に余裕距離（margin）を加える．ここで margin は NAT 型とする．その結果，抽象機械では停止距離 distance であったが，余裕距離を加えた distance + margin が ta - lo より小さいことを要求することになる．**図 4** に詳細化を示す．これで要件 2 が追加されたことになる．

これが抽象機械に対して詳細化要件を満たすことを示す必要がある．前述のとおり，初期化の詳細化に関する証明責務は $[C]\neg[B]\neg J$，operation の詳細化に関する証明責務は $(I \wedge J \wedge P) \Rightarrow (Q \wedge [L]\neg[K]\neg J)$ で与えられるが，ここでは opeartion の詳細化要件を SetSpeed で説明する．ただし P，Q は同一であるため省略する．区別のため詳細化後の変数に $'$ をつけると J は以下の 3 式で示される．

$$ta = ta' \wedge lo = lo' \wedge normalspeed = normalspeed' \wedge distance = distance' \quad (1)$$

$$ta' \leq lo' + margin \Rightarrow normalspeed' = 0 \quad (2)$$

$$ta' > lo' + margin \Rightarrow distance' + margin \leq ta' - lo' \quad (3)$$

K は以下の 3 式を満たすように値を代入することを意味し，

$$ta \leq lo \Rightarrow normalspeed = 0 \quad (4)$$

$$ta \leq lo \Rightarrow distance = 0 \quad (5)$$

$$ta > lo \Rightarrow distance \leq ta - lo \quad (6)$$

L は式 (2)，(3) かつ

$$ta' \leq lo' + margin' \Rightarrow distance' = 0 \quad (7)$$

を満たすように値を代入することを意味する．I は式 (4)，(6) となる．そうすると $[K]\neg J$ は式 (4)，(5)，(6) が成り立つすべての値に対して $\neg((1)\wedge(2)\wedge(3))$ となることを意味する．この否定は式 (4)，(5)，(6) を満たす値の中に $((1)\wedge(2)\wedge(3))$ を満たす値が存在するという意味となり，それに L による変換を適用するから，結局この証明責務は，すべての式 (2)，(3)，(7) を満たす値に対して，式 (1)〜(6) を満たす値が存在することを意味する．いいかえると，詳細化後の任意の演算結果に，詳細化前の何らかの演算結果が対応している必要がある，というのが operation の詳細化要件である．なお，詳細化前の任意の演算結果に，必ずしも詳細化後の演算結果が対応する必要はない．

詳細化要件のうち式 (4)，(5)，(6) は以下の理由で成立する．$margin \geq 0$ により式 (2) が成立すれば式 (4) が，式 (7) が成立すれば式 (5) が成立する．式 (6) に関しては，$ta' > lo' + margin$ の範囲では，式 (3) と $margin \geq 0$ により成立．また $ta' > lo'$，$ta \leq lo$ の範囲では式 (7) により式 (6) の $distance$ が 0 となるからこの場合も成立する．よってこの operation は詳細化要件を満たすことが分かる．

なお，事前条件が異なる場合には，詳細化前の事前条件 P が成立する場合に詳細化後の Q が成立する必要があるという解釈が加わる．事前条件 P が成立する範囲において，詳細化後の演算結果に対応して，詳細化前の演算結果が存在することを示せばよい．また，初期化に関しては事前条件や I を抜いた形であり，ほぼ同様に考えることができる．

各段階において詳細化要件が満たされる場合，詳細化段階の演算結果に対応して，その前段階の演算結果が存在する．最終的なプログラムの実行結果は決定的なアルゴリズ

```
MACHINE
  BrakeSpeed_1
REFINES
  BrakeSpeed
VARIABLES
  ta, lo, normalspeed, distance, ...
ABSTRACT_CONSTANTS
  margin
INVARIANT
  (ta <= lo + margin => normalspeed  = 0) &
  (ta > lo + margin => distance + margin <= ta - lo)...
OPERATIONS
  SetSpeed(tt, ll) =
  PRE tt : NAT & ll : NAT THEN
    ta := tt || lo := ll ||
    IF tt <= ll + margin THEN
      normalspeed := 0 || distance := 0 ||...
    ELSE normalspeed, distance, ... :(
      distance : NAT & normalspeed : SPEED &
     (tt <= ll + margin => normalspeed = 0) &
     (tt > ll + margin => distance + margin <= tt - ll))
    END
  END;
  sp <-- Normalspeed =
    sp := normalspeed;
```

図 4 ブレーキ曲線モデルの第 1 の詳細化
Fig. 4 First refinement of brake pattern.

表 2 詳細化の内容
Table 2 Precise of refinement.

モジュール	詳細化内容
抽象機械	
詳細化 1	余裕距離 margin を導入.
詳細化 2	空走時間を導入（delay/36000）．さらに distance を非常，常用，警報ブレーキ距離 e_distance，n_distance，w_distance に分離． distance = e_distance & e_distance <= n_distance & n_distance <= w_distance
詳細化 3	delay をプログラム処理周期 interval とブレーキ曲線ごとの応答時間 bdelay_e，bdelay_n，bdelay_w に書き換え．（delay = interval + bdelay_e & bdelay_e <= bdelay_n & bdelay_n <= bdelay_w）（制約 2） margin を非常，常用，警報余裕 margin_e，margin_n，margin_w に分離．(margin = margin_e & margin_e <= margin_n & margin_n <= margin_w)（制約 3）
詳細化 4	ta - lo = loc と書き換え．
詳細化 5	最大距離を MaxDist とし，loc と比べて小さい値を loc2 と定義．（制約 4）
詳細化 6	e_distance，n_distance，w_distance を Distance.mch の関数型変数 distance_e，distance_n で記述．（制約 5）
詳細化 7	emergencyspeed,normalspeed,warningspeed を max 関数を使って記述． not(loc2 <= margin_n) => normalspeed = max({sp | sp : SPEED & sp * (interval + bdelay_n) / 36000 + distance_n(sp,grad) <= loc2 - margin_n}))
詳細化 8	EmergencySpeed，NormalSpeed，WarningSpeed の一般化代入を書き換え．
実装	実装．実際には CalBrakeSpeed に引き継ぐ．

ムによるものであり，詳細化要件をさかのぼることで，この実行結果が各段階に導入された条件を満たすことが保証される．これが詳細化要件を証明する意義である．

4.3 さらなる詳細化

次に要件 3 を考慮する．ブレーキをかけ始めてもブレーキが完全に効くまでの時間（空走時間）があるため，その間に走る距離（空走距離）を考慮する．空走時間を ti とすると常用ブレーキの空走距離は normalspeed * ti であり，停止までの距離は distance + normalspeed * ti + margin となる．これにより要件 3 が織り込まれる．実際には ti は**表 2** の詳細化 2 に示したように delay/36000 で表現している．36000 の意味については後述する．

これ以降さらに詳細化をすすめる．表 2 に実際に行った詳細化を示した．まず先ほどの遅延時間 delay をプログラムの処理周期 interval と実際の応答時間 bdelay_n に分解した．ここで応答時間については非常ブレーキに対する応答時間を bdelay_e として，常用ブレーキに対する応答時間 bdelay_n はそれよりも大きいものとした．つまり delay <= interval + bdelay_n としている．これにより制約 2 が導入されている．またブレーキパターンごとの余裕距離を差をつけて導入することで制約 3 を導入した．また ta - lo を loc と置き換えた．

さらに実装上の制約を加える．目標距離の値の範囲は理想的には 0 から ∞ であるが，16 bit 整数の値をとるとすれば，0〜65535（符号付き整数の場合は 32767 まで）となる．これでも m 単位で 65 km（または 32 km）まで表現できるため実用的には問題はないが，目標距離の最大値を MaxDist とする処理をする．loc が MaxDist 以下の場合はそのままの値，loc が MaxDist を超える場合には MaxDist を loc2 とした．これにより制約 4 を導入したのが詳細化 5 である．

図 3 の段階では単位について決めていなかったが，詳細化する段階で単位を決定している．まず走行可能距離や余裕距離については外部から与えられるものであるが，その単位は m とした．これは実際に与えられる精度が m 単位であるためである．有効数字としては 3 桁あればよく，1 km を超える場合には 10 m 単位としてもよいが，ここは m で統一した．速度は計算結果として求まるものであり，運転士には km/h 単位で示されるが，360 km/h=100 m/s までの範囲で秒速に直したときの有効数字を 3 桁とするため，単位は 0.1 km/h とした．空走時間や処理周期などの時間の単位は実際の処理で定義しやすいよう ms とした．単位が決まればそれに応じて関係式も変わってくる．たとえば速度 $v \times 0.1$ km/h，空走時間 $t_i$ms としたときの空走距離（m）は $v/10 \times 1000/3600 \times t_i/1000 = v \cdot t_i/36000$ と表現される．それが前述の 36000 の意味である．結果として以下の制約が追加されたこととなる．

制約 7 距離の単位は m 単位で与える．
制約 8 計算結果は 0.1 km/h 単位とする．
制約 9 空走時間については ms 単位で与える．

最終的に許容される曲線の領域は一定の領域となる．これが求まっているのが詳細化 6 であり，そのときの制約は

```
((loc2 <= margin_n) => normalspeed = 0) &
(not(loc2 <= margin_n) =>
normalspeed * (interval + bdelay_n) / 36000 +
distance_n(normalspeed,grad) <= loc2 - margin_n)
```

である．ここで distance_n は Distance という別の抽象機械に定義される関数型の変数で normalspeed や勾配を表す grad に依存して値が変わることを意味している．distance_n の定義については 6 章で説明する．

安全という観点ではこの詳細化 6 の制約が満たされる領

```
REFINEMENT
  BrakeSpeed_7
REFINES
  BrakeSpeed_6
VARIABLES
  loc2, grad, normalspeed, ...
INVARIANT
  ((loc2 <= margin_n) => normalspeed = 0) &
  (not(loc2 <= margin_n) =>
    normalspeed = max({sp | sp : SPEED &
      sp * (interval + bdelay_n) / 36000 +
      distance_n(sp, grad) <= loc2 - margin_n})) & ...
OPERATIONS
  SetSpeed(tt, ll, gg) =
  PRE tt : NAT & ll : NAT & gg : GRAD THEN
    grad := gg ||
    IF tt < ll THEN
      loc2 := 0 || normalspeed := 0 || ..
    ELSE
      ANY xx WHERE xx : NAT &
        (tt - ll <= MaxDist => xx = tt - ll) &
        (not(tt - ll <= MaxDist) => xx = MaxDist)
      THEN
        loc2 := xx ||
        IF xx <= margin_n THEN normalspeed := 0
        ELSE normalspeed := max({sp | sp : SPEED &
           sp * (interval + bdelay_n) / 36000 +
          distance_n(sp, gg) <= xx - margin_n})
        END
      END
    END
  END;
  sp <-- Normalspeed =
    IF loc2 <= margin_n THEN sp := 0
    ELSE sp := normalspeed
    END;
```

図 5 詳細化 7 における不変条件と operation
Fig. 5 Refined operations and invariant in 7th refinement.

域に曲線が収まれば十分だが，実際にはなるべく大きい速度が好ましい．そこで，詳細化 6 の制約を満たす値の中の最大値を，詳細化 7 における normalspeed の値とした．詳細化 7 を図 5 に示す．ここでは制約 5 が導入されている．また，Normalspeed では，loc2 <= margin_n の範囲では normalspeed = 0 であるため，0 を直接出力している．

SetSpeed の中の ANY xx WHERE..THEN..END は非決定的選択であり，WHERE 以下を満たす xx を用いて THEN 以降の代入を行うものである．また，SetSpeed の引数が 3 つとなっているが，B メソッドでは operation の入出力の形を詳細化で変えられないため，このように引数を増やす場合は，抽象機械にまでさかのぼって修正する必要がある．ここで使用されている GRAD は勾配を表す集合である．また，勾配を保持する変数 grad もさかのぼって定義した．

詳細化 8 では normalspeed の関係式を使って代入文の右辺を置き換え，図 6 を得ている．この時点で normalspeed という変数を使用せずに，結果が得られている．そこで

```
REFINEMENT
  BrakeSpeed_8
REFINES
  BrakeSpeed_7
VARIABLES
  loc2, grad
OPERATIONS
SetSpeed(tt, ll, gg) ==
PRE tt : NAT & gg : GRAD & ll : NAT
THEN
   grad := gg ||
   IF tt < ll THEN loc2 := 0
   ELSIF tt - ll <= MaxDist
   THEN loc2 :=  tt - ll
   ELSE loc2 := MaxDist END
END;
sp <-- Normalspeed =
  IF loc2 <= margin_n THEN sp := 0
  ELSE
      sp := max({sp1 | sp1 : SPEED &
      sp1 * (interval + bdelay_n) / 36000 +
        distance_n(sp1, grad) <= loc2 - margin_n})
  END;
```

図 6 詳細化 8 における operation
Fig. 6 Refined operations in 8th refinement.

SetSpeed においても normalspeed を使用しない形とした．

4.4 実装

最終段の仕様である実装では，それまでの段階の要求を満たすアルゴリズムの記述が要求される．そのアルゴリズムで計算結果が得られれば，前述したとおり最上位の抽象機械まで詳細化要件をさかのぼることにより，各段階に導入された制約を満たす変数値の組の存在が証明できる．

ただ，このまま実装の記述を行った場合，他のモジュールからこのモジュールを利用する場合には，抽象機械で記述した条件だけが見え，途中で導入された詳細な条件が見えない．そこで，このモデル化は冒頭にも述べたように関係式が求まったところでいったん終了し，さらなる詳細化は抽象機械 CalBrakeSpeed を定義して行うこととした．図 7 に示す．SetSpeed に関しては，前の段階でほぼ実装に近い形であり，局所変数 ii を使用した点を除くとほとんど変わっていない．一方，Normalspeed では CalBrakeSpeed に定義された，Normal_Speed という operation を用いるという表現となっている．引数が 2 つあるが，これは停止位置までの距離と勾配となる．また cmargin_n は常用ブレーキに関する余裕距離であり，CalBrakeSpeed に定義されている定数である．CalBrakeSpeed の定義については次章で与えるが，その定義についてはこれまでのモジュールで定義された仕様を満たす必要がある．

ここでは実装といっても，他のモジュールを用いるという表現であり，最終的な計算アルゴリズムは実装されていない．実装については次章の CalBrakeSpeed の定義の中

```
IMPLEMENTATION
  BrakeSpeed_9
REFINES
  BrakeSpeed_8
IMPORTS
  CalBrakeSpeed
CONCRETE_VARIABLES
  loc2, grad
PROPERTIES
  interval = c_interval & margin_n = cmargin_n &
  bdelay_n = delay_n & ...
OPERATIONS
  SetSpeed(tt, ll, gg) =
  VAR ii IN
    grad := gg; ii := tt - ll;
    IF ii < 0 THEN loc2 := 0
    ELSIF ii <= MaxDist THEN loc2 :=  ii
    ELSE loc2 := MaxDist END
  END;
  sp <-- Normalspeed =
    IF loc2 <= cmargin_n THEN sp := 0
    ELSE sp <-- Normal_Speed(loc2, grad)
    END;
```

図 7 第 1 段階のブレーキ曲線計算の実装
Fig. 7 First implementation of brake pattern calculation.

表 3 第 1 段階における証明の数
Table 3 The number of proofs for the first model.

module	全体	自明	自動証明	手動証明
抽象機械	145	137	6	2
詳細化 1	116	105	7	4
詳細化 2	101	95	3	3
詳細化 3	133	113	13	7
詳細化 4	199	172	10	17
詳細化 5	180	141	17	22
詳細化 6	121	107	10	4
詳細化 7	269	238	23	8
詳細化 8	107	86	8	13
実装	177	111	52	14
計	1548	1305	149	94

で実施していく．

4.5 証明責務の生成と証明

各詳細化ごとに生成された証明責務の数を**表 3** にまとめた．各段階において 100〜300 の証明責務が生成されているが，そのうち大部分は自明，つまり $X \vdash X$ のような証明責務で，実際の証明作業が必要とされたのはおおむね 1 割以下である．これらは証明器により自動で証明したり，対話的に証明したりする必要がある．証明作業が必要なもののうち自動で証明された（表の自動証明の欄）のが 6 割程度であった．実装段階で証明器による証明を行った数が増えているのが目を引くが，それでも倍程度であった．

なお，この時点では CalBrakeSpeed や Distance が実装されていない．要件 1〜4 を満たす計算は CalBrakeSpeed

```
MACHINE
  CalBrakeSpeed
VARIABLES
  normal_speed, ···
INCLUDES
  Distance
INVARIANT
  normal_speed : LOCATION --> (GRAD --> SPEED) &
  !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
  (loc0 <= cmargin_n => normal_speed(loc0)(gr) = 0) &
  (not(loc0 <= cmargin_n) =>
    normal_speed(loc0)(gr) = max({sp | sp : SPEED &
      sp * (c_interval + delay_n) / 36000 +
      distance_n(sp, gr) <= loc0 - cmargin_n})) &  ...
OPERATIONS
  sp <-- Normal_Speed(loc0, gr) =
  PRE loc0 : LOCATION & gr : GRAD
  THEN
    sp := normal_speed(loc0)(gr)
  END;
```

図 8 第 2 段階のブレーキ曲線抽象機械モデル
Fig. 8 Second abstract machine of brake pattern.

や Distance が実装されて初めて実現可能となる．

5. 再モデル化（計算アルゴリズムの決定）

次に関係式の実装を行う．ここでは，それをプログラムコードのイメージに近い形へと変換していく．

5.1 抽象機械の記述

抽象機械 CalBrakeSpeed の仕様を**図 8** に示す．BrakeSpeed の詳細化では段階を踏んだが，最初から詳細な定義ができるのであれば，ここからスタートしてもよい．ここでも Distance という別の抽象機械を include し，その中の distance_n という関数型の抽象変数を利用している．その定義を**図 9** に示すが，詳細については 6 章で説明する．ここでは Distance を prefix なしで宣言しているため，名前の付け替えはない．この関数は速度，勾配に対して，走行距離を返す関数を示している．BrakeSpeed では distance を単純に自然数としていたが，この distance_n は LOCATION * GRAD --> NAT の形式を持つ 2 つの引数をとる関数（全関数）として定義している．この関数についても最終的には実装する必要があり，実際には速度の 2 乗に比例することになるが，その定義は 6.2 節で与える．なお，LOCATION は停止位置までの距離を示す集合である．

5.2 実装段階の記述（2 分探索によるアルゴリズム）

この仕様についても，詳細化を経て実装段階に至る．詳細化では，変数に関する制約の追加は行わず，operation の Normal_Speed の normal_speed を図 8 の max(..) に置き換える程度の変換を行った．実装段階においては仕様記述の仕組みがこれまでとは少々異なる．実装段階で使用できる変数は具象変数のみである．プログラムに変換可能な

```
MACHINE
  Distance
VARIABLES
  distance_n, ...
INVARIANT
  distance_n : SPEED * (0..(MaxSpeed * MaxSpeed)) &
  !gr.(gr : GRAD => distance_n(0, gr) = 0) &
  !(sp1, sp2, gr).(sp1 : SPEED & sp2 : SPEED &
    gr : GRAD => (sp1 >= sp2 =>
      (distance_n(sp1, gr) >= distance_n(sp2, gr)))) &
  !(sp, gr1, gr2).(sp : SPEED &
    gr1: GRAD & gr2 : GRAD => (gr1 >= gr2 =>
      distance_n(sp, gr1) >= distance_(sp, gr2)))
OPERATIONS
  dn <-- NormalDistance(sp, gr) =
  PRE sp : SPEED & gr : GRAD
  THEN
    dn := distance_n(sp, gr)
  END;
```

図 9 停止距離関数を定義する抽象機械

Fig. 9 Abstract machine of function that returns distance to stop.

範囲に変数型が制限されており，B0 言語と呼ばれる B のサブセットが使用される．具体的には変数は整数型，bool 型，列挙型の値しか使用できない．なお図 7 についても，B0 言語に従っている．配列は B0 言語に含まれないが，1 次元および 2 次元の配列に関するライブラリが提供されており，これにより使用可能である．集合表現を実装する場合は，配列表現に変換する必要がある．

ここで，集合の和を求める場合など，配列の各要素に順にアクセスするためには繰返し演算が必要となる．B メソッドにおいては，繰返し演算を行うために，while ループが提供されている．なお，for ループに相当する表現は提供されないため，繰返し演算はこの while ループの表現によるしかない．構文は以下のとおりとなっている．

```
WHILE P DO S INVARIANT I VARIANT V END
```

I では，ループにおける不変条件（局所変数の型や値の範囲の宣言を含む）を記述する．V ではループ実行ごとに減少する正整数式（variant）を定義する．これはループ実行が無限ループに陥らずに終了することを証明するために必要である．ループ変数の初期化式はループの外で与える．

上のように書かれた場合，この最弱事前条件，すなわち，演算が実行可能な条件は以下で与えられる [3].

$$I \wedge$$
$$\forall x \cdot (I \wedge P \Rightarrow [S]I) \wedge$$
$$\forall x \cdot (I \Rightarrow V \in N) \wedge$$
$$\forall x \cdot (I \wedge P \Rightarrow [n := V][S](V < n))$$

1 行目は invariant が成立すること，2 行目は P が成立する場合は，ループ内で一般化代入 S を実行しても invariant が依然として成立することを意味する．3 行目は variant が自然数であることを示し，最後の 4 行目は $V < n$ の V を

```
sp <-- Normal_Speed(loc0, gr) =
 IF loc0 <= cmargin_n THEN sp := 0
 ELSE VAR ll, hh, ii IN
   ll := 0; hh := MaxSpeed; ii := 0; sp := 0;
   WHILE ll /= hh DO
     sp := (ll + 1 + hh) / 2; ii := 0;
     ii <-- NormalDistance(sp, gr);
     ii := sp * (c_interval + delay_n) / 36000 +
       ii - loc0 + cmargin_n;
     IF ii <= 0 THEN ll := sp
     ELSE sp := sp - 1; hh := sp END
   INVARIANT
     ii : INT & ll : 1..MaxSpeed &
     hh : 1..MaxSpeed & sp : ll..hh &
     ll : {sp1 | sp1 : 1..MaxSpeed &
       sp1 * (c_interval + delay_n) / 36000 +
       distance_n(sp1, gr) <= loc0 - cmargin_n} &
     (hh + 1) /: {sp1 | sp1 : 1..MaxSpeed &
       sp1 * (c_interval + delay_n) / 36000 +
       distance_n(sp1, gr) <= loc0 - cmargin_n}
   VARIANT
     hh - ll
   END
 END
END;
```

図 10 ブレーキ曲線計算の実装

Fig. 10 Final implementation of brake pattern calculation.

S によって変換し，その後 n を元の V で置き換えることから，S 実行後の V が減少することを要求している．よって P が成立し続ける間は V が減少していくが，N を 0 より小さくはできないので，その前に P が不成立となる必要があり，それにより処理が終了することを保証できる．最弱事前条件が成り立てば演算結果が得られることになるが，その結果については不変条件を満たす必要がある．

B メソッドの教典である B-book [3] には様々なアルゴリズムの記述例が提供されている．最も単純なアルゴリズムは 0 から数字を 1 つずつ上げていき，許容範囲を超えたらループを脱出するというものである．しかし，この場合は結果の値が大きくなると繰返し回数が増え，実用的ではない．これに対し B-book では 2 分探索も紹介されている．

図 10 に今回適用したアルゴリズムを示す．これは 2 分探索を応用している．たとえば値の範囲を 0〜255 とする．最小値を `ll = 0`，最大値を `hh = 255` とし，その中央値である `(ll + 1 + hh)/2 = 128` の値のときにブレーキ曲線が許容されるかどうかを調べる．もし許容されるなら `ll = 128` として 128〜255 の間，許容されないなら逆に `hh + 1 = 128` として 0〜127 の間へと探索範囲を小さくし，さらに探索してゆく．最終的には `ll = hh` となる．`ll` が条件を満たし `hh + 1 = ll + 1` が条件を満たさないから `ll` が最大値となる．

ここで variant を `hh - ll` としておけば，`ll := sp` を実行すれば `ll` が大きくなり，`hh := sp` を実行すれば `hh` が小さくなるから，この値は減少する．実際にはビットイ

メージで上位ビットから値を確定させることになるから，ループの実行回数は値の範囲を 2 進表現したときのビット数となる．なお，最初の VAR の行は局所変数の定義を行っている．局所変数の型は最初の代入時に決定する．

このアルゴリズムで最大値が得られることの証明の概略を与える．ここで，図 10 の不変条件のうち ll が属し，hh + 1 が属さない集合 {sp1 | ..}を X とする．ll が X に含まれているから，max(X) <= ll．ここで ll が max(X) でないとすると，max(X) <= ll + 1 となる．ところが X の定義のうち，不等式の左辺が sp1 に関して増加関数となっているから，max(X) 以下の値はつねに X の要素である．また hh + 1 = ll + 1 であるから，hh + 1 は X の要素とならなければならないが，このことは hh + 1 の不変条件を満たさないから矛盾する．これは ll が max(X) でないとした仮定が誤っているためであり，よって ll = max(X) となる．この証明は実際に証明器で実行可能である．

ループの開始時に ll が X に含まれるのは，0 が X に含まれるからで，hh + 1 が X に含まれないのは，その前に hh := MaxSpeed という代入を行ったために hh + 1 の値が MaxSpeed + 1 となるからである．また，この例では抽象変数 distance_n が見られるが，このように while ループの INVARIANT 内では抽象変数が使用できる．実際にはこの値は ii <-- NormalDistance と書かれているところで，operation 呼び出しにより取得している．

また，非常ブレーキ速度 emergency_speed，常用ブレーキ速度 normal_speed，警報速度 warning_speed の間に

```
emergency_speed >= normal_speed
normal_speed >= warning_speed
```

が成り立つことが要件 4 で要求されていたが，この関係はそれぞれが max 関数で定義されれば証明できる．emergency_speed = max(A)，normal_speed = max(B)，warning_speed = max(C) の場合に，B の要素がつねに A に含まれ，C の要素がつねに B に含まれることを示せばよい．B の要素がつねに A に含まれれば max(B) が A に含まれるので，max(A) >= max(B) であることを示せる．

5.3 証明

CalBrakeSpeed についても証明責務をすべて証明することができた．速度から距離を与える関数の定義が残っているが，関数の定義を行えば max 関数と while ループによる 2 分探索を使うことにより計算が実現できる．このことは，関数の定義において停止までの走行距離が速度の 2 乗に比例する場合は，計算結果が 2 次関数の不等式を満たす値となり，平方根を含む値を近似的に求められることを示している．今回はあてはまらないが，異なる形式の関数の定義により，より多彩な数値計算ができることも示している．

ここで CalBrakeSpeed の詳細化は 3 段となったが，各詳細化ごとに生成された証明責務の数を**表 4** にまとめた．

表 4 第 2 段階における証明の数
Table 4 The number of proofs for the second model.

module	全体	自明	自動証明	手動証明
抽象機械	70	56	3	11
詳細化 1	103	96	6	1
詳細化 2	59	54	1	4
実装	346	109	141	96
計	578	315	151	112

最後の詳細化（実装）の証明責務の数が多く，対話的証明の数も多いことが分かる．証明責務の具体的な中身を調べると，while ループに関わるところで証明責務が多数発生していることが分かった．この要因を調べる．

INVARIANT で記述される ii : INT や sp : ll..hh という制約を，ループ実行ごとに確認する必要があることがその理由の 1 つである．ここでループの不変条件は 6 文ある．なおかつ，ループの中で条件分岐を使っており，その分岐後でそれぞれ確認を行うので単純に 1 つのループごとに $6 \times 2 = 12$ の証明責務が必要になる．

さらに証明責務が増える要素がある．たとえば sp := (ll + 1 + hh) / 2 という代入があるが，これに対して，sp : ll..hh であること，すなわち (ll + 1 + hh) / 2 >= ll かつ (ll + 1 + hh) / 2 <= hh を示す必要がある．この代入だけで，3 つの証明責務が生成される．このうちたとえば (ll + 1 + hh) / 2 >= ll を証明するには hh + 1 >= ll を示せば，ll + 1 + hh >= 2 * ll が示せるので，両辺を 2 で割ればよいが，この中間の式も自動では導出されないし，最後の 2 で割る指示も必要である．不等式を扱うことで自動化率が減る原因となっている．

つまり，while ループの使用とループ不変条件により証明責務の生成数が多くなることが分かった．

5.4 まとめ

与えられた速度から停止までの走行距離を与える関数を定義すれば，その他の余裕距離などを考慮したうえで許容される速度を算出できることを証明した．さらに max 関数と while ループを用いることにより浮動小数点計算で平方根を使用するような計算であっても，B メソッドに適用可能であることを示した．

一方で while ループを使って数値計算を行った場合，ループ不変条件を増やすことにより証明責務が増えることも分かった．2 分探索を行うことにより実用的な実装が可能であるが，一方で不変条件が増えるため，証明の手間が増える．証明を簡単にするために，while ループの制御を単純化し，線形探索を行えば，証明は簡単にはなるが，計算効率は落ちる．どちらを選択するかはユーザの判断になるが，探索により値を計算する実用システムであれば，2 分探索を選択せざるをえないと考えられる．

6. 補助関数の実装

6.1 抽象機械の記述

4, 5 章において distance_n という関数形の抽象変数と NormalDistance という operation が出てきた．これらについても詳細化により実装を与える必要がある．

4 章の BrakeSpeed では，パラメータを設定する operation と計算結果を得る operation を別としたが，実用的には関数として定義されていた方が使用しやすいため，こちらは関数形で定義した．抽象機械の記述は図 9 に示されている．!は全称限定子 $\forall$ を意味する．sp1 >= sp2 の場合に distance_n(sp1, gr) >= distance_n(sp2, gr) ということは速度が高いほど，停止距離が長くなることを示している．次の gr1 >= gr2 に続く部分は勾配が急であるほど（ブレーキが効かないので）停止距離が長くなることを示しており，ここで制約 6 が導入され，すべての制約が導入されることになる．そして，値を得る operation として，NormalDistance を定義しているが，この operation の実装が，これまでの制約を満たすことを要求される．

```
REFINEMENT
  Distance_1
REFINES
  Distance
CONSTANTS
  N_decl, E_decl
VARIABLES
  distance_n, distance_e
INVARIANT
  distance_n = SPEED * GRAD --> 0..MaxDist &
  !(sp, gr).(sp : SPEED & gr : GRAD =>
  distance_n(sp, gr) = sp * sp /
    (N_decl * 72 - gr * 72 * Gr1 / Gr2))
INITIALISATION
  distance_n(sp, gr) := %(sp, gr).
  (sp : SPEED & gr :GRAD | sp * sp /
    (N_decl * 72 - gr * 72 * Gr1 / Gr2))
OPERATIONS
  dn <-- NormalDistance(sp, gr) =
  PRE  sp : SPEED & gr : GRAD
  THEN
    dn := distance_n(sp, gr)
  END;
```

図 11 停止距離関数の詳細化
Fig. 11 Refinement of function that returns distance to stop.

6.2 詳細化における計算

次の詳細化においては，**図 11** のように記述した．初期化の箇所で用いている%はラムダ式の宣言であり，これにより distance_n を具体的な定義により初期化したことになる．もちろん，具体形を与えず，INVARIANT を満たすものとして定義することも可能であるが，最終的な実装までには，結局具体形を与える必要が出てくる．また N_decl, E_decl は常用ブレーキ，非常ブレーキの減速度である．

走行距離は初速や減速度に依存する．減速度は実際のブレーキのかかり具合いを随時判断するのではなく，性能に対して滑走などを考慮して余裕を持たせて指定する．実際の減速度は速度が高くなるにつれて小さくなるが，ここでは簡単のため一定値 α とする．初速が v の場合の走行距離は $v^2/2\alpha$ となるが，単位や有効数字を考慮する必要がある．include 先の BrakeSpeed では走行距離は m 単位，速度は 0.1 km/h 単位であり，ここでも同様とする．計算では時速（km/h）を秒速（m/s）に換算することになるが，1 km/h = 10/36 m/s より変換比 $x = 10/36$ を乗じるものとする．

減速度は通常数 km/h/s であり，0.1 km/h/s 単位で指定される．今回もこれを踏襲する．速度 v×0.1 km/h から β×0.1 km/h/s で減速した場合の走行距離は $(vx/10)^2/2(\beta x/10) = v^2x/(2 \cdot 10\beta) = v^2/72\beta$ で与えられる．定数部分の 72 が図 11 に現れている．減速度の有効数字が 2 桁であるため，結果の有効数字も 2 桁程度となるが，誤差については減速度設定の余裕でカバーする．

また，勾配は通常パーミル（‰, 1/1000）単位で与える．上り勾配では減速度が大きくなるが 0 とする．下り勾配では端数は余裕を見て上側に丸める．勾配を減速度に対応させるのにも変換が必要である．そのための換算比は，2 つの定数を用い，Gr1/Gr2 の形で与える．重力加速度が 9.807 m/s^2 であるから，これを km/h/s とする場合は $9.807 \sin\theta(\mathrm{m/s^2}) \approx 9.807\ \theta(\mathrm{m/s^2}) = 9.807 \times \theta/1000 \times 3600/1000(\mathrm{km/h/s}) = 35.31\ \theta/1000(\mathrm{km/h/s})$．これを 10 倍した値とするため Gr1 = 353, Gr2 = 1000 とした．ただし，車両の減速度から直接勾配の影響を引くと精度が落ちる．たとえば 5‰とした場合 5×353/1000=1715/1000 であるから 0.1 m/s^2 しか反映されない．そこで減速度に乗じる係数 72 を乗じてから 1000 で割ることとした．1‰に対して $353 \times 72/1000 = 25.416$ がかかる計算だから，0.1 km/h に対して 72 が対応する減速度よりも精度を持つことになる．

結果として以下の 2 つが制約として加わることになる．

制約 10 減速度は 0.1 km/h/s 単位で与える．

制約 11 勾配はパーミル単位とし，上り勾配は 0 とする．

なお，ここで使用したラムダ式は実装段階では使えない．しかし distance_n は抽象変数であり，別の形で実装されればよい．つまり，sp * sp / (N_decl * 72 - gr * 72 * Gr1 / Gr2) が dn の値として返されればよいのであり，distance_n そのものは実装される必要がない．そこで，実装は**図 12** のように記述した．

6.3 証明責務の数

このモジュールでの証明責務の数は**表 5** のとおりで，while ループを用いていないにもかかわらず実装段階での証明責務の数が多い．単線区間の閉そくの例では，場合分

```
dn <-- NormalDistance(sp, gr) =
VAR yy IN
  IF gr < 0 THEN yy := 0
  ELSE yy := gr * 72 * Gr1 / Gr2 END;
  IF yy < N_decl * 72 THEN
    dn := sp * sp / (N_decl * 72 - yy)
  ELSE
    dn := MaxSpeed * MaxSpeed
  END
END;
```

図 12 停止距離計算の実装

Fig. 12 Implementation of calculation for distance to stop.

表 5 停止距離補助関数に関する証明の数

Table 5 The number of proofs for calculation of distance to stop.

モジュール	全体	自明	自動証明	手動証明
抽象機械	35	34	1	0
詳細化 1	34	14	6	14
実装	176	41	99	36

けにより証明責務が大量に生成されていたが [2]，ここでは数値計算の部分で非常に多くの証明責務が生成されていた．これは数値計算の結果が処理系の整数（本論文では 32 bit）の範囲にあることをそのつど確認しているためである．たとえば計算結果 x が 32 bit 整数であるためには，x : INTEGER（無限の整数集合），x >= -2147483647，x <= 2147483647 の 3 つを確認する必要がある．さらに，計算途中に関してもそのつどこの 3 つの条件が満たされることを確認している．

証明責務が大量に発生する例として図 12 における

```
dn := sp * sp /(N_decl * 72 - yy)
```

という文を見てみる．この右辺の yy を

```
yy = gr * 72 * Gr1 / Gr2 = gr * 72 * 353/1000
```

で置き換えた

```
sp * sp /(N_decl * 72 - gr * 72 * 353/1000)
```

について以下のそれぞれで 3 つの条件の確認が必要である．

(1) `sp * sp/(N_decl * 72 - gr * 72 * 353/1000)`
(2) `sp * sp`
(3) `sp`
(4) `N_decl * 72 - gr * 72 * 353/1000`
(5) `N_decl * 72`
(6) `N_decl`
(7) `gr * 72 * 353/1000`
(8) `gr * 72 * 353`
(9) `gr`

さらに分母が 0 でない必要があるから not(N_decl * 72 - gr * 72 * 353/1000 = 0) を確認しており，この 1 行の計算だけで $3 \times 9 + 1 = 28$ もの証明責務が生成される．ただし値が整数値であることは，乗除算が INTEGER * INTEGER +-> INTEGER の型を持つ 2 項演算子（部分関数）であることから，証明は容易であり，実質的にはこれから 9 少ない 19 の証明が必要となる．その他についてはある程度対話的に証明する必要がある．a，b が整数定数のとき，a <= b が明示されないと x <= a から x <= b を自動では証明できない場合が多いためである．このような命題を処理する arithmetic prover はあるが，自動化の中に組み込まれていないため，対話的に実施する必要がある．

また，sp * sp を 32 bit 整数とするため，sp の値を 16 bit 整数とする必要がある（正確には $2^{15.5} = 32768\sqrt{2}$ までは許容される）．このように，計算の過程で制約に自乗を用いる場合などは整数の範囲がその分狭くなることに注意が必要である．このような制約は抽象機械の段階では課されないが，実装段階において課されるため，結局抽象段階にまでさかのぼって値の範囲を定義しなおす必要がある．つまり値の範囲は最初の段階での適切な設定が必要である．

ほかにも除算があると証明が難しくなる．たとえば上記 (7) の gr * 72 * 353/1000 が 32 bit 整数であることを示すには，gr * 72 * 353/1000 が gr * 72 * 353 より小さく，gr * 72 * 353 が 32 bit 整数であることを示せばよいが，これは自動では証明されない．精度を上げるために定数倍した値を扱う場合，除算を使用することで証明責務として多くの証明が必要となる場合があることが分かる．それを避けるためには，除算を用いない形に式を変形するのが 1 つの方法であるが，物理量との対応がとりにくく，式そのものが意図したものと合うかどうかの確認に難がある．また除算を排除するために両辺に除数を乗じた場合には，除算に関係ない部分にまで除数が乗じられるために値の範囲が制約を受ける可能性がある．

6.4 まとめ

補助関数についても実装段階までの仕様記述に対して証明によりその整合性を検証できた．これにより要件 1〜4，制約 1〜11 を満たすブレーキ曲線計算プログラムが生成できることが示される．しかし，証明責務を観察すると，数値計算を行う場合，その計算式によってはそれ自身にともなう証明責務が多く生成されることが分かった．これは計算の中間値について固定長の範囲の整数であることが要求されることに起因する．またそれに関連して中間値や計算結果が不等式を満たすことが要求されるが，不等号や乗除算に関わる証明責務については自動で証明することが難しく，これにより手間が増えることが分かった．

論理演算では主に条件分岐によって証明責務が増えてしまったが，数値計算であっても条件分岐などを用いれば，それによって証明責務は増えることから，証明責務を減らすためにはアルゴリズム上の工夫は必要である．

しかしながら数値計算の精度を求め，整数倍した値などを計算に使用する場合などには，その制約や実装段階に乗除算を多用することになり，それにともない証明責務が多

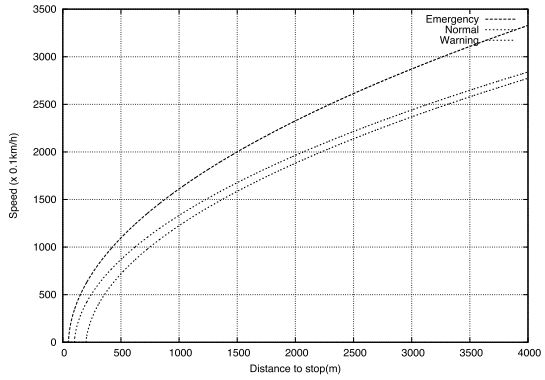

図 13 ブレーキ曲線の計算結果例
Fig. 13 Example of brake pattern calculation.

く生成されることになる．数値計算の精度と証明の手間の間にはトレードオフが発生する．

7. 計算例

補助関数の証明により，アルゴリズムの証明は終了した．後は計算が意図したものであるかを確認する．

減速度を常用 3.0 km/h/s，非常 4.0 km/h/s とし，余裕距離を警報 200 m，常用 100 m，非常 50 m，空走時間を非常 1 秒，常用 2 秒，警報 3 秒とした場合の出力例を**図 13** に示す．3 つのブレーキ曲線が関係を保たれていること，また放物線状に描けていることが分かる．

ここで図 13 は，B のモジュールを変換したソースコードに停止距離を 0 から 4000 まで変化させて，先の速度計算を行う operation を呼び出すコードを加えて，標準出力に出力させたものをグラフ化したものである．

パラメータの値は，まず制約を `Parameter.mch` といった抽象機械に記述し，実装値を `Parameter_i.imp` といった実装に記述することで証明が可能となる．車上装置において各種パラメータをメモリカードや ROM で設定することがよく行われるが，B ではファイルの入出力部分はサポートされていないため，値をメモリカードなどで設定する場合は，実装部分を B で記述せず，`Parameter.mch` に合わせたソースコードを記述することとなる．この場合はソースコードそのものは検証できないことに注意が必要である．

8. 数値計算の適用範囲

今回のモデル化を通じて，分数や平方根があっても B メソッドにおいて計算可能であることを示した．また 2 分探索のアルゴリズムが使用可能であることを示した．

同様にべき乗根についても証明の手間を惜しまなければ扱うことができる．ただし，べき乗根は引数が 32 bit の符号付き整数（31 bit の正整数）であるため，平方根の値は 15.5 bit となるし，3 乗根ならば 10.3 bit となる．一方，多項式も扱えるが，次数が高くなっても計算結果が 32 bit に収まる必要があり，定義域の方が制約を受ける．正負どちらでもとれるとすると，符号を別として 2 次式の 15.5 bit，3 次式の場合は 10.3 bit が適用範囲であり，いずれの場合も 3，4 乗根，3，4 次式程度が実用的な範囲といえる．

指数関数も有理数の範囲であれば理屈上は扱うことができる．B の言語仕様上はべき乗が定義されている．ここで $x = a^{b/c}$ を考えた場合，$x^c = a^b$ であるから，$x = \max(\{y|y \in \mathrm{NAT} \wedge y^c \leq a^b\})$ と定義すればよい．ただし，ここでも a^b を 32 bit 整数の範囲内にする必要があり，b の値を大きくすることはできない．

対数関数は言語仕様にはないが，B-book に $\log_m n = \min(\{x|x \in \mathrm{NAT} \wedge n \leq m^x\})$ という定義が示されている．あるいはこれまでの表現にならって $\log_m n = \max(\{x|x \in \mathrm{NAT} \wedge m^x \leq n\})$ とも書ける．これにより対数関数も扱える．この場合は m^x の値が 32 bit に収まるかが値の範囲の制約となるが，そもそも n の値が 32 bit であるから問題はない．$\log_m x/y$ については $\log_m x/y = \log_m x - \log_m y$ の定義で解決するし，m が分数の場合は底を変換すればよい．

こう考えると，B メソッドにおいても多項式，べき乗根，指数関数，対数関数の範囲であれば一応は扱うことができる．ただし，多項式や指数関数は実装上の制約から定義域の範囲が狭くなる．またべき乗根や対数関数などは実装で 2 分探索を行うと証明の手間がかかることから，これらの数学関数については，抽象機械の定義，その詳細化および実装をライブラリ化し，詳細化までの証明をしたうえで提供すれば，より利用しやすくなると考えられる．

9. 関連研究

B メソッドの適用例の代表的なものに文献 [5], [6] がある．これらは鉄道への適用であり，この中でも許容速度の計算という言及がなされているが，具体的な計算の中身には言及していない．それ以外には文献 [9] などが報告されているが，数値計算とは異なる適用範囲である．

最近は B メソッドよりも，システムのモデル化に重点を置いた Event-B に注目されており，ケーススタディが文献 [10] に紹介されているほか，鉄道に関連したものについても文献 [11], [12], [13] などが報告されている．

この中で文献 [11] では，ハイブリッドシステムへの適用ということで，鉄道やそれ以外の事例の検討を行い，その中で実数を意識した記述が行われているが，実際には Event-B では実数はサポートされていない．そこで，本来は実数のものを整数で扱っているが，三角関数を扱う航空機衝突回避システムの記述では，三角関数の性質のほか，実数では成り立ち，整数では成り立たない除算などの公理を追加している．これらは今回の論文のようにあくまでも整数の演算で行う方法とはやり方が異なる．また Event-B の場合コード生成は正式にはサポートされていない．

しかし Event-B では公理系を拡張するための Theory Plugin という仕組みがある．これを用いて実数に関する公

理系を構築することで，実数をサポートすることが提案されており [14]，実数理論の提供も始まっている．また Event-B のバックエンドに SMT（Satisfiability Modulo Theories）ソルバを使用する試みが行われている [15]．SMT ソルバでは限られた範囲ではあるが実数の理論を使用すれば，実数を取り扱うことが可能となる．このような形で Event-B における実数の使用が実用的になれば，B メソッドにフィードバックされ，使用しやすくなる可能性は十分にある．

なお，Event-B には本論文で使用したような while ループは提供されていない．同様なことは Event-B でも記述可能としており，文献 [10] においても while ループを Event-B で書いた場合の事例が紹介されているが，Event-B ではプログラム生成をターゲットとしておらず，今回の実装レベルに関してはその知見は必ずしも有効ではない．

10. まとめ

ATC システムのブレーキ曲線を計算するプログラムに B メソッドを適用し，B メソッドが整数演算しか使用できないという制限を持っているものの，実用的な精度を持ち，要求仕様を満たすことが計算機上で証明されたブレーキ曲線計算プログラムを生成できた．このことは数値計算を主とする場合であっても，B メソッドを用いて仕様を満たすことが保証されたプログラムを生成可能であり，プログラムの高信頼化を図ることができることを意味している．

また，計算の詳細を検討した結果，B メソッドを数値計算に適用する際の知見も得た．

- B メソッドでは小数を扱わないが，max 関数と while ループの使用により，有理数の平方根などの計算が使用可能である．ただし，計算では 2 分探索が実用的なアルゴリズムとなるが，ループ不変条件が多くなることから証明責務の生成数は大きくなる．
- B メソッドを用いて数値計算を記述する場合は，実装段階において，その値が中間値を含めて実装可能な範囲（現在は 32 bit 整数）であることを確認するために，大量の証明責務が発生しうる．特に乗除算や不等号が用いられると証明責務が自動で証明できないことが多く，計算精度を上げようとした場合に証明の手間が急激に増える傾向にある．
- 関数を定義する場合には抽象変数と値を得る operation をセットにするのが 1 つのテクニックである．

現在 B メソッドについてはその派生である Event-B がモデル化に使用され，プログラム生成のために B メソッドを使用する研究は活発ではないが，これにより数値計算に適用する事例が増えることを期待する．

謝辞 本論文をまとめるにあたり，貴重なご意見をいただきました九州大学の荒木啓二郎先生に，この場を借りて感謝申し上げます．

参考文献

[1] 寺田夏樹：鉄道信号システムへのフォーマルメソッド適用，鉄道総研報告，Vol.16, No.7, pp.15–20 (2002).
[2] 寺田夏樹：信号装置仕様の検証を通じた B メソッドにおける仕様記述法の検討，情報処理学会論文誌プログラミング，Vol.7, No.2, pp.20–35 (2014).
[3] Abrial, J.-R.: *The B-Book: Assigning programs to meanings*, Cambridge University Press (1998).
[4] 寺田夏樹：段階的詳細化による鉄道信号へのフォーマルメソッド適用法，鉄道総研報告，Vol.21, No.11, pp.41–46 (2007).
[5] Behm, P., Benoit, P., Faivre, A. and Meynadier, J.-M.: Météor: A Successful Application of B in a Large Project, *Proc. FM'99*, LNCS 1708, Vol.I, pp.369–387 (1999).
[6] Badeau, F. and Amelot, A.: Using B as a High Level Programming Language in an Industrial Project: Roissy VAL, *Proc. ZB2005*, LNCS 3455, pp.334–354 (2005).
[7] Potter, B., Sinclair, J. and Till, D.: *An Introduction to Formal Specification and Z*, Prentice Hall (1991).
[8] Atelier B, Clearsy (online), available from ⟨http://www.atelierb.eu⟩ (accessed 2014-5-28).
[9] Lecomte, T.: Safe and Reliable Metro Platform Screen Doors Control/Command Systems, *Proc. FM2008*, LNCS 5014, pp.430–434 (2008).
[10] Abrial, J.-R.: *Modeling in Event-B: System and Software Engineering*, Cambridge University Press (2010).
[11] Abrial, J.-R., Su, W. and Zhu, H.: Formalizing Hybrid Systems with Event-B, *Proc. ABZ2012*, LNCS 7316, pp.178–193 (2012).
[12] 佐藤直人，タイソンホアン，デビッドベイジン，來間啓伸：Event-B による列車監視システムのモニタリング要件の検証，情報処理学会論文誌，Vol.54, No.6, pp.1738–1750 (2013).
[13] Sabatier, D., Burdy, L., Requet, A. and Guéry, J.: Formal Proofs for the NYCT Line 7 (Flushing) Modernization Project, *Proc. ABZ2012*, LNCS 7316, pp.369–372 (2012).
[14] Butler, M. and Maamria, I.: Practival Theory Extension in Event-B, *Theories of Programming and Formal Methods*, LNCS 8051, pp.67–81 (2013).
[15] Déharbe, D., Fontaine, P., Guyot, Y. and Voisin, L.: SMT Solvers for Rodin, *Proc. ABZ2012*, LNCS 7316, pp.194–207 (2012).

寺田 夏樹 （正会員）

1971 年生．1994 年東京大学工学部計数工学科卒業．1996 年同大学大学院修士課程修了．同年財団法人（現在，公益財団法人）鉄道総合技術研究所入社．ATC，軌道回路等の鉄道信号システム，およびその形式手法適用に関する研究に従事．電気学会，計測自動制御学会各会員．